Курс лекцій

Бендюг В.І.,
Комариста Б.М.

[*Основи роботи з сучасними інтегрованими програмними комплексами*]

Основи створення прикладного програмного забезпечення .

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Хіміко-технологічний факультет

Основи роботи з сучасними інтегрованими програмними комплексами.

Основи створення прикладного програмного забезпечення

Курс лекцій

для студентів напряму підготовки
6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»

Затверджено методичною радою
хіміко-технологічного факультету НТУУ «КПІ»

Київ – 2014

Основи роботи з сучасними інтегрованими програмними комплексами. *Основи створення прикладного програмного забезпечення.* Курс лекцій *для студентів напряму підготовки 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»* [Електронний ресурс] / [уклад. Бендюг В.І., Комариста Б.М.]. – К: 2014. – 286 с. Систем. вимоги: MS Windows; Adobe Acrobat Reader — Назва з екрану.

Гриф надано методичною радою ХТФ НТУУ "КПІ",
протокол № 00 від 00.00.2014 р.

Електронне навчальне видання

Основи роботи з сучасними інтегрованими програмними комплексами.

Основи створення прикладного програмного забезпечення.

Курс лекцій

для студентів напряму підготовки
6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»

Укладачі: Бендюг Владислав Іванович, , канд. техн. наук, доцент
Комариста Богдана Миколаївна

Відповідальний редактор А. М. Шахновський, канд. техн. наук, доцент

Рецензент О. І. Букет, канд. техн. наук, доцент.

За редакцією укладачів

ЗМІСТ

ВСТУП

Дисципліна "*Основи роботи з сучасними інтегрованими програмними комплексами*" викладається згідно учбового плану бакалаврської підготовки студентів *напряму підготовки 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»*. Вона призначена навчити студентів:

- сучасних методів програмування, які використовуються для вирішення типових задач хімії і хімічної технології
- об'єктно-орієнтованому підходу до реалізації програм на персональному комп`ютері (ПК).

Ця дисципліна базується на початкових знаннях студентів про побудову та принципи роботи ПК (викладається в дисципліні *«Комп'ютерна техніка та організація обчислювальних робіт»*) та на їх початковому вмінні працювати на ПК в операційних системах Windows (в тому числі – з додатками Excel та Word). Після вивчення курсу студент повинен знати:

- стандарти сучасних об'єктно-орієнтованих мов програмування;
- інтерфейс середовища програмування Delphi;
- сучасні методи конструювання програм в тому числі засобами об'єктно-орієнтованої мови програмування Delphi;
- ефективні методи збереження і обробки інформації.

Крім того, студент повинен вміти:

- самостійно розробляти ефективні алгоритми і програми на сучасній алгоритмічній мові для вирішення поставленої задачі;
- налагоджувати програми на ПК до надійної працездатності в ОС Windows;
- виконувати налаштування середовища програмування для ефективної роботи;
- оформляти розроблені програми згідно вимогам ЄСПД.

Знання та навички, одержані в цій дисципліні, у подальшому використовуються в усіх дисциплінах, які потребують програмної реалізації розрахунків на комп'ютері, і в першу чергу – в курсах "Числові методи", "Математичне моделювання на ЕОМ", "Методи оптимізації хіміко-технологічних процесів", "Інформаційні системи та комплекси", в курсових і дипломних роботах і проектах.

ЧАСТИНА І

РОЗДІЛ 1 ВВЕДЕННЯ В DELPHI

Тема 1.1. Основи Delphi

1.1.1. Знайомство з Delphi

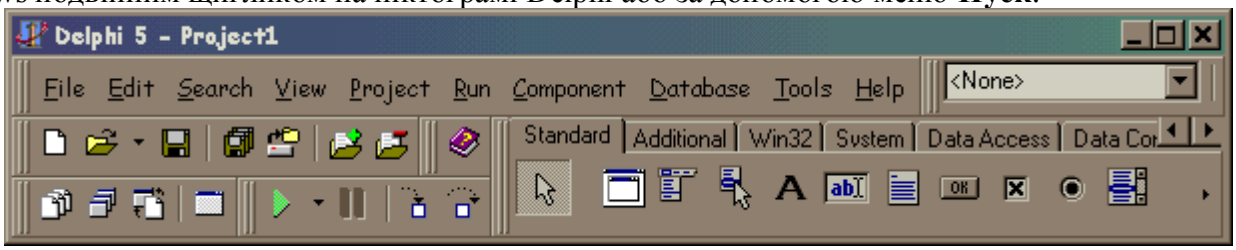
Delphi має графічний інтерфейс користувача, подібний використовуваному Microsoft у Visual Basic і C++. В даний час безліч фірм прийняли його як стандарт для розробки інтерфейсів власних додатків. Оскільки інтерфейс користувача створюється візуально, про Delphi говорять як про середовище, призначене для швидкого створення додатків. Основною складовою середовища швидкого створення додатків є технологія, що одержала назву *Two Ways Tools*. Це значить, що при розміщенні або зміні компонента в якій-небудь формі відповідний програмний код буде автоматично доповнений або модифікований. І навпаки, усі зміни, що вносяться в програмний код при розробці додатка, автоматично відбиваються на функціональних властивостях форми. Для створення інтерфейсу користувача Delphi пропонує широкий набір допоміжних засобів.

Інший аспект графічного середовища розробки – це довідкова система Microsoft. Delphi надає у ваше розпорядження засоби, що дозволяють постачати додаток контекстно-залежною довідковою інформацією (Help).

Delphi, як і всяке сучасне середовище розробки додатків, засноване на Об'єктно-орієнтованому програмуванні (ООП). Ця технологія програмування є тією основою, що і дозволяє реалізувати усі функціональні можливості Delphi. При створенні додатків на основі готових компонентів з використанням властивостей, методів і заздалегідь визначених оброблювачів подій можна обходитися програмним кодом невеликого розміру. Для розроблювача це означає, що при розробці інтерфейсу користувача своїх додатків він може одержати значну економію часу.

Мовою програмування Delphi є Object Pascal. Ця мова по програмних конструкціях нагадує Pascal сьомої версії середовища Borland Pascal. Під програмними конструкціями варто розуміти структури, що визначають, у якій послідовності виконуються інструкції в програмі. Прикладами програмних конструкцій можуть служити умовні оператори *If-Then-Else*, оператори циклу *Repeat-Until*, а також механізми виклику методів.

Основою Delphi є графічне середовище розробки додатків, назване *інтегрованим середовищем розробки* (Integrated Development Environment, IDE). Воно може бути запущене в Windows подвійним щигликом на піктограмі Delphi або за допомогою меню **Пуск**.



Малюнок 1.1.1 – Головне вікно Delphi

Основою створюваного в Delphi додатка завжди є *форма* (Form). Розроблювач додатка може розміщати у формі різні компоненти (поля, кнопки, системи меню і т.д.). Програмний код цих компонентів автоматично генерується Delphi. Для створення багатьох додатків буває досить розмістити у формі стандартні компоненти, оскільки їхнє число в Delphi дуже велике.

Інтегроване середовище розробки Delphi (IDE) включає кілька основних елементів.

- ***Піктограми (Speed buttons)***. Це піктографічні кнопки, що дублюють деякі команди меню. Найменування кожної піктограми можна одержати прямо на екрані в маленькому вікні контекстної підказки, що з'являється, якщо покажчик миші затримати на якийсь час на зображенні відповідної піктограми.
- ***Головне меню (Menu bar).*** Це стандартне меню в стилі Windows. Біля більшості команд меню в Delphi зображені піктограми. Це ті ж самі піктограми. Що і на панелі інструментів.
- ***Палітра компонентів (Component palette).*** Тут представлені піктограми компонентів, що включені в *бібліотеку візуальних компонентів (VCL).* Клацнувши мишею, можна вибрати кожен з компонентів, а клацнувши ще раз на поле проектованої екранної форми, можна вказати Delphi, куди варто вставити об'єкт обраного компонента.
- ***Категорії палітри.*** У кожен момент часу в полі палітри відкрита тільки одна сторінка (категорія) компонентів. Для того щоб відкрити іншу категорію, клацніть на корінці відповідної вкладки палітри.
- ***Інспектор об'єктів (Object Inspector).*** Це діалогове вікно відображає списки усіх властивостей і подій одного або більш об'єктів компонентів, обраних у проектованій екранній формі. Жодний інший засіб Delphi не використовується так часто в процесі розробки програми, як вікно Object Inspector.
- ***Вкладки властивостей і подій.*** Вікно інспектора об'єктів Object Inspector містить дві вкладки. Вкладку *властивостей (Properties)* об'єктів компонентів, *включених* в екранну форму, і вкладку *подій (Events)* об'єктів компонентів. Властивості описують атрибути об'єкта – розмір кнопки або шрифт текстової етикетки. Події ж представляють деякі дії, наприклад щиглик на кнопці або натискання клавіші.
- ***Вікно проектування екранної форми (Form window).*** При розробці більшості програм екранна форма є візуальним представленням головного вікна додатка. Але вона може відноситься і до інших вікон, наприклад до діалогового вікна додатка або дочірнього вікна MDI-додатка (MDI – *Multiple Document Interface, многодокументный інтерфейс*). Якщо в простих додатках, як правило, використовується тільки одна екранна форма (SDI – *Single Document Interface, однодокументний інтерфейс*), то в складних їхня кількість може досягти десятка. Крапкова сітка на полі проектування форми допомагає вибрати точне місце установки об'єктів і вирівняти їх. У працюючому додатку ця сітка відсутня.
- ***Редактор коду програми (Code editor window).*** У цьому вікні можна переглядати і редагувати код (текст) програми мовою Object Pascal, зв'язаний з будь-якою екранною формою розроблювального додатка. Більшість рутинних фрагментів коду – оголошення і заготівки процедур обробки подій – Delphi вставляє в програму автоматично. Після цього від розроблювача потрібно наповнити заготівки коду змістом – уключити необхідні оператори і вираження Object Pascal, що, власне, і визначають поводження програми в тій або іншій ситуації, наприклад реакцію на щиглик мишею або вибір команди меню. У цьому ж вікні можна переглядати і редагувати інші модулі Pascal-програм.
- ***Діалогове вікно Module Explorer.*** Цей засіб включений у Delphi починаючи з четвертої версії. Вікно Module Explorer виводить інформацію про поточний програмний модуль – перелік класів, список інших модулів, що він використовує, змінних, об'єктів, методів і т.д. Для переходу у вікні коду програми до оголошення того або іншого програмного об'єкту або до коду реалізації методу або функції потрібно клацнути на відповідному елементі у вікні Module Explorer.

Крім описаних, у Delphi є безліч інших діалогових вікон – *менеджер проекту Project Manager, броузер об'єктів Object Browser,* убудований налагоджувач, “інтелектуальне”

доповнення до редактора коду *Code Insight, редактор зображень Image Editor, редактор пакетів Package Editor, проектувальник меню Menu Desiner* і т.д.

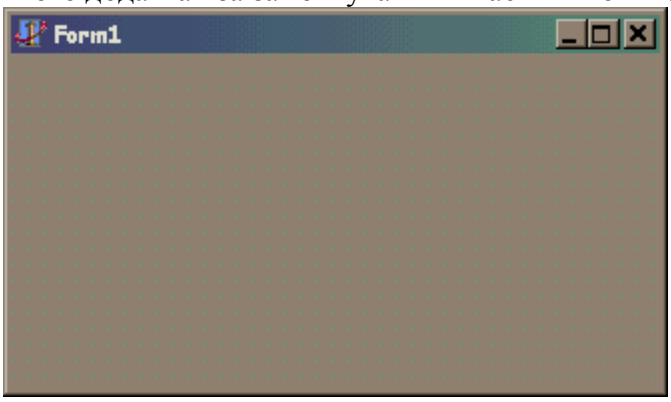
1.1.1.1. Палітра компонентів Component palette

У палітрі компонентів (Palette) відображаються компоненти, за допомогою яких користувач створює свої додатки. Компоненти є основними елементами кожного Delphi – додатка і, одночасно, основою бібліотеки візуальних компонентів Delphi – Visual Component Library (VCL). Вони дозволяють створювати інтерфейс користувача ваших прикладних програм.

Піктограми стандартних компонентів Delphi, відповідно до виконуваних ними функцій, розділені на групи; кожна з цих груп піктограм розташована на окремій сторінці палітри компонентів. Після запуску Delphi активної є сторінка Standard палітри компонентів. Вибрати іншу сторінку можна щигликом на вкладці з відповідною назвою. Наприклад, при створенні додатків для роботи з базами даних, як правило, бувають необхідні компоненти сторінок Data Access і Midas. Для відкриття потрібної сторінки варто виконати щиглик на вкладці з цією назвою.

1.1.1.2. Проектувальник форм Form Designer

Кожен Windows-додаток виконується у власному вікні – головному вікні відповідного додатка. Delphi призначає головне вікно для кожного додатка автоматично. Розробка додатка завжди починається зі створення нового проекту. Для кожного нового проекту в IDE автоматично відображається вікно проектувальника форми, що являє собою головне вікно вашого майбутнього додатка і за замовчуванням має ім'я Form1.



Малюнок 1.1.2 – Проектувальник форм Form Designer

Головне вікно – це перше, що бачить користувач після запуску додатка. Якщо користувач закриває це вікно, тим самим він закриває додаток.

Для вас, як розроблювача, головне вікно – це форма, відображувана при розробці додатка в проектувальнику форм (Form Designer), у якому ви створюєте графічний інтерфейс користувача вашого додатка.

Для того щоб розмістити компоненти у формі, необхідно спочатку виконати щиглик на піктограмі потрібного компонента, а потім у тім місці проектувальника форми, де повинний бути розташований компонент. Знайти зазначені компоненти в палітрі компонентів допоможуть спливаючі підказки (hints).

При виконанні додатка форма буде за замовчуванням відображатися в тім же виді, як і при проектуванні, тобто буде мати ті ж розміри і містити ті ж компоненти.

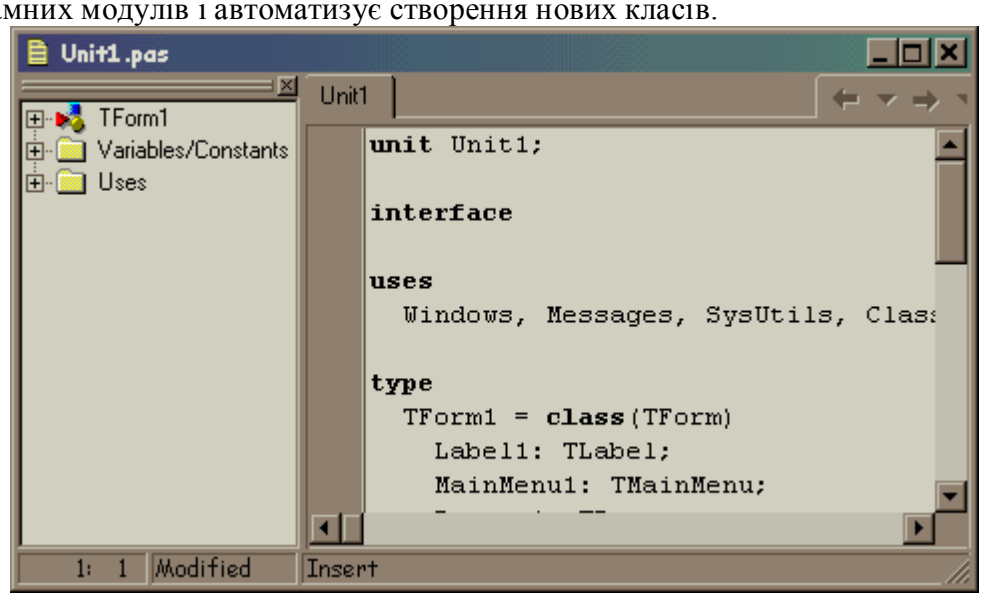
Проектувальник форм містить контекстне меню, за допомогою якого можна виконувати визначені команди редагування компонентів. Контекстне меню можна викликати, виконавши в будь-якому місці проектувальника форми щиглик правою кнопкою миші.

1.1.1.3. Вікно редактора коду Code Editor

Вікно редактора коду (Code Editor) має за замовчуванням заголовок Unit1.pas, воно знаходиться за вікном проектувальника форм. Редактор коду і проектувальник форм тісно зв'язані між собою.

У редакторі коду можуть бути відкриті кілька файлів. Кожен відкритий файл розміщається на окремій сторінці, а його назва відображається у верхній частині вікна на окремій вкладці. Назва файлу, відкритого першим, - крайнє ліворуч. Виконавши щиглик на імені потрібного файлу, можна активізувати сторінку з вихідним кодом модуля, який потрібно відредагувати або переглянути. Ім'я файлу, сторінка якого активізована, відображається в рядку заголовка редактори коду.

У лівій частині вікна редактора коду розташовується вікно *провідника коду* (Code Explorer), що є нововведенням Delphi 4. Провідник коду спрощує пошук інформації в коді програмних модулів і автоматизує створення нових класів.



Малюнок 1.1.3 – Вікно редактора коду Code Editor

У нижній частині вікна редактора коду розташований рядок стану, у якому міститься наступна інформація:

Таблиця 1.1.1 - Інформація в рядку стану вікна редактори коду

Інформація в рядку стану	*Значення*
7:1	Позиція курсору в тексті (рядок і стовпчик)
Modified	Указує на те, що після останнього збереження в тексті були зроблені зміни
Insert	Указує на те, що редактор знаходиться в режимі вставки

Вікно редактора коду ніколи не буває порожнім. Якщо відкритих файлів немає, вікно закрите.

1.1.1.4. Інтуїтивний помічник написання коду Code Insight

Редактор коду Delphi оснащений набором засобів, що забезпечують виконання цілого ряду допоміжних функцій. Ці засоби мають загальну назву Code Insight – *інтуїтивний помічник написання коду* (Insight - інтуїція). Code Insight має п'ять складових.

- ***Доповнення коду (Code Completion).*** За допомогою функції доповнення коду в програму можуть бути легко введені імена властивостей, методів і оброблювачів подій об'єктів. У Delphi існує дві можливості використання цієї функції. Спочатку необхідно ввести ім'я об'єкта, поставивши наприкінці крапку, наприклад: Edit1. після деякої паузи Delphi відобразить на екрані список усіх властивостей і методів даного об'єкта. Можна також увести перший символ імені події або методу, а потім натиснути клавіші <Ctrl+Space>. Після цього на екрані відобразиться список властивостей і методів, імена яких починаються з цієї букви. Такий спосіб активізації функції доповнення коду зручно використовувати в операторах присвоювання. Наприклад, якщо у вихідний код уведені символи x:= те після натискання <Ctrl+Space> на екрані відобразиться список всіх об'єктів, змінних і констант, використовуваних у даному додатку.

- ***Шаблони коду (Code Templates).*** Шаблони коду містять у собі часто використовувані програмні конструкції, які можна легко включити в програмний код. Шаблони коду активізуються натисканням клавіш <Ctrl+J>. Після цього на екрані з'явиться вікно зі списком наявних шаблонів. Для цілеспрямованого пошуку потрібного шаблона варто спочатку ввести ключове слово (наприклад, if або array), потім скористатися сполученням клавіш <Ctrl+J>. Після цього на екрані відобразиться список шуканих шаблонів коду.
- ***Контекстний список параметрів (Code Parameters).*** Після введення імені процедури і відкриваючої дужки функція контекстного списку параметрів виводить на екран довідкове вікно (Hint) зі списком параметрів функції або методу. У списку параметрів напівжирним шрифтом виділяється поточний аргумент функції.
- ***Швидка оцінка значення (Tooltip Expression Evaluation).*** Дана функція дозволяє легко перевірити значення змінних і властивостей. Якщо в режимі налагодження призупинити виконання додатка й установити курсор на імені змінної або властивості у вікні редактора коду, через якийсь час на екрані з'явиться вікно з поточним значенням цієї змінної.
- ***Спливаючі підказки про оголошення ідентифікаторів (Tooltip Symbol Insight).*** Ця функція відображає на екрані спливаючу підказку про тип і місце оголошення ідентифікатора при установці курсору на його імені у вікні редактора коду.

1.1.1.5. Інспектор об’єктів Object Inspector

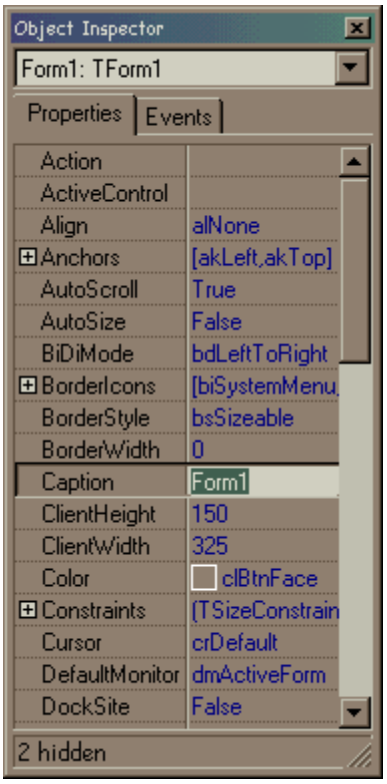
При розробці додатків часто приходиться використовувати інспектор об'єктів (Object Inspector). Якщо вікна Object Inspector немає на екрані, його можна відкрити за допомогою команди View⇨Object Inspector. Оскільки за допомогою інспектора об'єктів задаються і редагуються властивості й оброблювачі подій компонентів, його зручно постійно “мати під рукою”. Інспектор об'єктів можна активізувати за допомогою натискання клавіші <F11>. Крім того, за допомогою клавіші <F11> можна переключатися між вікнами проектувальника форм, редактори коду й інспектори об'єктів.

Вікно Object Inspector містить дві сторінки, кожну з яких можна активізувати, виконавши щиглик на вкладці з відповідною назвою. Перша сторінка має назву Properties. Лівий стовпчик цієї сторінки містить список усіх властивостей компонента, що редагується, доступних під час проектування. Друга сторінка Events. У її лівому стовпчику перераховані всі наявні оброблювачі подій компонента. У правих колонках обох сторінок можуть установлюватися значення відповідних властивостей або оброблювачів подій.

Функціональні можливості компонента, використовуваного в проектованому додатку, визначаються шляхом присвоєння властивостям компонента визначених значень і зв'язування з оброблювачами подій визначених процедур. Процедури обробки подій являють собою published-методи компонента, що виконуються при виникненні визначеної події.

Таким чином, інспектор об'єктів є інструментом, що використовується для формування зовнішнього вигляду і функціональних можливостей форми і компонентів у процесі розробки додатка.

Деякі властивості, відображені на сторінці Properties, мають початкові значення. Це стандартні значення даних властивостей. Значення в правому стовпчику можуть бути змінені користувачем під час проектування додатків. Крім того, значення властивостей компонентів можуть бути змінені або задані й у процесі виконання додатка, але для цього необхідно написати відповідний програмний код.



Малюнок 1.1.3 – Інспектор об'єктів Object Inspector

На сторінці Events у лівому стовпчику приведений список всіх оброблювачів подій компонента. Правий стовпчик при цьому порожній. Цей стовпчик призначений для процедур обробки подій. У ньому користувач встановлює імена процедур обробки подій, що повинні бути виконані при виникненні визначених подій. Кожен компонент має стандартний оброблювач події.

1.1.2. Файли і розширення їх імен

Типовий додаток у Delphi складається з файлів самих різних типів, кожний з яких має своє розширення імені.

- .~*. Це файли-копії модифікованих і збережених файлів (наприклад, Main.~dp). Ці файли можна час від часу видаляти. Вони можуть знадобитися для відновлення додатка в тім виді в якому він був до останнього збереження.
- .bpg. Файли категорії *Borland Project Group*, у яких зберігаються групові проекти. Ці текстові файли містять список залежностей між окремими модулями, що необхідні для компілятора і компоновщика при створенні багатомодульних додатків.
- .dcr. Файли типу *Delphi Component Resource* (ресурси компонентів Delphi). Ці файли містять піктограми, що ви можете побачити на палітрі компонентів з бібліотеки VCL. Файли цього типу видаляти не можна.
- .dcu. Файли цього типу містять компільований код і дані програмного модуля. Оскільки Delphi заново формує файл типу .dcu при кожнім сеансі компіляції додатка, їх можна зовсім безболісно видаляти.
- .dfm. Файли цього типу містять двоїнні дані, що стосуються властивостей екранної форми і включених у неї компонентів. Тут же утримується інформація про події і процедури їхньої обробки. Delphi копіює цю інформацію в остаточний виконуваний файл .exe. Файл .dfm видаляти не можна.
- .dll. Файли цього типу містять код бібліотеки динамічного компонування. Видаляти такі файли не можна.

- .dof. Ці файли зберігають опції проекту (*Delphi Options File*). У файлах даного типу Delphi зберігає настройки проекту, модифіковані в діалоговому вікні, що відкривається по команді Project⇨Options. Видаляти його можна тільки у випадку, якщо ви вирішили повернутися до настройок, заданих за замовчуванням.
- .dpk. У файлах цього типу зберігаються пакети (форма об'єднання декількох модулів). Видаляти ці файли не можна.
- .dpr. Файли проекту Delphi. Це, по суті, звичайні файли вихідного коду мовою Pascal. Delphi створює файл .dpr при першому збереженні нового додатка і видаляти його не можна.
- .dsk. Файли цього типу зберігають інформацію про конфігурації робочого столу додатка а також повний шлях до файлів проекту. Тому, якщо ви перенесли дані файли в інший каталог або "розкидали" файли модулів по різних каталогах, файл .dsk прийдеться видалити.
- .exe. Виконувані файли. У процесі роботи над проектом можна видаляти виконувані файли додатків, оскільки Delphi створює їх при кожній повторній компіляції.
- .pas. Файли цього типу містять вихідний код програми мовою Pascal. Типовий додаток, розроблений у середовищі Delphi, має по одному файлі .pas для модуля кожної екранної форми. Видаляти ці файли не можна.
- .res. Файли цього типу містять двоїнні ресурси (піктограми та інші растрові зображення).

1.1.3. Розробка додатків (програмних комплексів) у середовищі Delphi

1.1.3.1. Підготовка середовища до створення нового додатка та встановлення заголовка вікна

Для того щоб розробити інтерфейс користувача додатка, необхідно виконати наступні дії:

- вибрати необхідні компоненти;
- розташувати компоненти у формі;
- визначити зовнішній вигляд і функціональні можливості компонентів;
- задати в інспекторі об'єктів значення властивостей і процедур обробки подій;
- написати програмний код для заданих процедур обробки подій.

Перше, що потрібно зробити при створенні нового додатка, - дати йому ім'я. Найкраще привласнювати ім'я відразу ж після виклику команди File⇨New Application. Delphi використовує ім'я додатка (проекту) у самих різних цілях. Наприклад, це ім'я використовується при автоматичній генерації деяких оголошень і виражень. Якщо ж проект вами не іменований, Delphi використовує ім'я, задане за замовчуванням, що створює певні незручності.

Типовий проект у Delphi складається з файлів декількох типів. Одні з них містять текст програми, інші – двоїнні коди самого різного призначення: машинні інструкції, растрові зображення, дані і т.п. Тому доцільно зберігати усі файли одного проекту в окремому каталозі. Це полегшить роботу з проектом як з окремою структурною одиницею у файловій системі комп'ютера – копіювання, переміщення, видалення і т.п.

Для створення в Delphi додатка необхідно виконати наступні дії.

1) Виберіть у меню File⇨New Application. Інший варіант – виберіть File⇨New, а потім клацніть на вкладці Page і виберіть піктограму Application. Якщо до цього в Delphi був відкритий інший проект, вам буде надана можливість зберегти внесені в нього зміни або відмовитися від них.
2) Перш ніж вносити що-небудь у тільки що створений проект, виберіть у меню File⇨Save All або клацніть на піктограмі Save All. Після цього на екрані з'являться одне

за іншим два діалогових вікна. Перше – Save Unit1 As – вимагає задати ім'я для файлу коду головного програмного модуля. Можна привласнити файлу ім'я Main, це відразу вказує, що даний файл містить текст програми, зв'язаної з головним вікном додатка. До введеного імені файлу Delphi автоматично додасть розширення .pas (скорочення від *Pascal*).

3) Після цього з'явиться друге діалогове вікно Save Project1 As. Бажано називати проект по імені каталогу, у якому він зберігається. Після введення імені проекту Delphi додасть до нього розширення .dpr (скорочення від *Delphi Project*).

За замовчуванням у рядку заголовку вікна проектування екранної форми Delphi установлює напис Form1. Для зміни заголовка вікна необхідно клацнути мишею у вікні проектування екранної форми (його легко впізнати по крапковій сітці) для активації характеристик екранної форми у вікні Object Inspector. У верхнім полі Object Inspector повинне відобразитися TForm1. Клацніть на вкладці Properties вікна Object Inspector і з'явиться список властивостей об'єкта екранної форми додатка. Для зміни заголовка вікна виділіть властивість Caption і введіть нове значення в правому стовпчику того ж рядка замість напису Form1.

1.1.3.2. Запуск додатка, компіляція та компонування програми

Для запуску додатка необхідно натиснути <F9> або вибрати команду Run⇨Run з меню. Можна також клацнути на піктограмі Run (зелений трикутник). Зовсім не обов'язково цілком завершувати роботу над додатком, щоб запустити його на виконання. Після запуску додатка його необхідно закрити для можливості подальшого редагування додатка.

У рядку заголовка головного вікна Delphi відображається ім'я відкритого в даний момент проекту: Project1. Проект може бути відкритий у IDE як у режимі проектування, так і в режимі виконання. По заголовку головного вікна можна визначити в якому режимі відкритий проект. У лівій частині заголовка головного вікна, поруч із символом Delphi, знаходиться напис Delphi № – <Ім'я проекту>. Під час виконання проекту після цього напису випливає позначка [Running].

Коли ви натискаєте <F9> Delphi спочатку формує виконуваний файл. Цей процес поділяється на дві стадії. По-перше, компілятор Delphi транслює вихідний текст програми і формує двоїстий об'єктний код. По-друге, компонувальник Delphi зв'язує отриманий об'єктний код модуля з об'єктними кодами інших модулів, зокрема модулів бібліотек. У результаті утвориться виконуваний файл програми, що має те ж ім'я, що і проект, але з розширенням .exe.

Якщо вам потрібно тільки скомпілювати і скомпонувати проект, але не запускати його на виконання, натисніть <Ctrl+F9> або виберіть у меню команду Project⇨Syntax Check. Цей метод можна використовувати, щоб переконається в тім, що в програмі немає синтаксичних помилок.

Для відкриття проекту в середовищі Delphi виберіть команду File⇨Open або клацніть на піктограмі Open project. Коли з'явиться діалогове вікно Open Project, перейдіть у потрібний каталог і виберіть файл <Ім'я проекту>.dpr.

Література [1-3].

Тема 1.2. Об’єктно-орієнтоване програмування

Використання об’єктно-орієнтованого програмування є гарним рішенням при розробці великих програмних проектів. Чим проект об'ємніше і складніше, тим більше вигоди ви одержите при використанні об’єктно-орієнтованоої технології програмування. Однією з найбільших переваг об’єктно-орієнтованого програмування (ООП) є можливість багаторазового використання програмного коду. Якщо ви, приміром, створили клас, то можете породжувати від нього нові класи і змінювати їхні властивості і функціональне призначення.

Клас-предок при цьому залишається без змін, а відповідний вихідний код змінювати не прийдеться. Більш того, властивості і методи об'єкта інкапсульовані в ньому. Це значить, що ніхто і ніщо ззовні не може нічого змінити в об'єкті, якщо така зміна є неприпустимою. У результаті розробка

додатка полегшується, а програмісти можуть використовувати результати роботи колег, не вникаючи в подробиці.

Друга перевага полягає в тому, що додаток побудований з об'єктів, що є відображеннями реально існуючих предметів або процесів. Розглянемо компоненти Delphi ще раз під цим кутом зору. Якщо концепція об’єктно-орієнтованого програмування нова для вас, ви зштовхнетеся з деякими незнайомими поняттями. Нехай це вас не лякає. Щоб вивчити технологію ООП, треба насамперед зрозуміти її основні принципи.

У даній главі розглянуті основні поняття об’єктно-орієнтованого програмування. Розібравшись в них, ви зможете скласти собі ясне представлення про дану технологію.

На початку глави розглядається операційна система Windows як середовище розробки і виконання програм, а також вплив, що вона робить на розробку додатків. Наприкінці глави розглядаються деякі особливості реалізації ООП у середовищі Delphi.

1.2.1. Windows як середовище розробки та виконання програм

Програміст, що звик працювати в MS-DOS, повинний перестроїтися, щоб почати створювати програми для Windows.

У середовищі MS-DOS ви, напевно, звикли до наступної ситуації.

- Програмний код вашого додатка складається з операторів, що виконуються один за іншим.
- Одночасно може виконуватися тільки одна програма.
- У працюючій програмі щось постійно виконується.
- Існує прямий зв'язок між вашою програмою і комп'ютером. Це значить, що керування клавіатурою, екраном і т.д. здійснюється безпосередньо вашою програмою.

У середовищі Windows ви повинні настроїтися на наступну ситуацію.

- Код, що ви пишете, складається з процедур обробки повідомлень, що Windows посилає вашому додатку.
- Windows фіксує виникаючі в програмах і апаратурі події і потім посилає відповідні повідомлення у вашу програму.
- Одночасно можуть виконуватися кілька програм. Ці програми поділяють між собою робочу область пам'яті, процесор і інші ресурси системи.
- Працююча програма знаходиться в робочій області пам'яті й очікує повідомлень від Windows, на які вона повинна реагувати.
- Взаємодія з апаратним забезпеченням відбувається через графічний інтерфейс пристроїв (так називаний GDI). Програміста не повинні турбувати особливості пристроїв комп'ютера, цим займається Windows.

1.2.2. Керування програмою на основі повідомлень про події

Розробка додатків для середовища Windows здійснюється інакше, чим створення програм для MS-DOS. Windows пропонує вам так називану *кероване подіями середовище*, коли код програми виконується як реакція на події. Керування програмою на основі повідомлень про події не є новим, але не кожен програміст знайомий з цією концепцією. Вона є невід'ємною складовою частиною програмування для Windows. В основному поняття програмування за повідомленнями використовується для позначення процесу взаємодії між різними додатками. Це означає, що Windows у змозі керувати багатозадачною системою таким чином, щоб кілька програм могли виконуватися одночасно. Інакше кажучи, вони можуть спільно використовувати пам'ять, процесор і інші ресурси комп'ютера.

Як приклад подібного процесу представте, що створена вами програма для роботи з базою даних працює одночасно з текстовим редактором. Кожен з цих додатків виконується у своєму власному вікні. Представте також, що користувач працює з документом у текстовому редакторі

і використовує дані, які він хоче імпортувати з бази даних. Для цього йому необхідно в системі керування базами даних відкрити файл, що зберігає інформацію, наприклад, у dBASE-форматі, відкіля можна імпортувати дані.

Користувач робить щиглик мишею на значку додатка бази даних і цим самим породжує подію. Windows розпізнає цю подію і запускає додаток, що представлено даним значком.

Додаток реагує на це повідомлення і відкриває файл dBASE, звідки користувач може імпортувати дані.

Після того як повідомлення буде оброблено додатком, Windows знову одержить можливість керувати процесором і інформацію про те, що повідомлення оброблене — або про те, що Windows повинна зробити ще щось.

Якщо користувач, наприклад, уведе дані в додаток бази даних, вони повинні відобразитися на екрані. Додаток передасть необхідні повідомлення й інструкції Windows, а Windows відобразить дані. Таким чином, програма працює незалежно від апаратного забезпечення.

Це — головний принцип, на якому заснована система повідомлень. Як бачите, це система зі зворотним зв'язком. Додаток Windows повинний могти обробляти повідомлення від Windows, а також генерувати і посилати свої повідомлення.

При цьому Windows є як би посередником між користувачем, програмою і пристроями системи (Hardware). Усе це приводить до того, що програмування в Windows полягає головним чином в організації обробки повідомлень.

Delphi — це середовище розробки, що до певного ступеня звільняє програміста від нудної, рутинної роботи з керування обробкою повідомлень і залишає йому більше можливостей для творчості.

1.2.2.1. Створення та обробка повідомлень

Windows створює вхідне повідомлення для кожної вхідної події, що генерується користувачем за допомогою миші чи клавіатури. Windows зберігає вхідні дані в *черзі системних повідомлень*. Потім ці повідомлення посилаються в *чергу повідомлень додатка*.

Повідомлення додаткові від Windows формується шляхом створення *запису повідомлення* (Message Record) у черзі повідомлень. При цьому дотримується принцип FIFO — First In, First Out (перший зайшов — перший вийшов). Деякі повідомлення від Windows посилаються безпосередньо у вікно додатка і не ставляться в чергу.

Це так звані *позачергові повідомлення* (UM, Unqueued Messages). Типове UM — це повідомлення, що стосується тільки вікна додатка. Хоча більшість повідомлень породжується Windows, додаток також може створювати власні повідомлення, поміщати них у свою чергу повідомлень і посилати іншим додаткам.

Для обробки повідомлень головна програма додатка використовує безперервний цикл, так називаний *головний цикл повідомлень*.

Цей цикл містить деяку кількість функцій, призначених для обробки повідомлень. Як правило, вихід з цього циклу відбувається лише тоді, коли надходить повідомлення, що повинне завершити програму.

Головний цикл повідомлень починається з виклику функції, що переглядає чергу повідомлень (GetMessage()).

У випадку, коли користувач натискає клавішу, необхідно послати ще одне повідомлення. Це повідомлення містить *віртуальний код клавіші* (Virtual Key Code), що, хоча і констатує натискання клавіші, але не повідомляє безпосереднє значення символу клавіші. Тому для визначення символу спочатку викликається функція (TranslateMessage()). Потім викликається функція, що класифікує прийняте повідомлення (DispatchMessage()). Кожен об'єкт має свою процедуру, у якій визначені дії для кожного з можливих повідомлень. Коли ця процедура виконана, керування передається в початок циклу. Таким чином, повідомлення в цьому циклі вибираються й обробляються послідовно.

Якщо черга порожня, очікується нове повідомлення. У цьому випадку говорять, що додаток знаходиться в режимі чекання. По-англійському цей стан називається Idle. Під час чекання контроль над системою передається в Windows, завдяки чому інші додатки одержують можливість обробляти повідомлення.

1.2.2.2. Зв'язок між подіями та додатками

Яким образом Windows довідається, для якого додатка призначене дане повідомлення про подію? Кожен додаток виконується у своєму власному вікні, що має унікальний дескриптор — Handle, а тому що в записі повідомлення утримується інформація про дескриптор вікна, якому призначене дане повідомлення, то Windows "знає", за якою "адресою" його варто відіслати. Тому в Delphi не треба піклуватися про це; Windows самостійно здійснює керування формами і компонентами.

1.2.2.3. Обробка стандартних повідомлень

Додаток повинний бути в змозі обробити будь-яке повідомлення. Практично для кожної події Windows має стандартну процедуру обробки. У додатку повинно бути зазначено, що при виникненні події XY, для якої в програмі немає спеціального тексту, повинна виконуватися стандартна процедура Windows.

Більшість стандартних повідомлень програми обробляються автоматично, оскільки всі об'єкти Delphi мають убудований механізм — *процедуру обробки повідомлень* (Message Handler).

У кожнім з компонентів Delphi визначений механізм керування повідомленнями, що переводить усі повідомлення Windows, що посилаються визначеному об'єкту, у виклики методів.

1.2.3. Основні поняття

Як уже ясно з назви, сутність об'єктно-орієнтованого програмування полягає у використанні об'єктів. ООП являє собою розширення традиційних мов програмування (наприклад, C або Pascal) новим структурованим типом даних — класом.

Найбільш істотні відмінні риси ООП — це застосування класів, спадкування (inheritance), інкапсуляції (encapsulation) і поліморфізму (poly -багато, morphe — вид, форма). Нижче ці основні поняття ООП роз'ясняються докладніше.

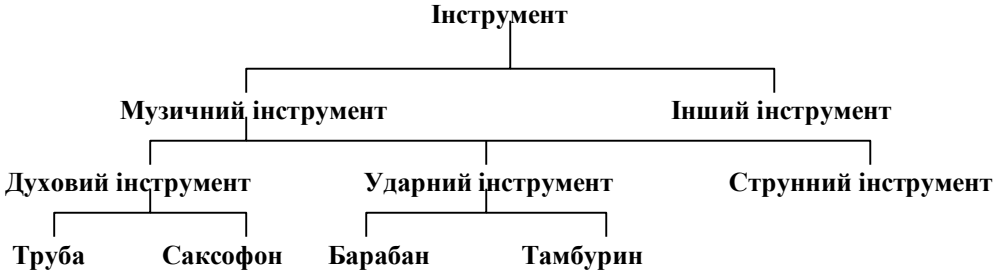
1.2.3.1. Класи

Клас — це абстрактне поняття, порівнянне з поняттям *категорія* в його звичайному змісті.

По визначених властивостях будь-якого елемента визначеної категорії можна встановити, що він належить до цієї категорії. Сама категорія визначається загальними властивостями, що мають всі екземпляри цієї категорії.

Це можна пояснити на прикладі музичних інструментів. Музичні інструменти поділяються на наступні категорії: духові, ударні і струнні.

Усі ці категорії належать до категорії музичних інструментів. У свою чергу, категорія музичних інструментів входить у категорію інструментів. На мал. 1.2.1 ця структура категорій графічно представлена у виді дерева.



Малюнок 1.2.1 – Дерево категорій

Музичні інструменти мають загальні властивості, але кожен інструмент сам по собі має особливі властивості, що визначають його призначення і відрізняють його від інших

інструментів. По тому ж принципу можна описати і класи в ООП. Певний музичний інструмент деякої категорії, наприклад моя труба або ваша труба, є об'єктом. Категорія, до якої цей інструмент належить, — це клас.

Клас в ООП — це абстрактний тип даних, що включає не тільки дані, але і функції і процедури.

Функції і процедури класу називаються методами і містять вихідний код, призначений для обробки внутрішніх даних об'єкта даного класу.

1.2.3.2. Наслідування

Класи містять *дані* і *методи*. В ООП методи і дані одного класу можуть передаватися іншим класам, тобто об'єкти можуть успадковувати властивості один одного. Клас, що успадковує властивості іншого класу, має ті ж можливості, що і клас, від якого він породжений. Цей принцип називається *спадкуванням* (inheritance). Породжений клас називається *нащадком* (descendant), а той, від якого він породжений — *предком* (ancestor). Завдяки новим властивостям, якими доповнюється нащадок, породжений клас може мати більші можливості, ніж його предок.

У прикладі дерева категорій музичних інструментів клас **Труба** відходить безпосередньо від класу **Духові інструменти**. У такий спосіб уже визначено, що в трубу можна дути, і цю її властивість не треба перевизначати заново.

Той же принцип дотримується і для властивості музичних інструментів видавати музичні звуки. Ця властивість класу **Музичні інструменти** перенесена на клас **Труба** через його прямого предка, клас **Духові інструменти**.

Механізм спадкування забезпечує можливість багаторазового застосування програмного коду. Таким чином, класи можуть бути представлені у виді ієрархії. Бібліотека VCL (Visual Component Library) у Delphi і є такою ієрархічною системою класів.

1.2.3.3. Інкапсуляція

Сполучення даних і методів їхньої обробки в одному об'єкті називається *інкапсуляцією* (encapsulation). Методи об'єкта визначають способи зміни даних.

Здатність духового інструмента видавати звуки обумовлена продуванням повітря через мундштук. Для духового інструмента характерна наявність мундштука, у який треба дути. Іншими словами, мундштук можна використовувати тільки визначеним способом, що відноситься до духового інструмента. Ніякий інший спосіб не годиться.

Представте, що вам треба написати програму, яка виконувала би дует духового і струнного інструментів. Для цього визначте класи **Духовий інструмент** і **Струнний інструмент**. Для класу **Духовий інструмент** визначите, що мається мундштук і що в нього треба дути, щоб одержати звук. Для класу **Струнний інструмент** визначите, що по струнах треба вдаряти, щоб одержати звук.

Обидва класи вже здатні грати музику, але це їхня властивість була успадкована від їхнього предка. Вони успадкували метод PlayMusic, що оголошений і реалізований як метод класу **Музичний інструмент**.

Таким чином, цей метод уже не потрібно створювати, і вам не треба знати код реалізації методу, щоб використовувати його в двох нових класах. Спосіб, яким реалізована можливість грати музику, не важливий. Цей принцип приховання інформації (information hiding) характерний для інкапсуляції й істотно полегшує написання великих і стабільно працюючих додатків.

Якщо клас був грамотно сконструйований і ретельно перевірений, він може багаторазово використовуватися в різних додатках. У прикладі з музичними інструментами це означає, що кожен клас має властивості класу **Музичний інструмент**, а способи створення звуку укладені усередині нього. Ці способи невидимі і недоступні за межами класу.

Якщо духовий інструмент одержує завдання грати, то цей інструмент "знає", що для цього треба дути в його мундштук, тому що це визначено в його класі.

Струнний інструмент, що одержує те ж завдання, не може використовувати мундштук духового інструмента, щоб дути на свої струни. Такого не може статися ще і тому, що мундштук

— це частина класу **Духовий інструмент**, а не класу **Струнний інструмент**. Це має на увазі, що обидва класи нічого не знають один про одного. Вони цілком розділені, і їм не відомі специфікації і властивості один одного. **Духовий інструмент** закритий для будь-яких спроб інших класів використовувати його мундштук. Також і струни **Струнного інструмента** і спосіб їхнього використання укладені усередині нього самого. Об'єкт *закритий*, тобто оточення не може випадково змінити цей об'єкт.

Зміст цієї закритості в тім, що ви не обов'язково повинні знати, як, наприклад, труба видає звук. Той факт, що для одержання звуку потрібний мундштук, схований у глибині класу **Духовий інструмент**. Не має значення, з яким мундштуком це відбувається і як. Це подробиці, що враховуються в самому об'єкті і не мають значення для його оточення. Вам необхідно лише викликати функцію PlayMusic.

1.2.3.4. Поліморфізм

Інша важлива концепція ООП — *поліморфізм*. Це означає, що той самий метод виконується по-різному для різних об'єктів. Наприклад, метод класу **Музичний інструмент** — PlayMusicForAnOrchestra — може бути визначений як загальний метод, що може використовуватися з будь-якою категорією музичних інструментів. Цей метод написаний таким чином, що не важливо, який саме інструмент одержує завдання грати, однак для класів, що описують конкретні інструменти, даний метод повинний бути *перевизначений* (override), що дасть можливість визначити конкретні дії, що враховують особливості даного інструмента

1.2.4 Реалізація ООП у Delphi

Перевага застосування об’єктно-орієнтованих методів при розробці додатків для Windows полягає в тому, що ви використовуєте попередньо створені об'єкти — елементи керування Windows (наприклад, вікна, кнопки і т.д.) Крім того, ви можете використовувати їх як предків для нових, визначених вами компонентів. Завдяки цьому ваші додатки здобувають характерний для Windows вид і вам не потрібно заново "винаходити колесо". У той же час ви можете робити зміни по своєму смаку. У Delphi це можна виконати значно простіше, ніж в інших мовах програмування, наприклад у C++.

1.2.4.1. Новий тип даних: клас

Концепція об’єктно-орієнтованого програмування припускає використання нового типу даних — *клас*. Тип "клас" належить до сукупності відомих у Delphi структурованих типів: множина, масив, запис і клас. Особливість типу "клас" полягає в тому, що він містить методи і властивості. Це значить, що клас описує групу даних і одну (або більш) процедуру або функцію, що має доступ до цих даних. Процедури і функції називаються *методами*. Тип "клас" — це структура даних, що складається з деякої кількості елементів:

- полів;
- методів;
- властивостей.

Поля містять дані визначеного типу. Методи — це функції і процедури, що виконують визначені дії. Властивості — це поля даних, що впливають на поводження об'єкта. Вони служать для опису об'єкта і відрізняються від звичайних полів тим, що присвоєння їм значень пов'язане з викликом методів.

1.2.4.2. Оголошення типів

Кожен новий клас у Delphi повинний бути оголошений. Для цього використовується зарезервоване слово class. Оголошення визначає функціональні можливості класу. У Delphi-версії мови Object Pascal новий клас з'являється в такий спосіб:

TNew = **class**(TOld);

Для оголошення класів у модулі відведений особливий розділ, що так і називається: *розділ оголошення типів.*

Класи повинні бути оголошені на рівні програми або на рівні модуля і не можуть бути оголошені усередині процедури або функції.

Зарезервоване слово class використовується для оголошення класу або методу класу.

Оголошення поля складається з ідентифікатора, що позначає поле, і типу даних поля. Оголошення методу складається з заголовка процедури або функції. У визначенні властивості вказується ідентифікатор властивості і методи його використання.

1.2.4.3. Об'єкти, класи та екземпляри

Клас і об'єкт — часто уживані терміни ООП. Однак іноді ці терміни вживаються неправильно. В даний час в ООП прийняте наступне визначення цих понять.

Клас — визначений користувачем тип даних, що має внутрішні дані і методи у формі процедур або функцій і звичайно описує родові ознаки і способи поводження ряду дуже схожих об'єктів. Об'єкт є *екземпляром класу*. Попередньо визначені об'єкти, використовувані в програмі (такі як, наприклад, компоненти Delphi), — це в дійсності екземпляри класів.

У мові програмування Delphi екземпляр класу реалізується змінною визначеного типу класу. Нижче приведений приклад оголошення об'єктів.

Приклад 1.2.1 – Оголошення класу

```
type
TForml = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  CloseBtn: TBitBtn
  OKBtn: TBitBtn
end;
var
  Forml: TForml;
```

В оголошенні типу визначений новий клас — TForml, наслідуваний від класу TForm, що утримується в VCL. На це вказує зарезервоване слово class. Даний тип містить покажчики на компоненти, що були поміщені у форму: два компоненти Label — об'єкти типу TLabel (або, інакше кажучи, екземпляри класу TLabel) і два екземпляри класу TBitBtn.

Для визначення екземпляра нового класу оголошена змінна Forml.

Після того як зроблені всі оголошення, можна дуже просто створити і ініціалізувати нові екземпляри класів (або об'єкти).

Отже, можна зробити наступні висновки.

- ООП відрізняється від колишніх методів програмування можливістю створення нових об'єктів.
- Екземпляри класів можуть поводитися різним чином. Кожен об'єкт має у своєму розпорядженні копію полів даних типу клас, але різні об'єкти мають різні поля і методи.
- Змінна може містити посилання на екземпляр класу (об'єкт). Змінна містить не сам об'єкт, а вказує на зарезервовану для нього область пам'яті. Як і змінні-покажчики, кілька змінних-об'єктів можуть посилатися на той самий об'єкт. Змінна-об'єкт може мати і нульове значення; це значить, що вона в даний момент ні на що не вказує.

1.2.4.4. Область видимості

Докладний опис поняття інкапсуляції пов'язаний з поняттям *області видимості ідентифікатора*. Область видимості ідентифікатора (імені змінної, процедури, функції або типу даних) - це частина програмного коду, у якій можливий доступ до цього ідентифікатора.

Область видимості ідентифікатора компонента, оголошеного в описі класу, простирається від його оголошення до кінця визначення класу, а також поширюється на всі нащадки цього класу і на всі блоки реалізації методів класу.

Область видимості ідентифікатора компоненту залежить від *атрибута видимості* розділу, у якому оголошений цей ідентифікатор. У Delphi використовується п'ять атрибутів видимості (називаних також директивами): published, public, protected, private і automated.

В оголошеннях типів класів маються розділи приватних (private) і загальних (public) оголошень. У розділі приватних оголошень розміщаються поля даних і методи, недоступні за межами модуля, що містить оголошення даного класу. Дані, описані в цьому розділі, можуть оброблятися тільки шляхом виклику методів усередині класу, а також усередині даного модуля. За межами класу всі його приватні елементи невідомі і вважаються неіснуючими.

Поля даних і методи, оголошені в розділі загальних оголошень класу, доступні для всіх процедур, програмний код яких розташований в області видимості даного об'єкта. У розділі загальних оголошень типу класу повинні бути оголошені поля даних і методи, до яких будуть мати доступ методи об'єктів інших модулів.

З атрибутом видимості protected з'являються ті елементи, до яких за межами даного модуля можуть мати доступ тільки методи класів, породжених від даного класу.

Директива published схожа на інші атрибути видимості (private, public і protected) тим, що вона може зустрічатися тільки в оголошенні типу класу. Опубліковане (published) поле або метод може використовуватися не тільки під час виконання програми, але і під час її розробки. Усі компоненти в палітрі компонентів Delphi мають у своєму розпорядженні published-інтерфейс, що використовується в першу чергу інспектором об'єктів. Правила видимості для директиви published — ті ж, що і для public.

Розходження між *загальними* (public) і *опублікованими* (published) елементами полягає в тому, що під час виконання програми можна одержати інформацію про типи (RTTI — Run-time type information) опублікованих елементів класу. За допомогою цієї інформації в додатку можна динамічно визначити і використовувати поля і властивості кожного, у тому числі і невідомого, типу класу.

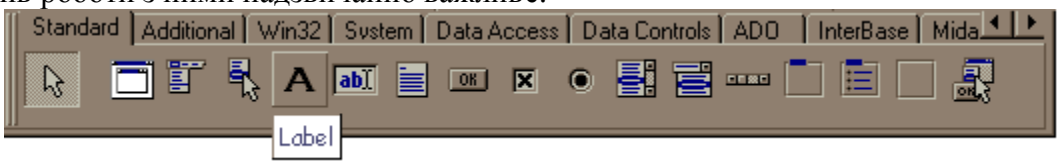
Директива automated уведена тільки для сумісності з Delphi 2.0. Новий клас TAutoObject у Delphi 4 цю директиву вже не використовує.

Література [2-3].

Тема 1.3. Візуальні компоненти

1.3.1. Категорії візуальних компонентів

Візуальний компонент – це об'єкт на екрані, такий як кнопка або меню, який можна помістити в екранну форму. По суті, екранна форма – це теж візуальний компонент, що містить у собі інші, більш прості, об'єкти компонентів. У процесі розробки додатків у середовищі Delphi значна частка часу витрачається на включення в проект і модифікацію візуальних компонентів, так що розуміння основних функціональних можливостей такого роду об'єктів і освоєння базових прийомів роботи з ними надзвичайно важливе.



Малюнок 1.3.1 – Сторінка Standard палітри компонентів зі спливаючою підказкою

Перелік категорій компонентів Delphi включає наступне.

- ***Standard***. Стандартні елементи керування Windows – кнопки, етикетки, поля введення, списки, прапорці і смуги прокручування.
- ***Additional***. Адаптований набір елементів – кнопки з графічними етикетками, кнопки для панелей інструментів, різні варіанти сіток, бітові зображення, штрихові графічні фігури, вікна зі смугами прокручування, групи прапорців, прості елементи для побудови діаграм.
- ***Win32***. Стандартні елементи керування 32-бітових операційних систем Windows 9x і Windows NT; у цю категорію включені сторінки і вкладки багатосторінкових діалогових вікон, списки зображень, текстовий редактор з розширеними можливостями, лінійні індикатори і регулятори, вікна з анімацією, вікна для перегляду деревоподібних і лінійних списків, рядок стану і панелі інструментів.
- ***Data Access***. Компоненти для організації доступу до баз даних за допомогою таблиць і запитів SQL і для формування звітів разом з компонентами категорії QReport.
- ***Data Control***. Елементи керування для роботи з даними; у їхнє число входять навігатори, поля редагування однорядкового і багаторядкового тексту, графічні зображення, прапорці і списки.
- ***QReport***. Надзвичайно широкий набір компонентів для формування звітів на основі інформації з баз даних.
- ***Dialogs***. Стандартні діалогові вікна загального призначення, що використовуються в процесі виконання таких операцій, як вибір файлу або каталогу, вибір шрифту або кольору, друк, пошук і заміна в текстовому документі.
- ***System***. Системні сервісні компоненти.
- ***Internet***. Компоненти для роботи з Internet і World Wide Web.
- ***Decision Cube***. Компоненти для створення складних звітів.
- ***Win3.1.*** Компоненти для операційного середовища Windows 3.1.
- ***Samples***. Зразки компонентів.
- ***ActiveX***. Компоненти для програмування елементів керування ActiveX і т.д.

1.3.2. Властивості компонентів загального призначення

Властивість Enabled. Визначає, чи належить елементу керування реагувати на події миші, клавіатури або таймера. Якщо властивість Enabled має значення True, то елемент керування реагує на події, інакше ці події ігноруються.

property Enabled: Boolean;

Властивість Parent. У властивості Parent утримується посилання на *батьківський* елемент керування (Parent), що містить даний елемент. Якщо батьківський елемент містить інші елементи керування, вони називаються *дочірніми* елементами (Child). Наприклад, якщо в додатку міститься рамка, що групує, із трьома опціями, то ця рамка є батьківським елементом трьох опцій. Останні, у свою чергу, є дочірніми елементами рамки, що групує.

property Parent: TWinControl;

Властивість Controls. Це масив покажчиків на всі дочірні компоненти даного елемента керування, за допомогою цього масиву можна звернутися до дочірнього елемента не по імені, а по порядковому номеру.

property Controls[Index: Integer]: TControl;

Властивість ControlCount. Містить кількість дочірніх елементів даного елемента керування. Значення ControlCount завжди на 1 більше, ніж максимально припустимий індекс для масиву Controls.

property ControlCount: Integer;

1.3.3. Положення елементів керування

Властивість Align. Визначає, як розташовується елемент керування усередині свого батьківського елемента.

Таблиця 1.3.1 – Значення властивості Align форми

Значення	*Положення елемента керування у формі*
alNone	Елемент залишається там, де він був поміщений у форму (установка за замовчуванням)
alTop	У верхнього краю по всій ширині батьківського елемента
alBottom	У нижнього краю по всій ширині батьківського елемента
alLeft	У лівого краю по усій висоті батьківського елемента
alRight	У правого краю по усій висоті батьківського елемента
alClient	Уся робоча область батьківського елемента

property Align: TAlign;

Властивість Brush. Визначає колір і зразок заливання елемента керування.

property Brush: TBrush;

Властивість Cursor. Визначає зображення покажчика миші в той момент, коли він знаходиться на елементі керування.

property Cursor: TCursor;

Властивість Height. Визначає висоту елемента керування в пікселях.

property Height: Integer;

Властивість Width. Визначає ширину елемента керування в пікселях.

property Width: Integer;

Властивість Top. Містить вертикальну координату лівого верхнього кута елемента керування в пікселях щодо форми або батьківського елемента керування, у якому даний елемент утримується.

property Top: Integer;

Властивість Left. Містить горизонтальну координату лівого краю елемента керування в пікселях щодо форми або батьківського елемента керування, у якому даний елемент утримується.

property Left: Integer;

1.3.4. Видимість компонентів

Властивість Visible. Визначає, чи буде даний компонент відображатися на екрані. Якщо властивість Visible має значення True, то компонент видний користувачеві.

Метод Show. Робить форму або елемент керування видимими і привласнює властивості Visible значення True.

Метод Hide. Робить елемент керування невидимим, привласнюючи його властивості Visible значення False.

Література [1].

Тема 1.4. Екранні форми та створення списків

1.4.1. Екранні форми як компоненти Delphi

1.4.1.1. Екранні форми і модулі, збереження екранних форм

Екранна форма представляє в проекті Delphi не тільки зовнішній вигляд вікна додатка, але і є повноцінним компонентом, що має власні властивості і події. Але, на відміну від інших компонентів, вона не представлена в палітрі компонентів Delphi.

Створити в Delphi об'єкт екранної форми можна двома способами: або відкривши новий додаток, або вибравши команду File⇨New Form. У першому випадку створюється екранна форма для головного вікна додатка, а в другому – додаткові вікна, наприклад діалогове вікно About або вікно заставки додатка.

Коли створюється нова форма, Delphi відкриває перед розроблювачем два нових вікна – вікно проектування форми і вікно редактора коду. У першому можна скомпонувати зовнішній вигляд вікна – розташувати в ньому елементи керування. Друге дає можливість запрограмувати мовою Object Pascal дії, що будуть виконуватися екранною формою як об'єктом додатка. Текст програми в цьому вікні називається *програмним модулем* (або просто *модулем*).

Delphi автоматично створює програмний модуль Object Pascal для кожної нової екранної форми в додатку. Небажано поєднувати в одному модулі коди програм, що відносяться до декількох екранних форм. Але, хоча кожній екранній формі і відповідає окремий модуль, не кожен модуль у проекті Delphi може мати зв'язану з ним екранну форму. Деякі модулі містять дані і програмні елементи, зв'язані з іншими модулями.

При збереженні нового проекту не призначайте властивості Name екранної форми те ж значення, що і файлу проекту. Наприклад, якщо властивості Name форми привласнене значення MainForm, не зберігайте файл модуля під ім'ям MainForm.pas (краще використовувати ім'я Main.pas). Оскільки внутрішнє ім'я модуля є ім'я файлу мінус розширення .pas, використання однакових імен для об'єкта форми і модуля приведе до конфлікту імен.

При установці значення властивості Name екранної форми завжди додавайте наприкінці слова Form пояснювальний текст. Наприклад, для властивості Name головного вікна додатка можна вибрати значення FormMain, для діалогового вікна About –FormAbout і т.д.

При збереженні файлу модуля привласніть йому ім'я, що відповідає назві екранної форми мінус слово Form. Наприклад. для модуля форми FormMain підійде ім'я файлу Main, для форми FormAbout – файл About і т.д. У результаті будуть створені файли – Main.pas і About.pas.

1.4.1.2. Деякі властивості екранних форм

Властивість ActiveControl. Цій властивості потрібно привласнити ім'я того об'єкта компонента (кнопки або прапорця), на який планується установити фокус уведення при появі вікна на екрані у працюючому додатку. Надалі фокус уведення буде переміщатися між елементами керування у вікні після натискання клавіші <Tab>.

property ActiveControl: TWinControl;

Доступно: для компонента TForm.

Властивість AutoScroll. Якщо привласнити цій властивості значення True, до вікна екранної форми буде підключена смуга пркрутки. Вона буде з'являтися або зникати в працюючому додатку в залежності від того, чи уміщаються усі компоненти в полі вікна при поточному настроюванні його розмірів.

property AutoScroll: Boolean;

Доступно: для компонентів TForm, TScrollBox, TTabSet.

Властивість HorzScrollBar. Ця властивість установлює значення складових для горизонтальної смуги прокручування. Смуга прокручування з'являється на екрані тільки у випадку, якщо складова Visible цієї властивості має значення True і складова Range має значення, більше чим ClientWidth.

property HorzScrollBar: TControlScrollBar;

Доступно: для компонентів TForm, TScrollBox.

Властивість Icon. Ця властивість задає піктограму, що зв'язується з об'єктом екранної форми.

property Icon: TIcon;

Доступно: для компонента TForm.

Властивість Menu. Ця властивість визначає, який компонент MainMenu буде використаний у якості головного меню вікна.

property Menu: TMainMenu;

Доступно: для компонента TForm.

Властивість PixelsPerInch. Ця властивість визначає, що додаток створює вікно форми, масштабоване відповідно до показника *піксель/дюйм*. Використовувати його потрібно в комбінації з властивістю Scaled. У результаті можна створити форму, що буде виглядати ідентично при будь-якій роздільній здатності дисплея.

property PixelsPerInch: Integer;

Доступно: для компонента TForm.

Властивість Position. Ця властивість визначає розмір і положення форми з появою її на екрані комп'ютера під час виконання додатка.

Таблиця 1.4.1 – Можливі значення властивості Position компонента TForm

Значення	***Положення форми***
poDesigned	Те ж положення і ті ж розміри, що і при розробці додатка.
poDefault	Delphi автоматично встановлює положення на екрані, висоту і ширину форми.
poDefaultPosOnly	Ті ж розміри, що і під час розробки додатка. Windows визначає лише положення форми на екрані.
poDefaultSizeOnly	Зберігається положення, у якому форма знаходилася при розробці; Windows визначає лише її висоту і ширину.
poScreenCenter	Ті ж розміри, що і під час розробки додатка. Форма розміщається в центрі вікна.

property Position: TPosition;

Доступно: для компонентів TForm, TScrollBar.

Властивість Scaled. Визначає, чи буде форма масштабуватись у відповідності зі значенням властивості PixelsPerInch. Якщо Scaled має значення True, а значення PixelsPerInch відрізняється від роздільної здатності системи, то розміри форми відповідно зміняться. Якщо Scaled має значення False, то розміри форми не зміняться, незалежно від значення властивості PixelsPerInch.

property Scaled: Boolean;

Доступно: для компонента TForm.

1.4.1.3. Деякі події екранних форм

OnActivate. Подія активізується, коли додаток активізує форму. Наприклад, якщо вікно екранної форми одержує фокус введення або якщо ви переключаєтеся на даний додаток, попрацювавши з іншими.

OnClose. Подія активізується, коли додаток закриває вікно екранної форми.

OnCloseQuery. Подія активізується безпосередньо перед закриттям вікна екранної форми, коли додаток викликає процедуру Close.

OnCreate. Подія активізується, коли програма формує об'єкт екранної форми в пам'яті.

OnDeactivate. Подія активізується, коли користувач “іде” з додатка, не закриваючи його, наприклад переключається на роботу з іншим відкритим додатком.

OnDestroy. Подія активізується перед виконанням процедури – деструктора об'єкта.

OnHide. Ця подія використовується для програмування дій, сполучених із прихованням вікна.

OnShow. Подія активізується безпосередньо перед тим, як вікно з'явиться на екрані.

1.4.2. Створення списків

1.4.2.1. Компоненти списків

- ***TListBox.*** Універсальний елемент керування Windows, що дозволяє створювати списки рядків, пункти яких користувачі можуть вибирати за допомогою миші або клавіатури.

Об'єкт ListBox містить масив типу TStrings з Items, що забезпечує доступ до інформації в ListBox.

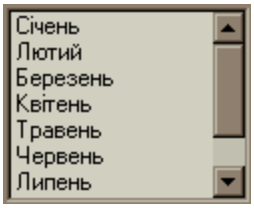
Категорія палітри: Standard.

- ***TComboBox.*** Цей стандартний елемент керування Windows об'єднує у собі компонент TListBox і поле редагування (об'єкт компонента TEdit). Працюючи з об'єктом ComboBox, користувач може або вибрати один з пунктів у вікні списку, що розкривається, або ввести нові дані в поле редагування.

  Категорія палітри: Standard.

1.4.2.2. Списки (компонент TListBox)

Властивість Style. Визначає як відображаються елементи в списку. Змінюючи значення Style, можна створювати нестандартні списки. Такі списки можуть містити графічні елементи і рядки різної висоти.



Малюнок 1.4.1 - Об'єкт ListBox зі списком рядків

Таблиця 1.4.2 – Можливі значення властивості Style компонента TListBox

Значення	*Дія*
lbStandard	Всі елементи – рядка однакової висоти
lbOwnerDrawFixed	Висота кожного елемента відповідає значенню властивості ItemHeight
lbOwnerDrawVariable	Елементи списку можуть розрізнятися по висоті

property Style: TListBoxStyle;

Доступно: для компонента TListBox.

Властивість ItemHeight. Установлює висоту елемента у визначеному розроблювачем списку в тому випадку, якщо властивість Style має значення lbOwnerDrawFixed.

property ItemHeight: Integer;

Доступно: для компонентів TListBox, TComboBox.

Властивість IntegralHeight. Визначає, чи буде відображатися на екрані елемент списку, що видний не цілком.

property IntegralHeight: Boolean;

Властивість Sorted. Визначає, чи розташовуються елементи списку або поля зі списком за абеткою.

property Sorted: Boolean;

Доступно: для компонентів TListBox, TComboBox.

Метод Add. Додає в кінець списку новий елемент і повертає позицію (індекс) даного елемента в списку. Перший елемент списку має індекс 0.

function Add(Item: Pointer): Integer;

Доступний: для об'єктів TList.

Метод Insert. Вставляє елемент між іншими елементами в список об'єкта TList. Ім'я елемента, що вставляється, необхідно передати через параметр Item, а позицію, у яку необхідно вставити елемент у списку, - через параметр Index. Індексація елементів списку починається з нуля.

procedure Insert(Index: Integer; Item: Pointer);

Доступний: для об'єктів TList.

1.4.2.3. Поля зі списками (компонент TComboBox)

Властивість DropDownCount. Визначає максимальну кількість одночасна відображуваних елементів у списку компонентів, що відкривається, TComboBox. За замовчуванням установлене значення 8.

property DropDownCount: Integer;

Властивість Text. Містить рядок, що відображається у вікні введення даних того або іншого компонента.

property Text: TCaption;

Властивість MaxLenghth. Визначає, яке максимальне число символів можна ввести в поле введення, елемент керування Memo або поле зі списком. За замовчуванням прийняте значення 0. Це означає, що обмежень на кількість символів немає.

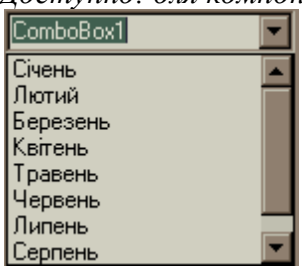
property MaxLength: Integer;

Доступно: для компонентів TEdit, TComboBox, TMaskEdit, TMemo.

Властивість SelText. Містить виділену в полі введення або в області введення поля зі списком частину тексту. За допомогою цієї властивості можна зчитувати або змінювати виділений текст.

property SelText: String;

Доступно: для компонентів TEdit, TComboBox, TMaskEdit.



Малюнок 1.4.2 - Об'єкт ComboBox зі списком рядків

Властивість SelLength. Повертає довжину виділеної в полі введення частини рядку (у символах).

property SelLength: Integer;

Доступно: для компонентів TEdit, TComboBox, TMaskEdit, TMemo.

Метод SelectAll. Виділяє в елементі керування весь текст. Для виділення частини тексту варто використовувати властивості SelStart і SelLength.

procedure SelectAll;

Доступний: для компонентів TEdit, TComboBox, TMaskEdit, TMemo

Література [1].

РОЗДІЛ 2 ІНТЕРФЕЙС КОРИСТУВАЧА

Тема 2.1. Використання кнопок та перемикачів

2.1.1. Компоненти

- ***TBevel.*** Цей візуальний компонент нагадує прямокутне поглиблення у вікні. З його допомогою можна відображати горизонтальні і вертикальні лінії, що проектувальники інтерфейсів часто використовують для розбивки вікон на зони. **Категорія палітри:** Additional.
- ***TBitBtn.*** Працює подібно кнопці, але на ньому часто відображається фрагмент картинки, називаний гліфом, що представляє функціональне призначення елемента.

Категорія палітри: Additional.

- ***TButton.*** Це стандартна кнопка Windows.
  Категорія палітри: Standard.
- ***TCheckBox.*** Стандартний елемент керування Windows, що представляє собою прапорець опції, що користувачі можуть встановлювати і скидати. Поруч з ним може відображатися пояснювальний текст.
  Категорія палітри: Standard.
- ***TGroupBox.*** Використовується для угруповання компонентів TRadioButton і інших об'єктів.
  Категорія палітри: Standard.
- ***TPanel.*** За допомогою цього компонента можна розбивати вікно на сегменти, а також створювати панелі інструментів і панелі стану.
  Категорія палітри: Standard.
- ***TRadioButton.*** Кілька об'єктів RadioButton можна поєднувати за допомогою компонента GroupBox або Panel, але в більшості випадків ці елементи простіше групувати за допомогою RadioGroup.
  Категорія палітри: Standard.
- ***TRadioGroup.*** Цей компонент нагадує GroupBox. Він істотно спрощує створення групи з декількох об'єктів RadioButton.
  Категорія палітри: Standard.
- ***TSpeedButton.*** Звичайно компонент TSpeedButton застосовуються при створенні панелей інструментів, однак об'єкти цього компонента можна вставляти безпосередньо у форми.
  Категорія палітри: Additional.

2.1.2. Основні кнопки

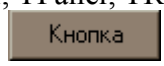
2.1.2.1. Кнопки виклику команд (компонент TButton)

Звичайні кнопки, застосовувані для виклику команд, прості у використанні. Досить уключити відповідний об'єкт у форму і задати значення властивості Caption. Щоб виконувати дії в результаті щиглика на кнопці, треба створити оьроблювач події OnClick і включити в нього необхідні оператори.

Властивість Caption. Для кнопок властивість Caption – це послідовність символів, що відображається на кнопці. За допомогою символу & (амперсанд) можна задати клавішу швидкого доступу. Символ, що міститься за амперсандом, зображується підкресленим. Після цього користувач може вибрати елемент керування за допомогою клавіші <Alt> і відповідного підкресленого символу.

property Caption: String;

Доступно: для компонентів TBitBtn, TButton, TCheckBox, TForm, TGroupBox, TLabel, TPanel, TRadioButton, TRadioGroup, TSpeedButton і т.д.



Малюнок 2.1.1 – Об'єкт Button

Властивість Cancel. Указує, чи є компонент TButton або TBitBtn кнопкою скасування. Якщо цій властивості привласнене значення True, то при натисканні користувачем клавіші <Esc> виконується процедура обробки події OnClick для даної кнопки. Хоча додаток може містити більше однієї кнопки Cancel, форма викликає процедуру обробки події OnClick лише для першої в послідовності табулятора кнпки.

property Cancel: Boolean;

Доступно: для компонентів TBitBtn, TButton.

Властивість Default. Указує, чи визначена кнопка як кнопка за замовчуванням. Якщо цій властивості привласнене значення True, то при натисканні користувачем клавіші <Enter> виконується процедура обробки події OnClick для даної кнопки.

property Default: Boolean;

Доступно: для компонентів TBitBtn, TButton.

Властивість ModalResult. Використовується для закриття модальних форм. За замовчуванням ModalResult має значення 0. якщо цій властивості привласнити інше значення, форма закривається і функція ShowModal повертає значення властивості ModalResult. Коли користувач закриває модальну форму, властивість ModalResult одержує ненульове значення і форма закривається.

Кнопки також мають властивість ModalResult. Його варто застосовувати, коли необхідно закрити модальну форму щигликом на кнопці. Наприклад, при створенні діалогового вікна з кнопками OK і Cancel варто привласнити властивості ModalResult кнопки OK значення mrOk, а властивості кнопки Cancel - значення mrCancel. Коли користувач виконає щиглик на одній з цих кнопок, модальне діалогове вікно закриється, тому що значення властивості ModalResult буде більше, ніж mrNone. Для використання властивості ModalResult не потрібна процедура обробки події.

Таблиця 2.1.1 – Значення властивості ModalResult для форм і кнопок

Константа	***Значення***
mrNone	0
mrOk	idOk
mrCancel	idCancel
mrAbort	idAbort
mrRetry	idRetry
mrIgnore	idIgnore
mrYes	idYes
mrNo	idNo
mrAll	mrNo+1

property ModalResult: TModalResult;

Доступно: для компонентів TForm, TBitBtn, TButton.

2.1.2.2. Прапорці опцій (компонент TCheckBox)

Властивість AllowGrayed. Визначає, чи може опція знаходитися в двох або трьох станах. Якщо властивість AllowGrayed має значення False (установка за замовчуванням), то опція може знаходитися тільки в двох станах: включена або виключена. Якщо властивість AllowGrayed має значення True, то таких можливих станів три:

- включена;
- виключена;
- включена частково, тобто деякі з вкладених опцій включені, а деякі ні (зображується сірим кольором).

property AllowGrayed: Boolean;

Доступно: для компонента TCheckBox.

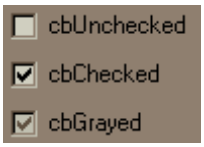
Властивість State. Визначає стан, у якому знаходиться опція.

Таблиця 2.1.2 – Значення властивості State компонента TCheckBox

Значення	***Дія***
cdUnchecked	Опція виключена – не має помітки “пташка”
cdChecked	Опція включена – має помітку, яка вказує, що користувач вибрав дану установку
cdGrayed	Опція включена частково – має помітку, однак зображується сірим кольором

property State: TCheckBoxState;

Доступно: для компонента TCheckBox.



Малюнок 2.1.2 – Об'єкти CheckBox з різними значеннями властивості State

2.1.2.3. Кнопки із залежною фіксацією (компонент TRadioButton)

Властивість Alignment. Визначає положення тексту щодо цього елемента.

Таблиця 2.1.3 – Значення властивості Alignment компонентів TCheckBox і TRadioButton

Значення	*Дія*
taLeftJustify	Текст з'являється ліворуч від опції
taRightJustify	Текст з'являється праворуч від опції

property Alignment: TLeftRight;

Доступно: для компонентів TCheckBox, TRadioButton.

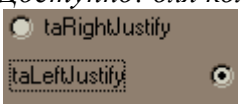
Властивість Checked. Визначає, чи обрана дана опція або даний пункт меню. На екрані це зображується позначкою у вигляді чорної крапки.

Таблиця 2.1.4 – Значення властивості Checked компонентів TCheckBox, TRadioButton і TMenuItem

Значення	*Дія*
True	З'являється позначка, що говорить про те, що опція або пункт меню обрані
False	Позначки немає, опція або пункт меню не обрані

property Checked: Boolean;

Доступно: для компонентів TCheckBox, TRadioButton, TCheckListBox, TMenuItem.



Малюнок 2.1.3 – Об'єкти RadioButton з різними значеннями властивостей Alignment і Checked

2.1.3. Яскраві кнопки

2.1.3.1. Кнопки із зображенням (компонент TBitBtn)

На елементах керування TBitBtn і TSpeedButton можуть бути поміщені зображення.

Властивість Glyph. Визначає зображення, що з'являється на кнопці. Під час проектування ви можете завантажити графічний файл у форматі BMP за допомогою щиглика на кнопці з трьома крапками у колонці значень властивості Glyph у вікні Object Inspector.

Кожна кнопка TBitBtn або TSpeedButton може містити максимум чотири зображення. Від стану кнопки залежить, яке з них відображається в даний момент. Завдяки цьому користувач може довідатися, у якому стані знаходиться ця кнопка. У табл. 2.1.5 показано, якому стану кнопки який номер зображення відповідає.

Таблиця 2.1.5 – Зв'язок між станами кнопок TBitBtn і TSpeedButton

Номер зображення	*Стан*	*Опис*
Перше	Up (Відпущена)	Зображення використовується, коли кнопка не натиснута. Якщо в растровому зображенні немає інших зображень, Delphi використовує це зображення й в інших випадках.
Друге	Disabled (Заборонена)	Зображення використовується, щоб показати, що ця кнопка не може бути обрана.

Номер зображення	*Стан*	*Опис*
Третє	Down (Натиснута)	Зображення використовується, коли користувач виконує щиглик на кнопці. Як тільки користувач відпускає кнопку миші, кнопка повертається в стан Up (Відпущена).
Четверте	Stay down (Залишається натиснутої)	Зображення використовується, коли кнопка залишається натиснутої, після того як користувач відпускає кнопку миші.

Якщо мається тільки одне зображення, Delphi намагається самостійно створити відсутні малюнки на основі заданого, хоча це не відноситься до стану Down (Натиснута). Якщо це вас не влаштовує, тоді необхідно привласнити властивості Glyph як значення растрове зображення, що містить кілька малюнків. У такому випадку у властивості NumGlyphs необхідно вказати їхню кількість. Усі малюнки повинні мати однакові розміри і знаходитися в растровому зображенні поруч один з одним.

property Glyph: TBitmap;

Доступно: для компонентів TBitBtn, TSpeedButton.

Властивість NumGluphs. Указує кількість малюнків, що утримуються в растровому зображенні властивості Glyph. Значення, привласнене цій властивості, використовується Delphi при промальовуванні зображення. Властивості NumGlyphs допускається привласнювати значення від 1 до 4. За замовчуванням прийняте значення 1.

property NumGlyphs: TNumGlyphs;

Доступно: для компонентів TBitBtn, TSpeedButton.

Властивість Layout. Визначає положення зображення на кнопці BitBtn або SpeedButton.

Таблиця 2.1.6 – Значення властивості Layout компонентів TBitBtn і TSpeedButton

Значення	*Дія*
blGlyphLeft	Зображення з'являється на кнопці ліворуч
blGlyphRight	Зображення з'являється на кнопці праворуч
blGlyphTop	Зображення з'являється на кнопці зверху
blGlyphBottom	Зображення з'являється на кнопці знизу

property Layout: TButtonLayout;

Доступно: для компонентів TBitBtn, TSpeedButton.

Властивість Spacing. Визначає відстань у пикселях між зображенням (зазначеним у властивості Glyph) і текстом (зазначеним у властивості Caption) на кнопці BitBtn або SpeedButton. За замовчуванням прийняте значення, рівне 4.

Якщо значення властивості Spacing – позитивне число, то між зображенням і текстом міститься відповідне число пікселей. Якщо значення властивості Spacing дорівнює 0, то між зображенням і текстом немає проміжку. Якщо значення Spacing дорівнює –1, то текст вирівнюється по центру між зображенням і краєм кнопки.

property Spacing: Integer;

Доступно: для компонентів TBitBtn, TSpeedButton.

Властивість Margin. Визначає відстань у пікселях між краєм зображення (зазначеним у властивості Glyph) і краєм кнопки. Місце виміру відстані залежить від взаємного розташування зображення і тексту, що зазначено у властивості Layout.

Якщо значення Margin дорівнює –1 (приймається за замовчуванням), то зображення і текст (зазначений у властивості Caption) вирівнюються по центру.

property Margin: Integer;

Доступно: для компонентів TBitBtn, TSpeedButton.

Властивість Style. Свойстово Style компонентів TBitBtn або TSpeedButton визначає зовнішній вигляд цих кнопок.

Таблиця 2.1.7 – значення властивості Style компонентів TBitBtn і TSpeedButton

Значення	*Дія*
bsAutuDetect	Вид кнопки відповідає версії Windows, у якій працює додаток
bsWin31	Використовується стандартний стиль Windows 3.1, незалежно від поточної версії Windows
bsNew	Використовується новий стиль, незалежно від поточної версії Windows

property Style: TButtonStyle;

Доступно: для компонентів TBitBtn, TSpeedButton.

Властивість Kind. Визначає, яке зображення і з яким текстом буде відображатися на кнопці BitBtn. Зображення і текст можна вибрати так, щоб користувач відразу знав, що відбудеться після щиглика на даній кнопці. Кілька різновидів кнопок заздалегідь визначені.

Таблиця 2.1.8 – Можливі значення властивості Kind компонента TBitBtn

Значення	*Дія*
bkCustom	Розроблювач сам визначає зображення на кнопці BitBtn. Для цього варто вказати потрібне растрове зображення у властивості Glyph. Як і для звичайних кнопок. Можна привласнити ненульове значення властивості ModalResult або написати програмний код процедури обробки події OnClick.
bkOK	На кнопці відображається зелена “пташка” і текст OK. Компонент стає кнопкою за замовчуванням (властивість Default автоматично здобуває значення True). Коли користувач виконує щиглик на такій кнопці, форма або діалогове вікно закривається. Значення ModalResult кнопки змінюється на mrOK.
bkCancel	На кнопці відображається червоний символ “X” і текст Cancel. Кнопка стає кнопкою скасування (властивість Cancel автоматично здобуває значення True). Коли користувач виконує щиглик на такій кнопці форма або діалогове вікно закривається. Значення ModalResult кнопки змінюється на mrCancel.
bkYes	На кнопці відображається зелена “пташка” (як і на кнопці OK) і текст Yes. Компонент стає кнопкою за замовчуванням (властивість Default автоматично здобуває значення True). Коли користувач виконує щиглик на такій кнопці, форма або діалогове вікно закривається. Значення ModalResult кнопки змінюється на mrYes.
bkNo	На кнопці відображається червоний символ заперечення (кружок, перекреслений по діагоналі) і текст No. Кнопка стає кнопкою скасування (властивість Cancel автоматично здобуває значення True). Коли користувач виконує щиглик на такій кнопці форма або діалогове вікно закривається. Значення ModalResult кнопки змінюється на mrNo.
bkHelp	На кнопці відображається блакитний знак питання і текст Help. Коли користувач виконує щиглик на цій кнопці, повинне з'явиться довідкове вікно.
bkClose	На кнопці відображаються дверцята і текст Close. Коли користувач виконує щиглик на такій кнопці форма закривається. Властивість Default кнопки має значення True.
bkAbort	На кнопці відображається червоний символ “X” і текст Abort. Властивість Cancel кнопки автоматично здобуває значення True.
bkRetry	На кнопці відображаються дві зелені стрілки і текст Retry.
bkIgnore	На кнопці відображаються зелений чоловічок і текст Ignore. Цю кнопку варто використовувати в додатку для того, щоб користувач міг продовжувати роботу після того, як відбудеться помилка.
bkAll	На кнопці відображаються дві зелені “пташки” і текст All. Властивість Default кнопки автоматично здобуває значення True

property Kind: TBitBtnKind;

Доступно: для компонента TBitBtn.

Малюнок 2.1.4 – Об'єкти BitBtn з різними значеннями властивості Kind

2.1.3.2. Компонент TSpeedButton

Властивість GroupIndex. За допомогою цієї властивості можна поєднувати кілька кнопок SpeedButton у групу. Звичайно кожна кнопка SpeedButton працює самостійно. За замовчуванням для GroupIndex прийняте значення, рівне 0, це значить, що кнопки не входять у групу.

При розміщенні декількох кнопок SpeedButton у формі або на панелі можна встановити, що кожна кнопка повинна працювати як складова частина групи кнопок. Це досягається присвоєнням властивості GroupIndex кожної кнопки того самого значення відмінного від нуля.

property GroupIndex: Integer;

Доступно: для компонента TSpeedButton.

Малюнок 2.1.5 – Об'єкти SpeedButton об'єднані в групу з зображеннями, визначеними у властивості Glyph

Властивість Down. Властивість Down кнопки SpeedButton визначає, у якому стані відображається кнопка: у натиснутому (Down) або в стані “відпущена” (Up). Якщо властивість Down має значення False, кнопка відображається на екрані як відпущена.

property Down: Boolean;

Доступно: для компонента TSpeedButton.

Властивість AllowAllUp. Визначає, чи можуть усі кнопки SpeedButton, що входять у групу, знаходитися одночасно в стані “відпущена”.

property AllowAllUp: Boolean;

Доступно: для компонента TSpeedButton.

2.1.4. Групи кнопок

2.1.4.1. Компоненти TPanel і TBevel

Компоненти TBevel і TPanel допоможуть вам поліпшити зовнішній вигляд вікна. Об'єкти Panel підтримують події OnClick і OnMouseDown. Компонент TBevel не підтримує події. За допомогою компонента TPanel можна створювати панелі інструментів і панелі стану.

Властивість BevelInner. Компонент-панель має дві рамки: зовнішню по краях елемента керування і внутрішню. Властивість BevelInner визначає вид внутрішньої рамки компонента-панелі.

Таблиця 2.1.9 – Значення властивості BevelInner компонента-панелі

Значення	*Дія*
bvNone	Внутрішньої рамки немає
bvLowered	Внутрішня рамка втиснена усередину; панель виглядає заглибленою
bvRaised	Внутрішня рамка видавлена назовні; панель виглядає піднятою

property BevelInner: TPanelBevel;

Доступно: для компонента TPanel.

Властивість BevelOuter. Компонент-панель має дві рамки: зовнішню по краях елемента керування і внутрішню. Властивість BevelOuter визначає вид зовнішньої рамки компонента-панелі.

Таблиця 2.1.10 – Значення властивості BevelOuter компонента-панелі

Значення	*Дія*
bvNone	Зовнішньої рамки немає
bvLowered	Зовнішня рамка втиснена усередину; панель виглядає заглибленою
bvRaised	Зовнішня рамка видавлена назовні; панель виглядає піднятою

property BevelOuter: TPanelBevel;

Доступно: для компонента TPanel.

Властивість BorderWidth. Визначає відстань між зовнішньою і внутрішньою рамками компонента-панелі. Відстань вказується в пікселях. За замовчуванням прийняте значення 0.

property BorderWidth: TBorderWidth;

Доступно: для компонента TPanel.

Властивість Shape. Визначає зовнішній вигляд елемента керування TBevel

Таблиця 2.1.11 – Значення властивості Shape компонента TBevel

Значення	*Вид*
bsBox	Прямокутник
bsFrame	Без рамок
bsTopLine	Лінія вгорі
bsBottomLine	Лінія внизу
bsLeftLine	Лінія ліворуч
bsRightLine	Лінія праворуч

property Shape: TBevelShape;

Доступно: для компонента TBevel.

Малюнок 2.1.6 – Об'єкт Panel із властивістю BevelInner рівним bvLowered (ліворуч) і об'єкт Bevel із властивістю Shape рівним bsBox (праворуч)

Властивість Style. Визначає стиль відображення видимих країв компонента TBevel.

Таблиця 2.1.12 – Значення властивості Style компонента TBevel

Значення	*Дія*
bsLowered	Краї поглиблені
bsRaised	Краї підняті

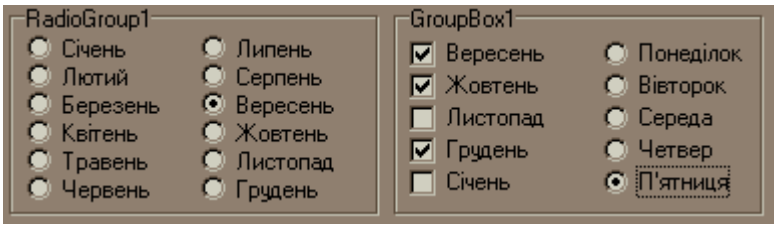
property Style: TBevelStyle;

Доступно: для компонента TBevel.

2.1.4.2. Використання компонентів TGroupBox та TRadioGroup

Компонент TGroupBox використовується для об'єднання прапорців опцій, кнопок із залежною фіксацією й інших елементів керування. За допомогою елемента керування TGroupBox можна створити рамку, що групує, щоб зібрати разом взаємозалежні елементи форми. При цьому компонент TGroupBox буде батьківським компонентом для розміщених усередині нього елементів керування.

Групу кнопок із залежною фіксацією простіше створювати за допомогою компонента TRadioGroup.



Малюнок 2.1.7 – Об'єкт RadioGroup (ліворуч) і об'єкт GroupBox із включеними в нього об'єктами CheckBox і RadioButton (праворуч)

Властивість Columns. Указує число стовпчиків у списку або групі опцій.

property Columns: LongInt;

Доступно: для компонентів TRadioGroup, TListBox.

Властивість Items. Містить покажчик на список рядків, що відображаються в даному компоненті. Тому що властивість Items указує на об'єкт типу TStrings, з ним можна виконувати операції додавання, видалення, вставки і переміщення елементів за допомогою методів Add, Delete, Insert, Exchange і Move об'єкта TStrings.

property Items: TStrings;

Доступно: для компонентів TRadioGroup, TListBox, TComboBox.

Властивість ItemIndex. Значення властивості ItemIndex – це номер обраного елемента в списку (якщо тільки MultiSelect не має значення True). Якщо не обраний жоден елемент, ця властивість має значення –1. Щоб вибрати елемент під час виконання програми, варто привласнити властивості ItemIndex як значення індекс потрібного елемента. При цьому необхідно врахувати, що індекс першого елемента в списку дорівнює 0.

property ItemIndex: Integer;

Доступно: для компонентів TRadioGroup, TListBox, TComboBox.

2.1.5. Корисні поради

✍ Для того щоб компонент TGroupBox мав контекстне меню, що відрізняється від контекстного меню форми, необхідно привласнити властивості PopupMenu компонента TGroupBox ім'я будь-якого об'єкта PopupMenu. Якщо у формі утримується кілька елементів GroupBox, роботу з ними полегшує контекстне меню, зміст якого залежить від того, у якому місці екрана користувач клацає правою кнопкою миші.

✍ Щоб у залежності від стану програми кнопка з'являлася і зникала, досить привласнити властивості Visible цієї кнопки значення True або False. Це справедливо для кожного з компонентів, описаних у даній главі, а також для більшості інших компонентів Delphi.

✍ Двічі клацнувши на значенні властивості Color (клацати треба саме на значенні, розташованому праворуч, а не на імені), ви відкриєте вікно Color Editor (Редагування кольору). Виберіть будь-який підходящий для вас колір. Крім того, за допомогою кнопки Define Custom Colors (Визначення користувальницького кольору) ви можете відкрити інший розділ редактора, що дозволяє сформувати новий колір, використовуючи червону, зелену і синю складові або значення відтінку, насиченості і яскравості.

✍ Задайте як значення властивості Hint компонента TSpeedButton текстовий рядок і встановіть властивість ShowHint рівною True. Тепер, якщо користувач затримає покажчик миші на кнопці, він побачить невелике вікно з підказкою.

✍ Властивість Spacing компонента TBitBtn дозволяє задати відстань між зображенням і текстом, відображуваними на кнопці. Значення за замовчуванням дорівнює 4. Якщо властивість дорівнює — 1, текст розташовується посередине між зображенням і границею кнопки. Задавши значення Spacing рівним 0, ви

вказуєте програмі, що текст повинний розташуватися впритул до зображення. Компонент TSpeedButton, у якому відсутнє властивість Caption, також містить властивість Spacing, за допомогою якого можна керувати положенням гліфа.

✍ Незважаючи на те що компоненти TPanel найчастіше використовуються для створення візуальних ефектів, ви можете створювати для них оброблювачі подій. Наприклад, оброблювач події OnClick дозволяє виконувати деякі дії після щиглика мишею на об'єкті Panel. Ще одна подія — OnResize — часто використовується для вирівнювання об'єктів Panel у робочій області. Коли користувач змінює розміри вікна, оброблювач події OnResize компонента TPanel може коректувати його розташування усередині вікна.

✍ Для групування кнопок із залежною фіксацією, перемикачів і інших елементів керування можна використовувати будь-як контейнер. Наприклад, якщо ви хочете створити дві групи кнопок із залежною фіксацією без рамок, використовуйте як контейнери об'єкти TPanel і забороніть відображення рамки.

2.1.6. Резюме

Система Windows надає три стандартних типи кнопок — звичайні кнопки (або кнопки виклику команд), прапорці опцій і кнопки з залежною фіксацією. Delphi инкапсулює ці стандартні елементи керування в компонентах TButton, TCheckBox і TRadioButton.

Масив Components форми надає доступ до всії об'єктів, що утримуються в ній, і дозволяє виконувати з ними різні дії. Ви можете, наприклад, зробити недоступними всі елементи керування у вікні, крім однієї кнопки.

Найчастіше об'єкт CheckBox діє, як двухпозиционный перемикач, але його можна настроїти і як перемикач із трьома станами, установивши властивість AllowGrayed рівним True. Такий об'єкт може знаходитися в одному з трьох станів: бути включеним, виключен або заблокованим.

Ви можете задавати кольори елементів керування CheckBox і RadioButton, але через обмеження, що накладаються Windows, колір об'єкта Button змінити не можна. Якщо вас не влаштовує колір кнопки, використовуйте компонент TBitBtn.

Компоненти TBitBtn і TSpeedButton можуть містити гліфи, що відбивають, як правило, призначення даного елемента керування. Гліф — це растрове зображення, представлене об'єктом TBitmap. На розміри гліфа не накладаються ніякі обмеження, але звичайно це 16-кольорові картинки розміром 16*16 пікселей. Гліф може містити від одного до чотирьох окремих зображень, що представляють різні стани кнопки.

Компоненти TSpeedButton звичайно використовуються в складі панелей інструментів, але SpeedButton можуть застосовуватися і як незалежні об'єкти.

Компоненти TGroupBox використовуються для групування декількох елементів керування. Звичайно GroupBox містять об'єкти RadioButton, але вони можуть включати й інші елементи. Натискаючи клавішу <Tab>, користувач може переходити від однієї групи елементів до іншої.

Компонент TRadioGroup спрощує створення набору об'єктів RadioButton. Як значення властивості Items задаються етикетки окремих перемикачів.

Об'єкти Panel і Bevel в основному призначені для створення візуальних ефектів, але їх можна використовувати при організації вікна, насиченого кнопками й іншими елементами керування.

Вкладка Samples палітри VCL містить два додаткових елементи керування: SpinButton і SpinEdit. Привласнивши імена своїх глифов властивостям

DownGlyph і UpGlyph цих об'єктів, ви зможете відображати на кнопках замість стрілок інші картинки.

Щоб зв'язати кнопки зі статичним текстом або вікном редагування, замість TSpinButton і TSpinEdit можна використовувати компонент TUpDown.

Література [1].

Тема 2.2. Створення панелей інструментів та робота з строковими компонентами

2.2.1. Компоненти

- ***TToolBar.*** Даний компонент забезпечує альтернативний спосіб створення панелей інструментів у додатку. Компонент TToolBar може включати об'єкти ToolButton.
  Категорія палітри: Win32.
- ***TToolButton.*** Компонент TToolBar може відображати один або кілька об'єктів ToolButton і керувати ними. Об'єкти ToolButton схожі на SpeedButton, однак на об'єктах ToolButton крім картинок, можуть відображатися текстові мітки. Крім того, з об'єктом ToolButton може бути зв'язано кілька зображень. Які дають можливість змінювати зовнішній вигляд кнопки після щиглика на ній.
  Категорія палітри: відсутній.
- ***TEdit.*** Цей компонент дозволяє вирішувати багато задач, що стосуються введення окремого рядка. Об'єкт Edit підтримує операції вирізання, копіювання і вставки.
  Категорія палітри: Standard.
- ***TLabel.*** Це текстові об'єкти, які можна включати у форми й інші контейнери. Вони використовуються для веведення пояснювального тексту поруч з компонентами або відображення іншої інформації на екрані.
  Категорія палітри: Standard.
- ***TMaskEdit.*** Цей компонент можна розглядати як модифіковане поле редагування. Використовуйте компонент TMaskEdit для введення даних, формат яких визначений заздалегідь, наприклад телефонних номерів або адрес Internet, цифри яких повинні займати визначені позиції.
  Категорія палітри: Additional.

2.2.2. Панелі інструментів (компоненти TToolBar)

Delphi містить два компоненти, призначені для створення панелей інструментів і піктограм: TToolBar і TCoolBar. Компонент TToolBar служить контейнером для спеціальних кнопок (TToolButton) і призначений для використання як панель інструментів. Компонент TCoolBar – це контейнер для віконних елементів керування, що утримуються в переміщуваних і змінюючих свої розміри областях (TCoolBand).

Додати до форми компонентів TToolBar можна в такий спосіб:

- Активізувати сторінку Win32 палітри компонентів і розмістити у формі елемент керування TToolBar. У формі з'явиться порожній елемент керування TToolBar з маркерами зміни розмірів.
- Для розміщення на створюваній панелі інструментів кнопки. Необхідно виконати щиглик правою кнопкою миші на компоненті TToolBar і у контекстному меню вибрати команду New Button.

- Щоб розташувати на панелі інструментів роздільник, необхідно в контекстному меню вибрати команду New Separator.

Властивість Style. Установлює зовнішній вигляд і функціональні можливості кнопок TToolButton, що утримуються в компоненті TToolBar.

Таблиця 2.2.1 – Можливі значення властивості Style компонента TToolButton

Значення	***Дія***
tbsButton	Кнопка має вигляд і можливості звичайної кнопки панелі інструментів (TSpeedButton)
tbsCheck	Після того як на кнопці був виконаний щиглик мишею, вона зображується натиснутої до наступного щиглика
tbsDivider	Кнопка зображується у виді вертикальної лінії і не володіє ніякими функціональними можливостями (використовується як роздільник між іншими кнопками)
tbsDropDown	На кнопці зображується спрямована вниз стрілка
tbsSeparator	Кнопка зображується у виді роздільника і не має ніякі можливості

property Style: TToolButtonStyle;

Доступно: для компонента TToolButton.

Властивість Grouped. За допомогою властивості Grouped сусідні кнопки TToolButton, що утримуються в компоненті TToolBar, можуть бути об'єднані в групи. Якщо властивості Style усіх цих кнопок додати значення tbsCheck, будуть визначені взаємно виключаючі можливості вибору, тобто в межах цієї групи може бути натиснута тільки одна кнопка.

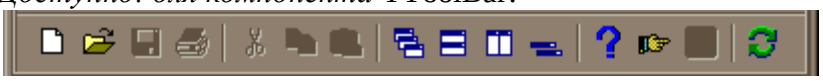
property Grouped: Boolean;

Доступно: для компонента TToolButton.

Властивість Flat. За допомогою властивості Flat можна відобразити панель інструментів у стилі, прийнятому в Windows 95. при цьому кнопки зображуються без рамок і виглядають як набір зображень. Лише коли покажчик миші переміщається на кнопку, з'являється рамка і кнопка зображується об'ємної.

property Flat: Boolean;

Доступно: для компонента TToolBar.



Малюнок 2.2.1 – Об'єкт ToolBar із включеними в нього об'єктами ToolButton і властивістю Flat рівним True

Властивість ShowCaptions. Визначає, чи відображаються на кнопках TToolButton написи. За замовчуванням установлене значення False, тобто написи не відображаються. Текст напису варто вказати у властивості Caption відповідної кнопки.

property ShowCaptions: Boolean;

Доступно: для компонента TToolBar.

2.2.3. Компонент TLabel, поля введення TEdit та TMaskEdit

Об'єкти компонента TLabel – це текстові об'єкти, які можна включати у форми й інші контейнери, наприклад у Panel. Компонент TLabel має безліч властивостей. Ви можете визначати властивість Font, щоб змінювати зовнішній вигляд тексту, задавати перенос тексту за словами, установлюючи властивість WordWrap рівним True, і змінювати колір об'єкта за допомогою властивості Color.

Компоненти TEdit і TMaskEdit використовуються для введення тексту.

Малюнок 2.2.2 – Об'єкти Label і Edit, а також об'єкт MaskEdit з маскою для введення дати

Властивість AutoSelect. Визначає, чи буде текст у полі уведення виділений, коли елемент керування стане активним. Якщо властивість AutoSelect має значення True, то текст виділяється, і навпаки. За замовчуванням установлене значення True.

property AutoSelect: Boolean;

Доступно: для компонентів TEdit, TMaskEdit.

Властивість CharCase. Визначає, як відображається текст, зазначений у властивості Text поля введення.

Таблиця 2.2.2 – Можливі значення властивості CharCase компонентів TEdit і TMaskEdit

Значення	***Дія***
ecLowerCase	Текст відображається малими літерами
ecNormal	Текст відображається рядковими і прописними літерами
ecUpperCase	Текст відображається прописними літерами

Якщо користувач буде використовувати регістр, відмінний від встановленого у властивості CharCase, текст усе рівно буде відображатися в тім регістрі, що зазначений у властивості CharCase.

property CharCase: TCharCase;

Доступно: для компонентів TEdit, TMaskEdit.

Властивість SelText. Містить виділену в елементі керування частина тексту. За допомогою цієї властивості можна зчитувати або змінювати виділений текст, уводячи новий рядок. Якщо компонент не містить виділений текст, то при присвоєнні властивості SelText як значення якого-небудь рядка послідовність символів, що утримується в ній, вставляється в текст, починаючи з поточної позиції курсору.

property SelText: String;

Доступно: для компонентів TEdit, TMaskEdit, TComboBox.

Властивість Text. Містить текст, що знаходиться в полі введення або в полі Memo.

property Text: TCaption;

Доступно: для компонентів TEdit, TMaskEdit, TMemo.

Метод GetSelTextBuf. Копіює виділений у полі введення або полі Memo текст (обсягом BufSize символів) у буфер, на який указує параметр Buffer, і повертає число, що відповідає кількості фактично скопійованих символів.

function GetSelTextBuf (Buffer: PChar; BufSize: Integer): Integer;

Доступний: для компонентів TEdit, TMaskEdit, TMemo.

Властивість MaxLength. Указує, яке максимальне число символів може ввести користувач у поле введення, поле Memo або в область уведення поля зі списком. За замовчуванням прийняте значення 0. Це означає, що на кількість символів, що вводяться, немає обмежень. Будь-яке інше число обмежує кількість символів, що вводяться.

property MaxLength: Integer;

Доступно: для компонентів TEdit, TMaskEdit, TMemo, TComboBox.

Властивість SelLength. Повертає довжину виділеної в елементі керування частини рядка (у кількості символів). Якщо SelLength використовується спільно з властивістю SelStart, можна задати, яка частина тексту буде виділена. Кількість виділених символів можна змінити, привласнивши інше значення властивості SelLength. При зміні значення SelStart змінюється і значення властивості SelLength.

При зміні значення властивості SelLength для поля введення або поля Memo вони повинні бути активними елементами керування в даний момент, інакше ця зміна не зробить ніякої дії.

property SelLength: Integer;

Доступно: для компонентів TEdit, TMaskEdit, TMemo, TComboBox.

Метод SelectAll. Виділяє в елементі керування весь текст. Для виділення частини тексту варто використовувати властивості SelStart і SelLength.

procedure SelectAll;

Доступно: для компонентів TEdit, TMaskEdit, TMemo, TComboBox.

Метод SetSelTextBuf. Заміняє виділений у полі введення або полі Memo текст умістом рядка з завершальним нулем, на який вказує параметр Buffer.

procedure SetSelTextBuf (Buffer: PChar);

Доступний: для компонентів TEdit, TMaskEdit, TMemo, TComboBox.

Властивість HideSelection. Визначає, чи залишається виділеним текст в елементі керування, коли активним стає інший елемент. Якщо ця властивість має значення True, то текст відображається невиділеним, поки даний елемент знову не стане активним. За замовчуванням установлене значення True.

property HideSelection: Boolean;

Доступно: для компонентів TEdit, TMemo.

Властивість Modified. Указує, чи змінював користувач текст у полі введення або в полі Memo. Якщо Modified має значення True, то текст змінювався, і навпаки.

property Modified: Boolean;

Доступно: для компонентів TEdit, TMaskEdit, TMemo.

Властивість ReadOnly. Визначає, чи може користувач змінити текст елемента керування. Якщо ця властивість має значення True, користувач не може редагувати вміст елемента керування, і навпаки. За замовчуванням прийняте значення False.

property ReadOnly: Boolean;

Доступно: для компонентів TEdit, TMaskEdit, TMemo.

Література [1].

Тема 2.3. Події та оброблювачі подій

2.3.1. Які бувають події

Додатки, створені за допомогою Delphi – це додатка для Windows. Однією з властивостей таких додатків є керування по подіях. Це означає, що програма виконується на основі згенерованих повідомлень про події, що обробляються програмним кодом додатка. Такий код необхідно написати для кожної події, на яку повинна реагувати програма. Процедура, призначена для реагування на яку-небудь подію, називається в Delphi *процедурою обробки події* (або *Event Handler*). Delphi генерує процедури обробки для кожної події і дає їм імена відповідно до імен компонентів, для яких ці процедури призначені. Наприклад, у випадку обробки події щиглика мишею (Click) на елементі керування Label1 заголовок такої процедури виглядає таким чином:

procedure TForm1.Label1Click(Sender: TObject);

Цей заголовок процедури автоматично створюється Delphi, коли розроблювач двічі клацає на імені оброблювача події на сторінці **Events** вікна **Object Inspector**. Крім того, у implementation-секції модуля генерується порожня процедура обробки події, що має наступний вигляд:

procedure TForm1.Label1Click(Sender: TObject);
begin

end;

Тепер необхідно лише вписати потрібний програмний код між зарезервованими словами Object Pascal begin і end.

При розгляді подій виділяються дві основні категорії: події, обумовлені діями користувача (далі – *користувальницькі події*), і звичайні події. Події останньої категорії називаються також *програмно-керованими.*

Процедури обробки користувальницьких подій складають головну частину програмного коду додатка. Вони забезпечують інтерактивну взаємодію додатка і користувача. У Delphi для цієї мети застосовуються попередньо визначені оброблювачі подій, наприклад OnClick, що можуть використовуватися практично всіма компонентами.

До найбільш часто використовуваних оброблювачів користувальницьких подій відносяться оброблювачі подій OnClick, OnDblClick, оброблювачі подій миші і клавіатури.

До звичайних подій відносяться події активізації, завершення, події зміни стану окремих компонентів і т.д., що є непрямим результатом дії користувача.

Як випливає з назви, користувальницькі події (user events) виникають у результаті дій користувача. Програми повинна адекватно на них реагувати.

Реакція програми на виконання користувачем щиглика на визначеному компоненті заснована на виклику оброблювача події OnClick. Завдання розроблювача полягає в створенні програмного коду обробки даної події.

Користувальницькі події підрозділяються на:

- події миші;
- операції Drag&Drop;
- події клавіатури.

2.3.2. Події, обумовлені діями користувача

2.3.2.1. Події миші

Без миші не можна уявити собі додаток для Windows. В даний час кожна миша має щонайменше дві кнопки. Це можна вважати стандартом. Таким чином, при розробці додатка можна виходити з того, що користувач у будь-якій ситуації може застосовувати праву і ліву кнопки миші. Користувачі програм Windows звикли використовувати в основному ліву кнопку. Ліва кнопка миші в Delphi використовується як основна. Коли говориться про натискання або відпускання незазначеної кнопки миші, завжди мається на увазі ліва кнопка.

Коли користувач переміщає мишу, Windows реєструє ці дії і “відправляє” додаткові необхідні повідомлення. Розрізняються чотири види дій з мишею, на які повинний реагувати додаток.

- Натискання кнопки миші (MouseDown).
- Відпускання кнопки миші (MouseUp).
- Переміщення миші (MouseMove).
- Щиглик (Click).

Оброблювач події OnMouseDown. Викликається, коли користувач натискає кнопку миші за умови, що покажчик миші знаходиться на елементі керування. Об'єкт, на якому знаходиться покажчик, одержує повідомлення про цю подію, і виконується процедура обробки події для цього об'єкта, якщо така процедура була визначена.

У процедурі обробки події варто визначити, що повинно відбутися як реакція на натискання кнопки миші.

property OnMouseDown: TMouseEvent;

Доступний: для всіх елементів керування.

Оброблювач події OnMouseUp. Викликається, коли користувач відпускає натиснуту кнопку миші. Звичайне повідомлення про дану подію передається об'єкту, на якому знаходився покажчик миші в момент натискання кнопки.

У даній процедурі обробки події варто установити, що повинно відбутися як реакція на відпускання миші. При цьому треба врахувати, що ця процедура обробки події викликається тим же компонентом, на якому знаходився покажчик миші в момент, коли кнопка була натиснута.

property OnMouseUp: TMouseEvent;

Доступний: для всіх елементів керування.

Оброблювач події OnMouseMove. Викликається періодично при переміщенні користувачем покажчика миші.

property OnMouseMove: TMouseMoveEvent;

Доступний: для всіх елементів керування.

Оброблювач події OnClick. Викликається, якщо користувач:

- натискає і відпускає кнопку миші (звичайно це ліва кнопка) у той момент, коли покажчик миші знаходиться на компоненті;
- натискає клавішу <Пробіл>, коли активна яка-небудь кнопка або опція;
- натискає <Enter>, коли активна форма містить попередньо обрану кнопку за замовчуванням (обумовлену за допомогою властивості Default);
- натискає <Esc>, коли активна форма містить кнопку **Cancel** (визначається за допомогою властивості Cancel) і т.д.

property OnClick: TNotifyEvent;

Доступний: для всіх елементів керування.

Оброблювач події OnDblClick. Викликається, коли користувач виконує подвійний щиглик на компоненті.

property OnDblClick: TNotifyEvent;

Доступний: для всіх елементів керування.

2.3.2.2. Події клавіатури

Не потрібно реагувати на кожне натискання клавіші і реєструвати його. Windows самостійно обробляє велику частину натискань клавіш, наприклад клавіш, що визначені для команд меню в сполученні з клавішею <Alt>. Для різних системних функцій, що можуть бути викликані за допомогою клавіатури, Windows має у своєму розпорядженні стандартні алгоритми обробки даних подій.

Розроблювач повинний самостійно передбачати перехоплення натискань тих клавіш, для яких він установив у додатку визначені функції. Події клавіатури для цих клавіш повинні бути оброблені в програмному коді додатка.

Оброблювач події OnKeyDown. Викликається для активного елемента керування, коли користувач натискає будь-яку клавішу.

Процедура обробки події OnKeyDown може реагувати на натискання всіх клавіш (включаючи функціональні і сполучення з клавішами <Shift>, <Alt>, <Ctrl>) і кнопок миші.

property OnKeyDown: TKeyEvent;

Доступний: для всіх елементів керування.

Оброблювач події OnKeyUp. Викликається, коли користувач відпускає натиснуту клавішу.

Процедура обробки події OnKeyUp може реагувати на натискання всіх клавіш (включаючи функціональні і сполучення з клавішами <Shift>, <Alt>, <Ctrl>) і кнопок миші.

property OnKeyUp: TKeyEvent;

Доступний: для всіх елементів керування.

Оброблювач події OnKeyPress. Викликається, коли користувач натискає клавішу з “літерним” символом з набору ASCII.

property OnKeyPress: TKeyPressEvent;

Доступний: для всіх елементів керування.

2.3.3. Програмно-керовані події

Пряма взаємодія між користувачем і програмою відбувається за допомогою описаних вище оброблювачів подій. Крім цього, існують і інші оброблювачі подій. Замість програмно-керованих їх можна було б назвати “компонентно-керованими”

Оброблювач події OnChange. Викликається, коли компонент або об'єкт змінюється.

property OnChange: TNotifyEvent;

Доступний: для компонентів TEdit, TMaskEdit, TMemo, TComboBox, TCoolBar, TScrollBar.

Оброблювач події OnEnter. Викликається, коли віконний елемент керування стає активним для введення. Однак оброблювач події OnEnter не викликається при переключенні між вікнами або між додатками. Оброблювач події OnEnter викликається після того, як значення властивості ActiveControl обновиться. Ця властивість указує, який елемент керування активний у даний момент часу або який елемент керування буде активним, коли форма стане активної.

property OnEnter: TNotifyEvent;

Доступний: для усіх віконних елементів керування.

Оброблювач події OnActivate. Подія OnActivate для додатка (Application) викликається, якщо додаток стає активним (тобто коли він запускається або коли ви переключаєтеся на роботу з цим додатком після роботи з іншим).

property OnActivate: TNotifyEvent;

Доступний: для компонентів TApplication, TForm.

Оброблювач події OnDectivate. Подія OnActivate для додатка (Application) викликається, коли користувач переключається з даного додатка на інший.

property OnDectivate: TNotifyEvent;

Доступний: для компонентів TApplication, TForm.

Оброблювач події OnException. Викликається, коли в додатку виникає виняткова ситуація. Якщо в додатку не мається програмного коду обробки виняткових ситуацій, метод HandleException викликає процедуру обробки події OnExcepnion і метод ShowException. Останній відображає вікно повідомлення, яке вказує, що відбулася помилка. У процедурі OnExcepnion розроблювач може самостійно вказати, що повинно відбутися у випадку виникнення виняткової ситуації.

property OnExcepnion: TExcepnionEvent;

Доступний: для компонента TApplication.

Оброблювач події OnHelp. Викликається, коли додаток одержує запит на виклик довідки.

Методи додатка HelpContext і HelpJump автоматично викликають OnHelp.

property OnHelp: THelpEvent;

Доступний: для компонента TApplication, TForm.

Оброблювач події OnHint. Якщо у властивості Hint компонента зазначений текст підказки, оброблювач події OnHint викликається, коли користувач поміщає покажчик миші на цей компонент. Якщо, крім цього, властивість ShowHint компонента має значення True (за замовчуванням прийняте значення False), то підказка з'являється автоматично.

property OnHint: TNotifyEvent;

Доступний: для компонента TApplication.

Оброблювач події OnActivate. Викликається, коли форма стає активною.

property OnActivate: TNotifyEvent;

Доступний: для компонента TApplication, TForm.

Оброблювач події OnShow. Викликається безпосередньо перед тим, як форма стає видимою.

property OnShow: TNotifyEvent;

Доступний: для компонента TForm.

Оброблювач події OnHide. Викликається безпосередньо перед тим, як форма стає невидимою.

property OnHide: TNotifyEvent;

Доступний: для компонента TForm.

Оброблювач події OnCreate. Викликається при створенні форми.

Коли додаток запускається, Delphi створює форми, викликаючи метод Create для кожної з них. У момент часу, коли створюється форма і її властивість Visible привласнюється значення True, послідовно викликаються наступні оброблювачі подій:

- OnCreate;
- OnShow;
- OnActivate;
- OnPaint.

У процедурі обробки події OnCreate варто визначити ті дії, що повинні бути виконаний, перш ніж користувач почне працювати з формою.

Не слід включати в текст OnCreate явних посилань на саму форму. Наприклад, якщо ім'я форми Form1 і вона містить поле введення Edit1, не можна послатися на нього за допомогою конструкції Form1.Edit1, оскільки в момент виконання процедури OnCreate змінна Form1 ще не визначена. Замість цього варто використовувати не цілком визначене ім'я: Edit1.

property OnCreate: TNotifyEvent;

Доступний: для компонента TForm.

Оброблювач події OnClose. Викликається безпосередньо перед закриттям форми.

Форма закривається за допомогою методу Close або коли користувач вибирає команду **Close** у системному меню форми.

Процедура, зв'язана з оброблювачем події OnClose, може перевірити, чи мають усі поля форми, призначені для введення даних, припустимі значення, перш ніж дозволити закриття форми.

property OnClose: TCloseEvent;

Доступний: для компонента TForm.

Оброблювач події OnCloseQuery. Викликається, коли починається спроба закрити форму. Це відбувається при виклику методу Close або коли користувач вибирає команду **Close** у системному меню форми.

Процедура обробки події OnCloseQuery має параметр CanClose, що визначає, чи дозволене закриття форми. Ця змінна має логічний тип; значення за замовчуванням – True.

Можна використовувати процедуру OnCloseQuery, щоб запитувати користувача, чи дійсно форма повинна бути закрита. На екрані може бути також відображене повідомлення, яке нагадує користувачеві, що варто зберегти дані, перш ніж форма буде закрита.

property OnCloseQuery: TCloseQueryEvent;

Доступний: для компонента TForm.

Література [2].

Тема 2.4. Конструювання меню

2.4.1. Компоненти

Слово *меню* походить від латинського *minute*, що означає *дуже маленький* або *деталізований*. Любий користувач Windows знає, що меню являє собою детальний перелік команд, таких як Open або Exit. У *рядках меню* (menu bar) більшості додатків перераховані імена декількох меню що розкриваються. У додатку можна викликати на екран і *спливаюче меню* (floating pop-up menu), клацнувши правою кнопкою миші.

Працюючи в середовищі Delphi, не так вже і складно створити зручне у використанні меню. Для цього Delphi надає у ваше розпорядження три компоненти меню і кілька методів роботи з ними, про які і піде мова далі.

Нижче перераховані три різновиди компонентів для конструювання меню, що маються в Delphi.

- ***TMainMenu***. Цей компонент варто використовувати для створення рядка головного меню додатка, що завжди виводиться безпосередньо під рядком заголовка вікна. Для створення динамічного меню (тобто меню, що змінюється в ході виконання програми) можна вставити в екранну форму кілька об'єктів компонента MainMenu і зв'язати з кожним з них індивідуальний набір команд, про що буде докладно розказане в одному з наступних розділів.
  Категорія палітри: Standard.
- ***TPopupMenu***. Цей компонент варто використовувати для створення спливаючого меню, що з'являється на екрані після щиглика правою кнопкою миші на полі клієнтської області вікна додатка. Можна виводити спливаюче меню й у відповідь на інші дії користувача, причому меню може з'являтися в будь-якому місці клієнтської області вікна додатка.
  Категорія палітри: Standard.
- ***TMenuItem***. Любий елемент спливаючого меню чи меню що розкривається є об'єктом класу TMenuItem. Хоча компонент MenuItem і відсутній на палітрі компонентів, його досить просто створити в екранній формі за допомогою такого спеціалізованого засобу Delphi, як Menu Designer. Для створення пунктів меню також можна використовувати оператори програми або *сценарії ресурсів* (resource script).
  Категорія палітри: відсутній.

2.4.2. Меню що розкриваються

Строго кажучи, меню що розкривається є вікном, яке з'являється на екрані при виборі одного з пунктів у рядку меню вікна додатка. Спливаючие меню — це різновид меню що розкривається. На відміну від меню що розкривається, спливаючому меню може з'являтися в будь-якому місці вікна (це єдине розходження між ними). Прийоми формування обох типів меню будуть описані в цьому розділі. Для створення меню що розкривається потрібно використовувати компонент MainMenu, а для створення спливаючих меню — компонентів PopupMenu.

2.4.2.1. Головне меню

У більшості додатків Windows мається *головне меню*, що створюється компонентом TMainMenu. Після подвійного щиглика на об'єкті цього компонента в екранній формі на екран виводиться діалогове вікно Menu Designer Delphi, що дозволяє включити в меню команди, специфічні для створюваної програми.

1) Виберіть команду Project⇨Options, а потім відкрийте вкладку Forms. Виберіть форму Main як екранну форму головного вікна додатка. Така установка пропонується Delphi за замовчуванням для додатка з єдиним вікном.
2) Вставте об'єкт компонента TMainMenu в екранну форму.
3) Задайте для властивості Menu екранної форми значення, що відповідає імені об'єкта компонента MainMenu.

Хоча, за невеликими виключеннями, у додатку може бути тільки одне головне меню, будь-яка екранна форма, навіть діалогове вікно, може мати рядок меню. Для установки цього рядка досить включити в екранну форму об'єкт компонента TMainMenu.

Виключення ж наступні.

- Властивість BorderStyle екранної форми не дорівнює bsDialog.

- Властивість FormStyle екранної форми не дорівнює fsMDlChild (дочірні вікна MDI-додатка можуть мати власні об'єкти компонента MainMenu, але ці меню зливаються з рядком меню головного вікна додатка).

2.4.2.2. Спливаючі меню

Для створення спливаючого меню, що, як правило, з'являється на екрані після щиглика правою кнопкою миші, виконаєте наступні операції.

1) Вставте об'єкт компонента TPopupMenu в екранну форму. За замовчуванням Delphi привласнить новому об'єктові найменування PopupMenu1.
2) Двічі клацніть на об'єкті PopupMenu1 і за допомогою Menu Designer введіть у меню список команд.
3) Привласніть значення PopupMenu1 властивості PopupMenu екранної форми.

2.4.2.3. Пункти меню

Кожен пункт меню являє собою об'єкт класу TMenuItem. Delphi створює ці об'єкти автоматично в міру того, як ви додаєте в меню нові пункти за допомогою Menu Designer. Об'єкт MenuItem можна створити і програмно. До нього (об'єкта) також приходиться часто звертатися в програмі. Наприклад, для того щоб сформувати маркер перед написом пункту меню, потрібно програмно привласнити значення True властивості Checked відповідного об'єкта MenuItem. Можна виконувати різні операції з пунктами меню (об'єктами MenuItem) у рядку меню (об'єкті MainMenu) або в спливаючому меню (об'єкті PopupMenu), звертаючись до відповідних елементів масиву Items у цих об'єктах.

Суворе дотримання прийнятих угод про іменування об'єктів у випадку роботи з об'єктами меню дозволить створити зрозумілу, а отже, і просту у використанні програму. За замовчуванням Delphi автоматично привласнює об'єктам MenuItem такі імена, як File1, Open1 і Save2, з якими можна погодитися тільки в експериментальних або демонстраційних програмах, життєвий цикл яких нетривалий. Варто завжди заміняти ці імена, керуючись наступними правилами.

- Для пунктів рядка меню додавати слово Menu до найменування пункту відповідного об'єкта MenuItem, наприклад MenuFile для пункту File і MenuEdit — для пункту Edit.
- Для пунктів меню що розкривається використовувати найменування відповідного пункту рядка меню як префікс. Наприклад, для об'єктів MenuItem у меню File необхідно вибирати найменування FileOpen, FileSave і FileSaveAs. Об'єкти в меню Edit одержують найменування типу EditCut, EditCopy і EditPaste. Деякі розроблювачі програм в середовищі Delphi приєднують до цих імен ще і слово Item — виходить щось типу FileOpenItem і EditCutItem. Можете так діяти і ви, особливо якщо плануєте використовувати мастер для створення нових оболонок додатків.

Такі об'єкти, як MenuFile (що представляє меню File) і FileOpen (що представляє пункт (команду) Open у меню File), є об'єктами того самого класу TMenuItem, як і всі інші об'єкти пунктів меню (до них відноситься навіть такий об'єкт, як роздільник, що відокремлює у великому меню одну категорію команд від іншої).

Властивості Caption об'єкта MenuItem в якості значення необхідно привласнити текст, що буде відображатися у відповідному пункті меню працюючої програми. Найпростіший спосіб розробки нового меню — спочатку увести всі значення Caption, а потім відповідно змінити пропоновані за замовчуванням значення властивостей Name для об'єктів MenuItem.

Значення властивостей об'єктів MenuItem можна змінювати й у процесі виконання програми. Таким чином, у додатку буде створене динамічне меню, команди в якому будуть з'являтися або зникати в залежності від ситуації. Щоб довідатися, як це робиться, виконаєте наступне.

- Вставте об'єкт компонента MainMenu в екранну форму і відкрийте Menu Designer, двічі клацнувши мишею на вставленому об'єкті.
- Сформуйте меню File, а в ньому — команду Exit (уведіть &File і E&xit, щоб призначити F і X як клавіші швидкого виклику цих пунктів меню).
- Виберіть по черзі ці пункти або у вікні Menu Designer, або у вікні проектування форми. Змініть ім'я об'єкта File1 на MenuFile, а Exit1 — на FileExit.

Тепер у вас є об'єкт компонента MainMenu, що "володіє" двома об'єктами MenuItem —MenuFile і FileExit. У програму можна включити оператори, що запрограмували б найрізноманітніші дії з цими пунктами меню. Наприклад, додайте в екранну форму кнопку і, двічі клацнувши на ній, сформуйте заготовку процедури обробки події за замовчуванням — щиглика на кнопці мишею. Потім у нову процедуру між рядків ключових слів begin і end додайте наступний оператор:

with FileExit **do** Visible := not Visible;

Уставте також оператор Close; у процедуру обробки команди File⇨Exit. Запустіть програму на виконання. У працюючій програмі після кожного щиглика на кнопці пункт меню Exit буде то зникати, то з'являтися. Властивість Visible дає програмістові простий і зручний спосіб динамічного керування набором команд у меню — у залежності від значення цієї властивості відповідний пункт меню може з'являтися або зникати.

Інший варіант динамічного керування меню — використання властивості Enabled, що може також приймати значення True або False, дозволяючи або блокуючи відповідний пункт. Наприклад, зміните приведений вище фрагмент програми в такий спосіб:

with FileExit **do** Enabled := not Enabled;

2.4.2.4. Імітування вибору команди

Для імітації вибору команди потрібно викликати в програмі метод Click відповідного об'єкта MenuItem. Це може знадобитися, наприклад, якщо ви хочете, щоб у програмі була виконана та ж операція, що і по команді меню. Якщо таким об'єктом є OptionsProject, у програмі можна викликати відповідну процедуру за допомогою наступного оператора:

OptionsProject.Click; *{Імітація вибору команди Project з меню Options.}*

У принципі, щиглик мишею на пункті рядка меню не повинний мати ніяких наслідків, оскільки з цим пунктом не зв'язана жодна команда. Однак і в такому випадку викликається процедура обробки події OnClick відповідного об'єкта MenuItem, у якій можна запрограмувати виконання яких-небудь дій, наприклад зміна найменувань команд у меню або вставку в них маркерів.

2.4.2.5. Властивість Items

Звернутися до об'єкта MenuItem можна, скориставшись властивістю Items об'єктів компонентів TMainMenu, TPopupMenu і TMenuItem. Items — це або об'єкт класу TMenuItem, методи якого можна викликати, використовуючи крапкову нотацію (наприклад, Items.Insert), або масив, до якого можна звернутися, наприклад, у такий спосіб: MyMenu.Items[N]. В об'єкті MainMenu властивість Items містить пункти рядка меню. Допустимо, у деякій програмі мається рядок меню з пунктами File, Edit і Help, кожному з яких відповідає своє меню що розкривається. Тоді наступний фрагмент програми змінить найменування першого пункту меню на Presto-Chango, заблокує пункт Edit і сховає (зробить невидимим) пункт Help:

```
with MainMenu1 do
begin
  Items[0].Caption := 'Presto-Chango';
  Items[1].Enabled := False;
  Items[2].Visible := False;
end;
```

У свою чергу, кожен об'єкт MenuItem також має властивість Items, яку можна використовувати для доступу до відповідних команд меню. Наприклад, щоб заблокувати

команду File⇨Exit, думаючи, що це єдина команда в даному раскрывающет меню, можна використовувати наступний оператор:

FileMenu.Items[0].Enabled := False;

Той же результат дасть і інший оператор, у якому об'єкт визначений явно:

FileExit.Enabled := False;

Як правило, якщо операція задається для окремого об'єкта MenuItem, простіше явно використовувати його ім'я. Якщо ж потрібно виконати однакові операції з декількома об'єктами, краще використовувати звертання до них через властивість Items. У будь-якому випадку звертання відбувається до того самого об'єкта. Наприклад, оператор блокує в циклі всі пункти об'єкта меню FileMenu:

```
For I := 0 to MenuFile.Count - 1 do
  MenuFile.Items[I].Enabled := False;
```

Зверніть увагу на те, що, оскільки в цьому операторі двічі використовується ім'я об'єкта MenuFile, можна записати його простіше, скориставшись конструкцією with-do. Хоча рядків начебто б стало більше, зміст оператора став більш ясним:

```
with MenuFile do
  for I := 0 to Count - 1 do
    Items[I].Enabled := False;
```

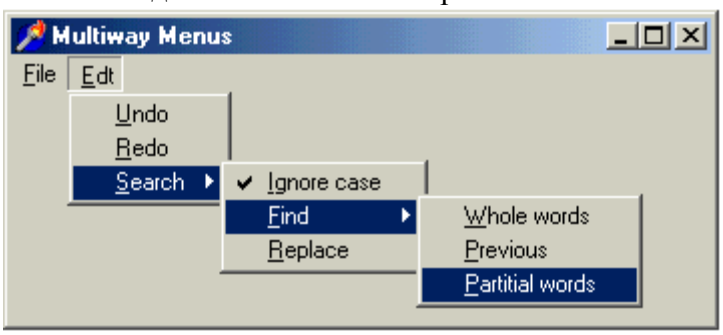
Оператор with повідомляє Delphi, що потрібно використовувати властивості Count і Items об'єкта MenuFile. Таким чином, відпадає необхідність у використанні крапкової нотації (кваліфікування) стосовно даних властивостей. Це, по-перше, скорочує обсяг тексту, що вводиться з клавіатури, (а виходить, і імовірність появи помилок) і, по-друге, спрощує роботу компілятора і допомагає йому сформувати більш ефективний машинний код. Тому рекомендується скрізь, де це можливо, використовувати язикову конструкцію with-do.

Значення властивості Count об'єкта MenuFile є не що інше, як кількість елементів у масиві Items — кількість "підлеглих" об'єктів MenuItem. Оскільки індекс початкового елемента масиву - 0, останній елемент масиву буде мати індекс Count - 1. Не забувайте про це, коли використовуєте звертання до елементів масиву Items у циклі.

2.4.2.6. Каскадне меню

Любий пункт меню що розкривається може, у свою чергу, містити підменю. Як показано на мал. 2.4.1, цю "вкладеність" можна розвивати, скільки завгодно. Але це робити не рекомендується, тому що при цьому ускладнюється шлях до вибору необхідної команди.

У Windows 95 вибір у такому меню трохи полегшується, оскільки не потрібно клацати мишею на кожнім рівні. Але навіть у цьому випадку створювати більш двох рівнів вкладеності не слід без вагомих на те причин.



Малюнок 2.4.1 – . Каскадне меню дозволяє реалізувати в програмі досить складну структуру команд.

У наступній покроковій інструкції описаний процес створення підменю за допомогою Menu Designer.

1) Створіть пункт меню, наприклад Search (див. мал. 2.4.1).

2) Виділите цей пункт, натисніть <Ctrl+→>, і буде сформовано підменю, яке можна редагувати за допомогою Menu Designer так само, як будь-яке меню що розкривається.
3) Повторіть ці операції для кожного наступного каскаду меню.

Каскадні меню не вимагають якихось спеціальних прийомів програмування. Процедури обробки команд каскадного меню створюються по такій же методиці, як і процедури звичайних однорівневих меню. Тому кожен пункт каскадного меню (об'єкт MenuItem) повинний мати унікальне в межах модуля ім'я — значення властивості Name.

2.4.2.7 Клавіші швидкого виклику пунктові меню

Асоціювати комбінації клавіш з пунктами (командами) меню можна подвійно. Виділивши знаком “амперсанд” (&) деяку букву в імені пункту (поставивши символ & перед необхідною буквою в значенні властивості Caption відповідного об'єкта), можна вказати *клавішу швидкого виклику* (“гарячу” клавішу), що разом із клавішею <Alt> дублює відповідну команду уже відкритого меню. Наприклад, якщо привласнити властивості Caption об'єкта MenuItem значення &Configure, користувач зможе відкрити меню Configure, натиснувши <Alt+C>.

Другий тип комбінацій клавіш для звертання до команди меню можна назвати *акселератором* (accelerator). Як правило, це комбінація якої-небудь алфавітно-цифрової клавіші і спеціальної (<Ctrl>, <Shift>), функціональної або клавіші керування курсором. Назву — *акселератори* -ці клавіатурні комбінації одержали тому, що з їх допомогою можна викликати команду меню, *не відкриваючи його*.

Не потрібно *явно* вказувати комбінацію акселератора в тексті пункту меню, тобто в значенні його властивості Caption. Delphi зробить це самостійно, як тільки ви визначите відповідну комбінацію як значення властивості Shortcut об'єкта.

Значення комбінацій-акселераторів можна або ввести у виді тексту, або вибрати зі списку властивості ShortCut, що розкривається. Щоб не шукати необхідну комбінацію клавіш можна використовувати перший спосіб, оскільки пошук у довгому списку не завжди зручний (значно швидше набрати на клавіатурі Ctrl+S, чим шукати такий же елемент у списку пропонованих значень властивості ShortCut). До речі, у списку перераховані не всі припустимі варіанти для значення ShortCut. Якщо як акселератор ви хочете використовувати клавішу <Enter>, можете сміло вводити в поле значення властивості ShortCut текст ENTER.

У табл. 2.4.1 перераховані деякі стандартні акселератори для пунктів меню, широко застосовувані у додатках Windows 9x. Змінювати звичні комбінації не рекомендується, оскільки користувачі, що вже звикли до них, навряд чи відмовляться від своїх звичок заради нової програми. Але деякі стандартні призначення можна при необхідності змінити. Зокрема, стандарт рекомендує використовувати клавішу <Ins> як акселератор команди переключення режиму Insert/Overtype (Вставка/Заміна) у меню Edit. Цю клавішу можна використовувати як акселератор команди вставки поля. У цьому Delphi надає розроблювачеві повну волю.

Таблиця 2.4.1 – Стандартні комбінації акселераторів у Windows 9x

Команда	*Меню*	*Акселератор*
Cascade (Каскад)	Window	<Shift+F5>
Collapse all (Згорнути усі)	Tree	<Ctrl+*> (на цифровій клавіатурі)
Collapse item (Згорнути елемент)	Tree	<-> (на цифровій клавіатурі)
Сміттю (Копіювати)	Edit	<Ctrl+C>
Cut (Вирізувати)	Edit	<Ctrl+X>
Delete (Видалити)	Edit	<Del>
Exit (Вихід)	File	<Alt+X>
Expand all (Розгорнути усі)	Tree	<*> (на цифровій клавіатурі)
Expand item (Розгорнути елемент)	Tree	<+> (на цифровій клавіатурі)

Команда	*Меню*	*Акселератор*
Expand/Collapse (Розгорнути/Згорнути)	Tree	<Enter>
Find (Знайти)	Edit	<Ctrl+F>
Find next (Знайти наступний)	Edit	<F3>
Insert/Overtype (Вставка/Заміна)	Edit	<Ins>
Help (Допомога)	Help	F1
New (Створити)	File	<Ctrl+N>
Open (Відкрити)	File	<Ctrl+0>
Paste (Вставити)	Edit	<Ctrl+V>
Print (Друк)	File	<Ctrl+P>
Replace (Замінити)	Edit	<Ctrl+H>
Save (Зберегти)	File	<Ctrl+S>
Select all (Виділити усі)	Edit	<Ctrl+A>
Tile (Розташувати усі)	Window	<Shift+F4>
Undo (Скасувати)	Edit	<Ctrl+Z>

2.4.3. Спливаючі меню

У принципі, спливаючі меню і меню що розкриваються відрізняються тільки положенням на екрані. Об'єкт компонента PopupMenu програмується в додатку так само, як і об'єкт компонента MainMenu, але з'являється він не поруч з рядком меню вікна, а в будь-якому місці екрана (точніше, у клієнтській області вікна додатка).

Для створення спливаючого меню потрібно вставити в екранну форму об'єкт компонента TPopupMenu, двічі клацнути на ньому і ввести в об'єкт необхідні команди за допомогою Menu Designer, як ми це робили для об'єкта компонента TMainMenu. Тепер ім'я об'єкта PopupMenu необхідно привласнити властивості PopupMenu екранної форми.

Дуже часто, але не завжди, команди спливаючого меню використовують ті ж процедури обробки, що і відповідні команди меню що розкривається. При бажанні можна створити й окремі процедури обробки команд спливаючого меню.

2.4.3.1. Права кнопка миші

Для того щоб запрограмувати появу на екран спливаючого меню після щиглика правою кнопкою миші, потрібно привласнити властивості PopupMenu екранної форми ім'я об'єкта компонента TPopupMenu. Тепер відповідне меню буде під час виконання програми з'являтися в тім місці екрана, де користувач клацне правою кнопкою миші.

Таке присвоювання можна виконати і програмно під час виконання програми. Таким чином, з'являється можливість у залежності від конкретної ситуації вибрати в програмі, який з декількох наявних об'єктів PopupMenu зв'язати з екранною формою. У будь-якому підходящому місці програми, а таким найчастіше є процедура обробки щиглика на якій-небудь командній кнопці або процедура обробки якої-небудь команди основного меню, використовуйте наступний оператор, що привласнює властивості PopupMenu екранної форми ім'я підходящого об'єкта компонента TPopupMenu:

PopupMenu := PopupMenu2; *{Присвоєння імені об'єкта властивості PopupMenu.}*

2.4.3.2. Інші методи виклику спливаючого меню

Вивести на екран спливаюче меню можна також, викликавши метод Popup об'єкта PopupMenu у відповідь на яку-небудь подію, на відміну від щиглика правою кнопкою миші. Під час виклику методу Popup як параметри повинні бути передані координати x и y:

PopupMenu1.Popup(100, 100);

Параметри x и y представляють координати щодо екрана Windows з початком відліку в лівому верхньому куті.

Якщо ви хочете вивести спливаюче меню в тім же місці, де був виконаний щиглик мишею, прийдеться конвертувати поточні координати миші перш, ніж передати їх як аргументи в метод Popup. Справа в тім, що координати миші зчитуються щодо границь активного вікна, а в Popup потрібно передати координати щодо границь екрана. Перетворення здійснюється за допомогою функції ClientToScreen. Для початку потрібно оголосити перемінну типу TPoint:

```
var
  Pt: TPoint;
```

Потім у процедуру обробки події MouseDown екранної форми включите наступні операторы:

```
Pt.X := X;
Pt.Y := Y;
Pt := ClientToScreen(Pt);
PopupMenu1.Popup(Pt.X, Pt.Y);
```

Два перших оператори привласнюють поточні координати покажчика миші, що передаються в цю процедуру при виклику, членам змінної типу TPoint. Третій оператор викликає метод ClientToScreen класу TControl, що є одним з далеких предків класу TForm. Цей метод перетворить значення членів об'єкта Pt із системи координат клієнтської області вікна в систему координат екрана. Останній оператор передає перетворені координати в метод Popup об'єкта компонента TPopupMenu.

2.4.4. Динамічне меню

У Delphi можна використовувати кілька методів створення меню, що динамічно змінюються в процесі виконання програми:

- цілком заміняти рядок меню у вікні додатка;
- вставляти і видаляти меню що розкривається;
- додавати і видаляти окремі команди;
- зливати об'єкти компонентів меню.

У наступних підрозділах про ці методи буде розказано більш докладно.

2.4.4.1. Заміна рядків меню

Найпростіший спосіб змінити рядок меню у вікні додатка — привласнити нове значення (новий об'єкт MainMenu) властивості Menu екранної форми. Для цього спочатку вставте в екранну форму два або більш об'єкти компонента TMainMenu і включіть в них необхідні пункти за допомогою Menu Designer. Потім у підходящому місці програми, наприклад у процедурі обробки щиглика на якій-небудь кнопці екранної форми, вставте оператор наступного виду:

```
Menu := MainMenu2;
```

Коли програма буде запущена на виконання, після щиглика на відповідній кнопці рядок меню зміниться відповідно до вмісту об'єкта MainMenu2.

2.4.4.2 Вставка та видалення меню що розкриваються

У програмі можна злити об'єкти меню (даний спосіб буде докладно описаний у розділі “Злиття і поділ об'єктів MainMenu”, приведеному нижче в цій главі), але є більш простий метод — уставити меню що розкривається в рядок меню вікна додатка або цілком видалити його. Щоб випробувати цю технологію на практиці, вставте один об'єкт компонента MainMenu в екранну форму, двічі клацніть на ньому і за допомогою Menu Designer включіть в нього меню що розкривається з різними наборами команд.

Потім сформуйте процедуру обробки події OnCreate для екранної форми. Вставте в цю процедуру оператори, що дозволяють зробити деякі меню невидимими:

```
MenuOption.Visible := False;
MenuWindow.Visible := False;
```

І нарешті, у програмі обробки щиглика на якій-небудь кнопці екранної форми або процедурі обробки команди в одному з “видимих” меню що розкривається встановіть значення True для властивості Visible раніше схованого меню. У результаті у відповідь на цю подію при виконанні програми на екрані з'явиться меню. Описана методика дозволяє швидко запрограмувати команду Advanced Menu, що заміняє скорочений варіант меню додатка повним варіантом. Але врахуйте, що невидимі пункти меню в програмі присутні, до їхніх процедур обробки команд можна звертатися і програма викликає ці процедури у відповідь на натискання відповідних клавіш-акселераторів.

2.4.4.3. Заміна пунктів меню

Для того щоб замінити видимий текст пункту меню, потрібно змінити значення властивості Caption відповідного об'єкта MenuItem. Наприклад, можна запрограмувати виведення у пункті меню команди Undo інформації про те, яка саме операція буде скасована:

```
EditUndo.Caption := '&Undo deletion';
```

2.4.4.4. Додавання вставка та видалення пунктів меню

Можна додати або вставити команди в будь-яке меню — об'єкт класу TMenuItem або видалити їх. Наприклад, можна додати пункти що відповідають іменам файлів у меню File — об'єкт MenuFile. Цю же методику можна використовувати і для модифікації меню в ході виконання програми.

Перший етап додавання або вставки пункту меню — створення відповідного об'єкта класу TMenuItem. Спочатку оголосіть відповідну змінну в підходящій процедурі обробки події:

```
var
  MI: TMenuItem;
begin
  …
end;
```

Потім у тілі процедури сформуйте новий об'єкт класу TMenuItem, викликавши метод Create даного класу, і передайте йому як параметр ім'я об'єкта — власника цього пункту, тобто ім'я об'єкта меню, у яке планується включити новий пункт:

```
MI := TMenuItem.Create(FileMenu);
```

Тепер у програмі сформований новий об'єкт пункту меню, до якого можна звертатися, посилаючись на ім'я перемінної MI. Привласніть значення властивостям цього об'єкта Caption, Visible і Shortcut, а також іншим у міру потреби:

```
MI.Caption := '&New command1;
MI.Visible := True;
MI.Shortcut := Shortcut(vk_Fl);
```

Коли закінчите ініціалізацію нового об'єкта, додайте його в існуюче меню за допомогою методу Add:

```
MenuFile.Add(MI); {Додати пункт у кінець меню File.}
```

З цього моменту об'єкт MenuFile стане “власником” об'єкта MI і сам видалить його при необхідності. Вам ні в якому разі не можна самостійно видаляти об'єкт MI і звільняти виділену для нього при виконанні Create пам'ять.

Тепер змінну MI можна використовувати для формування іншого пункту меню, але для кожного нового пункту обов'язково прийдеться викликати метод Create. Наприклад, не можна просто привласнити нове значення властивості Caption об'єкта MI і після цього знову додати його в меню. MI у дійсності є покажчиком на динамічно формований об'єкт. У програмі її можна використовувати як звичайну змінну вказівного типу.

Можна вставити пункт і між існуючими пунктами меню. Припустимо, що в меню мається об'єкт — роздільник FileSep. Для того щоб вставити MI у меню над цим

роздільником, потрібно спочатку з'ясувати значення індексу роздільника в масиві пунктів і привласнити це значення змінній типу Integer:

Index := MenuFile.IndexOf(FileSep);

Для вставки об'єкта MI потрібно замість методу Add викликати метод Insert і передати йому як параметри значення Index і покажчик на новий об'єкт:

MenuFile.Insert(Index, MI); *{Уставити пункт меню відповідно значенню Index.}*

Значення індексу в масиві пунктів можна використовувати і для видалення одного з них. Для цього потрібно викликати метод Delete класу TMenuItem:

MenuFile.Delete(Index);

Видалення виконується також за допомогою методу Remove. Наприклад, наступний рядок видаляє команду Exit з меню File:

MenuFile.Remove(FileExit);

Модифікувати меню що розкривається або рядок меню можна, звернувшись до властивості Items відповідних об'єктів. Ця властивість являє собою масив об'єктів класу TMenuItem. Наприклад, вираження MenuMy.Items[I] посилається на об'єкт класу TMenuItem, що у масиві пунктів меню MenuMy має індекс I.

2.4.4.5. Підключення програмного коду до пункту меню

Після створення нового пункту меню потрібно підключити до нього процедуру обробки події, у якій будуть запрограмовані операції виконання відповідної команди. Для цього буде потрібно зробити наступне.

1) Оголосити процедуру обробки події в класі екранної форми.
2) Включити текст відповідної процедури в секцію реалізації модуля.
3) Підключити процедуру до події OnClick об'єкта відповідного пункту меню.

Наприклад, додайте у визначення класу TForm1 наступне оголошення (як правило, це й інші оголошення включаються в розділ private або public оголошення класу; у даному випадку ми оголосимо процедуру як private — закриту):

```
procedure NewCommandClick(Sender: TObject);
```

Потім де-небудь у секції реалізації модуля вставте текст процедури:

```
procedure TForm1.NewCommandClick(Sender: TObject);
begin
  ShowMessage('New command executed!');
end;
```

Зверніть увагу на формат оголошення в секції реалізації процедури обробки команди, що є, по суті, методом класу екранної форми. Спочатку стоїть ім'я класу, потім — крапка, а після цього — ім'я власне процедури. Таким чином, цей метод можна викликати з об'єкта класу TForm1. І, нарешті, потрібно оголосити змінну MI типу TMenuItem, створити об'єкт цього класу, привласнити значення його властивостям і підключити процедуру до події OnClick цього об'єкта:

```
MI := TMenuItem.Create(FileMenu);
MI.Caption := '&New command';
MI.OnClick := NewCommandClick;
```

Вставте або додайте новий об'єкт у меню. Тепер, якщо під час виконання програми користувач вибере в меню нову команду, програма викликає процедуру NewCommandClick.

І, звичайно, можна підключити до події OnClick нового об'єкта вже існуючу процедуру обробки команди іншого пункту (і навіть, можливо, іншого меню). Для цього буде потрібно тільки змінити праву частину в останньому операторі присвоювання — замінити NewCommandClick ім'ям існуючої процедури.

2.4.4.6. Додавання в меню імен файлів

Об'єкти компонента TMenuItem необов'язково повинні бути командами. Це можуть бути й інформаційні об'єкти, наприклад імена недавно використаних файлів. Їхній список можна створити і модифікувати за допомогою методів Add, Insert і Delete. Для початку

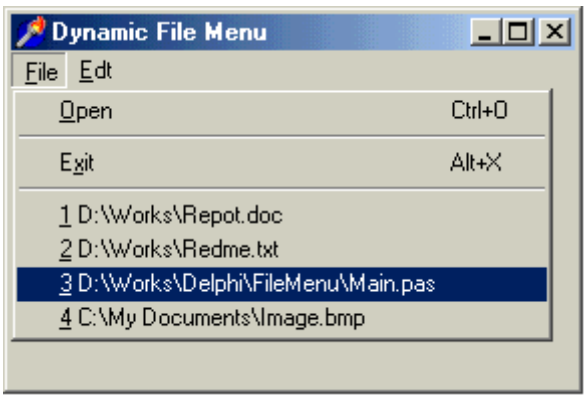
потрібно створити об'єкти класу TMenuItem і привласнити їх властивостям Caption імена файлів. Однак можна поступити інакше. Заздалегідь включити в це меню (як правило, у меню File) об'єкти-заглушки команд і потім привласнювати властивостям Caption імена файлів у міру того, як користувач їх відкриває.

Повний код головного модуля додатка що відображає імена відкритих файлів представлений у прикладі 2.4.1. На мал. 2.4.2 показано меню File цієї програми, у яке додано чотири пункти з іменами останніх відкритих файлів. Ці пункти можна вибрати, як і будь-які інші: кожний з них має клавішу швидкого виклику <Alt+N>, де N являє собою підкреслений у тексті пункту номер. Код обробки відповідної команди можна знайти в самому кінці приклада 2.4.1

Приклад 2.4.1 – Відображення імен відкритих файлів у меню

```
unit Main;
interface
uses
  Windows, SysUtils, Messages, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Menus, Buttons;
type
  TFormMain = class(TForm)
  MainMenu1: TMainMenu;
  MenuFile: TMenuItem;
  FileExit: TMenuItem;
  ButtonOpen: TButton;
  OpenDialog: TOpenDialog;
  FileOpen: TMenuItem;
  FileSep1: TMenuItem;
  FileSep2: TMenuItem;
  FileName1: TMenuItem;
  FileName2: TMenuItem;
  FileName3: TMenuItem;
  FileName4: TMenuItem;
  BitBtn1: TBitBtn;
  procedure FileExitClick(Sender: TObject);
  procedure ButtonOpenClick(Sender: TObject);
  procedure FileNamelClick(Sender: TObject);
  private
    {Оголошення закритих (private) членів.}
  public
    {Оголошення загальнодоступних (public) членів. }
end;
var
  FormMain: TFormMain;
implementation
{$R *.DFM}
procedure TFormMain.FileExitClick(Sender: TObject);
begin
  Close;
end;
{Запит імені файлу і додавання імені в меню File.}
procedure TFormMain. ButtonOpenClick(Sender: TObject);
var
  S: string;
  I, K: Integer;
```

```
begin
  if OpenDialog.Execute then with MenuFile do
    begin
      if not FileSep2.Visible then
      FileSep2.Visible := True; {Зробити роздільник видимим.}
      K:= IndexOf(FileName1);
      for I := Count - 1 downto ДО + 1 do
        begin {Змістити поточне ім'я файлу на один пункт униз.}
          S := Items[I - 1].Caption;
          S[2] := Chr(Ord('0') + (I - ДО + 1)); {Код клавіші швидкого виклику.}
          Items[I].Caption := S;
          Items[I].Visible := Items[I - 1].Visible;
        end;
      FileName1.Caption := '&1 ' + OpenDialog.Filename;
      FileName1.Visible := True;
      ShowMessage('Adding: ' + OpenDialog.Filename);
    end;
end;
{Прочитати ім'я файлу, обране в меню File.}
procedure TFormMain.FileName1Click(Sender: TObject);
var
  Filename: string;
begin
  with Sender as TMenuItem do
  begin
    Filename := Caption;
    System.Delete(Filename, 1, 2);
  end;
  ShowMessage('Selected: ' + Filename);
end;
end.
```



Малюнок 2.4.2 – Розповсюджений спосіб включення в меню File списку останніх відкритих файлів

Процедура FileName1Click у прикладі 2.4.1 демонструє метод, що дозволяє уникнути складностей, що час від часу виникають. Справа в тім, що в процедурі викликається системна процедура Delete мови Pascal для видалення перших двох символів у найменуванні пункту меню. Ці два символи - службові; вони визначають клавішу швидкого виклику пункту. Після видалення залишиться “чисте” ім'я файлу. Усе було б добре, якби перед цим оператором була відсутня вказівка with - do, у якій пропонується мати на увазі надалі члени класу TMenuItem, один із яких також називається Delete. Усунути можливість конфлікту слід шляхом кваліфікування, поставивши перед найменуванням процедури Delete префікс System і крапку. Таким чином, компілятор

одержав явну указівку використовувати процедуру Delete, що має справу зі строковими змінними, котра знаходиться в модулі System, а не однойменний метод класу TMenuItem. Подібний прийом вам може знадобитися в майбутньому при роботі з такими розповсюдженими іменами процедур, як Insert, Text, Assign і Close.

Нижче приведена інструкція, що допоможе вам сформувати меню File і додати в нього імена файлів у порядку звертання до них (за принципом тільки що відкритий — перший у списку).

1) Вставте об'єкт компонента MainMenu в екранну форму і за допомогою Menu Designer створіть меню File. У цьому меню один з пунктів зробіть роздільником — для цього в поле значення його властивості Caption уведіть знак дефіса (-). Після роздільника включіть в меню ще чотири порожніх пункти — це будуть заглушки для пунктів імен файлів. Як приклад можна використовувати мал. 2.4.2. Об'єкт меню назвіть MenuFile, а об'єкти пунктів-заглушок — FileName1, FileName2, FileName3 і FileName4. Об'єкти пунктів-роздільників назвіть FileSep1 і FileSep2. У табл. 2.4.2 перераховані імена і властивості найбільш істотних компонентів екранної форми.
2) Встановіть значення False для властивостей Visible роздільника і всіх чотирьох пунктів-заглушок. Програма зробить їх видимими після того, як користувач відкриє файли.
3) Сформуйте заготовку процедури обробки події для першого об'єкта-заглушки, вибравши її у вікні Menu Designer. Уставте код із процедури FileName1Click приклада 2.4.1. Ця процедура виводить повідомлення з ім'ям обраного в меню файлу. Цю ж процедуру підключіть до інших пунктів-заглушок.
4) Вставте в екранну форму об'єкт компонента OpenDialog.
5) Сформуйте заготовку процедури обробки команди Open або кнопки і вставте в неї код із процедури ButtonOpenClick приклада 2.4.1.

Таблиця 2.4.2 – Властивості об'єктів додатка що відображає імена відкритих файлів

Компонент	***Найменування об'єкта***	***Властивість***	***Значення***
Form	FormMain	Caption	Dynamic File Menu
Button	ButtonOpen	Caption	Open file
BitBtn	BitBtn1	Kind	bkClose
OpenDiatog	OpenDialog	Filter	All files (*.*)|*.*
MainMenu	MainMenu1		
MenuItem	FileOpen	Caption	&Open
MenuItem	FileOpen	ShortCut	Ctrl+O
MenuItem	FileExit	Caption	E&xit
MenuItem	FileExit	ShortCut	Alt+X
MenuItem	FileSep1	Caption	-
MenuItem	FileSep2	Caption	-
		Visible	False
MenuItem	FileName1	Caption	&1 name
		Visible	False
MenuItem	FileName2	Caption	&2 name
		Visible	False
MenuItem	FileName3	Caption	&3 name
		Visible	False
MenuItem	FileName4	Caption	&4 name
		Visible	False

2.4.4.7. Використання клавіш-акселераторів

У будь-який додаток, що використовує об'єкти компонентів TMainMenu або TPopupMenu, Delphi включає модуль Menus, у якому мається чотири досить корисні

підпрограми. Вони допомагають призначати пунктам клавіші-акселератори і потім програмувати їхнє використання. Перш ніж продовжити, включіть об'єкт компонента TMainMenu в екранну форму і за допомогою Menu Designer створіть меню з хоча б одним пунктом Demo. Залишимо для нього пропоноване у властивості Name за замовчуванням ім'я Demo1.

Для підключення до пункту меню клавіші-акселератора можна використовувати функцію ShortCut. Вона визначена в Delphi у такий спосіб:

function ShortCut(Key: Word; Shift: TShiftState): TShortCut;

Підключення клавіші <F9> як акселератор до пункту меню Demo1 (тобто присвоєння її значення властивості ShortCut) виконується наступним оператором, у якому використана функція ShortCut (як і раніше, додайте цей оператор у процедуру обробки щиглика на кнопці):

Demo1.ShortCut := ShortCut(vk_F9, []);

Delphi автоматично виведе назву клавіші-акселератора в пункті найменування, меню що розкривається, правіше команди Demo. Змінювати що-небудь вручну в значенні властивості Caption не прийдеться.

Порожні квадратні дужки в операторі означають порожню множину ознак спеціальних клавіш (тип даних TShiftState). Наприклад, якщо ви плануєте підключити як акселератор комбінацію <Alt+F9>, те задайте параметр Shift у такий спосіб:

Demo1.ShortCut := ShortCut(vk_F9, [ssAlt]);

Оскільки параметр Shift має тип множини Pascal, можна задати його множиною значень, розділених комами. Наприклад, це вираження установить як акселератор комбінацію <Ctrl+Shift+A> (порядок окремих елементів у множині значення не має):

Demo1.Shortcut := ShortCut(Ord('A'), [ssCtrl, ssShift]);

Не можна задавати для клавіші A її віртуальний код vk_A. Замість цього тут використовується функція Ord Pascal і символьний літерал в одиночних лапках, що визначає код ASCII відповідного символу.

Функцію ShortCutToKey можна використовувати для того, щоб розкласти комбінацію акселератора на окремі складові. Ця функція визначена в Pascal у такий спосіб:

procedure ShortCutToKey(Shortcut: TShortCut; **var** Key: Word; **var** Shift: TShiftState);

Як перший аргумент функції потрібно передати значення властивості ShortCut об'єкта пункту меню. Процедура привласнює віртуальний код клавіші параметру Key і множину ознак стану спеціальних клавіш — параметру Shift.

Інша функція, TextToShortCut, інтерпретує текстовий рядок і перетворить його в значення, яке можна привласнити властивості Shortcut. Наприклад, за допомогою наступного оператора можна задати як акселератор комбінацію клавіш <Alt+X>:

Demo1.ShortCut := TextToShortCut('Alt+X');

Цей метод можна використовувати в програмі для того, щоб запросити в користувача текстове визначення акселератора, а потім призначити його деякому пункту меню. У такий спосіб можна організувати в програмі адаптацію користувачем меню відповідно до своїх специфічних потреб:

```
var
  S: string;
begin
  S := InputBox(‘Input Dialog’, ‘Enter shortcut key’, ‘’);
  if Length(S) > 0 then
  Demo1.ShortCut := TextToShortCut(S);
end;
```

Функція InputBox дуже зручна для програмування запиту на введення користувачем деякого тексту. Перший строковий параметр функції — найменування діалогового вікна, другий — текст запиту, третій — пропоноване за замовчуванням значення, що

повертається, якщо користувач, не ввівши ні єдиного символу, відразу ж натисне <Enter>. У даному прикладі цей параметр порожній.

Для зворотного перетворення можна використовувати функцію ShortCutToText, що визначена в такий спосіб:

```
function ShortCutToText(Shortcut: TShortCut): string;
```

Наприклад, так можна вивести на екран інформацію про акселератор пункту меню:

```
ShowMessage('Demo1 shortcut = ' + ShortCutToText(Demo1.ShortCut));
```

2.4.4.8. Дозвіл та блокування команд

Вище вже було розглянуто як використовувати властивість Enabled об'єкта MenuItem для дозволу і блокування відповідного пункту меню. Коли властивість Enabled має значення False, найменування пункту виводиться в меню затіненим і користувач не може його вибрати.

При програмуванні додатка потрібно дуже ретельно продумати, у якій ситуації варто блокувати той або інший пункт меню. Найпростіша методика — розробити окрему процедуру, що керувала б “дієздатністю” пунктів меню відповідно до стану набору системних прапорців. Наприклад, визначите в програмі змінну FileIsOpen типу Boolean і розробіть процедуру, що керувала б станом пунктів Open, Save і інших у меню File відповідно значенню цієї змінної.

Про те, як використовувати дану методику для керування загальноприйнятими пунктами меню File, можна довідатися розглянувши наступний приклад. У класі екранної форми головного вікна додатка TMainForm оголошена процедура EnableMenu:

```
procedure EnableMenu;
```

Код реалізації процедури представлений у прикладі 2.4.2. Команди в меню File дозволяються або блокуються відповідно до поточної ситуації в програмі. Це тільки фрагмент коду додатка.

Приклад 2.4.2 - Процедура EnableMenu для блокування/розблокування пунктів меню

```
procedure TFormMain.EnableMenu;
var
  I: Integer;
begin
  with FileMenu do
    begin
      for I := 0 to Count - 1 do {Розблокувати команди меню File.}
        Items[I].Enabled := True;
      if not FileDirty then
        begin
          FileSave.Enabled := False; {Потрібно використовувати Save as.}
          if Length(Filename) = 0 then {Іншими словами, файлові не привласнене ім'я.}
            begin {Немає модифікацій, немає імені.}
              FileSaveAs.Enabled := False; {Нічого зберігати.}
              FilePrint.Enabled := False; { Нічого роздруковувати.}
            end;
        end;
    end;
end;
```

Прапор FileDirty у цій програмі повідомляє, чи вносилися які-небудь зміни у відкритий файл. Попередньо процедура EnableMenu розблокує в циклі всі пункти меню File (об'єкта MenuFile). Потім, якщо FileDirty скинутий (файл не змінювався з моменту завантаження в додаток або останнього збереження), пункт Save блокується. У цьому ж випадку, якщо виконуються ще які-небудь умови, блокуються і пункти меню Save As і Print.

Виклик процедури EnableMenu потрібно включити в процедуру обробки події OnClick об'єкта MenuFile. У результаті стан пунктів меню буде встановлено саме перед тим,

як меню з'явиться на екрані. От як виглядає процедура обробки події OnClick у даному додатку:

```
procedure TFormMain. MenuFileClick(Sender: TObject);
begin
  EnableMenu; {Дозвіл/блокування команд перед відкриттям меню.}
end;
```

У більш складних додатках приходиться розробляти окремі процедури блокування/розблокування пунктів для кожного меню що розкривається — File, Edit, Window і т.д. Ці процедури викликаються в процедурах обробки подій OnClick відповідних об'єктів — пунктів MainMenu. Але врахуйте, що це не блокує звертання до відповідних команд за допомогою клавіш швидкого виклику. Самий надійний спосіб цілком заблокувати команду — продублювати аналіз відповідних умов у процедурі її обробки. Наприклад, у процедурі обробки команди File⇨Save можна запрограмувати вихід із процедури без всяких операцій у випадку, якщо скинуто прапорець зміни файлу.

2.4.4.9. Злиття та розділення об'єктів MainMenu

У будь-якій екранній формі може бути присутнім кілька об'єктів MainMenu, що при необхідності можна зливати і створювати в такий спосіб динамічний рядок меню. Ця методика особливо ефективна при роботі з декількома екранними формами в додатку. Коли програма виводить на екран вторинну форму (неголовне вікно додатка), команди відповідного об'єкта MainMenu автоматично "уливаються" у меню головного вікна. Наприклад, діалогове вікно настроювання параметрів конфігурації додатка може за допомогою цієї технології уключити свої команди в рядок меню головного вікна. Після завершення роботи з діалоговим вікном "улиті" пункти автоматично зникнуть.

Злиття об'єктів MainMenu. Ключовим елементом технології злиття меню є байт GroupIndex класу TMenuItem. Його величина, що може змінюватися в діапазоні від 0 до 255, і визначає характер злиття. Якщо значення GroupIndex для двох об'єктів класу рівні, то одне меню заміняє інше. Якщо ж значення GroupIndex одного об'єкта менше значення іншого, перше меню буде вставлено ліворуч у друге. А якщо значення GroupIndex одного об'єкта більше значення іншого, перше меню буде додано праворуч у друге.

Властивостям GroupIndex різних об'єктів має сенс привласнювати значення, кратні 10, щоб легше було програмувати вставку одного меню між іншими.

Нижче приведена інструкція, що допоможе вам в експериментах зі злиттям двох об'єктів MainMenu в одній і тій же екранній формі.

1) Включіть два об'єкти компонента MainMenu в екранну форму. Для простоти можна використовувати в цьому експерименті імена об'єктів, пропоновані Delphi за замовчуванням (MainMenu1 і MainMenu2).
2) Двічі клацніть на об'єкті MainMenu1 і за допомогою Menu Designer створіть меню Demo з єдиною командою Advanced. Об'єктові цього пункту меню Delphi привласнить ім'я за замовчуванням (Demo1). Виберіть об'єкт Demo1 у вікні Object Inspector і встановіть для його властивості GroupIndex значення 0 (значення за замовчуванням).
3) Двічі клацніть на об'єкті MainMenu2 і за допомогою Menu Designer створіть меню Extra з єдиною командою Exit. Об'єктові цього пункту меню Delphi привласнить ім'я за замовчуванням (Extra1). Виберіть об'єкт Extra1 у вікні Object Inspector і встановіть для його властивості GroupIndex значення 1.
4) Створіть заготовку для процедури обробки команди Advanced — виберіть її в екранній формі або у вікні Menu Designer і вставте наступний оператор між рядків ключових слів begin і end:
   MainMenul.Merge(MainMenu2);
5) Створіть заготовку для процедури обробки команди Exit. Оскільки об'єкт MainMenu2 не відображається у формі, потрібно буде використовувати тільки Menu Designer для вибору команди. Вставте в текст процедури оператор Close.

6) Відкомпілюйте і запустіть додаток на виконання, натиснувши <F9>. Виберіть у меню команду Demo⇨Advanced — відбудеться злиття основного меню з додатковим. У результаті можна буде “добратися” до команди Exit.

Спробуйте варіювати значення властивостей GroupIndex об'єктів. Наприклад, установіть значення 1 для GroupIndex обох об'єктів — і Demo1, і Exit1. При такому наборі значень після вибору команди Advanced меню Extra замінить меню Demo. Спробуйте ще один варіант — нехай значення GroupIndex об'єкта Demo1 буде більше значення цієї ж властивості об'єкта Exit1. Тепер після вибору команди Advanced меню Extra буде вставлено ліворуч від меню Demo.

Поділ об'єктів MainMenu. Скасувати результат злиття меню можна, викликавши метод UnMerge будь-якого об'єкта MainMenu. Наприклад, вставте в екранну форму кнопку і включіть в процедуру обробки щиглика на цій кнопці (події OnClick) оператор, що від'єднує MainMenu2 від MainMenu1:

MainMenul.UnMerge(MainMenu2);

2.4.5. Компонент TActionList

Ретельно продуманий інтерфейс дозволяє користувачеві вибирати прийнятний варіант рішення однієї і тієї ж задачі в додатку з декількох можливих. Таким чином, додаток можуть використовувати люди з різним рівнем підготовки, причому в міру освоєння методики роботи з програмою можна переходити від більш очевидних, але повільних способів (наприклад, навігація по каскадним меню) до більш складних, але швидким (використання клавіш-акселераторів або піктографічних панелей інструментів).

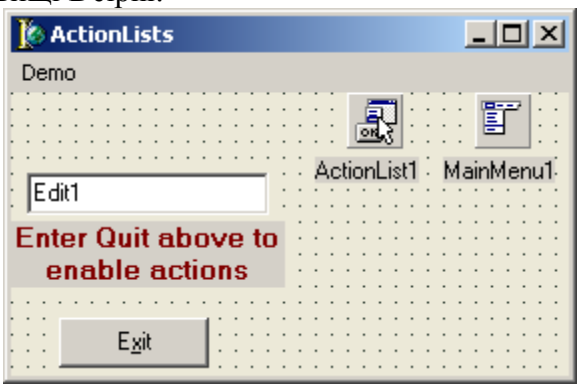
Що стосується розроблювачів, те чим ближче додаток до ідеалу в змісті різноваріантності інтерфейсу, тим більше турбот у програміста, що повинний у будь-якому варіанті забезпечити надійне функціонування програми. Додавання до програми з парою дюжин команд меню ще і панелей інструментів, а також командних кнопок вимагає великих зусиль за узгодженням роботи всіх цих компонентів, що повинні спільно використовувати різні фрагменти програмного коду. Зверніть увагу тільки на один аспект — потрібно узгоджено блокувати всі засоби звертання до деякої команди у випадку, якщо в даному стані програми вона не допускається. На перший план при цьому виступає необхідність уникнути дублювання, тобто ситуації, коли та сама, по суті, операція виконується різними фрагментами програми. Таке дублювання зрештою неминуче приводить до ненадійної роботи програми.

У Delphi для рішення цієї проблеми пропонується концепція використання спеціального компонента, що повинний містити код, поділюваний між іншими компонентами, що діють одноманітно. У категорію Standard палітри компонентів включений новий компонент TActionList, що може зберігати один або більш об'єктів типу Action. Кожен об'єкт Action являє собою свого роду вузол, до якого можна приєднати інші об'єкти. Для того щоб забезпечити єдність властивостей і подій для ряду об'єктів (наприклад, пунктів меню або кнопок), потрібно запрограмувати необхідні властивості і процедури для одного об'єкта Action і підключити до нього об'єкти з аналогічними характеристиками. У результаті кожний з таких підключених об'єктів буде “вибирати” характеристики з “вузлового” об'єкта Action і використовувати їх так, начебто вони запрограмовані саме для нього.

Компонент TActionList запрограмований у класі TActionList. Компонентові TAction відповідає клас TAction. На палітрі компонентів присутній тільки ActionList. Командні об'єкти формуються за допомогою спеціального редактора операцій Delphi (Action List Editor), про який і піде мова нижче. Взаємини між ActionList і Action багато в чому нагадують взаємини між компонентами TMainMenu і TMenuitem - об'єкт MainMenu “підпорядковує” (або він містить) кілька об'єктів MenuItem.

У прикладі 2.4.3 приведений код головного модуля проекту Actions, що допоможе нам розібратися в цій технології на прикладі програмування компонента CommandList. На мал. 2.4.3 показано, як виглядає цей додаток на стадії розробки в середовищі Delphi. Програма демонструє, як об'єкт компонента TActionList зберігає об'єкт Action, що забезпечує загальними властивостями і методами два і більш інших об'єкти.

Необхідно ввести в поле редагування Quit, щоб розблокувати кнопку Exit і однойменну команду меню. Тепер можна або вибрати команду меню, або клацнути на кнопці. Результат буде той же - програма завершить роботу і ви знову опинитесь в середовищі Delphi.



Малюнок 2.4.3 - Проект Actions демонструє методику використання компонента TActionLists

Приклад 2.4.3 – Використання об'єкта ActionList

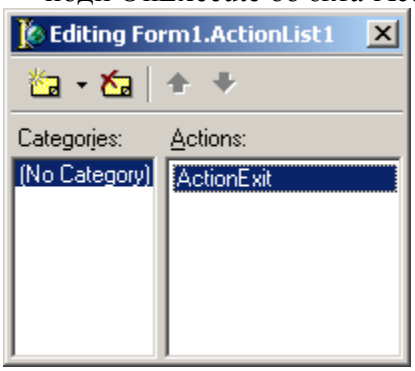
```
unit Main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Menus, ActnList, StdCtrls;
type
  TForm1 = class(TForm)
  MainMenu1: TMainMenu;
  Edit1: TEdit;
  Button1: TButton;
  Label1: TLabel;
  ActionList1: TActionList;
  Demo1: TMenuItem;
  Exit1: TMenuItem;
  ActionExit: TAction;
  procedure ActionExitExecute(Sender: TObject);
  procedure ActionExitUpdate(Sender: TObject);
  private
    {Оголошення закритих (private) членів.)
  public
    {Оголошення загальнодоступних (public) членів.)
end;
var
  Form1: TForm1;
Implementation
{$R *.DFM)
procedure TForml.ActionExitExecute(Sender: TObject);
begin
  Close; //Вихід із програми
end;
```

```
procedure TForml.ActionExitUpdate(Sender: TObject);
var
  Flag: Boolean;
begin  //Установка Flag, якщо користувач набрав Quit у Edit1.
  if Lowercase(Edit1.Text) = 'quit'
  then Flag := True
  else Flag := False;
    Button1.Enabled := Flag;  //Дозвіл або блокування Button1
    Exit1.Enabled := Flag;  //Дозвіл або блокування пункту меню
end;
end.
```

Нижче приведена покрокова інструкція, що допоможе вам освоїти технологію використання компонента TActionList.

1) Відкрийте новий додаток. Вставте в екранну форму об'єкти компонентів TMainMenu, TEdit, TButton, TLabel і TActionList (по одному кожного типу).
2) Двічі клацніть на MainMenu1 і створіть меню Demo, а в ньому — команду Exit. У результаті з'явиться об'єкт Exit1, що у вікні форми видний не буде.
3) Встановіть для властивостей Enabled об'єктів Button1 і Exit1 значення False (об'єкт Exit1 виберіть у списку, що розкривається, Object Inspector). У такий спосіб буде установлене вихідне значення властивостей двох командних об'єктів. Розблокування їх буде виконано через поділюваний об'єкт компонента TActionList.
4) Двічі клацніть на об'єкті компонента TActionList. Як і об'єкт будь-якого іншого невізуального компонента, зокрема TMainMenu, об'єкт ActionList буде представлений у вікні проектування екранної форми піктограмою. Під час виконання програми він узагалі видний не буде. Після подвійного щиглика відкриється вікно редактора операцій, у якому можна створювати об'єкти Action (мал. 2.4.4).
5) Клацніть на піктограмі New Action (перша ліворуч) і створіть тим самим новий об'єкт Action. За замовчуванням йому буде привласнене ім'я Action1. У цьому прикладі ми будемо використовувати тільки один об'єкт компонента TAction, але у своїх додатках ви можете створювати їх стільки, скільки вважаєте потрібним включити в один об'єкт компонента TActionList.
6) У списку вікна, що розкривається, Object Inspector виберіть тільки що створений об'єкт Action1. (Швидше за все, він вже обраний Delphi автоматично.) Можливо, знадобиться змінити його ім'я, щоб воно відбивало призначення цього об'єкта в програмі. У даному випадку можна установити для властивості Name цього об'єкта значення ActionExit. Властивості Caption привласніть значення E&xit.
7) Клацніть на вкладці Events у вікні Object Inspector. Двічі клацніть на кожній із двох подій об'єкта ActionExit — OnExecute і OnUpdate. Delphi створить заготовку процедур обробки цих подій. Вставте в них код із приклада 2.4.4.
8) Останній етап роботи з об'єктом компонента TAction — підключення до нього тих елементів керування (об'єктів), що мають ідентичні властивості і повинні однаково реагувати на події. Для цього виберіть об'єкт Exit1 у списку вікна, що розкривається, Object Inspector і привласніть його властивості Action значення ActionExit. Точно так само виберіть у списку об'єкт Button1 і привласніть його властивості Action те ж саме значення ActionExit. Тепер значення властивостей і процедури обробки подій об'єкта ActionExit будуть спільно використовуватися пунктом меню і кнопкою. Зверніть увагу на те, що значення властивостей Caption підключених об'єктів змінилися у відповідності зі значеннями властивості Caption об'єкта ActionExit.

9) Відкомпілюйте і запустіть додаток на виконання, натиснувши <F9>. Уведіть Quit у полі редагування. Після цього команда в меню і кнопка розблокуються. Тепер, якщо ви виберете кожну з них, буде викликана поділювана процедура обробки події OnExecute об'єкта ActionExit і програма завершить роботу.



Малюнок 2.4.4 - Вікно редактора операцій, що відкривається після подвійного щиглика на об'єкті компонента TActionList

Властивість Action класу TAction - це новинка Delphi 4. Властивість з таким же ім'ям оголошена як закритий член у класах деяких інших компонентів, таких як TMenuItem, TButton, TBitBtn, TSpeedButton і ін.

Кожен об'єкт компонента TAction (як, наприклад, ActionExit у нашій програмі) має дві події — OnExecute і OnUpdate. У процедурі обробки події OnExecute визначається, що буде виконано програмою при виборі користувачем будь-якого підключеного об'єкта, такого як Button або MenuItem. Підключення об'єктів елементів керування до об'єкта компонента TAction здійснюється шляхом настроювання властивості Action об'єктів, що підключаються. Цій властивості повинне бути привласнене ім'я об'єкта компонента TAction, до якого планується підключення (ActionExit у нашій програмі). У процедурі ж обробки події OnExecute мається поділюваний усіма підключеними елементами керування код — виклик функції Close.

Точно так процедура обробки події OnUpdate об'єкта компонента TAction дозволяє узгоджено змінити стан усіх підключених елементів керування. У нашій програмі в такий спосіб змінюється значення властивості Enabled на True або False у залежності від того, чи введений в об'єкт Edit1 текст quit (регістр символів у даному випадку не має значення). Уведений текст є значенням властивості Text цього об'єкта. Зміна ж значення властивості Enabled у процедурі обробки події OnUpdate об'єкта ActionExit впливає на підключені об'єкти Button1 і Exit1. Подія OnUpdate час від часу активізується, поки додаток знаходиться в стані чекання, так що в даній процедурі не можна програмувати тривалу обробку. Аналіз стану одного з об'єктів екранної форми й установка значення властивості Enabled по результаті цього аналізу рівним True або False — це саме та дія, яку варто виконувати в процедурі обробки події OnUpdate об'єкта компонента TAction.

Доцільно вставляти об'єкт компонента TActionList у модуль даних. Це дозволить усунути перевантаження екранної форми вікна додатка невізуальними об'єктами, що ускладнюють компонування елементів керування в ній. Крім того, доступ до модуля даних і його об'єктів ActionList і Action легко організувати з модуля будь-якої видимої екранної форми. Потрібно тільки подбати про взаємне оголошення модулів у директивах uses.

2.4.6. Корисні поради

✍ Для того щоб зібрати статистичну інформацію про звертання користувача до різних пунктів меню в процесі роботи з додатком, можна використовувати властивість Tag об'єктів MenuItem. При кожнім виборі пункту меню (події OnClick цього об'єкта) можна нарощувати код у даному полі. Перед завершенням сеансу роботи з додатком значення властивостей Tag для всіх об'єктів меню

можна роздрукувати й у такий спосіб одержати реальну статистику частоти виклику окремих пунктів. Результати аналізу цих даних допоможуть оптимизировать компонування меню, наприклад має сенс пункти, що викликаються частіше інших, розмістити вище в меню що розкривається.

✍ Можна оперативно перекомпоновувати меню в процесі роботи програми таким чином, щоб найчастіше використовувані пункти виявилися у верхній частині меню і до них можна було швидше добратися. Просто-напросто при кожнім виборі деякого пункту меню переставляйте його на першу позицію. Таким чином, останні обрані команди виявляться на самому початку списку.

✍ Працюючи в Menu Designer, привласніть властивості Break об'єкта MenuItem значення mbBarBreak (вертикальний роздільник) або mbBreak (без роздільника), щоб скомпонувати сегментоване меню. Хоча такі меню і рідко зустрічаються в додатках Windows, вони допомагають користувачеві візуально виділити окремі сегменти складних меню, особливо якщо в них багато пунктів і всі пункти не вміщаються у вікні додатка.

✍ Для настроювання розміщення спливаючого меню щодо положення покажчика миші в момент щиглика правою кнопкою можна використовувати властивість Alignment об'єкта компонента PopupMenu. Наприклад, значення paCenter властивості Alignment центрує верхню границю вікна меню щодо положення покажчика. Але врахуйте, що ця властивість застосовна тільки до тих спливаючої меню, що підключені до властивості PopupMenu екранної форми.

✍ Компонент TActionList має властивість Images, яку можна використовувати для збереження декількох растрових зображень. Ці зображення потім можуть відображатися іншими, видимими об'єктами — MenuItem або BitBtn. Для зв'язування кожного зображення з окремим об'єктом компонента TAction потрібно використовувати властивість ImageIndex цього об'єкта. Таким чином, вирішується задача зв'язування часто використовуваних зображень з безліччю об'єктів елементів керування.

2.4.7. Резюме

Об'єкт компонента TMainMenu використовуйте для формування рядка меню головного вікна додатка.

Об'єкт компонента TPopupMenu призначений для формування спливаючих меню, що з'являються на екрані додатка після щиглика правою кнопкою миші. Можна також викликати метод Popup для виведення на екран спливаючого меню в будь-якому заданому місці екрана.

Усі пункти меню, породжені об'єктами компонентів TMainMenu і TPopupMenu, є об'єктами класу TMenuItem. Присвоєння значень властивостям Checked, Visible, Enabled і іншим об'єкта класу TMenuItem у процесі виконання програми дозволяє створювати динамічні меню.

За допомогою властивості Items об'єктів компонентів TMainMenu, TPopupMenu і TMenuItem можна звернутися до будь-якого об'єкта компонента TMenuItem, що утримується в меню цих типів. Індекс пункту знаходиться в діапазоні від 0 до Count - 1. Властивість Items є членом класу TMenuItem, що є масивом. Таким чином, при програмуванні роботи з цією властивістю можна використовувати весь арсенал засобів, передбачених для роботи з властивостями-масивами, зокрема можна звертатися по індексу або викликати методи, наприклад Insert.

Намагайтеся використовувати в додатку стандартні призначення клавіш-акселераторів, хоча Delphi і не обмежує вас у виборі підходящої комбінації клавіш. Для настроювання клавіш-акселераторів під час виконання програми

можна використовувати стандартні функції Shortcut, ShortCutToKey, TextToShortCut і ShortCutToText.

Найпростіший спосіб створити динамічне меню — уключити кілька об'єктів компонента TMainMenu в екранну форму і привласнювати під час виконання програми їхні імена властивості Menu екранної форми. Імена об'єктів компонентів TPopupMenu можна привласнити властивості PopupMenu екранної форми й організувати контекстне меню в додатку.

Інший простий спосіб створення динамічного меню базується на використанні властивості Visible об'єкта MenuItem, якій можна привласнити значення True або False. Це дозволяє відображати або ховати під керуванням програми як окремий пункт, так і все меню що розкривається або навіть рядок меню.

Для керування компонуванням меню в процесі виконання програми можна використовувати методи Add, Delete, Insert і Remove. Наприклад, з їхньою допомогою можна включати в меню File пункти відповідно іменам файлів, що відкриваються. Можна з цією метою включити на стадії проектування меню пункти-заглушки, а потім під час виконання програми динамічно змінювати їхні властивості, зокрема Visible і Caption.

Можна під час виконання програми динамічно створювати об'єкти класу TMenuItem. Створення об'єкта виконується методом Create, а потім потрібно використовувати метод Add або Insert для включення нового пункту у вже існуюче меню. Не забудьте призначити процедуру обробки події OnClick нового об'єкта.

Кілька об'єктів MainMenu можна злити, використовуючи властивості GroupIndex і AutoMerge. Можна зливати об'єкти MainMenu, розміщені як в одній екранній формі, так і в різних. Але врахуйте, що в MDI- і OLE-додатках використовується інша технологія злиття меню.

Для формування переносних меню можна використовувати шаблони і сценарії ресурсів. У результаті один раз створене меню можна буде включити в безліч додатків.

У Delphi починаючи з 4-ої версії з'явився новий компонент TActionList у категорії Standard палітри компонентів. З його допомогою можна сформувати об'єкти компонента TAction, властивості і методи якого будуть спільно використовуватися декількома об'єктами елементів керування з аналогічними функціями, наприклад об'єктами MenuItem і Button.

Література [1].

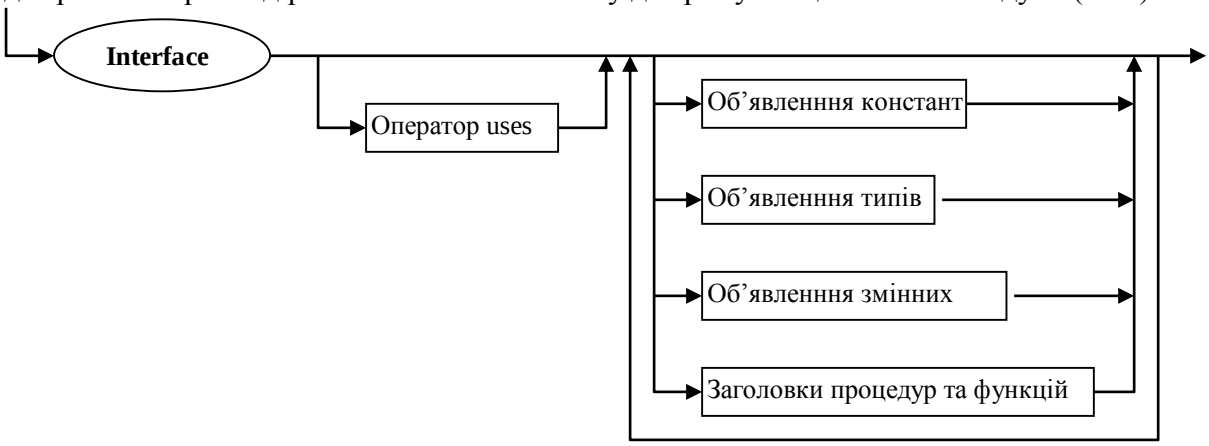
РОЗДІЛ 3 ОСНОВИ ПРОГРАМУВАННЯ В OBJECT PASCAL

Тема 3.1. Основи Object Pascal

3.1.1. Опис мови програмування Object Pascal

3.1.1.1. Синтаксис

Формально синтаксис Object Pascal можна представити графічно у виді синтаксичних діаграм. Як приклад розглянемо синтаксичну діаграму секції interface модуля (Unit).



Малюнок 3.1.1 – Синтаксична діаграма секції interface

На синтаксичних діаграмах *зарезервовані слова* Object Pascal обведені овалом. У правій частині діаграми знаходиться покажчик на іншу синтаксичну діаграму. При аналізі синтаксичної діаграми необхідно прямувати по шляху, відзначеному стрілками.

Секція interface кожного модуля починається директивою interface. Секція interface може або нічого не містити, або складатися з оператора uses і оголошень констант, типів і т.д. При наявності даного оператора його варто вводити перед іншими оголошеннями, зазначеними на діаграмі. Послідовність оголошень констант, типів, змінних, процедур і функцій не має значення.

3.1.1.2. Символи, що використовуються

У Object Pascal використовується набір символів основної таблиці кодів ASCII.

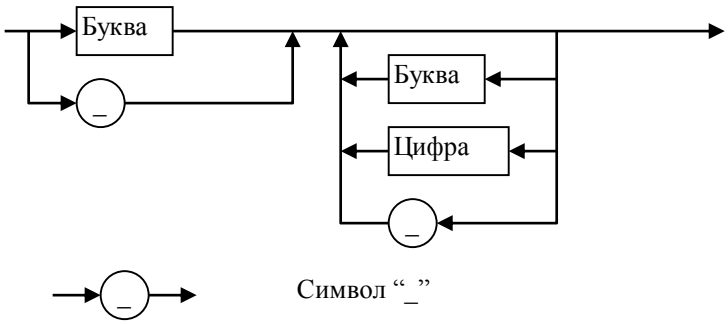
- Латинські прописні букви від A до Z, латинські малі літери від a до z.
- Цифри від 0 до 9.
- Шістнадцятиричні числа, утворені з цифр від 0 до 9 і букв від A до F (або від a до f).
- Символ пробілу і всі керуючі символи основної таблиці кодів, у тому числі і символ кінця рядка.

Крім стандартного набору символів, кожна мова програмування містить у собі зарезервовані слова, що мають строго визначене призначення і не можуть використовуватися як ідентифікатори.

Імена типів, змінних і процедур у програмі варто задавати, не використовуючи зарезервовані слова Object Pascal. Такі імена називаються *ідентифікаторами*.

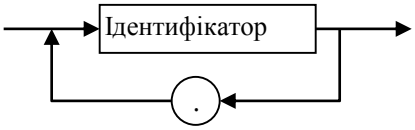
При завданні ідентифікаторів діють наступні обмеження:

- ідентифікатори можуть мати будь-як довжину, однак розпізнаються лише перші 255 символів;
- перший символ ідентифікатора повинний бути буквою або символом підкреслення (_);
- не допускається використання пробілів.



Малюнок 3.1.2 – Синтаксична діаграма ідентифікатора

Всі ідентифікатори в тексті програми повинні бути унікальними. Однак можлива ситуація, коли в різних модулях використовуються два однакових ідентифікатори. Щоб при наявності однакових ідентифікаторів вибрати потрібний, його варто *кваліфікувати*, тобто явно вказати необхідний ідентифікатор. Для цього перед ім'ям кожного ідентифікатора варто вказати через крапку ідентифікатор модуля.



Кваліфікований ідентифікатор

Малюнок 3.1.3 – Синтаксична діаграма операції кваліфікації

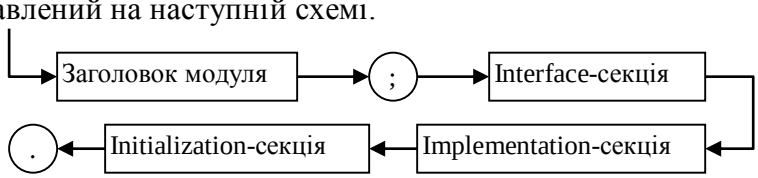
Прикладом кваліфікації може слугувати поштова адреса. В цьому випадку у якості ідентифікатора виступає назва вулиці, наприклад, вул. Будівельників. Але вулиця Будівельників може знаходитись в кількох містах. Щоб не помилитись потрібно вказати назву міста чи населеного пункту в якому знаходиться ця вулиця, наприклад, Олександрівка. В той же час населений пункт з тією ж назвою може міститись в іншій області чи районі. Так населені пункти з назвою Олександрівка знаходиться в Донецькій та Кіровоградській областях. Тому потрібно вказати область чи район в якому знаходиться населений пункт. З урахуванням вимог Object Pascal кваліфікований ідентифікатор поштової адреса для наведеного прикладу мав б наступний вигляд:

Донецька.Олександрівка.Будівельників

3.1.1.3. Програми

У Delphi написання будь-якої програми починається зі створення нового проекту в інтегрованому середовищі розробки – IDE.

При створенні в Delphi нової форми IDE автоматично створює програмний модуль (unit), синтаксис якого представлений на наступній схемі.



Модуль

Малюнок 3.1.4 – Синтаксична діаграма модуля

Створений Delphi модуль утворить каркас, використовуваний надалі для написання програмного коду. Після запуску нового проекту створюється також нова форма з ім'ям Form1. Кожна форма в Delphi описується в окремому модулі (файл із розширенням *.PAS), у якому містяться процедури обробки подій.

Приклад 3.1.1 – Код модуля, що містить форму (Form1). Текст у фігурних дужках є коментарем або директивами компілятора

unit Unit1;

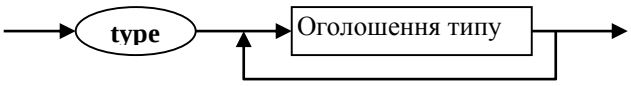
```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin

end;

end.
```

Модуль Object Pascal складається з п'яти секцій:

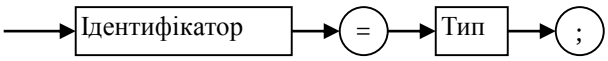
- заголовок модуля;
- секція interface;
- секція implementation;
- секція initialization (необов'язкова секція);
- секція finalization (необов'язкова секція).

3.1.1.4. Визначення та оголошення типів

Усі дані, використовувані програмою, повинні належати до якогось заздалегідь визначеного типу – стандартного (визначеного) або користувальницького (визначеного користувачем). Імена стандартних типів даних є визначеними ідентифікаторами і діють у будь-якій точці програми. Користувальницькі типи – це додаткові типи, що користувач може визначити самостійно. Тип визначає спосіб збереження даних у пам'яті і діапазон можливих значень.



Оголошення типів



Оголошення типу

Малюнок 3.1.5 – Синтаксична діаграма оголошення типу в Object Pascal

Для забезпечення виділення адресного простору під час компілювання всі змінні і константи повинні з'являтися на початку програми або програмного блоку. Для цього повинний бути визначений тип (type) змінних і констант. Він може бути визначений або заданий користувачем. Визначені користувачем типи даних повинні з'являтися в розділі оголошення типів, що знаходиться в програмі або модулі.

Типи, використовувані в Object Pascal, представлені далі.

- простий тип;
- строковий тип;
- структурований тип;
- посилальний тип;
- процедурний тип;
- тип variant.

Прості типи даних. Простий тип визначає упорядковану множину даних. Це означає, що для даних простого типу застосовні оператори =, <>, <, >, >= і <=. Простий тип може бути *порядковим* і *речовинним.*

Дані порядкового типу являють собою значення, упорядковані у визначеній послідовності. Кожен елемент послідовності має свій порядковий номер.

Порядковий тип, що задається користувачем – *піддіапазон* або *інтервал* – визначається за допомогою завдання підмножини значень порядкового типу. Інтервал задається вказівкою найменших і найбільших значень. Прикладом такого типу є тип integer. У наступному прикладі число *0* задається як найменше значення, а число *9* як найбільше значення піддіапазону:

Number=0..9;

Символи з кодової таблиці ASCII також мають порядковий тип. Тому можна задати тип:

UpperCaseCharacters=A..Z;

Ці два приклади представляють підмножини, що мають порядковий тип, вже визначений у Object Pascal. *Перелічуваний* тип, на відміну від порядкового, дає розроблювачеві можливість самому визначати тип даних.

Прикладом подібного визначення типу є наступний фрагмент програми:

TColo r= (Red, Orange, Yellow, Green, Blue); *{перелічуваний тип}*

THotColor = Red .. Yellow; *{піддіапазон}*

Строкові типи. Рядок може мати перемінну або фіксовану довжину. У приведеному нижче прикладі в прямокутних дужках указується довжина рядка як ціле число без знака.

Title = **string;**

FixString = **string**[25];

У Delphi визначені наступні строкові типи даних:

- короткий рядок (ShortString);
- довгий рядок (AnsiString);
- широкий рядок (WideString).

Короткі і довгі рядки використовуються у вираженнях. Короткий рядок являє собою рядок, для якого область пам'яті обсягом від 1 до 255 символів виділяється статично.

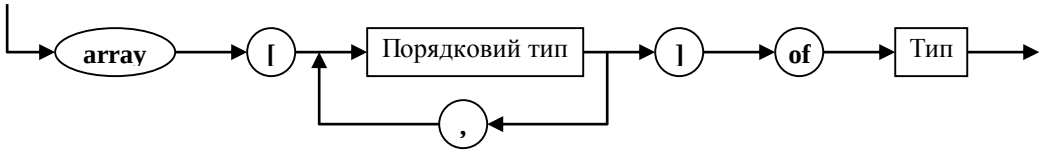
До типу *довгий рядок* відноситься рядок, для якого динамічно виділяється пам'ять, причому максимальна довжина обмежується тільки наявним у розпорядженні обсягом пам'яті. Зарезервоване слово string за замовчуванням представляє довгий рядок.

Тип *широкий рядок* складається з Unicode-символів (DBCS), що визначені як 16-розрядні символи і пам'ять для нього виділяється як і для довгого рядка динамічно.

Структуровані типи. Структурований тип є типом даних, що складається з інших типів. Структуровані типи даних підрозділяються на:

- тип “масив”;
- тип “запис”;
- тип “клас”;
- тип “посилання на клас”;
- тип “множина”;
- тип “файл”.

Масив. Тип “масив” (array) є нічим іншим як таблицею; доступ до даних масиву здійснюється за допомогою індексів.



Тип “массив”

Малюнок 3.1.6 – Синтаксична діаграма структурованого типу “масив”

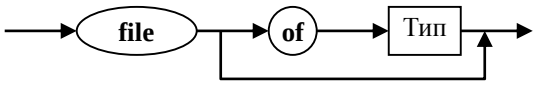
При використанні одного типу масиву як компонент типу іншого масиву він розглядається компілятором як єдиний багатомірний масив. Так, наприклад, наступне оголошення:

array[Boolean] **of array**[1..10] **of array**[Size] **of** extended

компілятором буде інтерпретуватися як

array[Boolean, 1..10, Size] **of** extended

Файл. Тип “файл” (file) забезпечує можливість доступу до даних, розміщених на зовнішньому носії інформації.



Тип “файл”

Малюнок 3.1.7 – Синтаксична діаграма структурованого типу “файл”

При використанні зарезервованого слова of говорять про застосування *типізованого файлу*, наприклад:

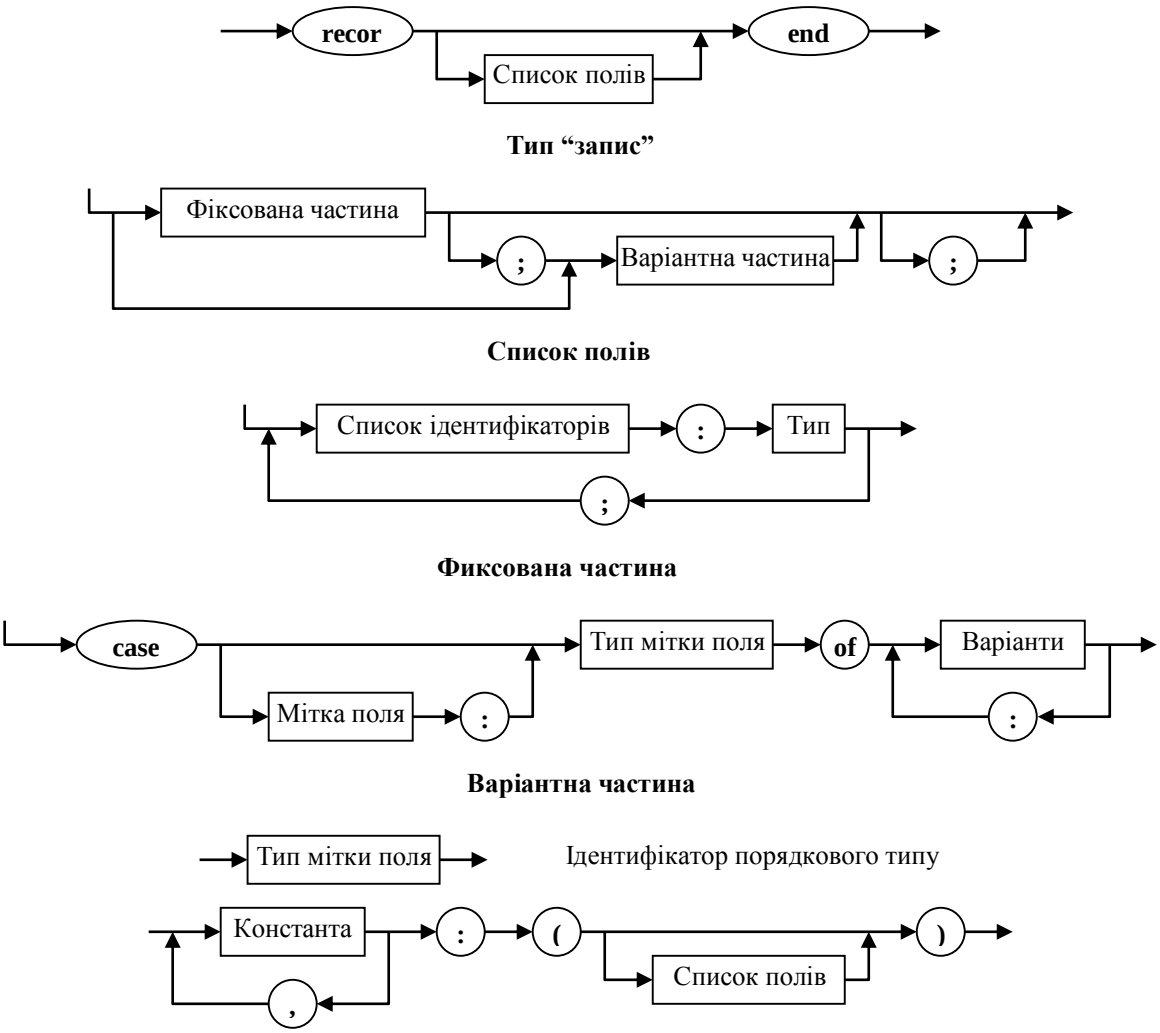
FData: **file of** extended;

Стандартний файл уведення/виведення має тип text. Такий файл складається із символів кодової таблиці ASCII.

Після зарезервованого слова of може стояти будь-який тип даних, за винятком:

- типу “файл”;
- структурованого типу з елементом типу “файл”;
- типу “клас”.

Запис. У типі “запис” (record) внутрішні змінні різних типів (поля) поєднуються в єдину логічну структуру.



Малюнок 3.1.8 – Синтаксична діаграма структурованого типу "запис"

Дані типу "запис" являють собою зв'язну структуру полів (даних різних типів), звертання до яких відбувається через ім'я запису. Розрізняють дві групи полів: поля, що належать до фіксованої частини запису, і поля, що належать до варіантної частини.

Фіксована частина являє собою перелік полів. Ці поля можуть належати до будь-якого типу даних. Прикладом визначення запису, призначеного для використання в адресному довіднику, може послужити наступний фрагмент програмного коду:

Приклад 3.1.2 – Фрагмент програмного коду модуля, що містить визначення запису

```
TClient = record
  Name: string[25];    //Прізвище
  FirstName: string[15];    //Ім'я
  City: string[25];    //Місто
  Street: string[25];    //Вулиця і номер будинку
  Phone: record
    CityCode: 0000..9999;    //Код міста
    Number: 0000000..9999999;    //Номер
  end; {запис Phone}
end; {запис TClient}
```

Якщо в розділі змінних модуля з'являються наступні змінні:

```
var
  Client: TClient;
```

то звертання до будь-яких полів здійснюється за допомогою кваліфікування:

Client.Phone.Number:=4411825;

Варіантна частина завжди міститься за фіксованою частиною. З погляду розроблювача розходження між ними полягає в тому, що змінні варіантної частини займають у пам'яті одне і теж місце. Припустимо, варто створити файл, що містить інформацію про клієнтів. Іноді при вказівці адреси повинна указуватися вулиця, в інший раз номер поштової скриньки, що складає максимум із шести цифр. Варіантна частина містить оператор case для того, щоб указати, який варіант поля буде використовуватися у визначеному випадку. Приклад доповнюється тепер варіантною частиною:

Приклад 3.1.3 – Фрагмент програмного коду модуля, що містить визначення запису

```
TAdressType = (AdrPostBox, AdrStreet);
TClient2 = record
  Name: string[25];    //Прізвище
  FirstName: string[15];    //Ім'я
  City: string[25];    //Місто
  Phone: record
    CityCode: 0000..9999;    //Код міста
    Number: 0000000..9999999;    //Номер
  end; {запис Phone}
case AdressType: TAdressType of    //Тип адреси
  AdrPostBox: (PostBoxNum: 0000..9999);    //Абонентська скринька
  AdrStreet (Street: string[25]);    // Вулиця і номер будинку
end; {запис TClient2}
```

У цьому прикладі використовується тип *мітки поля*. Тип “мітки поля” є перелічуваним типом, що повинний визначатися раніш чим він може використовуватися при декларуванні типу запису. У вищевказаному прикладі TAdressType є типом мітки поля. Це варіантне поле може приймати два значення: AdrPostBox і AdrStreet. У залежності від того, до якого варіанта ми звертаємося, компілятор по-різному інтерпретує дані, розташовані за адресою варіантного поля.

Client.AdressType:= AdrPostBox;

Client.PostBoxNum:=654;

На синтаксичних діаграмах видно, що варіантне поле може містити необов'язковий ідентифікатор – *мітку поля*, що визначає, яке з варіантних полів у даному випадку активне. *Ідентифікатором типу* називається символ, що знаходиться з лівої сторони від знака рівності в оголошенні типу.

Таким чином, для запису адреси клієнта в текстовий файл варто використовувати наступну конструкцію:

Приклад 3.1.4 – Фрагмент програмного коду для запису адреси в текстовий файл

```
if Client.AdressType = AdrPostBox then
  WriteLn(f, “Абонентська скринька № ”, Client.PostBoxNum) else
  WriteLn(f, “Поштова адреса: ”, Client.Street);
```

Мітку поля можна не вказувати, однак порядковий тип (тип мітки поля) необхідний.

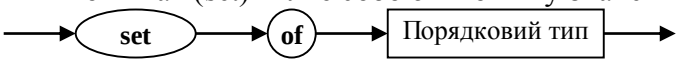
При рішенні питання, які з полів варто віднести до фіксованої частини запису, а які до варіантної, користуйтеся наступним критерієм: якщо поле часте використовується для збереження інформації, то воно заноситься у фіксовану частину запису, у противному випадку помістіть його у варіантну частину.

Клас. Тип “клас” (class) містить в собі більше, ніж просто структуру збереження даних, подібну до типу “запис”. Крім внутрішніх полів тип “клас” містить методи і властивості, що визначають його функціональні можливості.

Зарезервоване слово class використовується для вказівки *об'єктного типу*. Поняття *клас* і *об'єктний тип* можуть використовуватися при цьому як синоніми. Цей тип складається з визначеної кількості елементів, тобто полів, методів і властивостей. На відміну від інших типів

даних, тип “клас” може оголошуватись тільки як глобальний тип для даного модуля. Тому тип “об'єкт” не може визначатися усередині процедур і функцій.

Множина. Тип даних “множина” (set) являє собою множину значень порядкового типу.



Тип “множина”

Малюнок 3.1.9 – Синтаксична діаграма структурованого типу “множина”

Припустимо мається наступне визначення:

Приклад 3.1.5 – Фрагмент програмного коду, що містить визначення множини

```
type
  TColor=(Red, White, Blue);
  TColorSet = set of TColor;
var
  Color_Values: TColorSet;
begin
  Color_Values:= [Red, White];
end.
```

Змінна Color_Values охоплює тепер наступні множини:

[], [Red], [White], [Red, White], [White, Red]

Це значить, що всі наступні операції порівняння дають значення True (істинно):

[] **in** Color_Values

[Red] **in** Color_Values

[White] **in** Color_Values

[Red, White] **in** Color_Values

[White, Red] **in** Color_Values

Оператор in повертає True, якщо множина містить перемінну порядкового типу. Поряд з цим у Object Pascal визначені оператори порівняння множин. Припустимо, були оголошені множини A і B. Тоді дійсні наступні правила:

A=B Повертає значення True, якщо A и B містять ті самі елементи. У противному випадку повертається значення False.

A<=B Повертає значення True, якщо кожен елемент множини A входить у множину B.

A>=B Повертає значення True, якщо кожен елемент множини B входить у множину A.

Для додавання і зміни елементів множини можна використовувати оператори + і -.

Color_Values:= Color_Values + [Blue];

Color_Values:= Color_Values - [White];

{Color_Values тепер містить множину [Red, Blue]}

Процедурні типи. Дозволяють трактувати процедури і функції як значення, які можна привласнювати перемінним і передавати як параметри.

Функції можуть повертати дані будь-якого типу, за винятком типу “файл”.

Процедурні типи сумісні один з одним, якщо виконуються наступні умови:

- обоє процедурних типу мають однакова кількість параметрів;
- параметри займають однакові позиції і мають один тип; для сумісності імена параметрів не грають ніякої ролі;
- типи результатів, що повертаються функціями, ідентичні;
- будь-який процедурний тип сполучимо зі значенням nil.

Використання процедурного типу наочно демонструє наступний приклад.

Приклад 3.1.6 – Фрагмент програмного коду з використанням процедурного типу

```
type
  Proc = procedure;
  SwapProc = procedure(var X, Y: Integer);
```

```
StrProc = procedure(S: string);
MathFunc = function(X: Extended): Extended;
DeviceFunc = function(var F: Text): Integer;
MaxFunc = function(A, B: Extended; F: MathFunc): extended;
```

Пояснення до приклада:

- *Proc* визначений як процедурний тип для процедури, що не має ніяких параметрів;
- *SwapProc* являє собою процедурний тип, якому передаються параметр-перемінні;
- *StrProc* являє приклад оголошення процедурного типу з одним параметром-значенням типу string;
- *MathFunc* і *DeviceFunc* являють собою визначення процедурного типу, у яких функція визначається в залежності від переданого параметра-значення і перемінною-перемінній-параметр-перемінної відповідно;
- при визначенні *MaxFunc* для параметра-значення (F) указується процедурний тип *MathFunc*.

3.1.1.5. Константи

Константи бувають прості і типізовані. Проста константа з'являється як ім'я, якому привласнюється значення. При оголошенні типізованих констант указують не тільки її значення, але і її тип. Розходження полягає в тому, що значення простої константи не може змінюватися, у той час як для типізованої константи це можливо. Розуміється тип типізованої константи не може модифікуватися. Нетипізовані константи визначаються один раз на початку програми і потім зберігають своє значення. Типізовані константи визначаються один раз, як і нетипізовані, однак їхнього значення можна змінити під час виконання програми.

Объявление констант

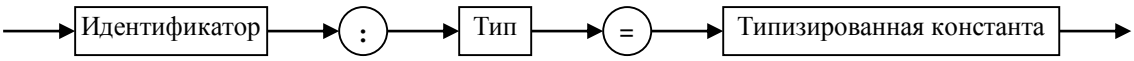
Малюнок 3.1.10 – Синтаксична діаграма роздягнула оголошення констант

Объявление констант

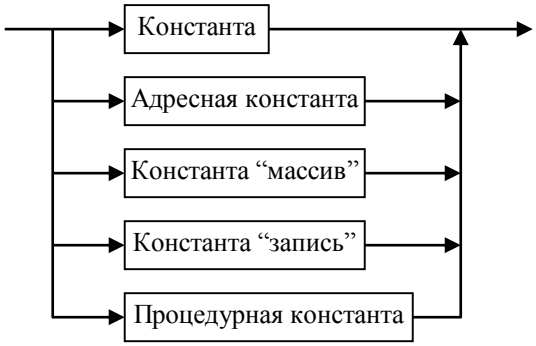
Малюнок 3.1.11 – Синтаксична діаграма оголошення констант

Можна привести наступні приклади визначення нетипизированых констант:

```
const
  Numbers = 10;
  HotColor = Red;
```



Объявление типизированных констант



Типизированная константа

Малюнок 3.1.12 – Синтаксична діаграма оголошення типізованих констант

Константа типу “масив”. Визначає значення елементів масиву. Елементи масиву можуть відповідати будь-як типові, за винятком типу “файл”, наприклад:

```
const
  Line: array[1..6] of integer = (1, 2, 6, 24, 120, 720);
```

Константа типу “запис”. Являє собою дуже зручну типізовану константу, оскільки з її допомогою ви можете инициализировать полючи запису. Для цього повинні виконаються деякі условая.

- Запис не повинний містити полів типу “ файл”.
- Поля повинні инициализироваться в тій же послідовності, у якій вони згадуються в оголошенні запису.
- Якщо застосовується запис з варіантною частиною, то може инициализироваться тільки те поле, що доступно у відповідності зі значенням мітки полючи.

Приклад 3.1.7 – Фрагмент програмного коду з використанням константи типу “запис”

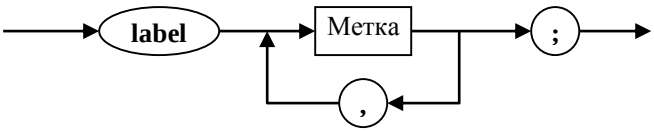
```
type
  TDate = record
    Day: 1..31;
    Month: 1..12;
    Year: 1900..2010;
  end;
const
  MyBirthday: TDate = (Day: 28; Month: 6; Year: 1977);
```

Константа процедурного типу. Вимагає, щоб був зазначений ідентифікатор процедури або функції, сумісної з типом константи.

3.1.2. Опис мови програмування Object Pascal (частина друга)

3.1.2.1. Мітки

Мітки являють собою адреси передачі керування в програмі. Як і в мові програмування Basic, мітки можуть бути представлені у виді чисел. У Object Pascal можуть використовуватися ідентифікатори міток, тому ці ідентифікатори варто повідомляти.



Объявление метки

Метка

Малюнок 3.1.13 – Синтаксична діаграма оголошення мітки

Застосування міток суперечить концепції структурованого програмування, тому застосовувати мітки не рекомендується. Однак при налагодженні програм мітки можуть надати істотну допомогу. Перехід на визначену мітку здійснюється оператором goto, наприклад:

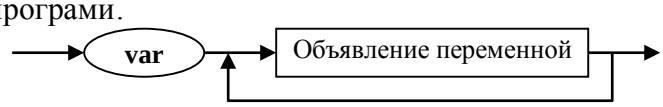
Приклад 3.1.8 – Фрагмент програмного коду з використанням міток

```
label 1, 2;
begin
…
 goto 1;
…
1: WriteLn (‘ результат 1’ );
2: WriteLn (‘ результат 2’ );
…
end;
```

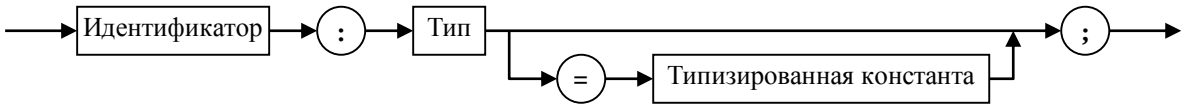
3.1.2.2. Змінні

Перемінні є найбільш важливими ідентифікаторами, використовуваними в програмному коді. Вони відповідають за виділення адресного простору, у якому розміщаються передані, що зберігаються, оброблювані і/або значення, що повертаються. Розмір виділюваної для них пам'яті і спосіб *інтерпретування* значень (переклад фізичних представлень даних визначеного типу у форму, зрозумілу для людини, і навпаки) залежать від типу перемінної.

Оголошення перемінних. Перемінні повинні з'являтися до моменту їхнього використання. Якщо перемінна оголошена, то під час виконання програми для неї резервується пам'ять. Такі перемінні являють собою *статичні перемінні*, для яких існує безпосередня залежність між ідентифікатором і місцем збереження значення в пам'яті. Поряд з такими перемінними існують *динамічні перемінні*. Пам'ять для цих перемінних виділяється в міру потреби в ході виконання програми.



Объявление переменных



Объявление переменной

Малюнок 3.1.13 – Синтаксична діаграма оголошення перемінної

Нижче приведені кілька прикладів оголошення перемінних.

Приклад 3.1.9 – Фрагмент програмного коду з оголошенням перемінних

```
var
```

A: integer;
B: boolean;
C: **arrey**[1..10] **of** extended;
D, E: **file**;

Для перемінних *область видимості* відіграє важливу роль. По суті, область видимості обмежується програмним блоком, у якому з'являється перемінна. Однак, якщо усередині блоку вкладений ще один блок, перемінна може з'являтися заново. Значення перемінної в зовнішньому блоці не міняється, оскільки воно відноситься до іншого розділові пам'яті. Усередині області видимості ім'я перемінної повинне бути унікальним.

Базові типи в Object Pascal. Як уже згадувалося, від типу перемінної залежить розмір резервируемой пам'яті. Для таких базових типів Object Pascal, як целочисленный (integer), дійсний (real), булев (boolean) і символьний (char) мається кілька варіантів типів, різниця між якими полягає насамперед в області припустимих значень.

Целочисленные типи Object Pascal. Діапазони значень для целочисленных типів представлені в табл. 3.1.1.

Таблиця 3.1.1 – Целочисленные типи Object Pascal

Тип	*Область значень*	*Фізичний формат*
Integer	Від –2147483648 до 2147483647	32 розряду, зі знаком
ShortInt	Від –128 до 127	8 розрядів, зі знаком
SmallInt	Від –32768 до 32767	16 розрядів, зі знаком
LongInt	Від –2147483648 до 2147483647	32 розряду, зі знаком
Byte	Від 0 до 255	8 розрядів, без знака
Word	Від 0 до 65535	16 розрядів, без знака

У стовпці з ім'ям “Тип” приведені ідентифікатори цілих типів. У стовпці “Фізичний формат” зазначений розмір пам'яті, виділюваної для даного типу. Крім того, у таблиці зазначено, яким образом інтерпретується перший розряд. Так, для типу longint виділяється 4 байти (32 розряду). Перший розряд першого байта є знаковим розрядом. Для запису значення використовуються тільки 31 розряд, що залишилися. Припустимі значення представлені в стовпці “Область значень”.

Дійсні типи Object Pascal. Наступна таблиця містить варіанти дійсних типів Object Pascal.

Таблиця 3.1.2 – Дійсні типи Object Pascal

Тип	*Область значень*	*Точність*	*Розмір у байтах*
Real	Від $2{,}9*10^{-39}$ до $1{,}7*10^{38}$	11-12 розрядів	6
Single	Від $1{,}5*10^{-45}$ до $3{,}4*10^{38}$	7-8 розрядів	4
Double	Від $5{,}0*10^{-324}$ до $1{,}7*10^{308}$	15-16 розрядів	8
Extended	До $3{,}4*10^{-4932}$ до $1{,}1*10^{4932}$	19-20 розрядів	10
Comp	Від $-2^{63}+1$ до $2^{63}-1$	19-20 розрядів	8

Значення стовпців “Тип” і “Область значень” відповідають попередній таблиці. У стовпці “розмір у байтах” указується загальний розмір пам'яті, виділюваної під даний тип. Даний розділ пам'яті включає області для збереження мантиси й експонента. Для збереження експонента немає необхідності в повному байті, цілком достатньо наявності декількох розрядів. Тип comp є виключенням. Усі 8 байтів використовуються в цьому випадку для мантиси.

Булеви типи Object Pascal. Булевы типи також мають кілька варіантів.

Таблиця 3.1.3 – Булеви типи Object Pascal

Тип	*Розмір*
Boolean	1 байт
ByteBool	1 байт
WordBool	2 байти (одне слово)
LongBool	4 байти (два слова)

Булеви типи є також *порядковими типами*. Останнє означає, що з перемінними даного типу можуть використовуватися оператори < і >. Для значень типу Boolean припустимі наступні вираження:

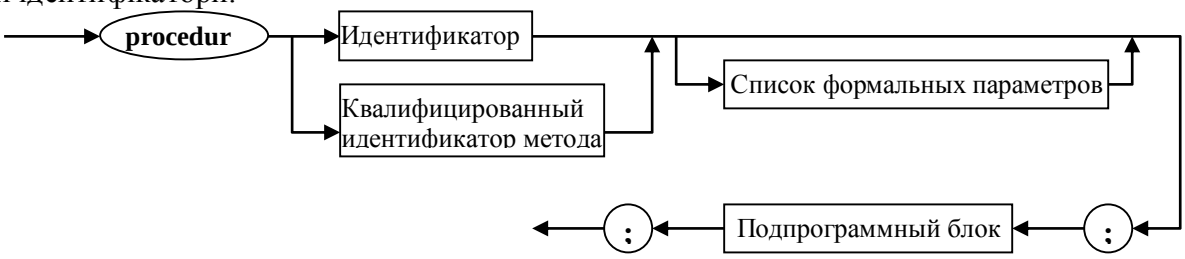
- False < True;
- Ord(False)=0;
- Ord(True)=1;
- Succ(False)=True;
- Pred(True)=False.

Завдяки тому, що тип Boolean займає в пам'яті один байт, він має перевагу в порівнянні з іншими булевыми типами. Всі інші варіанти призначені в першу чергу для забезпечення сумісності з іншими мовами програмування і Windows. Зверніть увагу на принципове розходження між типом Boolean і іншими типами. Для типу Boolean значення True має чисельне значення 1. по розуміннях сумісності з Visual Basic значення True типів даних ByteBool, WordBool і LongBool Delphi 4, у порівнянні з більш ранніми версіями, було змінено на чисельне значення, рівне –1. Значенню False відповідає, як і колись, значення 0.

3.1.2.3. Процедури та функції

Процедури. Нижче приведена синтаксична діаграма оголошення процедури:

Оголошення процедури починається з зарезервованого слова procedure. Потім випливає ім'я процедури (ідентифікатор), що повинний бути унікальним. Для оголошення методів, що, по суті, є звичайними процедурами, необхідно провести *квалифицирование* ідентифікатора, тобто перед ідентифікатором процедури або функції вказується тип класу, якому належить даний метод. Тому використовується поняття *кваліфікований ідентифікатор* методу. Якщо процедура являє собою просту підпрограму, то її ім'я повинне бути унікальним у межах модуля. Якщо мова йде про метод, то ім'я повинне бути унікальним тільки в межах класу. За рахунок квалифицирования можна домогтися того, що процедури і функції в програмі будуть мати унікальні ідентифікатори.



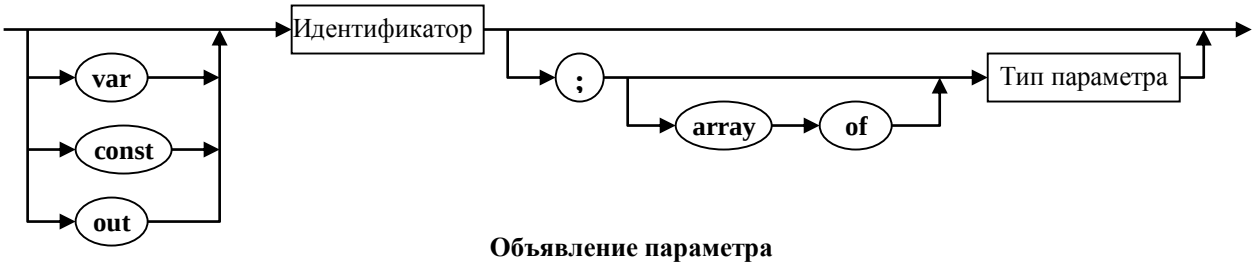
Объявление процедуры

Малюнок 3.1.14 – Синтаксична діаграма оголошення процедури

Після імені випливає, якщо необхідно, список параметрів, що передають необхідні для процедури або функції значення. Альтернативою використання списку параметрів процедури є застосування глобальних перемінних, однак цього робити не рекомендується.

На синтаксичній діаграмі оголошення процедури представлений так називаний *список формальних параметрів* (далі список параметрів). Мова йде про список параметрів, що забезпечують взаємодію між кодом реалізації процедури (*тілом процедури*) і зухвалим блоком програми, у якому відбувається виклик даної процедури. Синтаксична діаграма списку параметрів процедур і функцій представлена на малюнку 3.1.15.

Список формальных параметров



Объявление параметра

Малюнок 3.1.15 – Синтаксична діаграма списку параметрів процедур

У списку параметрів указуються не тільки імена перемінних, службовців для передачі даних, але і те, яким образом вони функціонують. При цьому розрізняють п'ять видів параметрів:

- параметр-значення;
- параметр-перемінна, називаний також *параметр-покажчик*;
- параметр-константа;
- параметр висновку;
- нетипізований параметр.

Параметр-значення. Передача параметра-значення являє собою найбільш простий спосіб передачі параметрів у процедури і функції. Це означає, що в процесі оголошення формального параметра вказуються його ідентифікатор і тип, наприклад:

procedure X (y: integer);

Параметр y у даному випадку являє собою параметр-значення. При виклику процедури X значення якої або перемінної целочисленного типу віддається в тіло даної процедури через параметр y. Для збереження переданого значення резервується область пам'яті відповідно до типу даних. Зміни значення параметра у усередині тіла процедури не впливають на значення перемінної з зухвалого блоку. Тому при виклику даної процедури для передачі параметрів можна використовувати наступне вираження:

X(10*z+70-a);

При цьому тип поточного значення або вираження повинний бути сполучимо з формальним параметром.

Перевага даного способу передачі параметрів складається в повному захисті значення перемінної від змін усередині тіла процедури. Недоліком є неможливість висновку отриманих даних через параметр-значення.

Параметр-перемінна. Передача у виді параметр-перемінної має прямо протилежні перевагу і недолік. Оголошення параметр-перемінної здійснюється за допомогою ключового слова var, розташованого перед параметром. Область дії даної директиви простирається до наступної крапки з коми. Приклад подібного роду оголошення виглядає в такий спосіб:

procedure A(**var** b, c: integer; d: char);

Параметри b і c являють собою параметрі-перемінні, параметр d – параметр-значення. При передачі параметр-перемінних не копіюються їхні значення, а передаються адреси фактичних параметрів. От чому параметр-перемінна називається також *параметр-покажчик*. Зміни в

значенні параметр-перемінної, зроблені в межах тіла процедури, відбивають на значеннях перемінних, що утримуються в зухвалому блоці. Тому параметр-перемінна звичайно застосовується для висновку отриманих даних.

Параметр-константа. Третій вид передачі, з використанням константи, відноситься до параметр-перемінної, що має усередині тіла процедури статус *read-only*. Це означає, що значення перемінної усередині тіла процедури змінюватися не може. Цей вид передачі привабливий тим, що копіювання значення параметра в процедурі не виконується, однак параметр усе-таки залишається цілком захищеним. Оголошення параметрів-констант забезпечується за допомогою вказівки перед параметром ключового слова const. Приклад оголошення виглядає в такий спосіб:

procedure X(**const** y: **string**; **var** z: boolean);

Параметр висновку. Застосовується в тому випадку, якщо процедура повинна повертати дані.

При декларуванні методу параметри висновку з'являються за допомогою вказівки перед параметром ключового слова out. Приклад оголошення виглядай у такий спосіб:

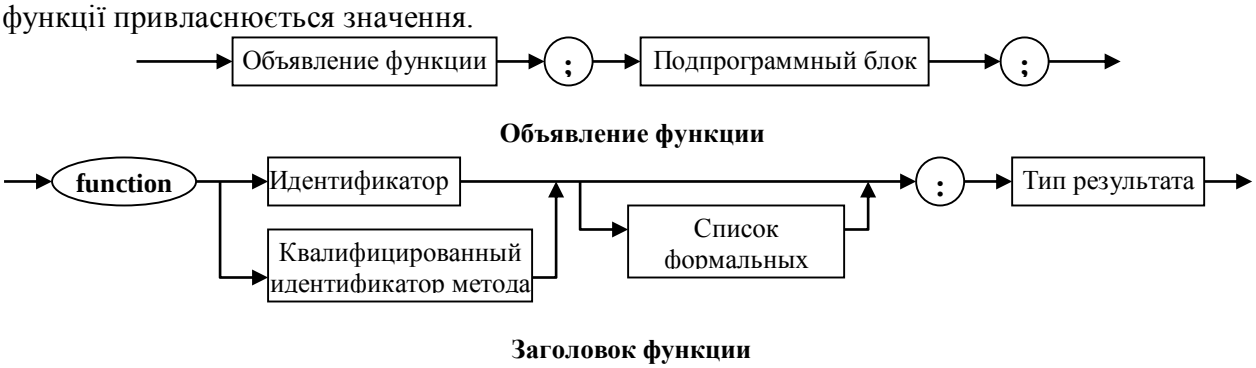
procedure X(**out** a: WideString; **var** y: integer);

Нетипізований параметр. Використовує формальний параметр, для якого тип не був визначений. Такі параметри називають *нетипізованими*. Нетипізовані параметри випливають за ключовими словами var, const або out, однак не містять оголошень типів.

Функції. Існує ще один спосіб передачі значень із процедури в зухвалий програмний блок: з використанням значення функції. Власне кажучи, функція являє собою процедуру, однак функція повертає значення визначеного типу, тому виклик функцій можна здійснювати у вираженнях.

Ідентифікатором значення, що повертається, функції може бути будь-який стандартний (за винятком типу file) визначений розроблювачем тип.

Тіло функції повинне містити щонайменше одну команду, за допомогою якої імені функції привласнюється значення.



Малюнок 3.1.16 – Синтаксична діаграма оголошення функції в Object Pascal

Кожна функція має призначену неявним образом локальну перемінну Result. Вона має той же тип, що і значення, що повертається. Якщо ви привласнюєте даної перемінної значення, то ця дія має той же ефект, що і присвоювання значення імені (ідентифікаторові) функції усередині тіла даної функції. Проте мається невелике розходження у використанні усередині тіла функції перемінної Result і імені функції. Використання перемінної Result у правій частині виражень у межах тіла функції дозволяє одержати поточне значення даної перемінної без виконання рекурсивного виклику даної функції. Наступний приклад роз'ясняє застосування заданої неявним образом перемінної:

Приклад 3.1.10 – Програмний модуль з використанням функції

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
function sintest(var x: extended): extended;
begin
  Result:=sin(x);
  x:=Result;
end;
var x: extended;
begin
  x:=0.5;
  Writeln('sin(0.5):', sin(x));
  Writeln('sintest(0.5):', sintest(x));
  Writeln('sintest.result:', x);
end.
```

Для виконання приклада варто відкомпілювати додаток у якості консольного (**Project/Options/Linker/Generate console application**). Результат виглядає в такий спосіб:

```
sin(0.5): 4.79425538604203E-0001
sintest(0.5): 4.79425538604203E-0001
sintest.result: 4.79425538604203E-0001
```

3.1.2.4. Область видимості змінних

Область видимості перемінної являє собою фрагмент програмного коду, у якому перемінна вважається відомою. Поняття області видимості відноситься насамперед до ідентифікатора перемінної. Він повинний бути унікальним на тім рівні програмного блоку, на якому з'являлася перемінна. У залежності від рівня, на якому з'являються перемінні, розрізняють наступні види перемінних:

- локальні перемінні;
- глобальні перемінні.

Область видимості глобальних перемінних визначається або на рівні програми, або на рівні модуля. Глобальні перемінні відомі в межах усього подібного коду додатка, однак можуть бути повторно визначені у внутрішньому програмному блоці.

Приклад 3.1.11 – Використання глобальної перемінної

```
program Scope
var
  A: integer; //глобальна перемінна A
procedure Set;
var
  A: integer; //локальна перемінна A
begin
  A:=4;
end; //кінець області видимості локальної перемінної A
begin
  A:=3; //присвоєння значення глобальної перемінної
  Set; //виклик процедури Set
```

Writeln(A); //*значення A дорівнює 3, не 4!*
end.

Якщо ідентифікатор глобальної перемінної використовується для визначення локальної перемінної, то усередині блоку, де це відбувається, локальна перемінна є дійсною. Крім того, вона має пріоритет перед глобальною перемінною.

3.1.2.5. Управління ходом виконання програми

Оператор може бути простим або структурованим. Структуровані оператори дають можливість керувати послідовністю виконання дій. У цьому випадку говорять про *керування ходом виконання програми*.

У теорії структурованого програмування розрізняють три види структур:

- конкатенації;
- умовні оператори;
- цикли.

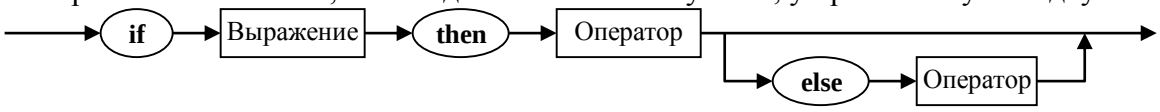
Конкатенація означає, що кілька команд ставляться друг за другом і в цій послідовності виконуються. Таку послідовність операторів можна розглядати як блок. По завершенні відпрацьовування останнього оператора блок вважається виконаним.

Блок, що містить оператори, має, таким чином, вхід і вихід. Якщо в блоці де-небудь виявляється помилка, то причина її виникнення відразу стає ясною. Це спрощує процес налагодження.

У випадку умовних операторів мова знову йде про блок, що містить кілька вкладених блоків, з яких виконується тільки один. Вибір виконуваного блоку визначається з умови.

Умовні оператори. Існує два види умовних операторів: if і case. Хоча оператор case може застосовуватися в будь-якому випадку, усе-таки для вибору одного з двох виконуваних блоків частіше використовується оператор if. Оператор case застосовується при здійсненні вибору з більш ніж двох блоків.

Оператор if. У найпростішому випадку існує тільки один вкладений блок. Якщо задана умова повертає значення True, то вкладений блок виконується, у противному випадку – немає.



Оператор if

Малюнок 3.1.17 – Синтаксична діаграма оператора if у Object Pascal

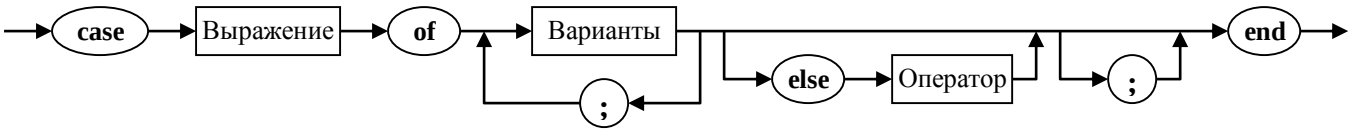
У конструкції “Якщо зазначена умова вірна, то виконується блок 1, у противному випадку блок 2” дві структури конкатенації вставлені в структуру вибору, і одна з них виконується в будь-якому випадку.

У Object Pascal програмні блоки позначаються за допомогою зарезервованих слів begin і end. Зверніть увагу на те, що крапкою з коми завершуються тільки оператори, але не блоки.

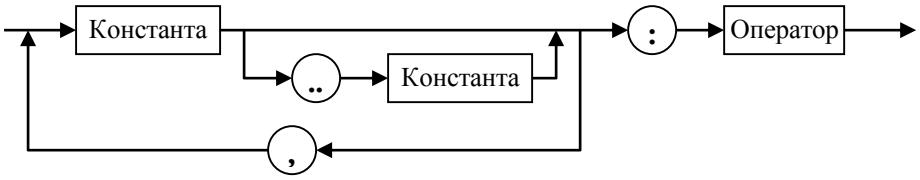
Приклад 3.1.12 – Оператор if

if A=True **then**
begin
…//*Блок 1*
end
else begin
…//*Блок 2*
end;

Оператор case. Якщо необхідно вибрати з декількох вкладених блоків і їхня кількість більше двох, такі програмні конструкції створюються за допомогою оператора case.



Оператор case



Вариант

Малюнок 3.1.18 – Синтаксична діаграма оператора case у Object Pascal

Приведене на мал. 3.1.18 *вираження* містить умова виконання потрібного блоку. Блок вибирається в тому випадку, якщо значення вираження ідентично значенню мітки або знаходиться в межах діапазону мітки даного блоку. Ця мітка являє собою просту константу, множину констант або область, границі якого визначені за допомогою констант. Якщо в блоці є галузь else, то вона буде виконуватися тільки в тому випадку, якщо *вираження* повертає значення, для якого не передбачено ніякого варіанта. Застосування веиви else рекомендується в тому випадку, якщо ви не упевнені, що оьдумали усі варіанти, до яких може привести вираження.

Області значень міток не можуть перекривати один одного. Блоки в операторі case повинні бути упорядковані по зростанню щодо значень мітки. Інша послідовність допускається, однак швидкість виконання програми в цьому випадку падає.

Приклад 3.1.13 – Приклади застосування оператора case

```
case I of
  1..5: Caption := 'Low';
  6..9: Caption := 'High';
  0, 10..99: Caption := 'Out of range';
else
  Caption := '';
end;
case MyColor of
  Red: X := 1;
  Green: X := 2;
  Blue: X := 3;
  Yellow, Orange, Black: X := 0;
end;
case Selection of
  Done: Form1.Close;
  Compute: CalculateTotal(UnitCost, Quantity);
else
  Beep;
end;
```

Оператори циклу. Структура циклу реалізована таким чином, що визначений програмний блок може виконуватися один раз, кілька разів або взагалі не виконуватися. Тому можна говорити про умову, що визначає повторне виконання блоку. Унаслідок цього дана умова називається також *умовою виходу з циклу*. У залежності від ситуації ви можете вибрати одну з наступних конструкцій циклу:

- оператор repeat;
- оператор while;
- оператор for.

Вибір конструкції залежить від характеру циклічного процесу. Якщо відомі початковий і кінцевий стани, то можна вибрати оператор for. Якщо ж існує впевненість у тім, що програмний блок повинний бути виконаний щонайменше один раз і може бути встановлена необхідність повторного виконання даного блоку, то варто використовувати оператор repeat (оператор циклу з умовою завершення). Якщо невідомо, чи належний виконуватися програмний блок узагалі, рекомендується використовувати оператор while (оператор циклу з умовою продовження).

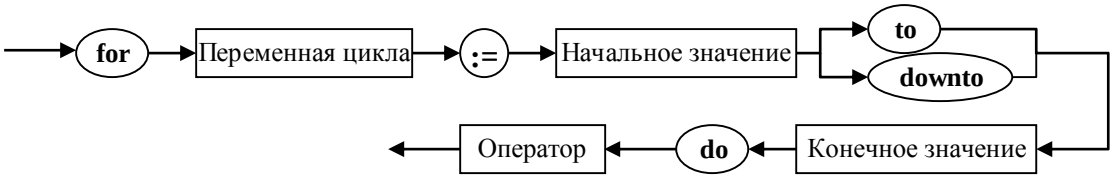
Оператор for. На мал. 3.1.19 *операторові* відповідає вкладений блок, що виконується тепер у залежності від *перемінної керування*, називаної також *перемінної циклу*. Характерною властивістю оператора for є те, що, на відміну від інших двох конструкцій циклу. Керування виходом їхнього циклу не здійснюється за допомогою вкладеного блоку. Причина полягає в тім, що початковий і кінцевий стани відомі і тому ясно, коли варто здійснити вихід їхнього циклу.

Перемінна циклу повинна бути порядкового типу. Типи початкового і кінцевого значень повинні бути сумісні по присвоюванню з типом перемінної циклу. Значення перемінної циклу після виконання останнього повинне розглядатися як невизначене. Нижче приводиться приклад використання оператора for, що пояснює концепцію вкладених структур.

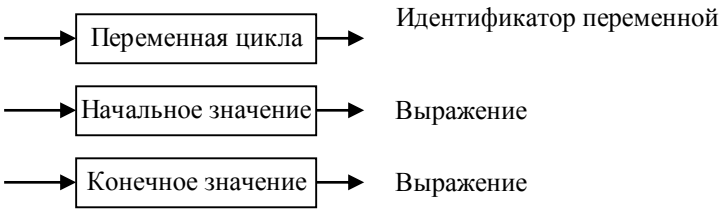
Приклад 3.1.14 - Приклад застосування оператора for

```
type
  TColor = (cRed, cOrange, cYellow, cGreen, cBlue, cViolet);
var
  Color: TColor;
  C: string;
begin
  for Color := cRed to cViolet do
    begin
      case Color of
        cRed: s := ‘Red’;
        cOrange: s := ‘Orange;
        cYelow: s := ‘Yelow;
        cGreen: s := ‘Green;
        cBlue: s := ‘Blue;
        cViolet: s := ‘Violet;
      else
        s := ‘Black’;
      end; //else
    end; //for
end.
```

З мал. 3.1.19 видно, що існує також можливість зворотного відліку, здійснювана за допомогою директиви downto.



Оператор for



Малюнок 3.1.19 – Синтаксична діаграма оператора for у Object Pascal

Ще одне зауваження, що стосуються області видимості перемінної циклу. Ця перемінна повинна бути локальної (хоча і необов'язково), тобто бути визначеної в тім блоці, де застосовується оператор for. Таким чином, ви не повинні повідомляти глобальну перемінну і на рівні модуля і використовувати неї в різних процедурах як перемінний цикл.

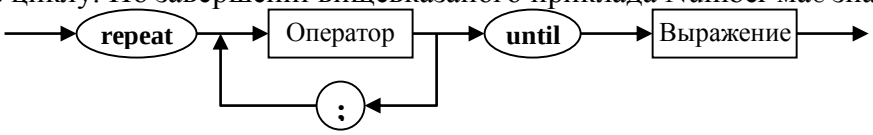
Оператор repeat. Під оператором на мал. 3.1.20 мається на увазі програмний блок, що, можливо, буде виконуватися заново. Особливий інтерес у даній конструкції представляє умова виходу з циклу. Цикл повинний коли-небудь закінчитися, а саме після того, як вираження прийме значення True. Вираження аналізується, як тільки виконувана програма досягне условия until. Якщо вираження повинне повертати значення True, то воно повиннео залежати від результатів, що розраховуються в блоці *оператор*. Якщо дана ситуація не настає, то виконання циклу продовжується. Зверніть увагу на те, що вираження формулюється таким чином, що воно повиннео давати в підсумку значення True. Розглянемо наступний приклад:

```
Number := 9;
repeat
  Number := Number – 2;
until Number = 0;
```

Тут цикл буде виконуватися нескінченно довго, оскільки Number ніколи не прийме значення, рівне 0. рекомендується умова виходу з циклу задавати за допомогою понять “більше” або “менше”:

```
Number := 9;
repeat
  Number := Number – 2;
until Number < 0;
```

Зверніть увагу на те, що обчислення у внутрішньому блоці оператора repeat … until привели до виходу з циклу. По завершенні вищевказаного приклада Number має значення –1.

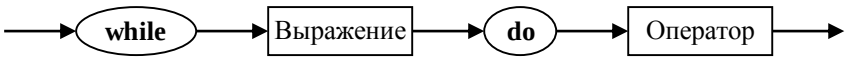


Оператор repeat

Малюнок 3.1.20 – Синтаксична діаграма оператора repeat у Object Pascal

Якщо є необхідність виконання циклічного процесу в приведеному прикладі доти, поки Number має позитивне значення, то більш підходящим є оператор while.

Оператор while. Синтаксична діаграма оператора while виглядає в такий спосіб:



Оператор while

Малюнок 3.1.21 – Синтаксична діаграма оператора while у Object Pascal

Розходження між оператором while і оператором repeat полягають у наступному.

- У конструкції while спочатку аналізується умова виходу з циклу, після цього виконується сам оператор. У конструкції repeat усі відбувається навпаки.
- Число циклів у конструкції while може бути 0 або більше, у конструкції repeat – 1 або більше.
- У конструкції while ітерація проводиться доти, поки виконується умова, у конструкції repeat – тільки доти, поки воно не буде виконано.

При написанні вихідного тексту варто враховувати, що у випадку застосування конструкції while спочатку повинне бути инициализировано умова виходу з циклу, а вуж потім виконаний сам оператор. Іноді це викликає труднощі і приводить до дублювання операторів. Уникнути подібної ситуації дозволяє використання процедури Break, що передає керування першому операторові, що випливає за оператором циклу. При виборі конструкції циклу варто керуватися характером циклічного процесу.

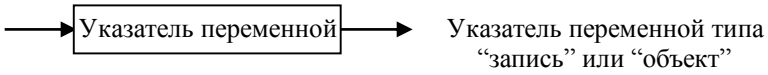
Повернемося до попереднього приклада, але скористаємося вже конструкцією while:

```
Number := 9;
while Number > 0 do
  Number := Number – 2;
```

У цьому випадку перемінна Number після закінчення циклу має значення –1 і цикл, на відміну від приклада, де використовувався оператор repeat, виконується тільки при позитивному значенні Number.

Оператор квалифицирования. Варто згадати ще один структурований оператор, використовуваний для спрощення квалифицирования перемінних у Object Pascal. Це оператор with, що дозволяє кваліфікувати перемінну в заданому блоці. Він має наступний синтаксис:

Оператор with



Малюнок 3.1.22 – Синтаксична діаграма оператора with у Object Pascal

Нижче приведений приклад використання оператора квалифицирования.

Приклад 3.1.15 - Приклад застосування оператора with

```
type
TDate = record
  Day: Integer;
  Month: Integer;
  Year: Integer;
end;

var
  OrderDate: TDate;
begin
…
with OrderDate do
  begin
```

```
    if Month = 12 then
      begin
        Month := 1;
        Year := Year + 1;
      end
      else
        Month := Month + 1;
  end; //with
…
end...
```

Без використання оператора with приклад 3.1.15 мав би наступний вид:

```
type
TDate = record
  Day: Integer;
  Month: Integer;
  Year: Integer;
end;

var
  OrderDate: TDate;
begin
…
  if OrderDate.Month = 12 then
    begin
      OrderDate.Month := 1;
      OrderDate.Year := OrderDate.Year + 1;
    end
    else
      OrderDate.Month := OrderDate.Month + 1;
…
end...
```

3.1.3. Стандартні процедури та функції Object Pascal

Через наявність великої кількості стандартних процедур і функцій Object Pascal пошук визначеної підпрограми, що виконує визначену функцію, являє собою непросту задачу. У зв'язку з цим стандартні процедури і функції відповідно до їхнього функціонального призначення були об'єднані і представлені у виді таблиць у Додатку А.

Література [2-3].

Тема 3.2. Програмування друку документів

3.2.1. Компоненти

Нижче перераховані компоненти Delphi, що допоможуть вам в організації печатки даних.

- ***TPrintDialog***. Цей компонент призначений для відкриття стандартного діалогового вікна Windows Print (Печатка), у якому можна вказати, які сторінки друкувати, а також одержати доступ до опцій настроювання принтера.
  Категорія палітри: Dialogs.

- ***TPrinterSetupDialog***. Цей компонент дозволяє відобразити діалогові вікна настроювання принтера з опціями, обумовленими встановленими драйверами. Таку можливість потрібно надати користувачам обов'язково, щоб вони були вільні у виборі принтера і вихідного порту і настроюванню параметрів печатки.
  Категорія палітри: Dialogs.
- ***TPrinter***. На палітрі його немає — він визначений як клас у модулі Printers. Серед інших членів цього класу у вашому розпорядженні властивість Canvas, що дозволяє в режимі WYSIWYG відобразити на екрані майбутній друкований аркуш з текстом і графікою.
  Категорія палітри: відсутній.

3.2.2. Друк текстового документові

Друкувати текст додатка Delphi можуть двома способами. У цьому розділі буде расказано про найпростіший з них — печатки у вихідний файл Text за допомогою процедур Write і Writeln. Text — це не дисковий файл; він є файлом лише в тім змісті, що задає приймач, у який направляється потік даних. Така технологія найкраще підходить для печатки тексту без графіки.

Технологія печатки за допомогою процедур Write і Writeln дозволяє використовувати шрифти TrueType і інші графічні шрифти, включаючи їхній похиле і напівжирне накреслення. А от власний набір символів принтера ця технологія використовувати не дозволяє, хіба що ви установите драйвер загального застосування для печатки тільки тексту. Деякі новітні принтери перетворять шрифти TrueType у власні внутрішні шрифти, але ця процедура схована від користувача.

3.2.2.1. Модуль Printers

Оголошення модуля Printers потрібно обов'язково включити в директиву uses будь-якого іншого модуля, що використовує функції печатки. Якщо зробити це для тільки що створеного модуля, вийти повинн наступне (доданий текст виділений напівжирним шрифтом):

uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, **Printers**;

Цей модуль надає програмі об'єкт Printer класу TPrinter, через який проходять усі дані, що посилаються на принтер, і який, крім того, надає таку корисну інформацію, як висота і ширина сторінки в пикселях.

Зверніть увагу на різницю в найменуваннях: об'єкт називається Printer (принтер), а модуль - Printers (принтери). Зштовхнувши з проблемами, перевірте, чи не переплутали ви ці два слова.

Для початку виконаємо простий приклад, що демонструє використання модуля Printers і визначеного в ньому об'єкта Printer. Приєднаєте принтер, уключите його і додержуйтеся вказівок.

Відкрийте новий додаток. Перейдіть у вікно редактора коду, у самому верху модуля знайдіть розділ uses і включите в нього оголошення модуля Printers.

Помістите у форму об'єкт компонента TButton (категорія палітри Standard) і, двічі клацнувши на цьому об'єкті, створіть процедуру обробки його події OnClick.

Скопіюйте в процедуру Button1Click програмний код із Приклада 5.9.1. Цей код реалізує найпростішу технологію печатки.

Скомпілюйте і запустите програму, натиснувши клавішу <F9>. Коли відкриється вікно програми, клацніть на створеній вами кнопці. Принтер повинний надрукувати в самому верху сторінки рядок Hello printer!. (Надрукована вона може бути дуже дрібно — це залежить від того, який саме у вас принтер.)

Приклад 3.2.1 - Елементарна технологія печатки тексту в процедурі обробки події OnClick командної кнопки

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  FPrn: System.Text;
begin
  AssignPrn(FPrn);
  Rewrite(FPrn);
  try
    Writeln(FPrn, 'Hello printer!');
  finally
    CloseFile(FPrn)
  end;
end;
```

Ця процедура насамперед створює вихідний файл, у который буде спрямований весь текст. Власне кажучи, мова йде про буфер в оперативній пам'яті — файлової перемінної FPrn. Щоб перенаправляти записуваний у цю перемінну вихідний потік Pascal на принтер і “відкрити” її для одержання даних, використовуються наступні дві команди:

```
AssignPrn(FPrn); {Направляємо вихідний потік на принтер.}
Rewrite(FPrn};   {Відкриваємо файл.}
```

Процедура AssignPrn, що зв'язує вихідний файл із принтером, визначена в модулі Printers. Що стосується процедури Rewrite, те це стандартна процедура мови Pascal, що відкриває зазначений текстовий файл.

Тепер звернемося до процедури Writeln, щоб надрукувати наш рядок. Процедура Writeln друкує зазначений рядок і символи перекладу рядка. Щоб уникнути можливих збоїв, помістимо команди печатки в блок try:

```
try
  Writeln(FPrn, 'Plain text is plain and simple!');
  { ... Сюди помістите операторы печатки.}
```

Залишився останній оператор, що сигналізує про те, що завдання для принтера кінчено, і передає йому команду перекладу сторінки. Помістивши цю команду в блок finally, ви будете упевнені, що вона виконається навіть у випадку помилок печатки:

```
finally
  CloseFile(FPrn);
end;
```

Зверніть увагу на те, що оператор try - finally обов'язково завершується ключовим словом end, що є частиною його структури.

Процедура Writeln може використовуватися не тільки для запису у вихідний потік єдиного рядка. З її допомогою можна друкувати відразу кілька значень перемінних простих типів, зокрема і числових:

```
Writeln(FPrn, 'Number of components = ', ComponentCount);
```

Якщо ж потрібно надрукувати в одному рядку кілька значень підряд, варто звернутися до процедури Write. Наприклад, що предыдет оператор еквівалентний трьом наступної:

```
Write(FPrn, 'Number of components = ');
Write(FPrn, ComponentCount);
Writeln(FPrn); {Переклад рядка.}
```

А от спроба передати вихідну файлову перемінну, зв'язану з принтером за допомогою процедури AssignPrn, процедурі Read або Readln приведе до помилки часу виконання. Читати дані з такого вихідного пристрою не можна.

3.2.2.2. Коди керування та шрифти

Для керування принтером через процедури Write і Writeln можна використовувати чотири керуючі коду, перерахованих у табл. 3.2.1.

Таблиця 3.2.1 – Чотири керуючі коду процедури Write і Writeln

Керуючий код	***Команда***
#9	Табуляція
#10	Новий рядок
#13	Повернення каретки
^L	Нова сторінка

Ці ASCII-коди можна або включати в рядок, або друкувати за допомогою процедури Write, як у наступному прикладі:

Write(FPrn, #9); *{Табуляція.}*

Write(FPrn, #13); *{Очистити вихідний буфер.}*

Write(FPrn, #10); *{Очистити буфер і почати новий рядок.}*

Write(FPrn, ^L); *{Очистити буфер і почати нову сторінку.}*

Ширина табуляції в процедур Write і Writeln у вісьмох разів більше середньої ширини символу поточного шрифту. Тому, якщо шрифт *пропорційний*, вирівняти стовпці тексту за допомогою символу табуляції при печатці його процедурами Write і Writeln не удасться — хіба що вони складаються тільки з цифр. З цифрами це можливо тому, що в більшості пропорційних шрифтів усі цифри мають однакову ширину.

Об'єкт Printer має властивість Canvas, що, серед інших різноманітних можливостей, дозволяє установити шрифт тексту, що друкується. За замовчуванням це шрифт System розміром 10 пунктів. Якщо потрібний інший, привласніть його ім'я і розмір властивостям Font.Name і Font.Size об'єкта Printer.Canvas:

```
with Printer.Canvas do
begin
  Font.Name := 'Courier New';
  Font.Size := 12;
end;
```

Цей код можна додати в оброблювач щиглика на кнопці Button1Click з попереднього приклада. Помістите його між викликом Rewrite і ключовим словом try. Потім натисніть <F9> для компіляції і запуску програми і клацніть на кнопці печатки. Тепер у вас повинний вийти той же текст, надрукований *моноширинным* шрифтом розміром 12 пунктів.

Використовуючи описану технологію печатки тексту, обов'язково викликайте процедури AssignPrn і Rewrite перед установкою шрифту.

Обраний шрифт визначає висоту і ширину символів, впливаючи в такий спосіб і на кількість рядків на сторінці (до цього ми ще повернемося). Можна користуватися будь-як установленим шрифтом, але рекомендується вибирати шрифти TrueType: вони звичайно краще підходять для печатки, оскільки Windows GDI надає символи для будь-якого графічного драйвера принтера, навіть якщо він не підтримує TrueType безпосередньо.

Освоїти технологію печатки за допомогою процедур Write і Writeln обов'язково прийдеться в тому випадку, якщо ви маєте справу зі старими текстовими принтерами. Для роботи з таким принтером використовуйте панель керування Windows і установите драйвер Generic/Text Only (Загальний/тільки текст). Гарантії, що це буде працювати, ні, тому потрібно попередити користувачів програми, що для печатки в Windows їм необхідний графічний принтер.

3.2.2.3. Відомості про принтер

На мал. 3.2.1 показане вікно додатка, що друкує зведення про можливості принтера і поточний шрифт. Крім вся іншого, програма демонструє, як визначити кількість рядків на сторінці, що в деяких випадках буває дуже корисно.

Малюнок 3.2.1 - Якщо клацнути на великій кнопці додатка, буде роздрукований звіт про здатність принтера, що дозволяє, установленому драйвері, кількості рядків на сторінці і середній кількості символів у рядку

Після запуску даного додатка виводиться наступна інформація:

Device = HP DeskJet 720C Series on LPT1:
Font = Courier New
Font Size = 12 points
PageHeight = 3150 pixels
PageWidth = 2400 pixels
Extent.Cx = 6720 pixels
Extent.Cy = 53 pixels
Lines per page = 56
Chars per line = 80

У прикладі 3.2.1 приведений вихідний код програми.

Приклад 3.2.1 – Печатка зведень про принтер

```
unit Main;
interface
uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs, Printers, StdCtrls, Buttons;
type
  TMainForm = class (TForm)
  PrintButton: TButton;
  CloseBitBtn: TBitBtn;
  procedure PrintButtonClick (Sender : TObject);
  private
    {Оголошення закритих (private) членів.}
   public
    {Оголошення загальнодоступних (public) членів.}
end;
var
  MainForm: TMainForm;
implementation
{$R *.DFM}
procedure TMainForm.PrintButtonClick(Sender: TObject);
var
  FPrn: System.Text;
  Extent: TSize;
  Metrics: TTextMetric;
  I, LinesPerPage, CharsPerLine, AverageWidth: Integer;
  S : String;
begin
  AssignPrn(FPrn);
```

```
Rewrite (FPrn);
with Printer.Canvas do
  begin
    Font.Name := 'Courier New';
    Font.Size := 12;
  end;
  {Заповнюємо текстовий рядок символами ASCII з кодами від 32 до 255.}
try
SetLength(S, 224);
for I := 32 to 255 do {Заповнюємо рядок тестовими символами}
S[I - 31] := Chr(I);
with Printer.Canvas do
  begin {Визначаємо кількість рядків на сторінці.}
    GetTextExtentPoint(Handle, @S[1], Length(S), Extent);
    LinesPerPage := PageHeight div (Extent.Cy + 2);
    if PageHeight mod Extent.Cy <> 0 then
      Dec (LinesPerPage);
    {Визначаємо середня кількість символів у рядку.}
    GetTextMetrics(Handle, Metrics);
    AverageWidth := Metrics.tmAveCharWidth;
    CharsPerLine := PageWidth div AverageWidth;
    {Друкуємо звіт}
    Writeln(FPrn, 'Device = ', Printers[PrinterIndex]);
    Writeln(FPrn, 'Font = ', Font.Name);
    Writeln(FPrn, 'FontSize = ', Font.Size, ' points');
    Writeln(FPrn, 'PageHeight = ', PageHeight, ' pixels');
    Writeln(FPrn, 'PageWidth = ', PageWidth, ' pixels');
    Writeln(FPrn, 'Extent.Cx = ', Extent.Cx, ' pixels');
    Writeln(FPrn, 'Extent.Cy = ', Extent.Cy, ' pixels');
    Writeln(FPrn, 'Lines per page = ', LinesPerPage);
    Writeln(FPrn, 'Chars per line = ', CharsPerLine);
  end;
finally
  CloseFile(FPrn);
end;
end;
end.
```

Підготовлений тестовий рядок програма не друкує, вона використовує її тільки для формування звіту. Значення Extent.Cx дорівнює довжині цього рядка в пикселях при обраному шрифті (Courier New розміром 12 пунктів). Значення Extent.Cy дорівнює висоті цього рядка в пикселях. Кількість рядків на сторінці обчислено виходячи з припущення, що всі рядки будуть друкуватися тим самим шрифтом. Кількість символів у рядку для пропорційного шрифту обчислюється приблизно. Ці значення дозволяють визначити, скільки інформації поміститься на друкованій сторінці. Але, перш ніж їх обчислювати, обов'язково установите потрібний шрифт.

Щоб відобразити статистичні зведення для іншого шрифту, безпосередньо перед викликом GetTextExtentPoint помістите операторы, аналогічні наступної:

```
Font.Name := 'Arial';
Font.Size := 24;
```

Додаток демонструє, як викликати функції Windows API, яким потрібен дескриптор контексту пристрою принтера. Цей дескриптор є значенням властивості Handle об'єкта

Printer.Canvas; його і потрібно передати викликуваної функції. Наприклад, виклик функції GetTextExtentPoint може бути таким:

GetTextExtentPoint(Handle, @S[1], Length(S), Extent);

Як покажчик на рядок (PChar) функції передається адреса першого символу рядка S — @S[1]. Аналогічно виглядає і виклик іншої функції Windows API — GetTextMetrics:

GetTextMetrics(Handle, Metrics);

Демонстрируемая додатком технологія працює тільки при печатці чистого тексту за допомогою функцій Write і Writeln і можливості її досить обмежені. Використанні другої, графічної технології печатки, надає набагато більше можливостей при печатці.

3.2.2.4 Друк списків рядків

Щоб роздрукувати рядка об'єкта класу TStrings або TStringList, скористайтеся функцією Writeln, як описано в попередньому розділі. От приклад, що друкує вміст Memo - об'єкта. У перелік модулів директиви uses додайте модуль Printers і в оброблювач події OnClick командної кнопки додатка помістите наступний код:

```
var
  FPrn: TextFile;
  I: Integer;
begin
  AssignPrn(FPrn);
  Rewrite(FPrn);
  try
    for I := 0 to Memo1.Lines.Count - 1 do
      Writeln(FPrn, Memo1.Lines[I]);
  finally
    CloseFile(FPrn);
  end;
end;
```

Цей код можна використовувати для печатки текстового вмісту будь-якого об'єкта, представленого його властивістю типу TStrings. Прикладом може бути властивість Lines об'єкта компонента TMemo або властивість Items об'єкта компонента TListBox.

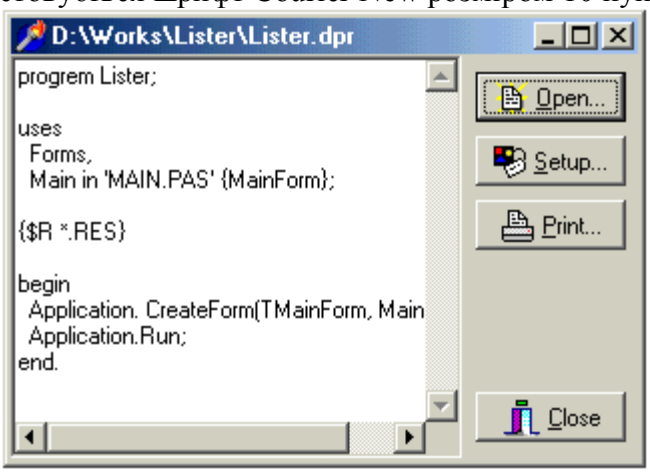
3.2.2.5. Діалогові вікна друку

Хоча користувальницький інтерфейс, наданий додатками для печатки, може бути кожним, найкраще дотримувати загальноприйнятих стандартів — помістити команди печатки в меню File. Пропоновані більшістю додатків команди перераховані нижче.

- File(Page Setup (Файл(Параметри сторінки). Ця команда необов'язкова. Вона відкриває діалогове вікно для настроювання колонтитулів, номерів сторінок, полів і іншого специфічних для додатка параметрів. Стандартного методу для реалізації цієї команди немає.
- File(Print Preview (Файл(Попередній перегляд). Ця команда теж необов'язкова. Вона дозволяє переглянути, як буде виглядати надрукована сторінка. Для її реалізації теж немає стандартного методу.
- File(Print Setup (Файл(Настроювання принтера). Дана команда відкриває діалогове вікно настроювання принтера. Звичайно в ньому є присутнім кнопка, що відкриває вікна з додатковими параметрами. Для реалізації розглянутої команди Delphi пропонує компонент PrinterSetupDialog.
- File(Print (Файл(Печатка). Ця команда відкриває стандартне діалогове вікно Windows, що дозволяє вказати, які потрібні сторінки і скільки необхідно копій. Кнопка ОК служить для початку печатки. З цього вікна можна відкрити і діалогове вікно настроювання принтера (того ж саме, для відкриття якого може служити окрема

команда File(Print Setup). Для реалізації цієї команди Delphi пропонує компонент PrinterDialog.

Перші дві команди необов'язкові. Дві інші стандартні, але можна обійтися тільки однієї з них, File(Print, оскільки до діалогового вікна настроювання драйвера можна одержати доступ з діалогового вікна Print. Програмний код додатка що демонструє як користуватися компонентами PrinterDialog і PrinterSetupDialog приведений у прикладі 5.9.2 (мал. 5.9.2). Цей додаток може роздрукувати будь-який текстовий файл (наприклад, .pas або . txt). Щоб показати, як друкувати числові значення, програма виводить на початку кожного рядка її номер. Для печатки використовується шрифт Courier New розміром 10 пунктів.



Малюнок 3.2.2 - Додаток Lister друкує будь-який текстовий файл, демонструючи принципи реалізації команд File(Print і File (Print Setup

Приклад 3.2.2 - Реалізації команд File⇨Print і File ⇨Print Setup

```
unit Main;
interface
uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons, Printers;
type
  TMainForm = class(TForm)
    Memo1: TMemo;
    SetupBitBtn: TBitBtn;
    PrintBitBtn: TBitBtn;
    CloseBitBtn: TBitBtn;
    OpenBitBtn: TBitBtn;
    OpenDialog1: TOpenDialog;
    PrintDialog1: TPrintDialog;
    PrinterSetupDialog1: TPrinterSetupDialog;
    procedure OpenBitBtnClick(Sender: TObject);
    procedure SetupBitBtnClick(Sender: TObject);
    procedure PrintBitBtnClick(Sender: TObject);
  private
    {Оголошення закритих (private) членів.}
  public
    {Оголошення загальнодоступних (public) членів.}
  end;
var
  MainForm: TMainForm;
implementation
  {$R *.DFM)
```

```
procedure TMainForm.OpenBitBtnClick(Sender: TObject);
begin
  with OpenDialog1 do
    if Execute then
      begin
        Memo1.Lines.LoadFromFile(FileName);
        Caption := Lowercase(FileName);
      end;
end;
procedure TMainForm.SetupBitBtnClick(Sender: TObject);
begin
  PrinterSetupDialog1.Execute;
end;
procedure TMainForm.PrintBitBtnClick(Sender: TObject);
var
  FPrn: System.Text; {Файлова перемінна для вихідних даних.}
  I: Integer; {Індекс Memo1.Lines.}
  LCol: Integer; {Ширина стовпця номерів рядків.}
begin
  if PrintDialog1.Execute then
    begin
      AssignPrn(FPrn); {Перенаправляємо висновок Write/Writeln на принтер.}
      Rewrite (FPrn) ; {Відкриваємо вихідний файл.}
    try
      Printer.Canvas.Font := Memo1.Font; {Скористаємося шрифтом об'єкта Memo1.}
      with Memo1.Lines do
        begin  {Установимо ширину стовпця номерів рядків, визначивши неї по розмірі
файлу}
          if Count < 10 then LCol := 1 {Рядка 0 .. 9.}
          else if Count < 100 then LCol := 2 {Рядка 10 .. 99.}
          else if Count < 1000 then LCol := 3 {Рядка 100 .. 999.}
          else LCol := 4;
          for I := 0 to Count - 1 do
            begin
              Write(FPrn, I + l:LCol, ': '); {Друкуємо номер рядка.}
              Writeln(FPrn, Lines [I]); {Друкуємо рядок.}
            end;
        end;
    finally
      CloseFile(FPrn); {Закриваємо вихідний файл.}
    end;
  end;
end;
end.
```

Давайте частково відтворимо цю програму і для початку реалізуємо команду Print Setup. Помістите у форму об'єкт компонента PrinterSetupDialog. У режимі конструювання форми він представлений маленьким значком, якого при виконанні програми у формі не буде. Помістивши у форму командну кнопку, створіть процедуру її обробки і вставте в неї наступний оператор:

```
PrinterSetupDialog1.Execute;
```

Якщо ви захочете, щоб подальші дії залежали від того, чи закрив користувач діалогове вікно за допомогою кнопки ОК (хоча це рідко буває необхідно), скористайтеся такими операторами:

```
if PrinterSetupDialog1.Execute then
  {... сюди помістите команди, виконувані, якщо користувач вибере кнопку ОК.}
```

Аналогічно реалізується і команда File(Print, тільки потрібно вибрати компонент PrintDialog. Загальна схема процедури, що відкриває діалогове вікно печатки, приведена в прикладі 3.2.3. Усередину блоку try потрібно помістити звертання до процедур Write і Writeln, як це зроблено в процедурі PrintBitBtnClick додатка Lister (див. приклад 3.2.2).

Приклад 3.2.3 - Шаблон для реалізації команди File⇨Print

```
if PrintDialog1.Execute then
  begin
    AssignPrn(FPrn);
    Rewrite(FPrn);
    try
      { ... Запис тексту в FPrn.}
    finally
      CloseFile(FPrn);
     end;
  end;
```

Звичайно текст друкується тим же шрифтом, яким виводиться на екран. Тому, якщо мова йде про об'єкт такого компонента, як TMemo, просто привласніть значення його властивості Font однойменній властивості об'єкта Printer.Canvas. Не забудьте при цьому попередньо відкрити вихідний файл:

```
Printer.Canvas.Font := Memo1.Font;
```

Задача об'єкта компонента PrintDialog — одержати від користувача інформацію, а використовувати неї для печатки — це уже ваша робота. Після виконання методу Execute скористайтеся властивостями Copies, FromPage і ToPage для організації циклу, у якому будуть роздруковані зазначені користувачем дані:

```
with PrintDialog1 do
  for Copy := 1 to Copies do
    for Page := FromPage to ToPage do
      { ... Тут виконується печатка сторінок.}
```

3.2.3. Клас TPrinter

Перш ніж приступити до освоєння техніки печатки графіки і тексту в режимі WYSIWYG ознайомимося з властивостями і методами класу TPrinter. Саме він стане вашим основним інструментом, коли ви перейдете від елементарних текстових операцій, виконуваних за допомогою процедур Write і Writeln, до повноцінної графічної печатки.

Для доступу до всім описаним нижче властивостям і подіям ви будете звертатися до заздалегідь оголошеного в програмі глобальному об'єктові Printer. А щоб він був доступний із програмного модуля, включите в перелік зовнішніх модулів директиви uses модуль Printers.

3.2.3.1. Властивості класу TPrinter

Нижче перераховані властивості класу TPrinter.

- ***Aborted***. Ця властивість дозволяє визначити, що користувач перервав печатку (закривши, наприклад, вікно диспетчера печатки). У такому випадку воно приймає значення True. Значення цієї булевой перемінної можна перевірити в циклі печатки і, якщо воно істинно, тобто був викликаний метод Abort об'єкта Printer, відразу вийти з циклу, не викликаючи метод EndDoc.

- ***Canvas***. Для печатки графіки і тексту в режимі WYSIWYG користуйтеся властивостями і методами об'єкта Canvas (наприклад, властивостями Pip і Brush) точно так само, як для відображення на екрані.
- ***Capabilities***. Ця властивість являє собою набір значень, що визначають можливості драйвера принтера. Його складові — pcCopies, pcOrientation і pcCollation.
- ***Copies***. Це кількість копій, що повинні бути надруковані.
- ***Fonts***. Це список усіх шрифтів, що підтримує поточний принтер. Його тип — TStrings. Цей список не має значення, якщо користуватися шрифтами TrueType, оскільки навіть для старих драйверів принтерів, що не підтримують TrueType, GDI надає потрібні символи. А в Windows 95, 98 і NT проблем і зовсім бути не повинне.
- ***Handle***. Значення цієї властивості можна передати будь-як функції Windows API, що потрібен дескриптор контексту пристрою (HDC), наприклад функції GetTextMetrics.
- ***Orientation***. Ця властивість може мати одне з двох значень: poPortrait (книжкова орієнтація сторінки) або poLandscape (альбомна орієнтація сторінки). Їм можна користуватися для установки параметрів сторінки, привласнивши йому значення перед печаткою, або для зчитування встановленого раніше значення.
- ***PageHeight***. Це висота сторінки в пикселях; вона залежить від типу принтера і для різних принтерів може дуже сильно відрізнятися.
- ***PageNumber***. Це номер поточної сторінки. Що він означає, визначає ваш додаток. Об'єкт Printer збільшує значення властивості PageNumber на одиницю при кожнім вашому звертанні до властивості NewPage. При печатці чистого тексту номер сторінки збільшується, коли процедура Writeln починає нову сторінку.
- ***PageWidth***. Це ширина сторінки в пикселях; вона залежить від типу принтера і для різних принтерів може дуже сильно відрізнятися.
- ***PrinterIndex***. Це індекс імені поточного принтера в списку властивості Printers.
- ***Printers***. Це список установлених принтерів, що представляє собою об'єкт класу TStrings. Ім'я поточного принтера визначається вираженням Printer.Printers[PrinterIndex].
- ***Printing***. У процесі печатки приймає значення True.
- ***Title***. Привласніть цій властивості рядок, що буде ідентифікувати завдання у вікні диспетчера печатки або служити заголовком завдання в мережі.

3.2.3.2. Методи класу TPrinter

Нижче перераховані методи класу TPrinter.

- ***Abort***. Використовується для переривання роботи принтера. Наприклад, перед циклом печатки можна помістити команду відкриття немодального діалогового вікна. Одна з кнопок такого вікна може бути призначена для переривання печатки. З щигликом на цій кнопці зв'яжіть процедуру, що змінює значення глобального прапора, що перевіряється в циклі. Якщо його значення змінилося, викличте метод Printer.Abort і вийдіть з циклу (EndDoc при цьому викликати не слід). Метод Abort привласнює прапорові Printer.Aborted значення True.
- ***BeginDoc***. Цей метод викликається перед початком чергового завдання для принтера. Не застосовуйте його в тих випадках, якщо користуєтеся описаної на початку глави технологією печатки чистого тексту за допомогою процедур Write і Writeln.
- ***EndDoc***. Використовується після закінчення завдання. Цей метод очищає буфер висновку і при необхідності виконує прогін останньої сторінки. При використанні технології печатки чистого тексту за допомогою процедур Write і Writeln метод EndDoc не застосовується. Не використовується він також після перериванні роботи

принтера за допомогою методу Printer.Abort (хоча в цьому випадку в Delphi версій 3 і 4 ситуація буде не такий критичної, як у Delphi 1 і 2).

- ***NewPage***. Викликайте цю процедуру перед печаткою нової сторінки. За вміст сторінки цілком відповідаєте ви. Метод просто очищає буфер висновку і при необхідності виконує прогін сторінки.

Ніколи не створюйте об'єктів класу TPrinter. Потрібний вам для організації печатки глобальний об'єкт Printer цього класу створюється в модулі Printers.

3.2.4. Друк графіки

Описана в попередньому розділі технологія печатки тексту являє собою компроміс між простотою і якістю. Професійно і сучасно оформлені документи роздруковуються іншим способом, про яке буде розказано нижче. Цей розділ присвячений печатки форм, графіки і тексту в режимі WYSIWYG. Крім того, ми розглянемо додаток - приклад, що демонструє спосіб організації попереднього перегляду даних, що друкуються, на екрані.

3.2.4.1. Друк форм

Включити в програму команду, що роздруковує копію форми, зовсім нескладно. Звертатися до об'єкта Printer і включати в модуль посилання на Printers у даному випадку не знадобиться — досить методу Print самої форми.

Тепер докладніше. Помістите у форму об'єкт компонента PrintDialog (ні на екрані, ні в друкованій копії під час виконання програми його не буде). Потім помістите у форму командну кнопку й в оброблювач її події OnClick упишіть наступні операторы:

if PrintDialog1.Execute **then** Print;

Указати, яку саме форму потрібно роздрукувати, можна так:

AboutBox.Print;

У приведеному операторі AboutBox — це значення властивості Name об'єкта форми, що повинна бути роздрукована. Метод Print спочатку створює в пам'яті растрову копію клиентной частини вікна, а потім викликає функцію Windows API StretchDIBits для її печатки. Для печатки зображення може бути масштабировано. Як саме, залежить від значення, що ви задасте для властивості об'єкта форми PrintScale програмно або у вікні Object Inspector. Існують такі варіанти.

- ***poNone***. Масштабирование не виконується. Розмір зображення залежить тільки від дозволу принтера і, якщо воно високе, як, наприклад, у лазерного принтера, картинка хоч і чітка, але настільки маленька, що розглянути її практично неможливо. Цією опцією ви будете користуватися рідко.
- ***poPrintToFit***. Зображення збільшується, наскільки дозволяє розмір сторінки. Хоча так затрачається багато фарби, якщо вам потрібна картинка на всю сторінку, ця опція — те що потрібно.
- ***poProportional***. Зображення масштабируется виходячи зі співвідношення дозволів принтера і монітора (кількості пикселей на дюйм) таким чином, щоб картинки на екрані і на папері здавалися однакового розміру. (У точності їхні розміри збігатися не будуть.) Така картинка і не занадто велика, і добре смотрится.

3.2.4.2. Друк графічних об’єктів

Графіка і текст у режимі WYSIWYG друкуються інакше, чим чистий текст, для якого досить процедур Write і Writeln. Хоча використовується той же об'єкт Printer (створений у модулі Printers), щоб почати і завершити завдання печатки, потрібно звернутися до методів класу TPrinter. Сторінку і весь її вміст потрібно буде сформувати самостійно. Це серйозна робота, але зате ви зможете цілком контролювати процес.

Основні етапи процедури печатки на прикладі об'єкта компонента TImage показані в прикладі 3.2.4. Компонент TImage використовувати необов'язково - у приклад він включений просто для демонстрації. От три основних етапи печатки.

1) Для початку печатки викличте метод BeginDoc об'єкта Printer.
2) Для формування зображення сторінки скористайтеся методами об'єкта Printer.Canvas. Наприклад, можете скористатися методом Draw для печатки на сторінці зображення з об'єкта класу TGraphic (растрового зображення, піктограми або метафайла) або такими методами, як Ellipse і Rectangle. Усе, що ви помістите в об'єкт Canvas, буде роздруковане (звичайно, наскільки дозволять можливості принтера). А щоб перехопити ймовірні помилки, помістите всі команди цього кроку в блок try.
3) Закінчивши печатку, викличте метод EndDoc об'єкта Printer. Це найкраще зробити в блоці finally.

Приклад 3.2.4 - Схема для печатки графіки

```
procedure TForm1.Print1Click(Sender: TObject);
begin
  Printer.BeginDoc;
  try
    Printer.Canvas.Draw(0, 0, Image1.Picture.Graphic);
  finally
    Printer.EndDoc;
  end;
end;
```

Щоб подивитися, як це працює, помістите у форму об'єкти компонентів Image і Button. Розмір об'єкта компонента Image значення не має. Властивістю Image - об'єкта Picture скористайтеся для завантаження будь-якого растрового файлу. Потім, двічі клацнувши на об'єкті компонента Button, створіть обработчик події OnClick і скопіюйте в нього текст приклада 3.2.4. Скомпілюйте і запустите програму, натиснувши кнопку <F9>, а потім, клацнувши на командній кнопці, роздрукуйте зображення.

Не страшно, якщо зображення буде зовсім маленьким. Так може вийти, якщо при високому дозволі принтера не застосувати масштабирование. Особливо це стосується лазерних принтерів, дозвіл яких (300 і більш крапок на дюйм) набагато вище дозволу монітора. Так що результат вийде занадто маленьким, щоб його можна було розглянути.

Щоб збільшити зображення, насамперед потрібно з'ясувати дозвіл друкуючого пристрою. Викличте функцію Windows API GetDeviceCaps (запит зведень про можливості пристрою), а як параметр, що визначає, яке значення з контексту пристрою принтера ви запитуєте, передайте їй спочатку logPixels, а потім — logPixels. Ви одержите два числа, що визначають кількість пикселей на логічний дюйм по горизонталі і вертикалі. Розділивши них на значення властивості форми PixelsPerInch, ви одержите коефіцієнти збільшення зображення для печатки.

Логічний дозвіл конкретного пристрою може трохи відрізнятися від реального. Наприклад, Windows відображає текст на екрані трохи більшим, ніж можна припустити з розміру шрифту в пунктах, щоб при низькому дозволі дрібний шрифт залишався помітним.

Удосконалений варіант процедури з приклада 3.2.4 приведений у прикладі 3.2.5. У ньому зображення масштабируется відповідно до дозволу принтера. Заміните текст процедури обробки події командної кнопки OnClick текстом, приведеним у прикладі 3.2.5, перекомпілюйте програму, запустите її і роздрукуйте зображення знову. Тепер воно повиннео бути ближче по розмірі до тому, що ви бачите на екрані, хоча і не точно таким же.

Приклад 3.2.5 - Настроювання масштабу печатки відповідно до дозволу принтера

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ScaleX, ScaleY: Integer;
  R: TRect;
begin
  Printer.BeginDoc;
  with Printer do
```

```
    try
      ScaleX := GetDeviceCaps(Handle, logPixelsX) div PixelsPerInch;
      ScaleY := GetDeviceCaps(Handle, logPixelsY) div PixelsPerInch;
      R := Rect(0, 0, Image1.Picture.Width * ScaleX, Image1.Picture.Height * ScaleY);
      Canvas.StretchDraw(R, Image1.Picture.Graphic);
    finally
      EndDoc;
    end;
  end;
```

Спочатку процедура з цього приклада инициализирует дві цілі перемінні Scale і Scale. Їм привласнюються значення коефіцієнтів горизонтального і вертикального масштабирования: отримані від функції GetDeviceCaps значення логічного горизонтального і вертикального дозволів принтера, розділені на значення властивості форми PixelsPerInch. Для монітора дозвіл в обох напрямках однакове, але не для всіх зовнішніх пристроїв це так. Тому функція GetDeviceCaps повертає обох значень дозволу.

Після ініціалізації перемінних Scale і Scale процедура поміщає в запис R типу TRect розміри вихідного зображення, множачи для їхнього одержання висоту і ширину вихідного зображення (Image1.Picture.Height і Image1.Picture.Width) на коефіцієнт масштабирования. І нарешті, за допомогою методу StretchDraw об'єкта Printer.Canvas процедура малює зображення, що бере з властивості Graphic об'єкта Picture1.

Насправді печатка починається тільки після виклику методу EndDoc або NewPage. А уся вищеописана процедура - це тільки підготовка зображення для печатки.

3.2.4.3. Печатка растрових зображень, піктограм і метафайлов

Для печатки графіки зовсім необов'язково користуватися компонентом TImage. У прикладі 3.2.6 приводиться інший спосіб печатки, що підходить для растрових зображень, піктограм і метафайлов. Створіть об'єкт класу TPicture і передайте його властивість Graphic методові Draw або StretchDraw об'єкта Printer.Canvas. Процедура, приведена в прикладі 3.2.6, друкує зображення з файлу Sample.bmp.

Приклад 3.2.6 - Печатка растрового зображення, піктограми або метафайла

```
procedure TForm1.Button1Click(Sender: TObject);
var
  P: TPicture;
  ScaleX, ScaleY: Integer;
  R: TRect;
begin
  P := TPicture.Create;
  try
    //Замініте рядок імені файлу в наступному операторі
    //ім'ям будь-якого файлу .bmp, .ico або метафайла
    P.LoadFromFile('D:\Delphi 4\Source\Data\Sample.bmp');
    Printer.BeginDoc;
    with Printer do
      try
        ScaleX := GetDeviceCaps(Handle, logPixelsX) div PixelsPerInch;
        ScaleY := GetDeviceCaps(Handle, logPixelsY) div PixelsPerInch;
        R := Rect(0, 0, P.Width * ScaleX, P.Height * ScaleY);
        Canvas.StretchDraw(R, P.Graphic);
      finally
        Printer.EndDoc;
      end;
  finally
```

```
    P.Free;
  end;
end;
```

Результат виконання процедур, приведених у прикладах 3.2.5 і 3.2.6, той самий. Яку з двох технологій вибрати, залежить від того, чи хочете ви помістити у форму об'єкт компонента Image або програмним шляхом створити об'єкт класу TPicture. Простіше, мабуть, перший спосіб, але з погляду використання пам'яті ощадливіше другий.

3.2.5. Корисні поради

- Якщо шрифт надрукованого тексту виявився не того розміру, переконаєтеся, що метод Print.BeginDoc викликаний перед установкою значень складових Printer.Canvas.Font.
- Не забудьте по закінченні печатки викликати CloseFile для файлової перемінної, котру ви передавали процедурі AssignPrn. При спробі призначити принтерові другий вихідний файл до закриття першого Delphi згенерує виняткову ситуацію.
- При печатці чистого тексту замість обчислення кількості рядків на сторінці, як описувалося в цій главі, можна скористатися більш простим способом аналізу кінця сторінки (наприклад, для печатки заголовка нової сторінки). Після кожного виклику WriteLn перевіряйте значення властивості PageNumber об'єкта Printer.
- При печатці залишайте на сторінці хоча б невеликі поля. У багатьох принтерів чіткість зображення в країв сторінки нижче.
- Завжди привласнюйте властивості Title об'єкта Printer рядок — заголовок завдання, щоб, заглянувши у вікно диспетчера печатки або список мережних завдань для принтера, можна було визначити своє.
- Діалогове вікно з кнопкою переривання печатки можна організувати так. Створіть для нього окрему форму, що відкривається методом Show. Створіть глобальну перемінн-прапор, що сигналізує про те, що користувач хоче зупинити печатка. Инициализируйте її значенням False, значення True привласніть їй під час щиглика на відповідній кнопці і, знайшовши його в циклі печатки, викличте метод Printer.Abort. Метод Printer.EndDoc у цьому випадку викликати не потрібно.
- Можна з'ясувати, чи була перервана печатка (наприклад, шляхом виклику методу Printer.Abort), перевіривши значення властивості-прапора Printer.Aborted (ця властивість доступна тільки для читання).

3.2.6. Резюме

Для взаємодії з користувачем через стандартні діалогові вікна печатки і настроювання принтера призначені компоненти Delphi TPrintDialog і TPrinterSetupDialog.

Для організації печатки служить компонент TPrinter, якого на палітрі VCL немає. Для його використання в директиві uses програмного модуля потрібно вказати модуль Printers. Після цього можна привласнювати значення властивостям глобального об'єкта Printer і звертатися до його методів, таким як Printer.BeginDoc і Printer.NewPage. Створювати об'єкти класу TPrinter не потрібно ніколи. Будь-які операції печатки виконуються через єдиний глобальний об'єкт Printer, створений у модулі Printers.

Існує два способи печатки. Перший, що використовує процедури Write і Writeln мови Pascal, служить тільки для печатки чистого тексту. Печатка графіки і тексту в режимі WYSIWYG здійснюється через об'єкт Printer, при цьому зображення спочатку формується в його складової Canvas (так само, як це робилося для відображення

графіки на екрані), а потім цілком роздруковується. Для запуску завдання печатки потрібно викликати метод Printer.BeginDoc, для прогону сторінки — NewPage, а для завершення завдання — EndDoc.

Щоб роздрукувати форму, досить звернутися до її методу Print. Метод сформує растрове зображення клієнтської області форми і роздрукує його за допомогою методів BeginDoc і EndDoc.

Якщо ви створюєте професійний додаток, постачите його командою попереднього перегляду документів. Створіть процедуру формування зображення сторінки і помістите неї в окремий модуль. Ця процедура повинна формувати зображення в переданому їй об'єкті Printer.Canvas для печатки або в об'єкті-властивості Canvas внутрішнього об'єкта програми класу TBitMap — для висновку на екран.

ЧАСТИНА II

РОЗДІЛ 1 РОЗРОБКА ДОДАТКІВ

Тема 1.1. Взаємодія з діалоговими вікнами (2 години)

У термінології Windows діалоговими вікнами називаються стандартного виду вікна без кнопок максимізації і мінімізації з елементами керування на зразок прапорців і перемикачів. Але в Delphi цьому визначенню може відповідати вікно будь-якої форми. За допомогою діалогового вікна додаток може про щось повідомити або задати користувачеві питання, надати можливість вибору опцій або текстове поле для введення даних. Розроблювачам Delphi пропонує ряд компонентів, що представляють собою стандартні діалогові вікна Windows, а також компоненти TTabControl, TPageControl і TTabSheet, завдяки яким можна створювати власні многостраничные вікна. З цієї глави ви довідаєтеся, як ними користуватися, а також познайомитеся з двома новими програмними технологіями, що з'явилися в Delphi 4: стикуванням (docking) елементів керування й обмеженням розмірів вікон.

1.1.1. Компоненти

Нижче перераховані включені в Delphi компоненти для створення стандартних діалогових вікон і їхніх розповсюджених елементів. Під стандартними діалоговими вікнами в даному випадку маються на увазі стандартні вікна Windows.

- ***TColorDialog***. Цей компонент активізує стандартне вікно вибору кольору. Якщо потрібно надати користувачеві можливість вибрати колір, швидше за все, компонент TColorDialog — це усе, що вам потрібно.
  Категорія палітри: Dialogs.
- ***TFindDialog***. Цей компонент активізує стандартне вікно пошуку файлу. Використовуйте його, щоб надати користувачеві засіб пошуку даних на диску.
  Категорія палітри: Dialogs.
- ***TFontDialog***. Це компонент, що активізує стандартне вікно вибору шрифту, у якому користувачі можуть вибирати шрифт і колір тла відображуваного тексту.
  Категорія палітри: Dialogs.
- ***TOpenDialog***. Цей компонент активізує стандартне вікно вибору файлу. Використовуйте його у випадках, коли користувач повинний указати місце розташування даних на диску, як правило, у відповідь на команду File⇨Open.
  Категорія палітри: Dialogs.
- ***TPageControl***. Якщо потрібно створити вікно зі сторінками-вкладками, то за допомогою даного компонента зробити це зовсім неважко. Кожна сторінка вікна об'єкта компонента TPageControl являє собою окремий об'єкт компонента TTabSheet. На сторінку можна перешкодити будь-які елементи керування: кнопки, прапорці, перемикачі — усе, що необхідно. Працюючи з таким вікном, користувач вибирає потрібну вкладку, клацнувши на її корінці. Це відмінний спосіб розмістити багато елементів на невеликому просторі, якщо, приміром, мова йде про програму з великою кількістю опцій.
  Категорія палітри: Win32.
- ***TReplaceDialog***. Цей компонент активізує стандартне вікно пошуку і заміни фрагмента тексту (як правило, у текстовому документі або таблиці).
  Категорія палітри: Dialogs.
- ***TSaveDialog***. Цей компонент активізує стандартне вікно збереження файлу. У даному

вікні, що відкривається, як правило, у відповідь на команду File⇨Save, користувач може ввести або вибрати ім'я файлу і знайти потрібний каталог.
Категорія палітри: Dialogs.

- ***TTabControl***. Цей компонент подібний компонентові TPageControl. Відрізняються вони тим, що сторінки TabControl ілюзорні, їх потрібно створювати програмно під час виконання, у той час як сторінки PageControl являють собою створювані при конструюванні форми окремі об'єкти компонента TTabSheet.
Категорія палітри: Win32.
- ***TTabSheet***. Об'єкти цього компонента являють собою сторінки, що поміщаються в многостраничное діалогове вікно, створене за допомогою компонента TPageControl. Щоб створити таку сторінку, клацніть правою кнопкою миші усередині об'єкта компонента TPageControl і виберіть з контекстного меню команду New Page.
Категорія палітри: немає.

1.1.2. Діалогові режими

У додатках, як правило, застосовуються два види діалогових вікон: модальний, утримуючий фокус уведення доти, поки не будуть закриті, і немодальні, з яких користувач може переключатися в інші вікна. Наприклад, вікно вибору опцій звичайно модальне, і користувач не може продовжити роботу з додатком, не закривши таке вікно. А от вікна пошуку і заміни звичайно немодальні: можна почати пошук, переключитися у вікно редагування, внести зміни в текст, а потім продовжити пошук.

1.1.2.1. Модальні діалогові вікна

Щоб відобразити форму у виді модального діалогового вікна, потрібно викликати її метод ShowModal. Нехай, наприклад, у додатку існує форма MyDialog. Тоді з оброблювача подій однієї з кнопок додатка можна відкрити цю форму в модальному режимі за допомогою наступного оператора:

MyDialog.ShowModal;

Метод ShowModal повертає значення властивості ModalResult форми, що відкривається їм. Установлюється це значення після щиглика на одній із кнопок, що служать для закриття вікна, наприклад на OK, Close або Cancel. Щоб з'ясувати, чи закрив користувач модальне вікно за допомогою кнопки OK, скористайтеся наступним фрагментом програмного коду:

if MyDialog.ShowModal = mrOk **then**
{... Якщо користувач клацнув на кнопці OK, почати деякі дії.}

Значення, що повертається, можна привласнити властивості ModalResult програмно. Для цього потрібно помістити в процедуру обробки події закриваючу форму кнопки, наприклад кнопки OK, такий оператор:

ModalResult := mrOk;

Можна надійти й інакше, просто вибравши для властивості цієї кнопки ModalResult значення mrOk. Для стандартних компонентів TButton і TBitBtn це виконується автоматично.

Що привласнюється ModalResult значення (яке буде повернуто методом ShowModal) не повинне бути нульовим.

Метод ShowModal повертає значення типу integer. Якщо ж необхідно повернути з модального вікна що-небудь ще, включите в клас форми функцію, що поверне потрібні дані. От приклад оголошення строкової функції:

TYourForm = **class**(TForm)
...public
function GetStringResult: **String**;

Нехай потрібно повернути додаткові значення, що користувач ввів у текстове поле (значення властивості Text об'єкта компонента TEdit). Тоді реалізація нашої функції може

виглядати так:

```
function TYourForm.GetStringResult: String;
begin
  with Edit1 do
    if Length(Text) = 0 then
      Result := 'Default string'
    else Result := Text;
end;
```

Залишилося одержати результат, викликавши для цього тільки що написану функцію відразу після закриття модального вікна:

```
if YourForm.ShowModal = mrOk then
  S := YourForm.GetStringResult;
```

1.1.2.2. Немодальні діалогові вікна

У Windows немодальне вікно визначається як дочірнє вікно, що не захоплює фокус уведення, дозволяючи користувачеві продовжувати роботу з додатком. У Delphi же це просто вікно будь-якої форми, з якого можна переключатися в інші вікна. Для того щоб відкрити вікно форми в такому режимі, скористайтеся методом Show:

```
MyDialog.Show;
```

Він привласнює властивості форми Visible значення True і викликає метод BringToFront, щоб вивести вікно на передній план, якщо його закривають інші вікна. Якщо потрібно сховати вікно, просто привласніть властивості Visible значення False:

```
MyDialog.Visible := False;
```

Щоб з'ясувати, чи активно діалогове вікно, перевірте значення цієї властивості.

1.1.3. Стандартні діалогові вікна

Компоненти палітри Dialogs надають у розпорядження розроблювача стандартні діалогові вікна Windows. Чотири з них - FontDialog, ColorDialog, OpenDialog і SaveDialog - служать для вибору шрифту, кольори і завдання імені файлу.

1.1.3.1. Діалогові вікна вибору шрифтові та кольору

У прикладі 1.1.1 показано, як користуватися компонентами TFontDialog і TColorDialog. Якщо метод Execute повернув True, виходить, користувач закрив вікно, клацнувши на кнопці OK, і можна з'ясувати, що він вибрав, звернувшись до властивості Font або Color (у залежності від того, про яке вікно мова йде).

Приклад 1.1.1 - Використання компонентів TFontDialog і TColorDialog

```
procedure TMainForm.OptionsFontClick(Sender: TObject);
begin
  if FontDialogl.Execute then
    Memol.Font := FontDialogl.Font;
end;
procedure TMainForm.OptionsBackgroundClick(Sender: TObject);
begin
  if ColorDialogl.Execute then
    Memol.Color := ColorDialogl.Color;
end;
```

Компонент TColorDialog має три корисних властивості-опції, яким можна привласнити необхідні значення (True або False) або у вікні Object Inspector, або програмно під час виконання.

- ***cdFullOpen***. Ця властивість дозволяє вибрати один із двох варіантів діалогового вікна — повний або скорочений. Значення False відповідає табличці квітів, один із яких користувач може вибрати. У випадку значення False вікно містить додаткові засоби

настроювання кольору, включаючи опції окремого настроювання червоної, зеленої і синьої складових. На “повільних” комп'ютерах для відкриття такого вікна потрібно кілька секунд, тому краще вибрати його скорочений варіант — у користувача буде можливість відкрити і повне вікно за допомогою кнопки **Визначити колір**.

- ***cdPreventFullOpen***. Ця властивість дозволяє відключити кнопку **Визначити колір**, щоб користувач не зміг відкрити повне вікно вибору кольору. Їм можна скористатися, щоб обмежити припустимі кольори деяким заданим набором. Для цього привласніть властивості значення True.
- ***cdShowHelp***. Щоб у вікні була присутня кнопка Help, привласніть цій властивості значення True. Якщо в додаток включена інтерактивна довідка, привласніть властивості діалогового вікна HelpContext відповідне значення. Якщо ж довідки ні, привласніть властивості cdShowHelp значення False.

Компонент TFontDialog має набагато більше властивостей-опцій. Призначення деяких з них, наприклад fdTrueTypeOnly (тільки TrueType) і fdFixedPitchOnly (тільки моноширинные), очевидно. Інші ж властивості мають потребу в поясненні.

- ***fdAnsiOnly***. Якщо привласнити цій властивості значення True, у списку шрифтів будуть присутні тільки ті, котрі відповідають наборові символів Windows. Наприклад, шрифту WingDings у списку не буде.
- ***fdEffects***. Якщо привласнити цій властивості значення False, у вікні не буде опцій вибору ефектів закреслювання або підкреслення тексту, а також вибору кольору. Якщо ці ефекти у вашій програмі не використовуються, їхній вибір завжди можна (і потрібно) відключити. На жаль, не можна відключати ці можливості по однієї; це обмеження накладає Windows.
- ***fdNoFaceSel***. Ця опція дозволяє при відкритті діалогового вікна очистити частину, що редагується, полючи зі списком, що служить для введення або вибору шрифту. Звичайно це поле містить назва пропонованого за замовчуванням шрифту. Щоб воно було порожнім, привласніть властивості fdNoFaceSel значення True.
- ***fdNoSimulations***. Привласніть цій властивості значення True, якщо не хочете, щоб у списку шрифтів відображалися шрифти, эмулирующие GDI. Ця опція відноситься тільки до принтерних шрифтів.
- ***fdNoSizeSel, fdNoStyleSel***. Ці дві опції визначають, чи будуть при відкритті вікна відразу виділені розмір і стиль шрифту відповідно. Якщо ви не хочете заздалегідь пропонувати користувачеві характеристики шрифту, установите для обох цих властивостей і для властивості fdNoFaceSel значення True.
- ***fdWysiwyg***. Якщо привласнити цій властивості значення True, у списку шрифтів будуть присутні тільки шрифти, що володіють властивістю *WYSIWYG* (абревіатура від *What You See Is What You Get* — Що бачиш, то й одержиш), тобто выглядящие однаково як на екрані, так і на принтері. Якщо ж ця властивість має значення False, може виявитися, що обраний користувачем шрифт на печатці виглядає зовсім не так, як передбачалося. Тому використовуйте значення False тільки у випадках, коли мова йде про текст на екрані, і ви упевнені, що друкувати його користувач не буде.
- ***fdLimitSize***. Значення цієї властивості True дозволяє обмежити можливі розміри шрифту мінімальної (властивість MinFontSize) і максимальної (властивість MaxFontSize) величинами.

Вікно, створюване компонентом TFontDialog, можна також використовувати для вибору шрифтів, сумісних з визначеним принтером.

1.1.3.2. Діалогові вікна відкриття та збереження файлові

Розглянемо додаток TabEdit — ця програма може служити шаблоном для будь-якого додатка, що має справу з операціями над файлами. Написання таких програм можна почати з

двох процедур — одна читає файл із диска, а інша його записує. От їхній код (приклад 1.1.2).

Приклад 1.1.2 - Процедури запису і читання файлів додатка TabEdit

```
{Читає файл із диска.}
procedure TMainForm.LoadFile(const Path: String);
begin
  with Pages[TabSetl.TabIndex] do
    try
      Memol.Lines.LoadFromFile(Path);
      Dirty := False;
      Page.Clear;
      SetFilename(Path);
    except on e: EReadError do
      MessageDlg('Error reading file', mtError, [mbOk], 0);
    end;
end;
{Записує поточний файл на диск.}
procedure TMainForm.SaveFile(Index: Integer);
begin
  with TabSetl, Pages[Index] do
    begin
      try
        Memol.Lines.SaveToFile(Filename);
        Dirty := False;
      except on e:EWriteError do
        MessageDlg('Error writing file', mtError, [mbOk], 0);
      end;
    end;
end;
```

В обох процедурах для перехоплення помилкових операторів використовуються виключення Pascal. Якщо відбувається помилка, керування передається операторові except, у блоці якого виводиться діалогове вікно з повідомленням. Більш докладно про обробку виключень буде розказано нижче, але вже зараз, глянувши на приведений приклад, ви можете переконатися, що це дуже просто. Помістите всі операторы, виконання яких може привести до помилки, у блок try, а оператори, які потрібно виконати у випадку її виникнення, — у блок except, завершивши отриману конструкцію оператором end. У блоці except частина on-do призначена для того, щоб оброблювач був виконаний тільки для визначених об'єктів виключень (у даному випадку — для EReadError і EWriteError), а для інших використовувалися стандартні оброблювачі Delphi.

Написавши процедури читання і запису, перейдемо до процедур обробки команд меню File. Їхній текст приведений у прикладі 1.1.3. Цим прикладом можна скористатися, щоб додати команди Open, Save і Save as у власний додаток.

Приклад 1.1.3 - Процедури обробки команд меню File додатка TabEdit

```
{Команда File⇨Open.}
procedure TMainForm.FileOpenClick(Sender: TObject);
begin
  with Pages[TabSetl.Tablndex] do
    begin
      if Dirty then FileSaveClick(Sender);
      if {усе ще} Dirty then Exit; {Файл не збережений}
      if FileOpenDialog.Execute then
        LoadFile(FileOpenDialog.Filename);
```

```
    end;
  end;
  {Команда File ⇨Close.}
  procedure TMainForm.FileCloseClick(Sender: TObject);
  var
    W: Word;
  begin
    with TabSetl, Pages[Tablndex] do
      begin
        if Dirty then
          begin
            W := MessageDlg('Save changes to ' + Tabs[Tablndex] + '?', mtWarning, [mbYes, mbNo,
mbCancel], 0);
            case W of
              mrYes: FileSaveClick(Sender);
              mrNo: Dirty := False;
              mrCancel: Exit;
            end;
          end;
          if {усе ще} Dirty then Exit; {Файл не збережений.}
          Page.Clear;
          Memo1.Clear;
          Filename := UntitledName;
          Tabs[Tablndex] := Filename;
      end;
  end;
  {Команда File ⇨Save}
  procedure TMainForm.FileSaveClick(Sender: TObject);
  begin
    with TabSet1, Pages[Tablndex] do
      if Filename = UntitledName then
        FileSaveAsClick(Sender)
      else
        SaveFile(Tablndex);
  end;
  {Команда File ⇨SaveAs.}
  procedure TMainForm.FileSaveAsClick(Sender: TObject);
  begin
    with TabSetl, Pages[Tablndex] do
      if FileSaveDialog.Execute then
        begin
          SetFilename(FileSaveDialog.Filename);
          SaveFile(Tablndex);
        end;
  end;
```

Процедура FileOpenClick спочатку зберігає поточний файл, якщо значення властивості Dirty дорівнює True, тобто якщо користувач уніс деякі зміни. Потім вона виводить діалогове вікно відкриття нового файлу, викликавши метод Execute об'єкта FileOpenDialog. І якщо цей метод повернув True, процедура намагається відкрити обраний користувачем файл.

Процедура FileOpenClick пропонує користувачеві зберегти внесені зміни і закриває файл. Для збереження змін вона звертається до процедури FileSaveClick, а та, у свою чергу, якщо ім'я

файлу не задано, — до процедури FileSaveAsClick. Обидві процедури, і FileSaveClick, і FileSaveAsClick, для запису файлу на диск викликають приведену в прикладі 1.1.2 процедуру SaveFile.

Процедура FileSaveAsClick викликає метод Execute об'єкта FileSaveDialog. Якщо цей метод повернув True, виходить, користувач вказав ім'я файлу. У цьому випадку процедура привласнює задане ім'я файлу ярликові поточної сторінки і викликає SaveFile для запису файлу на диск.

1.1.3.3. Фільтри та типи файлів

Якщо програма може відкривати файли різних типів, краще пропонувати користувачеві в діалоговому вікні збереження файлу тільки файли того ж типу, що й останній відкритий користувачем файл. Якщо, приміром, користувач відкрив текстовий файл, то програма повинна зберегти його як текстовий, а якщо він відкрив файл растрового зображення, то в списку повинні бути присутнім тільки файли *.bmp. Це і нагадає користувачеві, з файлом якого типу він працював, і зменшить імовірність виникнення помилок. Якщо ваша програма не уміє виконувати перетворення типів, краще не давати користувачеві приводу припускати, що цю операцію можна виконати, просто змінивши розширення імені файлу.

Для обмеження типів відображуваних у діалоговому вікні файлів служить властивість об'єкта цього вікна Filter. Якщо, приміром, мова йде про файли *.bmp, даному властивості потрібно привласнити значення 'Files (*.bmp)|*.bmp'і

SaveDialog1.Filter := 'Files (*' + ExtractFileExt(OpenDialog1.FileName) + ')|*' + ExtractFileExt(OpenDialog1.FileName);

За допомогою функції Format можна, уклавши в квадратні дужки повторюваний аргумент, скоротити приведений фрагмент коду, щоб уникнути повторного виклику функції ExtractFileExt.

SaveDialog1.Filter := Format('Files (*.%s)|*.%0:s', [ExtractFileExt(OpenDialog1.Filename)]);

1.1.3.4. Створення спискові файлів які недавно використовувались

У компонентів TOpenDialog і TSaveDialog є властивість HistoryList — список рядків (об'єкт класу TStrings), якому можна використовувати для збереження імен недавно відкривалися файлів. Цей список можна навіть зберегти на диску, щоб відновити при черговому запуску додатка.

Для створення списку недавно використовувалися файлів виконаєте наступне.

1) Помістите у форму об'єкт компонента TOpenDialog або TSaveDialog.
2) Для властивості цього об'єкта FileEditStyle виберіть значення fsComboBox.
3) Додайте імена обираних користувачем файлів у список властивості HistoryList (як це зробити, розповідається далі).

Щоб виконати п. 3, потрібно написати процедуру обробки події, зв'язаного з відкриттям або збереження файлу. Її код, якому можна, приміром, призначити події OnClick командної кнопки вікна, повинний бути таким:

```
with OpenDialog1 do
  if Execute then
    begin
      {Тут відкрийте файл.}
      FileName := Lowercase(FileName);
      HistoryList.Add(FileName);
    end;
```

То же саме можна зробити і з об'єктом компонента TSaveDialog. У результаті, коли користувач відкриє діалогове вікно, замість звичайного поля редагування імені файлу буде поле зі списком, що містить імена останніх файлів, що відкривалися їм. (Проводячи цей експеримент, не забудьте вибрати значення fsComboBox для властивості FileEditStyle, інакше поле зі списком не з'явиться.)

Для обмеження кількості імен файлів у списку HistoryList оголосите в модулі константу

maxHistoryList:

```
Const
  maxHistoryList = 6;
```

Файли повинні міститися в список таким чином, щоб останній був у самому верху. При цьому по досягненні максимальної кількості файлів файл, що відкривався найбільше давно, просто віддаляється зі списку. Робиться це так:

```
with OpenDialog1 do
  if Execute then
    begin
    {Тут відкрийте файл.}
    FileName := Lowercase(FileName);
    with HistoryList do
      begin
        if Count = maxHistoryList then
          Delete(Count - 1);
          HistoryList.Insert(0, FileName);
      end;
    end;
```

Якщо ви захочете додати в додаток команду, що очищає список недавно використовувалися файлів, використовуйте наступний код:

```
OpenDialog1.HistoryList.Clear;
SaveDialog1.HistoryList.Clear;
```

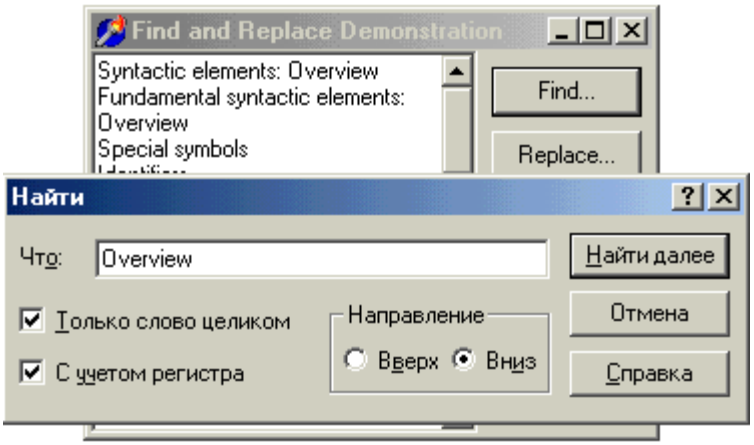
1.1.4. Діалогові вікна пошуку

1.1.4.1. Діалогові вікна пошуку

Компонент TFindDialog має два методи й одна подія. Діалогове вікно, показане на мал. 1.1.1, відкриває метод Execute і закриває метод CloseDialog. Пошук інформації виконується оброблювачем події OnFind, генерируемого після щиглика на кнопці **Find Next**.

Код цього оброблювача вам доведеться написати самостійно. Щоб одержати введений користувачем зразок пошуку, звернетеся до властивості FindText об'єкта FindDialog.

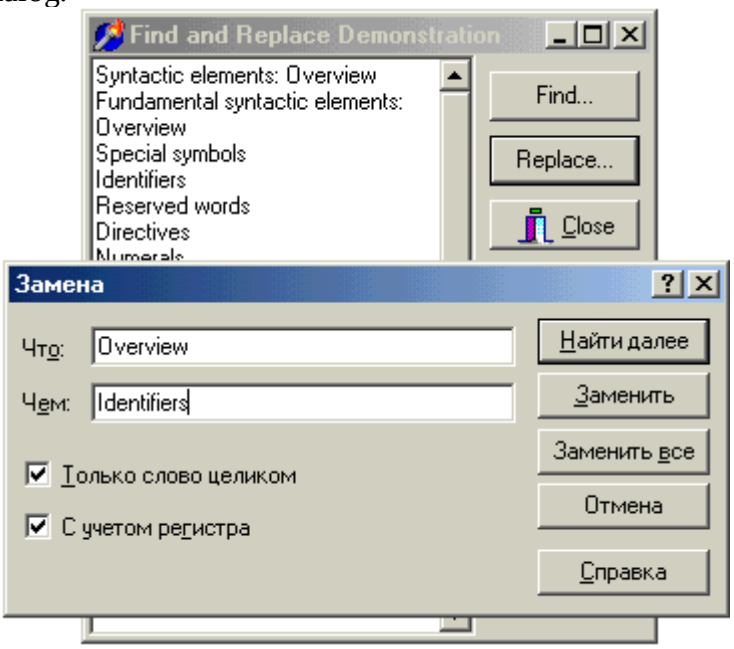
При створенні об'єкта компонента TFindDialog можна скористатися поруч властивостей групи Options, що визначають опції діалогового вікна пошуку, наприклад наявність прапорця обліку регістра.



Малюнок 1.1.1 - Це немодалыюе діалогове вікно відображене об'єктом компонента TFindDialog

1.1.4.2. Діалогове вікно пошуку та заміни

Компонент TReplaceDialog (мал. 1.1.2) являє собою розширену версію компонента TFindDialog. У нього включене ще одне текстове поле для введення тексту, що заміняє. Відповідно додано і ще одна властивість, ReplaceText, що містить цей текст. Для відображення діалогового вікна пошуку і заміни служить метод Execute, а для його закриття — метод CloseDialog.



Малюнок 1.1.2 - Це немодальне діалогове вікно відображене об'єктом компонента TReplaceDialog

Подій у компонента ReplaceDialog два. Оброблювач першого, OnFind, служить для пошуку введеного користувачем тексту, а оброблювач другого, OnReplace, виконує заміну. У їхній реалізації є трохи тонкостей, про які розповідається в наступному розділі.

1.1.4.3. Програмування пошуку та заміни

Помістити компоненти у форму просто, а от запрограмувати їхню правильну роботу набагато складніше. Вікно додатка FindRepl (мал. 1.1.2) містить список узятий з довідкової системи Delphi (роздягнув Syntactic elements). Кнопки **Find…**і **Replace…**служать для пошуку і заміни тексту в цьому списку. А як це зроблено, показано в прикладі 1.1.4.

Приклад 1.1.4 - FindRepl\Main.pas

unit Main;

interface

```
uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons;
type
  TMainForm = class(TForm)
    ListBox1: TListBox;
    FindDialog: TFindDialog;
    FindBitBtn: TBitBtn;
    CloseBitBtn: TBitBtn;
    ReplaceBitBtn: TBitBtn;
    ReplaceDialog: TReplaceDialog;
    procedure FindBitBtnClick(Sender: TObject);
    procedure FindDialogFind(Sender: TObject);
    procedure ReplaceBitBtnClick(Sender: TObject);
    procedure ReplaceDialogFind(Sender: TObject);
    procedure ReplaceDialogReplace(Sender: TObject);
  private
    FindIndex, FoundPos, FoundLen: Integer;
    FoundItem: Boolean;
  public
  end;
var
  MainForm: TMainForm;
implementation
{$R *.DFM}
{Початок роботи з FindDialog — відкриття вікна пошуку.}
procedure TMainForm.FindBitBtnClick(Sender: TObject);
begin
  FindDialog.Execute;
  FindIndex := 0;
  ListBox1.ItemIndex := -1;
end;
{Продовження роботи з FindDialog — виконання пошуку.}
procedure TMainForm.FindDialogFind(Sender: TObject);
var
  S: String;
begin
  while FindIndex < ListBox1.Items.Count do
    begin
      S := ListBox1.Items[FindIndex];
      Inc(FindIndex);
      if Pos(FindDialog.FindText, S) <> 0 then
        begin
          ListBox1.ItemIndex := FindIndex - 1;
          Exit;
        end;
    end;
    ShowMessage('No more matches!');
    FindDialog.CloseDialog;
end;
{Початок роботи з ReplaceDialog — відкриття вікна пошуку і заміни.}
procedure TMainForm.ReplaceBitBtnClick(Sender: TObject);
```

```
begin
  ReplaceDialog.Execute;
  FindIndex := 0;
  ListBox1.ItemIndex := -1;
  FoundItem := False;
end;
{Продовження роботи з ReplaceDialog — виконання пошуку.}
procedure TMainForm.ReplaceDialogFind(Sender: TObject);
var
  S: String;
begin
  while FindIndex < ListBox1.Items.Count do
    begin
      S := ListBox1.Items[FindIndex];
      Inc(FindIndex);
      FoundPos := Pos(ReplaceDialog.FindText, S);
      if FoundPos <> 0 then
        begin
          ListBox1.ItemIndex := FindIndex - 1;
          FoundLen := Length(ReplaceDialog.FindText);
          FoundItem := True;
          Exit;
        end;
    end;
    ShowMessage('No more matches!');
    ReplaceDialog.CloseDialog;
end;
{Продовження роботи з ReplaceDialog — виконання заміни.}
procedure TMainForm.ReplaceDialogReplace(Sender: TObject);
var
  S: String;
begin
  if frReplaceAll in ReplaceDialog.Options then
    ShowMessage('Replace All not implemented')
  else if not FoundItem then
    ShowMessage('Click Find to begin/continue search')
  else
    begin
      S := ListBox1.Items[FindIndex - 1];
      Delete(S, FoundPos, FoundLen);
      Insert(ReplaceDialog.ReplaceText, S, FoundPos);
      ListBox1.Items[FindIndex - 1] := S;
      FoundItem := False;
    end;
end;
end.
```

Запрограмувати пошук трохи простіше, ніж заміну. Насамперед створіть для кнопки **Знайти** оброблювач події OnClick. З цього оброблювача викличте метод Execute об'єкта компонента TFindDialog і підготуйте глобальні перемінні, котрі будуть потрібні для виконання пошуку. Наприклад, у програмі FindRep1 дія починається з таких операторів:

```
FindDialog.Execute;
```

FindIndex := 0;
ListBox1.ItemIndex := -1;

Перший рядок відкриває немодальне діалогове вікно для керування пошуком. Друга инициализирует глобальну перемінну, у якій буде зберігатися номер останнього знайденого рядка списку ListBox1, а третій рядок установлює номер поточного елемента списку рівним -1, щоб жоден з його елементів не був виділений.

Користувач клацає в діалоговому вікні на кнопці **Знайти далі**, і запускається оброблювач події FindDialogFind, що виконує пошук. Щоб з'ясувати, з якого рядка списку починати, оброблювач звертається до перемінного FindIndex. За допомогою функції Pos він визначає, чи входить значення властивості FindText у рядок S (у яку попередньо скопійоване значення чергового елемента списку). Якщо результат не дорівнює нулеві, тобто елемент знайдений, він виділяється, для чого його номер привласнюється властивості ItemIndex.

Зверніть увагу, що після того, як відповідність знайдена, діалогове вікно не закривається й об'єкт FindDialog свою роботу не завершує. Він просто передає керування програмі, щоб одержати від користувача нові інструкції. Коли оброблювач FindDialogFind доходить до кінця списку, він виводить ще одне діалогове вікно з повідомленням про те, що шуканий елемент не знайдений. Потім він викликає метод CloseDialog, що закриває діалогове вікно пошуку.

Код для виконання пошуку і заміни трохи складніше. Починається він подібним образом, але инициализирует ще і глобальну перемінну, необхідну для виконання заміни — прапор FoundItem, що вказує, що елемент поки не знайдений (див. процедуру ReplaceBitBtnClick). От як це робиться:

ReplaceDialog.Execute;
FindIndex := 0;
ListBox1.ItemIndex := -1;
FoundItem := False;

Перший рядок відкриває немодальне діалогове вікно для керування пошуком і заміною. Друга инициализирует глобальну перемінну, у якій буде зберігатися номер останнього знайденого рядка списку ListBox1. Третій рядок установлює номер поточного елемента списку рівним -1, щоб жоден з його елементів не був виділений. А четверта привласнює перемінної FoundItem значення False (елемент не знайдений).

Для реалізації пошуку і заміни потрібні два оброблювачі подій. Текст першого з них, ReplaceDialogFind, практично збігається з текстом оброблювача для вікна пошуку. Тільки знайшовши елемент, він привласнює відповідні значення парі глобальних перемінних, FoundLen і FoundItem, необхідних для наступного виконання заміни.

Другий оброблювач, викликуваний слідом за першим, перевіряє значення перемінної FoundItem. Якщо воно дорівнює True, виконується заміна. З приклада видно, як з'ясувати, чи не клацнув користувач **на кнопці** Замінити усі: для цього потрібно перевірити, чи входить константа frReplaceAll у набір опцій об'єкта компонента TFindDialog. Звичайно, що виконує заміну код для кожної програми унікальний і його приходиться писати самостійно.

1.1.5. Багатосторінкові документи

Delphi пропонує до ваших послуг три компоненти, що спрощують створення многостраничных вікон.

- ***TPageControl***. Цей компонент найкраще підійде в ситуації, коли потрібно створити многостраничное вікно з окремим набором елементів керування на кожній сторінці.
  Категорія палітри: Win32.
- ***TTabSheet***. Якщо компонент TPageControl являє собою многостраничное вікно, то TTabSheet представляє його сторінку. На палітрі компонентів його немає. Щоб створити чергову сторінку, клацніть правою кнопкою миші усередині об'єкта компонента TPageControl і виберіть з раскрывшегося меню команду **New Page**.

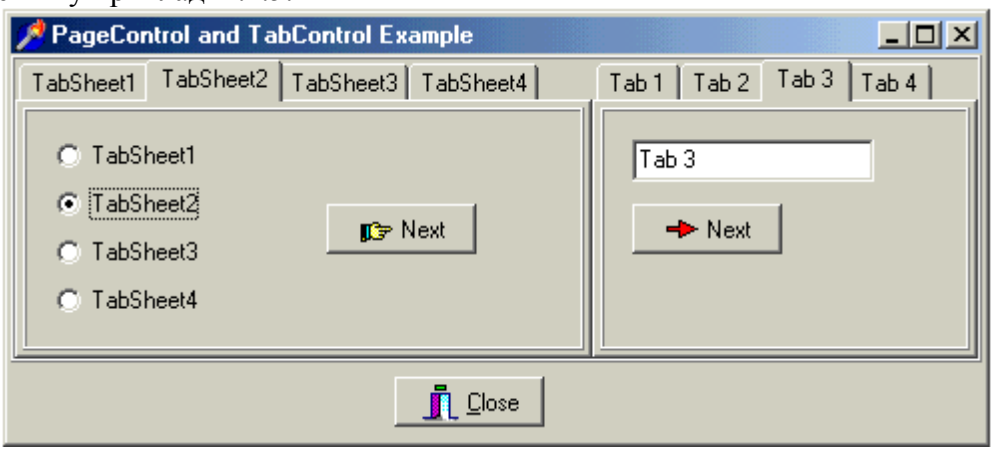
Категорія палітри: Відсутній.

- ***TTabControl***. Якщо ви хочете створити многостраничное діалогове вікно за допомогою одного простого компонента, замість пари TPageControl і TTabSheet скористайтеся TTabControl Зовні результати будуть однаковими, але за розміщення об'єктів на кожній зі сторінок відповідати будете ви. Особливо корисний компонент TTabControl для створення многостраничных вікон з тими самими елементами на різних сторінках, що відрізняються тільки змістом. Це може бути, наприклад, група текстових полів, текст яких міняється при виборі сторінки.

  Категорія палітри: Win32.

Про те, як користуватися цими трьома компонентами, розповідається в наступних розділах.

Як виглядають об'єкти компонентів TTabControl і TPageControl, показане на мал. 1.1.3. Сторінки об'єкта компонента TPageControl є об'єктами компонента TTabSheet. Код додатка приведений у прикладі 1.1.5.



Малюнок 1.1.3 - Демонстраційна програма PageTab може бути прикладом роботи з компонентами TPageControl (ліворуч) і TTabControl (праворуч)

Приклад 1.1.5 - PageTab\Main.pas

```
unit Main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtDlgs, Buttons, ExtCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    CloseBitBtn: TBitBtn;
    MainPanel: TPanel;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Sheet1Panel: TPanel;
    Sheet1RadioButton1: TRadioButton;
    Sheet1RadioButton2: TRadioButton;
    Sheet1RadioButton3: TRadioButton;
    Sheet1RadioButton4: TRadioButton;
    Sheet1BitBtn: TBitBtn;
    TabSheet2: TTabSheet;
    Sheet2Panel: TPanel;
    Sheet2RadioButton1: TRadioButton;
    Sheet2RadioButton2: TRadioButton;
    Sheet2RadioButton3: TRadioButton;
```

```
    Sheet2RadioButton4: TRadioButton;
    Sheet2BitBtn: TBitBtn;
    TabSheet3: TTabSheet;
    Sheet3Panel: TPanel;
    Sheet3RadioButton1: TRadioButton;
    Sheet3RadioButton2: TRadioButton;
    Sheet3RadioButton3: TRadioButton;
    Sheet3RadioButton4: TRadioButton;
    Sheet3BitBtn: TBitBtn;
    TabSheet4: TTabSheet;
    Sheet4Panel: TPanel;
    Sheet4RadioButton1: TRadioButton;
    Sheet4RadioButton2: TRadioButton;
    Sheet4RadioButton3: TRadioButton;
    Sheet4RadioButton4: TRadioButton;
    Sheet4BitBtn: TBitBtn;
    TabControl1: TTabControl;
    TabControlPanel: TPanel;
    TabControlEdit: TEdit;
    TabControlBitBtn: TBitBtn;
    procedure Sheet1RadioButton1Click(Sender: TObject);
    procedure Sheet1RadioButton2Click(Sender: TObject);
    procedure Sheet1RadioButton3Click(Sender: TObject);
    procedure Sheet1RadioButton4Click(Sender: TObject);
    procedure GoToSheet(Index: Integer);
    procedure NextSheets(Sheets: TTabSheet);
    procedure Sheet1BitBtnClick(Sender: TObject);
    procedure TabSheet1Show(Sender: TObject);
    procedure TabSheet2Show(Sender: TObject);
    procedure TabSheet3Show(Sender: TObject);
    procedure TabSheet4Show(Sender: TObject);
    procedure TabControlBitBtnClick(Sender: TObject);
    procedure TabControl1Change(Sender: TObject);
  private
    {Private declarations}
  public
    {Public declarations}
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.Sheet1RadioButton1Click(Sender: TObject);
begin
  GoToSheet(0);
end;
procedure TForm1.Sheet1RadioButton2Click(Sender: TObject);
begin
  GoToSheet(1);
end;
procedure TForm1.Sheet1RadioButton3Click(Sender: TObject);
```

```
begin
  GoToSheet(2);
end;
procedure TForm1.Sheet1RadioButton4Click(Sender: TObject);
begin
  GoToSheet(3);
end;
procedure TForm1.GoToSheet(Index: Integer);
begin
  PageControl1.ActivePage:=PageControl1.Pages[Index];
  PageControl1.Pages[Index].Visible:=True;
end;
procedure TForm1.Sheet1BitBtnClick(Sender: TObject);
begin
  NextSheets(PageControl1.ActivePage);
end;
procedure TForm1.NextSheets(Sheets: TTabSheet);
begin
  PageControl1.ActivePage:=PageControl1.FindNextPage(Sheets, True, False);
  PageControl1.ActivePage.Visible:=True;
end;
procedure TForm1.TabSheet1Show(Sender: TObject);
begin
  Sheet1RadioButton1.Checked:=True;
end;
procedure TForm1.TabSheet2Show(Sender: TObject);
begin
  Sheet2RadioButton2.Checked:=True;
end;
procedure TForm1.TabSheet3Show(Sender: TObject);
begin
  Sheet3RadioButton3.Checked:=True;
end;
procedure TForm1.TabSheet4Show(Sender: TObject);
begin
  Sheet4RadioButton4.Checked:=True;
end;
procedure TForm1.TabControlBitBtnClick(Sender: TObject);
begin
  with TabControl1 do
    begin
      case TabIndex of
        0..2:TabIndex:=TabIndex +1;
        3:TabIndex:=0;
      end;
      TabControl1Change(Sender);
    end;
end;
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  TabControlEdit.Text:='Tab '+IntToStr(TabControl1.TabIndex+1);
```

end;
end.

При натисканні кнопки **Next** находящейся в об'єкті PageControl, а також при щиглику на об'єкті RadioButton здійснюється перехід на нову сторінку (об'єкт TabSheet) об'єкта PageControl. Активна в даний момент сторінка об'єкта PageControl указується за допомогою об'єктів RadioButton.

При натисканні кнопки **Next** находящейся в об'єкті TabControl відбувається перехід на нову сторінку об'єкта TabControl, а також відображається ім'я активної сторінки в об'єкті Edit (див. мал. 1.1.3).

1.1.5.1. Компонент TPageControl

Розглянутий у даному розділі компонентів спрощує створення многостраничных діалогових вікон з окремим набором елементів керування на кожній сторінці. Приклад додатка з використанням многостраничных компонентів зображений на мал. 1.1.3. Ця програма відображає об'єкти компонента TPageControl (ліворуч) і TTabControl (праворуч). Зверніть увагу на те, що сторінки об'єкта компонента TPageControl можна створювати під час розробки, а сторінки компонента TTabControl — не можна. Справа тут у тім, що сторінка об'єкта компонента TPageControl — це окремий об'єкт, якому можна вибрати у вікні Object Inspector і розмістити в ньому інші об'єкти. А сторінки TTabControl — ілюзія, створювана шляхом програмної зміни його вмісту під час виконання.

Щоб ближче познайомитися з компонентом, виконаєте наступне.

1) Створіть новий додаток. На вкладці Win32 палітри знайдіть компонент TPageControl, клацніть на його кнопці й усередині форми. Тільки що створений об'єкт виглядає, як порожня панель.
2) Щоб створити сторінку об'єкта компонента TPageControl, клацніть усередині його панелі правою кнопкою миші. З раскрывшегося меню виберіть команду **New Page**. У результаті буде створений новий об'єкт компонента TTabSheet і доданий до набору таких же об'єктів, включених в об'єкт компонента TPageControl. Його сторінка з'явиться на екрані.
3) Коли ви вибираєте сторінку, клацнувши на її корінці, Object Inspector відображає властивості її контейнера, об'єкта PageControl1. Якщо ж ви хочете відобразити властивість TabSheet1 або TabSheet2, клацніть усередині відповідної сторінки.
4) Тепер справа за елементами керування. Щоб помістити на сторінку якої-небудь з компонентів, виберіть неї, клацніть на потрібній кнопці палітри й усередині сторінки. На сторінках PageControl-об'єкта можна розміщати об'єкти практично будь-яких компонентів, до приклада TLabel, TStringGrid або TDateTimePicker. Щоб до них звернутися, використовуйте їхні імена (Button1, CheckBox1 і т.п.), як якби вони розташовувалися безпосередньо у формі (на об'єкт-контейнер посилатися не потрібно).

1.1.5.2. Властивості компонента TPageControl

Для настроювання об'єктів компонента TPageControl у вашому розпорядженні мається цілий ряд їхніх властивостей. Скориставшись об'єктом, створеним у попередньому прикладі, спробуйте змінити значення наступних властивостей.

- ***ActivePage***. Це ім'я об'єкта компонента TTabSheet, що буде відображений при відкритті форми. Ця ж властивість служить і для зміни сторінки програмним шляхом під час виконання (для приклада звернетеся до процедури GoToSheet у прикладі 1.1.5).
- ***DockSite***. Якщо ця властивість має значення True, об'єкт може служити стикувальною станцією для інших об'єктів. (Докладніше про цю можливість Delphi розповідається в розділі “Створення стикувальних елементів інтерфейсу”.)
- ***Hint***. Це текст підказки, що з'являється, коли користувач затримує покажчик миші на об'єкті. Дана властивість використовується спільно з властивістю ShowHint, для якого потрібно вибрати значення True. Текст підказки, що ви тут уведете, буде відображатися

незалежно від обраної сторінки.

- ***HotTrack***. Ця властивість дозволяє створити спеціальний візуальний ефект: коли користувач переміщає покажчик миші по корінцях сторінок, той з них, що знаходиться під покажчиком, тьмяніє, щоб показати, що він активний. Для створення цього ефекту привласніть властивості HotTrack значення True.
- ***Images***. У цій властивості необхідно вказати ім'я об'єкта ImageList, що містить піктограми, для можливості розміщення піктограм на корінцях сторінок. Властивість використовується разом із властивостями ImageIndex сторінок, де вказуються індекси необхідних піктограм.
- ***MultiLine***. Ця властивість дозволяє виводити назви корінців у декількох рядах, якщо сторінок занадто багато й в одному рядку вони не містяться. Якщо дана властивість має значення False, а корінці сторінок не містяться в одному рядку, то з'являються горизонтальні стрілки прокручування. Але багато рядків використовувати зручніше.
- ***OwnerDraw.*** Якщо установити для цієї властивості значення True, то ви самі будете відповідати за відображення на кожній сторінці тексту або графіки (або і того, і іншого), для чого потрібно буде написати програмний код. Цей код варто помістити в оброблювач події OnDrawTab, якому як параметр передаються координати області, у якій можна малювати.
- ***ScrollOpposite.*** Ця властивість визначає, куди при виборі однієї з вкладок будуть переміщатися інші, якщо них багато. Нехай, наприклад, в об'єкті PageControl мається два ряди по трьох вкладки в кожнім. Тоді, якщо ScrollOpposite має значення True, при виборі вкладки в одному ряді, інший ряд переміщається назад.
- ***ShowHint.*** Якщо ця властивість має значення True, при затримці покажчика миші на об'єкті на екран виводиться маленьке спливаюче вікно підказки. Якщо ви хочете, щоб текст підказки залежав від обраної сторінки, установите для цієї властивості значення False і скористайтеся парою властивостей ShowHint і Hint кожної сторінки окремо.
- ***Style.*** Ця властивість визначає, як будуть виглядати корінці сторінок об'єкта. Корінці можуть виглядати так, як у папок (tsTabs), як кнопки, виконані в стилі Windows (tsButtons), або взагалі не мати обмежуючих ліній, у той час як обраний корінець злегка утоплений (tsFlatButtons). Якщо ви віддасте перевагу стилеві tsFlatButtons, установите для властивості HotTrack значення True — і при переміщенні покажчика миші по корінцях останні будуть підніматися.
- ***TabPosition.*** Від цієї властивості залежать, угорі (tpTop) або внизу (tpBottom) вікна будуть розташовуватися корінці сторінок. Оскільки звичайно вони розташовані вгорі, саме там користувач і очікує них побачити. Тому не коштує без особливих на те причин змінювати пропоноване за замовчуванням значення цієї властивості.

Коли властивість Style об'єкта компонента TPageControl має значення tsButtons або tsFlatButtons, рамка навколо сторінки відсутній. Щоб неї намалювати, найпростіше помістити на кожну сторінку об'єкт компонента TPanel або TBevel, а всі інші об'єкти розташувати поверх нього.

1.1.5.3. Компонент TTabSheet

Кожна сторінка об'єкта PageControl є об'єктом класу TTabSheet. Як уже говорилося, єдиним способом створення сторінок є щиглик на об'єкті правою кнопкою миші і вибір з контекстного меню команди New Page. Щоб у режимі конструювання форми вибрати об'єкт компонента TTabSheet, потрібно клацнути спочатку на корінці потрібної сторінки, а потім — усередині самої сторінки (нижче корінця, якщо корінці розташовані зверху). Після цього у вікні Object Inspector можна настроїти властивості цього об'єкта.

Існує спосіб одночасної установки значень властивостей для декількох сторінок, хоча і не занадто зручний. Щоб вибрати кілька сторінок відразу, утримуючи клавішу <Shift>, по черзі

клацніть на корінці кожної сторінки, а потім усередині неї (тобто послідовно виберіть кожну сторінку). Після вибору останньої сторінки знову клацніть на корінці кожної з них, щоб скасувати виділення їхнього контейнера, і тільки після цього відпустите клавішу <Shift>. Потім привласніть потрібні значення загальним властивостям виділених сторінок, наприклад значення crCross – властивості Cursor.

1.1.5.4. Властивості компонента TTabSheet

У цьому розділі описуються властивості компонента TTabSheet, що ви, імовірно, будете змінювати. Щоб випробувати них у справі, помістите у форму об'єкт PageControl, а потім додайте в нього кілька сторінок. (Для додавання сторінки потрібно клацнути на об'єкті компонента TPageControl правою кнопкою миші і вибрати з контекстного меню команду **New Page**. Для вибору сторінки потрібно спочатку клацнути на її корінці, а потім — усередині неї.)

- ***BorderWidth***. Позитивне ненульове значення цієї властивості визначає товщину лінії, що обмежує сторінку.
- ***Caption***. Це текст корінця сторінки. Ширина корінця буде підібрана автоматично відповідно текстові. Якщо властивість OwnerDraw об'єкта PageControl має значення True, підписати сторінку прийдеться самостійно, використовуючи програмний код.
- ***Cursor***. Ця властивість визначає форму, що покажчик миші буде приймати усередині сторінки. Його значення можна вибрати зі списку.
- ***Hint***. Це текст спливаючої підказки, що з'являється, коли покажчик миші затримується на сторінці. Щоб ця підказка відображалася, властивість ShowHint повинна мати значення True.
- ***ImageIndex***. Ця властивість використовується для розміщення піктограм на корінцях сторінок поруч з текстом корінця сторінки хранящимся у властивості Caption. Для розміщення піктограм необхідно включити у форму об'єкт ImageList і помістити в нього необхідні піктограми. Кожна піктограма в об'єкті ImageList має свій індекс. Для відображення піктограм на корінцях необхідно привласнити індекси необхідних піктограм властивостям ImageIndex об'єктів TabSheet, а також вказати ім'я об'єкта ImageList, що містить піктограми, у властивості Images об'єкта PageControl.
- ***PageIndex***. Ця властивість являє собою номер сторінки (нумерація починається з нуля). Його значення встановлюється автоматично. Якщо в режимі конструювання форми змінити номер однієї сторінки, відповідно зміняться і номера інших сторінок. Зробити це можна, щоб розташувати сторінки в потрібному порядку, але, утім, значення даної властивості змінюють рідко.
- ***PopupMenu***. Це ім'я об'єкта компонента TPopupMenu, якому можна зв'язати зі сторінкою, попередньо помістивши його у форму (компонент входить у категорію Standard). Якщо користувач клацне на сторінці правою кнопкою миші, з'явиться контекстне меню, кероване цим об'єктом. Можна зв'язати той самий об'єкт меню з усіма сторінками, а можна створити для кожної з них своє меню з окремим набором команд (можливість, що буває дуже важливої).
- ***ShowHint***. Установите для цієї властивості значення True, щоб при затримці на сторінці покажчика миші з'являлася спливаюча підказка. Разом із властивістю Hint властивість ShowHint дозволяє відображати для кожної сторінки власний текст підказки. Щоб для всіх сторінок відображалася та сама спливаюча підказка, установите для їхньої властивості ShowHint значення False, а для властивості ParentShowHint — True. Потім виберіть об'єкт PageControl, як значення властивості Hint уведіть текст підказки, а для властивості ShowHint укажіть значення True.
- ***TabVisible***. Ця властивість керує видимістю сторінки. Щоб у процесі роботи програми видалити на час сторінку, привласніть йому значення False. При цьому властивості сторінки PageIndex автоматично привласнюється значення -1. Сторінки, властивість

яких TabVisible має значення False, не можна вибирати за допомогою миші. Тому для їхнього вибору користуйтеся списком, що розкривається, у вікні Object Inspector. Одержавши в такий спосіб доступ до властивостей сторінки, можна при бажанні знову зробити неї видимої.

1.1.5.5. Компонент TTabControl

Для створення многостраничного діалогового вікна найпростіше скористатися компонентом TTabControl, кнопка якого розташована на вкладці Win32 палітри. На відміну від більш складного компонента TPageControl, у нього немає окремих сторінок; їхня наявність тільки удаване. Виглядають ці об'єкти однаково: набір сторінок з корінцями, що вибираються за допомогою миші. Однак коли користувач вибирає сторінку, відобразити її вміст повинний написаний вами оброблювач цієї події.

Компонент TTabControl зручний для створення форм уведення даних, уміст компонентів яких залежить від обраної сторінки. Прикладом може служити програма PageTab. Праворуч в екранній формі розташовується об'єкт TabControl, а в ньому — об'єкт Edit (див. мал. 1.1.3). При виборі сторінок об'єкта TabControl уміст текстового поля буде при цьому змінюватися, а саме текстове поле буде залишатися на місці. Однак помнете, що насправді перед вами та сама сторінка (тобто той самий об'єкт) з тими самими елементами. Для зміни тексту полів у демонстраційній програмі використовується наступна процедура:

```
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  TabControlEdit.Text:='Tab '+IntToStr(TabControl1.TabIndex+1);
end;
```

Приведена процедура демонструє технологію роботи з TabControl. Вона викликається для обробки події OnChange, коли користувач вибирає сторінку. Щоб визначити, яка зі сторінок обрана, програма звертається до властивості TabIndex об'єкта TabControl1. Нумерація сторінок починається з нуля, тому програма додає до номера сторінки одиницю і разом з текстом 'Tab ' вносить його в текстові поля (шляхом присвоєння властивості значення Text). Узявши цю технологію за основу, можна писати власну програму, що, наприклад, буде відображати на кожній сторінці текст відповідного файлу.

Щоб вибрати сторінку TabControl-об'єкта в режимі конструювання форми, потрібно скористатися його властивістю TabIndex. Мишею зробити це не удасться.

1.1.5.6. Властивості компонента TTabControl

Ми не будемо окремо перелічувати властивості компонента TTabControl, оскільки вони ті ж, що й у компонента TPageControl (читайте розділ цієї глави “Властивості компонента TPageControl”). Єдиною відмінністю є властивість Tabs, що у випадку TabControl являє собою об'єкт класу TStrings (простіше говорячи, список рядків — назв сторінок). Подвійний щиглик на поле цієї властивості дозволяє відкрити вікно String list editor, що служить у Delphi для редагування списків рядків. Кожен рядок у цьому вікні являє собою назву сторінки, що буде написано на її корінці. Просто введіть їх стільки, скільки сторінок повинне бути у вашого об'єкта.

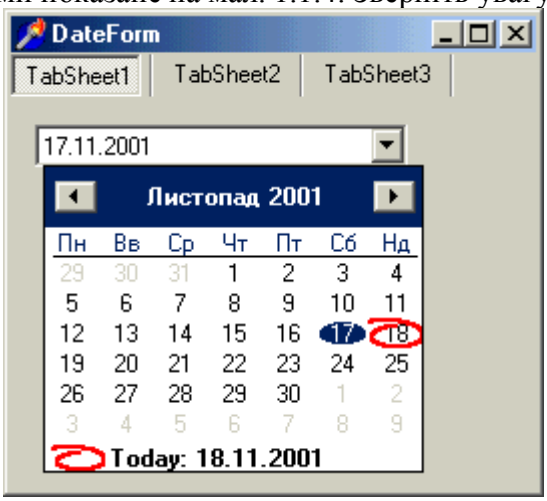
1.1.6. Обмеження розміру вікна

У попередніх версіях Delphi обмежити розмір вікна було неможливе (або, принаймні, нелегко). Часто це приводило до проблем, оскільки, змінюючи розмір вікна, користувач міг зробити невидимої частину його компонентів. Єдине, що можна було зробити, — узагалі заборонити зміна розмірів вікна, але з погляду зручності роботи користувача це обмеження не виправдане.

У Delphi 4 з'явилася нова властивість класу TControl — Constraints (об'єкт класу TSizeConstraints). З його допомогою можна задавати максимальну і мінімальну висоту і ширину більшості компонентів, що успадковують клас TControl. Є ця властивість і в самої форми.

Користуватися властивістю Constraints просто. Створіть новий додаток і у вікні Object Inspector знайдіть цю властивість для об'єкта Forml. Двічі клацніть на символі “+” ліворуч від імені властивості. Відкриється список з чотирьох його складових: MaxHeight, MaxWidth, MinHeight і MinWidth. За замовчуванням усі вони мають значення 0. Уведіть значення в пикселях (позитивні), що обмежують припустиму висоту (height) і ширину (width) об'єкта.

Для приклада розглянемо деякий додаток. У ньому створюється об'єкт компонента TPageControl, мінімальний розмір якого обмежений, завдяки чому користувач не може настільки зменшити розмір діалогового вікна, що його елементи виявляться недоступними. Розгорнути вікно на весь екран, щоб воно заслонило всю іншу інформацію, теж не можна. Вікно цієї програми показане на мал. 1.1.4. Зверніть увагу, що об'єкт PageControl займає весь простір вікна.



Малюнок 1.1.4 - Обмеження розміру вікна не дозволяє зменшити або збільшити його занадто сильно

Якщо максимальний розмір вікна обмежений значеннями властивості Constraints, то, клацнувши на кнопці максимізації вікна, можна збільшити його розмір до максимально припустимого і перемістити вікно в лівий верхній кут екрана. Це може смутити користувачів, що звикли до іншої дії цієї кнопки, тому такий ефект краще задокументувати в інструкції з експлуатації додатка.

Щоб самостійно відтворити демонстраційну програму, виконаєте наступне.

1) Помістите в порожню форму об'єкт компонента TPageControl. Клацніть усередині нього правою кнопкою миші і з раскрывшегося меню виберіть команду **New Page.** Додайте в такий спосіб кілька сторінок.
2) Щоб об'єкт PageControl1 займав усю площу форми привласніть його властивості Align значення alClient.
3) На сторінках створеного об'єкта розмістите кілька будь-яких елементів інтерфейсу. У прикладі на першій сторінці розташований об'єкт компонента TDateTimePicker.
4) Збільште вікно форми до бажаного розміру. Потім у вікні Object Inspector уведіть для властивостей MaxHeight і MaxWidth об'єкта Form1 ті ж значення, що Delphi привласнила властивостям Height і Width.
5) Тепер зменшите розмір вікна форми, наскільки вважаєте за можливе, щоб його елементи залишалися при цьому видні. У вікні Object Inspector для властивостей MinHeight і MinWidth об'єкта Form1 уведіть значення, що Delphi привласнила властивостям Height і Width.
6) Щоб корінці сторінок мали такий же вид як на мал. 1.1.4 привласніть властивості Style об'єкта PageControl1 значення tsFlatButtons.
7) Запустите програму, натиснувши клавішу <F9>. Спробуйте змінити розміри вікна (встановлені обмеження повинні діяти). Клацніть на кнопці максимізації вікна і переконаєтеся, що воно не стало більше заданого розміру.

Значення складові властивості Constraints можна змінювати і під час виконання програми.

Наприклад, можна включити в додаток опцію, що встановлює і скасовує обмеження розміру вікна.

Щоб написати процедуру обробки події, що відключають обмеження розмірів вікна в попередньому прикладі, додержуйтеся інструкцій.

1) Зі списку, що розкривається, у вікні Object Inspector виберіть об'єкт форми – Form1.
2) У цьому ж вікні клацніть на вкладці Events.
3) Двічі клацнувши на поле події OnConstrainedResize, створіть порожню процедуру його обробки. Помістите в неї програмний код із приклада 1.1.6.
4) Запустите програму, натиснувши клавішу <F9>. Спробуйте змінити розміри вікна. Тепер воно повиннео вільно розвертатися на весь екран, але можливість його зменшення як і раніше повинна залишатися обмеженої.

Приклад 1.1.6 - Вставка в процедуру обробки події OnConstrainedResize

```
procedure TForm1.FormConstrainedResize(Sender: TObject;
var MinWidth, MinHeight, MaxWidth, MaxHeight: Integer);
begin
  MaxWidth := Screen.Width;
  MaxHeight := Screen.Height;
end;
```

Подія OnConstrainedResize визначена тільки для класів TForm, TScrollBox і TPanel. Скористайтеся їм, щоб змінити задані при конструюванні форми обмеження. Можна, зокрема, як у даному прикладі, звернутися до властивостей глобального об'єкта Screen, щоб з'ясувати поточні розміри екрана і дозволити вікну займати його цілком.

1.1.7. Створення стикуючих елементів інтерфейсу

Як ви вже, імовірно, знаєте, Delphi, починаючи з 4-ої версії, дозволяє зістиковувати (dock) різні вікна. Щоб довідатися, як це працює, виберіть команду View(Project Manager і перетягнете за допомогою миші вікно, що відкрилося, усередину вікна редактора коду, до його правої або лівої границі, не відпускаючи кнопки. Коли границя вікна, що перетаскується, змінить форму, відпустите кнопку миші: вікно Project Manager пристикується зсередини до границі вікна редактори коду. При використанні монітора з високим дозволом, коли в границь вікна редагування залишається багато вільного місця, ця можливість особливо корисна. Щоб відокремити вікно від границі, просто перетягнете його в інше місце: ви побачите, як на визначеній відстані від границі відновиться його колишня форма.

Delphi дозволяє наділити такою можливістю і створювані вами елементи додатка. У процедре стикування об'єкт може відігравати роль:

- *стикувальної станції* (docking site);
- *стыкуемого елемента* (dockable control).

Стикувальною станцією, тобто об'єктом, до якого можна пристиковувати різні елементи керування, може бути будь-як об'єкт, що володіє властивістю DockSite. Ця властивість є в компонентів TForm, TPanel, TPageControl, TTabControl, TToolBar, TCoolBar і деяких інших. Об'єкти, що можуть відігравати роль стикувальної станції, повинні бути здатні виступати стосовно інших об'єктів як контейнери, тому такі елементарні компоненти, як кнопки і прапорці, для цієї ролі не підійдуть.

Що стосується стыкуемых елементів, те підходящі для них компоненти повинні мати властивості DragKind і DragMode. Є компоненти, що володіють тільки одним з цих двох властивостей, але вони не підійдуть. Щоб об'єкт можна було пристиковувати до інших об'єктів, у нього повинні бути обоє властивості.

Прикладами компонентів, що не можуть виступати в якості стыкуемых, є компоненти категорії Win3.1, а також ті, котрі під час виконання програми не мають візуального представлення, наприклад TTimer і TMainMenu.

Найчастіше як стикувальні станції використовуються об'єкти компонента TPanel. До них пристиковуються об'єкти, що містять інші елементи інтерфейсу, наприклад об'єкти компонентів TToolBar, TCoolBar і TPageControls.

Можливість стикування об'єктів з'явилася в Delphi 4 завдяки змінам, внесеним у реалізацію класів TControl і TWinControl. Клас об'єкта, що грає роль стикувальної станції, повинний бути похідним від класу TWinControl, а клас об'єкта стыкуемого елемента — від TControl. Оскільки і сам клас TWinControl успадковує TControl, багато стикувальних станцій можна, у свою чергу, пристиковувати до інших. Прикладом є об'єкт компонента TToolBar, у якому можна розмішати інші вікна, а можна його самого пристикувати до границі якого-небудь вікна (ви напевно не раз так надходили з панелями інструментів у різних додатках).

1.1.7.1. Створення стикуючої станції

Стикувальною станцією може служити будь-як компонент, що має властивість DockSite. Просто привласніть цій властивості об'єкта значення True, і він зможе приймати інші вікна (називані також стикувальними клієнтами). Як правило, як стикувальні станції виступають об'єкти компонентів TForm, TPanel або TCoolBar, але бувають і інші випадки — досить, щоб у компонента була властивість DockSite.

Найчастіше властивості стикувальної станції AutoSize привласнюється значення True. Коли обоє властивості об'єкта, і DockSite, і AutoSize, мають значення True, об'єкт під час виконання не видний доти, поки до нього не буде пристыкован хоч один елемент. Користувачеві нема чого бачити саму стикувальну станцію, якщо єдине її призначення — служити контейнером для інших об'єктів, як це буває, наприклад, у випадку панелей інструментів.

1.1.7.2. Створення елементу що стикується

Як уже говорилося, якщо в об'єкта є властивості DragKind і DragMode, вона може бути стыкуемым елементом. Наприклад, їм може бути об'єкт компонента TToolBar: виберіть для його властивості DragKind значення dkDook, а для властивості DragMode — значення dmAutomatic. Якщо хочете керувати стикуванням самостійно (що може знадобитися в найрідших випадках), установите для властивості об'єкта DragKind значення dkDrag (це значення властивість має за замовчуванням) і запрограмуйте обробку подій OnDragOver, OnEndDrag і OnDragDrop. Якщо властивість DragMode має значення dmManual, для того щоб почати операцію перетаскування, потрібно викликати метод BeginDrag.

Щоб установити тип об'єктів, у які можна перетаскувати стыкуемый елемент керування зі стикувальних станцій, привласніть ім'я цього класу властивості елемента FloatingDockSiteClass. Як правило, це значення TMainForm — нащадок базового класу форм додатка, створюваний Delphi.

Література [1].

Тема 1.2. Розробка багатодокументних додатків

Розроблювачі додатків для Windows можуть скористатися можливостями багатодокументного інтерфейсу (MDI — Multiple Document Interface), відгуки про які неоднозначні — від замилування до категоричного неприйняття. На практиці більшість програмістів відносяться до цих засобів стримано, однак не слід вважати концепцію MDI настільки вуж невдалої. Її доцільно використовувати в тих випадках, коли необхідно забезпечити середовище для роботи одночасно з декількома документами, кожний з яких відображається у своєму власному вікні. Для таких додатків MDI є зручним у використанні стандартним інтерфейсом, програмування якого в середовищі Delphi не представляє особливої праці.

У цій главі порозуміється, як за допомогою екранних форм Delphi створити основне і дочірнє вікна інтерфейсу MDI. Додатково розглядаються зв'язані з цією технологією теми — створення за допомогою команд керування вікнами меню Windows, організація в модулі екранної

форми підкласу з метою створення дочірнього вікна, включення імен вікон у меню Windows і злиття меню основного і дочірнього вікон Windows.

1.2.1. Компоненти

Нижче описуються компоненти Delphi, призначені для створення MDI-додатків.

- ***TForm***. Це той же компонент, що використовується при створенні додатків з єдиним документом (SDI-додатків — Single Document Interface). У палітрі візуальних класів він відсутній. У MDI-додатках об'єкти класу TForm використовуються для створення основного і дочірнього вікон. У випадку застосування основного вікна властивість FormStyle цього класу повинне мати значення fsMDIForm. Для дочірніх вікон цій властивості варто привласнити значення fsMDIChild, попередньо створивши нове вікно, як зазначено в приведених у даній главі інструкціях, наприклад скориставшись командою File⇨Open.
  Категорія палітри: відсутній.
- ***TMainMenu***. Основне вікно кожного MDI-додатка повинне включати об'єкт компонента TMainMenu. Як правило, у MDI-додатках об'єкт MainMenu повинний включати елементи File (Файл) і Window (Вікно), хоча не існує ніяких обмежень на імена раскрывающихся меню. Дочірні вікна MDI-додатка також можуть містити об'єкти MainMenu, призначені для установки в рядку меню основного вікна.
  Категорія палітри: Standard.

1.2.2. Основи програмування MDI – додатків

Будь-який MDI-додаток включає три основних елементи.

- Екранна форма основного вікна MDI-додатка.
- Одна або більш форм дочірніх вікон MDI-додатка.
- Основне меню MDI-додатка.

На відміну від звичайної технології програмування в Windows, у Delphi для створення стандартної MDI-рамки і клієнтських вікон використовується один об'єкт форми. Класично, рамка вікна є видимим об'єктом, а клієнтське вікно — невидимим об'єктом, що керує виконанням глобальних операцій, що створюють дочірні вікна і здійснює обробку повідомлень. У додатках, створюваних засобами Delphi, також існують і рамка, і клієнтські вікна, але вони рідко використовуються безпосередньо. На практиці з рамкою і клієнтським вікном можна звертатися, як з одним вікном, представленим у програмі об'єктом екранної форми основного вікна.

Дочірні вікна документів також являють собою екранні форми, але на відміну від таких вікон, як AboutBoxdialog, що можуть бути вставлені в модуль, дочірні вікна MDI-додатків не можуть існувати незалежно від основного вікна. Вони можуть розташовуватися тільки в клієнтській області основного вікна. Якщо дочірнє вікно мінімізується, що відповідає піктограма відображається в клієнтській області основного вікна, а не на панелі задач Windows.

За винятком зазначених вище особливостей, основне і дочірнє вікна MDI-додатків нічим не відрізняються від екранних форм звичайних SDI-додатків. У MDI-додатках можна використовувати панелі інструментів, рядка стану, об'єкти компонентів і графічні елементи, а також застосовувати будь-як технології програмування, що існують у Delphi. Однак застосування MDI-технології найбільш виправдано у випадку створення додатків, призначених для одночасної обробки декількох документів.

Дуже часто многодокументный інтерфейс розуміється як система керування файлами, однак дочірні вікна додатки необов'язково повинні бути зв'язані з дисковими файлами. Концепція MDI може використовуватися і для створення многооконных додатків інших типів. У цій главі

термін документ визначає будь-як інформацію, що може бути відображена у вікні, а не тільки дані дискового файлу, що містить деякий документ.

1.2.2.1. Форма основного вікна MDI – додатка

Для створення основного вікна MDI-додатка виконаєте наступні дії.

1) Створіть новий додаток.
2) Привласніть властивості Name форми деяке значення, наприклад MainForm.
3) Привласніть властивості FormStyle форми значення fsMDIForm.
4) Збережете створений проект у новій папці. Привласніть створеному модулеві ім'я Main.pas і задайте для проекту відповідне ім'я (якщо ви мають намір виконувати всі пропоновані в цій главі приклади, привласніть проектові ім'я MDITest).

Властивість FormStyle може мати значення fsMDIForm тільки у випадку використання форми основного вікна і лише одне подібне вікно може бути присутнім у кожнім MDI-додатків. Щоб об'єкт форми вікна створювався автоматично, виберіть команду Project(Options і переконаєтеся, що на вкладці Forms значення MainForm є присутнім у списках Main form і Auto-created forms.

Це усе, що необхідно виконати для створення головного вікна MDI-додатка. Тепер можна перейти до наступного розділу, у якому описується створення дочірніх вікон і меню розроблювального додатка.

Для того щоб на практиці випробувати описувану в даній главі технологію, створіть новий додаток, вибравши команду File(New Application. Інший варіант - вибрати команду File(New, після чого в діалоговому вікні, що розкрилося, New Items вибрати на вкладці New шаблон Application. Не варто вибирати шаблон MDI Application на вкладці Projects цього вікна. На основі цього шаблона буде створений новий MDI-проект, що включає пропоновані за замовчуванням меню, форму дочірнього вікна і процедури обробки подій. Нижче в цій главі буде пояснено, як використовувати шаблон MDI-проекту, однак у демонстраційних цілях краще просто створити порожній одновіконний додаток, а потім виконати зазначені дії.

1.2.2.2. Додавання до MDI – додатка екранних форм дочірніх вікон

Будь-який MDI-додаток обов'язкове повинно мати, принаймні, одну екранну форму і модуль дочірнього вікна. Для додавання дочірнього вікна до нового MDI-додатка виконаєте наступні дії.

1) Створіть новий додаток, вибравши команду File(New Application. У вікні Object Inspector задайте для властивості Name об'єкта Form1 значення MainForm. Властивості FormStyle перейменованого об'єкта MainForm привласніть значення fsMDIForm.
2) Виберіть команду File(Save All (або клацніть на піктограмі Save All, що зображує набір дисків). Замість імені Unit1.pas введіть ім'я Main.pas і клацніть на кнопці Save. Зміните ім'я Project1.dpr на MDITest.dpr і ще раз клацніть на кнопці Save.
3) Виберіть команду File(New Form або клацніть на піктограмі New Form. У результаті буде створений новий об'єкт форми, що має за замовчуванням ім'я Form1. Оскільки ім'я попередньої форми було змінено на MainForm, знову створена форма знову одержить за замовчуванням ім'я Form1, а не Form2. Зміните розмір вікна форми Form1, що спростить надалі вибір форм (споконвічно ця форма має той же розмір, що і MainForm, і, імовірніше всього, цілком неї закриває).
4) У вікні Object Inspector зміните значення властивості Name об'єкта Form1 на ChildForm. Властивості FormStyle привласніть значення fsMDIChild.
5) Виберіть команду File(Save All (або клацніть на піктограмі Save All). У діалоговому вікні, що розкрився на екрані, вкажіть ім'я файлу для створюваного модуля дочірнього вікна. Зміните запропоноване за замовчуванням значення Unit1.pas на Child.pas і клацніть на кнопці Save.
6) Щоб розкрити діалогове вікно Project Options, виберіть команду Project(Options. У списку Auto-create forms виберіть об'єкт ChildForm і клацніть на кнопці з одиночною

стрілкою вправо. Об'єкт буде перенесений у список Available forms. Якщо не виконати цю операцію, то дочірнє вікно буде автоматично виводитися на екран безпосередньо при запуску створюваної програми. Це не суперечить ніяким існуючим правилам, однак подібні дії можуть викликати розгубленість у користувача, що очікує появи дочірніх вікон тільки в результаті вибору їм визначених команд, наприклад File(New або File(Open. У більшості MDI-додатків створення дочірніх вікон здійснюється не автоматично, а під контролем самої програми. Закрийте діалогове вікно Project Options.

Після виконання описаних вище дій буде створений кістяк оболонки MDI-додатка. Тепер варто підготувати програмний код, що забезпечує створення дочірнього вікна у відповідь на вибір користувачем визначених команд, наприклад File(New або File(Open. Як це зробити, описується в наступному розділі.

MDI-додаток може використовувати дочірні вікна різних типів. Якщо необхідно, просто повторите приведені вище пп. 3-5 для кожного з необхідних типів дочірнього вікна. Кожна нова форма повинна мати унікальне ім'я, однак властивості FormStyle усіх цих форм повинне бути привласнене значення fsMDIChild. Після створення всіх необхідних форм збережете проект, вибравши команду File(Save All. Збережете модуль кожної з дочірніх форм під унікальним ім'ям. Наприклад, якщо створені форми мають імена Child1Form і Child2Form, збережете їхні модулі у файлах з іменами Child1.pas і Child2.pas. Додаткову інформацію про створення форм дочірніх вікон різних типів можна знайти в розділі “Використання дочірніх вікон різних типів”, приведеному нижче в цій главі.

Якщо ви виконали всі перераховані вище операції, то на даний момент у знову створеній тапці проекту повинні бути присутнім наступні три модулі мовою Pascal.

- ***Child.pas***. Це модуль форми дочірнього вікна, що звичайно містить специфічний для даного типу документа програмний код і іншу задану для дочірнього вікна інформацію. Додатково дочірнє вікно може включати меню, призначене для установки в рядок меню основного вікна.
- ***Main.pas***. Це модуль форми основного вікна додатка. Сюди містяться всі процедури обробки подій для будь-яких поміщених у форму основного вікна об'єктів, а також для команд меню. Як правило, принаймні одна з процедур обробки подій даного модуля повинна бути призначена для створення екземплярів об'єктів дочірніх вікон додатка. Підготовка відповідних підпрограм докладно розглядається в розділі “Дочірні вікна” цієї глави.
- ***MDITest.dpr***. Це файл проекту. Вносити в нього які-небудь зміни при розробці MDI-додатків приходиться дуже рідко.

Створіть копію поточного вмісту папки проекту. Нею можна буде скористатися в демонстраційних цілях при розгляді різних технологій програмування в цій главі.

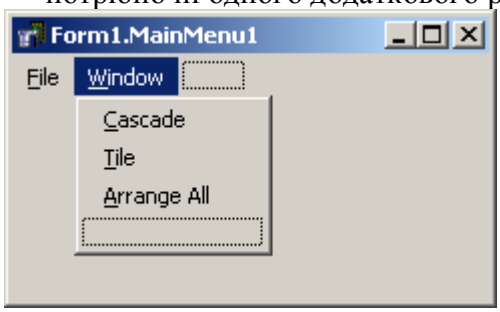
1.2.2.3. Створення основного меню MDI – додатка

Кожен MDI-додаток повинний включати об'єкт основного меню. Принаймні, одна команда в основному меню повинна бути призначена для створення об'єкта форми дочірнього вікна (як правило, це команда File(New). Приведена нижче послідовність дій призначена для включення основного меню в додаток MDITest, створене в попередньому розділі. Відкрийте (при необхідності) проект MDITest.dpr і виконаєте наступні дії.

1) Помістите у вікно MainForm об'єкт компонента TMainMenu, розташованого на вкладці Standard палітри. (Попередньо переконаєтеся, що обрано саме форму MainForm, а не форма ChildForm.) У вікні MainForm двічі клацніть на об'єкті MainMenu — і на екрані розкриється вікно Menu Designer Delphi.
2) Уведіть із клавіатури значення **&File** (амперсанд, поміщений перед буквою F, указує клавішу швидкого виклику даної команди). Натисніть клавішу <Enter>, і проектувальник меню створить меню File. Клацніть у вікні Menu Designer на меню File, і у вікні Object Inspector буде відображений об'єкт класу TMenuItem (з ім'ям File1).

Привласніть його властивості Name значення FileMenu замість існуючого значення File1.

3) За допомогою Menu Designer помістите в меню File пункти **&New**, **&Open...** і **&Save...**. Для цього клацніть під заголовком меню File і введіть ім'я відповідної команди. Для кожної команди буде створений екземпляр об'єкта TMenuItem. У прикладі для цієї глави використовуються прийняті за замовчуванням імена об'єктів New1, Open1 і Save1, однак при необхідності кожний з них можна перейменувати, для чого варто вибрати його у вікні Menu Designer і змінити у вікні Object Inspector значення властивості Name обраного об'єкта. Тепер меню містить команди File⇨New, File⇨Open і File⇨Save — мінімальний набір команд, необхідний для будь-якого MDI-додатка. (У розділі "Створення екземплярів об'єктів дочірніх вікон", приведеному нижче в цій главі, пояснюється, як підготувати програмний код, що забезпечує створення об'єктів дочірніх вікон, а також відкриття і збереження документів при виборі користувачем відповідної команди меню.)
4) У вікні Menu Designer створіть друге меню, що має ім'я **Window**. Для цього клацніть у вікні Menu Designer правее меню File і введіть із клавіатури значення **&Window**. Виберіть у Menu Designer знову створене меню Window і для відображеного у вікні Object Inspector об'єкта класу TMenuItem заміните прийняте за замовчуванням значення Window1властивості Name значенням WindowMenu.
5) Знову поверніться у вікно Menu Designer і клацніть нижче меню Window. Створіть у цьому меню три команди — **&Cascade**, **&Tile** і **&Arrange All**. Як і у випадку використання команд меню File, можна при необхідності вибрати кожний зі знову створених об'єктів пунктів меню і змінити їхнє значення властивості Name. Однак у демонстраційному прикладі цієї глави імена даних об'єктів залишені такими, якими вони були встановлені Delphi за замовчуванням: Cascade1, Tile1 і Arrange1. Про те, як написати процедури обробки для цих і інших команд меню Window, можна довідатися з розділу "Використання команд меню Window", приведеного нижче в цій главі. Вікно Menu Designer, що містить знову створені меню, показане на мал. 1.2.1.
6) Закрийте вікно Menu Designer або перемістите його убік, а потім виберіть вікно проектування екранної форми MainForm (для пошуку цього вікна натисніть клавішу <F12> або виберіть команду View(Project Manager). У вікні Object Inspector привласніть властивості WindowMenu об'єкта MainForm значення WindowMenu. Для цього клацніть на стрільці, розташованої в поле значення властивості WindowMenu екранної форми, і виберіть значення WindowMenu у списку всіх об'єктів класу TMenuItem, доступних у даній формі. Це забезпечить у додатку автоматичне включення заголовків усіх відкритих дочірніх вікон у зазначене меню і не буде потрібно ні одного додаткового рядка програми.



Малюнок 1.2.1 - Вікно Menu Designer Delphi з об'єктами меню, необхідними для будь-якого MDI-додатка

Безумовно, набір команд у меню будь-якого додатка визначається призначенням цього додатка. Однак будь-який MDI-додаток, як мінімум, повинне включати команди для створення і відкриття дочірніх вікон документів, а також меню, звичайно іменоване Window і призначене для відображення списку імен відкритих дочірніх вікон.

Властивості WindowMenu екранної форми як значення можна привласнювати імена об'єктів меню тільки самого верхнього рівня. Іншими словами, це повинніо бути імена об'єктів класу TMenuItem, що представляють ті елементи меню, що розташовані безпосередньо в рядку головного меню форми. Не можна призначати даній властивості імена об'єктів, що представляють окремі команди меню, наприклад Openl або Savel.

Екранні форми дочірніх вікон також можуть містити об'єкти класу TMainMenu. Коли користувач розкриє нове дочірнє вікно, команди меню цього вікна автоматично доповнять меню головного вікна у відповідності зі значенням властивості GroupIndex об'єкта меню цього дочірнього вікна. Докладно дана тема буде розглядатися в розділі "Об'єднання меню", приведеному нижче в цій главі.

1.2.2.4. Доступ до дочірніх вікон

Доступ до дочірніх вікон забезпечують три властивості компонента TForm, який варто застосовувати в більшості MDI-додатків. У даній главі приведена безліч прикладів використання кожного з цих властивостей.

- ***ActiveMDIChild***. Це посилання на поточне активне дочірнє вікно. Якщо дочірні вікна не існують, значення властивості ActiveMDIChild дорівнює nil. Дана властивість являє собою посилання на об'єкт класу TForm, отже, буде потрібно перетворити його в клас дочірнього вікна, наприклад TChildForm.
- ***MDIChildCount***. Це ціле, що містить лічильник дочірніх вікон, що належать основному вікну даного MDI-додатка. Якщо дочірні вікна не існують, значення властивості MDIChildCount дорівнює нулеві.
- ***MDIChildren***. Це масив посилань на об'єкти класу TForm, що представляють усі дочірні вікна, що належать основному вікну даного MDI-додатка. Вираження MDIChildren[0] визначає перше дочірнє вікно в масиві, а вираження MDIChildren [MDIChildCount-1] — останнє дочірнє вікно.

Якщо властивість ActiveMDIChild має значення nil або властивість MDIChildCount дорівнює нулеві, то дочірні вікна в даний момент відсутні. У цьому випадку програма не повинна виконувати звертання до масивові MDIChildren, хоча подібне звертання необов'язкове приведе до виникнення виняткової ситуації. Говорячи точніше, якщо властивість MDIChildCount дорівнює нулеві, то значенням вираження MDIChildren[0] буде nil.

1.2.3. Дочірні вікна

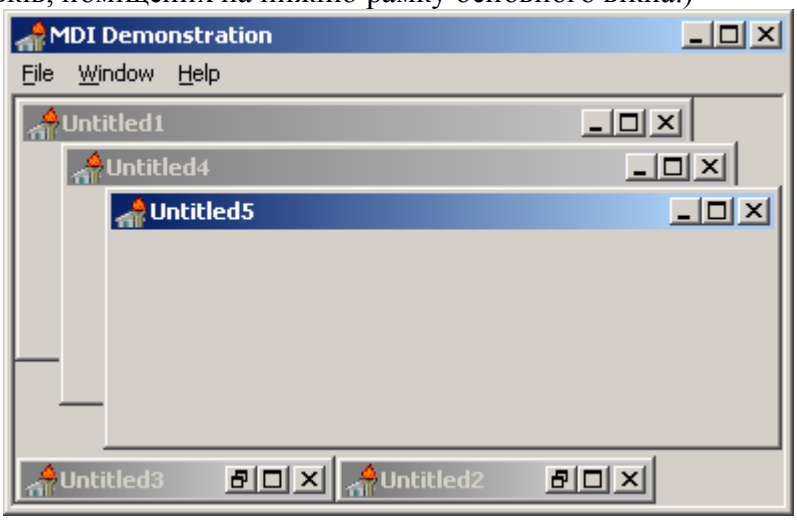
У MDI-додатках керування дочірніми вікнами здійснюється практично так само, як основним вікном SDI-додатка. Дочірні вікна можуть містити об'єкти компонентів, наприклад меню і кнопки. Як правило, у кожнім дочірнім вікні відображається деякий документ — скажемо, текстовий файл або растрове зображення. Користувач має можливість упорядкувати дочірні вікна каскадом або у виді мозаїки, а також мінімізувати них. Дочірні вікна можуть містити панелі інструментів і рядка станів, однак традиційно подібні об'єкти керування містяться тільки в основне вікно додатка.

Крім того, при необхідності активізація дочірнього вікна може супроводжуватися об'єднанням команд меню дочірнього й основного вікон MDI-додатка. Подібна можливість особливо зручна в тих випадках, коли в додатку використовуються різні типи дочірніх вікон, у кожнім з яких визначений особливий набір команд меню. У наступних розділах обговорюються згадані вище методи роботи з дочірніми вікнами, а також перелічуються всі існуючі оболонки MDI-додатків, якими можна скористатися при написанні нових програм.

1.2.3.1. Робота з дочірніми вікнами одному типові

Розглянемо як приклад додаток MDIDemo. Воно являє собою MDI-додаток, у якому використовуються дочірні вікна одного типу. Ця програма демонструє методи програмування типових команд меню, зокрема – File(New, File(Open, Window(Cascade і Window(Tile. Вид вікна

цього додатка з декількома відкритими в ньому дочірніми вікнами показаний на мал. 1.2.2. (У середовищі Windows 95, 98 і NT мінімізовані дочірні вікна представляються у виді їхніх заголовків, поміщених на нижню рамку основного вікна.)



Малюнок 1.2.2 - Додаток MDIDemo демонструє методи програмування MDI-додатків, що використовують дочірні вікна одного типу

У прикладі 1.2.1 приведений текст модуля Child.pas додатка MDIDemo. У цьому демонстраційному прикладі в дочірніх вікнах реальні дані не відображаються, тому даний модуль дуже простий. Однак він містить усі необхідні процедури, що повинні бути присутнім у типовому дочірнім вікні.

Приклад 1.2.1 – Модуль Child.pas додатка MDIDemo

```
unit Child;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

type
  TChildForm = class(TForm)
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    {Оголошення закритих (private) членів.}
  public
    {Оголошення загальнодоступних (public) членів. }
    procedure LoadData(const FileName: String); virtual;
    procedure SaveData(const FileName: String); virtual;
  end;

var
  ChildForm: TChildForm;

implementation

  {$R *.DFM}

procedure TChildForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
```

```
end;

procedure TChildForm.LoadData(const FileName: String);
begin
  ShowMessage('LoadData from ' + FileName);
  Caption := Lowercase(FileName);
end;
procedure TChildForm.SaveData(const FileName: String);
begin
  ShowMessage('SaveData to ' + FileName);
  Caption := LowerCase(FileName);
end;

end.
```

Модуль дочірнього вікна може включати процедури для завантаження і збереження даних документа. Ці процедури, названі в приведеному модулі LoadData і SaveData, оголошені як загальнодоступні методи класу TChildForm. Оголошення їх загальнодоступними викликане необхідністю забезпечити доступ до них за допомогою команд Open і Close в основному модулі.

Крім того, у приведеному прикладі процедури LoadData і SaveData з'являються як віртуальні. Завдяки цьому в іншому модулі може бути оголошений новий клас, похідний від класу TChildForm, у якому дані процедури будуть перевизначені. Наприклад, у програмі можна створити іншу форму дочірнього вікна, клас якої буде створений на базі класу TChildForm замість звичайно використовуваного для цих цілей класу TForm. Тут можна буде визначити іншу реалізацію процедур LoadData і SaveData, призначених для читання і запису специфічних для цього модуля даних. Подібний метод спадкування класу називається створенням підкласу форми і докладно розглядається нижче в цій главі.

Для кожного типу дочірнього вікна варто завжди надавати процедуру обробки події OnClose. Привласніть параметрові Action значення caFree, що забезпечить у додатку автоматичну ліквідацію об'єкта дочірнього вікна після закриття цього вікна користувачем. Якщо параметрові Action не привласнити значення caFree, закриття користувачем вікна викликає його мінімізацію в клієнтській області основного вікна.

На додаток до події OnClose може знадобитися написати процедуру обробки події OnCloseQuery, що буде виводити вікно з повідомленням про те, що користувач закриває вікно документа, не виконавши попередньо його збереження. У цій процедурі привласніть параметрові CanClose значення False, якщо необхідно запобігти закриття вікна, або ж значення True, якщо вікно документа може бути закрито. Основне вікно MDI-додатка не може бути закрито, а отже, і робота програми не може бути довершена доти, поки не будуть закриті всі дочірні вікна цього додатка. Дане досить корисна властивість забезпечує захист від можливої втрати даних, наприклад від утрати результатів внесення змін у дочірнім вікні, що при виході з програми випадково виявилася закрито іншим вікном.

У процедурах LoadData і SaveData додатка MDIDemo не виконується читання або запис яких-небудь реальних файлів, тому перевіряти роботу команд Open, Save і Save As можна без усяких побоювань. Для підтвердження того факту, що програма в потрібний момент викликає відповідну процедуру, у кожній з них процедура ShowMessage виводить вікно з повідомленням, що містить обране користувачем ім'я файлу. Як правило, коли в дочірнім вікні відображається уміст файлу деякого документа, у процедурах LoadData і SaveData необхідно буде помістити ім'я обраного файлу у властивість Caption об'єкта дочірньої форми. У результаті ім'я файлу і шлях до нього будуть відображені в рядку заголовка вікна.

Модифікувати додаток MDIDemo так, щоб у ньому виконувалися читання і запис реальних даних, зовсім нескладно. Наприклад, помістите у форму ChildForm об'єкт Memo (якщо це вікно в даний момент на екрані не відображається, відкрийте файл проекту MDIDemo.dpr,

виберіть команду View(Project Manager, клацніть на знаку “плюс” елемента Child і двічі клацніть на елементі ChildForm). Привласніть властивості Align знову створеного об'єкта Memo1 значення alClient — і зображення цього об'єкта займе всю клієнтську область дочірнього вікна. Вкажіть у відповідній властивості ім'я необхідного шрифту, видалите значення Memo1 у властивості Lines (для відкриття вікна редактора клацніть на кнопці з многоточием) і привласніть властивості ScrollBars значення ssBoth.

Для зміни тексту модуля ChildForm відкрийте за допомогою Project Manager файл Child.pas і перейдіть у вікні редактора коду на відповідну вкладку. Замість показаного в прикладі 1.2.1 оператора ShowMessage уставте приведені нижче операторы, що забезпечують читання і запис текстових файлів:

{Замініте оператор ShowMessage у підпрограмі LoadData наступним оператором:}

Memo1.Lines.LoadFromFile(FileName);

{Замініте оператор ShowMessage у підпрограмі SaveData наступним оператором:}

Memo1.Lines.SaveToFile(FileName);

У головну (батьківську) форму необхідно вставити об'єкти діалогових вікон OpenDialog і SaveDialog, а також об'єкт головного меню MainMenu.

Тільки що був створений MDI-додаток текстового редактора. Текст основного модуля програми приведений у прикладах 1.2.2 1.2.4.

Для компіляції і запуску модифікованої програми натисніть клавішу <F9>. Для відкриття в новому додатку дочірнього вікна виберіть команду File(New або File(Open. У дочірнім вікні документа, що розкрилося, можна буде вводити або редагувати текст. Для збереження виконаних змін виберіть команду File(Save або File(Save As.

Внеся в додаток MDIDemo приведені вище зміни, можна перетворити його в цілком працездатний додаток, причому зміни у файлах збережуться. Якщо ви хочете усього лише випробувати нову технологію, попередньо збережете копії будь-яких файлів, що будуть редагуватися.

Вхідний до складу імені файлу досить довгий шлях до нього може виглядати в заголовку вікна досить неестетично, тому краще скористатися функцією ExtractFileName для включення в рядок заголовка вікна одного лише імені файлу, без повного шляху. З цією метою необхідно в декларацію uses модуля додати ім'я SysUtil і оголосити перемінну string у класі TChildForm. Привласніть цієї перемінної строкове значення повного шляху до обраного користувачем файлові, а властивості Caption вікна привласніть результат виклику функції ExtractFileName. Докладну інформацію про цій і інших файлових функціях можна знайти в розділі “File Management routines” інтерактивної довідкової системи Delphi.

1.2.3.2. Створення екземплярів об’єктів дочірніх вікон

У прикладі 1.2.2 приведений приклад оголошення в додатку MDIDemo класу TMainForm і текст процедури обробки події OnClick меню File. У цьому фрагменті програми показано, як створювати нові екземпляри об'єктів класу TChildForm. Оскільки розміри основного модуля додатка MDIDemo досить великі, текст його доцільно розглядати частинами.

Приклад 1.2.2 – Процедура обробки події OnClick об'єкта меню File додатка MDIDemo

```
Unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, Menus, Child;

type
  TMainForm = class(TForm)
    MainMenu1: TMainMenu;
```

```
    FileMenu: TMenuItem;
    FileOpen: TMenuItem;
    FileSave: TMenuItem;
    FileSaveAs: TMenuItem;
    FileNew: TMenuItem;
    N1: TMenuItem;
    FileExit: TMenuItem;
    WindowMenu: TMenuItem;
    WindowCascade: TMenuItem;
    WindowTile: TMenuItem;
    WindowArrangeIcons: TMenuItem;
    N2: TMenuItem;
    WindowCloseAll: TMenuItem;
    WindowMinimizeAll: TMenuItem;
    HelpMenu: TMenuItem;
    HelpAbout: TMenuItem;
    OpenDialog: TOpenDialog;
    FileClose: TMenuItem;
    N3: TMenuItem;
    SaveDialog: TSaveDialog;
    procedure FileNewClick(Sender: TObject);
    procedure FileOpenClick(Sender: TObject);
    procedure FileCloseClick(Sender: TObject);
    procedure FileSaveClick(Sender: TObject);
    procedure FileSaveAsClick(Sender: TObject);
    procedure FileExitClick(Sender: TObject);
    procedure WindowCascadeClick(Sender: TObject);
    procedure WindowTileClick(Sender: TObject);
    procedure WindowArrangeIconsClick(Sender: TObject);
    procedure WindowMinimizeAllClick(Sender: TObject);
    procedure WindowCloseAllClick(Sender: TObject);
    procedure HelpAboutClick(Sender: TObject);
    procedure FileMenuClick(Sender: TObject);
    procedure WindowMenuClick(Sender: TObject);
  private
  {- Private declarations }
    procedure CreateChild(const Name: string);
  public
  {- Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

uses About;

{$R *.DFM}

const
```

```
  maxChildren = 10;  { Optional: No maximum required }

procedure TMainForm.CreateChild(const Name: String);
var
  Child: TChildForm;
begin
  Child := TChildForm.Create(Application);
  Child.Caption := Name;
end;

procedure TMainForm.FileNewClick(Sender: TObject);
begin
  CreateChild('Untitled' + IntToStr(MDIChildCount + 1));
end;

procedure TMainForm.FileOpenClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    CreateChild(Lowercase(OpenDialog.FileName));
    with ActiveMDIChild as TChildForm do
      LoadData(OpenDialog.FileName);
  end;
end;

procedure TMainForm.FileCloseClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    ActiveMDIChild.Close;
end;

procedure TMainForm.FileSaveClick(Sender: TObject);
begin
  if Pos('Untitled', ActiveMDIChild.Caption) = 1 then
    FileSaveAsClick(Sender)
  else with ActiveMDIChild as TChildForm do
    SaveData(Caption);
end;

procedure TMainForm.FileSaveAsClick(Sender: TObject);
var
  FExt: String;
begin
  with SaveDialog do
  begin
    FileName := ActiveMDIChild.Caption;
    FExt := ExtractFileExt(FileName);
    if Length(FExt) = 0 then
      FExt := '.*';
    Filter := 'Files (*' + FExt + ')|*' + FExt;
    if Execute then
```

```
    with ActiveMDIChild as TChildForm do
      SaveData(FileName);
  end;
end;

procedure TMainForm.FileExitClick(Sender: TObject);
begin
  Close;
end;
```

Процедура CreateChild, оголошена як закритий член класу TMainForm, призначена для створення екземплярів об'єктів класу TChildForm. Закриті члени класу доступні тільки для методів, що належать даному класові. Це дає впевненість, що підпрограми інших модулів не будуть мати можливості випадково викликати процедуру або функцію даного класу, що здійснює деякі критичні дії. У процедурі CreateChild показано, як створюється об'єкт форми в процесі виконання програми. Для цього необхідно оголосити перемінну з типом відповідного класу:

```
var
  Child: TChildForm;
```

Потім у тілі процедури необхідно викликати метод Create даного класу, указавши як аргумент значення Application. Знову створений об'єкт повинний бути призначений оголошеної вище перемінної в якості її значення:

```
Child := TChildForm.Create(Application);
```

Причина передачі методові Create аргументу Application замість MainForm полягає в тому, що аргумент Application представляє рамку вікна MDI-додатка, що є реальним вікном, що містить створюване дочірнє вікно, тоді як MainForm — це безпосередній батько дочірніх вікон.

Процедурою обробки команди File(New є процедура FileNewClick, що викликає метод CreateChild. Процедура обробки команди File(Open виконує цієї ж дії, але додатково викликає і загальнодоступний метод LoadData модуля Child.pas. Усе вищесказане є лише пропонованим вашій увазі прикладом конкретної технології створення дочірніх вікон і завантаження в них даних з файлів; кожен розроблювач вільний створювати програмні модулі так, як він вважає потрібним.

Властивість ActiveMDIChild об'єкта MainForm можна використовувати для виконання операцій з поточним дочірнім вікном. Якщо не існує жодного дочірнього вікна, то ця властивість містить значення nil; дана умова завжди варто перевіряти, приступаючи до роботи з властивістю. Наприклад, у тестовій програмі, у підпрограмі обробки події File⇨CIose (процедурі FileCloseClick), закриття активних дочірніх вікон виконується за допомогою наступних операторів:

```
if ActiveMDIChild <> nil then ActiveMDIChild.Close;
```

Ніколи не використовуйте операторы, що виконують подібні дії, без перевірки зазначеної умови. Якщо жодного з дочірніх вікон не існує, після виконання приведеної нижче команди виникне виняткова ситуація (причиною буде спроба викликати метод Close для посилання, що має значення nil):

```
ActiveMDIChild.Close; /// ???
```

Виконувати перевірку збереження даних документа випливає безпосередньо в модулі відповідного дочірнього вікна — у підпрограмах обробки подій OnClose і OnCloseQuery. У подібному випадку закрити дочірнє вікно буде неможливо, поки його об'єкт не упевниться в тім, що це припустимо. (От приклад реальних переваг объектно-ориентированного програмування.)

Процедури FileSaveClick і FileSaveAsClick об'єкта MainForm викликають метод SaveData активного об'єкта дочірнього вікна. Якщо заголовок цього вікна має значення Untitled, то процедура FileSaveClick автоматично викликає процедуру FileSaveAsClick, у противному випадку вона викликає метод SaveData з аргументом, рівним поточного імені файлу, обраному з заголовка вікна. У процедурі FileSaveAsClick використовується метод для обмеження

відображуваних у діалоговому вікні Save файлів лише тими, котрі мають розширення імені, що збігає з розширенням файлу в поточному вікні.

Завершує приведений фрагмент програми процедура обробки події File⇨Exit об'єкта MainForm — процедура FileExitClick, у якій просто викликається метод Close. Таке спрощення припустиме, оскільки батьківський об'єкт автоматично починає спроби закрити і звільнити всі приналежні йому дочірні вікна. Додаток завершить свою роботу тільки тоді, коли всі його дочірні вікна будуть закриті.

1.2.3.3. Використання команд меню Window

У більшості MDI-додатків передбачено меню Window, що призначено для виконання різних операцій над дочірніми вікнами — розташування каскадом, мозаїкою або упорядкування їхніх піктограм у клієнтській області основного вікна. У прикладі 1.2.3 приведений текст підпрограм обробки подій меню Window додатка MDIDemo, що демонструють методи написання власних команд для роботи з вікнами, наприклад Close All.

Приклад 1.2.3 - Підпрограми обробки подій OnClick команд меню Window додатка MDIDemo

```
procedure TMainForm.WindowCascadeClick(Sender: TObject);
begin
  Cascade;
end;

procedure TMainForm.WindowTileClick(Sender: TObject);
begin
  Tile;
end;

procedure TMainForm.WindowArrangeIconsClick(Sender: TObject);
begin
  ArrangeIcons;
end;

procedure TMainForm.WindowMinimizeAllClick(Sender: TObject);
var
  I: Integer;
begin
  for I := MDIChildCount - 1 downto 0 do
    MDIChildren[I].WindowState := wsMinimized;
end;

procedure TMainForm.WindowCloseAllClick(Sender: TObject);
var
  I: Integer;
begin
  for I := MDIChildCount - 1 downto 0 do
    MDIChildren[I].Close;
end;
```

Перші три процедури реалізують стандартні команди меню Window — Cascade, Tile і Arrange Icons. Оскільки ці команди використовуються дуже часто, у класі TForm реалізовані методи Cascade, Tile і ArrangeIcon. У відповідь на вибір відповідної команди меню Window досить просто викликати один з цих методів. Дані методи можна викликати й в інших ситуаціях, наприклад при обробці події OnClick для піктограм панелі інструментів додатка.

Двома іншими стандартними командами, які можна помістити в меню Window, є Next і Previous. Для їхньої реалізації досить викликати методи Next і Previous об'єкта основного вікна. Додатково може знадобитися деактивизировать даної команди, якщо існує не більш одного дочірнього вікна. З цією метою можна скористатися приведеними нижче операторами, поміщеними в процедуру обробки події OnClick меню Window (передбачається, що WindowNext і WindowPrevious є об'єктами класу TMenuItem):

```
WindowNext.Enabled := MDIChildCount > 1;
WindowPrevious.Enabled := WindowNext.Enabled;
```

Крім стандартних команд меню Window, можна підготувати й інші команди, що виконують операції над дочірніми вікнами. Наприклад, у додатку MDIDemo передбачена команда Window⇨Minimize all, у реалізації якої використовується приведений нижче цикл for:

```
for I := MDIChildCount - 1 downto 0 do
MDIChildren[I].WindowState := wsMinimized;
```

Для розташування піктограм у визначеному порядку в циклі for виконується послідовна обробка дочірніх вікон, починаючи від останнього відкритого вікна до першого. Для доступу до кожного з дочірніх вікон використовується масив MDIChildren об'єкта основного вікна. Елементи даного масиву мають тип TForm, тому, якщо використовується не один з успадкованих методів або властивостей (наприклад, WindowState у приведеному вище прикладі), при виклику методів або звертанні до властивостей об'єктів дочірніх вікон варто використовувати вираження приведення типів. Так, нижче представлений приклад виклику функції YourMethod для першого дочірнього вікна:

```
if MDIChildCount > 0 then
TChildForm(MDIChildren[0]).YourMethod;
```

Інший варіант запису цих же команд передбачає використання конструкції with-do:

```
if MDIChildCount > 0 then
  with MDIChildren[0] as TChildForm do YourMethod;
```

У коректно написаному MDI-додатку завжди варто перевіряти існування хоча б одного дочірнього вікна, перш ніж виконувати які-небудь дії з використанням масиву MDIChildren. Досить переконатися або в тім, що властивість MDIChildCount має відмінне від нуля значення, або в тім, що значення властивості ActiveMDIChild не дорівнює nil.

Хоча команда Window(Close All реалізується не у всіх MDI-додатках, її використання має визначений сенс. Так, працюючи над різними проектами, можна відкривати безліч вікон. Тому може виявитися корисним мати спосіб закрити їхній усі відразу, без необхідності послідовно закривати одне вікно за іншим. У додатку MDIDemo подібна дія реалізована в процедурі WindowCloseAllClick, що використовує для послідовного закриття вікон у порядку від останнього відкритого до першого наступний цикл for:

```
for I := MDIChildCount - 1 downto 0 do MDIChildren[I].Close;
```

Внутрішньо масив MDIChildren реалізований як об'єкт класу TList, отже, він допускає виконання таких операцій, як звертання до попереднього або наступного елемента, обумовлених номером дочірнього вікна.

1.2.3.4. Інші команди MDI – додатка

У прикладі 1.2.4 приведена частина тексту, що залишилася, модуля Main.pas.

Приклад 1.2.4 - Інші процедури основного модуля додатка MDIDemo

```
procedure TMainForm.HelpAboutClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;

procedure TMainForm.FileMenuClick(Sender: TObject);
begin
  FileNew.Enabled := MDIChildCount < maxChildren;
```

```
  FileOpen.Enabled := FileNew.Enabled;
  FileClose.Enabled := MDIChildCount > 0;
  FileSave.Enabled := FileClose.Enabled;
  FileSaveAs.Enabled := FileClose.Enabled;
end;

procedure TMainForm.WindowMenuClick(Sender: TObject);
var
  I: Integer;
begin
  with WindowMenu do
  for I := 0 to Count - 1 do
    with Items[I] as TMenuItem do
      Enabled := MDIChildCount > 0;
end;

end.
```

Процедура HelpAboutClick виводить на екран діалогове вікно AboutBox, що тут розглядатися не буде. Процедури FileMenuClick і WindowMenuClick призначені для обробки подій OnClick об'єктів FileMenu і WindowMenu. Програма звертається до цих процедур, коли користувач відкриває відповідне меню. Призначення даних процедур складається в активізації і блокуванні команд меню в залежності від поточної ситуації в роботі додатка. Наприклад, команда File(New може бути заблокована, якщо значення властивості MDIChildCount стане дорівнює значенню властивості maxChildren, що дозволяє обмежити число дочірніх вікон, що пользовать зможе відкрити. Усі команди меню Window повинні бути заблоковані, якщо не існує жодного дочірнього вікна. Якщо це меню містить і рекомендовані вище команди Next і Previous, то їхня активізація повинна здійснюватися окремо, коли значення властивості MDIChildrenCount стане більше одиниці.

Немає необхідності обов'язково обмежувати число дочірніх вікон. Ця функція включена в додаток MDIDemo винятково з демонстраційних розумінь. Будь-який MDI-додаток може відкривати стільки дочірніх вікон, скільки дозволять існуючий обсяг пам'яті й інших системних ресурсів.

1.2.3.5. Використання дочірніх вікон різних типів

У MDI-додатках можуть застосовуватися дочірні вікна різних типів. Для кожного використовуваного типу дочірнього вікна необхідно створити в проекті окрему форму і помістити в основний модуль програмний текст, призначений для створення екземпляра цього об'єкта. Як приклад виконаєте приведені нижче дії, ціль яких — визначити в додатку MDIDemo новий тип дочірнього вікна, призначеного для висновку на екран растрових графічних зображень.

1) Відкрийте файл проекту MDIDemo.dpr. Додайте в додаток MDIDemo ще один модуль, для чого виберіть команду File⇨New Form або клацніть на піктограмі New Form.
2) Зміните значення властивості Name нової форми з Form1 на ChildBmpForm. Властивості FormStyle привласніть значення fsMDIChild. Виберіть команду Project⇨Options і перемістите ім'я форми ChildBmpForm зі списку **Auto-create forms** у список **Available forms**. Тепер у правому списку повинне бути присутнім два імена форм — ChildForm і ChildBmpForm. Лівий список повинний містити імена двох інших форм — MainForm і AboutForm. Закрийте діалогове вікно Project Options, клацнувши на кнопці OK.
3) Збережете проект, вибравши команду File(Save All. У діалоговому вікні запиту імені файлу, що зберігається, замість прийнятого за замовчуванням значення Unit1.pas уведіть Childbmp.pas.

4) Виберіть форму ChildBmp, клацнувши де-небудь усередині її вікна. Перейдіть на вкладку Events вікна Object Inspector. Для створення процедури обробки події OnClose двічі клацніть на поле, розташованому в рядку цієї події правее його імені. Введіть оператор, яким перемінної Action буде привласнюватися значення caFree. Це забезпечить звільнення об'єкта форми при закритті відповідного дочірнього вікна. Якщо не виконати цей крок, то при закритті вікно буде мінімізовано в клієнтській області основного вікна додатка (що цілком припустимо, однак не зовсім те, що малося на увазі користувачем). Процедура обробки події OnClose повинна виглядати в такий спосіб:

```
procedure TChildBmpForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
end;
```

5) Вставте у форму ChildBmp компонент TImage, розташований на вкладці Additional палітри. Властивості Align об'єкта Image1 привласніть значення alClient, що забезпечить заповнення їм усієї клієнтської області дочірнього вікна. Властивості Stretch об'єкта привласніть значення True. Зазначені призначення при виконанні додатка забезпечать вставку графічного об'єкта в дочірнє вікно й автоматичну зміну його розмірів відповідно до розмірів вікна.
6) Перейдіть у вікно редактора коду і виберіть вкладку з модулем ChildBMP. В оператор uses, розташований на початку модуля, додайте ім'я модуля Child. Це необхідно, оскільки новий клас форми повинний успадковувати вихідну форму Child (бути похідним від неї). Повний текст оператора uses тепер повинний виглядати так, як показано нижче (доданий текст виділений напівжирним шрифтом):
   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialoas. **Child**;
7) Не виходячи з вікна редактора коду, зміните на вкладці ChildBmp назва базового класу об'єкта TChildBmpForm з TForm на TChildForm. Це забезпечить спадкування класом TChildBmpForm властивостей і методів класу TChildForm. Подібна технологія називається створенням підкласу форми. Змінене оголошення класу TChildBmpForm повинне виглядати в такий спосіб (змінений текст виділений напівжирним шрифтом):
   TChildBmpForm = **class**(**TChildForm**)
8) Додайте в клас TChildBmpForm оголошення процедур LoadData і SaveData. Оголошення цих процедур повинні містити директиву override, що вкаже Delphi на необхідність замінити ними віртуальні методи, успадковані з класу TChildForm. Помістите ці оголошення в розділ public оголошення класу TChildBmpForm:

```
public
  procedure LoadData(const FileName: String); override;
  procedure SaveData(const FileName: String); override;
```

Реалізацію цих двох процедур помістите в секцію implementation модуля ChildBmp. Повний текст даного модуля показаний у прикладі 1.2.5. Його можна використовувати як довідкове керівництво при введенні тексту, якщо ви виконуєте всі дії, що тут рекомендуються.

У вікні Object Inspector виберіть об'єкт ChildBmpForm і перейдіть на вкладку Events. Двічі клацніть на поле, розташованому в рядку події OnCreate правее імені цієї події. Уведіть текст процедури FormCreate, приведений у прикладі 1.2.5. Це дозволить забрати наслідуваний об'єкт Memo1, що у противному випадку буде конфликтовать з об'єктом Image1 знову створеного класу.

Приклад 1.2.5 – Повний текст вихідного модуля ChildBmp

```
unit Childbmp;

interface

uses
```

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, Child;

type
  TChildBmpForm = class(TChildForm)
    Image1: TImage;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
    {Private declarations}
  public
    procedure LoadData(const FileName: String); override;
    procedure SaveData(const FileName: String); override;
    {Public declarations}
  end;

var
  ChildBmpForm: TChildBmpForm;

implementation

{$R *.DFM}

procedure TChildBmpForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
end;

procedure TChildBmpForm.LoadData(const FileName: String);
begin
  Image1.Picture.LoadFromFile(FileName);
  Caption := LowerCase(FileName);
end;

procedure TChildBmpForm.SaveData(const FileName: String);
begin
  Image1.Picture.SaveToFile(FileName);
  Caption := LowerCase(FileName);
end;

{Звільнення об'єкта Memo1 у методі FormCreate для того, щоб запобігти конфлікт даного
об'єкта з об'єктом Image1 знову створеного класу TChildBmpForm}
procedure TChildBmpForm.FormCreate(Sender: TObject);
begin
  inherited;
  Memo1.Free;
end;

end.
```

Звільнення об'єкта Memo1 у методі FormCreate виконується для того, щоб запобігти конфлікт даного об'єкта з об'єктом Image1 знову створеного класу TChildBmpForm. Це ілюструє

одну з найважливіших потенційних проблем, зв'язаних зі створенням підкласу форми - у новому класі можуть бути додані об'єкти, що будуть конфліктовать з об'єктами, успадкованими від класу-попередника. Деякі пропозиції по поліпшенню результатів спадкування в ієрархії класів можна знайти в прикладі 1.2.6.

Останній етап доробки додатка MDIDemo для читання і запису файлів растрових зображень — перепрограмування основного модуля таким чином, щоб він одержав можливість створювати екземпляри об'єктів нового дочірнього класу. Виберіть у вікні редактора програм вкладку модуля Main (якщо файл модуля Main MainForm.pas ще не відкритий, скористайтеся для його відкриття командою View⇨Project Manager). Додайте в оператор uses основного модуля значення ChildBmp, після чого цей оператор повинний прийняти наступний вид:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Menus, Child,
  ChildBmp;
```

Перепишіть процедуру CreateChild так, як показано в прикладі 1.2.6. Для компіляції і запуску модифікованої програми натисніть клавішу <F9>. Відкрийте будь-як файл із растровим графічним об'єктом. (Якщо раніше ви помістили об'єкт Memo1 у модуль Child, як це пропонувалося вище, те програма зможе працювати і з текстовими файлами.) Програма вибирає необхідний тип вікна, ґрунтуючись на розширенні імені обраного файлу.

Приклад 1.2.6 - Модифікована процедура CreateChild модуля Main

```
procedure TMainForm.CreateChild(const Name: String);
var
  Child: TChildForm;
  FExt: String[4];
begin
  FExt := ExtractFileExt(Name);
  if FExt = '.bmp' then
    Child := TChildBmpForm.Create(Application)
  else
    Child := TChildForm.Create(Application);
  Child.Caption := Name;
end;
```

У новій процедурі CreateChild створюється дочірнє вікно класу TChildBmpForm, якщо розширенням обраного файлу є .bmp. У противному випадку створюється об'єкт класу TChildForm. Оскільки процедури LoadData і SaveData є віртуальними (див. приклади 1.2.1 і 1.2.5), тип об'єкта дочірнього вікна визначає і тип даних файлу, що завантажується.

Щоб одержати більш глибоке представлення про цінності використання віртуальних методів у формах підкласів, розглянемо докладніше процедуру FileOpenClick модуля Main, приведену в прикладі 1.2.2. Якщо дочірнім вікном є об'єкт класу TChildForm, процедура викликає вихідну процедуру LoadData. Якщо дочірнє вікно представлене об'єктом TChildBmpForm, процедура викликає перевизначені методи. Ці виклики контролюються не безпосередньо поміщеним у програму текстом, а саме типом об'єкта дочірнього вікна, на яке указує властивість ActiveMDIChild. Це одне з важливих переваг объектно-ориентированного програмування: об'єкт дочірнього вікна визначає тип даних, що він здатний зчитувати і зберігати.

Завдяки використанню в програмі объектно-ориентированной технології відносно нескладно додати до нашої програми й інші типи дочірніх вікон. Досить створити і запрограмувати модуль нової форми так, як це було зроблено для TChildBmpForm. Потім необхідно забезпечити спадкування новим класом від класу TChildForm і написати для нього процедури LoadData і SaveData. І, нарешті, буде потрібно модифікувати процедуру CreateChild у модулі Main, забезпечивши в ній можливість створення екземплярів об'єктів нового класу, після чого робота буде довершена.

1.2.3.6. Об'єднання меню

У MDI-додатках, що використовують дочірні вікна декількох типів, може знадобитися модифікувати команди меню або навіть додати нові меню в залежності від типу активного дочірнього вікна. Зробити це нескладно, однак буде потрібно приділити увага деяким не цілком очевидним деталям.

У батьківському вікні (у тім, у якому властивості FormStyle привласнене значення fsMDIForm) об'єкт MainMenu повинний містити глобальні команди, що можуть бути застосовані до всіх типів дочірніх вікон. Наприклад, меню батьківського вікна, як правило, включає меню Window, що містить команди Cascade, Tile і інші, застосовні до усіх вікон незалежно від їхнього типу. Меню основного вікна повинне включати і такі команди, як File(New, File(Open і File(Close, призначені для відкриття і закриття дочірніх вікон.

У кожне дочірнє вікно, у якому необхідно використовувати специфічні команди або вносити зміни в існуючі команди з обліком різних можливих умов, варто помістити власний об'єкт MainMenu. Иожно іменувати подібні об'єкти відповідно до імені модуля, наприклад ChildFormMenu або ChildBmpFormMenu. У ці об'єкти MainMenu варто помістити команди, що будуть поєднуватися з командами меню основного вікна.

У MDI-додатках об'єднання меню здійснюється автоматично для кожного дочірнього вікна, що містить об'єкт MainMenu. З цієї причини варто привласнити властивості AutoMerge кожного з об'єктів MainMenu значення False. Говорячи точніше, значення False потрібно привласнити тільки параметрові AutoMerge об'єкта MainMenu основного вікна. Значення властивостей AutoMerge об'єктів MainMenu дочірніх вікон просто ігноруються, однак корисно і їм привласнити значення False.

Потім варто привласнити значення властивостям GroupIndex окремих команд меню. Дані значення визначають, куди повинні вставлятися меню, а чи також належні вони заміщати існуючі команди меню або просто додаватися до них. Об'єднання меню в додатку виконується у відповідності з наступними правилами обробки значень властивостей GroupIndex об'єктів MainMenu.

- Елементи меню з однаковими значеннями властивості GroupIndex заміщаються. Наприклад, якщо властивості GroupIndex елемента MainMenu основного вікна привласнене значення 10, те будь-який елемент MainMenu дочірнього вікна з цим же значенням властивості GroupIndex замінить даний елемент основного вікна.
- Елементи меню дочірніх вікон з унікальними значеннями властивості GroupIndex вставляються в меню основного вікна. Елементи, що вставляються, меню з великими значеннями властивості GroupIndex містяться правее елементів меню основного вікна з меншими значеннями цієї властивості. Елементи меню дочірнього вікна з меншими значеннями властивості GroupIndex містяться левее елементів меню основного вікна з великими значеннями цієї властивості.

Крім об'єднання меню, з'являється можливість викликати методи об'єктів TMenuItem для активізації і блокування пунктів меню, для вставки або зміни команд і для виконання інших дій над пунктами меню. Наприклад, за допомогою приведеного нижче оператора об'єкт дочірнього вікна може помістити маркер вибору в команду Option(Save:

MainForm.OptionsSave.Checked := True;

Для подібної операції важливим моментом є одержання доступу до об'єкта MainForm з дочірнього модуля. Для цього необхідно в оператор uses дочірнього модуля додати значення Main (або ім'я, що використовується для основного модуля). Щоб запобігти поява взаємних перехресних посилань, варто виконати це додавання не в операторі uses секції інтерфейсу дочірнього модуля, а в новому операторі, поміщеному в секцію реалізації модуля. Оскільки оператор uses основного модуля уже посилається на даний дочірній модуль, приміщення посилання на основний модуль у дочірній модуль приведе до появи взаємних перехресних посилань, що в мові Object Pascal заборонено. У даному випадку варто знайти в дочірньому

модулі ключове слово implementation і вставити після нього новий оператор uses так, як показано в приведеному нижче прикладі (це дозволить одержати доступ до елементів меню об'єкта MainForm):

implementation
*{$R *.DFM}*
uses Main;

Взаємні перехресні посилання між двома модулями виникають у тому випадку, якщо ці модулі (назвемо їхній Chicken (курча), і Egg (яйце)) посилаються один на одного в їхніх операторах uses секції інтерфейсу. Object Pascal забороняє подібні конструкції в зв'язку з тим, що модуль Chicken використовує оголошення, зроблені в модулі Egg, для яких, можливо, прийдеться скористатися оголошеннями, виконаними в модулі Chicken. Для дозволу подібних парадоксів помістите імена вторинних модулів в оператор uses секції інтерфейсу первинного модуля, а оператори uses з ім'ям первинного модуля — у вторинні модулі секції реалізації. Взаємні посилання між модулями в їхній секції інтерфейсу заборонені, тоді як для секцій реалізації подібні обмеження відсутні.

1.2.4. Інші MDI – технології

Нижче приведені деякі з оригінальних рішень побудови MDI-додатків, оскільки вони можуть виявитися корисними і при розробці інших MDI-додатків.

1.2.4.1. Отримання доступу до клієнтської області та рамці вікна

Іноді необхідно одержати доступ до клієнтських областей і рамок вікон MDI-додатка. Імовірно, це може знадобитися тільки при безпосереднім звертанні з програми до функцій Windows API, для яких потрібно вказати дескриптори вікна. У Delphi клас TForm здатний забезпечити практично будь-які можливості, що можуть знадобитися для більшості MDI-додатків.

Для посилання на клієнтську область основного вікна можна скористатися властивістю ClientHandle об'єкта основного вікна додатка. У типових MDI-додатках це забезпечує виконання будь-яких глобальних операцій (наприклад, обробку повідомлень), що застосовні до всіх дочірніх вікон.

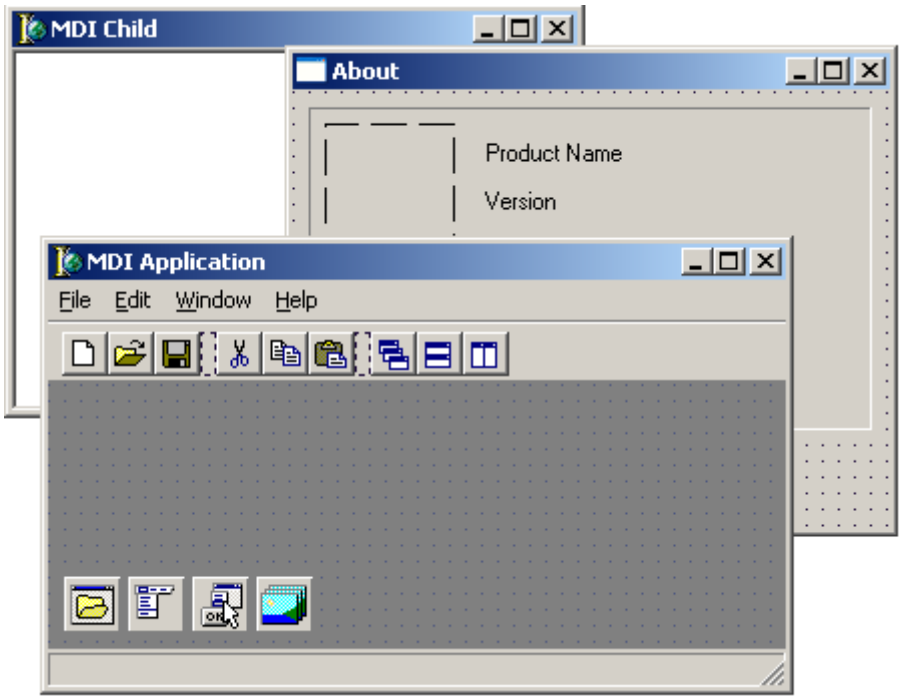
Значення властивості ClientHandle, що має тип HWnd (дескриптор вікна), доступно тільки у формах, властивість FormStyle яких має значення fsMDIForm.

На рамку вікна можна посилатися за допомогою властивості Handle основної форми. Його значення завжди посилається на дескриптор вікна як у звичайних, так і в MDI-додатках. Для форми, властивість FormStyle якої має значення fsMDIForm, властивість Handle посилається на рамку основного вікна MDI-додатка.

1.2.4.2. Використання шаблона MDI –додатка

Щоб швидко створити прототип деякого MDI-додатка, можна скористатися наявної в Delphi шаблоном стандартного MDI-додатка. Спочатку необхідно створити порожню папку, призначену для збереження файлів нового проекту. Потім варто вибрати команду File⇨New, перейти в діалоговому вікні, що **розкрилося, New** Items на **вкладку** Projects і вибрати **піктограму MDI** Application. Клацніть на кнопці **ОК** і вкажіть у діалоговому вікні, що **розкрилося, Select** Directory папку, призначену для розміщення файлів проекту.

Закрийте діалогове вікно **Select Directory**, і Delphi створить стандартний MDI-додаток з рядком стану, панеллю інструментів і рядком меню, що містить меню File, Edit, Window і Help. На мал. 1.2.3 показане вікно форми, що містить дочірні вікна Child і About, що також були створені з використанням шаблона.



Малюнок 1.2.3 - На базі шаблона MDI Application створюється форма, що містить рядок стану, панель інструментів і рядок меню. Крім того, забезпечується створення вікон Child і About

Нижче приведені рекомендації, що допоможуть вам освоїти роботу із шаблоном MDI Application.

- У модулі ChildWin.Pas з'являється клас TMDIChild. Можна додати в нього процедури LoadData і SaveData або ж організувати читання і запис даних безпосередньо в модулі Main. Якщо створюваний додаток буде мати тільки один тип дочірнього вікна, має сенс помістити обробку даних у модуль Main. У противному випадку кожний із класів дочірніх вікон буде сам відповідати за введення і висновок власних даних. Усі ці класи варто створювати на базі класу TMDIChild.
- За замовчуванням створений додаток називається MDIApp. Щоб перейменувати його, виберіть команду File(Save Project as і вкажіть нове ім'я файлу. Збережете і відкомпілюйте проект. Після цього можна буде видалити з папки проекту усі файли з іменами MdIApp.*.
- У модулі Main.Pas з'являється клас TMainForm. (Ознайомтеся з іменами різних класів, що повідомляються в цьому модулі. Одні імена (наприклад, Edit1) прийняті системою за замовчуванням, інші (наприклад, SaveBtn) обрані у відповідності зі своїм призначенням. Привласнене в шаблоні ім'я можна змінити, однак зробити це необхідно скрізь, де воно використовується. Наприклад, недостатньо просто змінити значення, привласнене властивості Name деякого об'єкта, — необхідно також виконати глобальний пошук і заміну імені цього об'єкта в будь-яких операторах, що містять посилання на нього. Delphi автоматично обновляє лише оголошення імен об'єктів, але глобальний пошук і заміна імені даного об'єкта в будь-яких операторах не виконується.
- Виконаєте команду View(Project Manager і виберіть значення ChildWin, що приведе до відкриття модуля і форми дочірнього вікна. Тепер можна помістити у форму дочірнього вікна будь-які необхідні об'єкти, створити поєднувані меню і написати процедури обробки команд дочірнього вікна.
- У модулі Main додайте в підпрограму FileOpenItemClick оператори завантаження даних. Як уже указувалося вище, можна або здійснювати читання файлів безпосередньо, або викликати процедуру LoadData для вікна, обумовленого властивістю ActiveMDIChild.

- Вставте в основну форму об'єкт SaveDialog і помістите в процедури FileSaveItemClick і FileSaveAsItemClick оператори, що здійснюють збереження даних документа. Інший варіант — забезпечити виклик процедури SaveData, которую необхідно буде додати в клас дочірнього вікна.
- При необхідності підготуйте програмний код для реалізації команд Cut, Copy і Paste, що забезпечує виклик процедур роботи з Clipboard. Наприклад, при реалізації команди Cut можна вставити у форму ChildWin об'єкт Memo і викликати з відповідної процедури метод CutToClipboard цього об'єкта.
- У модуль ChildWin додайте процедуру обробки події OnCloseQuery. Для запобігання закриття користувачем дочірнього вікна без збереження відображуваного в ньому документа привласніть значення False властивості CanClose.

1.2.5. Корисні поради

✍ У MDI-додатку значення fsMDIForm може бути привласнено властивості FormStyle тільки одного з вікон, причому це вікно повинне бути основним вікном додатка. (Для визначення, яке з вікон є основним вікном додатка, скористайтеся командою Project(Options.)

✍ Якщо замість того щоб з'являтися в межах основного вікна додатка, дочірні вікна виводяться на екран як незалежні і розміщаються на поверхні робочого столу, то, імовірніше всього, властивість FormStyle форми цього дочірнього вікна має некоректне значення. (У попередніх версіях Delphi і Windows подібна помилка могла викликати переривання роботи програми через порушення обший захисту системи, однак у нових версіях цього не спостерігається.) У всіх формах дочірніх вікон властивість FormStyle повинна мати значення fsMDIChild. Властивість FormStyle основного вікна додатка повинне мати значення fsMDIForm і тільки одне вікно у всьому додатку може бути основним.

✍ Якщо дочірнє вікно не повинне автоматично з'являтися на екрані при запуску додатка, виберіть команду Options⇨Project і на вкладці Forms діалогового вікна, що розкрилося, перемістите назву форми дочірнього вікна зі **списку Auto-create** forms у **список Available** forms. Проте не забороняється автоматично створювати в додатку перше дочірнє вікно безпосереднє при його запуску. Наприклад, MDI-додаток текстового редактора може при запуску автоматично виводити для користувача дочірнє вікно з порожнім документом. При цьому додаткові дочірні вікна будуть створюватися в додатку так, як описувалося вище в цій главі (тільки перше дочірнє вікно буде автоматично створюватися при запуску програми).

✍ Якщо в MDI-додатку закриття дочірнього вікна викликає лише його мінімізацію, переконаєтеся, що в модулі дочірнього вікна мається підпрограма обробки події OnClose, у якій перемінної Action привласнюється значення caFree. Якщо зазначене значення перемінної Action не привласнюється, то при закритті дочірнього вікна його об'єкт не буде звільнятися. Проте це не є помилкою і навіть може використовуватися для досягнення визначених цілей.

✍ У процесі виконання програми екземпляри форм і інших об'єктів (наприклад, екземпляр об'єкта TChildForm, що обговорювався в цій главі) створюйте шляхом присвоєння екземпляра створюваного об'єкта перемінної відповідного типу. Не намагайтеся викликати метод Create неинициализированной перемінної даного об'єктного типу. Наприклад, що випливають рядки містять подібну типову помилку:

```
var
  Child: TChildForm;
```

```
begin
  Child.Create(Application); {???}
```

✍ Помилкова команда буквально жадає від дочірньої форми породити саму себе що неможливо. Подібна команда завжди буде викликати помилку порушення доступу, оскільки перемінна Child ще не була инициализирована, а виклик методу неинициализированной перемінної є помилкою програмування (хоча і не є помилкою компіляції). Для звертання до методу Create класу і присвоєння посилання на створений при цьому екземпляр об'єкта як значення перемінної Child скористайтеся наступними операторами:

```
var
  Child: TChildForm; //Як тип можна використовувати і TForm
begin
  Child := TChildForm.Create(Application);
```

✍ Зберігати посилання на об'єкти дочірніх вікон необов'язково. Переважніше створювати екземпляри об'єктів дочірніх вікон з використанням значення Application для вказівки батька, що створює дочірнє вікно. Батько відповідає за підтримку і звільнення створюваного об'єкта. Як демонстрацію цього принципу можна зазначеним нижче образом скоротити процедуру CreateChild у модулі Main додатка MDIDemo. Однак це утруднить виконання операцій зі знову створюваним дочірнім вікном, зокрема — присвоєння значення його рядкові Caption:

```
procedure TMainForm.CreateChild(const Name: String);
begin
  TChildForm.Create(Application);
end;
```

✍ У MDI-додатках з декількома типами дочірніх вікон привласнюйте модулям цих вікон імена на зразок ChildXXX.pas, де XXX — розширення імені файлу, оброблюваного цим вікном документа. Наприклад, привласніть ім'я Childtxt.pas модулеві вікна, у якому обробляються файли .txt, і ім'я Childbmp.pas —вікну, що обробляє файли .bmp. Безумовно, можна привласнювати створюваним модулям будь-які імена, однак дана пропозиція корисна тим, що забезпечує ідентифікацію модулів по типі файлів, з яким ці модулі працюють.

✍ У MDI-додатках меню поєднується автоматично, тому варто привласнити властивості AutoMerge всіх об'єктів MaimMenu значення False. Для основного вікна MDI-додатка ця вимога є обов'язковим. Властивість AutoMerge призначена для керування об'єднанням меню тільки в однодокументных додатках.

✍ Уникайте приміщення яких-небудь компонентів в основне вікно MDI-додатка. При приміщенні в клієнтську область основного вікна MDI-додатка малозначні елементи керування (наприклад, TLabel) узагалі не будуть прорисовуватися, а звичайні елементи керування (наприклад, TButton) можуть функціонувати некоректно. Це має місце внаслідок недоробок у MDI-інтерфейсі Windows. Проте розташовані уздовж верхньої або нижньої границі основного вікна панелі інструментів цілком прийнятні. Всі елементи іншого типу або поміщайте у форми дочірніх вікон, або оформляйте, як елементи панелей керування.

1.2.6. Резюме

☝ У MDI-додатках для створення основного і дочірнього вікон використовується компонент Delphi Form, відсутній у палітрі візуальних компонентів. Головне вікно

виконує роль рамки і клієнтської області вікна додатка, використовуваних у звичайних додатках Windows.

- Кожен MDI-додаток має три основних типи компонентів: форму основного вікна, одну або кілька форм дочірніх вікон і основне меню. Властивість WindowMenu форми основного вікна використовується для вказівки елемента меню верхнього рівня, у якому будуть перелічуватися заголовки відкритих дочірніх вікон.
- Властивість FormStyle форми основного вікна повинне мати значення fsMDIForm. У додатку може існувати тільки одна подібна форма. У формах усіх дочірніх вікон властивість FormStyle повинна мати значення fsMDIChild. У додатку може використовуватися стільки різних форм дочірніх вікон, скільки визначено в його алгоритмі.
- Для створення екземпляра дочірнього вікна в процесі виконання програми варто викликати метод Create класу вікна. Як параметр об'єкта-батька методові Create повинне передаватися значення Application.
- У кожнім класі дочірнього вікна повинні бути реалізовані процедури обробки подій OnClose і OnCloseQuery. У цих процедурах варто виводити для користувача попередження, якщо він почне спробу закрити вікно з документом, зміни в якому не збережені.
- Властивість ActiveMDIChild основної форми містить посилання на поточне активне дочірнє вікно. Якщо значення цієї властивості дорівнює nil, виходить, у додатку не існує жодного дочірнього вікна. Перш ніж використовувати значення властивості ActiveMDIChild для виклику методів або установки значення властивостей об'єкта дочірнього вікна, перевірте, чи не містить зазначена властивість значення nil.
- Значення властивості MDIChildCount основного вікна дорівнює кількості відкритих у додатку дочірніх вікон. Якщо дочірні вікна відсутні, значення цієї властивості дорівнює нулеві.
- Масив MDIChildren забезпечує індексований доступ до всіх дочірніх вікон, що належать додаткові.
- Об'єкти MainMenu дочірніх вікон автоматично поєднуються з об'єктом MainMenu основного вікна. Для керування процесом об'єднання використовується властивість GroupIndex.
- Властивість Handle основної форми містить посилання на об'єкт рамки основного вікна MDI-додатка. Властивість ClienHandle містить посилання на клієнтську область цього вікна. У більшості MDI-додатків використовувати ці властивості не буде потрібно.
- Для швидкого створення типового MDI-додатка можна скористатися шаблоном MDI Application, що передбачає створення панелі меню з основними меню, панелі інструментів і рядка стану додатка.

Література [1-2].

Тема 1.3. Обробка виключних ситуацій

Обробка помилок — це один з неприємних обов'язків розроблювача програмного забезпечення. І її ніяк не можна обійти: якщо додаток буде зненацька завершуватися через яку-небудь внутрішню помилку, навряд чи ви знайдете бажаючого їм користуватися. У надійному додатку повинні оброблятися всі ситуації, що можуть привести до помилкових результатів або до припинення роботи програми.

Такі ситуації називаються *винятковими*. Обробці виняткових ситуацій і присвячена ця

глава. У мові Object Pascal мається досить складний механізм обробки виняткових ситуацій. У кожнім додатку Delphi є оброблювачі виняткових ситуацій за замовчуванням, що видають повідомлення про помилки і запобігають “зависання” програми при виникненні таких ситуацій. Убудовуючи додаткову обробку виняткових ситуацій, можна домогтися більшої стійкості додатків до несподіваних ситуацій, а при виникненні такої ситуації виконати всі підготовчі дії перед завершенням програми — звільнення виділеної пам'яті, збереження критичних даних і закриття файлів.

1.3.1. Поняття виключної ситуації

Досвідчені розроблювачі завжди піклуються про обробку помилок під час циклу розробки. Обробка виняткових ситуацій у Object Pascal — це объектно-ориентированная технологія, що зводить написання коду до мінімуму. Вона дозволяє створювати код програми одночасно з обробкою помилок, і, як ви довідаєтеся в цій главі, оброблювачі виключень зовсім не заважають написанню основних алгоритмів на відміну від інших методів. Якщо ж ви волієте використовувати безліч вкладених операторів if, прапори типу Boolean і значення, що повертаються спеціальними функціями, то механізм обробки виняткових ситуацій не для вас.

1.3.1.1. Як виникають виключні ситуації

Існує безліч джерел виняткових ситуацій. Наприклад, програмою можуть бути сгенерированы виключення через який-небудь ненормальний стан. Виключення генеруються компонентами Delphi для різних подій, таких як присвоєння властивості значення, що виходить за припустимі межі, або спроба індексувати неіснуючий елемент масиву.

У процедурах і функціях бібліотек також можуть бути сгенерированы виключення. Виняткова ситуація виникає при виконанні некоректної математичної операції, такий як розподіл на нуль. Серед інших прикладів операцій, що приводять до виключень, — звертання по посиланню з покажчиком nil, виділення пам'яті, розмір якої перевищує розмір найбільшого вільного блоку, і виконання неприпустимого приведення типів.

Потенційно будь-який оператор програми може стати причиною виняткової ситуації. Однак деякі з операторів відносно безпечні, і ваша задача написання стійкої програми складається в основному в тім, щоб правильно вирішити, для яких операторів потрібна захист, а які безпечні. У прикладах цієї глави розглянуті стандартні випадки виникнення виняткових ситуацій, що допоможе вам успішно реалізувати захист для своєї програми.

Однак усієї вищесказане зовсім не означає, що потрібно писати операторы для обробки всіх можливих виняткових ситуацій. У Delphi мається глобальний оброблювач помилок, що автоматично додається до кожного додатка й автоматично генерує виняткові ситуації, виводячи діалогові вікна з указівкою проблеми і розташування проблемного оператора. Цей спосіб обробки виключень підходить для тестування програм і прикладів, таких як більшість прикладів цієї глави, але для обліку особливостей конкретної програми потрібне написання спеціальних оброблювачів.

От типи операцій, що можуть привести до виняткових ситуацій:

- обробка файлу;
- виділення пам'яті;
- робота з ресурсами системи Windows;
- робота з об'єктами і формами, створюваними під час виконання програми;
- апаратні конфлікти і конфлікти операційної системи.

1.3.1.2. Деякі корисні терміни

У Object Pascal є кілька ключових слів для створення й обробки виключень. Це try, except, on-do-else, finally, raise і at. Усі вони будуть описані нижче. Не використовуйте них для інших цілей.

Компіляторами Borland Pascal і Turbo Pascal не підтримуються ключові слова обробки виняткових ситуацій, прийняті в Object Pascal. Обробка виняткових ситуацій - це нововведення, що з'явилося в компіляторі Object Pascal середовища Delphi.

Нижче приведені формальні визначення термінів, зв'язаних з обробкою виняткових ситуацій у додатках.

- ***Виняткова ситуація (виключення)***. Незвичайна або несподівана ситуація, що перериває нормальне виконання програми.
- ***Генерування виняткової ситуації***. Формування повідомлення про незвичайну або несподівану ситуацію. Методи компонентів, підпрограми динамічних бібліотек, вираження, збої в роботі устаткування і навіть присвоєння невірного значення властивості компонента потенційно можуть згенерувати виняткову ситуацію. Виняткові ситуації можуть бути сгенерированы й у вашій програмі при визначенні ситуації, що вимагає спеціальної обробки. Для генерації виключення використовується ключове слово raise.
- ***Оброблювач виключення***. Фрагмент програми, що дозволяє проблемну ситуацію, приведшую до виключення. У цьому фрагменті система повинна бути приведена до стабільного стану так, щоб програма могла продовжити роботу в звичайному режимі.
- ***Екземпляр виключення***. Об'єкт класу, найчастіше похідного від класу Exception, що визначений у модулі SysUtils. Екземпляр виключення звичайно містить інформацію, що описує природу виниклої виняткової ситуації. При генеруванні виняткової ситуації для її екземпляра виділяється область пам'яті. Після обробки ця пам'ять автоматично звільняється.
- ***Блок try***. Один або трохи операторів, що випливають за ключовим словом try. Для цих операторів будуть оброблятися виключення або звільнятися ресурси, такі як виділена пам'ять.
- ***Блок except***. Один або трохи операторів, що обробляє виключення, сгенерированные в попередньому блоці try. Ці блоки починаються з ключового слова except і можуть містити оператори on-do-else. Блок except повинний випливати безпосередньо після блоку try.
- ***Блок finally***. Один або трохи операторів для виділення пам'яті, закриття файлів або виконання інших критичних операцій при генеруванні виняткової ситуації в попередньому блоці try. Блоки починаються з ключового слова finally і повинні безпосередньо випливати за блоком try. У блоках finally виняткові ситуації не обробляються — обробка відбувається тільки в блоках except.
- ***Блок захищених операторів***. Один або трохи операторів, для яких маються спеціальні оброблювачі кожного виключення, генерируемого цими операторами. Блок захищених операторів складається з двох блоків — try і except.
- ***Блок захищених ресурсів***. Один або трохи операторів, для яких маються спеціальні оброблювачі, що звільняють ресурси. Ці оброблювачі викликаються при виникненні виняткових ситуацій, зв'язаних з використанням ресурсів. Як ресурси можуть виступати області пам'яті, ресурси Windows і файли. Блок захищених ресурсів складається з двох блоків — try і finally.
- ***Стійкий додаток***. Програма, у якій використовуються блоки захищених операторів і блоки захищених ресурсів для обробки всіх можливих помилкових ситуацій і для безпечного звільнення ресурсів.

Запам'ятаєте таке правило: *стійкий додаток вимагає часу на написання, але не залишає слідів при виконанні*. Ціль використання виключень - написати стійку програму, що елегантно обробляє всі можливі помилкові стану.

1.3.1.3. Блоки захищених операторів

Блок захищених операторів — це основний засіб для обробки виняткових ситуацій У прикладі 1.3.1 показана схема створення блоку захищених операторів за допомогою блоків try і except. Зверніть увагу, що конструкція закінчується ключовим словом end і крапкою з коми. У коментарях зазначено, які оператори можуть бути поміщені в ту або іншу частину конструкції.

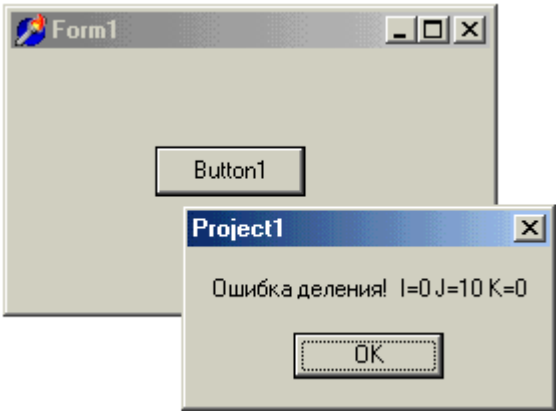
Приклад 1.3.1 - Схема створення блоку захищених операторів

```
{Незахищені оператори.}
try
{Захищені оператори, що можуть згенерувати виключення.}
except
{Оператори для обробки сгенерированных виключень.}
end;
{Інші незахищені оператори.}
```

Розглянемо на реальному прикладі, як діє схема приклада 1.3.1. Для цього створимо блок захищених операторів. Виконаєте наступні операції.

- Відкрийте новий додаток.
- Помістите об'єкт Button у форму.
- Двічі клацніть на об'єкті Button1 і, як показано в прикладі 1.3.2, напишіть обработчик події OnClick із блоком захищених операторів.
- Скомпілюйте і запустите програму, натиснувши <F9>. Клацніть на кнопці програми, щоб примусово створити виняткову ситуацію.

На мал. 1.3.1 показане вікно повідомлення, що відкривається за допомогою оператора ShowMessage в оброблювачі виключення.



Малюнок 1.3.1 - Це повідомлення з'являється при обробці виняткової ситуації розподілу на нуль у процедурі OnClick

Приклад 1.3.2 - У цьому оброблювачі події OnClick показано як можна створити блок захищених операторів

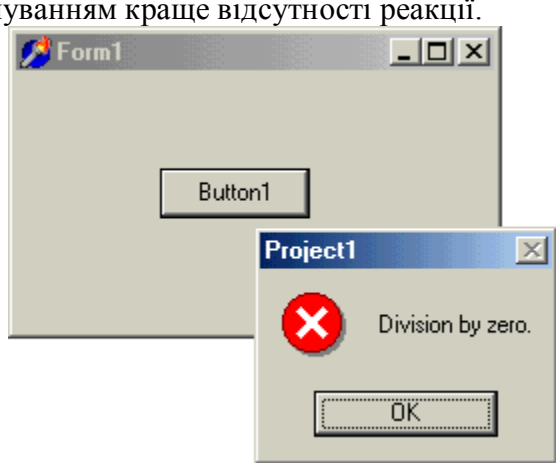
```
procedure TForm1.Button1Click(Sender: TOtgect);
var
  I, J, K: Integer;
begin
  I := 0;
  J := 10;
  try
    K:= J div I;
  except
  ShowMessage('Помилка розподілу! ' + ' I=' + IntToStr(I) + ' J=' + IntToStr(J) + ' K='
  + IntToStr(K));
  end;
```

end;

При тестуванні виключень бажано закривати утиліту відстеження повідомлень WinSight32, якщо вона запущена. Також, якщо ви використовуєте Delphi 4, виберіть команду Tools⇨Debugger Options і на вкладці **Language Exception** зніміть прапорець **Stop on Delphi Exception**. Якщо ви не знайшли в себе вищеописану команду, виберіть команду Tools⇨Environment Options, вкладку **Debugger**, пункт **Delphi Exceptions** у списку **Exceptions** і опцію **User Program** на панелі **Handled By**. Якщо ви використовуєте Delphi 3, виберіть команду Tools⇨Environment Options, вкладку **Preferences** і зніміть прапорець **Break on Exception**. Тепер виключення не будуть отлавливаться середовищем Delphi і керування винятковими ситуаціями цілком передається вашому додаткові. (Аналогічний ефект можна одержати, якщо скомпілювати програму і запустити файл *.exe за допомогою провідника Windows.)

У процедурі приклада 1.3.2 виконується математично неприпустима операція. У вираженні J div I відбувається розподіл на нуль, і бібліотекою Object Pascal генерується виняткова ситуація. Однак, оскільки вираження поміщене в блок try, виключення обробляється: виводиться повідомлення про помилку. Процедура ShowMessage у блоці except буде викликана тільки в тому випадку, якщо відбудеться збій в одному або декількох операторах блоку try. Програма одержує повний контроль над усіма винятковими ситуаціями — при виконанні операторів блоку except значення K не визначене. Звичайно, це тільки найпростіший приклад, але, якщо програма складна і містить безліч вкладених операторів, такий механізм просто неоціненний.

Щоб побачити, що відбудеться, якщо в незахищених операторах буде сгенерирована виняткова ситуація, видалите ключові слова try, except і перше end з оброблювача події OnClick. Інші оператори нехай залишаться незмінними. Потім натисніть <F9>, щоб відкомпілювати і запустити змінену програму. Клацніть на кнопці, і з'явиться діалогове вікно оброблювача виняткових ситуацій за замовчуванням (мал. 1.3.2). Цей експеримент показує, що навіть без використання блоків захищених операторів, усі виняткові ситуації в остаточному підсумку обробляються. Для комерційного додатка така реакція на виняткові ситуації може бути неадекватної, але програма, принаймні, не приводить до аварійних ситуацій. Реакція за замовчуванням краще відсутності реакції.



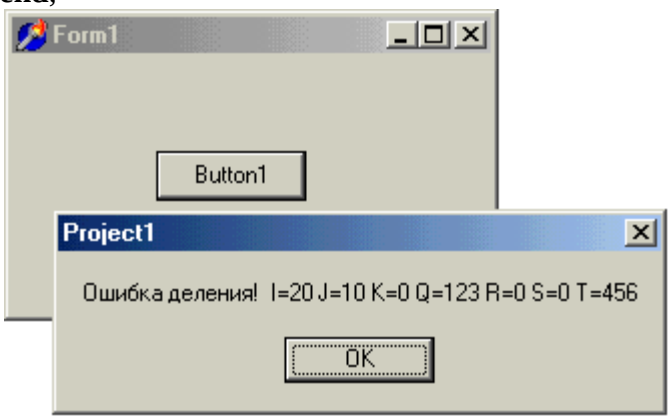
Малюнок 1.3.2 - Оброблювачем виняткових ситуацій за замовчуванням виводиться діалогове вікно для виключень, не оброблених явно

При виникненні виняткової ситуації в блоці try виконання негайно переходить до першого оператора блоку except. Після генерування виняткової ситуації подальші оператори блоку try не виконуються. Цей факт має важливі наслідки. Наприклад, розглянемо захищений блок із приклада 1.3.3. Цей блок можна вставити в оброблювач події OnClick об'єкта Button1 у попередньому додатку. Якщо яка-небудь з операцій розподілу викликає виняткову ситуацію, то керування негайно перейде до оператора ShowMessage, після якого продовжиться звичайне виконання операторів процедури, розташованих після ключового слова end. Наприклад, якщо помилка виникне в другій операції розподілу (що і відбудеться при значеннях, привласнених перемінним), третє вираження в блоці try не буде виконано. На мал. 1.3.3 показане повідомлення,

виведене за допомогою процедури ShowMessage.

Приклад 1.3.3 - Якщо одна з трьох операцій розподілу приведе до помилки, то виконання буде передано процедурі ShowMessage

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K, Q, R, S, T: Integer;
begin
  I := 20; J := 10; Q := 123; T := 456;
  try
    K:= (I div J) - 2;
    R := Q div K;
    S := T div R;
  except
    ShowMessage('Помилка розподілу! '+' I=' + IntToStr(I) + ' J=' +
    IntToStr(J) + ' K=' + IntToStr(K) + ' Q=' + IntToStr(Q) + ' R=' +
    IntToStr(R) + ' S=' + IntToStr(S) + ' T=' + IntToStr(T));
  end;
end;
```



Малюнок 1.3.3 - Повідомлення про виняткову ситуацію оброблювача події OnClick приклада 1.3.3

На прикладі процедури з приклада 1.3.3 показано одне з основних переваг обробки виняткових ситуацій. Воно полягає в тому, що немає необхідності кілька разів перевіряти наявність помилки в декількох однотипних операторах. Корисно порівняти обробку виключень з обробкою помилок без використання виключень, яку можна виконати багатьма способами. У прикладі 1.3.4 приведений типовий приклад обробки помилок “у чоло”. Намагайтеся уникати такого підходу. У результаті виконання цієї процедури з'явиться повідомлення, представлене на мал. 1.3.3.

Приклад 1.3.4 - Завдяки використанню виняткових ситуацій вам ніколи не прийдеться писати такі безладні процедури

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K, Q, R, S, T: Integer;
  {Подпроцедура для видачі повідомлення про помилку.}
  procedure ReportError;
  begin
    ShowMessage('Помилка розподілу! '+' I=' + IntToStr(I) + ' J=' +
    IntToStr(J) + ' K=' + IntToStr(K) + ' Q=' + IntToStr(Q) + ' R=' +
    IntToStr(R) + ' S=' + IntToStr(S) + ' T=' + IntToStr(T));
  end;
begin
```

```
  I := 20; J := 10; Q := 123; T := 456; R:=0; S:=0;
  if J = 0 then ReportError else
    begin
      K:= (I div J) – 2;
      if ДО = 0 then ReportError else
        begin
          R := Q div K;
          if R = 0 then ReportError else
            begin
              S := T div R;
            end;
        end;
    end;
end;
```

1.3.1.4. Деякі примітки

Нижче приведені три заслуживающих уваги зауваження про вплив блоків захищених операторів на хід виконання програми.

- Після обробки виняткової ситуації в блоці except продовжується нормальне виконання операторів процедури або функції, розміщених після захищеного блоку.
- Якщо немає операторів для обробки сгенерированной виняткової ситуації, виконання поточної процедури або функції негайно переривається і виключення передається нагору по ланцюжку викликів, поки не буде знайдений підходящий оброблювач.
- Неопрацьовані виняткові ситуації, зрештою, досягають оброблювача за замовчуванням, що видає діалогове вікно з повідомленням про помилку. До оброблювача виключень за замовчуванням попадають усі виключення, для яких оброблювач не написаний вами. Про оброблювача виключень за замовчуванням не потрібно піклуватися — у Delphi він додається автоматично до кожному компилируемому додаткові. Далі в цій главі буде описано, як замінити оброблювач за замовчуванням своїм кодом, що іноді потрібно при створенні складних проектів.

1.3.1.5. Блоки захищених ресурсів

Висновок повідомлення про помилку — це тільки один аспект обробки виняткових ситуацій. Стійкий додаток повинний відновлювати стабільний стан системи при виникненні аварійних ситуацій. Наприклад, при виникненні дискової помилки або вичерпанні ресурсів системи додаток повинний звільнити усі виділені для своїх нестатків блоки пам'яті, що займали би місце доти, поки користувач не виконає перезавантаження. Стійка програма повинна відновлювати після помилок нормальну роботу, закриваючи відкриті файли, звільняючи виділені ресурси Windows і роблячи все можливе для відновлення порядку.

Для створення блоків захищених ресурсів використовуються блоки try і finally. У прикладі 1.3.5 показана основна схема, що дуже нагадує блок захищених операторів. У дійсності різниця лише в тім, що слово except замінене словом finally. Однак, незважаючи на зовнішню схожість, блоки захищених ресурсів і блоки захищених операторів виконуються по-різному.

Приклад 1.3.5 - Схема створення блоку захищених ресурсів

```
{Виділення пам'яті або інших ресурсів.}
try
  {Оператори, що можуть привести до виняткової ситуації.}
finally
  {Звільнити ресурс; виконується в будь-якому випадку}
end;
{Продовжити, якщо в блоці try не виникло виняткових ситуацій.}
```

Оператори блоку finally виконуються завжди, незалежно від того, чи генерується в блоці

try виключення. Звичайно оператори блоку finally служать для звільнення пам'яті, закриття файлів і виконання інших необхідних операцій для відновлення стабільності системи при виникненні виняткової ситуації.

Оператори поза блоком try, що генерують виняткову ситуацію, приводять до негайного виходу з процедури або функції без виконання блоку finally. Щоб гарантувати виконання блоку finally, помістите в блок try усі потенційно “небезпечні” оператори.

Часто при складанні блоку захищених ресурсів у новачків виникають сумніву де помістити оператор виділення ресурсу. Незважаючи на те що цей оператор теж може привести до виняткової ситуації (наприклад, виділення пам'яті може бути не виконане через її недолік), він не повинний належати блокові try. Це правило дуже важливе! Щоб краще запам'ятати його, потрібно просто не забувати про призначення блоку finally. Блок finally призначений для звільнення ресурсів, а як можна звільнити ще невиділені ресурси? Тому оператор виділення ресурсів необхідно поміщати перед блоком try. Усередині цього блоку можна розмістити будь-які оператори, здатні генерувати виняткові ситуації. Виняткові ситуації можуть привести до того, що виділений ресурс буде безцільно займати місце, до того як користувач виконає перезавантаження.

Пояснимо цей важливий метод на простому прикладі, що показує, як можна використовувати блоки захищених ресурсів, щоб запобігти засмічення пам'яті системи. (Засмічення пам'яті може привести до зникнення піктограм, змінам системного шрифту і “зависанню” клавіатури, а також викликати загальні симптоми недостачі ресурсів.)

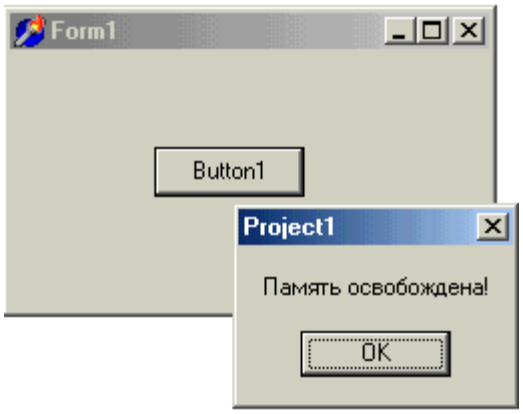
Виконаєте наступні дії.

1) Відкрийте новий додаток.
2) Помістите об'єкт Button у форму.
3) Двічі клацніть на кнопці і заповните процедуру оброблювача події OnClick, використовуючи код із приклада 1.3.6.
4) Натисніть клавішу <F9>, щоб скомпілювати і запустити програму. Клацніть на кнопці, щоб подивитися, що відбудеться, якщо виникне виняткова ситуація в процедурі, у якій виділяється блок пам'яті. На мал. 1.3.4 показане вікно повідомлення, що з'явиться при запуску цієї тестової процедури.

Приклад 1.3.6 - Приклад створення блоку захищених ресурсів

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
  P: Pointer;
begin
  I := 0; J := 10;
  GetMem(P, 4098);  //Виділення пам'яті
  try
    K:= J div I;  //Генерується виключення
    ShowMessage('Results: ' + ' I=' + IntToStr(I) + ' J=' + IntToStr(J) + ' K=' + IntToStr(K));
  finally
    FreeMem(P, 4098);  //Гарантированно виконується
    ShowMessage('Пам'ять звільнена!');
  end;
end;
```

Процедура з приклада 1.3.6 нагадує розглянутий вище блок захищених операторів, але в цьому випадку доданий метод GetMem для виділення 4 098 байт оперативної пам'яті. Зверніть увагу, що цей метод (як і безпечні оператори присвоєння для целочисленных перемінних) поміщений поза блоком try. У блок try укладені потенційно “небезпечні” вираження. Якби операція розподілу була поза блоком try, то при виникненні виняткової ситуації виконання процедури було б негайно припинене, а виділений блок пам'яті так і залишився б загубленим.



Малюнок 1.3.4 - Хоча виняткова ситуація виникає в тестовій процедурі, це повідомлення, доводять, що пам'ять дійсно була звільнена

Блок finally запобігає такій помилці, викликаючи метод FreeMem незалежно від того, чи виникає виняткова ситуація. (Привласніть перемінної I значення 2 і знову запустите програму, щоб перевірити це.) Оператор ShowMessage наприкінці поміщений тільки для демонстрації. На практиці оператори блоків finally звичайно виконуються непомітно для користувача.

Оператори блоку finally виконуються обов'язково, навіть якщо в попередньому блоці try ніяких виняткових ситуацій не виникло.

У блоці finally не обробляються виняткові ситуації — вони можуть бути оброблені тільки в блоці except. Оператори блоку finally гарантоване виконуються незалежно від того, що відбулося в попередньому блоці try. У даному випадку виняткова ситуація не обробляється, і тому при запуску процедури з приклада 1.3.6 оброблювач виняткових ситуацій за замовчуванням видає повідомлення про помилку. Однак, як описано в наступному розділі, можна одночасно використовувати блоки захищених операторів і захищених ресурсів.

1.3.1.6. Вкладені блоки try-except і try-finally

У тих, хто вперше зіштовхується з винятковими ситуаціями, напевно виникає питання, чи можна в одній процедурі одночасно захистити ресурси й обробити виключення. Справа в тім, що це окремі операції, тобто не можна змішувати конструкції try-except і try-finally. У Object Pascal немає такого гібрида, як try-except-finally.

Однак можна вкласти блок try-except усередину блоку try-finally і одночасно обробити виняткові ситуації і запобігти втраті ресурсів. У прикладі 1.3.7 показана основна схема для забезпечення такого додаткового рівня захисту.

Приклад 1.3.7 - Схема створення змішаних блоків захищених операторів і захищених ресурсів

```
{Виділення ресурсу.}
try
  try
    {Оператори, що можуть згенерувати виключення.}
  except
    {Оператори обробки виключень.}
  end;
finally
  {Звільнення ресурсу.}
end;
```

Вкладені блоки try-except і try-finally необхідно використовувати, як показано в прикладі 1.3.7, якщо в блоці except обробляється тільки один визначений тип виняткових ситуацій. У цьому випадку інші типи виняткових ситуацій можуть привести до передчасного завершення процедури. У цьому і попередньому прикладах у блоці except обробляються усі виняткові ситуації, що можуть виникнути в попередньому блоці try. Однак, як буде пояснено далі в цій главі, звичайно перехоплюються тільки визначені виняткові ситуації, а всі інші ідуть нагору по

ланцюжку викликів, можливо, досягаючи оброблювача за замовчуванням. При відсутності блоку try-finally ці й інші необроблювані виняткові ситуації можуть привести до втрати ресурсів.

Нижче в цій главі буде описано, як обробляти визначені типи виключень за допомогою операторів on-do. У прикладі 1.3.8 показаний конкретний приклад схеми 1.3.7. У прикладі 1.3.9 показаний той же самий приклад, але вкладений блок try-except виведений в окрему викликувану функцію, що робить код менш ефективним, але більш наочним. Протестуйте обидва приклади, як і раніш, використовуючи кнопку на формі для виклику процедури. Повідомлення програм говорять про те, що виділена пам'ять звільняється навіть при виникненні виняткової ситуації.

Приклад 1.3.8 - Використання блоку try-except, вкладеного в блок try-finally, для обробки виняткових ситуацій і звільнення ресурсів

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
  P: Pointer;
begin
  I := 0; J := 10;
  GetMem(P, 4098);
  try
    try
      K:= J div I;
      ShowMessage('Results: K=' + IntToStr(K));
    except
      ShowMessage('Помилка розподілу! ' + ' I=' + IntToStr(I) + ' J=' + IntToStr(J));
    end;
  finally
    FreeMem(P, 4098);
    ShowMessage('Пам'ять звільнена!');
  end;
end;
```

Функція й оброблювач події з приклада 1.3.9 еквівалентні процедурі в прикладі 1.3.8, але блок try-except помешен у викликувану функцію GetInt. У результаті код процедури OnClick став більш читабельним.

Приклад 1.3.9 - Використання блоку try-except, вкладеного в блок try-finally, для обробки виняткових ситуацій і звільнення ресурсів

```
function GetInt: Integer;
var
  I, J, K: Integer;
begin
  I := 0; J := 10;
  try
    K:= J div I;  // Генерування виключення.
    Result := ДО;  //Присвоєння результату функції (не виконується).
    ShowMessage('Результат: ДО=' + IntToStr(ДО));
  except
    Result := 0;  //Присвоєння результату функції при помилці.
    ShowMessage('Помилка розподілу! ' + ' I=' + IntToStr(I) + ' J=' + IntToStr(J));
  end;
end;
procedure TForm.ButtonClick(Sender: TObject);
var
  K: Integer;
```

```
  P: Pointer;
begin
  GetMem(P, 4098);  // Виділення пам'яті
  try
    K:= GetInt;  //Може привести до виключення
  finally
    FreeMem(P, 4098);  //Гарантированно виконується.
    ShowMessage('Пам'ять звільнена!');
  end;
end;
```

У прикладі 1.3.9 показано, як обробка виняткових ситуацій відокремлює звичайні операторы програми від логіки обробки помилок. Щоб зрозуміти, що мається на увазі, видалите ключові слова try, except, всі оператори в блоці except і перший оператор end у функції GetInt. Скорочена функція буде виглядати приблизно так:

```
begin
  I := 0;
  J := 10;
  K:= J div I;  //Генерування виключення і вихід із процедури!
  Result := ДО;
  ShowMessage('Результат: ДО=' + IntToStr(ДО));
end;
```

Натисніть клавішу <F9>, щоб відкомпілювати і запустити програму, і потім клацніть на кнопці. Згенерується виняткова ситуація. В оброблювачі події OnClick як і раніше звільняється виділена пам'ять, незважаючи на те, що виняткова ситуація більше не обробляється. Якби не використовувався блок try-finally, то виключення привело б до завершення процедури OnClick, а виділена пам'ять залишилася б незвільненої.

1.3.2. Обробка та генерування виключень

У попередніх розділах були коротко описані основні методи обробки виняткових ситуацій. У цій частині глави ви довідаєтеся, як обробляти визначені типи виключень, а також як генерувати і повторно генерувати виключення. Однак спочатку необхідно розглянути різні параметри середовища, що впливають на обробку виключень системою розробки.

1.3.2.1. Екземпляр виключення

Як згадувалося раніше, виключення — це подія (або ситуація), що перериває виконання програми. Однак фізично виключення являє собою об'єкт класу, звичайно — похідного від класу Exception, визначеного в модулі SysUtils. Цей об'єкт називається *екземпляром виключення*.

Генерування виключення приводить до створення екземпляра виключення. Якщо виключення обробляється в блоці except, то екземпляр автоматично знищується. Якщо немає операторів, що обробляють виключення, то екземпляр передається нагору по ланцюжку викликів, поки для нього не найдеться підходящий оброблювач або поки не буде досягнутий оброблювач за замовчуванням.

Єдиний безпечний спосіб знищити екземпляр виключення — обробити виключення. При обробці виключення екземпляр знищується автоматично. У наступних розділах ви довідаєтеся, як звертатися до екземплярів виключення по посиланню. Ніколи не намагайтеся звільнити або знищити об'єкти виключень, спроба звільнити екземпляр приведе до фатальної помилки додатка.

1.3.2.2. Обробка деяких виключень

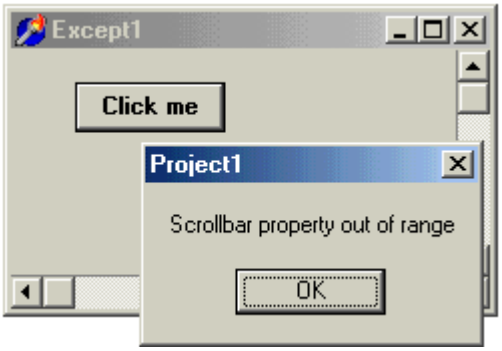
Екземпляр виключення визначає тип виниклої виняткової ситуації. У Delphi мається безліч класів, похідних від класу Exception. Кожний з них служить для опису визначеного типу виключень. Інформацію класу можна використовувати для поліпшення обробки виняткових

ситуацій. Також можна написати окремі оброблювачі для різних класів виключень. Для виконання цих задач у блоці except використовуються оператори on-do.

Оператор on-do виконує двох задач: дозволяє розпізнати визначений тип виключення й одержати посилання на екземпляр виключення. Звичайно екземпляр виключення використовується для одержання інформації про те, яка саме виникла проблема, і цю інформацію можна використовувати для дозволу проблеми і видачі повідомлень користувачеві.

1.3.2.3. Звернення до екземпляру виключення

У додатку Except1, показано, як можна використовувати оператор on-do для звертання по посиланню до екземпляра виключення. На мал. 1.3.5 показане вікно програми і діалогове вікно, викликуване щигликом на кнопці. У додатку виникає виняткова ситуація при спробі привласнити некоректні значення двом елементам керування вікна - смугам прокручування. У прикладі 1.3.10 приведений оброблювач події OnClick для кнопки.



Малюнок 1.3.5 - Діалогове вікно з'являється після щиглика на кнопці

Приклад 1.3.10 – Процедура обробки події OnClick

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
  try
    ScrollBar1.SetParams(0, 500, 0);
    ScrollBar2.SetParams(0, 500, 0);
  except
    on E: Exception do
      ShowMessage(E.Message);
  end;
end;
```

Програмою викликається метод SetParams для двох смуг прокручування, але, щоб викликати виняткову ситуацію, аргументи Min і Max навмисне розташовані в зворотному порядку. Для компонента TScrollBar потрібно, щоб аргумент Min був менше або дорівнює аргументові Max, і тому при першому виклику методу SetParams генерується виняткова ситуація.

У блоці except оператором on-do з'являється перемінна E класу Exception. (Можна використовувати будь-як ім'я, але ім'я E зручне у використанні.) У цьому операторі не створюється новий екземпляр класу Exception; у ньому просто створюється посилання (E) на екземпляр виключення, у якому описана виникла проблема. У програмі використовується властивість виключення Message для відображення повідомлення в діалоговому вікні.

Однак перехоплювання в такий спосіб кожного типу виключень — це не краще рішення. Краще отлавливать тільки визначені типи виключень. При такому підході програма відповідає тільки за виключення, обробка яких підготовлена, наприклад за невірний параметр, розподіл на нуль і т.п. А більш серйозні помилки, такі як збої пам'яті, передаються оброблювачеві за замовчуванням.

1.3.2.4. Перехоплення певних типів виключень

Оператор on-do можна використовувати для перехоплення визначеного типу виключень. Наприклад, обробляти виключення розподілу на нуль і дозволити іншим оброблювачам,

розташованим вище по ланцюжку викликів, обробляти інші виключення. Укажіть необхідний клас виключень в операторі on-do, як показано в прикладі 1.3.11. Процедура в цьому прикладі практично не відрізняється від процедури з приклада 1.3.2, але цього разу обробляється тільки виключення EDivByZero.

Приклад 1.3.11 - Створення блоку захищених операторів з обробкою визначеного типу виключень

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
begin
  I := 0; J := 10;
  try
    K:= J div I;
  except
    on E: EDivByZero do
      begin
        ShowMessage(E.Message + ' I=' + IntToStr(I) + ' J=' + IntToStr(J) + ' K=' + IntToStr(K));
      end;
  end;
end;
```

Зверніть увагу, що за допомогою оператора ShowMessage спочатку відображається повідомлення E.Message, що визначено для всіх об'єктів виключень як строкова властивість. Часто в діалогових вікнах повідомлень про помилки виводиться цей рядок.

1.3.2.5. Класи виключень

Нижче перераховані класи виключень, визначені в різних модулях Delphi. Цей список не повний, але охоплює найбільш необхідні виняткові ситуації. Усі ці класи є похідними (необов'язково безпосередньо) від класу Exception. Приклад їхнього використання показаний у прикладі 1.3.11. Наприклад, щоб перехопити помилку EPrinter, можна скористатися таким кодом:

```
try
  {Операції печатки.}
except
  on E: EPrinter do
    ShowMessage(E.Message);  //Вивести повідомлення про помилку
end;
```

- ***EAbort***. “Тихе” виключення, генерируемое при виклику процедури Abort.
- ***EAbstractError***. Генерується при спробі викликати абстрактний метод. Якщо в програмі створюється екземпляр класу з одним або декількома абстрактними методами, при компіляції видається попередження. Виключення EAbstractError можна одержати, тільки якщо ви ігнорували це попередження.
- ***EAccessViolation***. Указує на одну з трьох помилок доступу: використання покажчика nil, запис у сторінку пам'яті, що містить код, і спроба доступу до невиділеної сторінки пам'яті. Не генеруйте це виключення явно.
- ***EArrayError***. Указує на одну з трьох заборонених операцій з масивом: індекс, що виходить за рамки індексного простору, спроба додавання елемента до масиву фіксованого розміру і вставка елемента в збережений масив. (Див. також EBitsError.)
- ***EAssertionFailed***. Генерується процедурою Assert при передачі логічного вираження зі значенням False.
- ***EBitsError***. Указує на одну або двох некоректних операцій з логічним масивом (при використанні класу TBits): негативний індекс або індекс, що виходить за рамки

індексного простору.

- ***EClassNotFound***. Генерується, якщо в програмі виробляється спроба звертання до об'єкта класу, не зв'язаного з додатком (наприклад, якщо оголошення класу було вилучено з вихідного коду форми або програмою виробляється спроба зчитування невідомого об'єкта з потоку).
- ***EComponentError***. Генерується при будь-яких помилках при реєстрації компонента. Також генерується, якщо в програмі виконується спроба одержати для компонента інтерфейс COM, а інтерфейс COM не підтримується компонентом.
- ***EControlC***. Тільки для консольних додатків. Генерується, якщо користувач натискає <Ctrl+C>, і ніколи не генерується додатками Windows.
- ***EConvertError***. Генерується при помилках перетворення, таких як спроба перетворення рядка з нечисловим символом у ціле число за допомогою методу StrToInt.
- ***EDatabaseError***. Загальна помилка баз даних, сгенерированная компонентами доступу до даних і керування даними. Про спеціальні виключення, зв'язаних з базами даних, можна довідатися з оперативної довідки Delphi. Шукайте класи виключень з іменами, що починаються з EDB, наприклад EDBClient і EDBEditError.
- ***EDateTimeError***. Генерується при введенні некоректного значення часу або дати для об'єкта компонента TDateTimePicker.
- ***EDivByZero***. Розподіл на нуль при операціях з цілими числами.
- ***EFCreateError***. Указує на помилку при створенні файлу.
- ***EFilerError***. Указує на помилку в операціях з файловим потоком.
- ***EFOpenError***. Указує на помилку відкриття файлу.
- ***EInOutError***. Помилка файлового виводу-введення-висновку. У поле ErrorCode екземпляра виключення утримується код помилки виводу-введення-висновку. Ця виняткова ситуація може виникнути, тільки якщо програма скомпільована з директивою {$I + } (вона використовується за замовчуванням). Коди помилок наступні: 2 — файл не знайдений, 3 — невірне ім'я файлу, 4 — занадто багато відкритих файлів, 5 — доступ заборонений, 100 — кінець файлу, 101 — диск переповнений і 106 — некоректне введення. Також припустимі інші коди помилок. (Читайте керівництво MS Dos Technical Reference фірми Microsoft і іншу документацію по інтерфейсі Win32.)
- ***EIntError***. Основний клас для математичних помилок при операціях з цілими числами. Об'єкти цього класу виключень ніколи не створюються, однак клас можна використовувати для перехоплення всіх помилок у целочисленных операціях. Для створення об'єктів виключень використовуйте похідні класи, такі як EDivByZero, ERangeError і EIntOverflow.
- ***EIntfCastError***. Генерується при спробі некоректного приведення типів з використанням оператора as.
- ***EIntOverflow***. Переповнення для перемінної типу Integer.
- ***EInvalidArgument***. Вихідне за припустимі рамки значення в спеціалізованих математичних і фінансових функціях з модуля Math.
- ***EInvalidCast***. Невірне вираження приведення типів.
- ***EInvalidGraphic***. Генерується програмою при спробі відкриття нерозпізнаваного графічного файлу (наприклад, відкриття помилково текстового файлу замість файлу крапкового зображення *.bmp).
- ***EInvalidGraphicOperation***. Указує на некоректну графічну операцію, таку як спроба

змінити розмір піктограми.

- ***EInvalidGridOperation***. Генерується при некоректних операціях з компонентами grid, наприклад при спробі звертання до неіснуючого осередку.
- ***EInvalidOp***. Указує на некоректну операцію з коми, що плаває. Генерується при невизначених математичних помилках, наприклад коли для процесора передається невизначена інструкція, виробляється спроба виконати некоректну операцію або переповняється стек процесора.
- ***EInvalidOperation***. Указує на некоректну операцію над компонентом, зокрема на операцію, що вимагає дескриптора вікна для компонента з невизначеною властивістю Parent. Також може бути сгенерирован при некоректних операціях перетаскування.
- ***EInvalidPointer***. Генерується при використанні невірного покажчика. Наприклад, при спробі звертання до покажчика nil або передачі виділеного блоку пам'яті більш одного разу.
- ***EMathError***. Базовий клас для математичних помилок при роботі з числами з крапкою, що плаває. Ніколи не використовується прямо як об'єкт виключення. Використовуються похідні класи виключень EInvalidArgument, EInvalidOp, EOverflow, EUnderflow і EZeroDivide.
- ***EMCIDeviceError***. Указує на помилку при роботі з драйвером MCI (Media Control Interface).
- ***EMenuError***. Указує на некоректну операцію з меню. Може бути сгенерирован, наприклад, при некоректному звертанні до об'єкта меню.
- ***ENoResultSet***. Генерується об'єктом TQuery, якщо в запиті відсутній оператор SELECT.
- ***EOleCtrlError***. Указує на невдалу спробу встановлення зв'язку з елементом керування Active.
- ***EOleException***. Генерується, якщо виклик методу IDispatch привів до помилки.
- ***EOleSysError***. Генерується при невдалому виклику методу IDispatch.
- ***EOutOfMemory***. Генерується, якщо неможливо виділити необхідну пам'ять.
- ***EOutOfResources***. Генерується при невдалій спробі створити дескриптор Windows.
- ***EOverflow***. Переповнення при роботі з числами з крапкою, що плаває, (значення занадто велике).
- ***EPackageError***. Генерується при виникненні помилок, зв'язаних з пакетами.
- ***EPrinter***. Указує на помилку, що виникла під час печатки.
- ***EPrivilege***. Генерується при порушенні рівнів привілею процесора, наприклад, при спробі виконати операцію, неприпустиму для даного рівня привілею.
- ***EPropertyError***. Генерується, якщо властивість об'єкта не можна установити, наприклад якщо значення, що привласнюється, виходить за рамки припустимих або має некоректний тип даних.
- ***EPropReadOnly***. Указує на спробу запису у властивість, призначена тільки для читання (при використанні технології OLE).
- ***EPropWriteOnly***. Указує на спробу читання з властивості, призначеного тільки для запису (при використанні технології OLE).
- ***ERangeError***. Значення, що виходить за припустимі рамки, для індексування масиву або короткого рядка, а також для присвоєння перемінним скалярного або перечислимого типу. Ця виняткова ситуація не генерується для довгих рядків. Щоб виняткова ситуація була сгенерирована, програма повинна компілюватися з

директивою {$R+}, що звичайно робиться тільки під час налагодження. За замовчуванням використовується директива {$R-}, а це виключає перевірку діапазонів і відповідно можливість виникнення цієї виняткової ситуації.

- ***EReadError***. Генерується при некоректній спробі вважати дані потоку, а також якщо неможливо вважати дані властивості при створенні об'єкта форми.
- ***ERegistryException***. Указує на помилку при виконанні операцій з реєстром Windows, таку як спроба змінити область реєстру з невідповідними привілеями користувача.
- ***EResNotFound***. Генерується, якщо визначений ресурс, наприклад піктограма, не знайдена. Розповсюджена причина цієї виняткової ситуації — вилучена або закомментированная директива {$R *.DFM} у розділі implementation модуля форми.
- ***ESocketError***. Указує на помилку при роботі з об'єктом сокета Windows.
- ***EStackOverflow***. Генерується, якщо стік поточного потоку досяг останньої сторінки пам'яті, тобто програмі грозить недостача пам'яті. Причиною цієї проблеми звичайно є великі локальні перемінні в процедурах і функціях, а також глибоко вкладені рекурсивні підпрограми. (Перемістите великі перемінні в глобальну область.)
- ***EStreamError***. Базовий клас для помилок файлових потоків. Об'єкт цього класу створюється, якщо не удається виділити потік.
- ***EStringListError***. Указує на помилку в операціях зі строковим списком, наприклад при звертанні до списку по невірному індексі.
- ***EThread***. Указує на проблему, зв'язану із синхронізацією потоків.
- ***ETreeViewError***. Генерується при використанні невірного індексу для компонента TTreeView.
- ***EUnderflow***. Утрата значимості при операціях з числами з крапкою, що плаває, (значення занадто мале).
- ***EVariantError***. Указує на помилку, зв'язану з невірним використанням варіантного типу даних, наприклад на таку, як невірне приведення типу, або індекс, що виходить за межі індексного простору.
- ***EWin32Error***. Генерується при виникненні помилки Windows. Оброблювачем за замовчуванням виводиться діалогове вікно з кодом помилки і рядком повідомлення. Щоб одержати повідомлення операційної системи, можна використовувати функцію Win32Check з модуля SysUtils.
- ***EWriteError***. Генерується при помилках запису у файловий потік.
- ***EZeroDivide***. Розподіл на нуль при операціях з числами з крапкою, що плаває.

Ці класи також можна використовувати тільки для визначення проблеми, не створюючи посилання на екземпляр виключення. Наприклад, оператор on-do у прикладі 1.3.11 можна замінити наступним оператором:

```
on EDivByZero do
  ShowMessage('Помилка розподілу');
```

У першому рядку порівнюється клас екземпляра виключення з класом EDivByZero і, якщо об'єкт належить цьому класові, викликається метод ShowMessage. Виключення інших типів передаються нагору по ланцюжку викликів. Цей метод використовується, якщо не потрібно звертатися по посиланню до екземпляра виключення. Для звертання по посиланню до екземпляра виключення E и відображення відповідного повідомлення можна використовувати і такий код:

```
on E: EDivByZero do
  ShowMessage(E.Message);
```

Щоб довідатися про властивості визначених типів виключень, звернетеся до оперативної довідки Delphi і знайдіть визначення цих класів. З визначень класів можна довідатися, які властивості є в класу, крім властивості Message. Наприклад, у модулі SysUtils клас EInOutError

визначений у такий спосіб:

```
EInOutError = class(Exception)
public
  ErrorCode: Integer;
end;
```

З цього оголошення видно, що, крім властивості Message (наслідуваного всіма класами виключень від класу Exception), у класі EInOutError також мається властивість ErrorCode.

1.3.2.6. Обробка декількох виключних ситуацій

У блоці except можна обробляти кілька різних типів виключень. Наприклад, у наступному фрагменті програми показано, як обробити три типи об'єктів виключень (інші типи передаються нагору по ланцюжку викликів):

```
on EDivByZero do
  …;
on EInOutError do
  …;
on EOutOfMemory do
  …;
```

Також можна визначити екземпляри виключень для доступу до іншої інформації, властивої визначеним типам виключень. Наприклад, у наступному фрагменті програми перехоплюються три типи виключень і визначається екземпляр виключення E для кожного типу:

```
try
  {Оператори, що можуть генерувати наступні виняткові ситуації.}
except
  on E: EDivByZero do
    ShowMessage(E.Message + ' (I=0)');
  on E: EInOutError do
    with E do
      ShowMessage(Message + ' #:' + IntToStr(ErrorCode));
  on E: EOutOfMemory do
    begin
      ShowMessage(E.Message);
      //Звільнення виділеної пам'яті.
    end;
end;
```

У кожнім оброблювачі виключення в попередньому фрагменті програми по-різному використовується інформація об'єкта E. У першому оброблювачі перехоплюються помилки розподілу на нуль і виводиться властивість Message об'єкта E плюс рядок з додатковою інформацією про те, що саме відбулося не так.

В другому оброблювачі перехоплюються помилки виводу-введення-висновку. У ньому використовується оператор with для доступу до властивостями об'єкта E и відображається рядок E.Message разом з перемінної E.ErrorCode, визначеної в класі EInOutError. Це значення перетвориться в рядок за допомогою функції IntToStr.

У третьому оброблювачі обробляються помилки недостачі пам'яті. У ньому виводиться повідомлення E.Message. Коментар поміщений туди, де варто звільнити виділений блок пам'яті, що міг бути виділений програмою для одержання додаткової пам'яті. Блок варто звільняти для того, щоб дозволити користувачеві спокійно закрити додаток або перервати операцію, приведшую до недостачі пам'яті.

При використанні операторів on-do для обробки визначених класів виключень генерування інших типів неопрацьованих виключень приведе до негайного припинення роботи процедури. Тому в таких випадках рекомендується використовувати блок try-finally для звільнення виділеної пам'яті, закриття файлів і виконання інших необхідних операцій.

1.3.2.7. Генерування нових виключних ситуацій

Більшість виняткових ситуацій генерується автоматично. Але, щоб повідомити про помилку або незвичайні умови, можна генерувати і власні виняткові ситуації.

Для генерування виняткової ситуації використовується ключове слово raise. Після оператора raise створюється екземпляр виключення класу Exception або якого-небудь похідного класу. Припустимо, що перемінна ErrorFlag типу Boolean служить індикатором помилки (значення True відповідає помилці). Тоді виняткову ситуацію можна згенерувати програмним шляхом:

```
if ErrorFlag then
  raise Exception.Create('Error');
```

Замість класу Exception можна використовувати більшість описаних вище класів. Також можна генерувати виняткові ситуації власних класів, про що більш докладно буде розповідатися нижче. Результат виконання цих операторів збігається з генеруванням виняткових ситуацій компонентами, математичними вираженнями, збоями устаткування або будь-яких інших джерел.

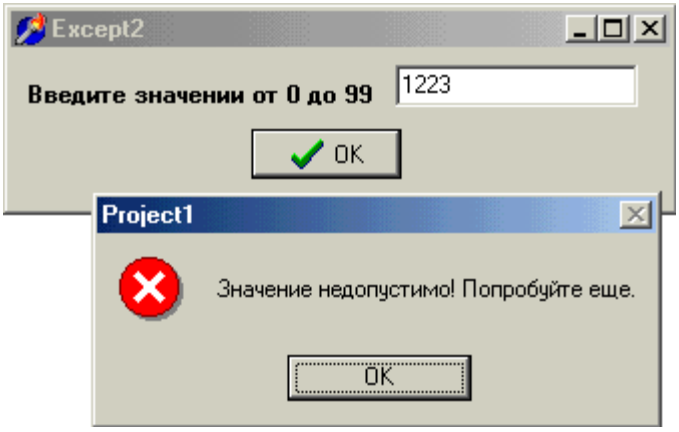
Можна написати оброблювач виняткових ситуацій для обробки виключень, сгенерированных програмно. Також можна дозволити оброблювачеві виняткових ситуацій, установленому додатком за замовчуванням, перехопити них.

Сгенерированная виняткова ситуація негайно переходить до найближчого оброблювача виняткових ситуацій у процедурі або функції, у якій викликаний оператор raise. Якщо ж оброблювач не утримується в даній процедурі або функції, то відбувається негайний вихід з неї.

Щоб побачити, як генеруються виняткові ситуації, розглянемо програму Except2. Необхідно ввести число від 0 до 99 у поле **Edit**. На мал. 1.3.6 показана виведена програмою інформація. Для перевірки введеного значення на наявність помилок і виходу з програми необхідно натиснути кнопку **OK**. Якщо ввести яке-небудь заборонене число, наприклад 100, у поле **Edit** сгенерированная виняткова ситуація не дозволить завершити програму. Оброблювач виключень за замовчуванням відображає повідомлення про помилку. У прикладі 1.3.12 описаний оброблювач події OnClick.

Приклад 1.3.12 - Оброблювач події OnClick

```
procedure TMainForm.Button1Click(Sender: TObject);
var
  N: Integer;
begin
  N := StrToInt(Edit1.Text);
  if (N < 0) or (N > 99) then
    raise ERangeError.Create('Значення неприпустиме! Спробуйте ще.')
  else
    begin
      ShowMessage('Вийшло! Клацніть на кнопці OK, щоб завершити програму.');
      Close;
  end;
end;
```



Малюнок 1.3.6 - Уведення числа, що виходить за межі припустимих значень, приводить до генерування виняткової ситуації, що не дозволяє завершити програму

В оброблювачі події OnClick об'єкта Button за допомогою функції StrToInt текстова перемінна з текстового поля об'єкта Edit1 перетвориться в число типу integer і результат заноситься в перемінну N. За допомогою оператора if виконуються відповідні перевірки і передається керування операторові, що генерує виняткову ситуацію, за умови, що N не попадає в діапазон від 0 до 99. За допомогою оператора raise створюється екземпляр класу виключень ERangeError. У даній програмі оброблювачем виключень за замовчуванням буде отриманий об'єкт ERangeError, після чого з'явиться діалогове вікно з повідомленням про помилку.

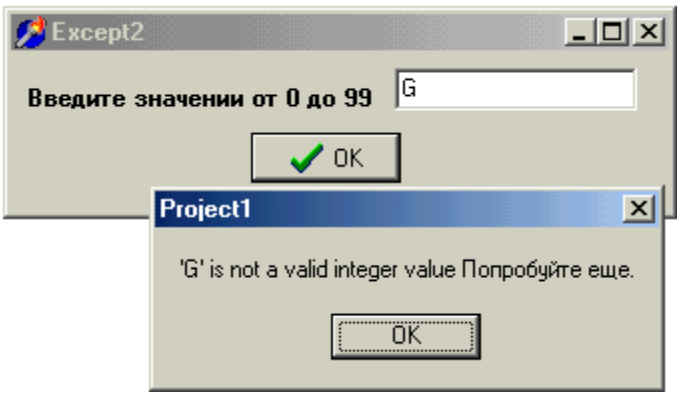
Менш очевидним фактом є те, що простий оброблювач подій може також дати збій через виникнення виняткової ситуації іншого типу, наприклад помилки перетворення типів, сгенерированной при виклику функції StrToInt. Якщо ввести в поле Edit не целочисленное значення, а 3.14159 або буквений символ G, а потім клацнути на кнопці, відобразиться інше повідомлення, таке як 3.14159 is not a valid Integer value (3.14159 не є числом типу Integer). Воно виводиться на екран оброблювачем виключень за замовчуванням. Більш важливим є те, що виняткова ситуація, сгенерированная при виклику функції StrToInt, стане причиною негайного виходу з програми. Причому наступні оператори процедури будуть пропущені.

Найкращою версією оброблювача цієї події буде оброблювач з використанням блоку try-except. Це дасть користувачеві можливість виправити неправильно введене значення незалежно від причини виникнення помилки. У прикладі 1.3.13 показаний остаточний варіант оброблювача події OnClick.

Приклад 1.3.13 - Оброблювач події OnClick

```
procedure TMainForm.Button1Click(Sender: TObject);
var
  N: Integer;
  S: String;
  //Вкладений оброблювач виключень.
  procedure Handler(Message: String);
  begin
    ShowMessage(Message + ' Спробуйте ще.');
    Edit1.SetFocus;
  end;
begin
  try
    N := StrToInt(Edit1.Text);
    if (N < 0) or (N > 99) then
      raise ERangeError.Create('Значення неприпустиме!')
    else
      begin
        ShowMessage('Вийшло! Клацніть на кнопці OK, щоб завершити програму.');
```

```
      Close;
    end;
  except
    on E: EIntError do Handler(E.Message);
    on E: EConvertError do Handler(E.Message);
  end;
end;
```



Малюнок 1.3.7 – Висновок повідомлення при виклику виняткової ситуації EConvertError у додатку Except2

У приведеній процедурі використовується вкладена процедура для висновку повідомлення про помилку з доданими словами **Спробуйте ще**. У ній також викликається метод SetFocus об'єкта Edit1, щоб максимально спростити для користувача введення нового значення відразу після прочитання повідомлення про помилку. В основному тілі процедури генерується виняткова ситуація, якщо значення N виходить за припустимі межі. Був використаний і блок try для того, щоб інші типи виняткових ситуацій, що можуть бути сгенерированы при виклику методу StrToInt, теж були оброблені нашою програмою. У блоці except описано, як надходити з двома типами помилок: помилками при перетворенні типів, генерируемыми методом StrToInt, і помилками входження в інтервал, генерируемыми самою процедурою. У будь-якому випадку операторами on-do буде викликаний вкладений оброблювач для висновку повідомлення про помилку і передане керування назад полю редагування. Інші типи помилок усе ще передають керування оброблювачеві виключень за замовчуванням.

Результат роботи програми при введенні як значення символу G відображений на малюнку 1.3.7.

Одна процедура обробки не повинна бути вкладена в іншу процедуру. Вона повинна описуватися, як окрема процедура або як метод класу. Це необхідно робити, наприклад, для того, щоб кілька процедур могли викликати один оброблювач.

1.3.2.8. Повторне генерування виключних ситуацій

При розробці додатків приходиться писати безліч процедур і функцій, що забезпечують обробку виняткових ситуацій. І часто необхідно додати деякі нові можливості до вже існуючих підпрограм. Наприклад, мається програма, у якій закривається файл, якщо відбулася помилка при звертанні до диска. В іншому модулі може також знадобитися обробка подібної виняткової ситуації. Замість того щоб описувати закриття файлу в двох різних місцях (такий метод свідчить про недостатню практику в програмуванні, тому що приводить до ускладнення майбутнього супроводу програми), можна задати вторинний оброблювач, що буде повторно генерувати виняткову ситуацію. У цьому випадку виняткова ситуація залишиться діючої так, що після виконання вторинним оброблювачем своїх дій об'єкт виключення перейде по ланцюжку викликів, поки не досягне відповідного оброблювача.

Якщо ви знайомі з объектно-ориентрованным програмуванням, то можете уявити собі повторне генерування виняткової ситуації як розподіл на підкласи під час виконання програми. Оброблювач виняткової ситуації можна доповнити іншими оброблювачами. Вони будуть

перехоплювати спеціальні типи виключень, виконувати деякі дії і потім повторно генерувати виняткову ситуацію, щоб залишити її діючої для додаткової обробки. Ця ситуація схожа із ситуацією, коли в методі похідного класу викликається метод наслідуваного класу, щоб додати до нього додаткові оператори.

Для обробки виняткової ситуації без руйнування екземпляра виключення використовується ключове слово raise без аргументів. Для всіх типів виняткових ситуацій використовується така схема (коментарі показують, де можна розміщати один або трохи операторів, що можуть згенерувати первісну виняткову ситуацію):

```
try
  {Оператори, що генерують виключення.}
except
  ShowMessage('Помилка!') ;
raise;
  {Повторне генерування виключення.}
end;
```

Якщо який-небудь оператор у блоці try згенерує виняткову ситуацію, то буде виконаний перехід до методу ShowMessage. Після цього оператор raise без аргументів повторно згенерує виняткову ситуацію, і на екрані з'являться два повідомлення: перше — виведене методом ShowMessage, друге — виведене оброблювачем виключень, установленим за замовчуванням. (Деякі виняткові ситуації є “тихими”, повідомлення про них не виводяться. Більш докладно це описано нижче.)

У попередньому фрагменті програми перехоплюються усі виняткові ситуації, що, узагалі говорячи, нерозумно і може привести до великих проблем. Наприклад, може бути неправильно оброблена помилка недостачі пам'яті. Однак, тому що усі виняткові ситуації генеруються повторно, оброблювач виключень за замовчуванням як і раніше має можливість обробляти критичні помилки. За винятком рідких випадків, коли маються досить вагомі причини для ігнорування цього правила, у блоці except, що перехоплює усі виняткові ситуації, варто завжди викликати raise, як було описано вище.

Щоб повторно згенерувати конкретні типи виняткових ситуацій, використовуйте звичайну схему, але в блок on-do вставляйте оператор raise, як показано в даному фрагменті програми:

```
try
  {Оператори, що генерують виключення.}
except
  on E: EOverflow do
    begin
      ShowMessage(Помилка!');
      raise;  //Повторне генерування виключення.
    end;
end;
```

Тут перехоплюється екземпляр класу EOverflow, виводиться повідомлення і повторно генерується це виключення. Інші виняткові ситуації, що виникли в операторах блоку try, приведуть до виходу з процедури або функції, а блок except буде пропущений.

Повторне генерування виняткової ситуації приводить до негайного виходу з процедури або функції, у якій був викликаний оператор raise.

1.3.3. Створення класів виключних ситуацій

Коли необхідно згенерувати виняткову ситуацію, можна скористатися одним із класів Delphi, похідних від Exception (переглянете приведений вище список). Також можна створити свій власний клас виняткових ситуацій, похідний від Exception, або новий клас, що буде вашою

власною розробкою, тобто не буде заснований на класі Exception.

Якщо ви хочете, щоб оброблювачем за замовчуванням перехоплювалися всі необроблювані виняткові ситуації нового класу, зробіть цей клас похідним від Exception. Узагалі говорячи, така схема є найбільш прийнятної, але якщо розпізнавання оброблювачем за замовчуванням сконструйованих вами виключень не потрібно, те можна створити новий клас незалежно від Exception.

Усі необроблювані виняткові ситуації класу, що не є похідним від класу Exception, приведуть до фатальних помилок додатка. У програмі *повинні* оброблятися *всі можливі* виняткові ситуації, що не базуються на класі Exception.

1.3.3.1. Класи виключних ситуацій користувача

Користувальницькі класи виключень дуже корисні для нагромадження інформації про помилки, що можуть відбутися в програмі. Наприклад, математична процедура, що виконує деякі обчислення, може повертати заборонені значення перемінних в об'єкті користувальницького класу виняткових ситуацій. У такому випадку з'явиться можливість у програмному блоці except вивести на екран ці значення, і користувачі вашої програми одержать інформацію, що вони зможуть використовувати для запобігання помилок, наприклад, при введенні коректних значень у діалогове вікно. Ніхто не оцінить повідомлення про помилку *Помилка файлового введення-висновку*, а повідомлення *Ім'я файлу містить неприпустимий символ @* дасть куди більше інформації. Користувальницькі класи виключень можна використовувати для створення таких повідомлень про помилку. Користувальницькі класи виключень дуже корисні і при налагодженні програм.

Найпростіший спосіб створити користувальницький клас виключень — визначити новий клас, заснований на класі Exception. Наприклад, уставте наступне оголошення type у розділ interface модуля (це, звичайно, можна зробити й у частині реалізації, але краще створити клас, доступний для інших модулів, щоб у них можна було створювати виключення нового класу і керувати ними):

```
type
  TCustomException = class(Exception);
```

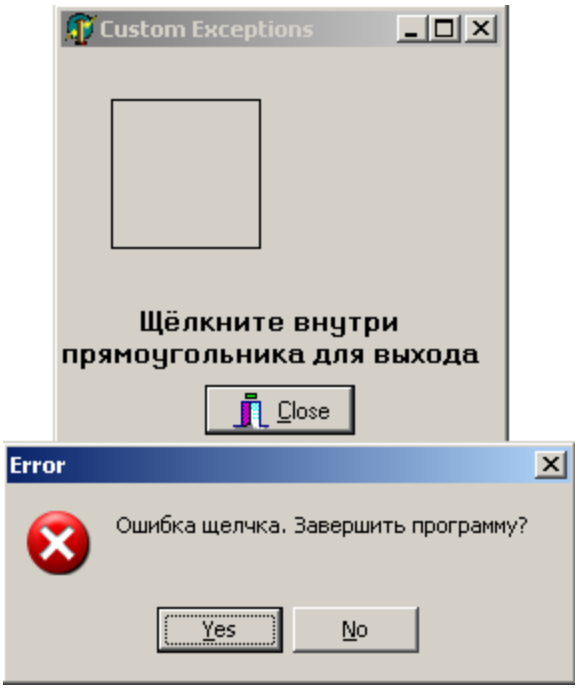
Новий клас просто забезпечує унікальне ім'я для ідентифікації специфічного типу помилки. Тому клас не має потреби в тілі. З цим оголошенням можна генерувати виняткові ситуації класу TCustomException і перехоплювати помилки даного специфічного типу. Наприклад, можна вставити наступний оператор в оброблювач події OnClick:

```
raise TCustomException.Create('Custom exception');
```

Коли ви запустите програму і клацнете на кнопці, оброблювач виключень за замовчуванням одержить екземпляр виключення і виведе повідомлення. Ваш власний оброблювач виняткової ситуації може також перехоплювати об'єкти TCustomException, наприклад:

```
try
  {Оператори, що генерують виключення.}
except
  on E: TCustomException do ShowMessage(E.Message);
end;
```

Даний фрагмент програми схожий на інші фрагменти, розглянуті в цій главі, але тут перехоплюються виключення тільки класу TCustomException. Інші ж типи виключень передаються нагору по ланцюжку викликів, поки для них не буде знайдені відповідний оброблювач або не буде досягнутий оброблювач виключень за замовчуванням.



Малюнок 1.3.8 - Додатку MouseExcept показує, як створювати і застосовувати користувальницькі класи виключень. Клацніть усередині маленького прямокутника для виходу з програми; клацніть де-небудь в іншій частині вікна, щоб штучно згенерувати виняткову ситуацію

Одна з причин створення власного класу виключенні складається в збереженні значення, що забезпечує більш докладну інформацію про помилку. Припустимо, що програма повинна визначити, чи клацнув користувач усередині графічної фігури, намальованою програмою. Оскільки фігура не є об'єктом, потрібно відстежити всіх щигликів у вікні форми, де користувач може клацнути, і відреагувати на це відповідним чином, наприклад здійснити вихід, якщо користувач клацнув мишею усередині фігури. Звичайно ж, це можна організувати, використовуючи компоненти Delphi, але наша задача — продемонструвати правильне застосування користувальницьких класів виключень.

Додаток MouseExcept показує як створювати і застосовувати користувальницькі класи виключень. На мал. 1.3.8 представлені результати роботи програми, а в прикладі 1.3.14 приведений її вихідний текст.

Приклад 1.3.14 - MouseExcept\Main. pas

```
unit Main;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
type
  TMainForm = class(TForm)
    BitBtn1: TBitBtn;
    Label1: TLabel;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure FormPaint(Sender: TObject);
    private
     {Оголошення закритих (private) членів.}
      procedure CheckMouseLocation(X, Y: Integer);
      procedure ExitOnClick(X, Y: Integer);
    public
      {Оголошення загальнодоступних (public) членів.}
```

```
    end;
var
  MainForm: TMainForm;
implementation
{$R *.DFM}
const
  rLeft = 25;
  rTop =25;
  rRight = 100;
  rBottom = 100;
type
  TMouseException = class(Exception)
    X, Y: Integer;
    constructor Create(const Msg: string; XX, YY: Integer);
  end;
constructor TMouseException.Create(const Msg: string; XX, YY: Integer);
begin
  X := XX;  //Зберегти значення X і Y в об'єкті.
  Y := YY;
{Створити рядок повідомлення.}
  Message := Msg + ' (X=' + IntToStr(X) + ', Y=' + IntToStr(Y) + ')';
end;
procedure TMainForm.CheckMouseLocation(X, Y: Integer);
begin
  if (X < rLeft) or (X > rRight) or (Y < rTop) or (Y > rBottom) then
    raise
      TMouseException.Create('Невірний щиглик.', X, Y);
end;
procedure TMainForm.ExitOnClick(X, Y: Integer);
begin
  try
    CheckMouseLocation(X, Y);  //Невірні значення приводять до виникнення виключення.
    Close;  //Вихід із програми
  except
    on TMouseException do
      begin
        if MessageDlg('Помилка щиглика. Завершити програму?',
           mtError, [mbYes, mbNo], 0) = mrYes then
            Close
        else
          raise;
      end;
  end;
end;
procedure TMainForm.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  ExitOnClick(X, Y);
end;
procedure TMainForm.FormPaint(Sender: TObject);
begin
```

```
  Canvas.Rectangle(rLeft, rTop, rRight, rBottom);
end;
end.
```

Використовуючи постійні значення, описані в частині реалізації модуля, у програмі MouseExcept за допомогою оброблювача події OnPaint можна намалювати невеликий прямокутник для початкової форми. Якщо клацнути де-небудь у вікні, оброблювач події OnMouseDown, процедура FormMouseDown, одержить значення координат покажчика миші x и y. Ця процедура викликає процедуру ExitOnClick, передаючи ці x и y як аргументи. Процедура ExitOnClick є закритою і вона додана в клас форми.

У блоці try процедури ExitOnClick виконуються два оператори. Спочатку викликається процедура CheckMouseLocation, у якій перевіряється, чи попадає крапка з координатами x и y в прямокутник. Потім у процедурі використовується метод close для виходу з програми. Якщо виникають якісь проблеми з визначенням x и y, а також якщо користувач клацнув поза прямокутником, то оператор close пропускається, тому що в процедурі CheckMouseLocation генерується виключення. Після генерування виключення на екрані з'явиться повідомлення про помилку і пропозиція вийти з програми. Ця процедура написана таким чином, що якщо ви відповісте No у діалоговому вікні повідомлення, то в блок except повторно згенерується виняткова ситуація. У програмі показані обидві реакції на користувальницькі виняткові ситуації — власна реакція програми і реакція оброблювача за замовчуванням.

Об'єкт виключення визначений як екземпляр класу, що у програмі описаний так:

```
type
  TMouseException = class(Exception)
    X, Y: Integer;
    constructor Create(const Msg: string; XX, YY: Integer);
  end;
```

Клас TMouseException є похідним із класу Exception. На додаток до властивостей, наслідуваним від класу Exception, у класі TMouseException мається два числа типу Integer — X і Y, а також власний конструктор, що необхідний для ініціалізації об'єкта даного класу. Вихідний текст конструктора виглядає так:

```
constructor TMouseException.Create(const Msg: string; XX, YY: Integer);
begin
  X := XX;  //Зберегти значення X и Y в об'єкті.
  Y := YY;
  {Створити рядок повідомлення.}
  Message := Msg + ' (X=' + IntToStr(X) + ', Y=' + IntToStr(Y) + ')';
end;
```

Перші два оператори конструктора заносять передані значення параметрів XX і YY у перемінні X і Y класу TMouseException. У строкову перемінну Message (яка успадковується від Exception) заноситься рядок з повідомленням і цими двома значеннями.

Для того щоб зрозуміти, як застосовується користувальницький клас виключень, гляньте на процедуру CheckMouseLocation. Якщо крапка з координатами X і Y знаходиться поза прямокутником, намальованого програмою, то процедура згенерує виключення класу TMouseException, використовуючи наступний оператор:

```
raise
  TMouseException.Create('Mouse location error', X, Y);
```

Цим оператором створюється об'єкт TMouseException, у якому зберігаються значення x и y поточної позиції курсору, і виконується негайний вихід із процедури CheckMouseLocation. Повернемося до процедури ExitOnClick. Якщо виняткова ситуація була сгенерирована на попередньому кроці, то оператор Close буде пропущене і керування передане блокові except.

На перший погляд, цей приклад здається надмірно складним. У великих додатках з безліччю графічних зображень клас, подібний TMouseException, може сильно спростити

програму. Крім того, без користувальницьких класів виключень у програмі неминуче використовувалася би безліч умовних операторів if, щоб перевірити місце розташування покажчика миші, що сильно утрудняє процес налагодження. Збір всієї обробки помилок у користувальницький клас виключень і використання блоків try-except для виявлення помилок зменшують кількість операторів і поліпшують процес пошуку помилок.

1.3.3.2. Базовий клас виключних ситуацій

Якщо клас виключень є похідним від класу Exception, то в ньому успадковується безліч членів вихідного класу. Наприклад, базовий клас Exception забезпечує велика кількість методів для створення об'єктів виняткових ситуацій різними способами. Імена всіх цих конструкторів починаються з Create і кожний з них грає свою роль у створенні виведених строкових повідомлень про помилки для різних аргументів. Ці конструктори можна викликати для створення екземплярів виключень похідних класів. Коротко зупинимося на деяких конструкторах класу Exception. Один з них обов'язково вам підійде.

Найпростіший конструктор Create уже зустрічався в прикладах цієї глави. Для нього просто передається строкова перемінна або константа як повідомлення про помилку:

raise Exception.Create('Trouble in Paradise');

Уставте попередній оператор в оброблювач події Ondick об'єкта Button, запустите програму і клацніть на кнопці, щоб побачити результат. Виконаєте це для інших прикладів, розглянутих у даному розділі.

Щоб створити повідомлення, що містить додаткову інформацію, наприклад значення перемінних, використовується альтернативний конструктор CreateFmt. Для цього конструктора передається рядок з командами форматування і значення, які потрібно вставити в рядок. Наприклад, у наступному операторі генерується виняткова ситуація з повідомленням про помилку, у якому приводяться два значення перемінних типу integer:

raise Exception.CreateFmt('Error: X=%d Y=%d', [X, Y]);

У результаті виконання цього оператора користувач побачить наступне повідомлення про помилку:

Error: X=-l Y=12

При створенні об'єктів виняткових ситуацій даним методом можна використовувати набір з ресурсів рядків повідомлень про помилки, убудованих у Delphi, що утримуються в модулі SysConst. У табл. 1.3.1 перераховані константи і зв'язані з ними рядка. Ці рядки в будь-якому випадку будуть підключаться до вашого додатка, тому краще використовувати них, чим визначати свої власні. Для їхнього використання додайте оголошення uses у частину реалізації модуля:

```
uses
  SysConst;
```

Наприклад, що випливає оператор згенерує виняткову ситуацію, використовуючи одну з визначених констант:

raise Exception.Create(SDiskFull);

Таблиця 1.3.1 - Стандартні повідомлення про помилки, визначені в модулі SysConst

Ідентифікатор	*Рядок*
SInvalidInteger	'"%s" is not a valid integer value';
SInvalidFloat	'"%s" is not a valid floating point value';
SInvalidDate	'"%s" is not a valid date';
SInvalidTime	'"%s" is not a valid time';
SIinvalidDateTime	'"%s" is not a valid date and time';
STimeEncodeError	'Invalid argument to time encode';
SDateEncodeError	'Invalid argument to date encode';
SOutOfMemory	'Out of memory';
SInOutError	'I/O error %d';

Ідентифікатор	*Рядок*
SFileNotFound	'File not found';
SInvalidFilename	'Invalid filename';
STooManyOpenFiles	'Too many open files';
SAccessDenied	'File access denied';
SEndOfFile	'Read beyond end of fife';
SDiskFull	'Disk full';
SInvalidInput	'Invalid numeric input';
SDivByZero	'Division by zero';
SRangeError	'Range check error';
SIntOverflow	'Integer overflow';
SInvalidOp	'Invalid floating point operation';
SZeroDivide	'Floating point division by zero';
SOverflow	'Floating point overflow';
SUnderflow	'Floating point underflow';
SInvalidPointer	'Invalid pointer operation';
SInvalidCast	'Invalid class typecast';
SAccessViolation	'Access violation at address %p. %s of address %p';
SStackOverflow	'Stack overflow';
SControlC	'Control-C hit';
SPrivilege	'Privileged instruction';
SOperationAborted	'Operation aborted';
SException	'Exception %s in module %s at %p.'#$OA'%s%s';
SExceptTitle	'Application Error';
SInvalidFormat	'Format "%s" invalid or incompatible with argument';
SArgumentMissing	'No argument for format "%s"';
SInvalidVarCast	'Invalid variant type conversion';
SInvalidVarOp	'Invalid variant operation';
SDispatchErro	'Variant method calls not supported';
SReadAccess	'Read';
SWriteAccess	'Write';
SResultTooLong	'Format result longer than 4096 characters';
SFormatTooLong	'Format string too long';
SVarArrayCreate	'Error creating variant array';
SVarNotArray	'Variant is not an array';
SVarArrayBounds	'Variant array index out of bounds';
SExternalException	'External exception %x';
SAssertionFailed	'Assertion failed';
SIntfCastError	'Interface not supported'
SAssertError	'%s (%s, line %d)';
SAbstractError	'Abstract Error';
SModuleAccessViolation	'Access violation at address %p in module "%s". %s of address %p';
SCannotReadPackageInfo	'Cannot access package information for package "%s"';
SErrorLoadingPackage	'Can"t load package %s.'#13#10'%s';
SInvalidPackageFile	'Invalid package file "%s"';
SInvalidPackageHandle	'Invalid package handle';
SDuplicatePackageUnit	'Cannot load package "%s." It contains unit "%s,"' + ';which is also contained in package "%s"';
SWin32Error	'Win32 Error. Code: %d.'#10'%s';
SUnkWin32Error	'A Win32 API function failed';
SNL	'Application is not licensed to use this feature';

Інші конструктори класу Exception є варіаціями попередніх. Нижче приведений приклад використання всіх конструкторів. Для конструктора CreateResFmt завантажується строкотабличный ресурс з убудованою командою форматування %s (зверніть увагу, що цьому конструкторові необхідно передавати набір рядків). При використанні конструктора CreateHelp користувач одержує можливість вивести довідку помилково (використання ідентифікатора контекстної довідки залежить від вашого оброблювача). У конструкторі CreateFmtHelp комбінується використання рядка форматування й ідентифікатора контекстної довідки. У конструкторі CreateResHelp комбінується використання строкотабличных ресурсів і ідентифікатора контекстної довідки. У конструкторі CreateResFmtHelp використовуються всі параметри (%s або %d) і ідентифікатор контекстної довідки.

```
raise Exception.CreateResFmt(SInvalidInteger, [IntToStr(N)]);
raise Exception.CreateHelp('Ой, простите', 123);
raise Exception.CreateFmtHelp('Помилка N=%d', [N], 123);
raise Exception.CreateResHelp(SFileNotFound, 123);
raise Exception.CreateResFmtHelp(SInvalidInteger, [S], 123);
```

1.3.4. Інші методи роботи з виключеннями

У наступних розділах обговорюються більш складні методи, що має сенс використовувати в спеціальних, або виняткових, ситуаціях.

1.3.4.1. “Тихі” виключення

“Тихі” виключення — це виключення, що не роблять видимого впливу на програму. Однак вони корисні для переривання вкладених процесів. Екземпляр “тихого” виключення — це об'єкт класу EAbort, похідного від класу Exception. Оброблювач за замовчуванням запрограмований так, що не виводить повідомлення про помилки при одержанні виключення EAbort. Замість цього він просто знищує екземпляр виключення і повертає керування програмі.

Для генерування виключення класу EAbort використовується процедура. Наприклад, нехай цикл while продовжується за умови істинності логічного прапора NormalCondition і в ньому перевіряється інший прапор (SpecialConditionFlag). Тоді вихід їхнього циклу можна виконати за допомогою методу Abort:

```
while NormalCondition do
  begin
    if SpecialConditionFlag = False then Abort;
  end;
```

Виклик методу Abort еквівалентний генеруванню виключення EAbort. В останньому методі ще потрібно вказати рядок повідомлення для об'єкта (однак, оскільки це виключення “тихе”, програмою цей рядок не відображається):

```
raise EAbort.Create('Операція перервана.');
```

Важливість “тихих” виключень у тім, що вони не роблять ніякого видимого ефекту. Цикл закінчується після генерування виключення класу EAbort, виключення переходить до оброблювача за замовчуванням, що повертає керування програмі. Передачу рядка можна використовувати для налагодження, наприклад якщо в програмі є оброблювач виключень, що перехоплює усі виключення, у тому числі і EAbort. Якщо ж перехоплювати “тихі” виключення не потрібно (щоб вони залишалися “тихими”), можна використовувати блок try-except (у ньому викликається метод ShowMessage, якщо виключення не належить класові EAbort):

```
try
  {Оператори, у яких може бути викликаний метод Abort.}
except
  on E: Exception do
    if not (E is EAbort) then ShowMessage(E.Message);
    raise;
```

end;

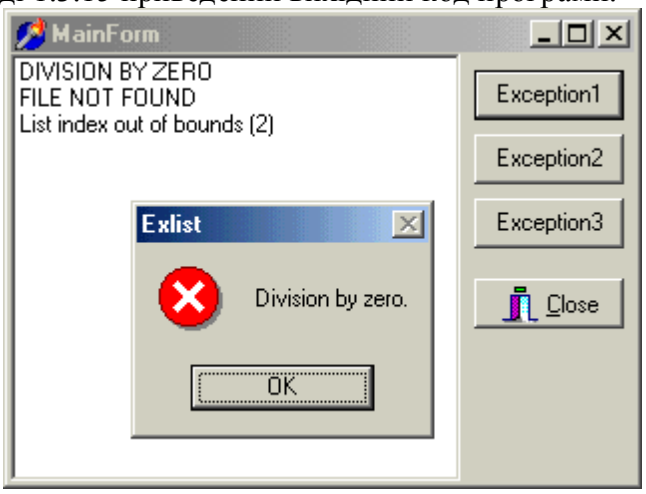
1.3.4.2. Заміна обробника виключень по умовчанню

Для по-справжньому серйозної обробки помилок можна написати користувальницький оброблювач виключень за замовчуванням, що працює на рівні об'єкта Application. Цей метод надає доступ до всіх необроблюваних виключень, що можуть придатися як для налагодження можливостей обробки виключень у додатку, так і для заміни діалогового вікна оброблювача виключень за замовчуванням більш наочним і зрозумілим.

Навіть якщо для вашого додатка не потрібно писати обработчик виключень за замовчуванням, я рекомендую випробувати цей метод. Можливість перехоплення необроблюваних виключень на рівні додатків потенційно корисна для налагодження обробки помилок у програмі. У прикладі додатка цей метод використовується для підтримки списку всіх необроблюваних виключень у ході виконання програми. Такий список може бути безцінним для відстеження джерела виникнення проблем.

У компоненті TApplication мається процедура HandleException, за допомогою якої обробляються будь-які необроблювані виняткові ситуації, що виникають під час роботи програми. Не варто зловживати оброблювачами виключень за замовчуванням і цілком рятуватися від них. Це майже напевно приведе до серйозних проблем. Замість того щоб нарощувати оброблювач виключень за замовчуванням, можна написати процедуру для події OnException об'єкта Application. Тоді замість висновку повідомлення про помилку за замовчуванням метод HandleException викликає процедуру OnException, якщо, звичайно, вона визначена.

На мал. 1.3.9 показаний додаток ExList. Усі необроблювані виключення проходять через новий оброблювач. Цей оброблювач додає повідомлення про помилку в компонент ListBox. У такий спосіб у додатку забезпечується повне відстеження необроблюваних виняткових ситуацій. При запуску програми щиглик на одній із трьох кнопок Exception генерує один із трьох типів виключень. Після закриття останнього діалогового вікна за замовчуванням програмою будуть додані повідомлення про помилки в компонент форми ListBox. Після щиглика на кнопці Exception1 генерується виняткова ситуація “розподіл на нуль”. Щигликом на кнопці Exception2 викликається виняткова ситуація “файл не знайдений”. Після щиглика на кнопці Exception3 усі перераховані рядки представляються у виді прописних букв, але в програмі утримується навмисна помилка, що викликає виняткову ситуацію “вихід індексу за припустимі межі”. У прикладі 1.3.15 приведений вихідний код програми.



Малюнок 1.3.9 - У додатку ExList показано, як написати оброблювач виключень за замовчуванням

Приклад 1.3.15 - ExList\Main.pas

unit Main;

interface

uses

```
  Windows, SysUtils, Messages, Classes, Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls;
type
  TMainForm = class(TForm)
    ListBox1: TListBox;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    {Private declarations}
  public
    {1. Оголошення оброблювача події OnExcaption.}
    procedure NewOnException(Sender: TObject; E: Exception);
  end;
var
  MainForm: TMainForm;
implementation
{$R *.DFM)
{2. Виконання оброблювача події OnException.}
procedure TMainForm.NewOnException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  ListBox1.Items.Add(E.Message);
end;
{3. Призначення оброблювача події Application.OnException.}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Application.OnException := NewOnException;
end;
{Генерування виняткової ситуації розподілу на нуль.}
procedure TMainForm.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
begin
  I := 0; J := 10;
  try
    K:= J div I;  {Розподіл на нуль!}
{Наступний оператор не виконується, але потрібний для того, щоб оптимізатор Object Pascal не видалив попередній оператор, визначивши, що значення більше К в цій процедурі не використовується.}
    ShowMessage('K=' + IntToStr(ДО));
  except
    raise;
   end;
end;
{Генерування виняткової ситуації "файл не знайдений".}
procedure TMainForm.Button2Click(Sender: TObject);
```

```
var
  T: TextFile;
begin
  AssignFile(T, 'XXXX.$$$');
  Reset(T);  {Відкрити неіснуючий файл!}
  try
    {Тут звичайно використовується T.}
  finally
    CloseFile(T);  {З метою безпеки.}
  end;
end;
{Генерування виняткової ситуації "вихід індексу за припустимі межі".}
procedure TMainForm.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  with ListBox1.Items do
    for I := 0 to Count do  {Повинно бути Count - 1!}
      Strings[I] := Uppercase(Strings[I]);
end;
end.
```

Привласнити оброблювач події об'єкта Application сутужніше, ніж, скажемо, події OnClick об'єкта Button, тому що об'єкт Application недоступний для Object Inspector Delphi. Отже, щоб створити оброблювач події OnException, потрібно виконати три дії, що пронумеровані в коментарях у прикладі (ці коментарі виділені напівжирним шрифтом, щоб них було легше знайти):

1) Оголосите обработчик події OnException у класі форми. Він повинний мати параметри, показані в прикладі, хоча ім'я процедури може бути обране на ваше бажання.
2) Реалізуйте обработчик події OnException. Не забудьте почати його ім'я з імені класу форми і крапки.
3) Призначте обработчик події Application.OnException. Методом HandleException об'єкта Application перевіряється, чи призначена програмою процедура для події OnException. Якщо процедура призначена, то вона викликається методом HandleException для будь-якої необроблюваної виняткової ситуації, що одержує метод.

Користувальницький оброблювач події onException одержує тільки виключення, виведені з класу Exception, крім “тихого” виключення EAbort.

В оброблювачі події OnException можна виконати будь-як необхідні дії. У прикладі програми першим оператором викликається метод додатка ShowException. Ця дія виконується оброблювачем за замовчуванням. Крім того, у програмі використовується метод Add для строкового списку Items компонента ListBox1, щоб зберегти копію кожного повідомлення про помилку. Помітимо, що оброблювач події одержує об'єкт виключення, як параметр E. (Цей об'єкт десь знищується; вам не потрібно його звільняти.)

Оброблювач події за замовчуванням можна “змусити мовчати”, видаливши виклик методу Application.ShowException. Однак так надходити не рекомендується, оскільки це може привести до того, що серйозний недолік у вашій програмі залишиться непоміченим. Ваш оброблювач події OnException, можливо, буде відображати деякі повідомлення у відповідь на одержувані необроблювані виключення.

1.3.5. Корисні поради

✍ Виняткові ситуації можна вважати засобом для повернення декількох типів об'єктів із

процедур і функцій. Виражаючи формально, ми могли б генерувати виключення, щоб усюди передавати об'єкти і використовувати блоки try-except для перехоплення цих об'єктів. Не рекомендується надходити таким чином, оскільки можна вийти за рамки припустимого. Це могло б привести до втрати продуктивності мінімум у 100, якщо не в 1 000, раз через просте повернення значення з функції, але це корисно для розуміння того, що використовувати виключення для контролю над помилками необов'язково. Значення об'єктів виключень (особливо — приналежним вашим власним класам) визначається вами довільно.

✍ Методом за замовчуванням HandleException компонента TApplication пересилається повідомлення Windows wm_CancelMode, щоб розблокувати миша, якщо вона була захоплена в той час, коли виникла виняткова ситуація. При цьому також закриваються поля списків, полючи зі списками, що розкриваються, що розкриваються списки і меню.

✍ При побудові ієрархії класів виключень уважно розмістите будь-які операторы on-do, що посилаються на нові класи, щоб спочатку обробити об'єкти виключень, що розташовані далі по ієрархії від класу Exception. Розглянемо наступний код, що аналогічний іншим прикладам з цієї глави (в операторі ShowMessage просто використовується буква до, тому оптимізатор Object Pascal не видаляє попередній оператор розподілу):

```
try
  K:= J div I;
  ShowMessage('ДО=' + IntToStr(ДО));
except
  on E: EDivByZero do  //Цей оператор повинний йти першим!
    ShowMessage('Помилка розподілу.');
  on E: EIntError do  //Всі інші помилки перехоплюються тут.
    ShowMessage('Інша математична помилка при роботі з цілими числами.');
end;
```

✍ У цьому коді використовуються два оператори on-do (їхній може бути скільки завгодно в блоці except). Першим оператором перехоплюється виключення EDivByZero, можливо, щоб виконати спеціальну обробку помилки цього типу. Другим оператором перехоплюються всі інші виключення, похідні від класу EIntError, що є базовим класом і для EDivByZero. Було би помилкою переставити оператори on-do, оскільки, якщо клас виключень, що породжує, буде перехоплений першим, у числі інших буде оброблене і виключення EDivByZero (усі виключення є похідними від класу Exception). Тому спеціальна обробка даного виключення виконана не буде. У додатку Delphi маються два оброблювачі подій за замовчуванням. Один з них забезпечується бібліотекою VCL і призначений для захисту кожного вікна процедури, через яке проходить потік повідомлень. Цим оброблювачем перехоплюються необроблювані виключення і виводиться повідомлення про помилку. Виконання програми продовжується. Інший оброблювач виключення за замовчуванням розташований у бібліотеці часу виконання RTL (Run-Time Library) і надається в розділі SysUtils. Він знаходиться в ієрархії нижче оброблювача VCL за замовчуванням, тому малоймовірно, що ви побачите його в роботі. Якщо цим оброблювачем перехоплюється виключення, виводиться докладне повідомлення (з адресними даними для налагодження), а потім завершується додаток. Такі виняткові ситуації непоправні.

1.3.6. Резюме

Досвідчені розроблювачі піклуються про обробку помилок під час написання програми. Механізми обробки виняткових ситуацій у Delphi дозволяють легко виправити всі помилки.

Виняткові ситуації можуть виникнути з багатьох причин. Їхніми джерелами можуть стати компоненти, математичні вираження, операції файлового введення-висновку, операції з пам'яттю і т.д. Виняткову ситуацію можна згенерувати навмисне, щоб передати інформацію про помилку іншим операторам.

У блоках захищених виражень використовуються ключові слова try-except для обробки виключення, що виникло в операторах блоку try.

У блоках захищених ресурсів використовуються ключові слова try-finally. Ці блоки використовуються для виконання необхідних завершальних операцій, у тому числі при виникненні виняткової ситуації в блоці try.

Виключення — це об'єкт класу (звичайно — похідного від класу Exception), у якому описані основні властивості виключення як об'єкта. Він створюється при генеруванні виняткової ситуації і знищується при її обробці. Ніколи не намагайтеся звільнити екземпляр виключення явно!

Для обробки визначених типів виключень застосовуються оператори on-do у блоці except. При використанні цього методу помнете, що усі виняткові ситуації, не оброблювані операторами on-do, приводять до припинення виконання процедури або функції після блоку except. У таких випадках рекомендується використовувати блоки try-finally для звільнення ресурсів.

Для штучного генерування виняткової ситуації використовується ключове слово raise з конструктором для нового екземпляра виключення. Щоб повторно згенерувати виключення в блоці except, використовується оператор raise без аргументів.

Можна створювати свої класи виключень “з нуля” або на базі класу Exception. У більшості випадків варто створювати нові класи виключень на базі класу Exception, оскільки оброблювачем за замовчуванням розпізнаються виключення тільки таких класів.

У класі Exception мається безліч конструкторів для створення повідомлень про помилки на основі рядків, значень і ресурсів строкових таблиць.

Для генерування “тихого” виключення класу EAbort використовується метод Abort. Оброблювачем за замовчуванням не виводиться вікно повідомлення для виключень EAbort.

Можна додати свій оброблювач виключень за замовчуванням. Для цього варто оголосити, реалізувати і привласнити оброблювач події OnException у класі, похідному від TForm. Даний метод використовується для пошуку неопрацьованих виключень або для заміни діалогового вікна, що з'являється по умовчанню, з повідомленням про помилку.

Література [1, 2].

РОЗДІЛ 2 БАЗИ ДАНИХ

Тема 2.1. Робота з базами даних

Мова Delphi не тільки надає всі засоби програмування для Windows, але і є втіленою мрією розроблювача баз даних. За допомогою Delphi можна створювати і редагувати програмне забезпечення для будь-яких типів баз даних форматів dBASE, Paradox і системи ODBC, застосовуваної в Microsoft Access. Крім того, можна розробляти додатка клієнт/сервер для операцій з вилученими серверами даних. Такі додатки можуть працювати в мережах різного масштабу — від невеликих локальних з персональними комп'ютерами до глобальних зі стаціонарними машинами. Тому, хто працює з Delphi, ніколи не знадобиться інша система керування базами даних.

У цій главі ви познайомитеся з компонентами Delphi, що забезпечують доступ до інформації з баз даних і її обробку, а також з методикою їхнього використання. Після оволодіння основами нескладно навчитися пошуку інформації в базі за допомогою мови запитів SQL (Structured Query Language — мова структурованих запитів) і створювати додатка реляционных баз даних на основі моделі “головний/підлеглий”.

Усі версії Delphi включають інструмент Borland Database Engine (BDE), що забезпечує повний набір програмних засобів для роботи з таблицями більшості популярних форматів систем баз даних, таких як dBASE і Paradox. Редакція Client-Server Delphi, що включає BDE, забезпечує доступ до вилучених серверів баз даних, таким як Oracle, Sybase, Microsoft SQL Server і Informix.

2.1.1. Компоненти

Для роботи з базами даних у Delphi призначені наступні компоненти.

- ***TBatchMove***. Виконує пакетні операції з записами і таблицями, наприклад дублювання набору даних, додавання записів з одного набору даних в іншій і відновлення або видалення записів по визначеній ознаці.
  Категорія палітри: Data Access.
- ***TDatabase***. Забезпечує додаткові можливості обробки баз даних, такі як реєстрація на сервері і використання локальних псевдонімів. Об'єкти компонента Database при необхідності створюються Delphi автоматично, але користувач може створювати їхній і явно за власною ініціативою.
  Категорія палітри: Data Access.
- ***TDataSource***. Зв'язує компоненти набору даних Table або Query з компонентами зв'язку з даними, такими як DBEdit і DBMemo. У будь-якому додатку баз даних обов'язково повинний бути об'єкт DataSource.
  Категорія палітри: Data Access.
- ***TDBChart***. Повноцінний компонент для створення діаграм на основі інформації з бази даних.
  Категорія палітри: Data Controls.
- ***TDBCheckBox***. Зв'язаний з даними компонентів TCheckBox.
  Категорія палітри: Data Controls.
- ***TDBComboBox***. Зв'язаний з даними компонентів TComboBox.
  Категорія палітри: Data Controls.
- ***TDBCtrlGrid***. Прокручиваемый набір панелей, кожна з яких представляє один запис бази даних. Кожна панель може містити один або більш об'єктів керування зв'язком з даними. Звичайно використовується з об'єктом DBNavigator для табулированного (не табличного) перегляду даних.

Категорія палітри: Data Controls.

- ***TDBEdit***. Зв'язаний з даними компонент введення однорядкового тексту TEdit.
  Категорія палітри: Data Controls.
- ***TDBGrid***. Зв'язана з даними сітка з декількох стовпців і рядків, у якій відображаються записи бази даних (по одному записі в кожнім рядку). Звичайно використовується з об'єктом DBNavigator для перегляду даних у виді таблиці.
  Категорія палітри: Data Controls.
- ***TDBImage***. Зв'язаний з даними графічний компонент TImage. Звичайно використовується для відображення об'єктів типу BLOB (Binary Large Object — великий двоичный об'єкт), що містять растрові зображення.
  Категорія палітри: Data Controls.
- ***TDBListBox***. Зв'язаний з даними компонентів TListBox.
  Категорія палітри: Data Controls.
- ***TDBLookupComboBox***. Зв'язаний з даними компонентів TComboBox з можливістю пошуку потрібної таблиці. У ранніх версіях Delphi він називався TDBLookupCombo.
  Категорія палітри: Data Controls.
- ***TDBLookupListBox***. Зв'язаний з даними компонентів TListBox з можливістю пошуку потрібної таблиці. У ранніх версіях Delphi він називався TDBLookupList.
  Категорія палітри: Data Controls.
- ***TDBMemo***. Зв'язаний з даними компонент уведення многострочного тексту TMemo.
  Категорія палітри: Data Controls.
- ***TDBNavigator***. Складний діалоговий засіб перегляду і редагування бази даних. За допомогою кнопок компонента TDBNavigator користувач може переміщатися по записах бази даних, уставляти нові записи, стирати запису і виконувати інші операції.
  Категорія палітри: Data Controls.
- ***TDBRadioGroup***. Зв'язаний з даними компонентів TRadioGroup.
  Категорія палітри: Data Controls.
- ***TDBRichEdit***. Як і елемент керування TDBMemo, відображає і дозволяє редагувати текстові дані у форматі RTF (Rich Text Format — розширений текстовий формат).
  Категорія палітри: Data Controls.
- ***TDBText***. Зв'язаний з даними текстовий компонент, що працює в режимі тільки для читання. Відображає інформацію з бази даних, для якої не потрібне редагування з боку користувача. (Для формування написів біля полів уведення даних в екранних формах використовується стандартний компонент Label, а не DBText.)
  Категорія палітри: Data Controls.
- ***TQuery***. Формує SQL-запити до Borland Database Engine або сервера SQL.
  Категорія палітри: Data Access.
- ***TSession***. Усі приєднання до бази даних відбуваються в контексті об'єкта компонента TSession, що керує цими з'єднаннями. Для будь-якого додатка бази даних Delphi автоматично створює глобальний об'єкт компонента TSession. Однак для забезпечення декількох сеансів зв'язку в додаток можна додавати компоненти TSession, наприклад для доступу до таблиць, що знаходиться на різних комп'ютерах мережі.
  Категорія палітри: Data Access.
- ***TStoredProc***. Дозволяє додаткові виконувати збережені процедури на сервері бази даних. Цей компонент необхідний, в основному, при розробці систем доступу до баз даних класу клієнт/сервер.
  Категорія палітри: Data Access.

- ***TTable***. Надає додаткові можливість доступу до баз даних за допомогою Borland Database Engine. Цей компонент звичайно зв'язаний з об'єктом DataBase.
  Категорія палітри: Data Access.
- ***TUpdateSQL***. Цей компонент призначений для розробок, у яких потрібно обновити набори даних, призначених тільки для читання, що формує SQL-сервер. За допомогою компонента TUpdateSQL виконуються команди INSERT, UPDATE і DELETE навіть у наборах даних, позначених як тільки для читання (read-only). Це відбувається, наприклад, коли додаток опитує в одному запиті кілька таблиць, навіть якщо самі таблиці не позначені як тільки для читання. Цей компонент призначений лише для розроблювачів з великим досвідом побудови СУБД класу клієнт/сервер. Для більш простих задач рекомендується використовувати компонент TQuery.
  Категорія палітри: Data Access.

В елементів керування, зв'язаних з даними, є двійники, що не зв'язані з даними, але можуть використовувати інформацію з баз даних. Наприклад, компонент TDBListBox схожий на звичайний елемент керування ListBox, але має можливість одержувати інформацію з бази даних, представленої об'єктом Table, за допомогою об'єкта DataSource.

У ранніх версіях Delphi на вкладці палітри Data Access мався компонент Report, що забезпечував доступ до генератора звітів по базі даних ReportSmith, розробленому фірмою Borland. Цього компонента в новій версії більше немає. Для створення звітів по базі даних тепер існують вкладки палітри QReport.

2.1.2. Створення баз даних

Перераховані вище компоненти дозволяють упровадити концепції объектно-ориентированного програмування в розробку додатків, що мають справа з базами даних. І, що більш важливо, компоненти баз даних стандартизують доступ до баз даних різних форматів. Це значить, що додаток може одержати доступ до даних, що знаходиться у файлах формату dBASE, таблицях Paradox, файлах Microsoft Access або іншої системи ODBC (Open Database Connectivity — відкрита сумісність баз даних) або в редакції Client-Server Delphi на вилученому SQL-сервері. Крім того, у додатках баз даних можуть використовуватися й інші компоненти Delphi, методи взаємодії і програмування на Object Pascal.

2.1.2.1. Робота з майстром форм баз даних

Додаток баз даних, на відміну від іншого Windows-сумісного програмного забезпечення, здатно зчитувати інформацію з таблиць баз даних і записувати неї. Виражаючи термінами користувальницького інтерфейсу програми, екранна форма додатка будується так само, як будь-яке інше вікно.

У звичайній екранній формі, призначеної для роботи з базою даних, повинні бути елементи керування введенням, написи і сітки, а також компоненти, що забезпечують зв'язок з таблицями бази даних. Програмувати всі ці об'єкти вручну досить утомливо, тому для розробки нового додатка баз даних краще скористатися шаблоном майстра розробки форм баз даних (Database Form Wizard; надалі — *майстер форм БД*). У ранніх версіях Delphi він називався експертом форм баз даних (Database Form Expert) і знаходився в меню Help. Для запуску майстра форм БД потрібно відкрити новий додаток за допомогою команди File⇨New, клацнути на корінці вкладки Business, а потім вибрати піктограму **Database Form Wizard**.

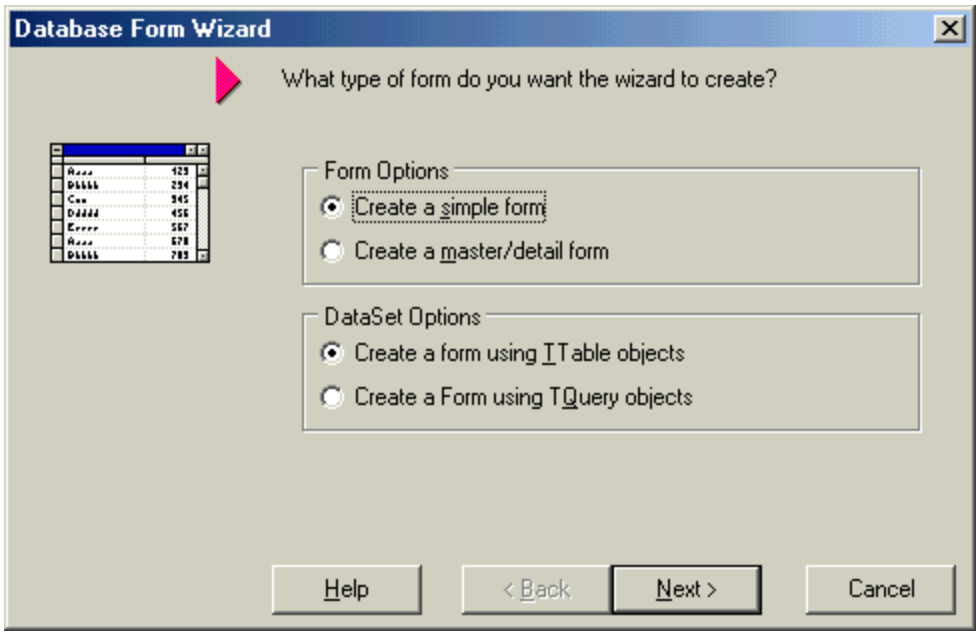
Майстер форм БД — це інтерактивний засіб для конструювання екранних форм баз даних. Після серії коротких відповідей на питання і вибору опцій майстер форм БД створить нову екранну форму, належним образом укомплектовану компонентами баз даних. Можна модифікувати отриману форму, переміщати в ній компоненти, але чаші усього для одержання закінченого додатка цій формі потрібна лише незначна остаточна обробка.

За допомогою наступних операцій можна створити форму для таблиці з приклада бази даних у Delphi або з будь-якої іншої таблиці бази даних.

1) Виберіть пункт меню File⇨New, щоб відкрити новий додаток. Виберіть вкладку **Business** і двічі клацніть на піктограмі **Database Form Wizard**.
2) Майстер форм БД послідовно відкриє трохи діалогових вікон з опціями для створення різноманітних форм. Простежите, щоб у першому вікні були обрані опції **Create a simple form** і **Create a form using TTable objects** (мал. 2.1.1). Це установки, задані за замовчуванням.
3) Клацніть на кнопці **Next** для переходу до наступного вікна. При необхідності можете повернутися в попереднє вікно, клацнувши на кнопці **Prev.**
4) Виберіть таблицю в комбінованому списку з позначкою **Drive or Alias Name**. Виберіть **DBDEMOS** або будь-який інший псевдонім бази даних з наявних у системі. У списку **Table Name** знаходяться назви таблиць, що складають базу даних **DBDEMOS.** Виберіть Animals.dbf або будь-яку іншу таблицю (мал. 2.1.2), а потім клацніть на кнопці **Next** для переходу до наступного вікна.
5) Тепер майстер видасть список полів обраної таблиці. Для переносу всіх полів у список обраних клацніть на кнопці з подвійною стрілкою, а для переносу окремих полів виділите них за допомогою миші, утримуючи натиснутої клавішу <Ctrl> або <Shift>, і перетягнете них у список обраних. Виконуючи виділення, перетаскування і щигликів на кнопках зі стрільцями нагору і вниз, розташуєте поля в потрібному порядку. Після щиглика на кнопці **Next** відкриється наступне вікно майстра.
6) Визначите порядок розміщення полів. При виборі опції **Horizontal** полючи записів будуть розташовуватися друг за другом, а при виборі опції **Vertical** — одне під іншим. При виборі опції **In a grid** дані упорядкуються у виді таблиці, у рядках якої будуть запису, а в стовпцях — полючи. Наприклад, виберіть опцію **In a grid**, клацніть на кнопці **Next**, і відкриється наступне вікно майстра.
7) В останнім діалоговому вікні майстер форм БД пропонує створити або екранну форму Form, або і форму Form, і модуль даних DataModule з невізуальними компонентами. (Майстри попередніх версій Delphi не використовували модулів даних.) У цьому прикладі переконаєтеся, що встановлено прапорець **Create a main form**, і виберіть опцію **Form and DataModule**. Клацніть на кнопці **Finish** (у попередніх версіях — **Create**) для створення нової форми бази даних.

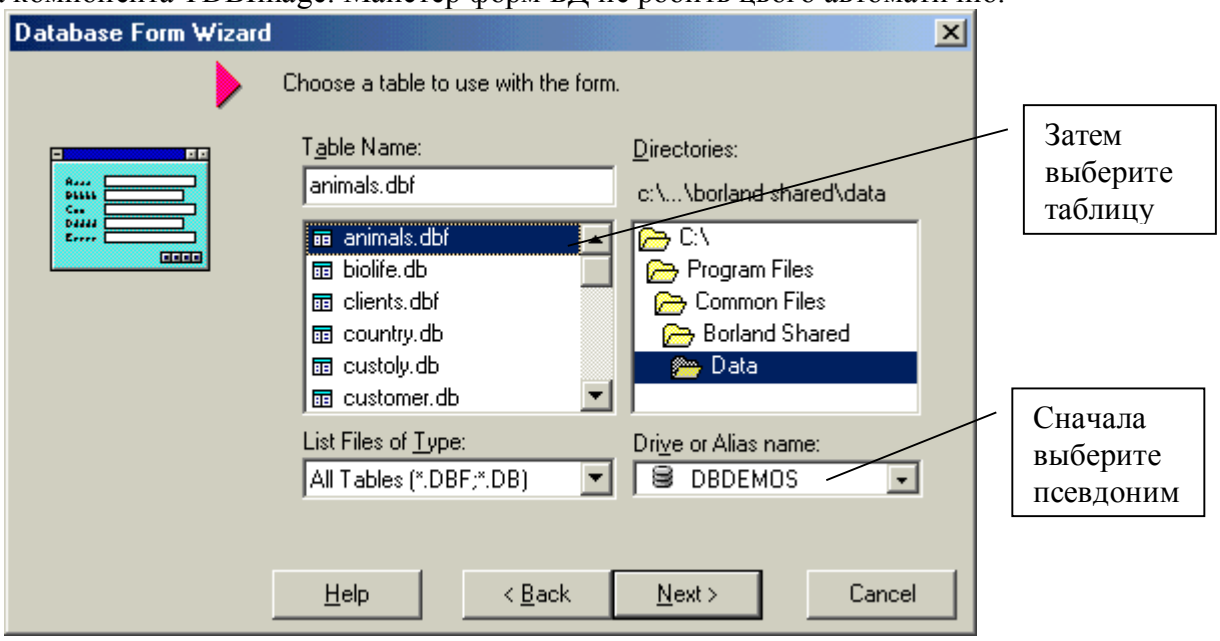
Якби прапорець Create a main form був скинутий, то для відображення створеної форми довелося б вставляти в головний модуль програми додаткові оператори, наприклад метод ShowModal. Якщо ж прапорець установлений, нова форма визначається як головна форма цієї програми, і за допомогою диспетчера проекту можна видалити стару головну форму, що тепер не потрібна.

Натисніть <F9> для компіляції і запуску додатка бази даних. На мал. 2.1.3 показане вікно програми. Без єдиного рядка програмного коду вийшов цілком працездатний додаток бази даних з об'єктом DBNavigator і кнопками для перегляду записів, додавання нових рядків, видалення записів і виконання інших операцій. Цими кнопками можна вільно користуватися, але слід дотримуватися обережності, тому що усі внесені зміни відразу записуються в базу даних. У поле Area запису Angel Fish замість Computer Aquarium укажіть значення Tropical Water. Закрийте, а потім запустите знову програму, щоб переконатися в тім, що відредаговані дані потрапили в базу.



Малюнок 2.1.1 – Початкове діалогове вікно майстра форм БД

Виконуючи описані вище дії, ви, можливо, помітили, що полючи растрових даних не відображаються у виді графіки, а позначені як об'єкти BLOB (Binary Large Object — великий двоичный об'єкт). Це відбувається тому, що компонент TDBGrid не уміє відображати графікові. Надалі дана проблема буде вирішена шляхом вставки в проект ще одного елемента керування — об'єкта компонента TDBImage. Майстер форм БД не робить цього автоматично.



Малюнок 2.1.2 – Вибір псевдоніма бази даних і назви таблиці в майстру форм БД

Хоча запрограмувати DBNavigator легко, багато користувачів спочатку його лякаються. У цьому компоненті є убудовані підказки, які можна активізувати, установивши значення True для властивості ShowHints. Тоді в кнопок DBNavigator з'являються підказки, задані за замовчуванням. Їхній текст можна змінити, відредагувавши властивість Hints.

DBNavigator — це могутній елемент керування, але варто бути обережним, щоб не зробити нічого зайвого (наприклад, не видалити або модифікувати запису, що фіксуються в базі даних, як тільки позначка активного поля переміститься на інший рядок або в інший запис). Однак будь-які зміни можна скасувати, клацнувши на кнопці, позначеної значком X. Запис реєструється в базі даних після щиглика на кнопці *контрольної позначки* (check mark). За допомогою кнопок із зображенням *плюса* і *мінуса* можна додати і видалити запис. Клацнувши на

кнопці з *круговою стрілкою*, можна обновити дані в результаті перезавантаження з таблиці. Інші кнопки служать для переміщення по записах таблиці.

NAME	SIZE	WEIGHT	AREA	BMP
Angel Fish	2	2	Computer Aquariums	(BLOB)
Boa	10	8	South America	(BLOB)
Critters	30	20	Screen Savers	(BLOB)
House Cat	10	5	New Orleans	(BLOB)
Ocelot	40	35	Africa and Asia	(BLOB)
Parrot	5	5	South America	(BLOB)
Tetras	2	2	Fish Bowls	(BLOB)

Малюнок 2.1.3 – Приклад додатка бази даних, створеної майстром форм БД для бази даних із псевдонімом DBDEMOS і таблицею animals.dbf

Щоб додати в додаток додаткові форми баз даних, виберіть команду меню Database⇨Form Wizard. Запуститься той же майстер форм БД, що вибирався на вкладці Business діалогового вікна New Item.

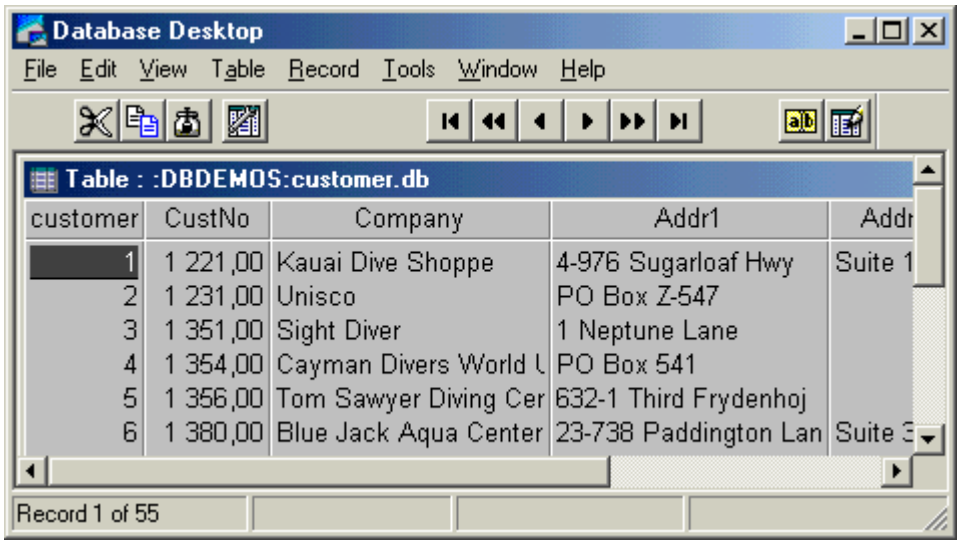
2.1.2.2. Терміни та компоненти баз даних

При роботі з базами даних (при їхньому створенні або розробці додатків відповідного типу) потрібно добре засвоїти зміст трьох взаємозалежних термінів, що використовуються в програмуванні баз даних і в роботі з Borland Database Engine.

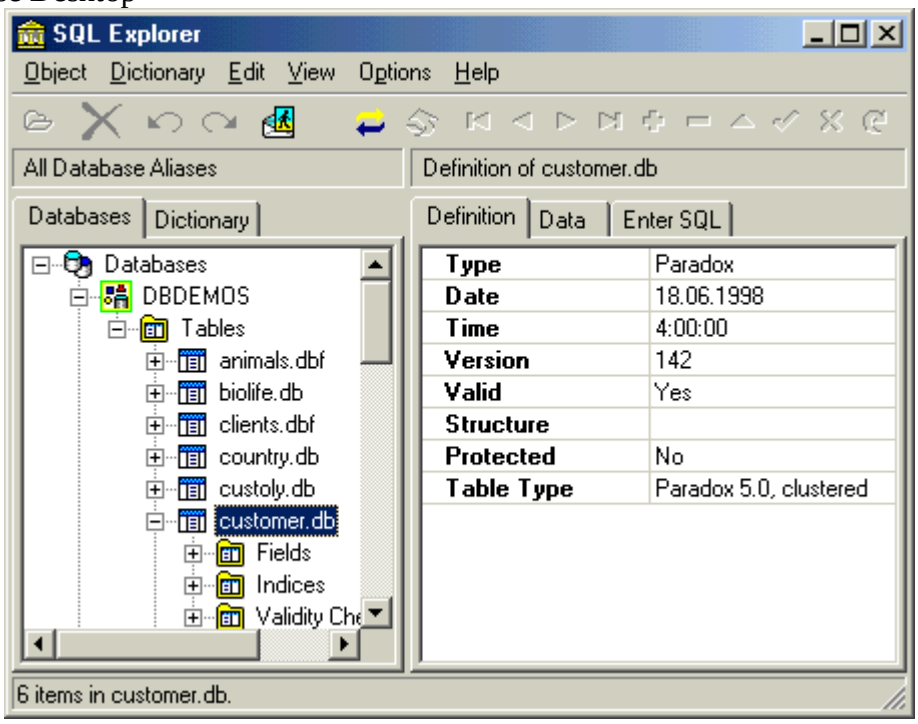
- ***Таблиця*** (Table). Це окремий файл даних, що складається з рядків (записів) і стовпців (полів). Одне з полів таблиці повинне містити первинний ключ, по якому індексується інформація у файлі. Таблиця також може індексуватися по вторинних ключах. Таблицю часто називають *набором даних* (dataset).
- ***Запит*** (Query). Як і таблиця, запит представляє набір даних SQL у форматі зі смугою прокручування і навігатором. Хоча для доступу до даних SQL можна використовувати і таблицю, запит спрощує виконання цієї задачі, особливо якщо дані надходять з SQL-сервера (об'єкт Query також знижує завантаження мережі). Об'єкти запитів можна також використовувати для створення логічних об'єднань різнорідних наборів даних (таблицям це недоступно) і об'єднання даних з різних джерел, наприклад таблиць Paradox і SQL.
- ***База даних*** (Database). Це одна або кілька таблиць (звичайно їх більш двох). Якщо одна таблиця зв'язана з іншої за значенням ключа у визначеному полі, така конструкція відома як *реляционная база даних*.
- ***Псевдонім*** (Alias). Це ім'я, зареєстроване в Borland Database Engine, під яким ховається назва твердого диска і шлях до файлів бази даних. Для звертання до баз даних варто використовувати псевдоніми, а не фізичне ім'я і шлях до файлу. Якщо використовуються псевдоніми, то файли бази даних можна перенести в інший каталог або навіть на інший мережний диск, а всі додатки, що використовують ці файли, будуть працювати без змін.

2.1.2.3. Створення нової бази даних

Додаток Database Desktop використовується для створення нових таблиць баз даних, зміни полів в існуючих базах даних, перегляду інформації про базу даних і створення псевдонімів для наборів даних. Такі редакції Delphi, як Professional і Client-Server, також забезпечені провідниками баз даних, за допомогою яких можна переглядати дані в таблицях і їхній структурі. У цих версіях Delphi провідник знаходиться в меню Database.



Малюнок 2.1.4 – Перегляд таблиці Costumer.db бази даних DBDEMOS за допомогою Database Desktop



Малюнок 2.1.5 - Перегляд таблиці Costumer.db бази даних DBDEMOS за допомогою SQL Explorer

На мал. 2.1.4 показана таблиця Customer.db бази даних DBDEMOS, відображена за допомогою Database Desktop. Засіб Database Desktop, що мається у всіх версіях Delphi, призначено для створення і зміни таблиць баз даних різних форматів. У деяких версіях Delphi є також утиліта аналізу баз даних SQL Explorer. На мал. 2.1.5 показана та ж таблиця, що і на мал. 2.1.4, але відображена за допомогою SQL Explorer. SQL Explorer поставляється тільки з Delphi редакції Client-Server, редакція Professional містить у собі Explorer без можливостей роботи з SQL.

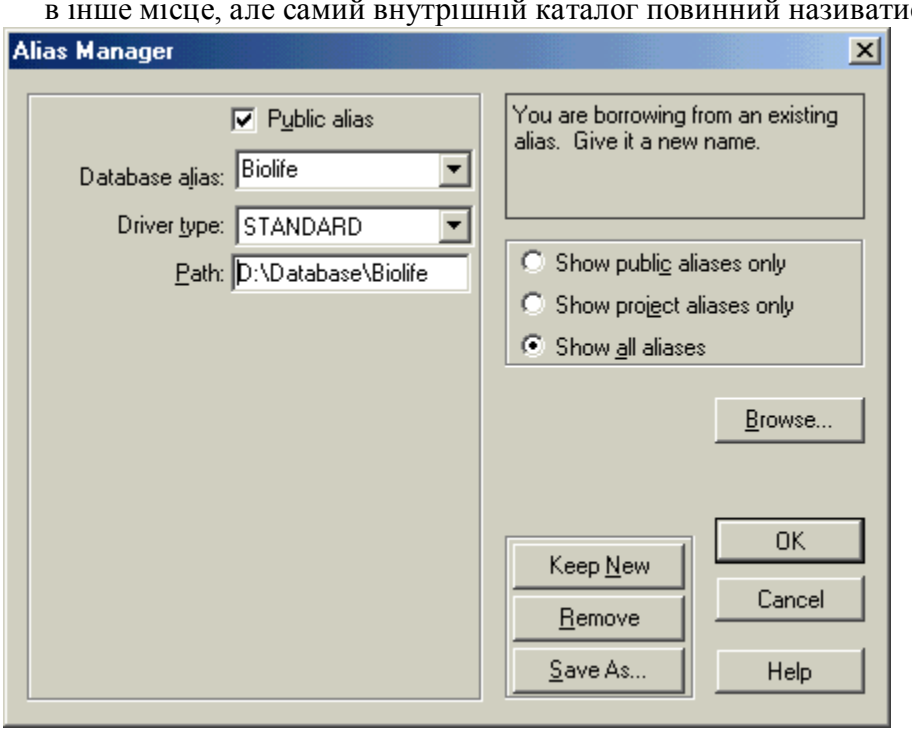
Database Desktop — це автономний додаток, тобто його можна запускати або з меню Tools середовища Delphi, або окремо з каталогу, у якому знаходиться Delphi. Засіб SQL Explorer убудований у Delphi і може викликатися тільки командою меню Database(Explore.

Перед створенням нової бази даних варто визначити місце збереження інформації і привласнити псевдонімові значення, що вказує на це місце. Про створення нової бази розповідається в приведеному нижче розділі “Створення бази даних Biolife”.

2.1.2.4. Робота з базою даних Biolife

Для роботи з існуючою базою даних Biolife виконаєте наступні операції.

1) Скопіюйте з каталогу C:\Program Files\Common Files\Borland Shared\Data три файли, Biolife.px, Biolife.mb і Biolife.db, у каталог D:\Database\Biolife. Можете скопіювати їх і в інше місце, але самий внутрішній каталог повинний називатися Biolife.



Малюнок 2.1.6 – Реєстрація додатком Database Desktop псевдоніма Biolife і шляхи

2) Запустите Database Desktop за допомогою провідника Windows, панелі задач або із середовища Delphi, виконавши команду меню Tools⇨Database Desktop. Коли з'явиться вікно Database Desktop, виконаєте команду Tools⇨Alias Manager.
3) Для створення нового псевдоніма клацніть на кнопці **New**. Уведіть як псевдонім ім'я Biolife і переконаєтеся, що в списку **Driver Type** обраний тип STANDARD (це тип, пропонований за замовчуванням).
4) Клацніть на кнопці **Browse** і відкрийте список **Drive (or Alias)**. Виберіть диск D:\ або ім'я диска, на якому знаходиться каталог Database. Потім двічі клацніть на каталозі Database і наступному за ним Biolife. У поле **Directories** з'явиться напис D:\Database\Biolife. На мал. 2.1.6 показане вікно **Database Desktop** на цій стадії.
5) Клацніть на кнопці **Keep New**, а потім на кнопці **OK**, щоб закрити вікно **Alias Manager**. На питання, чи зберегти псевдоніми **Public Aliases** у файлі IDAPI32.CFG, відповісти **Yes**.
6) Виберіть у вікні **Database Desktop** пункт меню File⇨Open⇨Table…У діалоговому вікні, що **відкрилося, Open** Table у **поле** Alias виберіть псевдонім Biolife. Після цього в діалоговому вікні **Open Table** з'явиться файл Biolife.db. Виберіть його щигликом миші і натисніть кнопку **Відкрити**.

Тепер можна працювати з підключеною базою даних Biolife у вікні Database Desktop.

2.1.2.5. Створення бази даних Sample

Нижче розповідається про створення нової порожньої бази даних Sample. На цьому прикладі можна навчитися використовувати додаток Database Desktop для створення будь-яких таблиць баз даних.

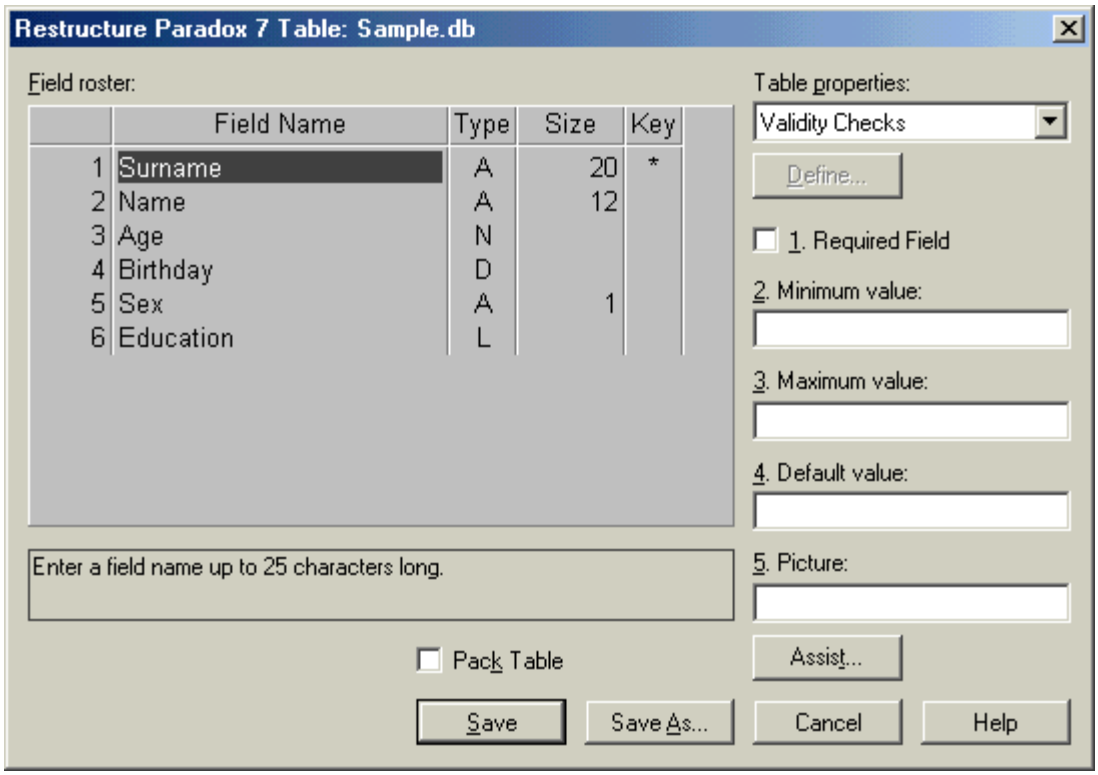
Першою справою, виділите місце для збереження файлів. Імена файлів і каталогів можуть бути будь-якими. За допомогою провідника Windows створіть каталог D:\Database\Sample, причому імена диска і каталогів можете вибрати на свій розсуд. Створивши каталог у результаті

виконання наступних операцій, можна вказати псевдонім для шляху до бази даних і файли таблиць.

1) Запустите Database Desktop за допомогою провідника Windows, панелі задач або із середовища Delphi, виконавши команду меню Tools⇨Database Desktop. Коли з'явиться вікно Database Desktop, виконаєте команду Tools⇨Alias Manager.
2) Для створення нового псевдоніма клацніть на кнопці **New**. Введіть ім'я псевдоніма (це може бути будь-як рядок, але звичайно псевдонімом служить назва файлу, у якому знаходиться база даних). Уведіть як псевдонім ім'я Sample
3) Переконаєтеся, що в списку **Driver Type** обраний тип STANDARD (за замовчуванням). При створенні бази даних класу клієнт/сервер можна вибрати тип INTRBASE. При мінімальній інсталяції Delphi доступний тільки тип STANDARD.
4) Клацніть на кнопці **Browse** і відкрийте список **Drive (or Alias)**. Виберіть диск D:\ або ім'я диска, на якому знаходиться каталог Database. Потім двічі клацніть на каталозі Database і наступному за ним Sample. У поле **Directories** з'явиться напис D:\Database\Sample.
5) Клацніть на кнопці **ОК**, щоб закрити вікно **Directory Browser** (див. мал. 2.1.6). Клацніть на кнопці **ОК**, щоб закрити вікно **Alias Manager**. На питання, чи зберегти псевдоніми Public Aliases у файлі IDAPI32.CFG, відповісти **Yes**. Тепер під псевдонімом Sample додатка Delphi розуміють зазначене ім'я диска і фізичний шлях до цієї бази даних.

У результаті реєстрації псевдоніма Sample і шляхи до файлів додатком Database Desktop власне файли для збереження інформації бази даних не створюються. Щоб створити файл бази даних виконаєте наступне.

1) Запустите Database Desktop, якщо він ще не запущений.
2) Виконавши команду File(New, виберіть пункт підменю Table. Зі списку доступних форматів баз даних виберіть Paradox 7 (за замовчуванням). При створенні нової бази даних формат Paradox — це найкращий варіант вибору, тому що він надає найбільшу кількість типів полів даних і самі широкі можливості індексування. Звичайно ж, можна вибрати будь-як інший формат, наприклад dBASE або Intrbase.
3) Тепер відкривається головний екран для вставки і редагування полів. Введіть імена полів, тип (для вибору типу зі списку натисніть клавішу пробілу) і довжину для текстових полів і позначте зірочкою (або будь-яким іншим символом) поле з первинним ключем, по якому буде індексуватися база даних. У табл. 2.1.1 для приклада показані деякі полючи. На мал. 2.1.7 показана повна структура таблиці бази даних у середовищі Database Desktop.
4) Клацніть на кнопці **Save As**. Уведіть рядок Sample у поле **Filename**, а потім виберіть псевдонім, що буде відноситися до нової бази даних. Виберіть псевдонім SAMPLE, якщо він був визначений раніше, і клацніть на кнопці **Save** для створення таблиці і всіх зв'язаних файлів, зокрема — індексів, що залежать від формату бази даних, обраного в п. 2. Тепер у каталозі D:\Database\ Sample повинні з'явитися файли Sample.db і Sample.px.



Малюнок 2.1.7 – Повна структура таблиці бази даних у середовищі Database Desktop

Таблиця 2.1.1 – Поля бази даних Sample

Поле	***Тип***	***Розмір***	***Ключ***
Surname	Alpha	20	*
Name	Alpha	12	
Age	Number		
Birthday	Date		
Sex	Alpha	1	
Education	Logical		

2.1.3. Компоненти для роботи з БД

Після створення нової або вибору існуючої бази даних, для якої був зареєстрований псевдонім, можна писати на Delphi додаток для вставки, редагування і перегляду інформації в базі даних. Як розповідалося раніше, це можна зробити за допомогою майстра форм БД (вибрати File⇨New, клацнути на корінці вкладки **Business** і двічі клацнути на піктограмі **Database Form Wizard**).

Однак додаток бази даних можна створити й іншим методом, вставивши в порожню екранну форму об'єкти необхідних компонентів. При цьому необов'язково використовувати майстер. У процесі самостійного створення екранної форми для бази даних можна ознайомитися з різними компонентами, включеними в Delphi спеціально з цією метою, і зрозуміти, як вони взаємодіють між собою. Чітке представлення про ці відносини дуже допомагає при розробці різноманітних додатків баз даних.

Для додатків і власне БД доцільно виділяти окремі каталоги, оскільки в процесі розробки Delphi генерує безліч файлів, що не потрібно передавати кінцевим користувачам.

2.1.3.1. Категорія палітрі компонентів Data Access

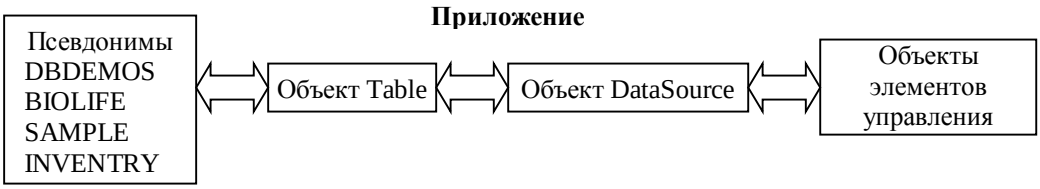
Категорія палітри Data Access містить компоненти, що забезпечують доступ до таблиць БД. Можна навіть сказати, що компоненти з категорії Data Access — це шлюзи до інформації, що утримується в базі даних. Для створення більшості додатків досить використовувати об'єкти двох компонентів — TTable і TDataSource. Інші компоненти категорії Data Access виконують операції

SQL, такі як пошук за значенням параметра (SQL) або глобальні операції відновлення всіх полів у виділених записах (BatchMove). Компоненти TDatabase і TSession використовуються автоматично при організації доступу до таблиць баз даних і потрібні тільки при створенні досить складних програмних продуктів, що забезпечують рівнобіжне функціонування безлічі одночасних з'єднань. Компоненти TStoredProc і TUpdateSQL необхідні для створення систем баз даних класу клієнт/сервер.

У третин версії Delphi для створення звітів уключалися програма ReportSmith і компонент TReport. Починаючи з четвертої версії ці елементи замінені компонентами генерації звітів з великої категорії палітри QReport.

Перший компонент, об'єкт якого звичайно вставляється у форму, — це TTable. Він служить мостом, що з'єднує додаток і псевдонім бази даних, як показано на мал. 2.1.8 До об'єкта компонента TTable потрібно додати об'єкт компонента TDataSource, що приєднує елементи керування роботою з даними до бази даних. Об'єкт DataSource передає дані об'єктам і одержує них з об'єктів і з Table. Об'єкт Table обробляє поточні транзакции в базі даних.

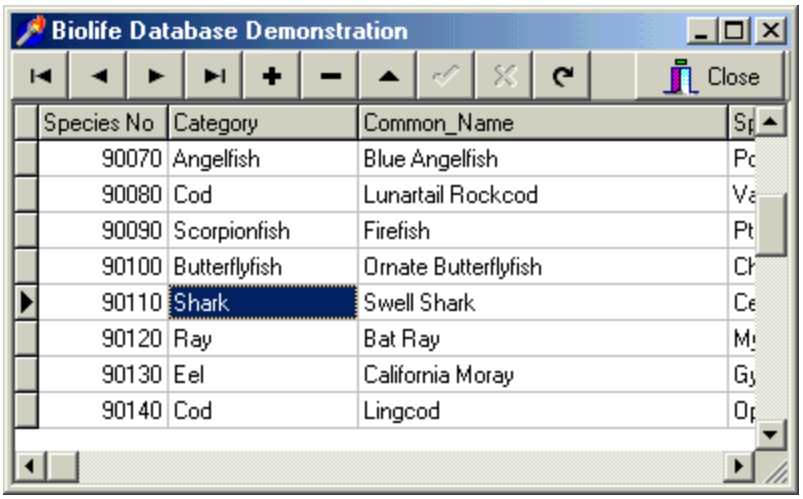
Взаємодія об'єктів Table і DataSource визначається програмою Borland Database Engine (BDE), що бере на себе операції читання і запису даних у потрібному форматі. Елементи керування зв'язком з даними додаються для створення інтерактивних екранних форм уведення даних. Наприклад, приєднавши об'єкт DBNavigator до об'єкта DataSource, що, у свою чергу, підключений до таблиці, представленої об'єктом Table, можна створити прокручиваемую панель для перегляду, редагування, вставки і видалення записів, відображуваних об'єктами DBEdit і іншими елементами керування. Щоб відносини між об'єктами не здавалися такими складними, треба зрозуміти, який з об'єктів приєднується до іншого і яким образом це відбувається. Однак після створення одного або двох нескладних додатків екран уведення даних буде так само простий і зрозумілий, як будиночок, побудований з кубиків.



Малюнок 2.1.8 - Об'єкт Table служить мостом між додатком і базою даних, обумовленої псевдонімом. Об'єкт DataSource з'єднує об'єкт Table з елементами керування зв'язком з даними, такими як DBEdit і DBNavigator

Проробивши наступні операції, можна створити об'єкти компонентів TTable і TDataSource і об'єкти керування роботою з даними для введення і перегляду інформації в БД Biolife (ці ж інструкції застосовні для будь-якої іншої бази даних).

1) Запустите новий додаток, вибравши команду меню File(New Application. Цього разу новий додаток буде створено без допомоги майстра форм БД.
2) Вставте у форму об'єкти компонентів TTable і TDataSource із вкладки палітри Data Access. Вони повинні з'явитися у виді піктограм, що не відображаються в працюючому додатку. У цьому прикладі для всіх об'єктів будуть використовуватися імена Delphi за замовчуванням, хоча при бажанні можна вибрати й інші. Рекомендується в найменуваннях залишати корені Table і Source, наприклад MasterTable і MasterDataSourse. Невізуальні компоненти, наприклад TTable і TDataSource, можна поміщати в модуль даних так само, як у видиме вікно форми. Модулі даних можуть використовуватися і як загальні модулі для доступу до баз даних, причому їх можна використовувати в різних додатках. У розділі “Робота з модулями даних” це питання освітлене докладніше.
3) Виберіть об'єкт Table1 і на вкладці **Properties** вікна **Object Inspector** клацніть на стрільці вниз, розташованої біля властивості DatabaseName. У списку зареєстрованих псевдонімів, що розкрився, і імен баз даних виберіть ІМ'Я BIOLIFE або будь-яке інше.



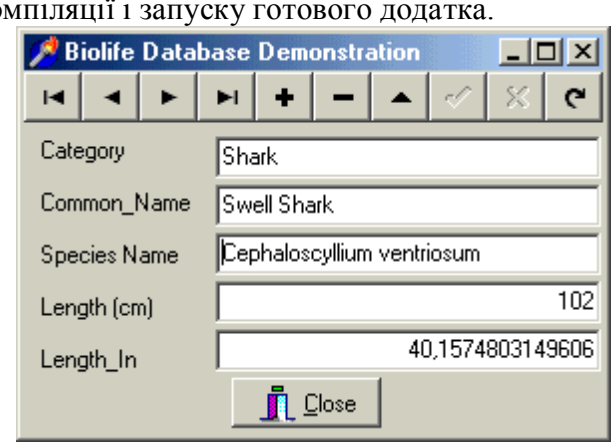
Малюнок 2.1.9 – Готовий додаток BiolifeDemo

4) База даних може складатися з однієї або декількох таблиць, називаних також наборами даних. Хоча в нашому прикладі бази даних дані знаходяться тільки в одній таблиці, крім псевдоніма DatabaseName, потрібно задати ім'я таблиці TableName. Виділите властивість TableName об'єкта Table1 і виберіть файл таблиці Sample.db або який-небудь іншої. Запам'ятаєте: у всіх об'єктів Table повинні бути визначені властивості DatabaseName і TableName.
5) Потім виберіть об'єкт DataSource1. Підключите його до таблиці Table1 за допомогою списку, що розкривається, розташованого напроти властивості DataSet. На цьому етапі, коли виконані мінімальні вимоги для з'єднання додатка з базою даних, в екранну форму можна вставляти зв'язані з даними елементи керування для перегляду і редагування інформації. Запам'ятаєте: всі об'єкти DataSource повинні підключатися до об'єктів Table за допомогою властивості DataSet.
6) DBNavigator— це компонент, що використовується майже завжди. Виберіть його на вкладці палітрі Data Controls і вставте у форму в зручне місце, де-небудь біля верхньої границі. Задайте для властивості DataSource об'єкта DBNavigator значення DataSource1. Це вказує зв'язаному з даними елементові керування, де брати дані для відображення. Запам'ятаєте: усі зв'язані з даними елементи керування повинні бути підключені до об'єкта DataSource через властивість DataSource.
7) Для забезпечення перегляду і редагування інформації в базі даних вставте у форму об'єкт DBGrid, що знаходиться на вкладці Data Controls палітри. У властивості DataSource об'єкта DBGrid укажіть DataSource1. Розміри і розташування елементів керування у вікні можуть бути довільними.
8) Виберіть об'єкт Table1 і двічі клацніть на властивості Active, щоб привласнити йому значення True. Повинна відкритися таблиця бази даних, причому зверніть увагу на те, що програма не запускалася. В об'єкті DBGrid зазначені назви полів і приведена інформація, записана в базі даних. База даних відкрилася, як тільки об'єкт DBGrid став активним елементом керування. Це зручно при розробці додатків баз даних, наприклад, для коректування розмірів відображуваних компонентів. Для закриття таблиці варто установити значення False для властивості Active. У цьому випадку, щоб відкрити таблицю під час запуску додатка, у процедуру обробки події OnCreate екранної форми необхідно включити наступний оператор:
   Table1.Open; *//Відкриває базу даних і таблицю.*

Коли таблиця відкрита, змінити її структуру за допомогою додатка Database Desktop неможливо; тобто, якщо таблиця не відкривається з Database Desktop, варто установити властивість Active об'єкта Table рівним значенню False або можна зберегти і закрити розроблювальний додаток, а потім знову відкрити таблицю в середовищі Database Desktop. На мал. 2.1.9 показане готовий додаток BiolifeDemo.

У більшості випадків для перегляду і редагування інформації в таблиці бази даних використовується об'єкт DBGrid. Але екранні форми введення даних можна будувати, використовуючи й інші зв'язані з даними елементи керування. Наприклад, за допомогою об'єкта DBEdit можна створити екранну форму для введення даних у базу, у якій одночасно відображається тільки один запис.

Для створення програми, показаної на мал. 2.1.10, у порожню екранну форму додайте об'єкти компонентів TTable і TDataSource. Потім виділите таблицю Table і установите значення BIOLIFE для її властивості DatabaseName, а для властивості TableName задайте значення Biolife.db. Властивості DataSet об'єкта DataSourse привласніть значення Table1. Після цього додайте у форму елемент керування DBNavigator, п'ять окремих об'єктів компонента TDBEdit з відповідними написами (Label) і об'єкт компонента TButton (див. мал. 2.1.10). Властивості DataSourse об'єктів DBNavigator і DBEdit привласніть значення DataSourse1. Для властивостей DataField кожного з об'єктів DBEdit установите значення імені відповідного поля таблиці Biolife.db. Потім, після установки властивості Active таблиці Table1 рівним True, натисніть <F9> для компіляції і запуску готового додатка.



Малюнок 2.1.10 - У цій версії додатка, що працює з БД Biolife, замість об'єкта DBGrid використовуються поля DBEdit для перегляду і редагування інформації з одного запису

2.1.3.2. Елементи керування які пов'язані з даними БД

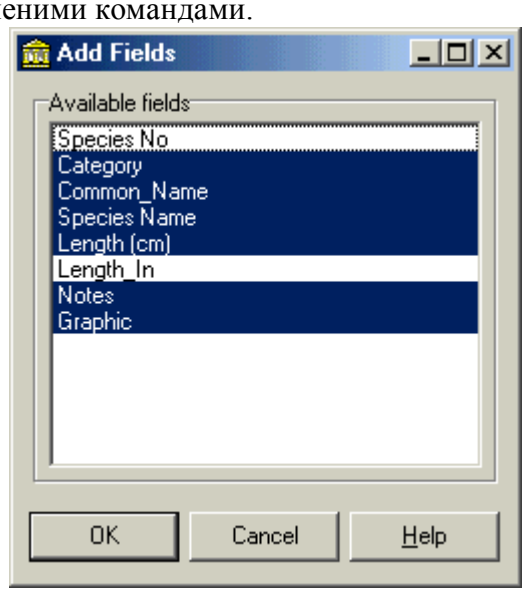
При проектуванні екранних форм уведення даних виникає питання, яким образом будуть відображатися поля. Існує два основних рішення: для висновку даних у рядках і стовпцях таблиці використовується DBGrid, а для відображення окремих полів використовуються спеціальні елементи керування, такі як DBText, DBEdit і DBMemo. За замовчуванням DBGrid відтворює всі доступні поля в тім порядку, у якому вони були введені. А при використанні елементів керування можна відображати будь-які поля у формі в довільному порядку.

Часто, однак, потрібно обмежити кількість виведених об'єктом DBGrid полів або змінити їхній порядок. Можна навіть створити поле, що обчислюється — поле, якому не відповідає яке-небудь значення з файлу бази даних і яке відображає результат обчислень з використанням значень інших осередків. Ці завдання можна виконати, зробивши зміни у віртуальних полях, що компонент Table будує на основі фізичних полів набору даних. У віртуальному полі може відображатися змінена версія фізичного поля або тільки його частина, наприклад тільки прізвище службовця або повна назва штату замість його абревіатури.

Для виконання таких операцій без зміни фізичних полів таблиці потрібно дати вказівки компонентові Table створити безліч об'єктів, що відображають полючи, що накладаються на фізичні стовпці набору даних. Деякі компоненти можуть прямо переноситися з фізичних стовпців у віртуальні. Наприклад, можна вказати компонентові Table використовувати поле Name у тім же виді, у якому воно задається в наборі даних. Інші полючи можуть бути якими завгодно. Ці віртуальні поля можуть підраховуватися на основі значень інших полів або відображати неопрацьовану інформацію з таблиці в спеціально отформатированном виді.

Спробуйте використовувати ці можливості в додатку бази даних Biolife з попереднього розділу або в будь-якому іншому додатку, створеному за допомогою майстра форм БД. Для

редагування полючи таблиці двічі клацніть кнопкою миші на об'єкті Table1 в екранній формі. Як показано на мал. 2.1.11 і 2.1.12, двічі клацніть на об'єкті Table для виклику на екран редактора компонента Field, що з'явиться у виді великого вікна, у даному випадку названого Form1.Table1. Помістивши курсор у це вікно, клацніть правою кнопкою миші, і з'явиться меню з двома дозволеними командами.



Малюнок 2.1.11 - Діалогове вікно Add Fields

- ***Add Fields***. Використовується для додавання в об'єкт Table фізичних стовпців набору даних. Якщо, наприклад, потрібно відобразити значення полючи Category з файлу набору даних, виберіть його за допомогою команди **Add Fields**. Якщо це поле не додавати, воно буде приховано при висновку, наприклад, в об'єкті DBGrid. Кожне додане поле представлене об'єктом у класі форми. На мал. 2.1.11 показане діалогове вікно **Add Fields.**
- ***New Field***. Використовується для створення нового віртуального поля, що може бути одного з трьох типів: Data, Calculated або Lookup. Поля типу Data виходять у результаті звичайної передачі неопрацьованих даних з файлу. Наприклад, можна створити поле даних, що перетворить вихідний текст, заданий прописними буквами, у текст, що складається з малих літер. Поле типу Calculated запускає процедуру обробки події, яку можна використовувати для обчислень і інших перетворень значень полів. Поле типу Calculated може показувати кількість днів між двома датами. Тип третього поля, яке можна створити за допомогою команди **New Field**, називається Lookup. Воно містить інформацію з іншого набору даних, наприклад назва компанії, що відповідає заданому ідентифікаційному номерові. Кожне нове поле представлене об'єктом у класі форми. На мал. 2.1.12 показане діалогове вікно **New Field.**

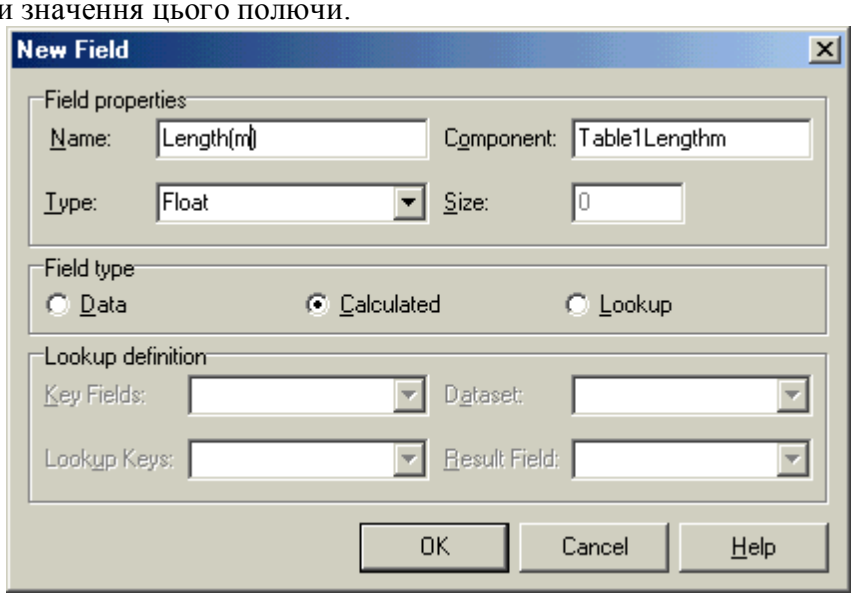
Щоб створити демонстраційний приклад Biolife скористайтеся інструкціями з створенню додатка BiolifeDemo з попереднього розділу “Категорія палітри Data Access”. Двічі клацніть лівою кнопкою миші на об'єкті Table1 і за допомогою команди меню **Add Fields** додайте стовпці Category, Common_Name, Species Name, Length (cm), Graphic і Notes в об'єкт Table1. Поля Species No і Length_In не відображається в сітці, що вийшла, тому що воно не було обрано (див. мал. 2.1.11).

Потім скористайтеся командою **New Field**, щоб створити поле, що обчислюється, Length(m) типу Float. У програмі це поле представлене об'єктом Table1Lengthm типу TFloatField (див. мал. 2.1.11).

Не бійтеся експериментувати з редактором полів, тому що він не може змінити фізичні дані. Для відновлення вилучених полів використовується кнопка **Add**. (Клацніть на кнопці **Clear all** і на питання, чи хочете ви видалити усі компоненти полів, відповісти **Yes**. Потім клацніть на кнопках **Add** і **OK**, щоб повернути поля назад. Як бачите, ці видалення тимчасові.) В екранній

формі кожне поле оголошене як компонент полючи даних типу TStringField, TFloatField, TDateField, TIntegerField і ін. Для перепрограмування оголошень цих об'єктів можна використовувати редактор полів.

Кожне поле з набору даних об'єкта Table представлено об'єктом у класі форми. Для зміни властивостей цих об'єктів використовується вікно Object Inspector точно по тій же методиці, що і для завдання значень для властивостей інших об'єктів компонентів. Якщо в попередньому прикладі вибрати об'єкт Table1SpeciesName у списку, що розкривається, у верхній частині вікна Object Inspector і привласнити його властивості ReadOnly значення True, користувачі не зможуть змінити значення цього полючи.



Малюнок 2.1.12 - Діалогове вікно New Field

Для вибору об'єкта полючи двічі клацніть на компоненті TTable і виберіть об'єкт із переліку імен стовпців. Об'єкт можна вибрати по-іншому, використовуючи список, що розкривається, на панелі Object, Inspector. В обох випадках для зміни властивостей полів об'єкта використовується вікно Object Inspector.

За допомогою редактора полів можна задавати кількість, імена і типи полів даних, що доступні компонентові TTable завдяки об'єктові DataSource. З його допомогою можна також виконувати різні обчислення і вставляти значення в будь-які додані віртуальні поля. При додаванні полючи в редакторі полів не створюється новий стовпець у таблиці даних. Для фізичної зміни інформації в таблиці використовується додаток Database Desktop. У редакторі полів усього лише міняється спосіб представлення цієї інформації на екрані.

Наприклад, якщо в наборі даних два полючи містять дати, може знадобитися вивести кількість днів між ними. Таку інформацію безглуздо зберігати в базі даних, тому що, швидше за все, вона буде щодня обновлятися. Нижче приведена інструкція, що допоможе вам створити додаток, що програмно обчислює значення одного з полів.

1) Уставте заново об'єкт DBGrid у додаток, якщо це необхідно, і установите для його властивості DataSource значення DataSource1. Установите для властивості Active таблиці Table1 значення True, щоб вивести дані з БД у виді сітки осередків. Якщо проект не був збережений або з'явилися інші проблеми, повторите операції, описані в попередньому розділі.
2) Двічі клацніть лівою кнопкою миші на об'єкті Table1, щоб відкрився редактор полів. У вікні редактора клацніть правою кнопкою миші і виберіть команду **Add Fields**. Виберіть поля Category, Common_Name, Species Name, Length (cm), Graphic і Notes (див. мал. 2.1.11). (У цьому прикладі може використовуватися будь-як інша база даних.) Щоб закрити вікно, клацніть на кнопці **ОК**. Якщо якесь з полів було додано помилково, клацніть правою кнопкою миші, щоб повернутися у вікно редактора.

Клацніть на кнопці **Delete**, щоб забрати поле. У такий спосіб віддаляється тільки об'єкт, що представляє це поле в екранній формі. Дані у файлі фактично не міняються.

3) У вікні редактора полів клацніть правою кнопкою миші і виберіть команду **New Field** (див. мал. 2.1.12). Введіть ім'я нового поля, наприклад Length (m), щоб воно відрізнялося від імен інших полів. На мал. 2.1.12 показане ім'я об'єкта компонента (TTable1Lengthm) у другому вікні редагування. Це автоматично сгенерированное ім'я об'єкта, під яким він оголошений у класі форми. Його можна змінити зараз або задати пізніше інше значення властивості Name цього об'єкта у вікні Object Inspector.
4) Для тільки що створеного віртуального поля виберіть тип об'єкта зі списку **Field type**. У нашому випадку об'єктові відповідає тип FloatField. Заданий тип даних потрібний для оголошення об'єкта в класі форми. Кожне поле у формі стає об'єктом компонента.
5) Переконаєтеся, що в групі перемикачів обраний перемикач **Calculated**, і клацніть на кнопці **ОК**, щоб додати нове поле до об'єкта Table1. Тепер редактор полів можна закрити або згорнути. Помітьте, що в об'єкті DBGrid з'явився новий стовпець, названий Length(m). При необхідності перетягнете вертикальну смужку праворуч від стовпця **Category**, щоб зменшити ширину цього полючи і побачити всі інші. Також переконаєтеся, що в секцію оголошення класу форми тепер включений ще один об'єкт — нове віртуальне поле:
   Table1Lengthm: TFloatField;
6) Тепер можна запрограмувати обчислення значення нового поля. У списку вікна, що розкривається, Object Inspector виберіть об'єкт Table1Lengthm. (Цей компонент не можна вибрати у формі, тому що полючи даних не мають у ній візуального представлення.) Виберіть сторінку **Events** вікна **Object Inspector**, а потім двічі клацніть на поле події OnGetText для створення процедури обробки події, поки ще порожній. Вставте в процедуру наступний текст (тут приведений процедура цілком, а в програму потрібно вставити тільки операторы, що знаходяться між ключовими словами begin і end):
   {Підрахунок значення віртуального поля Length(m) -довжина в метрах, з використанням значення полючи Length (cm) таблиці бази даних.}
   ```
   procedure TForm1.Table1LengthmGetText(Sender: TField; var Text: String;
     DisplayText: Boolean);
   begin
     Text:=FloatToStr(Table1Lengthcm.Value / 100);
   end;
   ```
7) Натисніть <F9> для компіляції і запуску програми.

Елементи керування, зв'язані з даними, активізують подія OnGetText для одержання значення полючи. Процедура обробки події в попередньому фрагменті програми ставить у відповідність перемінної Text дані, які потрібно вивести. У цьому прикладі програма перераховує довжину в сантиметрах полючи Length (cm) у метри. Довжина особи в сантиметрах (Length (cm)) зберігається у властивості Value об'єкта полючи Table1Lengthcm. Це один з об'єктів, що були додані в об'єкт компонента TTable за допомогою команди редактори полів Add Field.

Тип властивості Value об'єкта полючи залежить від виду цього об'єкта. У нашому випадку властивість Value об'єкта Table1Lengthcm містить дані речовинного типу з крапкою, що плаває. Довжина особи в метрах підставляється як параметр у функцію FloatToStr мови Delphi. Результат привласнюється перемінної Text, що використовується для висновку підрахованого значення в об'єкті DBGrid. На мал. 2.1.13 показане вікно готової програми з полем, що обчислюється, Length(m), крайнім праворуч у сітці таблиці даних. У додаток були додані об'єкти компонентів TDBImage і TDBMemo для відображення полів Graphic і Notes бази даних Biolife.db тому що об'єкт DBGrid не може відображати поля утримуючі растрові зображення і многострочный текст.

При підрахунку значень полів, що обчислюються, з використанням виключень може виникнути повторюване повідомлення про помилку. Причина такого зациклення в тім, що при

закритті діалогового вікна помилки додаток намагається перерахувати невірно задане поле, що, у свою чергу, знову приводить до помилки. Якщо таке случиться, переключитеся в середовище Delphi, виберіть команду меню Tools⇨Environment Options, на вкладці **Debugger** діалогового вікна, що з'явилося, виберіть **опцію Delphi** Exceptions і клацніть на **кнопці User** Program у **групі Handled** By. (У Delphi 3 і старших версіях цієї опції відповідає прапорець **Break on Exception** на вкладці **Preferences** діалогового вікна, що з'являється при виборі пункту меню Tools⇨Options.) Оскільки програма запускається в отладчике Delphi, вибір цих опцій дійсний тільки для додатків, що виконуються. Таким чином, при виникненні повторюваної помилки програма припиняє роботу і керування передається Delphi. Для виходу з програми можна скористатися командою Run(Program Reset (<Ctrl+F2>). Поки не испробуете всі ці способи установки контролю над ситуацією, не перезавантажуйтеся. Іноді, однак, усе-таки прийдеться перезавантажитися. Усі ці технічні подробиці відносяться тільки до запуску додатка із середовища Delphi.

2.1.3.3. Великий подвійний об'єкт

BLOB — це абревіатура від Binary Large Object (великий двоичный об'єкт). У таблиці бази даних поле BLOB може містити будь-як вид даних, але найчастіше в ньому зберігаються графічні зображення. Наприклад, у базі даних по продаваному майну полючи BLOB можуть містити зображення предметів, призначених для продажу.

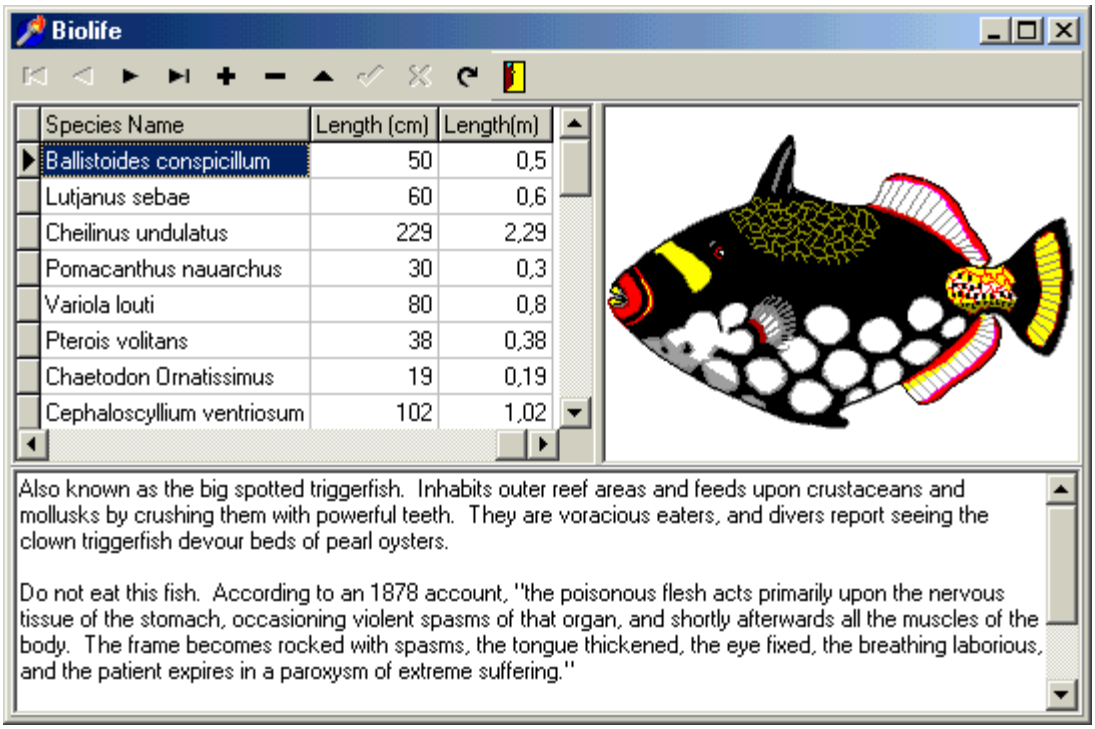
Додаток Biolife з попереднього розділу демонструє використання полів BLOB. На мал. 2.1.13 показане запущений додаток Biolife. У прикладі 2.1.1 утримується вихідний код модуля програми.

Приклад 2.1.1 – Текст додатка Biolife

```
unit Biolife;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DBCtrls, Grids, DBGrids, Buttons, Db, DBTables, ExtCtrls, StdCtrls;
type
  TBioForm = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Table1: TTable;
    DataSource1: TDataSource;
    DBNavigator1: TDBNavigator;
    SpeedButton1: TSpeedButton;
    DBGrid1: TDBGrid;
    Panel4: TPanel;
    DBMemo1: TDBMemo;
    Table1Category: TStringField;
    Table1Common_Name: TStringField;
    Table1SpeciesName: TStringField;
    Table1Lengthcm: TFloatField;
    Table1Graphic: TGraphicField;
    Table1Notes: TMemoField;
    Splitter1: TSplitter;
    DBImage1: TDBImage;
    Table1Lengthm: TFloatField;
    procedure SpeedButton1Click(Sender: TObject);
    procedure Table1LengthmGetText(Sender: TField; var Text: String;
      DisplayText: Boolean);
    procedure DBImage1DblClick(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;
var
  BioForm: TBioForm;
implementation
{$R *.DFM}
procedure TBioForm.SpeedButton1Click(Sender: TObject);
begin
  close;
end;
procedure TBioForm.Table1LengthmGetText(Sender: TField; var Text: String;
  DisplayText: Boolean);
begin
 Text:=FloatToStr(Table1Lengthcm.Value / 100);
end;
procedure TBioForm.DBImage1DblClick(Sender: TObject);
begin
 with DBImage1.Picture do
  ShowMessage('W=' + IntToStr(Width) +
  ' H=' + IntToStr(Height));
end;
end.
```

У більшості програм, як і в додатку Biolife, події обробляються об'єктами компонентів. Для створення такої програми запустите мастер форм БД, відкрийте псевдонім DBDEMOS і виберіть файл Biolife.dbf. Після того як майстер сконструює екранну форму, пройдіться по компонентах і додайте об'єкт DBImage для висновку в растровому полі BLOB, що у таблиці даних Biolife позначено міткою BLOB Graphic. (Щоб власноручно створити версію цієї програми, установите значення DataSource1 для властивості DataSource об'єкта DBImage, а значення Graphic — для властивості DataField . Все інше зрозуміло без додаткових пояснень.)



Малюнок 2.1.13 - Готовий додаток Biolife з полем, що обчислюється, Length(m) демонструє можливості Delphi читати і записувати поля BLOB у базу даних. Тут об'єкт BLOB— це растрове зображення, що знаходиться праворуч у вікні програми

Для ілюстрації програмування інтерактивних подій у базі дані додатки Biolife реалізована обробка події OnDblClick для об'єкта DBImage1. Код цієї процедури, приведений у прикладі, визначає розмір вікна, що вийшло, BLOB (250x150 пикселей). Запустите програму, а потім двічі клацніть на зображенні об'єкта BLOB, щоб побачити його розміри. У більш складних додатках можна обробляти подія завантаження растрового зображення або запускати графічний редактор для створення графічних образів з їхнім наступним включенням у базу даних. Як показано в прикладі, процедура DBImage1.Picture надає доступ до растрових зображень.

2.1.4. Мова структурованих запитів

Мова структурованих запитів (SQL — Structured Query Language) — це стандартизована мова для одержання доступу до даних і виконання операцій з ними, розроблений в Американському національному інституті стандартів (The American National Standards Institute — ANSI) у 1986 році. Оскільки в цій мові немає спеціальних структур, не зовсім правильно буде назвати SQL мовою програмування. Однак у ньому визначені команди, схожі на оператори програми, такі як SELECT, JOIN або UPDATE для виконання різних операцій з таблицями баз даних. Мова SQL архаїчна і з ним важко працювати, але тому що майже всі СУБД підтримують хоча б підмножина цієї мови, його знання може виявитися корисним.

Оскільки Delphi і Borland Database Engine виконують більшість операцій з базами даних, доти, поки користувач не починає розробляти додатка класу клієнт/сервер для вилучених баз даних, йому потрібна тільки одна команда мови SQL — SELECT. У цьому розділі буде докладно розказано, як використовувати цю команду пошуку інформації в таблицях бази даних.

Borland Database Engine підтримує підмножина стандарту SQL для баз даних форматів Paradox, dBase, Oracle і ряду інших. До таблиць БД одного з цих форматів можна застосовувати команди SQL SELECT, INSERT, UPDATE і DELETE для виконання операцій вибірки, вставки, відновлення і видалення відповідно. Також сервери SQL забезпечують додаткові команди, ознайомитися з якими можна в спеціальній документації.

2.1.4.1. Компонент Query

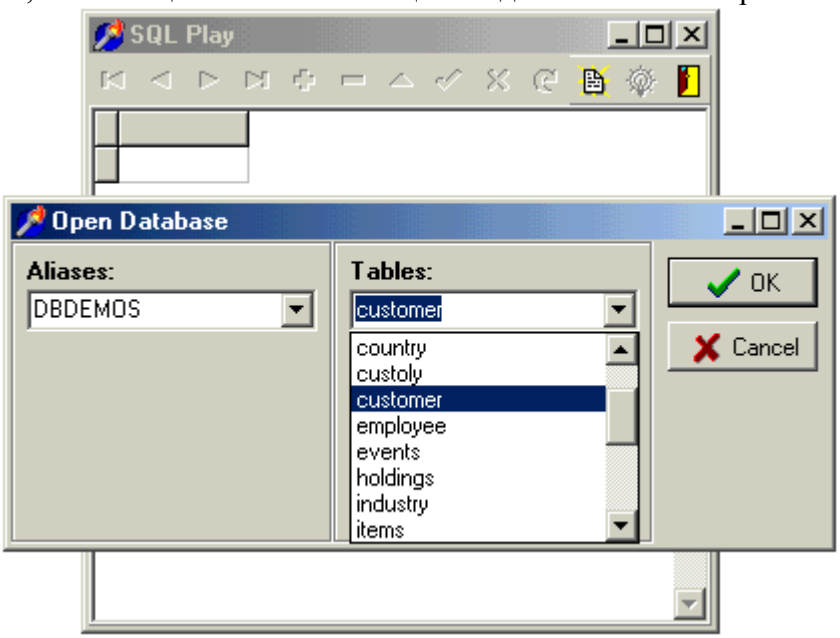
Компонент Query у додатку використовується для передачі базі дані команди SQL, тобто, щоб користуватися командами SQL, кожній базі даних потрібний один об'єкт компонента Query. У додатках Delphi об'єкти Query застосовуються в основному для пошуку по аргументах (наприклад, для вибору всіх людей з карими очима, що народилися у вівторок).

З застосуванням компонента Query відпадає необхідність в об'єкті Table, тому що компонент Query забезпечує шлюз до бази даних по псевдоніму. З компонентом Query необхідно використовувати об'єкт компонента DataSource для приєднання спеціалізованих елементів керування до компонента Query. Елементи керування звичайно відображають результати виконання команд SQL.

2.1.4.1. Редактор побудови SQL-запитів

Додаток SQLPlay являє гарний приклад використання редактори для виконання команд SQL у визначеній базі даних. У цьому додатку показано, як задавати і відповідати на команди SQL, використовуючи компонент Query. Також у програмі демонструється використання Borland Database Engine для одержання всіх псевдонімів баз даних і імен таблиць. Це необхідно при створенні додатка, що надає користувачеві можливість вибрати таблицю бази даних зі списку доступних.

Щоб відкрити діалогове вікно, необхідно клацнути на кнопці Open додатка SQLPlay. У цьому діалоговому вікні вибирається база даних, наприклад DBDEMOS, і таблиця, наприклад Customer, а потім щигликом на кнопці OK здійснюється повернення в головне вікно програми.



Малюнок 2.1.14 - Список доступних псевдонімів баз даних і таблиць у діалоговому вікні Open додатка SQLPlay

Для введення команд SQL призначене вікно Memo під об'єктом DBGrid, у якому відображається інформація з бази даних. Для виконання команд необхідно клацнути на кнопці Perform. Наприклад, можна відкрити базу даних DBDEMOS і таблицю Customer. Потім ввести у вікно Memo команду, що повинна видавати список усіх клієнтів зі штату Каліфорнія:

Select * From CUSTOMER where State="CA"

Для виконання команди SQL в обраній таблиці бази даних потрібно клацнути на кнопці Perform або натисніть <Alt+P>. У прикладі 2.1.2 приведений діалоговий модуль Open.pas додатка SQLPlay. Він містить процедуру перегляду всіх зареєстрованих псевдонімів баз даних і імен таблиць; ця інформація представлена у виді списку, що розкривається, з якого користувач може вибрати базу даних для перегляду і редагування.

Приклад 2.1.2 – Програмний код модуля Open додатка SQL Play

```
unit Open;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls, Db, DBTables;
type
  TOpenForm = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    procedure FormActivate(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  OpenForm: TOpenForm;
implementation
{$R *.DFM}
procedure TOpenForm.FormActivate(Sender: TObject);
begin
  Session.GetAliasNames(ComboBox1.Items);
end;
procedure TOpenForm.ComboBox1Change(Sender: TObject);
begin
  Session.GetTableNames(ComboBox1.Text, '*.*', False, False, ComboBox2.Items);
ComboBox2.ItemIndex := 0;
end;
end.
```

При активізації форми (див. процедуру FormActivate обробки події OnActivate цієї форми) модуль, у якому описане діалогове вікно, викликає процедуру GetAliasNames, що одержує список псевдонімів баз даних від Borland Database Engine. Це не проста процедура, а один з методів класу TSession. Щоб використовувати об'єкти цього класу, потрібно додати імена модулів Db і DBTables у директиву опису модулів uses. Тоді можна буде посилатися на об'єкт Session, що Delphi створює автоматично у всіх додатках баз даних для виклику методів класу TSession. Метод GetAliasNames об'єкта Session вставляє і видаляє імена псевдонімів із властивості TStrings об'єкта TStringList. У приведеній програмі цей метод використовується для ініціалізації властивості Items об'єкта ComboBox1.

Оскільки набір імен таблиць залежить те того, який псевдонім вибере користувач, другий об'єкт ComboBox не може инициализироваться в процедурі FormActivate. Замість цього процедура ComboBox1Change запускається тоді, коли користувач вибирає базу даних, тобто змінює уміст вікна редагування елемента керування. При запуску ця процедура викликає інший метод GetTableNames об'єкта TSession, щоб одержати інформацію для об'єкта ComboBox2. Метод GetTableNames класу TSession містить п'ять параметрів:

procedure GetTableNames(**const** DatabaseName, Pattern: **string**; Extensions, SystemTables: Boolean; List: TStrings);

- ***DatabaseName***. Це ім'я бази даних або псевдонім, наприклад DBDEMOS.
- ***Pattern***. Установите в цьому параметрі фільтр імен файлів, наприклад *.db для вибору таблиць відповідного формату.
- ***Extensions***. Установите значення True для цього параметра, щоб виводити імена файлів з розширенням. Цей параметр є значимим тільки для настільних додатків баз даних.
- ***SystemTables***. Установите значення True для цього параметра, щоб можна було одержати як таблиці з локальної системи, так і набори даних з вилученого сервера. Цей параметр має сенс тільки для додатків класу клієнт/сервер.
- ***List***. Через цей параметр передається значення властивості TStrings або об'єкт класу TStringList. Процедура GetTableNames очищає список і вставляє в нього імена таблиць даних. У прикладі додатка цей параметр використовується для ініціалізації властивості Items об'єкта ComboBox2 типи TStrings. Якщо властивість ItemIndex має значення 0, то в поле редагування комбінованого списку виводиться перший елемент списку.

У прикладі 2.1.3 приведений головний модуль додатка SQLPlay, що виконує команди SQL і відображає обрані таблиці бази даних. Після приклада буде розказано, як взаємодіють об'єкти компонентів для відображення результатів пошуку в об'єкті DBGrid (ці ж методи можуть використовуватися і для інших зв'язаних з даними елементів керування).

Приклад 2.1.3 – Програмний код головного модуля Main додатка SQL Play

```
unit Main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DBCtrls, Grids, DBGrids, Buttons, Db, DBTables, ExtCtrls, StdCtrls;
type
  TMain = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    DBNavigator1: TDBNavigator;
    CloseBitBtn: TSpeedButton;
    DBGrid1: TDBGrid;
    Panel4: TPanel;
    OpenBitBtn: TSpeedButton;
    Query1: TQuery;
    PerformBitBtn: TSpeedButton;
    Memo1: TMemo;
    DataSource1: TDataSource;
    procedure CloseBitBtnClick(Sender: TObject);
    procedure OpenBitBtnClick(Sender: TObject);
    procedure PerformBitBtnClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Main: TMain;
implementation
```

```
uses SQLplay;
{$R *.DFM}
procedure TMain.CloseBitBtnClick(Sender: TObject);
begin
  close;
end;
procedure TMain.OpenBitBtnClick(Sender: TObject);
begin
  if OpenForm.ShowModal = mrOk then
  begin
    Query1.Close;
     try
      Query1.DatabaseName := OpenForm.ComboBox1.Text;
      Query1.SQL.Clear;
      Query1.SQL.Add('Select * From '+OpenForm.ComboBox2.Text);
      Memo1.Lines := Query1.SQL;
      Query1.Open;
      Memo1.SetFocus;
      PerformBitBtn.Enabled := True;
     except;
      ShowMessage('Unable to open database');
     end;
  end;
end;
procedure TMain.PerformBitBtnClick(Sender: TObject);
begin
   Query1.Close;
  try
    Query1.SQL := Memo1.Lines;
    Query1.Open;
  except
    ShowMessage('Invalid query');
  end;
end;
procedure TMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Query1.Close;
end;
end.
```

У прикладі головної програми процедура OpenBitBtnClick відкриває екранну форму OpenForm, викликаючи її метод ShowModal. Якщо користувач клацає на кнопці **ОК**, програма закриває об'єкт Query1 і готується відкрити нову базу даних. Об'єкти типу Query функціонують так само, як і об'єкти Table і, крім того, можуть виконувати оператори SQL. Для цього в програму потрібно ввести кілька рядків. Спочатку програма одержує псевдонім бази даних DatabaseName з вікна редагування компонента ComboBox1. Потім потрібно два оператори для очищення поточного списку рядків SQL об'єкта Query і виклику методу Add, щоб вставити в цей список такий оператор:

Select * From Animals

Команда Select мови SQL шукає в наборі даних, у нашому випадку — Animals, позначені записи. Зірочка * позначає всі стовпці (або полючи) таблиці Animals. Оскільки в команду не включений оператор WHERE, що задає умову вибору рядків, у результаті виводяться всі рядки

(або запису) таблиці Animals. Для того щоб можна було редагувати команди SQL, властивість SQL об'єкта Query1 зв'язано з об'єктом Memo. Потім програма викликає метод Open об'єкта Query1, що передає SQL-запит наборові даних.

Виклик методу Open об'єкта Query1 виконує тільки команду Select мови SQL Для виконання інших команд мови SQL, таких як Insert або Delete, потрібен виклик методу ExecSQL.

Процедура PerformBitBtnClick виконує команди мови SQL, що вводяться у вікні Memo. Спочатку програма закриває об'єкт Query1. (Об'єкти Query і Table завжди варто закривати перед внесенням істотних змін у їхні властивості.) Привласніть значення об'єкта TStrings або TStringList властивості SQL, а потім відкрийте базу даних для виконання команди, у нашому випадку —Select.

Щоб поэкспериментировать з додатком SQLPlay, можна запустити його і відкрити базу даних DBDEMOS. Вибравши таблицю набору даних HOLDINGS, клацнемо на кнопці OK для повернення в головне вікно програми, а потім клацнемо у вікні редагування Memo. Уводимо приведену нижче команду SQL і натискаємо комбінацію клавіш <Alt+P> для виконання команди, що вибирає з набору дані записи про усіх вкладників, розмір внеску яких більше 10 000:

Select * From HOLDINGS where shares >10000

Умова пошуку можна ускладнити. Додаємо ще одне обмеження під двома попередніми лініями для вибору вкладників, що купила акції за ціною понад $50, розмір внеску яких більше 10 000,.

and pur_Price > 50

Після відкриття бази даних за допомогою кнопки Open додатка SQLPlay у вікні можна вивести будь-як таблицю з тієї ж бази даних. Наприклад, забравши всі командні рядки з вікна редагування Memo, можна ввести замість них команду Select * from Animals і натиснути <Alt+P> для висновку всіх записів таблиці Animals.

2.1.5. Бази даних моделі "головний/підлеглий"

Більшість баз даних є *реляционными*, що позначає усього лише, що вони складаються з двох або більш таблиць, зв'язаних між собою загальними полями. Наприклад, у таблиці Customer є поле ID, що також використовується в таблиці Purchases для позначення клієнта, що зробив покупку. Такі установки називаються *відношенням "один-ко-многим"*. В основній таблиці (Customer) у кожного запису є унікальний ключ (поле ID). У підлеглій таблиці (Purchases) кожному клієнтові відповідає одна або більш зв'язаних записів. Такі відносини називаються *моделлю "головний/підлеглий"* (master-detail model).

Відносини "головний/підлеглий" у додатках баз даних Delphi визначають властивості MasterSource і MasterFields об'єкта компонента Table.

- ***MasterSource***. Для підлеглого об'єкта Table містить значення імені об'єкта компонента DataSource, підключеного до головного об'єкта Table. У головному об'єкті Table ця властивість повинна бути порожньо.
- ***MasterFields***. У підлеглому об'єкті Table містить рядок з ім'ям одного або декількох полів (або стовпців), що з'єднують дві таблиці або приєднують одну таблицю до іншої.

Рядок MasterFields може містити кілька імен полів, поділюваних крапками з коми. От кілька прикладів присвоєнь, що показують можливі формати:

DetailTable.MasterFields := 'Symbol';

DetailTable.MasterFields := 'Symbol/Exchange';

У головній таблиці не потрібно визначати властивість MasterFields. Привласнювати рядок цій властивості випливає тільки в підлеглому об'єкті Table. У підлеглій таблиці також необхідно визначити властивість IndexFieldNames або IndexName. По можливості краще користуватися властивістю IndexName, що прискорює пошук. Властивість IndexFieldNames визначається, поле, що коли приєднується, не проіндексовано.

2.1.6. Робота з модулями даних

Доступ декількох додатків до тим самим таблиць бази даних — явище достатнє незвичайне. Наприклад, компанії з великою базою даних може знадобитися створити десятки додатків для пошуку зведень, формування звітів, сортування інформації в базі даних і керування нею. У такому випадку буде корисно створити модуль даних, у якому визначені правила доступу до таблиць бази даних і відповідні інструкції. Установивши модуль даних (Data Module) за допомогою списку об'єктів Delphi, програміст може легко створювати нові додатки баз даних.

Звичайно модуль даних містить компоненти TTable, TQuery, TDataSource і інші, у яких визначені псевдонім бази даних, імена таблиць та інші властивості. Модуль даних може приєднуватися до будь-яких файлів проектів, що використовують таблиці баз даних. При такій організації невізуальні компоненти баз даних утримуються в окремому, невидимому вікні, не захаращуючи екранну форму під час конструювання. Але найбільша перевага використання модулів даних полягає в підтримці доступу декількох додатку до бази даних. Часто буває, що правила й інструкції потрібно змінити. Тоді необхідно тільки модернізувати компоненти модуля даних і перекомпілювати додатка, а не перепрограмувати кожен об'єкт Table і DataSource у кожній програмі.

Щоб створити модуль даних для БД Biolife, про яку йшла мова вище в цій главі, і додати його в список об'єктів Delphi, необхідно виконати наступні операції.

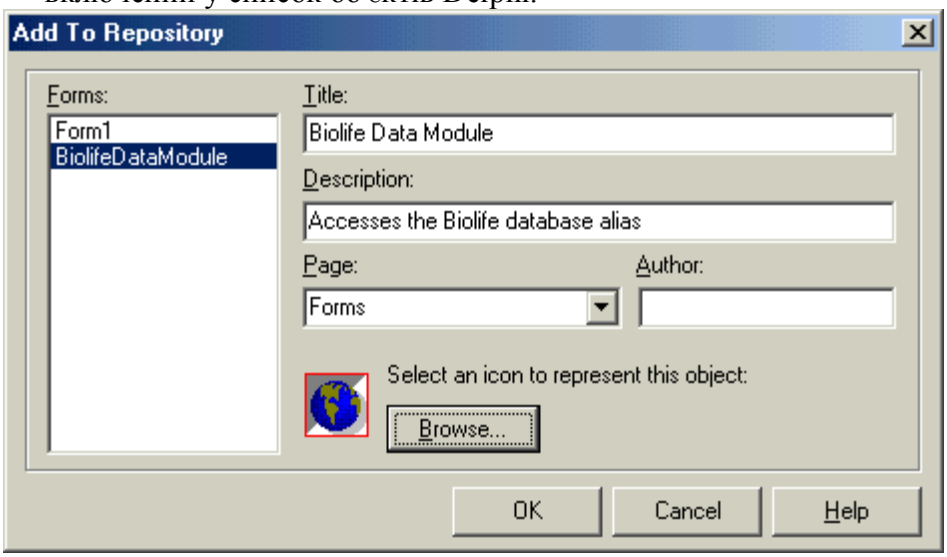
1) Створіть окремий каталог для збереження файлів. Відкрийте новий додаток і виберіть команду File(New Data Module.
2) За допомогою вікна Object Inspector зміните ім'я модуля даних на BiolifeDataModule. На цьому етапі краще зберігати усі файли в обраному каталозі і перейменувати файл, зв'язаний з модулем даних (у цьому прикладі — Unit2.pas). Наприклад, назвіть його BiolifeDM.pas.
3) Знову викличте вікно форми BiolifeDataModule (скористайтеся командою View(Window List, якщо ви його втратили). Додайте в це вікно об'єкти компонентів TDataSource і TTable з палітри Data Access. Зміните значення властивості Name об'єкта DataSource1 на BiolifeDataSource, а властивості Name об'єкта Table1 — на BiolifeTable. Вікно форми порожньо і використовуватися не буде (хоча можна створити тестову програму для цього модуля даних).
4) Тепер варто визначити правила доступу до бази даних, використовувані в об'єктах компонентів модуля даних. Виберіть об'єкт BiolifeTable, у вікні Object Inspector клацніть на кнопці зі стрілкою вниз у рядку властивості DatabaseName і виберіть у списку пропонованих значень BIOLIFE. Також виберіть Biolife.DB у списку пропонованих значень для властивості TableName.
5) Виберіть у модулі даних об'єкт BiolifeDataSource. Потім у вікні Object Inspector привласніть значення BiolifeTable властивості DataSet.
6) Збережете усі файли, викликавши команду меню File⇨Save All або клацнувши на піктограмі **Save All** панелі інструментів.

На цьому закінчується програмування модуля даних. При створенні додатків можна додавати додаткові компоненти і встановлювати інші властивості для доступу до таблиць бази даних. Хоча в прикладі використовується тільки два компоненти, у більш складному модулі даних може бути безліч об'єктів компонентів, таких як Table, DataSource і інших, для доступу до різних таблиць баз даних.

Не утруждайте себе установкою значення для True властивості Active. Значення цієї властивості автоматично міняється на False при збереженні модуля даних у списку об'єктів Delphi.

Щоб зробити цей модуль даних доступним для інших додатків, додайте його в список об'єктів Delphi.

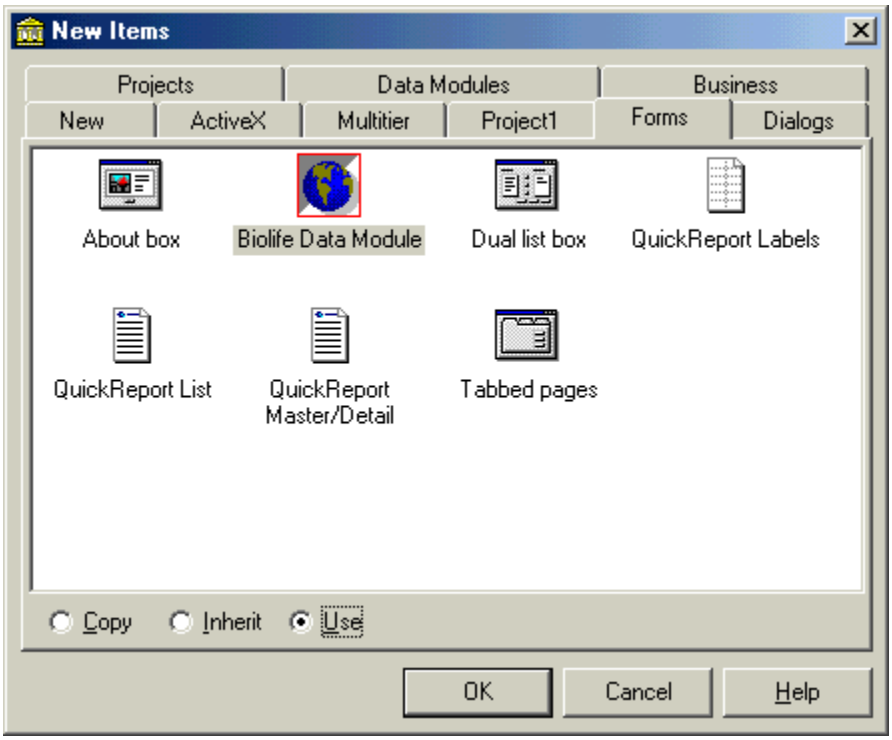
1) Викличте екранну форму модуля BiolifeDataModule (у разі потреби скористайтеся командою меню View⇨Window List). Установите курсор усередині цього вікна і клацніть правою кнопкою миші. З'явиться контекстне меню. Виберіть команду **Add to Repository**. З'явиться діалогове вікно, зображене на мал. 2.1.15.
2) У списку екранних форм **Forms** виберіть WinesDataModule (цей рядок у списку вибирається Delphi за замовчуванням). Уведіть назву (поле Title), опис (поле Description) і назва вкладки (поле Page), у которую буде включений цей модуль даних при виклику пункту меню File⇨New. Також можна заповнити поле **Author** і клацнути на кнопці **Browse** для вибору піктограми, що буде відображатися в діалоговому вікні **New Items**. Піктограми, що поставляються в складі програмного продукту Delphi, знаходяться в каталозі Images\Icons. На мал. 2.1.15 показаний приклад заповнення цих полів.
3) Для закриття діалогового вікна **Add to Repository** клацніть на кнопці **ОК**. Якщо модуль не був збережений, з'явиться запрошення зберегти його перед тим, як модуль буде включений у список об'єктів Delphi.



Малюнок 2.1.15 – Діалогове вікно Add to Repository

Тепер модуль даних можна використовувати в будь-якому додатку. Для цього потрібно просто вибрати команду меню File⇨NewМожно виконати наступні дії з модулем даних Biolife.

1) Відкрийте новий додаток.
2) Виберіть команду меню File⇨New і клацніть на корінці вкладки **Forms** діалогового вікна **New Items**. У вікні вкладки серед інших з'явиться модуль даних Biolife.
3) Клацніть на кнопці **Use** унизу діалогового вікна (див. мал. 2.1.16). Зверніть увагу, що на цьому кроці існує три можливих способи включення модуля: копіювання (кнопка **Сміттю**), спадкування (кнопка **Inherit**) і використання (кнопка **Use**).



Малюнок 2.1.16 - У діалоговому вікні New Items виберіть новий модуль даних Biolife Data Module

4) Двічі клацніть на піктограмі модуля даних Biolife (Biolife Data Module). У поточному додатку відкриється вікно модуля даних, а у вікно редактора коду додасться вихідний код модуля мовою Object Pascal.
5) Переключитеся у вікно редагування коду і виберіть модуль Unitl (вихідний код головної програмної форми). Розшукайте директиву опису модулів uses у верхній частині вікна і зробіть у ній наступні зміни (виділена жирним шрифтом). Після таких змін модуль даних BiolifeDM (під цим ім'ям був збережений вихідний код модуля) буде доступний з модуля головної форми. Для того щоб Delphi внесла ці зміни автоматично, можна скористатися командою меню File⇨Use Unit, але зміни будуть внесені в секцію реалізації головного модуля, а не в секцію інтерфейсу, як у першому випадку:
   uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, **BiolifeDM**;
6) Додайте у вікно головної екранної форми об'єкти компонентів TDBNavigator і TDBGrid із вкладки палітри Data Controls. Виділите обидва об'єкти (утримуючи клавішу <Shift>, клацніть лівою кнопкою миші на кожнім об'єкті. У вікні Object Inspector задайте для властивості DataSource наступне значення (у нашому прикладі це єдиний пропонований варіант):
   BiolifeDataModule. BiolifeDataSource
7) Переключитеся назад у вікно модуля даних Biolife (якщо його не видно на екрані, скористайтеся командою меню View(Window List). Виберіть об'єкт Table і у вікні Object Inspector установите значення True для його властивості Active. Відкриється база даних Biolife і в полях об'єктів DBNavigator і DBGrid будуть виведені дані.

Тільки що розглянутий приклад додатка, що використовує модуль даних BiolifeDM, демонструє спільне використання об'єктів доступу до баз даних декількома додатками. У головну форму програми не потрібно вставляти об'єкти Table і DataSource, тому що вони уже включені в модуль даних. Інші додатки можуть використовувати ті ж об'єкти для доступу до бази даних Biolife. Щоб змінити ці об'єкти (приміром, поміняти псевдонім бази даних, ім'я таблиці або додати віртуальне поле за допомогою редактора полів, як розповідалося вище в цій главі), потрібно просто перекомпілювати додаток після внесення змін.

Об'єкти зі списку об'єктів Delphi можуть додаватися в проект трьома способами: шляхом копіювання (кнопка **Сміттю**), спадкування (кнопка **Inherit**) і використання (кнопка **Use**). Не для кожного типу об'єкта доступні всі ці опції, наприклад проект може тільки копіюватися. При копіюванні об'єкта зі списку об'єктів Delphi у каталозі проекту створюється повна копія файлів об'єкта. Усі зміни об'єкта дійсні тільки для цієї копії. При спадкуванні об'єкта Delphi будує новий клас на основі цього об'єкта. У даний клас можна вносити зміни, але при перекомпіляції успадковуються всі зміни, внесені у вихідний об'єкт у списку об'єктів Delphi. При використанні об'єкта Delphi просто посилається на його файли, де б вони ні знаходилися. Усі зміни, внесені в об'єкт, відбивають у додатку після компіляції. Звичайно модулі даних краще успадковувати або використовувати.

2.1.7. Корисні поради

- Завжди реєструйте псевдоніми баз даних у Database Desktop і не використовуйте в додатках фізичні імена дисків і шляхи до файлів. Використання в додатку фізичних імен і шляхи до бази даних не забороняється Delphi, але потрібно врахувати, що такий додаток буде непрацездатно, якщо БД перемістити в інший каталог, на інший твердий або мережний диск. Якщо ж для доступу до таблиць бази даних використовуються псевдоніми, шлях до них не втратиться, коли них, наприклад, перенесуть на інший диск.
- Майстер форм БД створює вторинну екранну форму для нового додатка. Для використання цієї форми як головне вікно програми виберіть пункт меню Projects⇨Options…і в діалоговому вікні, що **з'явилося, Projects** Options виберіть потрібну форму в якості головної в **поле Main** form на **сторінці** Forms. Тут же можна видалити іншу порожню екранну форму проекту.
- Драйвер Windows ODBC driver for Access розроблений для використання з об'єктами пакета Microsoft Office. Щоб створити в середовищі Delphi додаток для роботи з базами даних формату Access, потрібно установити повну версію додатка ODBC Desktop Driver. Цей драйвер поставляється компанією Microsoft у складі пакета ODBC SDK, у який включені драйвери для баз даних форматів Access, Btrieve, dBASE, FoxPro, Paradox, Excel і Text.
- Для поширення додатка бази даних користувачам потрібно передати exe-файл програми і додаток Borland Database Engine. Інформація про файли, які необхідно надавати користувачам кінцевого продукту, приведена у файлі Redist.txt, що поставляється в складі Delphi.

2.1.8. Резюме

Компоненти баз даних Delphi поділяються на двох категорій: компоненти категорії Data Access забезпечують шлюзи і зв'язки між додатком і базою даних; компоненти категорії Data Controls містять елементи керування, орієнтовані на роботу з даними, і інші об'єкти для перегляду і редагування полів даних, а також для переміщення по записах бази даних.

У редакції *Desktop, Professional* і *Client-Server* програмного продукту Delphi включений додаток Borland Database Engine, що забезпечує повний набір інтерфейсів програмних функцій для більшості популярних форматів баз даних, таких як dBASE, Paradox і системи ODBC, тобто Microsoft Access і BTrieve. Редакція *Client-Server* програмного продукту Delphi включає усі функціональні можливості редакції *Desktop* і, крім того, забезпечує доступ до вилучених серверів баз даних, таким як Oracle, Sybase, Microsoft SQL Server, Interbase і Informix.

Новий компонент TDataSet більше не залежить від Borland Database Engine. Досвідчені програмісти тепер можуть розробляти класи, похідні від класу TDataSet, щоб можна було викликати функції зовнішнього сервера бази даних без установки і використання Borland Database Engine. Для більшості додатків це не є необхідним.

На початковому етапі створення нового додатка бази даних скористайтеся майстром форм БД (команда **Database Form Wizard** меню **Database** середовища Delphi (у колишніх версіях — **Database Form Expert** з меню **Help**). Майстер форм БД створює екранну форму з компонентами категорій Data Access і Data Controls, що згодом можна модифікувати.

Таблиця — це набір даних (щоскладається з рядків і стовпців набір осередків). Рядка відповідають записам, а стовпці — полям. База даних може складатися з однієї або декількох таблиць. Псевдонім бази даних — це ім'я, під яким ховається інформація про твердий або мережний диск, а також шляхи й іменах файлів, у яких знаходиться база даних. Завжди використовуйте псевдоніми для посилання на базу даних. Додаток не прийдеться перекомпілювати, якщо файли бази даних будуть перенесені в інший каталог або на мережний диск.

Більшість додатків баз даних використовує об'єкти компонентів TTable і TDataSource. Об'єкт Table забезпечує зв'язок з таблицею бази даних (найкраще, якщо база представлена псевдонімом). Об'єкт DataSource приєднує до таблиці Table один або більш об'єктів з категорії палітри Data Controls.

Для створення і зміни таблиць баз даних використовуйте Database Desktop. За допомогою цієї програми базу даних можна також переглядати і редагувати, не звертаючи до Borland Database Engine.

Borland Database Engine підтримує стандартна підмножина команд мови SQL (Select, Insert, Update) і Delete для баз даних форматів Paradox, dBASE, Oracle і ін. Можливість використання повної версії мови SQL залежить від характеристик підтримуючу мову сервера.

Команди SQL виконуються компонентом Query. Оскільки різні версії мови SQL працюють, з компонентами баз даних Delphi по-різному, із усієї мови краще використовувати тільки команду Select для пошуку і висновку набору записів, що задовольняють зазначеним аргументам.

Додаток бази дані моделі “головний/підлеглий” поєднує двох таблиць даних, у яких є загальне поле, наприклад номер клієнта або торговельний знак акції. У моделі “головний/підлеглий” потрібно дві пари об'єктів Table і DataSource.

Помістите об'єкти Table, DataSource і об'єкти інших компонентів категорії Data Access у модуль даних і потім уключите його в список об'єктів Delphi. Після цього створений модуль може використовуватися або успадковуватися декількома додатками для одержання доступу до таблиць бази даних.

У наступній главі продовжена розповідь про програмування баз даних. У ній описується побудова звітів і діаграм на основі інформації з бази даних.

Література [1 - 3].

Тема 2.2. Розробка діаграм та звітів

2.2.1. Компоненти

Не так уже складно організувати збереження даних на комп'ютері, але потрібне дійсна майстерність, щоб витягти інформацію з бази даних і представити неї в доступній формі. Таке представлення даних робить інформацію коштовної і корисної. Програмісти баз даних і менеджери системи, імовірно, більше часу витрачають на розробку додатків для перегляду і печатки діаграм і звітів, чим на будь-які інші види програм.

У Delphi такі задачі вирішуються легко. У бібліотеці Delphi TeeChart утримується безліч типів діаграм для наочного відображення даних. А в бібліотеці Delphi QuickReport маються редактор із широкими функціональними можливостями і бібліотека компонентів для розробки усіх видів звітів, що друкуються. Delphi поставляється з повною стандартною версією кожної бібліотеки з готовими до використання компонентами, установленими на палітрі VCL.

У цій главі розглядаються основні і допоміжні прийоми програмування для бібліотек компонентів TeeChart і QuickReport і створення професійних звітів на основі інформації, що зберігається в таблицях баз даних.

2.2.1 Компоненти

У даній главі розглядаються наступні компоненти Delphi.

- ***TChart***. Цей компонент використовується для створення діаграм за даними, що зберігається у файлах або введеним у програму безпосередньо.
  Категорія палітри: Additional.
- ***TDBChart***. Цей компонент використовується для створення діаграм за даними, що зберігається в таблицях бази даних.
  Категорія палітри: Data Controls.
- ***TDecisionGraph***. За допомогою цього компонента створюються графи даних комбінованої таблиці бази даних.
  Категорія палітри: Decision Cube.
- ***TQRBand***. Звіти генеруються зі смуг, кожна з яких є об'єктом цього компонента.
  Категорія палітри: QReport.
- ***TQRChart***. Використовуйте цей компонент для вставки діаграм у звіти бази даних.
  Категорія палітри: QReport.
- ***TQRChildBand***. Цей компонент служить для розширення звичайних смуг звіту. Він використовується для створення смуг, що розширюються за допомогою інших компонентів, що переміщаються або також розширюються, і для смуг, що займають кілька сторінок.
  Категорія палітри: QReport.
- ***TQRDBImage***. Використовуйте цей компонент для вставки в звіт об'єкта Image, звичайно зв'язаного з полем Blob у таблиці бази даних.
  Категорія палітри: QReport.
- ***TQRDBRichText***. Використовуйте цей компонент для вставки текстового об'єкта таблиці бази даних (формат RTF) у звіт.
  Категорія палітри: QReport.
- ***TQRDBText***. Компонент призначений для печатки більшості типів полів таблиці бази даних у текстовій формі. Це вірно не тільки для текстових полів, але і для числових полів, полів дат, полів грошових величин і ін.
  Категорія палітри: QReport.
- ***TQRExpr***. Компонент служить для створення виражень, наприклад обчислення значення елемента звіту, за допомогою інших полів таблиці бази даних.
  Категорія палітри: QReport.
- ***TQRImage***. Компонент використовується для вставки в звіт растрових або інших типів зображень, таких як піктограми. Цей компонент підтримує ті ж типи зображень, що і компонент TImage.
  Категорія палітри: QReport.
- ***TQRLabel***. Компонент використовується для вставки в звіт статичної мітки.
  Категорія палітри: QReport.

- ***TQRMemo***. Компонент використовується для вставки в звіт многострочных текстових об'єктів.
  Категорія палітри: QReport.
- ***TQRPreview***. Застосовуйте цей компонент, щоб забезпечити попередній перегляд звіту. Однак частіше замість використання цього компонента легше викликати метод Preview компонента TQuickRep.
  Категорія палітри: QReport.
- ***TQRRichText***. Компонент служить для вставки в звіт текстового об'єкта (формат RTF).
  Категорія палітри: QReport.
- ***TQRShape***. Компонент служить для вставки в звіт графічних фігур, таких як прямокутник або коло.
  Категорія палітри: QReport.
- ***TQRSubDetail***. Використовуйте цей компонент, щоб зв'язати кілька наборів даних в одному звіті. Звичайно це робиться для печатки звітів баз даних, що використовують модель “головний/підлеглий”.
  Категорія палітри: QReport.
- ***TQRSysData***. Компонент використовується для вставки в звіт (звичайно — у заголовок) системних об'єктів, таких як поточна дата і час, номерів сторінок і інших елементів.
  Категорія палітри: QReport.
- ***TQuickRep***. Компонент використовується для створення нового звіту. За допомогою властивості компонентів TBands уставляються різні типи смуг, у яких утримуються поля бази даних і інші типи об'єктів для печатки в підсумковому звіті.
  Категорія палітри: QReport.

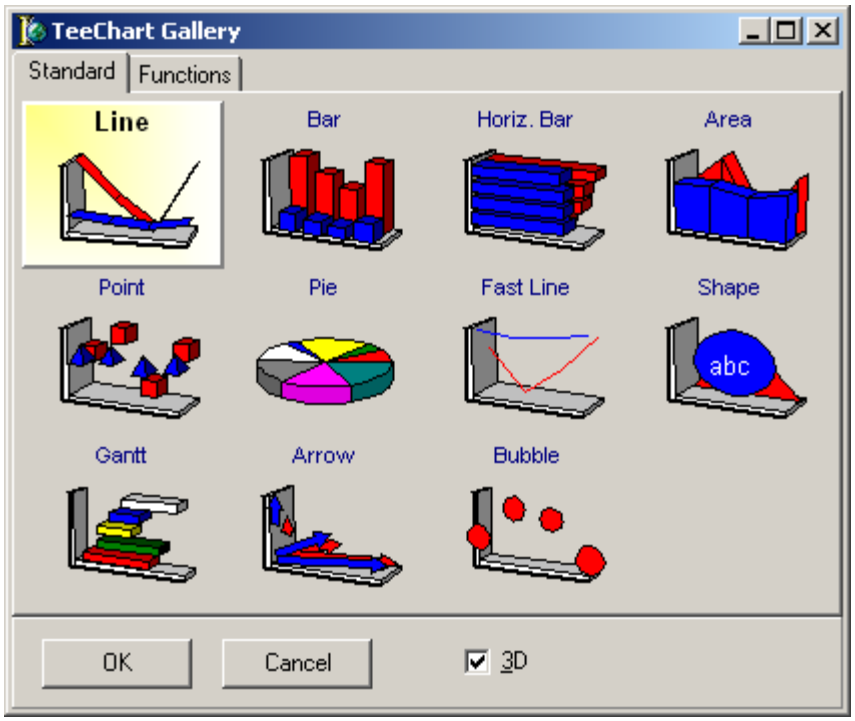
Багато компонентів генератора звітів QR (QuickReport) будуються на основі компонентів керування даними. Інші ж будуються з елементів керування, що не мають відносини до бази даних. Наприклад, за допомогою компонента QRImage у звіт уставляється растрове зображення. Компонент QRDBImage служить для того ж, але одержує дані з поля таблиці бази даних (у цьому випадку, можливо, з поля BLOB).

2.2.2. Створення діаграм за допомогою бібліотеки TeeChart

Діаграми будь-якого типу, які можна створити засобами бібліотеки TeeChart, складаються з наборів крапок даних. Кожен набір являє собою об'єкт, що належить діаграмі. На мал. 2.2.1 показане діалогове вікно, у якому вибирається тип використовуваного в діаграмі набору. Кожен набір має безліч опцій, вибираючи які, можна задати спосіб представлення набору на діаграмі. Існують можливості додавання етикеток, настроювання квітів, вибору різноманітних рамок, завдання ефектів об'ємності і вибору серед безлічі інших опцій.

Установите прапорець 3D при настроюванні палітри компонентів, якщо бажано працювати як із двох-, так і з тривимірними діаграмами.

У каталозі установки Delphi утримується повна документація по бібліотеці TeeChart. Файл TChartVx.doc (x — це номер версії) має формат Microsoft Word. Щоб скористатися повною оперативною довідкою, виберіть будь-як компонент TeeChart або інший елемент у вікні проектування екранної форми Delphi і натисніть <F1>.

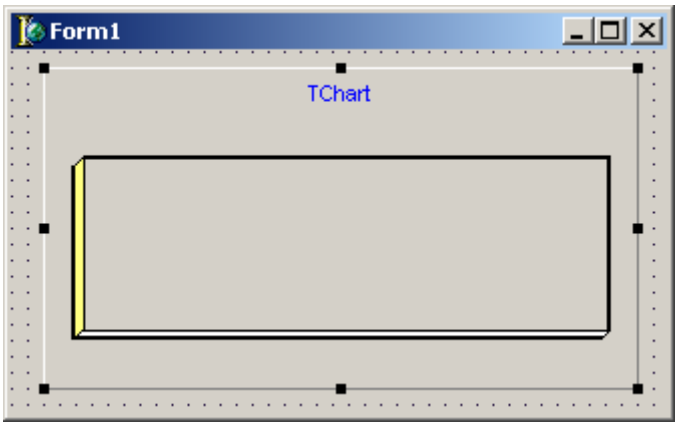


Малюнок 2.2.1 - Щоб вибрати тип набору, використовуйте піктограму у вікні TeeChart Gallery

2.2.2.1. Початок роботи з діаграмами

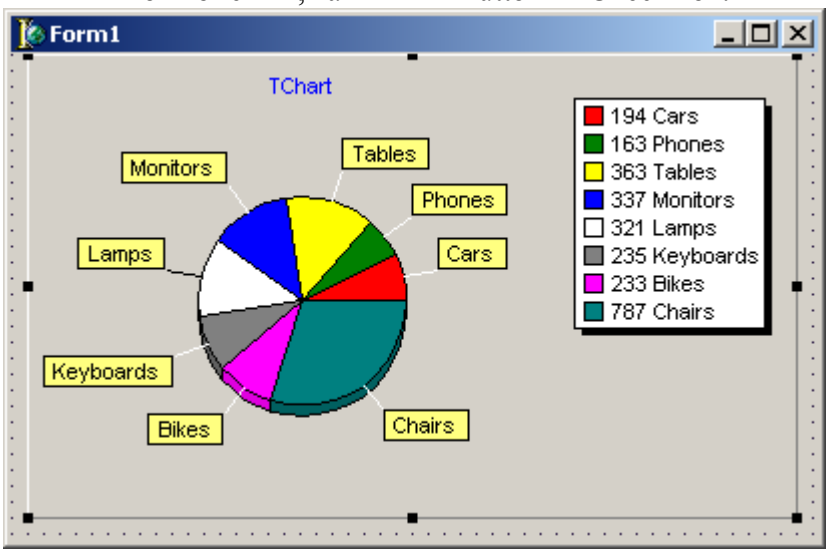
Легше всего почати використання бібліотеки TeeChart зі створення декількох простих діаграм. Запустите Delphi, а потім випливайте приведеним нижче інструкціям (не зберігайте проект).

1) Відкрийте новий додаток.
2) Клацніть на вкладці Additional палітри компонентів.
3) Клацніть спочатку на піктограмі Chart (вона виглядає як маленька кругова діаграма), а потім — усередині вікна форми, щоб помістити новий порожній об'єкт Chart. Вікно проектування екранної форми на вашому екрані повинне виглядати так, як на мал. 2.2.2.
4) Клацніть правою кнопкою миші, коли покажчик буде знаходитися усередині об'єкта Chart. З'явиться контекстне меню діаграм. Відкрийте редактор діаграм бібліотеки TeeChart за допомогою команди **Edit Chart**. (Ще один спосіб зробити це — двічі клацнути на об'єкті Chart.)
5) Щоб додати об'єкт набору в діаграму, клацніть на кнопці редактора **Add**. З'явиться діалогове вікно **TeeChart Gallery**, показане на мал. 2.2.1. Виберіть набір **Pie** зі сховища, а потім клацніть на кнопці **OK** або натисніть <Enter>, щоб закрити діалогове вікно **TeeChart Gallery**. Перемістите вікно редактора убік так, щоб можна було бачити форму й об'єкт Chart. Як показано на мал. 2.2.3, редактор TeeChart відображає зразок діаграми обраного типу з довільними крапками даних, доданими в набір.



Малюнок 2.2.2 - Для створення нової порожньої діаграми у вікні проектування екранної форми використовуйте компонент Chart

Клас TChart є похідним від класу компонента Delphi TPanel, і, отже, об'єкт Chart є буквально розширеним об'єктом Panel. Між іншим, це означає, що в об'єкт Chart можна вставити й об'єкти інших компонентів, таких як TButton і TCheckBox.



Малюнок 2.2.3 - Редактор TeeChart відображає обрану Delphi діаграму з довільними крапками даних

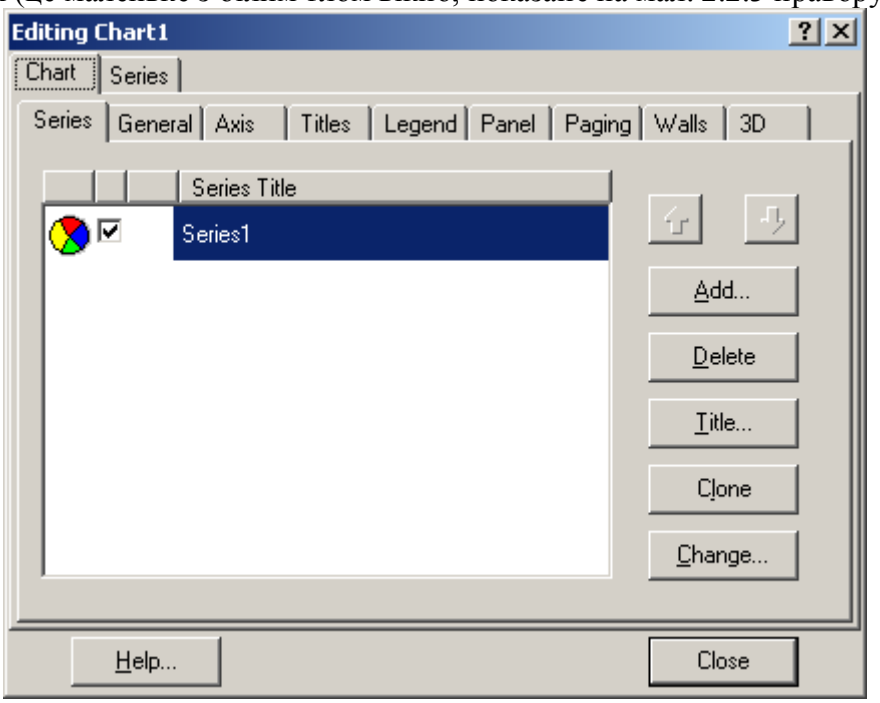
При створенні нової діаграми бібліотека TeeChart автоматично надає довільні крапки даних, щоб можна було побачити, як приблизно буде виглядати діаграма. Ці дані не відображаються під час виконання. Коли запускається програма, діаграмою використовуються власні дані, що можуть надходити з різних джерел.

Випробуйте інші типи наборів діаграм у вікні **TeeChart Gallery** (див. мал. 2.2.1). Повторите попередні операції, починаючи з п. 4, а потім виберіть кнопку Add, щоб додати додаткові об'єкти набору. Діаграма може мати кілька наборів того самого типу — на такому сполученому графіку можна, наприклад, порівняти динамікові курсу різних цінних паперів на фондовому ринку, кожний з яких показаний у виді лінійного графіка на координатній сітці x, y. У діаграмі також можуть поєднуватися набори декількох типів, наприклад, щоб відобразити стовпчасту діаграму і лінійний графік на одній і тій же координатній сітці.

Щоб змінити тип відображуваної діаграми для обраного набору, наприклад для заміни стовпчастої діаграми крапкової, скористайтеся кнопкою **Change** редактори **TeeChart**. Щоб видалити обраний набір, клацніть на кнопці **Delete** (з'явиться попереджуюче повідомлення, якщо кнопка Delete натиснута випадково).

Багато опцій редактора змінюються в залежності від типу набору, що обраний для діаграми. Якщо малюнки в цій главі відрізняються від тих, котрі ви бачите на екрані, то, можливо, вами обраний інший тип набору, наприклад полосчатая діаграма замість лінійної.

Виберіть різні вкладки в редакторі **TeeChart** (мал. 2.2.4), на яких можна знайти масу опцій для установки квітів, стилів діаграм і використання інших можливостей. Щоб випробувати них, клацніть на вкладці **General** і зніміть прапорець об'ємного відображення — тривимірні діаграми будуть перетворені в двомірні. Спробуйте клацнути на вкладці **Legend** і зняти прапорець **Visible**, щоб включити в діаграму панель етикеток, на якій буде виведена розшифровка секторів кругових діаграм (це маленьке з білим тлом вікно, показане на мал. 2.2.3 праворуч угорі).



Малюнок 2.2.4 - У редакторі TeeChart надається безліч опцій для створення і редагування діаграм

Зверніть увагу на те, що в редакторі **TeeChart** мається два ряди вкладок. Дві вкладки, що знаходяться у верхньому ряді, — найважливіші.

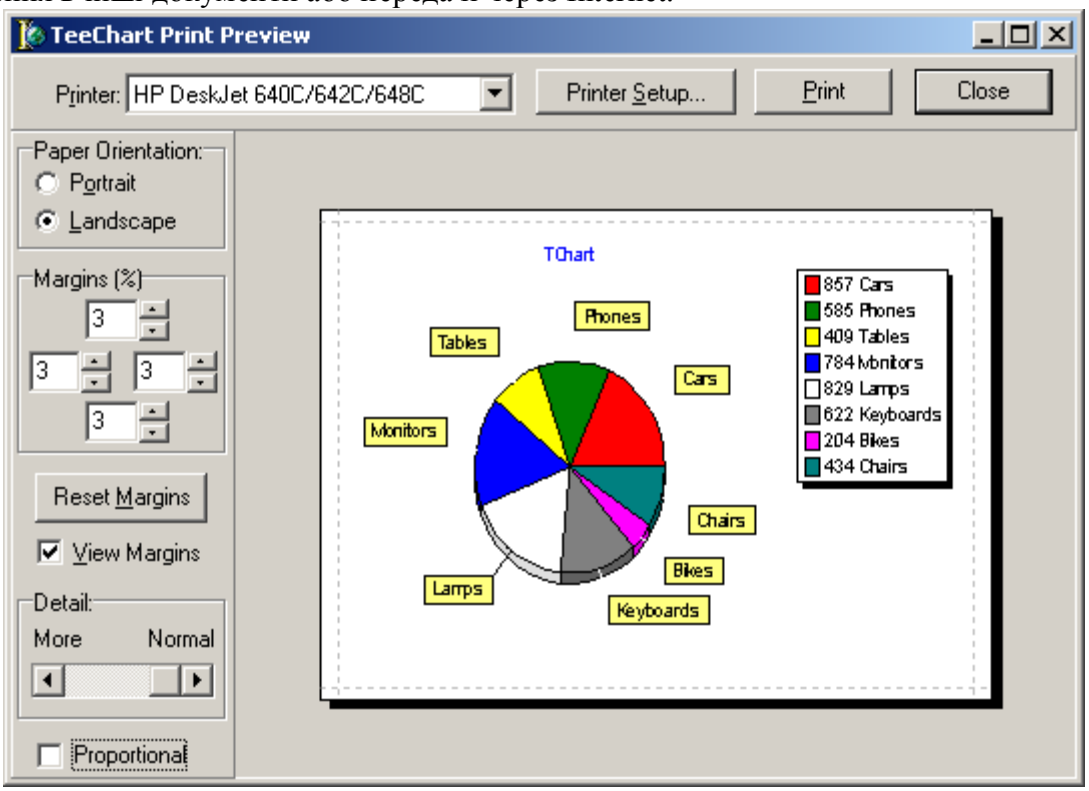
- ***Chart***. Виберіть спочатку вкладку **Chart**, а потім — вкладки в нижньому ряді, щоб змінити властивості діаграми. Наприклад, клацніть на вкладці **Titles** і введіть заголовок діаграми. Зміни негайно відіб'ються в зразку діаграми.
- ***Series***. Клацніть на вкладці **Series** у верхньому ряді, а потім виберіть об'єкт набору зі списку, що розкривається. Потім можете вибрати вкладку з другого ряду, щоб змінити властивості набору. Тепер поэкспериментируем. Клацніть на **Series** у верхньому ряді, виберіть об'єкт Series1, клацніть на вкладці **Marks** у другому ряді і виберіть одну з наступних опцій у панелі **Style**: опцію **Value**, якщо потрібно показати тільки значення крапок даних, опцію **Percent**, якщо потрібно вивести значення у відсотках, або опцію **Label & Percent**, щоб відобразити і текстові етикетки, і значення у відсотках для кожної крапки даних, і т.д.

Є й інший спосіб змінити властивості діаграми. Виберіть об'єкт Chart (спочатку закрийте редактор **TeeChart**), а потім використовуйте вікно Object Inspector, щоб настроїти властивості діаграми, як для інших компонентів Delphi. Більшість властивостей доступно й у редакторі **TeeChart**, і в Object Inspector. Можна використовувати будь-як метод, але, імовірно, ви переконаєтеся, що легше використовувати редактор, оскільки спосіб організації властивостей і опцій на діалогових сторінках дозволяє швидше них знайти. У Object Inspector ті ж самі елементи надані в одному великому вікні, що ускладнює пошук.

У редакторі **TeeChart** опції і можливостей так багато, що охопити них в одній главі неможливо. Однак нижче коротко описується, як використовувати нерозглянуті опції самостійно. У наступних розділах приводяться специфічні опції діаграм.

2.2.2.2. Друк та експорт діаграм

За допомогою бібліотеки TeeChart надаються широкі можливості для печатки і попередньо перегляду (причому кінцеві користувачі можуть скористатися ними під час виконання програми). Крім того, можна зберегти будь-як діаграму в графічному файлі для її включення в інші документи або передачі через Internet.



Малюнок 2.2.5 - Діалогове вікно TeeChart Print Preview

Печатка діаграм у Delphi

Щоб віддрукувати діаграму в Delphi, клацніть правою кнопкою миші усередині об'єкта Chart і виберіть з локального контекстного меню команду **Print Preview**. По цій команді виводиться вікно **TeeChart Print Preview**, як показано на мал. 2.2.5.

У вікні попереднього перегляду діаграма відображається так, як вона буде виглядати при печатці. Маються різноманітні опції для вибору книжкової або альбомної орієнтації листа, настроювання полів сторінки й елементів діаграми (кінцевий результат залежить від типу діаграми — за допомогою смуги прокручування **Chart Detail** можна побачити зміни зображення).

Якщо використовується миша, перетягнете пунктирну лінію полий і перемістите діаграму на сторінці. Спробуйте зробити це зараз. Клацніть мишею усередині вікна попереднього перегляду і перетягнете його, щоб змінити положення і розмір діаграми.

Після завершення настроювання діаграми і вибору опцій (для вибору опції печатки принтера клацніть на кнопці **Printer Setup**) клацніть на кнопці **Print**, щоб почати печатка. Також можете клацнути на кнопці **Close** і закрити вікно **Print Preview.**

Коли закривається вікно **Print Preview**, всі установки приймають значення, задані за замовчуванням. Тому не закривайте вікно доти, поки не завершиться печатка діаграми.

Печатка діаграм під час виконання

Користувачі програми можуть використовувати вікно **Print Preview** і всі опції, описані в попередньому розділі. Для цього додайте в екранну форму об'єкт компонента TButton і двічі клацніть на ньому, щоб створити процедуру обробки події OnClick. Перейдіть у вікно редактора коду Delphi і відшукайте ключове слово implementation. Додайте оператор uses, як показане нижче (додане виділено напівжирним шрифтом):

implementation

```
{$R *.DFM}
uses
  TePrevi;
```

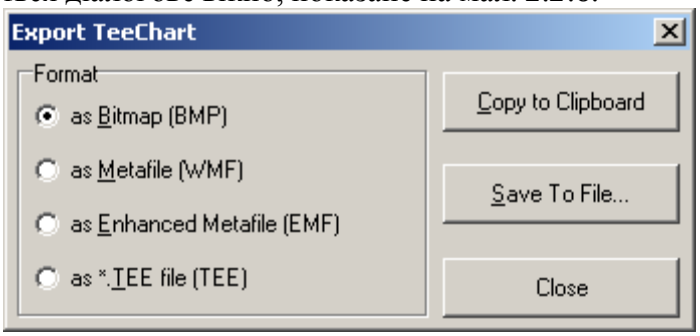
Таким чином, модуль TePrevi (TeeChart Print Preview) став доступний для даного модуля. Тепер можна запрограмувати процедуру обробки події, як описано нижче:

```
procedure TForm.Button1Click(Sender: TObject);
begin
  ChartPreview(Forml, Chart1);
end;
```

Тут викликається процедура ChartPreview з модуля TePrevi, що передаються як аргументи об'єкт екранної форми діаграми (тут — Form1) і об'єкт діаграми (Chart1). У результаті буде виведене вікно **Print Preview**, показане на мал. 2.2.5. У модулі цілком обробляються всі опції печатки і не потрібно писати ніякого додаткового коду. Коли користувач закриває вікно, програма продовжує свою роботу.

Експорт діаграм у Delphi

Щоб експортувати діаграму в Delphi, клацніть правою кнопкою усередині об'єкта діаграми. В открывшет контекстному меню TeeChart (у разі потреби створіть зразок діаграми, як уже порозумівалося в цій главі) виберіть команду **Export Chart**, у результаті виконання якої виводиться діалогове вікно, показане на мал. 2.2.6.



Малюнок 2.2.6 - Діалогове вікно TeeChart Export використовується для копіювання зображення діаграми в буфер обміну Windows або його збереження у файлі зображення

У діалоговому вікні **TeeChart Export** надаються чотири опції формату для копіювання діаграми в буфер обміну Windows або її збереження у файлі на диску.

- ***as Bitmap (BMP)***. Використовується для збереження діаграми у файлі растрового зображення, розмір якого залежить від розміру діаграми.
- ***as Metafile (WMF)***. Використовується для збереження діаграми в стандартному метафайле Windows. Діаграма представляється не у виді зображення, а у виді набору команд, за допомогою яких малюється точно таке ж зображення, як і у вихідній програмі. Метафайлы використовують лише невеликий обсяг дискового простору і легко масштабируются для нових розмірів зображення. Опцію можна використовувати, якщо необхідно, щоб зображення можна було переглядати в Windows 3.1.
- ***as Enhanced Metafile (EMF)***. Використовується для збереження діаграми в розширеному метафайле Windows. Цей файл подібний стандартному метафайлу, але може проглядатися тільки користувачами Windows 95, 98 і NT. Розширені метафайлы мають переваги в багатьох відносинах, включаючи точність відтворення і масштабирования, у порівнянні зі стандартними метафайлами. Розширені метафайлы використовують 32-розрядні координати, а стандартні — 16-розрядні координати. Для досягнення більш високої якості використовуйте розширені метафайлы, де це можливо.
- ***as *.TEE file (TEE)***. Використовується для збереження діаграми у файлі типу TeeChart, тобто у власному форматі діаграм Delphi, що дозволить згодом відкривати збережені

файли безпосередньо з додатків Delphi під час виконання програми при наявності в ній об'єкта Chart.

Після вибору однієї з трьох опцій формату клацніть на кнопці **Copy to Clipboard**, щоб скопіювати зображення в буфер обміну Windows. Потім можете перейти в іншу програму, наприклад у текстовий редактор, і вставити зображення в документ (звичайно, мається на увазі, що програма може обробляти зображення в обраному форматі). Можна також зберегти діаграму у виді файлу, клацнувши на кнопці **Save to File**. У діалоговому вікні, що з'явилося, введіть ім'я файлу і каталог.

Експорт діаграм під час виконання

Для експорту діаграми під час виконання викличте один з наступних чотирьох методів TChart:

SaveToBitmapFile(**const** Filename: **String**);
SaveToMetafile(**const** Filename: **String**);
SaveToMetafileEnh(**const** Filename: **String**);
SaveChartToFile(AChart:TCustomChart; **const** AName:**String**);

Для кожного методу передайте як аргумент ім'я файлу, що може включати букву диска і шлях до файлу. Розширення імені файлу повинне збігатися з типом файлу — воно не додається автоматично. Наприклад, щоб зберегти діаграму як бітове відображення Windows, можна використовувати оператор:

Chart1.SaveToBitmapFile('З:\Test.bmp');

У деяких редакціях оперативної довідки TeeChart попередня процедура невірно іменується як SaveToBitmap. Її правильне ім'я - SaveToBitmapFile.

Подібним чином викликаються й інші функції. Наступні два оператори зберігають діаграму, представлену у виді метафайла Windows або розширеного метафайла (як і раніше, відповідальність за правильне розширення імені файлу, що відповідає його типові, покладається на вас):

Chart1.SaveToMetafile('C:\Test.wmf');
Chart1.SaveToMetafileEnh('З:\Test.emf');

Для четвертого методу крім імені файлу необхідно вказати ім'я об'єкта Chart у якому знаходиться необхідна діаграма. Після цього збережену діаграму можна будеть відкрити під час виконання програми за допомогою методу:

LoadChartFromFile(**Var** AChart:TCustomChart; **Const** AName:**String**);

Для використання методів SaveChartToFile і LoadChartFromFile необхідно додати оператор uses, як показане нижче (додане виділено напівжирним шрифтом):

implementation
*{$R *.DFM}*
uses
TeeStore;

2.2.2.3. Джерела даних діаграм

Як уже демонструвалося в попередньому розділі, за допомогою бібліотеки TeeChart створювати діаграми дуже просто. Але вибір найбільш підходящого типу діаграми для конкретного набору крапок даних найчастіше являє собою досить складну і задачу, що віднімає багато часу. Щоб створити вдалу діаграму, потрібно врахувати, відкіля надходять дані, що вони собою представляють, і мати чітке представлення про призначення графічного відображення цих даних.

У наступних розділах приводяться практичні приклади діаграм, що можуть допомогти вам почати розробку діаграм для своїх власних даних. У цих прикладах використовуються три основних типи джерел даних.

- ***Програмні дані***. Це можуть бути константи, безпосередньо задані в коді програми. Дані можуть обчислюватися під час роботи програми.

- ***Дані з файлу***. Тип файлу вибирається при розробці програми. Це може бути файл даних, що завантажуються з Internet, або текстовий файл з інформацією, що уводиться вручну або вставляється з інших документів.
- ***Дані з БД***. Ці дані завантажуються з таблиць БД. Для підключення діаграми до такого джерела необхідно, як мінімум, використовувати компоненти TDataSource і TTable. (Про ці й інші компоненти, призначених у Delphi для роботи з базами даних, докладно розказано вище.)

У наступних підрозділах описується, як програмувати створення діаграм, що використовують дані кожного з трьох джерел. У прикладах програм також демонструються додаткові прийоми роботи з діаграмами.

Діаграми, що представляють програмні дані

Один з методів передачі даних у діаграму складається в їхньому безпосереднім введенні у вихідний код програми. Також можна використовувати цей метод для відображення даних, породжуваних різними процедурами. Для таких типів даних використовується компонент Chart з категорії Additional палітри.

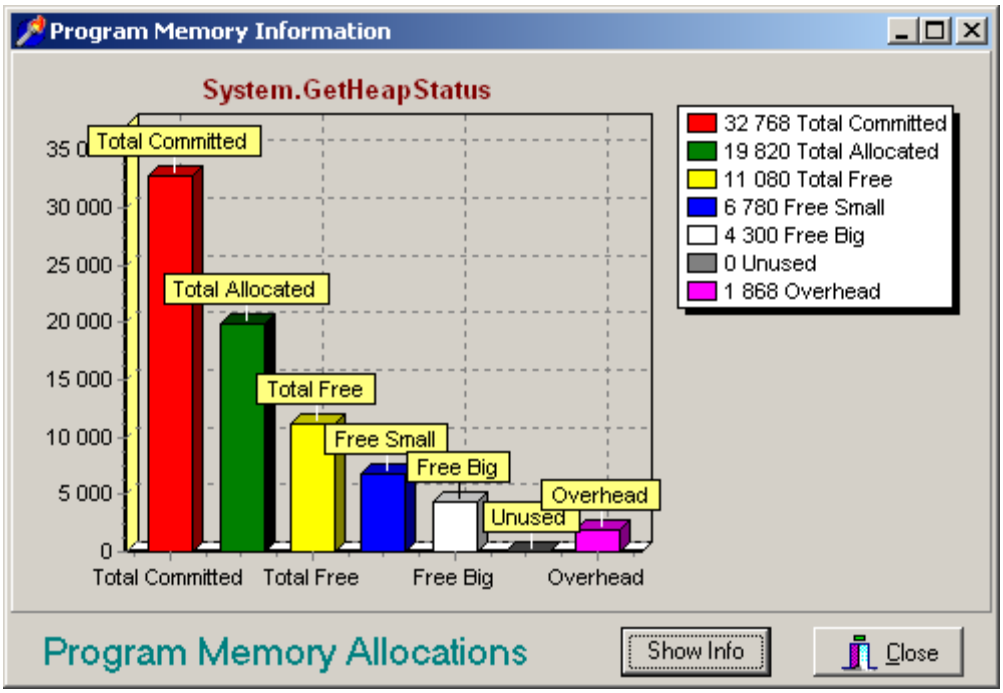
Помістите об'єкт компонента TChart в екранну форму, двічі клацніть на ньому, виберіть у контекстному меню команду **Edit Chart**, і відкриється редактор **TeeChart**. Клацніть на кнопці **Add**, щоб створити один або кілька об'єктів наборів і вибрати тип діаграми для відображення. Цього разу можна також вибрати різні опції для діаграми.

Після завершення проектування діаграми додайте операторы програми, щоб передати в діаграму реальні дані. Для вставки даних викличте метод Add, Add, Add або AddXY для об'єктів наборів, за допомогою яких будуть додані крапки даних. Наприклад, за допомогою наступного оператора до об'єкта Series1 додаються перемінна AValue (мається на увазі тип Double) і етикетка крапки діаграми (рядок 'Value'), а також указується, що для об'єкта Chart повинний бути обраний колір clTeeColor:

```
Series1.Add(AValue, 'Value', clTeeColor);
```

Замість кольору clTeeColor можна використовувати будь-як інше значення типу TColor, наприклад clBlue або clRed, а також задати кольору в шестнадцатеричном форматі. (Опис TColor приводиться в оперативній довідці Delphi).

Розглянемо додаток MemInfo у якому показано як надавати дані для діаграми. На мал. 2.2.7 відображена діаграма пам'яті програми, використовувана як виведений звіт за допомогою функції Delphi System.GetHeapStatus. Необхідно клацнути на кнопці **Show Info**, щоб вивести різні дані об окремих складових використовуваної пам'яті. Цей модуль програми можна додати у власний проект і використовувати для рішення виникаючих проблем або просто як зведення про використовувану пам'ять комп'ютера. У прикладі 2.2.1 представлений вихідний код програми.



Малюнок 2.2.7 - У додатку MemInfo відображається інформація про використання пам'яті програмою і демонструється додавання даних до діаграми

Приклад 2.2.1 – Вихідний код програми MemInfo

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls,
  TeEngine, Series, ExtCtrls, TeeProcs, Chart, Buttons;

type
  TMainForm = class(TForm)
    Chart1: TChart;
    Button1: TButton;
    Series1: TBarSeries;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

var
```

```
  HeapStatus: THeapStatus;

procedure TMainForm.Button1Click(Sender: TObject);
begin
  HeapStatus := System.GetHeapStatus;
  with Series1, HeapStatus do
  begin
//  Add(TotalAddrSpace, 'Total address space', clTeeColor);
//  Add(TotalUncommitted, 'Total Uncommitted', clTeeColor);
    Add(TotalCommitted, 'Total Committed', clTeeColor);
    Add(TotalAllocated, 'Total Allocated', clTeeColor);
    Add(TotalFree, 'Total Free', clTeeColor);
    Add(FreeSmall, 'Free Small', clTeeColor);
    Add(FreeBig, 'Free Big', clTeeColor);
    Add(Unused, 'Unused', clTeeColor);
    Add(Overhead, 'Overhead', clTeeColor);
  end;
end;

end.
```

Щоб одержати дані про використання пам'яті, у програмі MemInfo викликається функція GetHeapStatus з модуля System. Вона повертає запис THeapStatus, що привласнюється перемінної HeapStatus, оголошеної в секції реалізації модуля MemInfo. За допомогою наступних операторів запис HeapStatus заповнюється такими значеннями, як TotalCommitted і TotalAllocated, що представляють інформацію про використання пам'яті програмою. (Зведення про кожної перемінній у записі THeapStatus можна одержати в оперативній довідці Delphi):

HeapStatus := System.GetHeapStatus;

Щоб додати отримані значення в об'єкт діаграми Series1, у процедурі обробки події OnClick кнопки **Show Info** викликається метод Add. Наприклад, за допомогою наступного оператора в діаграму додається значення TotalFree:

Add(TotalFree, 'Total Free', clTeeColor);

Оператор створює новий стовпець для значення в HeapStatus.TotalFree з етикеткою, представленою переданим рядком. Значення clTeeColor задає режим автоматичного вибору кольору для цього елемента діаграми. Інший елемент додається аналогічно.

У додатку MemInfo не відображаються смуги для двох значень GetHeapStatus, TotalAddrSpace і TotaLUncommitted. Ці глобальні системні константи, не зв'язані безпосередньо з іншими значеннями пам'яті програми, що показані в діаграмі. Це приклад того, як занадто велика кількість даних може скоріше зашкодити, захаращуючи діаграму. Тому два оператори закомментированы на початку процедури обробки подій об'єкта Button.

Метод Add, використовуваний у додатку MemInfo, підходить для простих стовпчастих діаграм. Він також непоганий для створення кругових діаграм, у яких відображається розподіл деякого сумарного параметра між складовими. Інші типи діаграм, такі як лінійні графіки, відображають дані, розкладені по координатних осях x и y. Для цих типів діаграм іноді використовуються тільки значення x або тільки в, або обоє значення. Щоб додати крапки даних для цих типів наборів, викличте метод Add, Add або AddXY. У наступному операторі додається крапка YValue (мається на увазі, що відповідна перемінна оголошено в програмі як Double), на осі в задається напис ‘Y-Label’ і колір відображення цих даних (червоний):

Series1.Add(YValue, 'Y-Label', clRed);

Координата x для цієї крапки даних обчислюється автоматично, використовуючи значення, відібрані на вкладці **Axis** редактори **TeeChart**. Аналогічно, якщо використовується значення x, викличте процедуру Add, щоб створити нову крапку даних для набору:

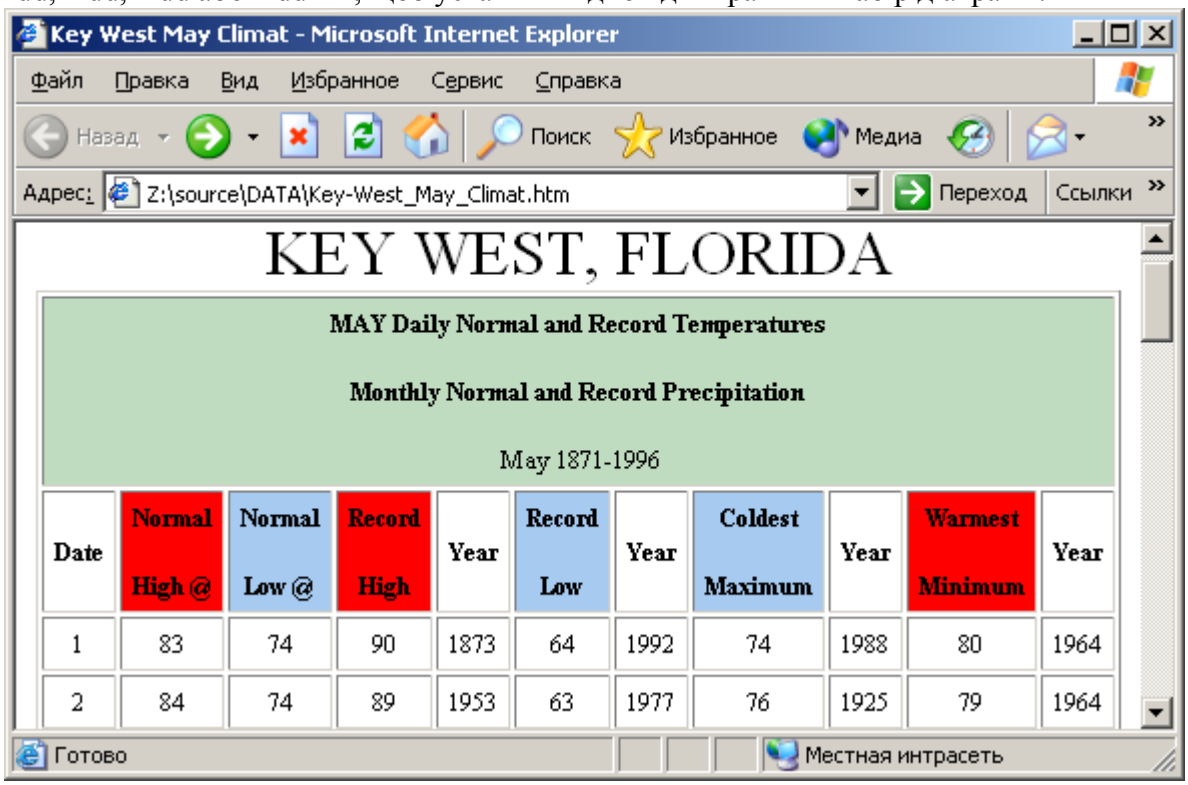
Series1.Add(XValue, 'X-Label', clWhite);

І нарешті, якщо для діаграми використовуються два значення як координати осей x и y, викличте процедуру AddXY, використовуючи оператор, подібний приведеному нижче:

Series1.AddXY(XValue, YValue, 'X-Label', clBlue);

Діаграми, що відображають дані з файлу

Можливо, найбільш розповсюдженим джерелом інформації при побудові діаграм є файли. Основна проблема при читанні таких даних полягає в знанні або визначенні формату файлу. Це може бути текстовий файл, файл, завантажений з Internet, або файл, що містить значення, що зберігаються в двоичном виді. Якщо вам відомий формат файлу або він якимсь образом визначається, тоді можете написати процедуру завантаження даних у пам'ять, а потім викликати метод Add, Add, Add або AddXY, щоб уставити відповідні крапки в набір діаграми.



Малюнок 2.2.8 - Дані про погоду для Ки-Уэста за травень з 1871 по 1996 рік, отримані по Internet зі служби National Weather Service

Як реальний приклад, що демонструє проблеми, що виникають при читанні файлів даних, може служити додаток MayTemp. Цей додаток відображає графік числових даних, що завантажуються з web-вузла Internet http://www-mfi.nhc.noaa.gov. Це web-адреса офісу служби погоди National Weather Service у Майями (штат Флорида), що надає прогноз погоди, діаграми й іншу інформацію. У нашому випадку ми цікавилися даними про погоду в Ки-Уэст (штат Флорида) за травень і, зокрема, значеннями середньої, найвищої і щонайнижчої температур. Ці дані розміщені у файлі Key-west_May_Climat.htm, показаному на мал. 2.2.8 у тім виді, у якому він відображається за допомогою Internet Explorer.

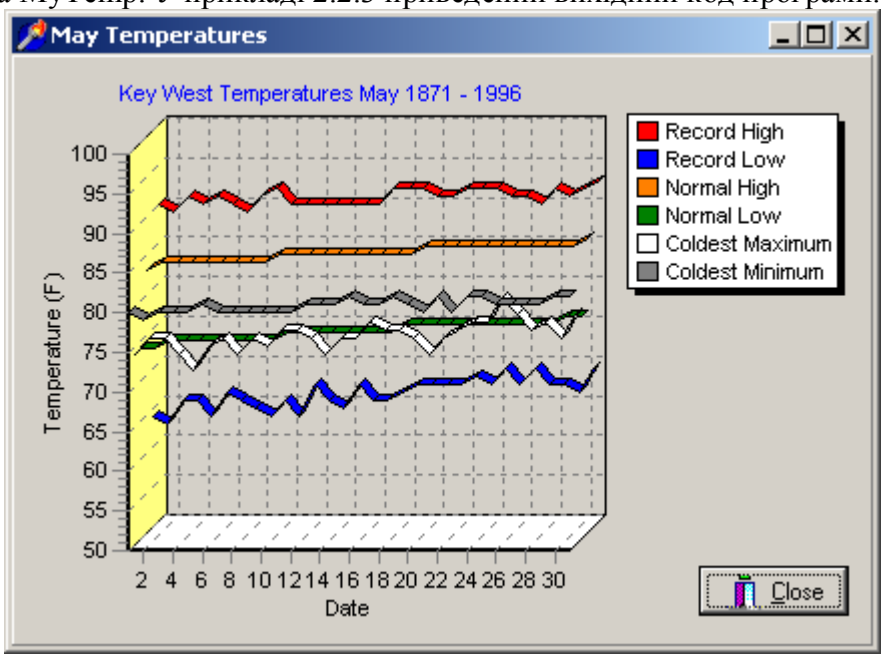
Ці дані потрібно переслати з Internet у файл, а потім у програму Delphi для відображення у виді графіка. Спочатку вибираємо дані у вікні Internet Explorer, натискаємо <Ctrl+C>, щоб скопіювати текст у буфер обміну, а потім викликаємо редактор Notepad. Натиснувши <Ctrl+V>, уставляємо дані в текстовий редактор. Після видалення непотрібного тексту зберігаємо дані в текстовому файлі KeyWestMayClimate.txt. Цей файл показаний у прикладі 2.2.2.

Приклад 2.2.2 – Фрагмент даних з файлу KeyWestMayClimate.txt

```
1  83  74  90  1873  64  1992  74  1988  80  1964
2  84  74  89  1953  63  1977  76  1925  79  1964
3  84  75  91  1873  66  1992  76  1897  80  1994
```

4 84 75 90 1956 66 1925 74 1989 80 1984
5 84 75 91 1873 64 1992 72 1889 80 1991
6 84 75 90 1946 67 1992 75 1911 81 1984
7 84 75 89 1956 66 1954 76 1903 80 1991
8 84 75 91 1873 65 1988 74 1898 80 1991
9 84 75 92 1873 64 1992 76 1928 80 1978
10 85 75 90 1873 66 1960 75 1891 80 1995
11 85 76 90 1873 64 1944 77 1891 80 1995
12 85 76 90 1926 68 1944 77 1900 80 1996

Дані тут показані в звичайній текстовій формі. Рядки не зовсім точно вирівняні, а стовпці не озаглавлені, але в цьому випадку ми заздалегідь знаємо, що це за значення. Тому їх можна відразу ж читати і заносити в перемінні програми. Оскільки ми обробляємо текстовий файл, легше всего завантажити дані за допомогою процедури Read Object Pascal і одержати їхнє візуальне представлення. На мал. 2.2.9 показана остаточна діаграма, відображувана за допомогою додатка MyTemp. У прикладі 2.2.3 приведений вихідний код програми.



Малюнок 2.2.9 - Підсумкова діаграма, що відображає дані температури з рядка даних приклада 2.2.2

Приклад 2.2.3 - Вихідний код програми MayTemp

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, TeEngine, Series, ExtCtrls, TeeProcs, Chart,
  StdCtrls, Buttons, TeCanvas;

type
  TMainForm = class(TForm)
    Chart1: TChart;
    Series1: TLineSeries;
    Series2: TLineSeries;
    Series4: TLineSeries;
    Series5: TLineSeries;
```

```
    Series6: TLineSeries;
    Series3: TLineSeries;
    BitBtn1: TBitBtn;
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormActivate(Sender: TObject);
var
  FileName:String;
  F: TextFile;                    //Перемінна файлу
  Date: Integer;                  //Перший стовпець у файлі даних.
  NormalHigh : Integer;           //Наступний стовпець
  NormalLow : Integer;            //і т.д. ...
  RecordHigh : Integer;
  RecordHighYear : Integer;
  RecordLow : Integer;
  RecordLowYear : Integer;
  ColdestMaximum : Integer;
  ColdestMaximumYear : Integer;
  WarmestMaximum : Integer;       // ... до
  WarmestMaximumYear : Integer;   //останнього стовпця у файлі даних
begin
{Вказівка імені текстового файлу і повного шляху до нього }
  FileName:=ExtractFilePath(ParamStr(0))+'\KeyWestMayClimate.txt';
  AssignFile(F, FileName);   //Ініціалізація перемінної файлу
  Reset(F);                  //Відкрити файл
  while not Eof(F) do        //Повторити до кінця файлу
  begin
    Read(F,                  //Читати один рядок даних
      Date,                  //в окремі перемінні
      NormalHigh,
      NormalLow,
      RecordHigh,
      RecordHighYear,
      RecordLow,
      RecordLowYear,
      ColdestMaximum,
      ColdestMaximumYear,
      WarmestMaximum,
      WarmestMaximumYear);   //Кінець оператора Read
```

{Один рядок файлу даних завантажений у цю крапку. Наступні оператори додають дані крапок у кожний із шести об'єктів наборів лінійної діаграми. Порожні строкові аргументи можуть використовуватися для іменування крапок даних. Ці рядки тут не використовуються, оскільки вісь X у цьому прикладі діаграми уже відображає значення дня (1, 2, ..., 31).}

```
    Series1.AddXY(Date, NormalHigh, '', clTeeColor);
    Series2.AddXY(Date, NormalLow, '', clTeeColor);
    Series3.AddXY(Date, RecordHigh, '', clTeeColor);
    Series4.AddXY(Date, RecordLow, '', clTeeColor);
    Series5.AddXY(Date, ColdestMaximum, '', clTeeColor);
    Series6.AddXY(Date, WarmestMaximum, '', clTeeColor);

  end;
end;

end.
```

У програмі MayTemp використовуються стандартні прийоми Object Pascal для відкриття текстового файлу і читання даних з нього. Щоб ці операції вироблялися під час виконання, у програмі використовується процедура обробки події OnActivate для екранної форми головного вікна додатка. Це гарантує, що дані завантажені і правильне додані в об'єкти набору діаграми при запуску програми, але перед тим, як вікно стане видимим.

У процедурі обробки події TMainForm.FormActivate з'являються перемінні типу Integer для кожного значення даних — один перемінна на один елемент для кожного рядка (див. мал. 2.2.9). Щоб вважати файл даних у ці перемінні, у процедурі також з'являється файлова перемінна:

```
var
  F: TextFile;
```

За допомогою двох операторів перемінна инициализируется, при цьому використовується строкова перемінна FileName і відкривається файл, готовий до читання даних:

```
AssignFile(F, FileName);
Reset(F);
```

Для вказівки імені і повного шляху до текстового файлу використовується наступний рядок коду, де ім'я і шлях до файлу заносяться в строкову перемінну FileName (програма завжди буде шукати файл у тім же каталозі де знаходиться сам файл додатка):

```
FileName:=ExtractFilePath(ParamStr(0))+'\KeyWestMayClimate.txt';
```

При цьому функція ParamStr(0) повертає ім'я файлу, що виконується, (файлу додатка), а функція ExtractFilePath витягає повний шлях до цього файлу.

Далі в циклі while продовжується виконання, поки всі рядки даних не будуть завантажені в перемінні програми. У цьому фрагменті, наприклад, показано, як програма завантажує перше значення в кожен рядок, що представляє дані (1, 2, 3, ..., 31):

```
while not Eof(F) do
  begin
    Read(F, Date,
    …
end;
```

Інші перемінні завантажуються в такий же спосіб, за допомогою одного оператора Read, при цьому вони представляються у виді списку і розділяються комами.

Для читання окремих значень однієї і того ж вихідного текстового рядка в окремі перемінні використовується процедура Read. Для читання цілого рядка в перемінну типу string використовується процедура Readln.

Після завантаження кожного рядка даних більшість значень уставляється програмою в набір діаграми. У даному випадку діаграма має шість об'єктів набору, по одному для кожного рядка, виведеної в остаточному графіку (див. мал. 2.2.9). Кожна крапка даних додається за допомогою виклику методу AddXY. Наприклад, за допомогою наступних двох операторів уставляються значення NormalHigh і NormalLow у відповідні об'єкти набору:

```
Series1.AddXY(Date, NormalHigh, ' ', clTeeColor);
Series2.AddXY(Date, NormalLow, ' ', clTeeColor);
```

Перший аргумент, Date, той самий для обох крапок даних — вісь x у цьому випадку представляє день місяця. Наступний аргумент використовується як значення для цієї крапки даних. Строковий аргумент у даному випадку не використовується, оскільки в діаграмі уже відображаються значення температури по осі в. Останній аргумент указує на використання попередньо запрограмованого кольору для цієї крапки даних.

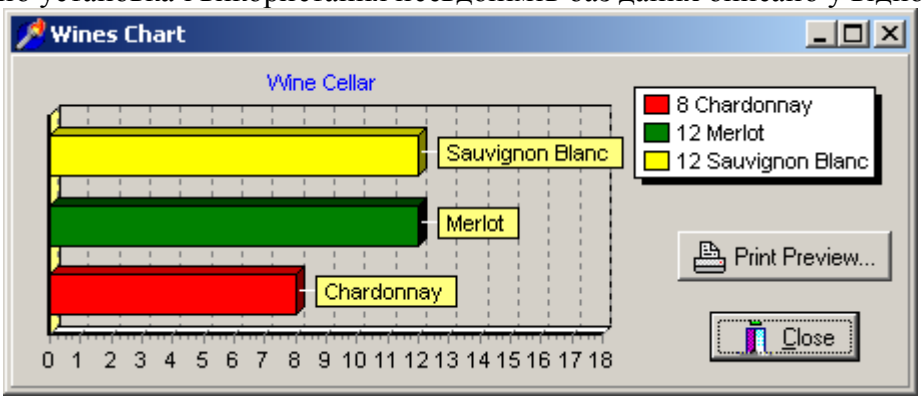
У додатку MyTemp діаграма будується по випадковим даним TeeChart, і вона не дуже добре виглядає під час проектування. Так виходить тому, що випадкові значення виходять за межі обмеженого діапазону координат x и y, якому необхідно використовувати для цієї діаграми. Щоб відключити генерацію випадкових даних, можна відкрити редактор **TeeChart**, клацнути на вкладці **Series**, а потім вибрати кожен об'єкт набору зі списку, що розкривається. Для кожного набору виберіть вкладку **Data Source** і опцію **No Data** зі списку джерел, що розкривається, даних. Якщо необхідно відключити генерацію випадкових даних для розроблювальних діаграм, вивчите методику роботи з цією частиною редактора **TeeChart**.

Діаграми, що представляють дані з БД

Третій, і останній, спосіб одержати дані для діаграми — завантажити значення полів з таблиць БД. У цьому способі для відкриття бази даних використовуються такі компоненти, як TTable і TDataSource, що розглядалися вище. Однак замість компонента TChart, що використовувався дотепер, для завантаження даних з таблиці БД використовується компонент TDBChart, розташований на вкладці Data Controls палітри. Компонент TDBChart поводиться точно так, як TChart, але його дані надходять з таблиці БД. Всі інші властивості й опції аналогічні описаними в попередніх підрозділах.

У додатку WinesChart показано, як створювати діаграму за даними, що зчитується з БД. На мал. 2.2.10 показана діаграма, що формується програмою відповідно до бази даних Wines. У прикладі 2.2.4 приведений вихідний код програми.

Щоб додаток коректний працювало з базою даних необхідно створити для неї псевдонім. Для створення псевдоніма БД виберіть команду Tools⇨Database Desktop, а потім у вікні програми **Database Desktop** виберіть команду Tools⇨Alias Manager Клацніть на кнопці **New** і введіть ім'я нового псевдоніма, наприклад WINES. Установите каталог, що відповідає каталогові в якому розташована БД. Клацніть на кнопці **Keep New**, а потім виберіть **OK**. Відповісти **Yes** на запрошення зберегти псевдоніми. Тепер програма повинна нормально працювати з БД. Більш докладно установка і використання псевдонімів баз даних описано у відповідній главі.



Малюнок 2.2.10 - У програмі WinesChart показано, як можна зчитувати дані з таблиці бази даних і відображати них у виді діаграми

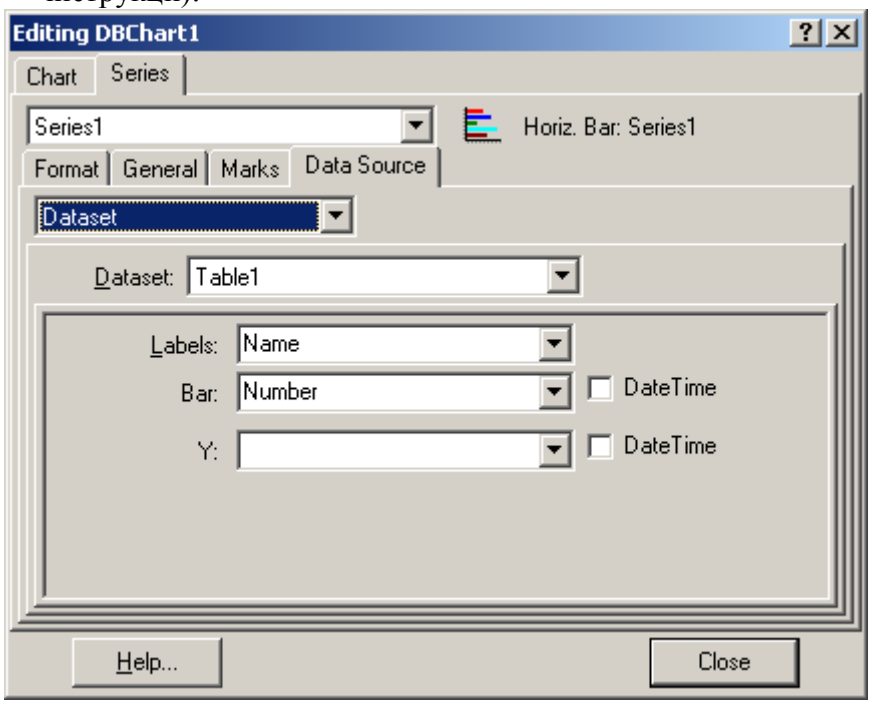
Приклад 2.2.4 - Вихідний код програми WinesChart

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, TeEngine, Series, ExtCtrls, TeeProcs, Chart, DBChart,
  Db, DBTables;

type
  TMainForm = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBChart1: TDBChart;
    Series1: THorizBarSeries;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure BitBtn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses
  TePrevi;

procedure TMainForm.BitBtn2Click(Sender: TObject);
begin
  ChartPreview(MainForm, DBChart1);
end;

end.
```

Щоб створити додаток WinesChart, виконаєте наступні операції.

1) Вставте об'єкти компонентів TTable і TDataSource в екранну форму разом з об'єктом компонента TDBChart. Уставте також двох кнопок Print Preview і Close.
2) Установите значення для властивості DatabaseName об'єкта Table1 рівним WINES. Потім заміните значення властивості Active значенням True.
3) Установите значення для властивості DataSet об'єкта DataSource1 рівним Table1.
4) Виберіть об'єкт DBChart1 і клацніть правою кнопкою миші, викликавши контекстне меню. Виберіть команду **Edit Chart** і перейдіть у редактор **TeeChart**.
5) Клацніть на кнопці редактора **Add** і виберіть набір діаграми **Horiz. Bar**. Клацніть на кнопці **OK**. З'явиться приклад діаграми для випадкових даних.

6) Щоб зв'язати діаграму з базою даних WINES, виберіть вкладку **Series** редактори **TeeChart** угорі вікна. Оскільки мається лише один об'єкт набору, він вже обраний (у своїй програмі, якщо використовується кілька об'єктів набору, вибирайте об'єкти по черзі зі списку, що розкривається, а потім виконуйте з ним наступний пункт інструкції).



Малюнок 2.2.11 - Використовуйте редактор TeeChart для підключення набору діаграми до таблиці Table бази даних, у даному випадку — до Wines

7) Клацніть на вкладці **Data Source** у редакторі **TeeChart**. Відкрийте список, що розкривається, нижче другого ряду корінців вкладок і виберіть елемент Dataset. Установите Table1 у поле **Dataset:**. У такий спосіб діаграма зв'язується з таблицею бази даних. Щоб указати, які поля необхідно відображати, виберіть Name у поле **Labels:** і Number — у поле **Bar:**. При такому настроюванні буде створена діаграма, що відображає число пляшок у винному льосі, причому кожен стовпець буде мати етикетку, що відповідає назві вина (полю Name у БД). На мал. 2.2.11 показаний редактор TeeChart із заповненою сторінкою Series⇨Data Source.
8) Закрийте редактор **TeeChart**, щоб переглянути завершену діаграму у вікні форми. На діаграмі відображені зведення, витягнуті з бази даних WINES. Вам не потрібно запускати програму, оскільки властивість Active об'єкта Table1 установленоі рівним True і дані актуальні усередині Delphi.

Щоб завершити програму, можна змінити заголовки діаграми і вибрати інші властивості. На практиці саме на цьому етапі найкраще проводити експерименти з різними опціями діаграми. У тексті програми (див. приклад 2.2.4) показано, як програмувати кнопку Print Preview, що вже розглядалося в цій главі.

Печатка діаграм як частини звіту по базі даних розглядається далі в цій главі.

2.2.3. Створення звітів за допомогою засобу QuickReport

За допомогою бібліотеки QuickReport можна істотно спростити підготовку до висновку на печатку в додатку, особливо якщо необхідно створити звіт на основі даних з файлу або бази даних. У комплекті з бібліотекою поставляється редактор, що дозволяє спростити компонування сторінок. Використовуючи підручний засіб попереднього перегляду, на екрані можна одержати точно таке ж зображення звіту, яким воно буде при остаточній печатці. Крім того, звіт можна зберегти у файлі на диску.

У каталозі установки Delphi утримується повна документація по бібліотеці QuickReport. Відкрийте файл Qrptxman.doc (x- це номер версії) за допомогою Microsoft Word. Щоб викликати повну оперативну довідку для з'ясування деталей, виберіть будь-як компонент на вкладці QReport палітри і натисніть <F1>.

2.2.3.1. Початок роботи зі звітами

Засіб QuickReport являє собою *генератор звітів по розділах*. Двома словами, це означає, що звіт, об'єкт компонента TQuickRep, створюється шляхом вставки одного або декількох об'єктів розділів (band) на порожню сторінку. У кожен розділ містяться різні елементи, що повинні включатися в звіт: заголовки, що текет дата і час, номери сторінок, полючи запису бази даних, підсумкові дані й інші елементи. При печатці звіту генератором звітів QuickReport розділи заповнюються даними і повторюються при необхідності, у результаті чого формується кінцевий документ. Наприклад, на кожній сторінці автоматично збільшується значення об'єкта номера сторінки і полючи записи заповнюються даними, що витягаються з таблиць БД.

При роботі з QuickReport ви переконаєтеся, що розділи генератора звітів і їхніх об'єктів "високоінтелектуальні". У більшості випадків можна скомпонувати непоганий звіт, створивши кілька розділів і помістивши в них відповідні об'єкти. При цьому практично не потрібно програмувати що-небудь вручну. А за допомогою зручного засобу попереднього перегляду, яке можна використовувати й у режимі проектування, і під час виконання додатка, легко переглядати компонування звіту, не витрачаючи даремно папір. Коли усі виправлення в рядках будуть зроблені, просто клацніть на кнопці **Print speed** у вікні попереднього перегляду, щоб направити результат безпосередньо на принтер. Звіти також можна зберігати у файлах на диску, а потім повторно завантажувати у вікно попереднього перегляду для печатки.

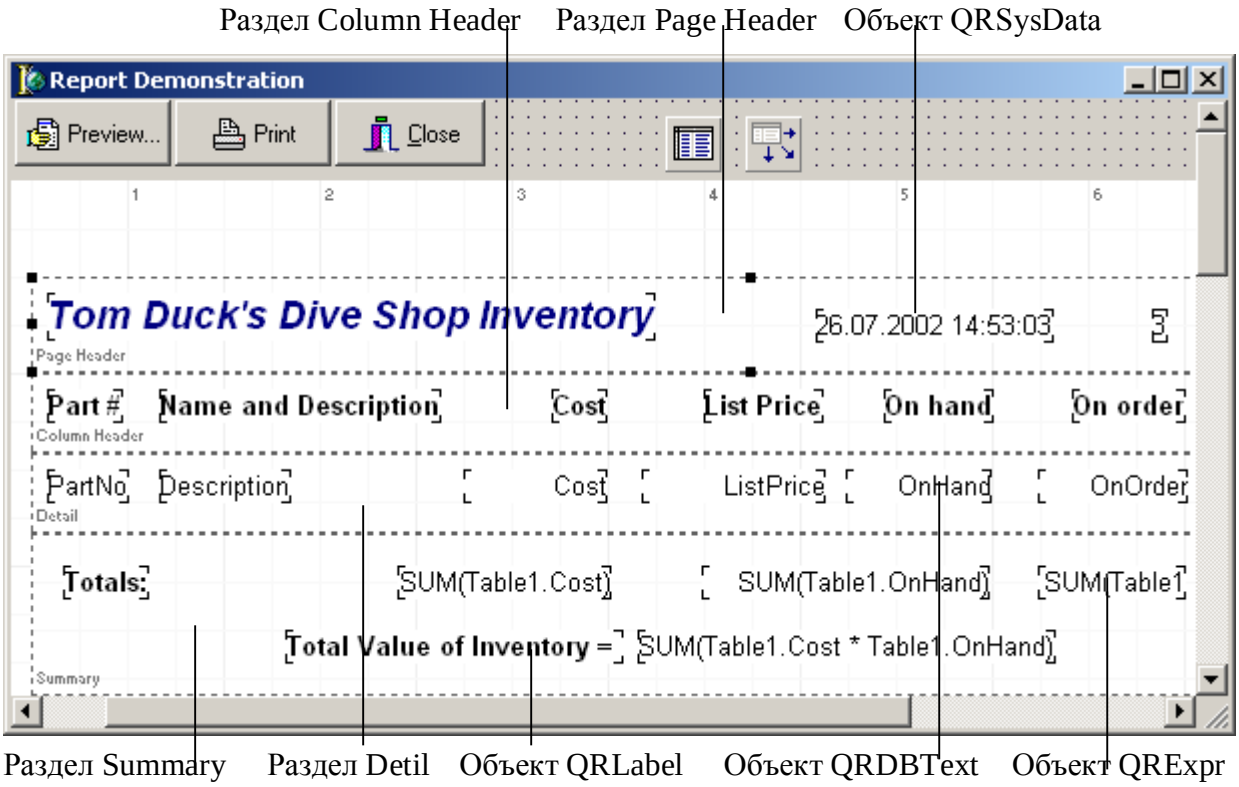
Щоб навчитися використовувати генератор звітів QuickReport і його компоненти, виконуйте приводяться в цьому і наступному підрозділах інструкції. Навіть якщо у вас немає принтера, послідовно виконуючи вказівки, що приводяться в підрозділах цієї глави, ви зможете освоїти проектування звітів. Розглянемо додаток Report для ілюстрації роботи зі звітами. Для зв'язку додатка з демонстраційними базами даних необхідно виконати наступне.

1) Клацніть на вкладці сторінки Data Access, виберіть компонент TTable і помістите його в екранну форму. Виберіть компонент TDataSource і помістите об'єкт даного компонента поруч з об'єктом Table1. Точність розміщення не має значення — жоден з компонентів не буде видний у працюючому додатку.
2) Клацніть спочатку на компоненті Table1, а потім — на вкладці Properties у вікні Object Inspector. Установите значення властивостей у такий спосіб: для DatabaseName — DBDEMOS, для TableName — Parts.db і для Active — True. Якщо псевдонім DBDEMOS не включений у список властивості, що розкривається, DatabaseName компонента Table1, переустановите Delphi і запустите неї знову. Інший варіант - вибрати будь-як іншу базу даних і таблицю, що мається у вашій системі, а потім посилатися на їхній полючи замість пропонованих тут. Більш докладні зведення про використання компонентів баз даних можна почерпнути з відповідної глави.
3) Клацніть на об'єкті DataSource1 і, використовуючи Object Inspector, установите значення властивості DataSet рівним Table1.
4) Виберіть вкладку QReport на палітрі VCD і клацніть на першій піктограмі ліворуч — QuickRep. Потім клацніть на поле вікна форми, щоб створити порожній об'єкт звіту, за замовчуванням називаний QuickRep1. На мал. 2.2.12 показане вікно екранної форми поки ще в режимі проектування. Сітка і номери рядків і стовпців, що видні на екрані, призначені для полегшення компонування елементів звіту і не будуть включатися в готовий документ.
5) Клацніть у середині порожнього об'єкта звіту, а потім у вікні Object Inspector двічі клацніть на знаку "плюс" (+), розташованому ліворуч від властивості Bands. Буде відкритий список подсвойств, кожне з яких установлене рівним True або False. Якщо значення подсвойства дорівнює True, створюється розділ, що заноситься в об'єкт звіту.

6) Установите значення подсвойства HasDetail рівним True. Тим самим буде створений об'єкт роздягнула в звіті. Якщо придивитися повнимательней, то можна побачити блідий напис Detail усередині роздягнула. Вона вказує тип розділу, але не друкується в готовому звіті. На мал. 2.2.12 показаний об'єкт QuickRep1 з об'єктом роздягнула.
7) Переконаєтеся, що об'єкт QuickRep1 як і раніше обраний (він включений у список, що розкривається, у вікні Object Inspector), і установите значення властивості DataSet рівним Table1. Тепер об'єкт звіту зв'язаний з компонентами бази даних, з яких витягається інформація для остаточного документа.
8) Клацніть усередині об'єкта роздягнула Detail, щоб виділити його, і зверніть увагу, що в Object Inspector відображається ім'я об'єкта DetailBand1. При формуванні документа роздягнув Detail буде повторюватися для того, щоб були включені вес рядки даних.
9) Щоб надати ці дані, помістите детальний об'єкт на розділ. Знайдіть піктограму компонента TQRDBText на вкладці QReport палітри. (На цій вкладці розміщена безліч піктограм. Затримаєте покажчик миші над кожної і почекайте якийсь час, щоб побачити її назва. Так можна знайти потрібний компонент.) Клацніть спочатку на компоненті TQRDBText, а потім — усередині роздягнула звіту Detail. Можете перетягнути текстовий об'єкт (названий за замовчуванням QRDBText1) у будь-яке місце усередині роздягнула Detail — бажано куди-небудь уліво.
10) Переконаєтеся, що об'єкт QRDBText1 обраний, потім у вікні Object Inspector установите значення для двох властивостей: для DataSet — Table1 і для DataField — PartNo. Тим самим указується, що в першому стовпці звіту перелічуються номери частин елементів. Зверніть увагу на те, що в розділі відображається ім'я обраного поля.
11) Повторите п. 9, щоб додати додаткові поля таблиці бази даних у розділ Detail. Наприклад, повторно виберіть компонент TQRDBText у палітрі і клацніть у розділі Detail, щоб створити об'єкт QRDBText2. Установите значення властивості цього об'єкта DataSet рівним Table1, але DataField цього разу установите рівним Description. Додайте додаткові компоненти TQRDBText для полів Cost, ListPrice, OnHand і OnOrder. Зміните розмір вікна форми, якщо необхідно бачити велику частину розроблювального звіту. На мал. 2.2.12 показаний закінчений розділ Detail.

Якщо ви додержувалися інструкцій, збережете зараз звіт. Його розробку ми продовжимо в наступних підрозділах. Використовуйте імена проекту і роздягнула, задані за замовчуванням, і збережете них у будь-якому тимчасовому каталозі.

У залежності від типу полючи, що друкується в стовпці, можна установити різні значення властивості Alignment. Наприклад, для стовпця Description у прикладі звіту значення Alignment, мабуть, повинне бути встановлено рівним taLeftJustify, тобто вирівнювання по лівому краї. Інші цифрові поля будуть мати кращий вигляд при установці taRightJustify. Вам, може бути, прийдеться небагато повозитися з цими й іншими властивостями, щоб підібрати таке розташування стовпців, яке б вас задовольняло.



Малюнок 2.2.12 – Форма додатка Report1 під час розробки. Кожен об'єкт компонента TQRDBText представляє поле таблиці бази даних, що повинне бути видрукуване в готовому звіті

Роздягнув звіту являє собою об'єкт-контейнер. Це означає, що, видаливши розділ, ви видалите й об'єкти в цьому розділі. Дуже важливо запам'ятати: ніколи не відключайте розділ шляхом установки подсвойства Bands об'єкта QuickRep рівним False, інакше прийдеться переустановлювати всі об'єкти, що ви так старанно вставляли в розділ. З метою безпеки після створення кожного розділу і вставки об'єктів збережете проект.

Вище був створений завершений звіт, якому можна переглядати або друкувати. Щоб у цьому переконатися, клацніть правою кнопкою усередині об'єкта QuickRepl (але не усередині роздягнула Detail). Виберіть команду **Preview** з контекстного меню, щоб переглянути, який вийшов звіт. На мал. 2.2.13 показаний результат.

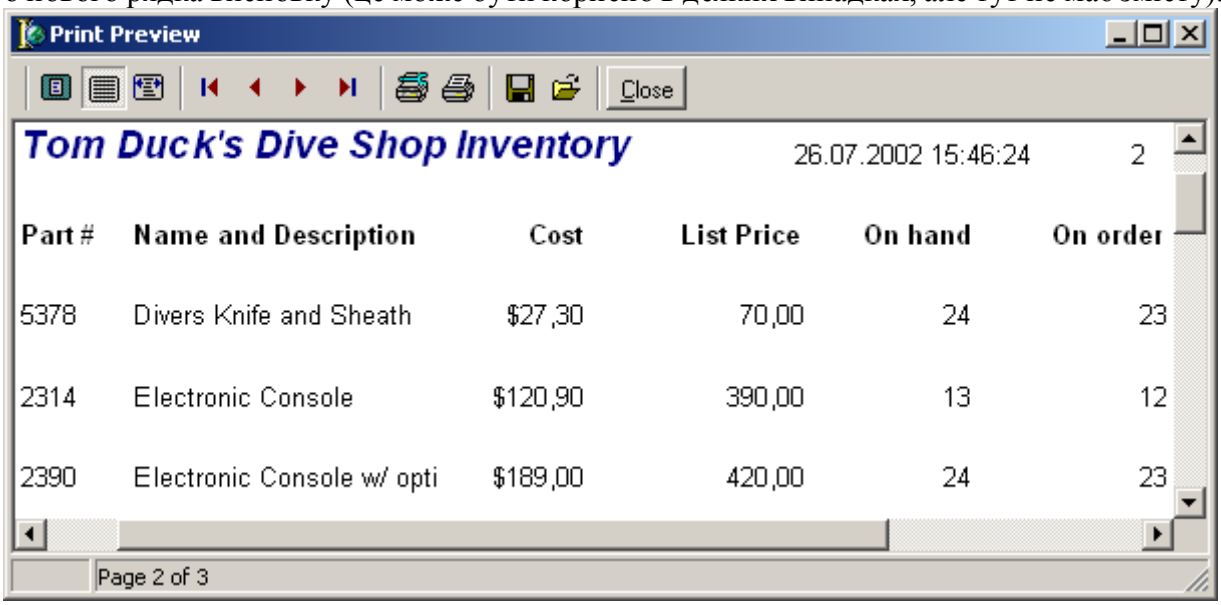
Зараз ви випробували команди попереднього перегляду і печатки усередині Delphi. Таке ж вікно попереднього перегляду побачать і кінцеві користувачі додатка. Далі в цій главі буде описано, як забезпечується подібна можливість у завершеному додатку.

Поэкспериментируйте з кнопками на сторінці попереднього перегляду. Клацніть на кнопці **Zoom**, щоб відобразити повну сторінку (однак текст при цьому навряд чи удасться прочитати). Клацніть на кнопці **Zoom to 100%**, щоб збільшити екранне представлення звіту, що друкується, (ця опція корисна для точної установки позицій об'єктів перед печаткою). Клацніть на кнопках зі стрільцями, щоб пролистать звіт. Клацніть на **Printer setup**, щоб настроїти опції принтера; клацніть на **Print to print** (печатка почнеться негайно; діалогове вікно властивостей печатки не виводиться, тому перевірте, щоб установка принтера була обрана завчасно). Дві інші кнопки, **Save** і **Open**, використовуються для збереження поточного звіту у файлі на диску і перезавантаження раніше збереженого звіту. Клацніть на кнопці **Close**, щоб повернутися до сторінки форми діаграми.

2.2.3.2. Друк заголовків стовпчиків

Дотепер наш звіт виглядав нормально, але стовпці в ньому ідентифіковані, а велика частина даних у заключному висновку безглузда. Кожен стовпець потрібно озаглавити, і краще друкувати з застосуванням різних шрифтів, щоб заголовки стовпців виділялися.

Заголовок стовпця являє собою просто групу об'єктів QRLabel, що можуть відображати будь-який бажаний текст. Ці об'єкти дуже схожі на компоненти Label, але розроблені спеціально для застосування в розділах об'єкта QuickRep. Оскільки в звітах заголовки стовпців повинні друкуватися лише вгорі кожної сторінки, необхідно створити інший розділ звіту, що буде містити заголовки. Якщо найменування помістити в розділ Detail, вони будуть повторюватися для кожного нового рядка висновку (це може бути корисно в деяких випадках, але тут не має змісту).



Малюнок 2.2.13 - За допомогою команди генератора звітів Preview звіт відображається таким, яким він буде з'являтися на видрукуваній сторінці

Якщо ви усе ще знаходитеся у вікні попередньо перегляду, закрийте його і повернетеся до Delphi, а потім виконаєте інструкції, приведені нижче, щоб додати в звіт заголовки стовпців.

1) Виберіть об'єкт QuickRepl (його ім'я повинне з'явитися у вікні Object Inspector). Двічі клацніть на знаку “плюс” (+), розташованому поруч із властивістю Bands, і установите властивість HasColumnHeader рівним True. Буде доданий інший розділ до об'єкта QuickRep1, позначений затіненим заголовком Column Header (цей текст не з'являється в підсумковому звіті). На мал. 2.2.12 показаний звіт з розділом Column Header.
2) Виберіть компонент TQRLabel із вкладки QReport палітри і клацніть у розділі Column Header, щоб уставити заголовок, за замовчуванням називаний QRLabel1. Перетягнете об'єкт QRLabel1 до верхньої границі стовпця, якому необхідно позначити, а потім використовуйте Object Inspector, щоб увести текст заголовка у властивість Caption. Можете ввести будь-як текст, але він не повинний збігатися з ім'ям полючи стовпця. На мал. 2.2.12 показаний завершений розділ Column Header за допомогою об'єктів QRLabel угорі кожного стовпця.
3) Звичайно краще друкувати заголовки стовпців різними шрифтами. Можете вибрати шрифти для кожного окремого компонента QRLabel (і більшості інших), використовуючи їхньої властивості Font. Але щоб змінити шрифт для цілого роздягнула, виберіть розділ Column Header (у списку вікна, що розкривається, Object Inspector відображається ColumnHeaderBand1), а потім клацніть на овалі поруч із властивістю Font і виберіть Bold. Можете вибрати також кольору висновку, імена серії шрифтів, розміри крапок і зробити інші зміни для обраного шрифту і його властивостей.
4) Після присвоєння стовпцям заголовків клацніть усередині об'єкта QuickRep1 (але не усередині одного з розділів) і виберіть команду Preview з контекстного меню. Потім можете переглянути і надрукувати звіт з озаглавленими стовпцями.

Якщо ви точно виконували всі інструкції, збережете зараз додаток. Ми продовжимо розробку цього звіту в наступних розділах.

Для всіх об'єктів властивості шрифтів беруться з батьківського контейнера. Розділи використовують шрифт звіту, об'єкти в розділах - шрифт роздягнула і т.д. Щоб змінити шрифт для цілого звіту, виберіть об'єкт QuickRep і установите його властивість Font. Потім можна модифікувати властивості Font для окремих розділів і об'єктів, що утримуються в них.

2.2.3.3. Друк системної інформації

Іноді на сторінки звіту бажано додати системну інформацію. Це може бути заголовок звіту, що тече дата і час, номер сторінки. Точно так, як ми надходили з полями звіту і заголовками стовпців, для додавання системної інформації потрібно створити інший тип роздягнула, у який помістити компоненти об'єкта QuickReport.

Закрийте вікно попереднього перегляду генератора звітів QuickReport, якщо воно відкрито, і повернетеся в Delphi. Потім виконаєте наступні інструкції, щоб додати в звіт заголовок, дату і час, а також номер сторінки.

1) Виберіть об'єкт QuickRep1 і двічі клацніть на знаку “плюс” (+), розташованому ліворуч від властивості Bands. Установите властивість HasPageHeader рівним True. У звіт додасться третій розділ, як показано на мал. 2.2.12. Як і для інших розділів, затінений текст Page Header відображається тільки під час проектування і не друкується в звіті.
2) У розділі Page Header звичайно використовуються об'єкти QRLabel або QRSysData. Щоб додати заголовок угорі сторінки, виберіть компонент TQRLabel із вкладки QReport палітри і клацніть усередині роздягнула Page Header, щоб створити об'єкт QRLabel7. Установите властивість об'єкта Caption і виберіть розмір шрифту, стиль і колір, використовуючи властивість Font.
3) Виберіть компонент TQRSysData і помістите два його об'єкти в розділ Page Header. Кожний з об'єктів може відображати різні типи заданої системної інформації. Виберіть кожен об'єкт і установите значення властивості Data рівним типові даних, що вам необхідний. У даному випадку обране значення першого об'єкта рівним qrsDateTime, а значення другого об'єкта — рівним qrsPageNumber. Розташування об'єктів може бути довільним. На мал. 2.2.12 показаний звіт із завершеним розділом Page Header.

Якщо ви додержувалися всіх інструкцій, збережете додаток. Ми продовжимо розробку цього звіту в наступних підрозділах.

Як і раніш, клацніть правою кнопкою усередині об'єкта QuickRep1 (але не усередині роздягнула), а потім виберіть команду **Preview**. Остаточний звіт стає красивим, але ще кілька елементів можна додати, щоб його удосконалити.

Замість використання роздягнула Page Header для печатки заголовка звіту можна установити значення подсвойства роздягнула HasTitle рівним True, а потім додати мітку заголовка до розділу Title Band (цей розділ друкується тільки на першій сторінці).

Щоб віддрукувати дані внизу кожної сторінки, установите значення подствойства HasFooter властивості Bands об'єкта QuickRep рівним True, Використовуйте отриманий у результаті роздягнув аналогічно розділові Header Band. Природно, об'єкти, що вставлені в цей розділ, друкуються внизу, а не вгорі кожної сторінки.

2.2.3.4. Стовпчики підсумування

Щоб обчислити підсумкові значення в числових стовпцях і представити інші підсумкові дані, необхідно додати в звіт один додатковий розділ і пару об'єктів. Наприклад, у розроблювальному звіті (див. мал. 2.2.13) можна скласти значення в стовпцях On Hand і On Order. У демонстраційних цілях показується і підсумкове значення для стовпця List Price, хоча це навряд чи має сенс у даному випадку. Однак знати підсумкове значення запасів винного магазина було б дуже корисно, тому воно буде включено в остаточний звіт.

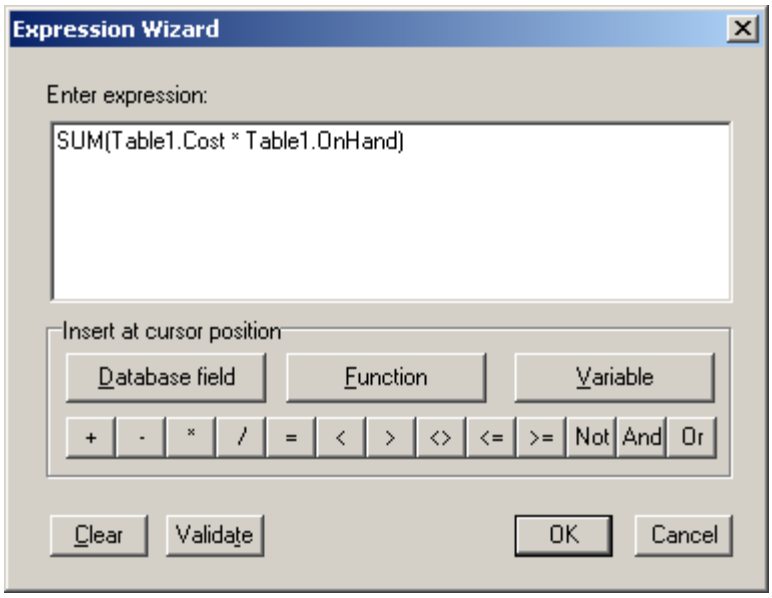
Щоб додати в звіт підсумковий розділ, закрийте діалогове вікно попереднього перегляду генератора звітів QuickReport, якщо воно відкрито, і повернетеся в Delphi. Потім виконаєте наступні інструкції:

1) Виберіть об'єкт QuickRep1 і двічі клацніть на властивості Bands у вікні Object Inspector. Установите значення подствойства HasSummary рівним True. На мал. 2.2.12 новий розділ відображений у нижній частині звіту. Як і у випадку з іншими розділами, затінена напис Summary з'являється тільки в Delphi, але не у видрукуваному підсумковому документі.
2) Роздягнув Summary з'являється наприкінці звіту, після останнього рядка даних. У нього можна додати різні об'єкти генератора звітів QuickReport. У даному випадку нам необхідні об'єкти QRLabel, щоб ідентифікувати сумарні значення, і об'єкти QRExpr, щоб зробити обчислення. (Піктограма QRExpr ясно позначена $E = mc^2$.) Помістите в розділ Summary два об'єкти QRLabel і чотири об'єкти QRExpr. Перетягнете контур розділу Summary, щоб продовжити його, і розташуєте об'єкти, як показано на мал. 2.2.12. Установите властивість Caption об'єкта QRLabel, як показано на малюнку, і використовуйте властивість Font, щоб вибрати колір для висновку на печатку.
3) Щоб обчислити сумарні значення, виберіть кожен об'єкт QRExpr і установите кожну властивість Expression, як описано нижче. Ці вираження відображаються в проектованій формі й у завершеному звіті вони заміняються обчисленими значеннями. Для того щоб послатися на конкретне поле таблиці бази даних, вираження посилається на об'єкт Table1 у формі, використовуючи крапкову нотацію й ім'я полючи (Cost, OnHand і т.д.). Це не програмування мовою Pascal — текст вираження обчислюється генератором звітів QuickReport:
   Sum(Table1.Cost)
   Sum(Table1.OnHand)
   Sum(Table1.OnOrder)
   Sum(Table1.Cost * Table1.OnHand)
4) Крім властивостей Expression об'єкта QRExpr, можна увести властивість Mask, щоб установити конфігурацію значення, що друкується. Те ж саме можна зробити і для об'єктів QRDBText у розділі Detail — за допомогою масок полегшується контроль над вирівнюванням стовпців, вони дозволяють указувати кількість десяткових цифр у числовому значенні. У наступному прикладі показані результати дії масок:

#,###.00	2,609.20
$#,###,###.00	$24,891.05
####	43

5) Після установки властивостей Expression і Mask в об'єктах QRExpr клацніть правою кнопкою миші на об'єкті QuickRep1 (не клацайте усередині роздягнула) і виберіть команду Preview з контекстного меню.

Замість того щоб уводити вираження у властивості Expression об'єкта QRExpr, можна клацнути на властивості, і відкриється вікно построителя виражень **Expression Wizard** (мал. 2.2.14), у більш ранніх версіях Delphi - **Expression Builder**. Щоб використовувати редактор клацніть на кнопці **Function**, виберіть функцію зі списку, що **з'явився, Available** functions і клацніть на **кнопці** Continue у нижній частині діалогового **вікна Expression** Wizard, наприклад виберіть **функцію** SUM. Потім клацніть на кнопку **Database field**, виберіть об'єкт у списку **Select dataset** (у цьому прикладі є лише один об'єкт –Table1) і виберіть поле в списку **Available field**, наприклад Cost. Клацніть на кнопці **OK** у нижній частині діалогового вікна, щоб додати це поле у вираження. При бажанні можна вибрати будь-яке інше доступне поле, таке як OnHand. І нарешті, клацніть на кнопці **OK**, щоб завершити вираження, що показане у верхній частині вікна. Якщо воно вас задовольняє, клацніть на кнопці **OK**, щоб увести вираження у властивість Expression об'єкта QRExpr і повернутися до проектування форми.



Малюнок 2.2.14 - Замість уведення виражень у властивості Expression об'єкта QRExpr можна використовувати засіб Expression Wizard, щоб створювати вираження

У масці символ “ґрати” (#) представляє будь-як цифру, але друкується лише в тому випадку, якщо в цій позиції розташована цифра. Нуль представляє будь-як цифру і завжди друкується, навіть якщо в цій позиції немає цифри, Кома вказує роздільник, якому необхідно використовувати у великих значеннях. Уведення маски відрізняється в залежності від типу полючи даних. Об'єкти DateField, DateTimeField і TimeField форматируются за допомогою виклику функції Delphi DateTimeToStr. Для цих об'єктів, якщо не уведено властивості Mask, висновок відповідає параметрам у файлі ініціалізації Windows Win.ini у розділі [International]. Об'єкти BCDField, CurrencyField і FloatField форматируются за допомогою виклику функції Delph FioatToTextFmt. Для цих компонентів, якщо не уведений формат Mask або Display, значення форматируется за допомогою властивості полючи Currency.

Якщо ви виконували всі інструкції, збережете додаток. Ми продовжимо розробку звіту в наступних підрозділах.

2.2.3.5. Сортування даних звіту

Сортування є операцією бази даних, а не функцією генератора звітів QuickReport. Щоб відсортувати звіт, потрібно просто вибрати індекс для таблиці бази даних і надрукувати звіт. Якщо індекс не створений для поля, по якому необхідна сортування таблиці бази даних, використовуйте для його створення програму **Database Desktop**.

Але що робити, якщо не можна створити новий індекс, оскільки таблиці бази даних доступні тільки для читання або тільки через вилучений сервер? У таких випадках можна друкувати звіт у текстовий файл на диску, а потім відсортувати рядка. Якщо є якісь спеціальні вимоги сортування, можна написати код для її виконання, якщо ж необхідна просто сортування за алфавітом або цифровим сортуванням, можна скористатися стандартною програмою. Завантажите текст у текстовий редактор і використовуйте команду sort програми. Потім можете віддрукувати відсортований текстовий файл. Звичайно це значно легше, ніж намагатися відсортувати реальні записи бази даних.

Щоб відсортувати звіт по винному магазині в нашому прикладі по описах елементів, закрийте вікно попереднього перегляду, якщо воно відкрито, і повернетеся в Delphi. Виконаєте наступні інструкції.

1) Виберіть об'єкт Table1. У вікні Object Inspector (натисніть <F11>, якщо воно не відображено), клацніть на стрільці вниз, розташованої поруч із властивістю IndexName. Будуть відображені доступні індекси полів. Виберіть ByDescription.
2) Клацніть правою кнопкою на об'єкті QuickRep1 (але не усередині роздягнула) і виберіть команду Preview, щоб переглянути і віддрукувати завершений звіт.

Якщо ви дотримували наших інструкцій, збережете додаток. Вам воно знадобиться в наступному розділі.

2.2.4. Друк звітів під година виконання програми

Дотепер ми в Delphi розробляли, переглядали і друкували звіти в повному обсязі. Щоб створити тільки звіт по базі даних для себе, вам не буде потрібно записувати фрагменти коду або завершений додаток. Однак може знадобитися надати можливості генератора звітів для кінцевих користувачів вашого додатка і забезпечити для них можливості попереднього перегляду і печатки, представлені в цій главі.

Нижче описуються дії, необхідні для завершення приклада додатка Report1 і додавання об'єктів Buttons для попереднього перегляду і печатки звіту. Можете завантажити файл проекту Report1.dpr у Delphi або, якщо ви дотепер виконували всі інструкції цієї глави, закрити вікно попереднього перегляду QuickReport (якщо воно відкрито), повернутися в Delphi і виконати наступні інструкції.

- Додайте два об'єкти BitBtn із вкладки Additional палітри в екранну форму й установите їхню властивість Caption рівним Preview... and Print. (Можете додати і третій об'єкт компонента TBitBtn і установити значення bkClose для його властивості Kind, щоб забезпечити користувачам швидкий спосіб виходу з додатка.)
- При бажанні можете завантажити растрове зображення у властивості Glyph для кожної кнопки. Наприклад, зображення Report.bmp і Print.bmp, що поставляються з Delphi у підкаталозі Images\Buttons.
- Двічі клацніть на об'єкті BitBtn1, щоб створити процедуру обробки події OnClick для кнопки Preview. Включите в модуль процедуру, використовуючи приклад 2.2.5 як керівництво. Двічі клацніть на об'єкті BitBtn2, щоб створити другу процедуру обробки події OnClick для кнопки Print. Додайте цю процедуру, також використовуючи приклад 2.2.5 як керівництво.
- Натисніть клавішу <F9>, щоб оттранслировать і виконати програму. Клацніть на кнопці Preview, щоб викликати вікно перегляду генератора звітів. Клацніть на кнопці Print для печатки завершеного звіту (печатка починається відразу ж після щиглика на цій кнопці). З'явиться індикатор ходу роботи на екрані під час висновку на печатку. Також ви побачите виділені дані в різних компонентах звіту.

Приклад 2.2.5 -. Процедура обробки події OnClick об'єкта BitBtn додатка Reportl

```
{Відповідь на щиглика на кнопці Preview...}
procedure TMainForm.BitBtn1Click(Sender: TObject);
begin
  QuickRep1.Preview;
end;

{Відповідь на щиглика на кнопці Print}
procedure TMainForm.BitBtn2Click(Sender: TObject);
begin
  QuickRep1.Print;
end;

end.
```

2.2.5. Корисні поради

Щоб попередньо переглянути і роздрукувати діаграму при використанні редактора **TeeChart**, клацніть спочатку на вкладках **Chart** і **General**, а потім — на кнопці **Print**

Preview. Можете також клацнути на кнопці **Export**, щоб вивести діаграму в метафайл або файл растрового зображення. Під час проектування значно легше використовувати ці кнопки, чим виходити з редактора **TeeChart** і клацати усередині об'єкта діаграми для її попереднього перегляду.

Щоб увести фіксовані мітки осей діаграми, такі як у додатку MayTemp у цій главі, відкрийте редактор **TeeChart**, виберіть вкладку **Chart** і клацніть на вкладці **Axis**. На цій сторінці мається й інший набір вкладок. Виберіть вкладку **Scales** і зніміть прапорець **Automatic**. Потім клацніть на кожній із двох кнопок **Change** і введіть максимальне і мінімальне значення, щоб відобразити них на осях, обраних ліворуч. (Переконаєтеся, що прапорець **Visible** установлений, інакше ваші зміни не будуть відображатися в завершеній діаграмі.) Якщо потрібно установити тільки максимальне або мінімальне значення, клацніть на кнопці Change для такого значення й установите прапорець **Auto** для інших.

Не встановлюйте знову прапорець **Automatic**, інакше вам доведеться зняти його і повторно ввести максимальне і мінімальне значення шкали.

Щоб запобігти печатка в звіті одного об'єкта поверх іншого, установите значення їхніх властивостей AutoSize рівними True. Також перетягнете окремі об'єкти (QRDBText, QRExpr, QRLabel і ін.), щоб забезпечити їхнє належне розташування. Використовуйте властивість Mask, щоб установити конфігурацію форматів висновку.

У прикладі додатка Report 1 цієї глави маються команди Preview і Print. Однак, оскільки в генераторі звітів вікно попереднього перегляду має свою власну кнопку **Print**, може відпасти потреба в додаванні команди Print у вашому додатку — виклик методу Preview надає всі можливості перегляду і печатки.

Щоб друкувати у фоновому режимі, замість методу Print викличте метод PrintBackground. У такому випадку користувачі зможуть продовжувати використовувати додаток під час печатки великих звітів.

Для додавання діаграми в звіт, що друкується, виберіть об'єкт QRChart з палітри і вставте цей об'єкт у розділ звіту. Хоча компонент TQRChart знаходиться на вкладці QReport палітри, це об'єкт TeeChart; крім того, він описаний у файлі оперативної довідки TeeChart.

2.2.6. Резюме

Програма TeeChart дозволяє створювати різні типи діаграм — кругові, стовпчасті, тривимірні і т.п. Об'єкт діаграми може мати один або більш наборів об'єктів, що можуть бути того самого або різних типів.

Клацніть правою кнопкою миші усередині об'єкта Chart і виберіть команду Print Preview, щоб переглядати і друкувати діаграми усередині Delphi.

Щоб забезпечити можливість попереднього перегляду і печатки діаграм під час виконання, додайте ім'я модуля TeePrevi у директиву опису використаних модулів і викличте процедуру ChartPreview, як було показано в прикладі додатка цієї глави WinesChart.

QuickReport є генератором звітів по розділах. Для створення звіту досить помістити об'єкт компонента TQuickRep в екранну форму. Для створення розділів використовується властивість Bands. Щоб запрограмувати звіт, необхідно помістити в розділи окремі об'єкти. Навіть для створення складних типів звітів майже не потрібно вводити програмний код вручну.

Література [1].

РОЗДІЛ 3 СТВОРЕННЯ НЕЗАЛЕЖНОГО ПРОГРАМНОГО КОМПЛЕКСУ

Тема 3.1. Довідкова система додатку (2 години)

Лекція 8

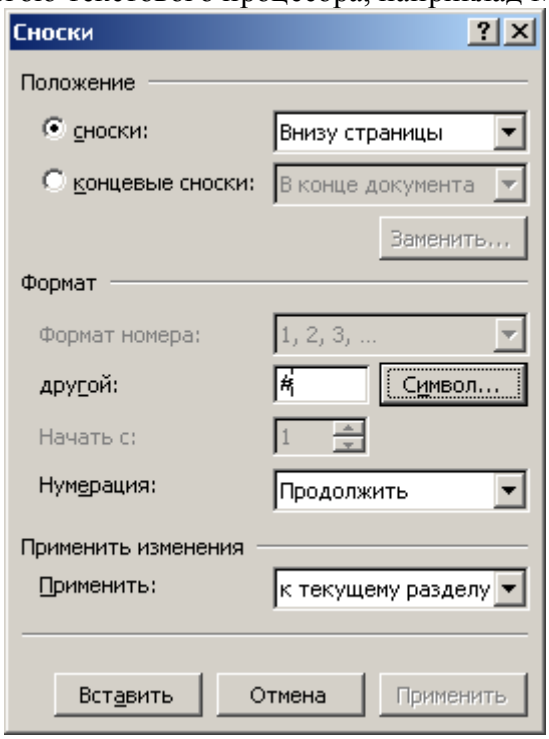
Бажано, щоб кожна програма мала довідкову систему. Фізично довідкова система являє собою набір файлів визначеної структури, використовуючи які програма Winhelp, що є складовою частиною Windows, виводить довідкову інформацію в стандартному виді.

Процес створення довідкової системи можна представити як послідовність наступних кроків:

- Створення файлу документа довідкової системи.
- Створення файлу довідкової системи.

3.1.1 Створення файлів документів довідкової системи

Файл документа довідкової системи являє собою rtf-файл, якому можна створити за допомогою текстового процесора, наприклад Microsoft Word.



Малюнок 3.1.1 - Діалогове вікно Виноски

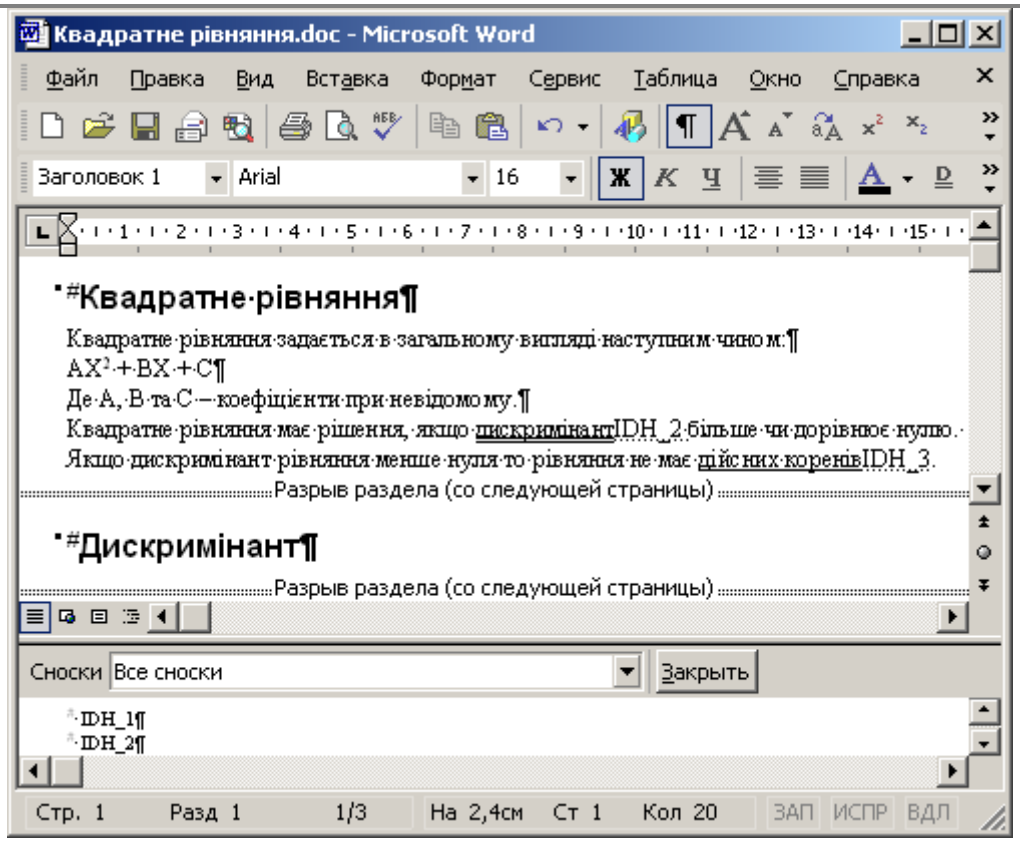
Спочатку варто набрати текст розділів довідки, оформивши заголовки розділів одним зі стилів “Заголовок”, наприклад “Заголовок 1”. При цьому текст кожного розділу довідки повинний знаходитися на окремій сторінці документа (закінчуватися символом “розрив сторінки”).

Після набору тексту випливає за допомогою виносок позначити заголовки розділів довідки (виноски використовуються компілятором довідкової системи для перетворення rtf-файлу в hlp-файл, файл довідки). Один заголовок може бути позначений декількома виносками.

Найбільше часто використовувані виноски перераховані в табл. 3.1.1.

Таблиця 3.1.1 - Часто використовувані виноски

Виноска	*Призначення*
#	Задає ідентифікатор розділу довідки, що може використовуватися в інших розділах для переходу до позначеного цією виноскою розділові
$	Задає ім'я розділу, що буде використовуватися для ідентифікації роздягнула довідки в списку пошуку й у списку переглянутих тим під час використання довідкової системи
K	Задає список ключових слів, при виборі яких зі списку діалогу пошуку здійснюється перехід до розділу довідки, заголовок якої позначений цією виноскою



Малюнок 3.1.2 - Приклад оформлення посилання на інший розділ довідки

Для того щоб позначити заголовок розділу довідки, необхідно установити курсор перед першою буквою тексту заголовка роздягнула довідки і з меню **Вставка** вибрати команду **Виноска**. У діалоговому вікні **Виноски, що**[1] відкрилося, (мал. 3.1.1) у групі **Положення** потрібно установити перемикач у положення **виноски:.** Після цього варто натиснути кнопку **Символ…**і в діалоговому вікні, що **з'явилося,** Символ вибрати символ “#”. Після чого потрібно натиснути кнопку **Вставити.**

У результаті в документ буде уставлена виноска # і в нижній частині вікна документа з'явиться поле введення тексту виноски, у якому поруч зі значком виноски варто ввести ідентифікатор розділу довідки, що позначається.

Як ідентифікатор можна використовувати абревіатуру заголовка роздягнула довідки або наскрізний номер роздягнула, поставивши перед ним, наприклад, букви TI (topic identifier).

Бажано, щоб ідентифікатор роздягнула довідки починався з префікса IDH_ (для зручності після приставки IDH_ необхідно вказувати осмислений ідентифікатор, що надалі полегшить роботу зі створення довідкової системи). У цьому випадку під час компіляції rtf-файлу буде перевірена коректність посилань на розділи довідки. Компілятор виведе список

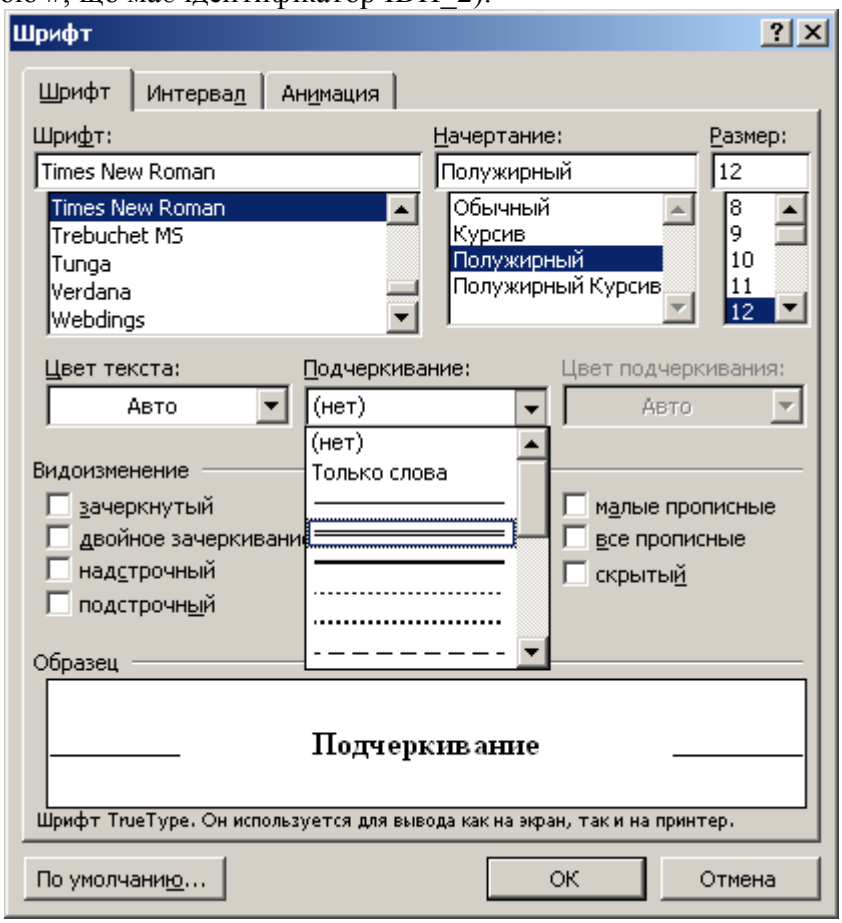
[1] Здесь и далее описание диалоговых окон справедливо для MS Word 2002, в других версиях вид окон может отличаться

ідентифікаторів, що перераховані в розділі [MAP] файлу проекту (див. нижче), але яких немає в rtf-файлі.

Як правило, розділи довідки містять посилання на інші розділи. Поняття (слова), вибір яких викликає перехід до іншого розділові довідки, виділяються відмінним від основного тексту довідки кольором і підкреслюються.

При підготовці тексту довідки слово, вибір якого повинний забезпечити перехід до іншого розділові довідки, варто підкреслити подвійною лінією і відразу за ним, без пробілу, помістити ідентифікатор розділу довідки, до якого повинний бути виконаний перехід. Вставлений ідентифікатор необхідно оформити як схований текст.

На мал. 3.1.2 приведений приклад файлу документа довідкової системи зі словом *"дискримінант"*, позначеним як посилання на інший розділ довідки (тут передбачаються, що роздягнув довідки, у якому знаходяться зведення про дискримінант, позначений виноскою #, що має ідентифікатор IDH_2).



Малюнок 3.1.3 - Діалогове вікно Шрифт

Для того щоб підкреслити фрагмент тексту подвійною лінією, необхідно спочатку виділити цей фрагмент, потім з меню **Формат** вибрати команду **Шрифт** і в діалоговому вікні, що **відкрилося,** Шрифт (мал. 3.1.3) на вкладці **Шрифт** вибрати в списку, що **розкривається, Підкреслення**: подвійну лінію.

Для того щоб фрагмент тексту оформити як схований текст, необхідно спочатку виділити цей фрагмент, а потім на вкладці **Шрифт** у групі **Видозміна** установити прапорець **схований** (див мал. 3.1.3).

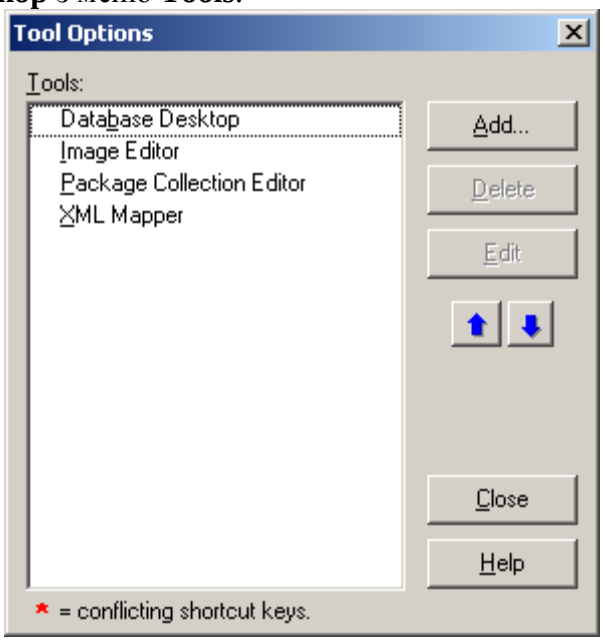
Крім посилання, що забезпечує перехід до іншого розділові довідки, у документ можна вставити посилання на коментар — текст, що з'являється в спливаючому вікні. Під час роботи довідкової системи посилання на коментарі виділяються кольором і підкреслюються пунктирною лінією. При підготовці документа довідкової системи коментарі, як і розділи довідки, розташовують на окремій сторінці, однак текст коментарю не повинний починатися заголовком, оформленим одним зі стилів "Заголовок".

Виноска # повинна бути поставлена перед текстом коментарю. Посилання на коментар оформляються в такий спосіб: спочатку треба підкреслити одинарною лінією слово, вибір якого повинний викликати поява коментарю, потім відразу після цього слова вставити ідентифікатор коментарю й оформити його як схований текст. Словосполучення "*дійсних коренів*" оформлене як посилання на коментар на мал. 3.1.2

3.1.2. Створення файлові довідкової системи

Після того як створений файл документа довідкової системи (rtf-файл), можна приступити безпосередньо до створення довідкової системи. Для цього зручно скористатися програмою **Microsoft Help Workshop**, що поставляється разом з Delphi і знаходиться у файлі Hcw.exe, шлях до якого: C:\Program Files\Borland\Delphi6\Help\Tools.

Запустити Microsoft Help Workshop можна з Windows або, вибравши команду **Help Workshop** з меню **Tools**.



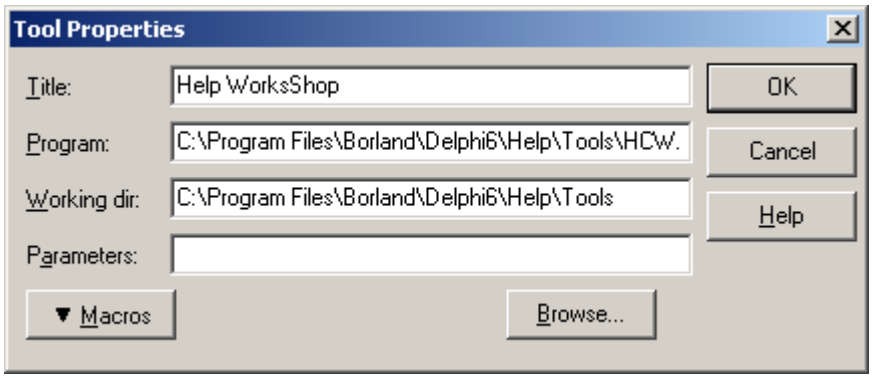
Малюнок 3.1.4 - Діалогове вікно Tool Options

Якщо в меню **Tools** немає команди **Help Workshop**, то треба вибрати команду **Configure Tools…**,потім у діалоговому вікні, що **відкрився, Tool** Options (рис3.1.4) натиснути кнопку **Add** і в наступному діалоговому вікні **Tool Properties** (мал. 3.1.5) ввести в поле **Title:** назва програми — "Help Workshop", а в поле **Program:** — повне ім'я файлу програми, що виконується, Microsoft Help Workshop, тобто з указівкою шляху до йому. Для пошуку імені файлу можна скористатися кнопкою **Browse…**...

Після натискання кнопки **ОК** стає доступним діалогове вікно **Tool Options**, у списку якого з'являється рядок *Help Workshop*. Процес настроювання меню **Tools** закінчується натисканням кнопки **Close**.

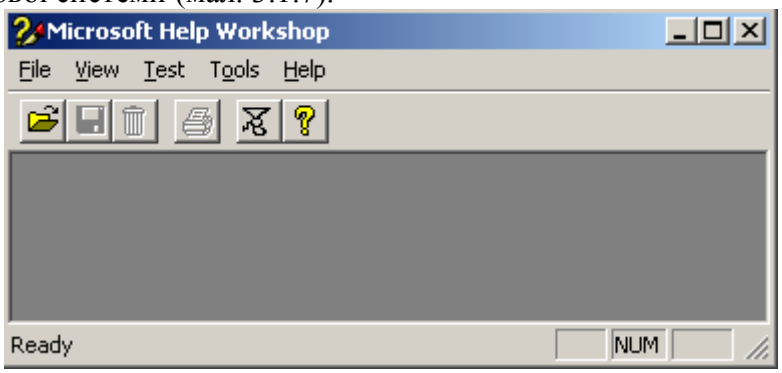
Після запуску програми Help Workshop на екрані з'явиться головне вікно програми (мал. 3.1.6).

Для того щоб приступити до створення нової довідкової системи, необхідно з меню **File** вибрати команду **New**, а потім у діалоговому вікні, що відкрилося, вибрати тип створюваного файлу **- Help** Project. У результаті цих дій відкривається вікно **Project File**.

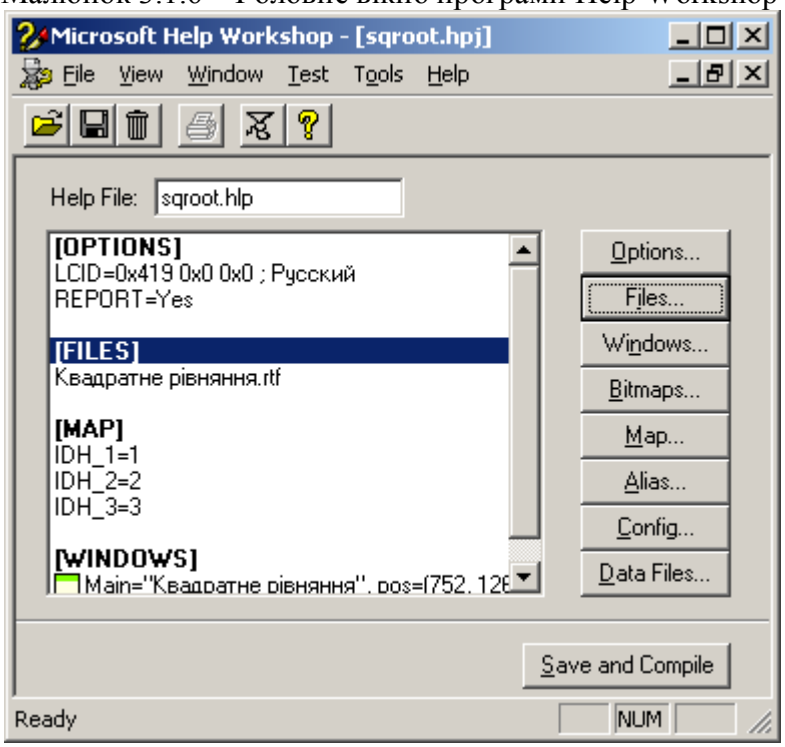


Малюнок 3.1.5 - Діалогове вікно Tool Properties

У цьому вікні спочатку треба вибрати папку, де знаходиться програма, для якої створюється довідкова система, і де вже повинний знаходитися файл документа довідкової системи (rtf-файл). Потім у поле **Ім'я файлу:** варто ввести ім'я файлу проекту довідкової системи і натиснути кнопку **Зберегти**. У результаті відкривається діалогове вікно проекту довідкової системи (мал. 3.1.7).



Малюнок 3.1.6 – Головне вікно програми Help Workshop



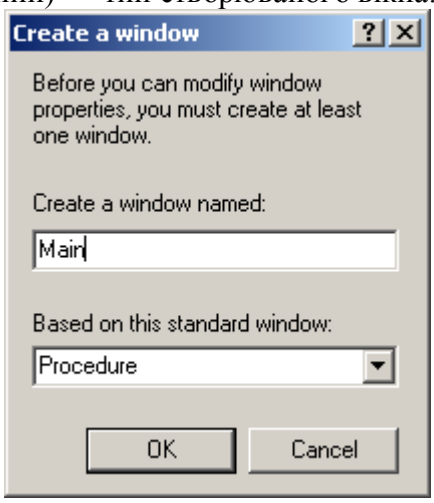
Малюнок 3.1.7 - Вікно проекту довідкової системи

Вікно проекту довідкової системи надає можливість додати необхідні компоненти в проект, задати характеристики вікна довідкової системи, виконати компіляцію проекту і спробний запуск створеної довідкової системи.

Додавання до проекту файлу документа довідкової системи (rtf-файлу). Для цього необхідно натиснути кнопку **Files...**, потім у діалоговому вікні, що **відкрилося, Topic**

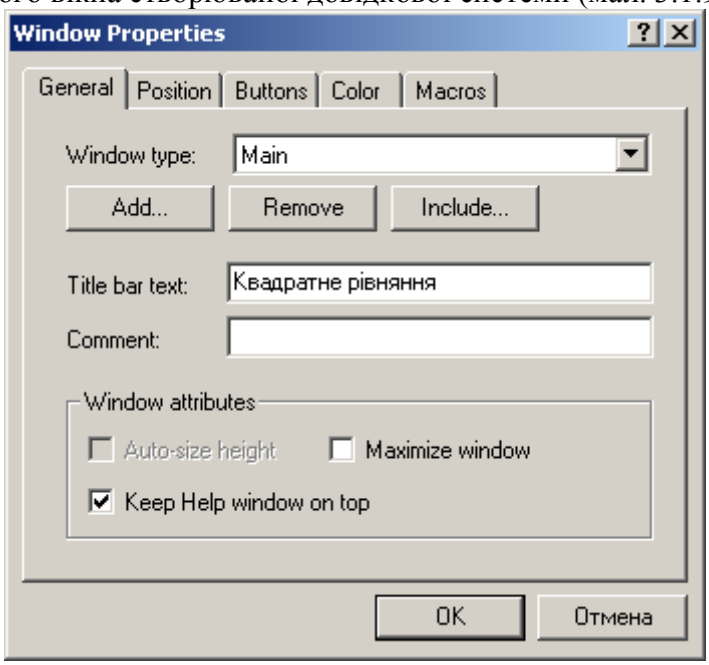
Files - **кнопку…** Add... У результаті відкриється стандартне вікно **Відкрити**, у якому варто вибрати потрібний rtf-файл.

Завдання характеристик вікна довідкової системи. Для цього необхідно натиснути кнопку **Windows…**(см. мал. 3.1.7). У результаті відкривається діалогове вікно **Create a window** (мал. 3.1.8), де в поле **Create a window named:** варто ввести слово “main” (основний) — тип створюваного вікна.



Малюнок 3.1.8 - Діалогове вікно Create a window

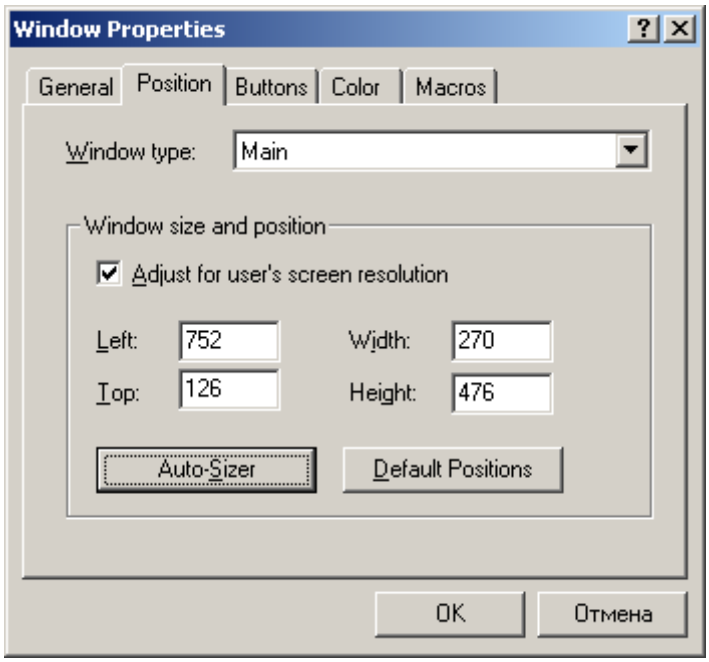
Натискання кнопки **ОК** здійснює перехід у діалогове вікно **Window Properties**. На вкладці **General** цього діалогового вікна в поле **Title bar text:** варто ввести заголовок головного вікна створюваної довідкової системи (мал. 3.1.9).



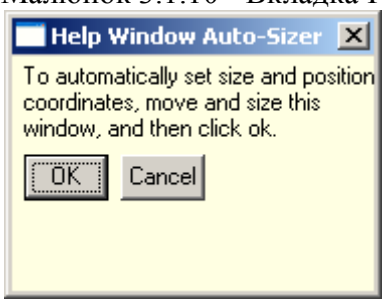
Малюнок 3.1.9 - Вкладка General діалогового вікна Window Properties

На вкладці **Position** діалогового вікна **Window Properties** можна задати положення і розмір вікна довідкової системи (мал. 3.1.10).

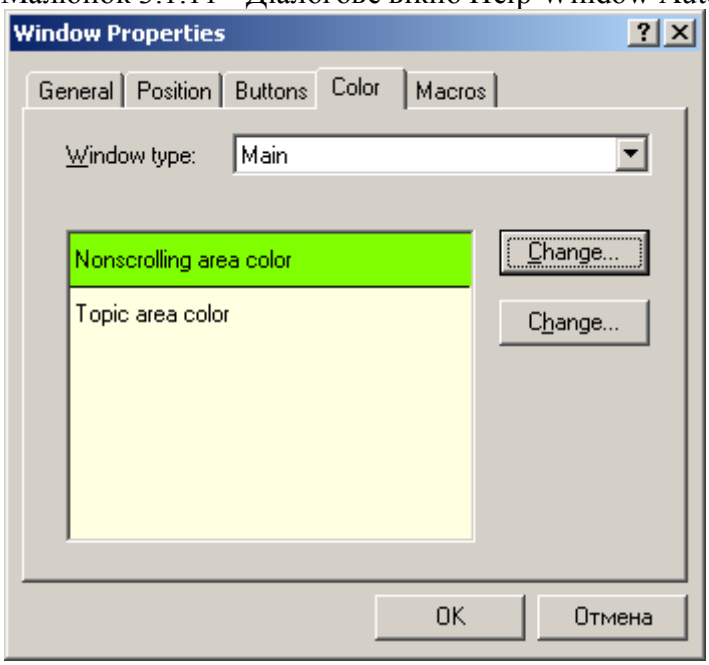
На цій же вкладці знаходиться кнопка **Auto-Sizer**, при натисканні якої відкривається вікно **Help Window Auto-Sizer** (мал. 3.1.11), розмір і положення якого визначаються вмістом полів вкладки **Position**. За допомогою миші можна змінювати розмір і положення цього вікна. Після натискання кнопки **ОК** координати і розмір вікна **Help Window Auto-Sizer** будуть записані в поля вкладки Position.



Малюнок 3.1.10 - Вкладка Position діалогового вікна Window Properties



Малюнок 3.1.11 - Діалогове вікно Help Window Auto-Sizer

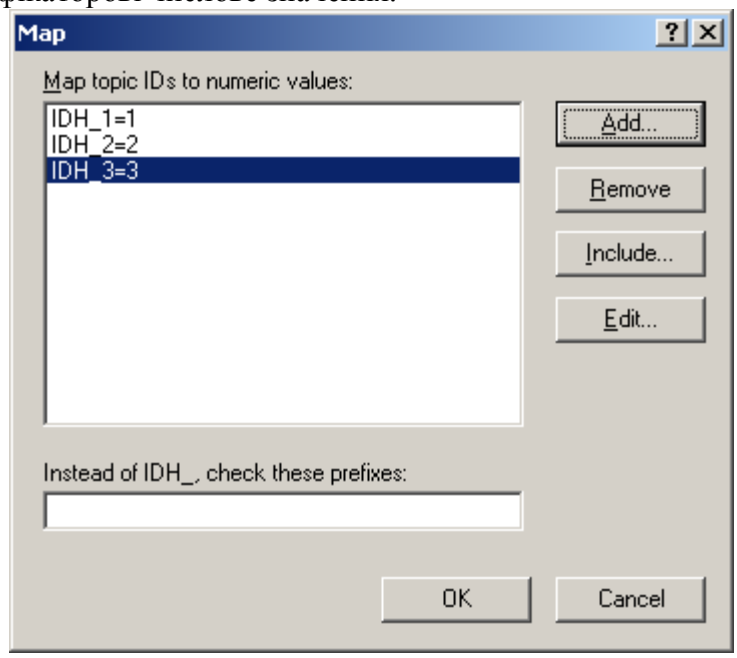


Малюнок 3.1.12 - Вкладка Color діалогового вікна Window Properties

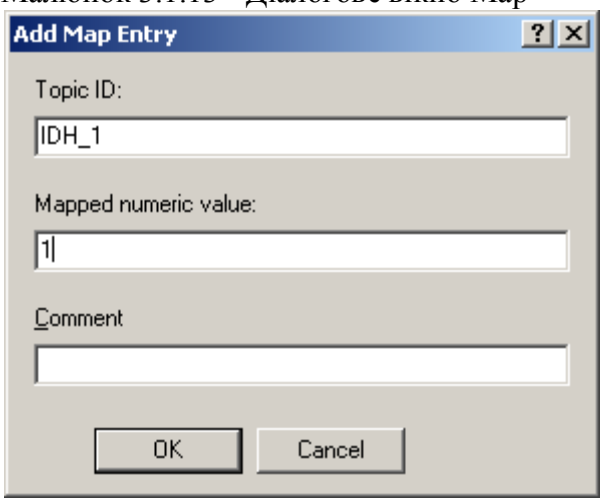
На вкладці **Color** (мал. 3.1.12) можна задати колір тла області заголовка роздягнула довідки (Nonscrolling area color) і області тексту довідки (Topic area color). Для цього треба натиснути відповідну кнопку **Change** і в стандартному вікні **Колір** вибрати потрібний колір.

Призначення числових значень ідентифікаторам розділів довідки. Ці значення будуть використовуватися для пошуку потрібного розділу довідки тією програмою, для якої створюється довідкова система.

Після установки характеристик вікна довідкової системи потрібно визначити ідентифікатори розділів довідкової системи. Для цього треба у вікні проекту довідкової системи натиснути кнопку **Map…**... У результаті цього відкривається діалогове вікно **Map** (мал. 3.1.13). Щоб додати ідентифікатор розділу в список, треба натиснути кнопку **Add…**і в поле **Topic ID:** діалогового вікна, що **відкрилося, Add Map** Entry (мал. 3.1.14) ввести ідентифікатор розділу довідки, а в поле **Mapped numeric value:** — відповідному ідентифікаторові числове значення.



Малюнок 3.1.13 - Діалогове вікно Map



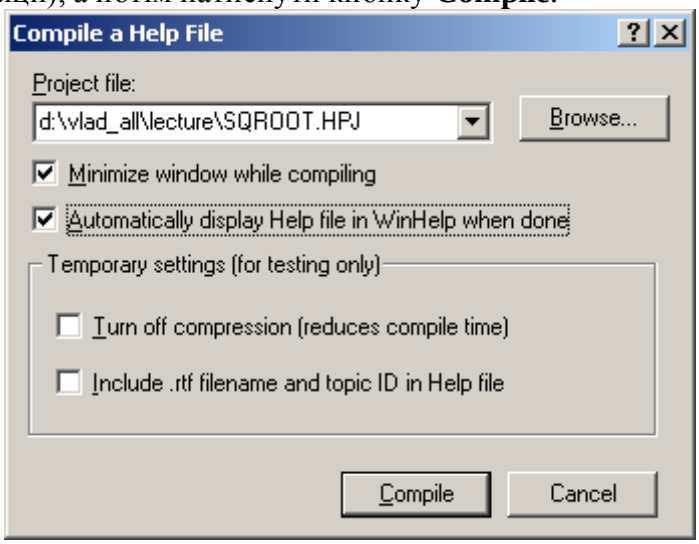
Малюнок 3.1.14 - Діалогове вікно Add Map Entry

На мал. 3.1.7 приведений приклад вікна проекту довідкової системи після додавання rtf-файлу, установки характеристик вікна довідкової системи і призначення числових значень ідентифікаторам розділів.

Перший раз компіляцію проекту довідкової системи краще виконати вибором з меню **File** команди **Compile…**,у результаті виконання якої відкривається діалогове вікно **Compile a Help File** (мал. 3.1.15).

Компіляцію можна виконати, натиснувши кнопку, що **знаходиться** на панелі інструментів, Compile.

У цьому вікні варто установити прапорець **Automatically display Help file in WinHelp when done** (Автоматично показувати створену довідкову систему по завершенні компіляції), а потім натиснути кнопку **Compile**.

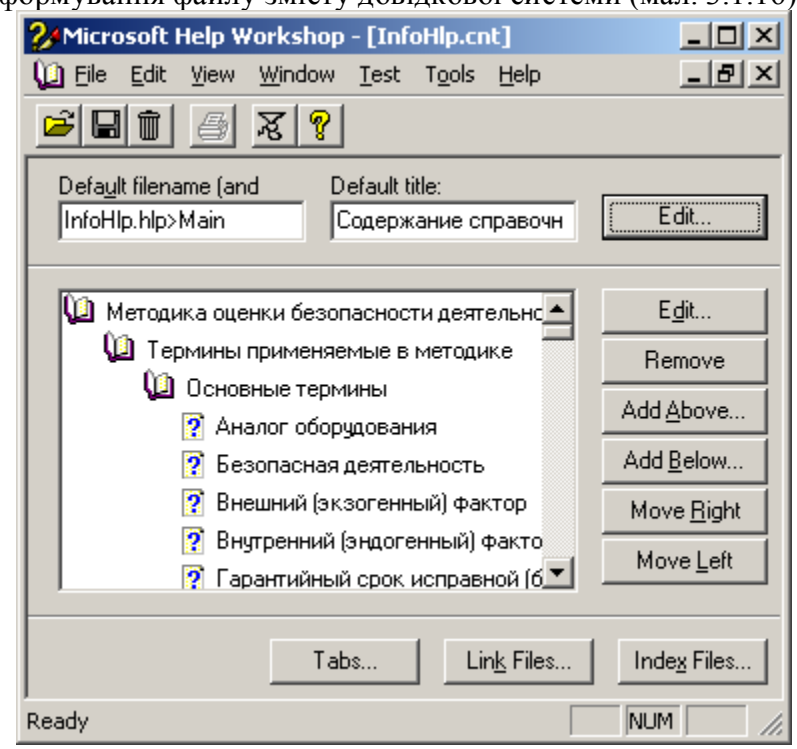


Малюнок 3.1.15 - Діалогове вікно Compile a Help File

По завершенні компіляції на екрані з'являється вікно з інформаційним повідомленням про результати компіляції і, якщо компіляція виконана успішно, — те і вікно створеної довідкової системи.

3.1.3. Створення файлові змісту довідкової системи

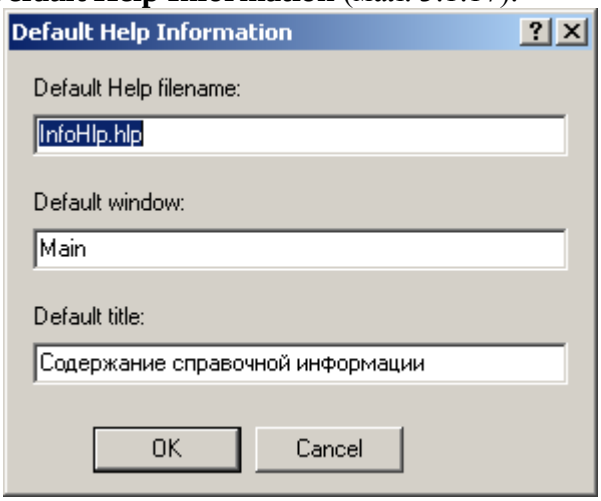
Для створення файлу змісту довідкової системи необхідно запустити програму **Help Workshop** і вибрати пункт меню File⇨New…після чого з'явиться діалогове вікно **New** у якому необхідно вибрати **Help Contents** і натиснути клавішу **OK**. Після цього відкриється вікно формування файлу змісту довідкової системи (мал. 3.1.16).



Малюнок 3.1.16 – Вікно змісту довідкової системи

У верхній частині вікна знаходяться поля введення **Default filename (and window):** і **Default title:**. Для заповнення цих полів краще скористатися кнопкою **Edit…**,розташованої

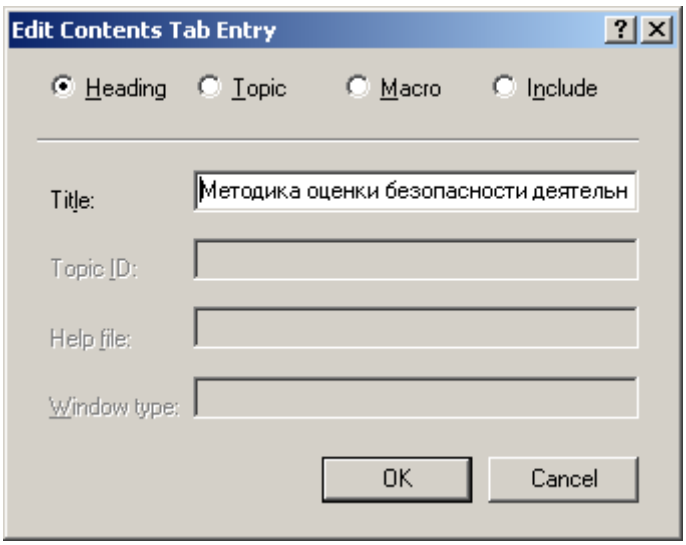
праворуч від поля **Default title:**. Після натискання кнопки **Edit…**відкриється діалогове вікно **Default Help Information** (мал. 3.1.17).



Малюнок 3.1.17 – Діалогове вікно Default Help Information

У поле **Default Help filename:** необхідно ввести ім'я файлу довідки для якого створюється зміст, у даному прикладі це файл InfoHlp.hlp (при цьому файл довідки *.hlp повинний знаходиться в тім же каталозі, де буде знаходиться файл змісту *.cnt). У поле **Default window:** діалогового вікна **Default Help Information** необхідно вказати тип вікна довідкової системи, створеного раніше при створенні файлу проекту довідкової системи, у якому за замовчуванням будуть виводиться статті (теми) довідки. У нашому прикладі це вікно Main. У поле **Default title:** необхідно ввести текст, що буде відображатися в заголовку вікна довідкової системи. Після цього потрібно натиснути кнопку **OK**.

Для створення розділів довідкової системи варто натиснути кнопку **Add Above…**(додати вище) після чого відкриється діалогове вікно **Edit Contents Tab Entry**. У верхній частині діалогового вікна **Edit Contents Tab Entry** варто установити перемикач у положення **Heading** (заголовок), після чого стане доступним для введення тільки одне поле **Title:** (мал. 3.1.18) у якому необхідно ввести заголовок створюваного розділу довідкової системи і потім натиснути кнопку **OK**. Уведений заголовок відобразиться у вікні(мал. 3.1.16).

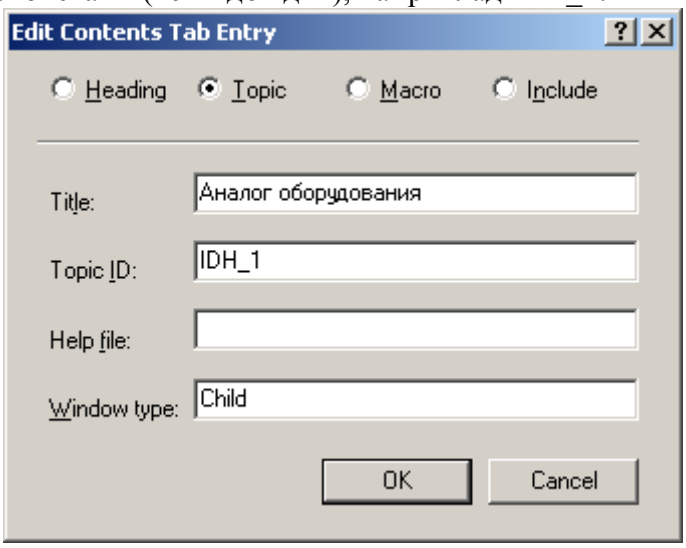


Малюнок 3.1.18 – Створення заголовка роздягнула довідкової системи

Аналогічно вводяться інші заголовки розділів довідкової системи.

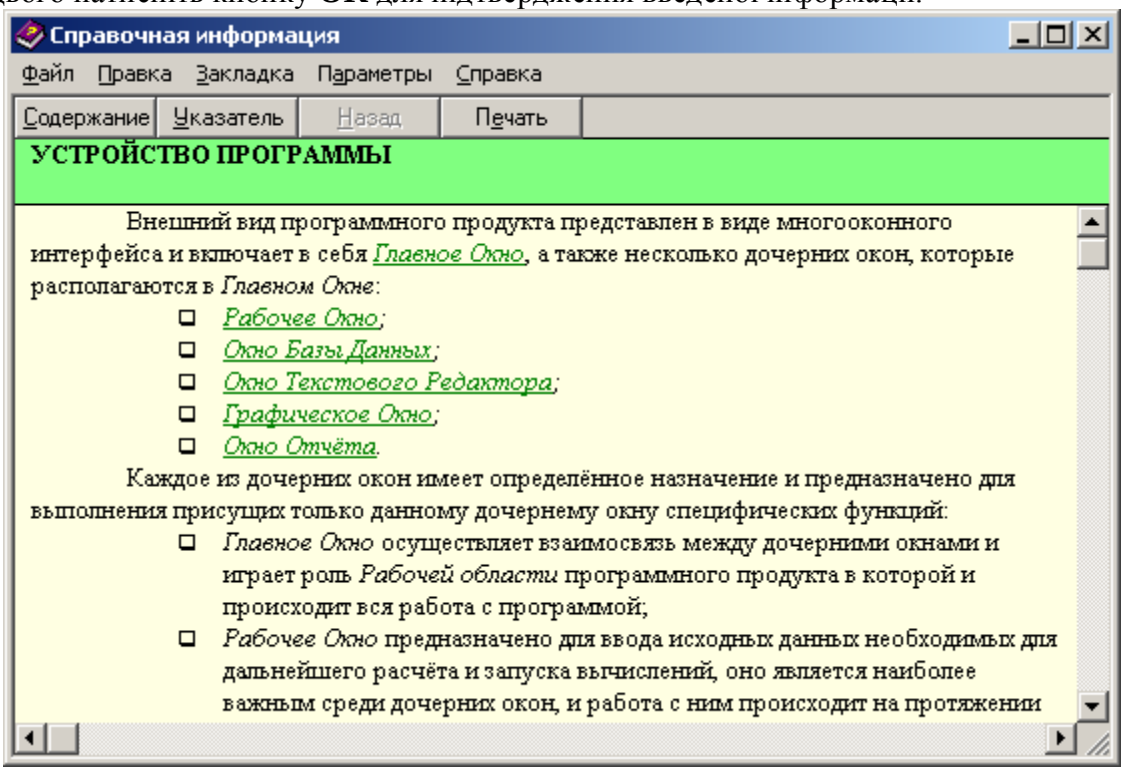
Для того щоб помістити довідкову статтю (тему) у необхідний раздел довідки варто виділити потрібний заголовок у вікні ліворуч і натиснути кнопку **Add Above…**(додати вище) або **Add Below…**(додати нижче) у залежності від того де ви хочете вставити довідкову статтю (тему) – вище або нижче виділеного заголовка розділу. При цьому знову з'явиться діалогове вікно **Edit Contents Tab Entry** у верхній частині якого перемикач за

замовчуванням встановлений у положення **Topic** (тема) (мал. 3.1.19). У поле **Title:** діалогового вікна **Edit Contents Tab Entry** необхідно ввести текст, що буде ім'ям даної довідкової статті (теми довідки). У поле **Topic ID:** варто ввести ідентифікатор даної довідкової статті (теми довідки), наприклад IDH_1.



Малюнок 3.1.19 – Створення роздягнула довідкової системи

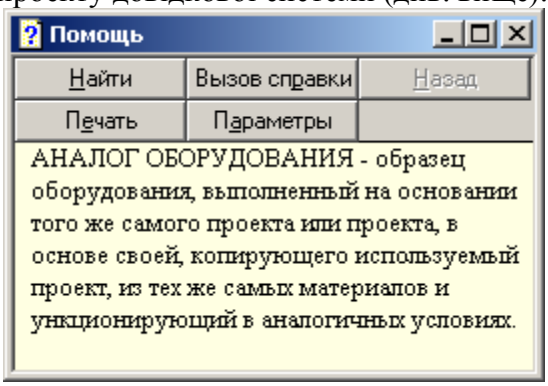
У поле **Window type:** діалогового вікна **Edit Contents Tab Entry** потрібно ввести ідентифікатор вікна довідкової системи, у якому буде відображатися дана тема довідки. Після цього натисніть кнопку **OK** для підтвердження введеної інформації.



Малюнок 3.1.20 – Вікно Main довідкової системи для відображення великих обсягів довідкової інформації

Типів вікон довідкової системи може бути кілька. Наприклад, вікно довідки типу Main (мал. 3.1.20) може мати великі розміри, включати різні кнопки, а також головне меню для зручності використання довідкової системи і призначатися для висновку на екран великих довідкових статей. Вікно довідки типу Child може мати невеликий розмір не містити головного меню і включати необхідний мінімум кнопок. Це вікно можна використовувати для висновку на екран невеликих довідкових повідомлень (мал. 3.1.21). Вікно Child створюється аналогічно як і вікно Main шляхом завдання відповідних значень

полям і вибору необхідних опцій у діалоговому вікні **Window Properties** при створенні файлу проекту довідкової системи (див. вище).



Малюнок 3.1.21 – Вікно Child довідкової системи для висновку невеликих довідкових повідомлень

Після завдання всіх тим довідкової системи (довідкових статей) і створення заголовків розділів для організації ієрархічної (багаторівневої) довідкової системи варто скористатися кнопками **Move Right…**(змістити вправо) і **Move Left…**(змістити вліво) (див. мал. 3.1.16). Для створення підрозділу (зсуву на рівень нижче в ієрархії) необхідно виділити необхідний рядок в області відображення структури довідкової системи і натиснути кнопку **Move Right…**... Для переміщення рядка на рівень вище в ієрархії потрібно натиснути кнопку **Move Left…**... Усі внесені в ієрархії довідкової системи зміни відразу ж стають видні в області відображення структури.

Для внесення змін потрібно виділити потрібний рядок в області відображення структури і натиснути кнопку **Edit…**(редагування), що знаходиться ліворуч (див. мал. 3.1.16). Після цього відкриється діалогове вікно **Edit Contents Tab Entry** у якому необхідно внести необхідні зміни і натиснути кнопку **OK** для підтвердження.

Для видалення рядка необхідно виділити її і натиснути кнопку **Remove** (видалення).

3.1.4. Використання довідкової системи

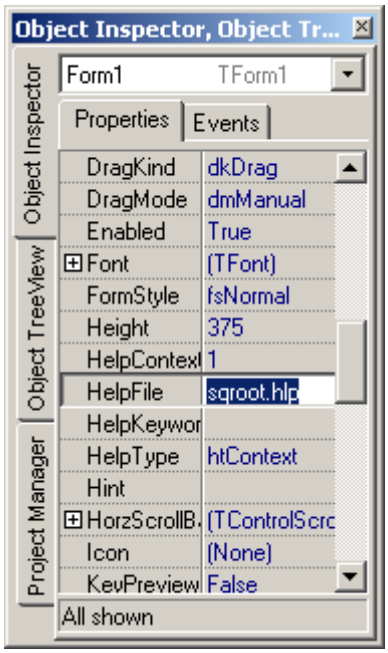
Для того щоб під час роботи програми користувач, натиснувши клавішу <F1>, міг звернутися до довідкової системи, треба щоб властивість головного вікна додатка містилася ім'я файлу довідкової системи, а властивість HelpContext — число, що відповідає номерові того розділу довідкової системи, що повинний бути виведений. У даному прикладі це файл довідки sqroot.hlp (мал. 3.1.22).

Файл довідкової системи додатка краще помістити в ту папку, у якій знаходиться файл програми, що виконується.

Для кожного компонента можна задати свій розділ довідки. Роздягнув довідки, що з'являється, якщо фокус знаходиться на компоненті, і користувач натискає клавішу <F1>, визначається значенням властивості HelpContext цього компонента. Якщо значення властивості HelpContext елемента керування дорівнює нулеві, то при натисканні клавіші <F1> з'являється той розділ довідки, що заданий для форми додатка або батьківського компонента.

Інший спосіб висновку довідки використовується, коли в діалоговому вікні є кнопка **Довідка**. У цьому випадку для даної командної кнопки створюється процедура обробки події OnClick, що звертанням до функції winhelp запускає програму Windows Help (winhlp32.exe). При виклику функції winhelp у якості її параметрів указуються:

- ідентифікатор вікна, що запитує довідкову інформацію;
- ім'я файлу довідкової системи;
- константа, що визначає дію, що повинна виконати програма Windows Help і уточнююча дія параметр.



Малюнок 3.1.22 - Завдання файлу довідкової системи додатка у вікні Object Inspector

Ідентифікатор вікна визначає властивість Handle, що доступно тільки під час роботи програми.

Якщо необхідно вивести розділ довідки, то використовується константа HELP_CONTEXT. У цьому випадку уточнюючий параметр задає розділ довідки, що буде виведений на екран.

Нижче, як приклад, приведена процедура обробки події OnClick для кнопки **Довідка** (BitBtn1) вікна програми. Тут як довідкову систему задається файл довідки myhelp.hlp.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  winhelp(Form1.Handle, 'myhelp.hlp',HELP_CONTEXT,1);
end;
```

Для виклику довідкової системи можна також скористатися наступною конструкцією:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  Application.HelpFile := 'myhelp.hlp';
  Application.HelpCommand(HELP_FINDER, 0);
end;
```

У цьому прикладі використовується кнопка з графічним зображенням розташована у формі. Коли користувач клацає на кнопці, з'являється зміст довідкової системи визначеного в процедурі файлу довідки (у даному випадку це myhelp.hlp).

Література [3].

Тема 3.2. Створення установочної дискети (2 години)

Лекція 9

Сучасні програми поширюються на дискетах або CD-дисках. Процес установки програми, що, як правило, припускає не тільки створення каталогу і перенос у нього виконуваних файлів і файлів даних із проміжного носія, але і настроювання програми, для багатьох користувачів є досить важкою задачею.

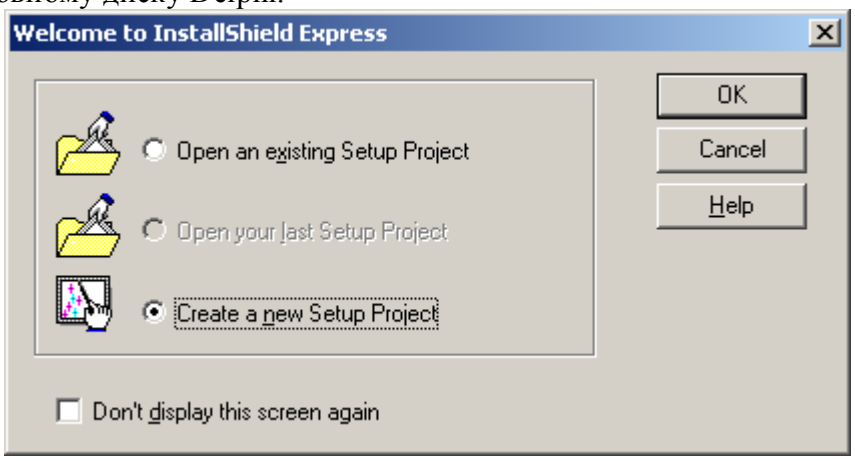
Тому задачу установки прикладної програми на комп'ютері користувача звичайно покладають на спеціальну програму, що знаходиться на тім же диску, що і встановлювана

програма. Таким чином, програміст крім основної задачі повинний розробити програму установки — інсталяційну програму.

Інсталяційна програма може бути створена точно так само, як і будь-яка інша програма. Однак задачі, розв'язувані під час інсталяції, є типовими. Тому були розроблені спеціальні програми, що дозволяють програмістам створювати інсталяційні програми, не написавши ні одного рядка коду.

3.2.1. Програма InstallShield Express

Найбільш популярної серед таких спеціальних програм є програма InstallShield Express. Borland настійно рекомендує використовувати саме цю програму, тому вона є на настановному диску Delphi.



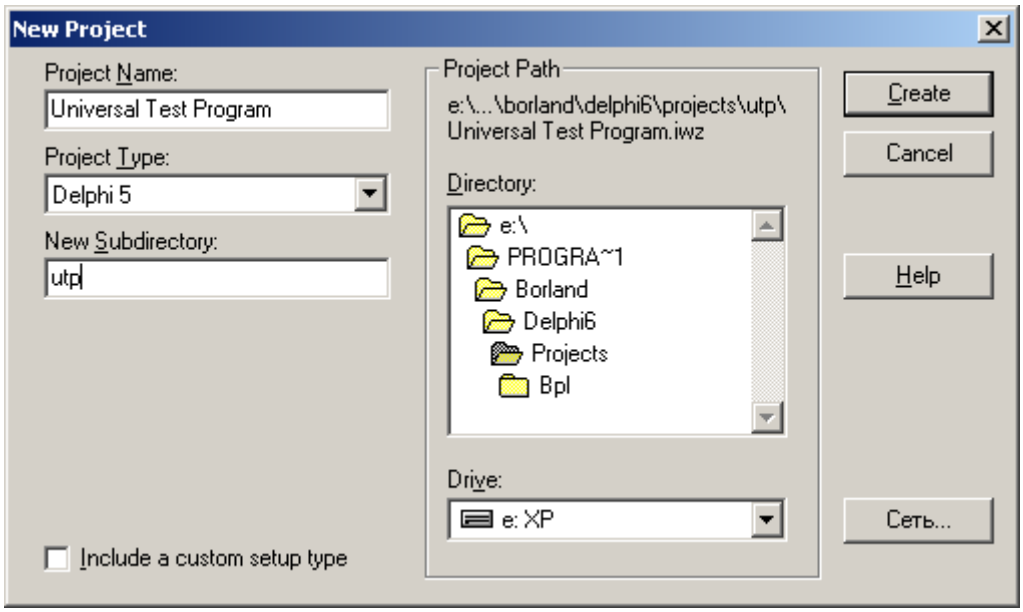
Малюнок 3.2.1 - Діалогове вікно Welcome to InstallShield Express

Процес створення інсталяційної дискети за допомогою InstallShield Express розглянемо на прикладі. Нехай потрібно створити інсталяційну дискету для програми **Універсальна тестирующая програма**.

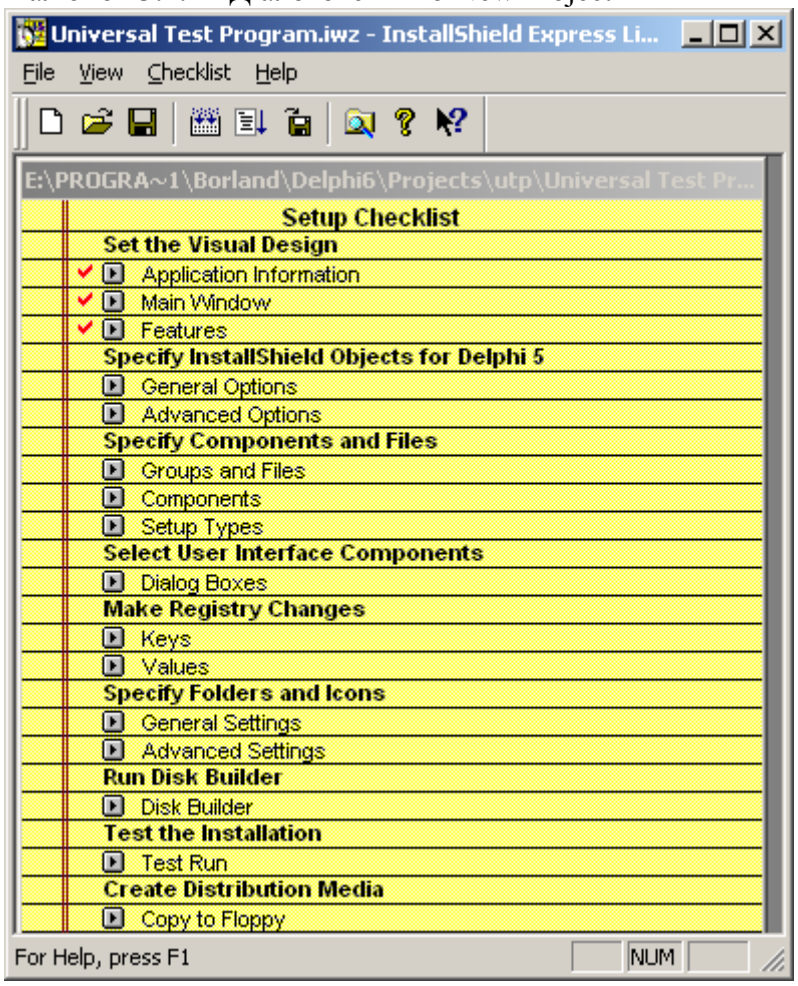
Утиліта InstallShield Express під час установки Delphi автоматично не встановлюється. Для того щоб установити InstallShield Express, потрібно запустити програму установки Delphi (уставити настановний диск у CD-дисковод) і в діалоговому вікні, що **відкрилося, Delphi Setup** Launcher вибрати **команду InstallShield Express Custom Edition for** Delphi. У результаті цих дій запускається майстер установки, що запропонує вибрати каталог, куди буде встановлена утиліта. По завершенні процесу установки в підменю Програми головного меню Windows з'являється команда **Express for Delphi**, вибір якої дозволяє запустити InstallShield Express.

Для того щоб створити інсталяційну дискету, потрібно запустити InstallShield Express, у діалоговому вікні **Welcome to InstallShield Express** вибрати перемикач **Create a new Setup Project** (Створити новий проект) і натиснути кнопку **ОК** (мал. 3.2.1).

У поля діалогового вікна, що **відкрилося, New** Project (мал. 3.2.2) потрібно ввести інформацію про проект: у поле **Project Name:** — ім'я проекту, у поле **New Subdirectory:** — ім'я каталогу, у який програма InstallShield Express буде поміщати всі створювані файли, у тому числі й образ інсталяційної дискети. Використовуючи списки **Directory:** і **Drive:** можна вказати, де повинний бути розміщений каталог проекту.



Малюнок 3.2.2 - Діалогове вікно New Project



Малюнок 3.2.3 - Вікно проекту створення інсталяційної програми

Після натискання кнопки **Create** відкривається вікно проекту створення інсталяційної програми, що містить список команд, що дозволяють настроїти інсталяційну програму, тобто визначити параметри і поводження створюваної програми установки (мал. 3.2.3).

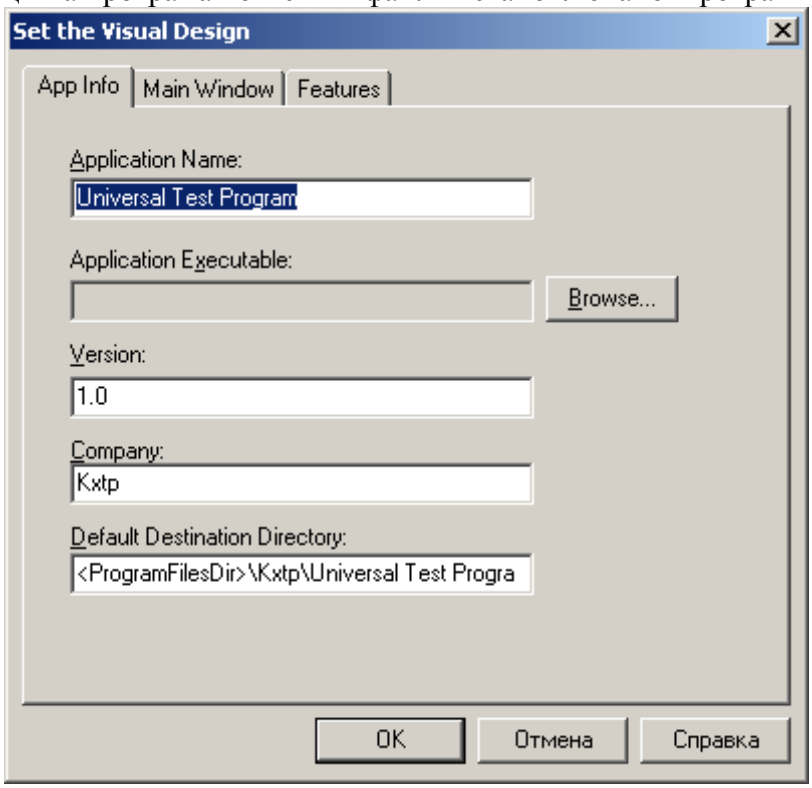
3.2.2. Настроювання програми установки

Команди настроювання настановної програми об'єднані в групи (заголовки груп виділені напівжирним шрифтом). Для вибору команди досить клацнути в рядку команди або на кнопці зі стрілкою. У результаті відкривається діалогове вікно групи.

3.2.2.1. Загальна інформація

Команди групи **Set the Visual Design** дозволяють задати загальну інформацію про встановлюваний додаток і визначити вид головного вікна програми установки.

При виборі команди **Application Information** відкривається вкладка **App Info** діалогового вікна **Set the Visual Design** (мал. 3.2.4), на якій можна задати ім'я додатка, ім'я виконуваного файлу й ім'я каталогу, що буде створений під час установки й у який інсталяційна програма помістить файли встановлюваної програми.



Малюнок 3.2.4 - Вкладка App Info діалогового вікна Set the Visual Design

Варто звернути увагу, що деякі полючи вкладки вже заповнені. Зокрема, програма InstallShield Express як ім'я додатка (**Application Name:**) і імені каталогу додатка (**Default Destination Directory:**) пропонує використовувати ім'я проекту (очевидно, що програміст може задати інші імена).

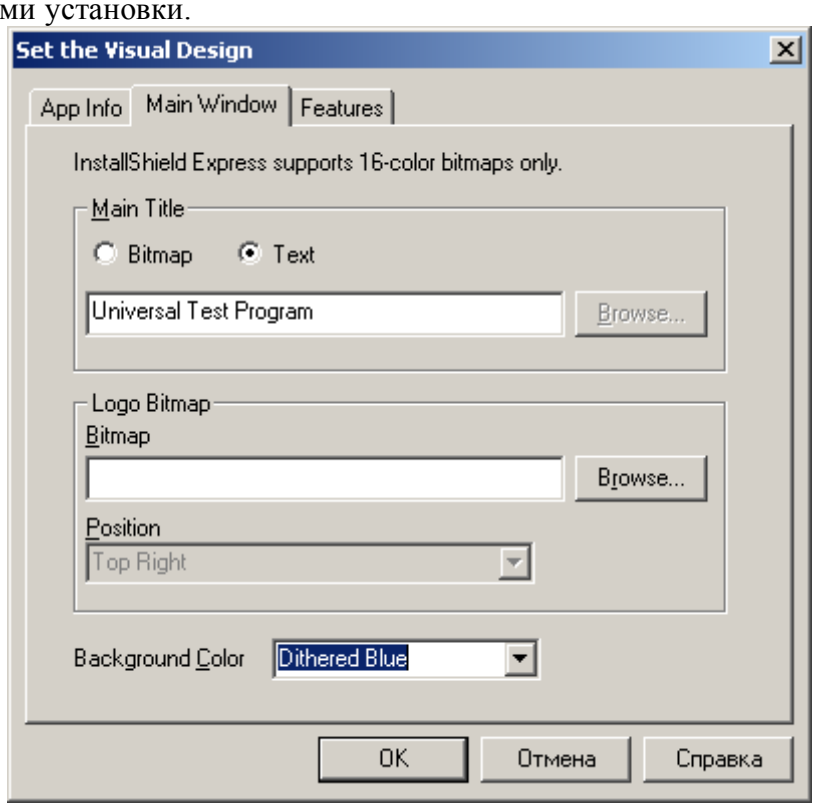
Варто звернути увагу на поле **Default Destination Directory:** (визначений каталог). У цьому полі зазначене ім'я каталогу встановлюваного додатка і його положення. Визначеним каталогом, у який програма установки помістить каталог установлюваної програми, є каталог програм (Program Files). Оскільки під час створення інсталяційної програми не можна знати, як на комп'ютері користувача буде називатися каталог програм і на якому диску він буде знаходитися, те замість імені каталогу використовується його позначення (<ProgramFilesDir>). Під час установки додатка на комп'ютер користувача інсталяційна програма одержує з реєстру Windows ім'я каталогу програм (у більшості випадків це C:\Program Files) і замінить рядок на ім'я цього каталогу.

Позначення каталогів Windows, використовувані в програмі InstallShield Express, приведені в табл. 3.2.1.

Таблиця 3.2.1 - Позначення каталогів Windows, використовувані програмою InstallShield Express

Позначення	*Каталог*
<WINDIR>	Каталог Windows, наприклад C:\Windows
<WINSYSDIR>	Системний каталог Windows, наприклад C:\Windows\System
<ProgramFilesDir>	Каталог програм, наприклад C:\Program Files
<INSTALLDIR>	Каталог, у який інсталяційна програма встановлює додаток

Вкладка Main Window (мал. 3.2.5) дозволяє задати характеристики головного вікна програми установки.



Малюнок 3.2.5 - Вкладка Main Window діалогового вікна Set the Visual Design

Група **Main Title** дозволяє задати тип заголовка головного вікна програми установки. Якщо обрано перемикач **Text**, заголовок буде являти собою текст. Якщо обрано перемикач **Bitmap**, то як заголовок буде використовуватися 16-кольорова ілюстрація, файл якої можна задати за допомогою кнопки **Browse……**.

Група **Logo Bitmap** дозволяє задати емблему (логотип) розроблювача програми. У поле **Bitmap** потрібно ввести ім'я файлу, у якому знаходиться логотип. Використовуючи список, що **розкривається,** Position, можна задати положення логотипа: у центрі, у лівій або в правій верхній частині вікна.

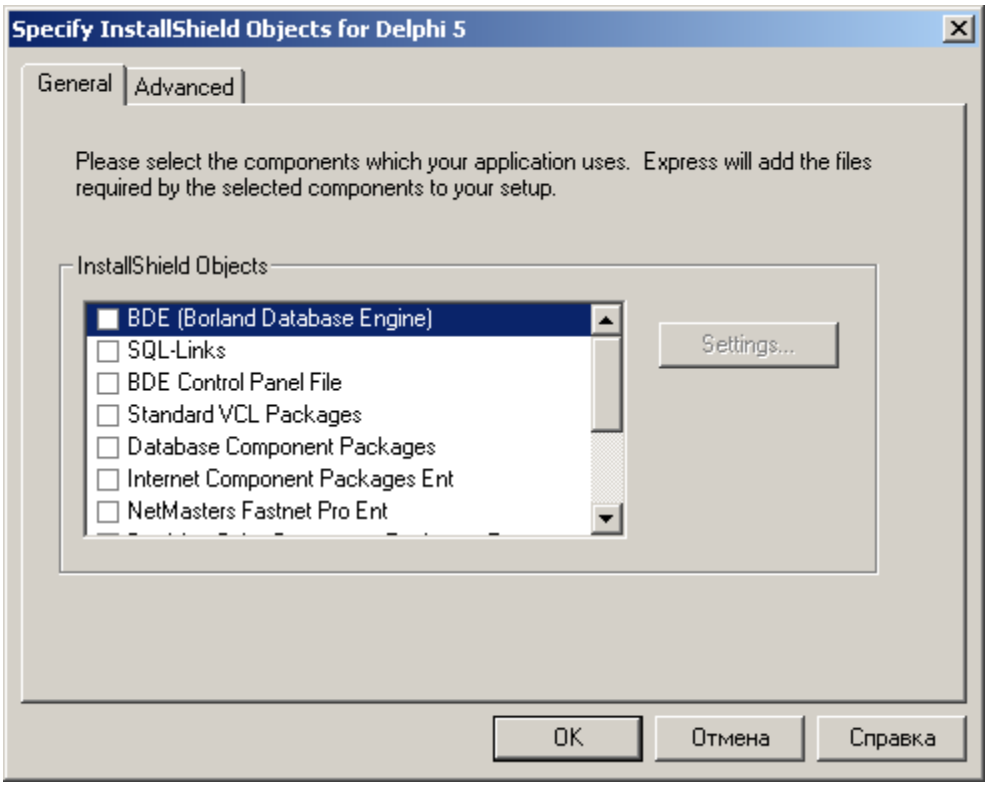
У списку, що **розкривається, Background** Color можна вибрати колір тла головного вікна інсталяційної програми.

Натискання кнопки **ОК** закриває діалогове вікно групи **Set the Visual Design**, а програма InstallShield Express позначає галочкою команди, що програміст використовував для настроювання (мал. 3.2.3).

3.2.2.2. Вибір об’єктів Delphi для установочної дискети

Група команд **Specify InstallShield Objects for Delphi** дозволяє задати, які об'єкти Delphi, наприклад динамічні бібліотеки або пакети компонентів, повинні бути поміщені на комп'ютер користувача і, отже, на настановну дискету (найпростішим програмам ніякі об'єкти Delphi не потрібні, усе необхідне для їхньої роботи знаходиться у виконуваному файлі програми, в exe-файле).

Об'єкти, які потрібно помістити на настановну дискету, вибираються в списку **InstallShield Objects** діалогового вікна **Specify InstallShield Objects for Delphi** (мал. 3.2.6).



Малюнок 3.2.6 - На вкладці General діалогового вікна Select InstallShield Objects for Delphi можна вказати об'єкти, які необхідно помістити на настановну дискету

3.2.2.3. Файли та компоненти які потрібно включати

Команди групи **Specify Components and Files** дозволяють задати файли, що повинні бути перенесені на комп'ютер користувача і, отже, на настановну дискету.

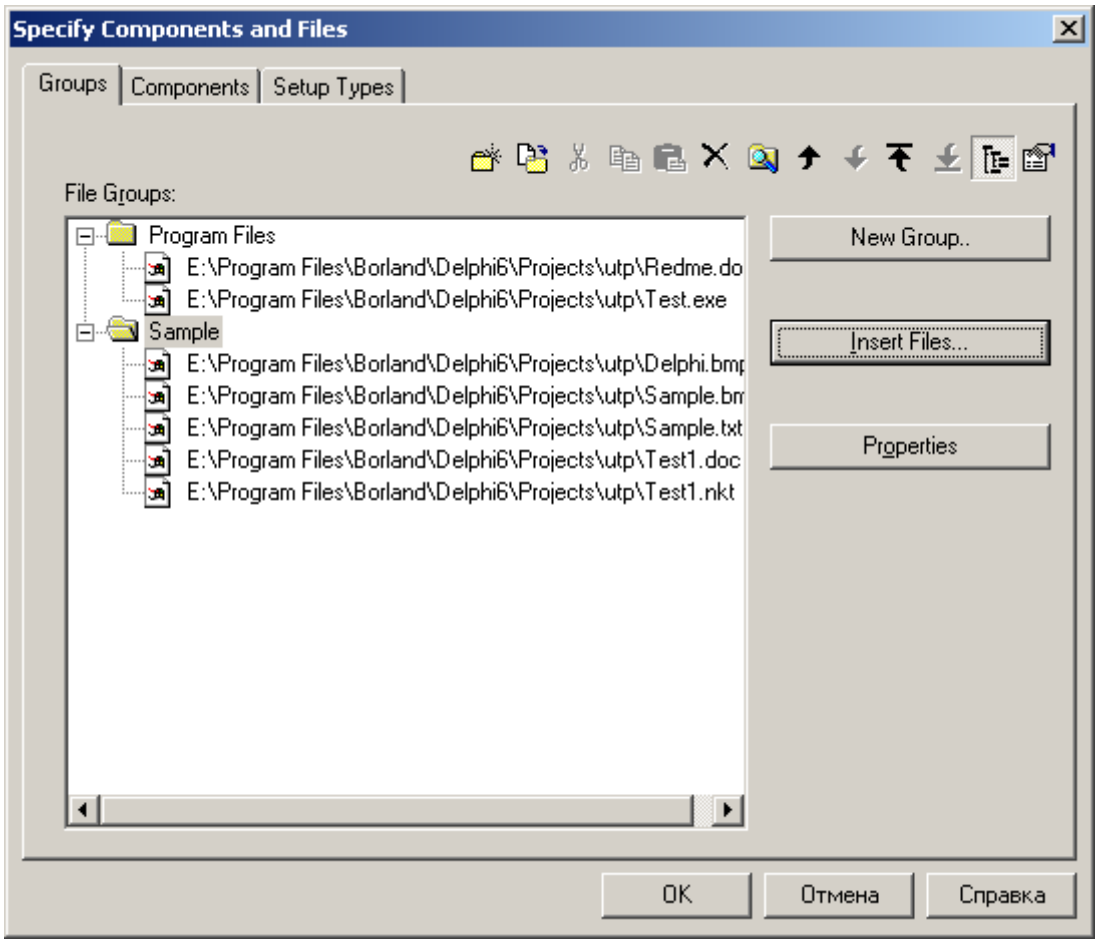
При установці складних програм інсталяційна програма дозволяє користувачеві вибрати тип установки (як правило, це повна, мінімальна або вибіркова установка). У випадку вибіркової установки користувач може вибрати потрібні компоненти, що, як правило, об'єднані в групи, причому можливо вибір як усієї групи, так і окремого компонента.

Перед виконанням команди **Groups and Files** потрібно визначити, які файли необхідно перенести на комп'ютер користувача. Для розглянутого приклада це: файл програми (test.exe), файли тесту, файл readme.doc (містить коротку інформацію про програму). При цьому передбачається, що файл програми і файл readme.doc будуть поміщені в головний каталог додатка, а файли тесту — у підкаталог Sample головного каталогу.

У результаті вибору команди **Groups and Files** відкривається діалогове вікно **Specify Components and Files** (мал. 3.2.7).

Автоматично створена група **Program Files** відповідає головному каталогові встановлюваної програми. Звичайно в цю групу поміщають виконуваний файл програми, файл довідки, файл read.me. Для того щоб додати в цю групу файл, необхідно натиснути кнопку **Insert Files…**і, використовуючи стандартне вікно **Відкрити**, вибрати потрібний файл. Щигликом на квадратику зі знаком плюс можна розкрити групу і подивитися, які файли в ній знаходяться.

При бажанні програміст може розмістити деякі файли в окремому каталозі. Тоді для цих файлів необхідно створити свою групу. Для того щоб створити групу, потрібно натиснути кнопку **New Group…**... Потім у діалоговому вікні, що **відкрилося, Add** group (мал. 3.2.8) у поле **Group Name:** треба ввести ім'я групи, а в поле **Destination Directory:** — ім'я каталогу, у який повинні бути скопійовані файли групи. За замовчуванням каталогом нової групи є каталог <INSTALLDIR>, тобто головний каталог додатка.



Малюнок 3.2.7 - Діалогове вікно Specify Components and Files

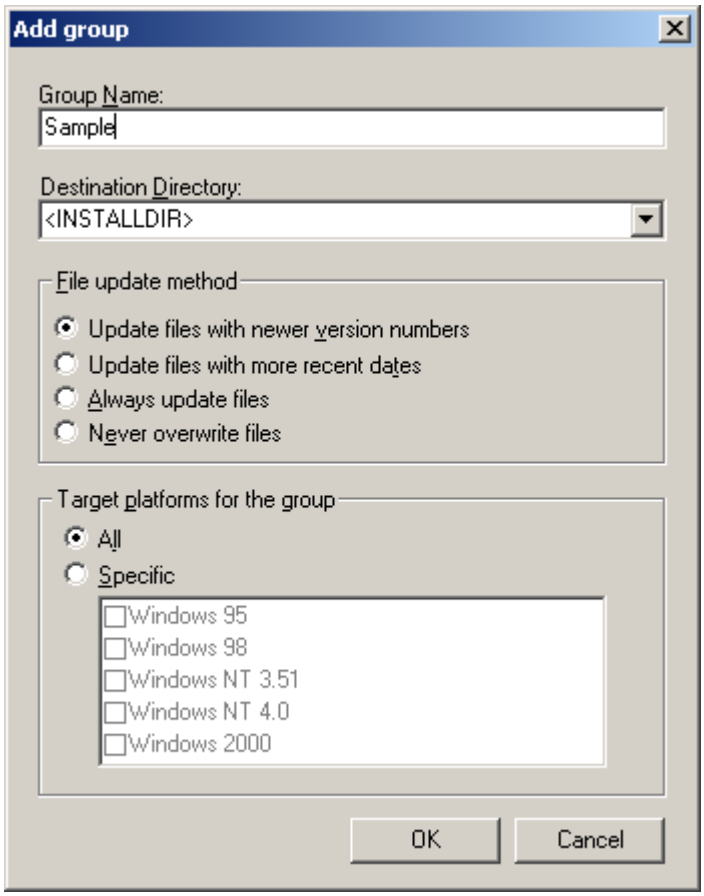
Якщо каталог, куди повинні бути поміщені файли групи, повинний бути створений у цьому каталозі, то ім'я нового каталогу варто ввести в поле **Destination Directory:** після рядка <INSTALLDIR>, розділивши імена символом “зворотна коса риса” (\). Ім'я каталогу також, можна вибрати зі списку, що розкривається. Наприклад, ім'я <WINDIR> відповідає головному каталогові Windows (у більшості випадків це C:\Windows).

Як приклад на мал. 3.2.7 приведений вид діалогового вікна **Specify Components and Files** після того як була створена група Sample і визначені файли, що повинні бути перенесені на комп'ютер користувача.

3.2.2.4. Настроювання діалогів

У процесі роботи інсталяційної програми користувач бачить послідовність сменяющих друг друга стандартних діалогових вікон. Розробляючи програму інсталяції, програміст може задати, які вікна повинні з'являтися, а також визначити їхній вид.

Для того щоб задати діалогові вікна, що будуть з'являтися на екрані під час роботи інсталяційної програми, треба з групи **Select User Interface Components** вибрати команду **Dialog Boxes**. У списку **Settings For**, що відкрився діалогового вікна **Dialog Boxes** (мал. 3.2.9), перераховані діалогові вікна (див. табл. 3.2.2), що можуть з'являтися під час роботи інсталяційної програми.

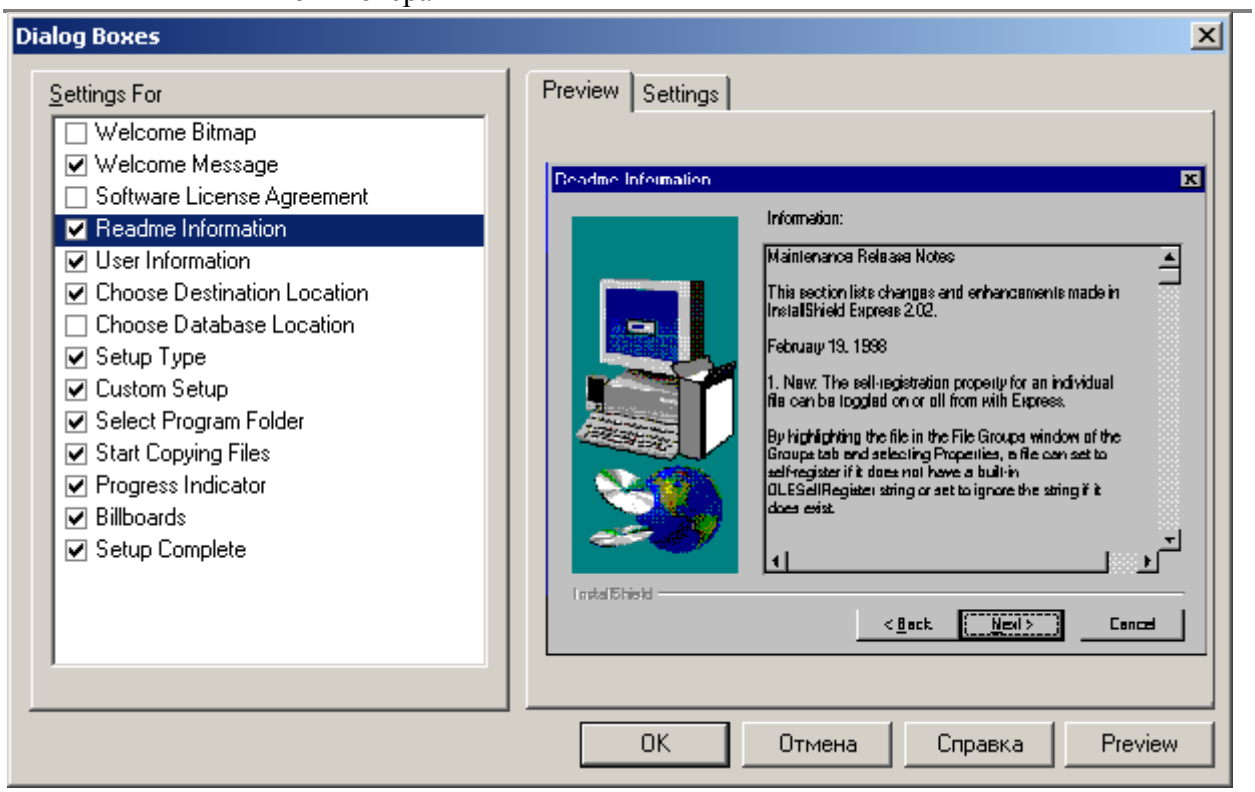


Малюнок 3.2.8 - Діалогове вікно Add group

Таблиця 3.2.2 - Діалогові вікна процесу установки

Діалогове вікно	*Призначення*
Welcome Bitmap	Висновок ілюстрації, що може служити як інформацію про встановлювану програму
Welcome Message	Висновок інформаційного повідомлення
Software License Agreement	Висновок ліцензійного повідомлення, що знаходиться у файлі. Дозволяє перервати процес установки програми у випадку незгоди користувача з пропонованими умовами
Readme Information	Висновок інформації про встановлювану програму
User Information	Запитує інформацію про користувача (ім'я, назва фірми) і, можливо, номер установлюваної копії
Choose Destination Location	Надання користувачеві можливості змінити визначений каталог, у який установлюється програма
Setup Type	Надання користувачеві можливості вибору типу установки програми (Typical — звичайна установка, Compact — мінімальна установка, Custom — вибіркова установка)
Custom Setup	Вибір установлюваних компонентів при вибірковій (Custom) установці
Select Program Folder	Надання користувачеві можливості вибору меню панелі задач, у которое буде поміщена команда запуску встановлюваної програми
Start Copying Files	Висновок інформації, уведеної користувачем на попередніх кроках, з метою її перевірки перед початком безпосередньої установки програми

Діалогове вікно	*Призначення*
Progress Indicator	Показ відсотка виконаної роботи під час установки програми
Setup Complete	Інформує користувача про завершення процесу установки. Дозволяє задати програму, що повинна бути виконана відразу після завершення установки (як правило, це сама встановлена програма). Якщо встановлювана програма вимагає перезавантаження комп'ютера, то виводиться діалог запиту про необхідність перезавантаження комп'ютера



Малюнок 3.2.9 - Діалогове вікно Dialog Boxes

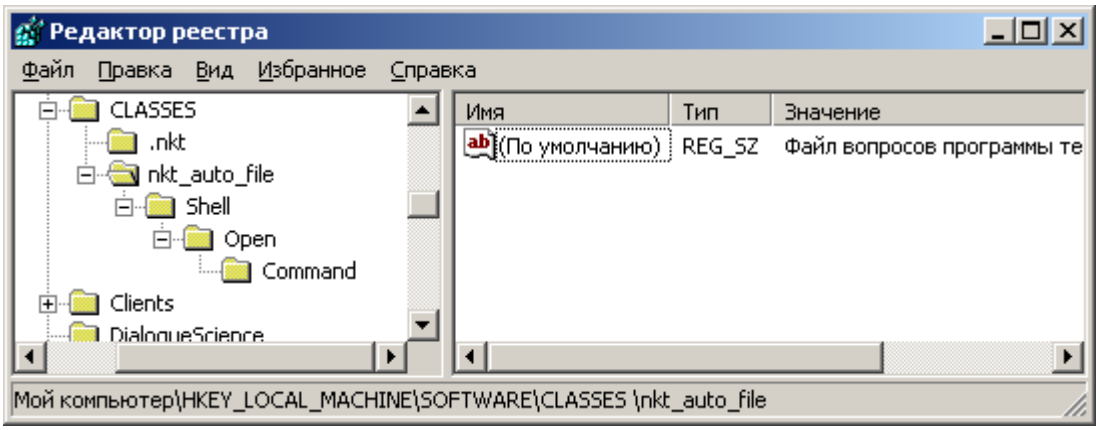
Для того щоб діалогове вікно з'являлося під час роботи інсталяційної програми, необхідно установити прапорець, розташований ліворуч від назви діалогового вікна. Якщо діалог припускає висновок специфічної інформації, про що свідчить поява вкладки **Settings** поруч із вкладкою **Preview**, то треба вибрати цю вкладку і ввести значення параметра. Більшість діалогових вікон як параметр запитують ім'я текстового або bmp (тільки 16 квітів) файлу.

У найпростішому випадку програма інсталяції може обмежитися висновком наступних діалогів:

- Readme Information
- Choose Destination Location
- Select Program Folder
- Progress Indicator
- Setup Complete

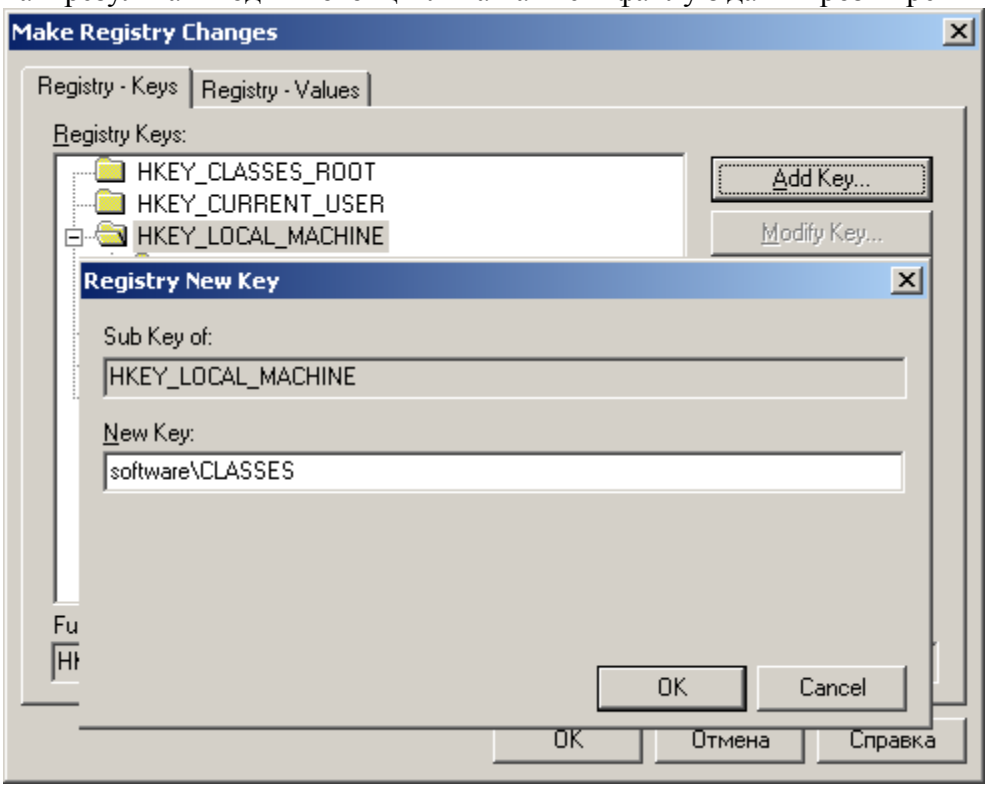
3.2.2.5. Зміни в реєстрі

Група команд **Make Registry Changes** (зміни в реєстрі) дозволяє задати, які зміни треба внести до реєстру Windows комп'ютера користувача.



Малюнок 3.2.10 – Розділи .nkt і nkt_auto_file реєстру

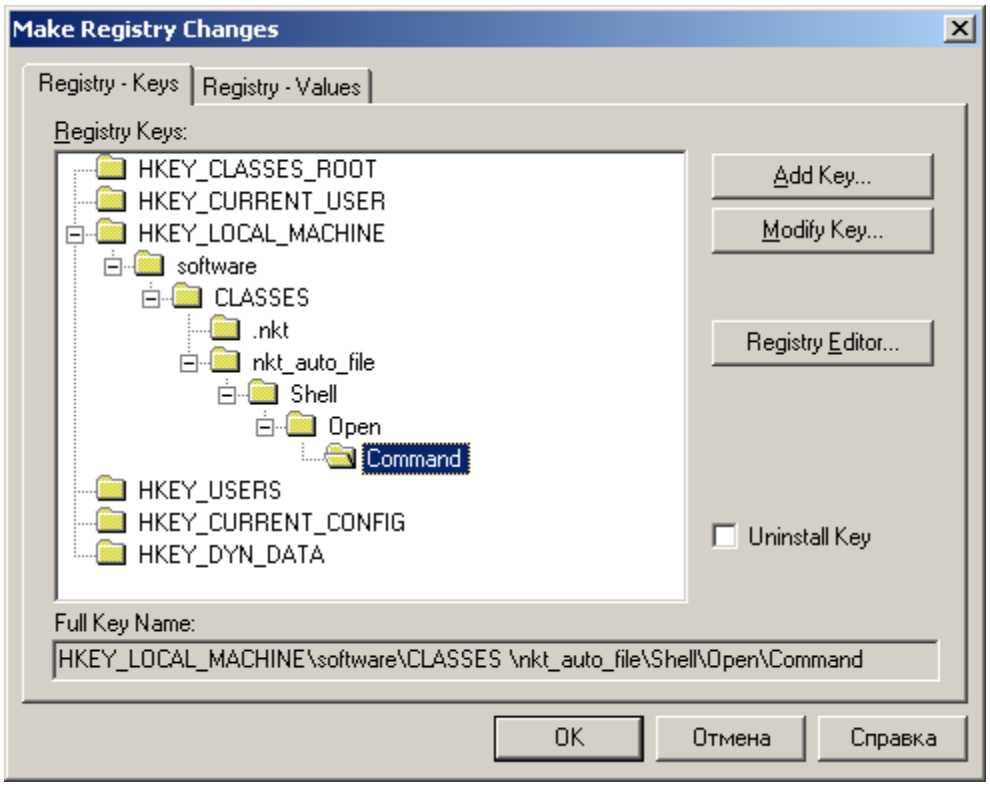
Реєстр Windows являє собою ієрархічно організовану базу даних, у якій знаходяться настроювання Windows, пристроїв і додатків. Наприклад, у реєстрі зберігається інформація про те, з якою програмою зв'язані файли визначеного типу і, отже, яка програма буде запущена в результаті подвійного щиглика на імені файлу з даним розширенням.



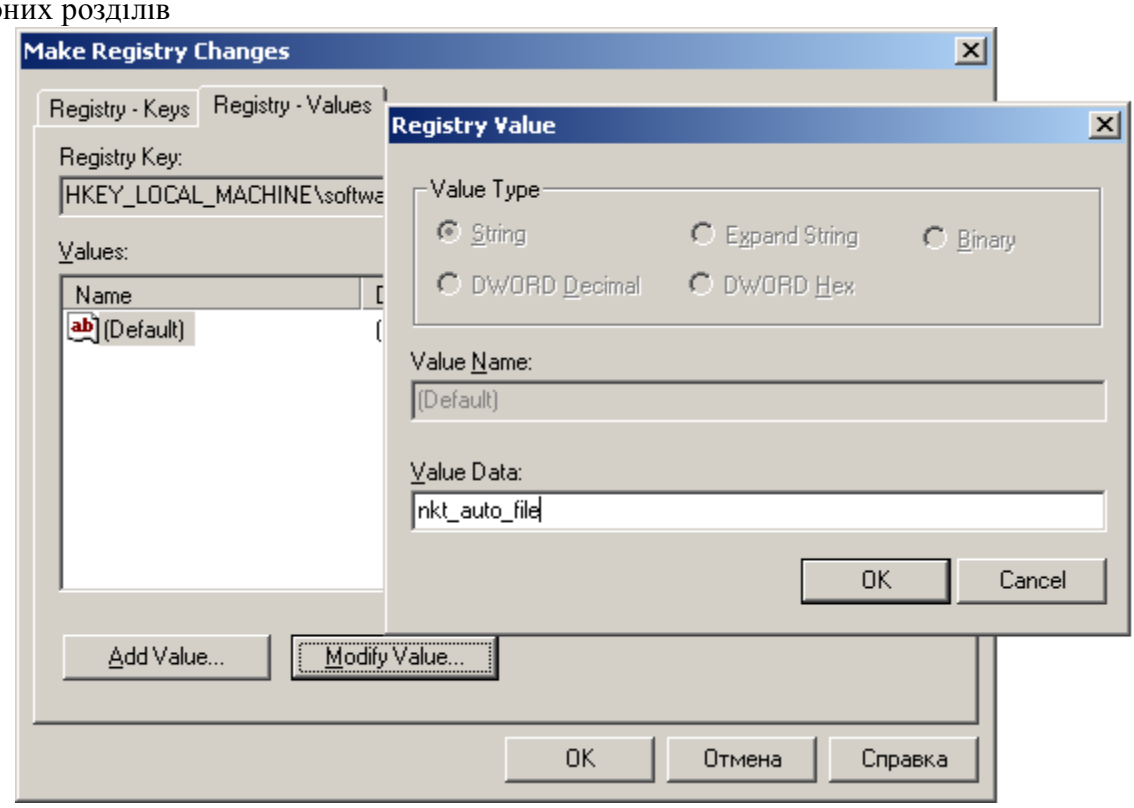
Малюнок 3.2.11 - Створення нового розділу реєстру

Розглянемо, як вказати інсталяційній програмі зміни, які треба внести до реєстру Windows. Нехай необхідно, щоб після установки на комп'ютер користувача програма тестування могла бути запущена в результаті подвійного щиглика на імені файлу тесту (файл із расширением.nkt). Для цього треба, щоб у реєстрі, у розділі HKEY_LOCAL_MACHINE\software\CLASSES був раздел.nkt (мал. 3.2.10), що має строковий параметр nkt_auto_file, і, що знаходиться в цьому ж розділі і имеющий ієрархічній структурі, роздягнув nkt_auto_file (мал. 3.2.10).

Параметр роздягнула nkt_auto_fiie визначає текст, що виводиться поруч з ім'ям файлу у вікнах Windows, наприклад, при перегляді вмісту папки, а параметр command — команду (програму), що запускається в результаті подвійного щиглика на імені файлу.



Малюнок 3.2.12 - Діалогове вікно Make Registry Changes після створення всіх потрібних розділів



Малюнок 3.2.13 - Уведення значень параметрів розділів реєстру

Таким чином, для того щоб користувач міг запустити програму тестування подвійним щигликом на імені файлу тесту, інсталяційна програма повинна додати до реєстру його комп'ютера приведені вище розділи. Щоб вона це зробила, треба у вікні проекту з групи **Make Registry Changes** вибрати команду **Keys**. У розглянутому прикладі розділи .nkt і nkt_auto_file повинні знаходитися в розділі HKEY_LOCAL_MACHINE\Software \CLASSES, тому треба виділити рядок HKEY_LOCAL_MACHINE, натиснути кнопку **Add Key…**і в поле **New Key** діалогового

вікна, що **з'явилося, Registry New** Key увести рядок: software\CLASSES (мал. 3.2.11). Після натискання кнопки **ОК** будуть створені розділи Software і CLASSES.

Аналогічним образом створюються розділи .nkt і nkt_auto_file\Shell\Open\Command (перед тим як виконати команду **Add Key…**варто виділити розділ CLASSES). На мал. 3.2.12 приведений вид вікна **Make Registry Changes** після створення всіх необхідних розділів.

Після створення розділів необхідно задати значення параметрів. Щоб це зробити, треба у вкладці **Registry — Keys** виділити потрібний розділ і потім вибрати вкладку **Registry — Values**. В вкладці **Registry — Values** треба вибрати потрібний параметр, натиснути кнопку **Modify Value…**і в поле **Value Data:** діалогового вікна, що **з'явилося, Registry** Value увести значення параметра (мал. 3.2.13).

Нижче, у табл. 3.2.3, перераховані розділи, параметри розділів і значення, які необхідно привласнити параметрам у діалоговому вікні **Make Registry Changes** на вкладці **Registry — Values**.

Таблиця 3.2.3 - Розділи, їхні параметри і значення параметрів

Роздягнув	*Параметр (Name)*	*Значення (Data)*
.nkt	Default	nkt_auto_file
nkt_auto_file	Default	Файл питань програми тестування
Command	Default	<INSTALLDIR>\test.exe “%1”

Як було сказано вище, параметр Command визначає програму, що запускається в результаті подвійного щиглика на імені файлу з зазначеним розширенням. У розглянутому прикладі як параметр програмі передається ім'я файлу, що і відбито в команді наявністю рядка %1. При запуску програми параметр %1 буде замінений ім'ям файлу, на якому зроблений подвійний щиглик. Варто особливо звернути увагу на подвійних лапк, у які укладений рядок %1. Якщо їх не поставити, то, у випадку якщо в імені каталогу або в імені файлу є пробіли, програма одержить як параметр тільки першу (до пробілу) подстроку а не повне ім'я файлу.

3.2.2.6. Місце розташування та команда запускові додатка

Команди групи **Specify Folders and Icons** дозволяють задати меню, у которое буде поміщена команда запуску встановлюваної програми, а також команду запуску і значок, що ідентифікує програму.

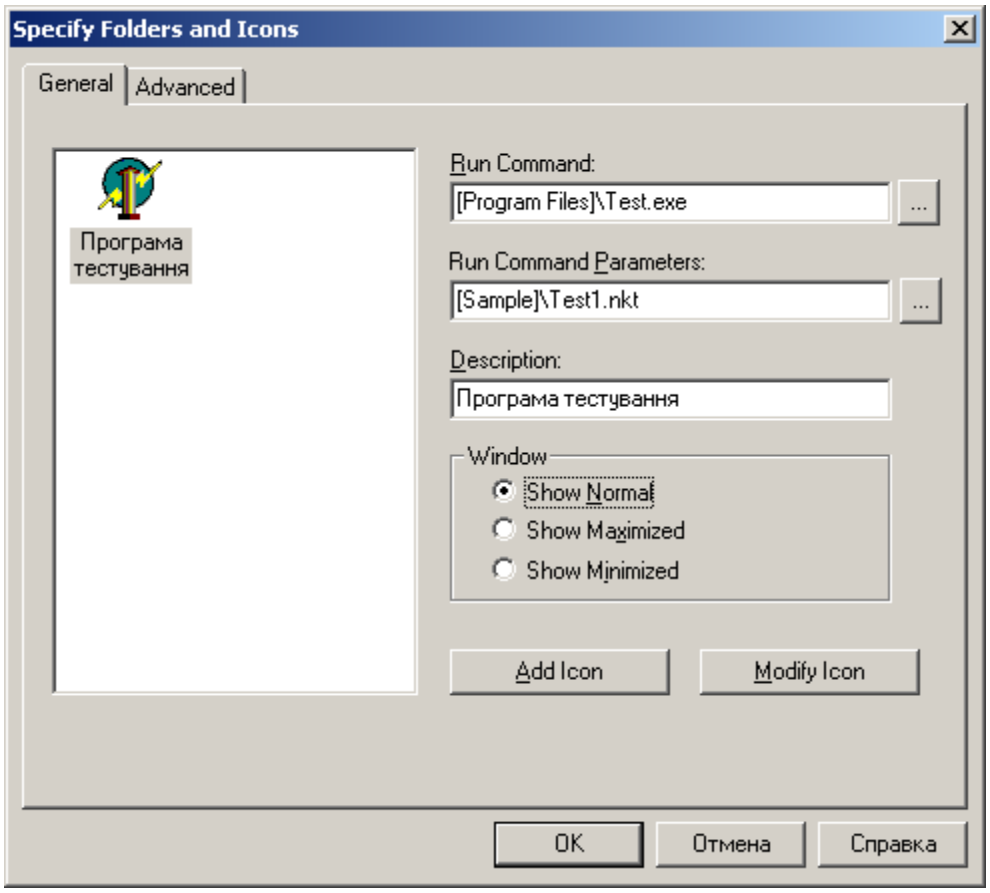
При виборі команди **General Settings** з цієї групи відкривається вкладка **General** діалогового вікна **Specify Folders and Icons** (мал. 3.2.14), на якій можна задати команду запуску встановлюваного додатка.

У поле **Run Command:** (Команда запуску) потрібно ввести ім'я виконуваного файлу встановлюваної програми, а в поле **Description:** (Підпис) — текст, що буде з'являтися в меню команд, а у випадку розміщення значка на робочому столі — під значком, що ідентифікує програму.

Після введення команди запуску і підпису потрібно натиснути кнопку **Add Icon**, у результаті чого в полі значків буде виведений значок програми і підпис (їхній вид дає представлення про те, що буде поміщено на робочий стіл).

Група перемикачів **Window** дозволяє задати розмір вікна програми після запуску. Звичайно після запуску програми вікно програми має такий же розмір, як і під час розробки програми (що відповідає обраному перемикачеві **Show Normal**). При установці перемикача в положення **Show Maximized** вікно програми займає весь екран. При виборі перемикача **Show Minimized** після запуску вікно програми автоматично звертається.

На вкладці **Advanced** (мал. 3.2.15) діалогового вікна **Specify Folders and Icons** у поле **Start in:** можна задати робочий каталог програми, а в поле **Icon:** — файл значка, установлюваного додатка.



Малюнок 3.2.14 - Вкладка General діалогового вікна Specify Folders and Icons

У поле **Shortcut Key:** можна установити сполучення “гарячих клавіш” для запуску програми. Для цього необхідно виділити іконку програми для якої ви хочете призначити сполучення “гарячих клавіш” і перемістити курсор у поле введення **Shortcut Key:**. Після цього потрібно натиснути на клавіатурі клавішу, що у сполученні з клавішами <Ctrl> і <Alt> буде складати сполучення “гарячих клавіш”. Наприклад, при натисканні клавіші <A>, буде призначене сполучення “гарячих клавіш” <Ctrl>+<Alt>+<A> (див мал. 3.2.15).

Значки додатків знаходяться у файлах з расширением.ico. Файл значка можна створити, наприклад, за допомогою вхідної до складу Delphi утиліти **Image Editor**, яку можна запустити в Delphi, вибравши пункт меню Tools⇨ Image Editor.

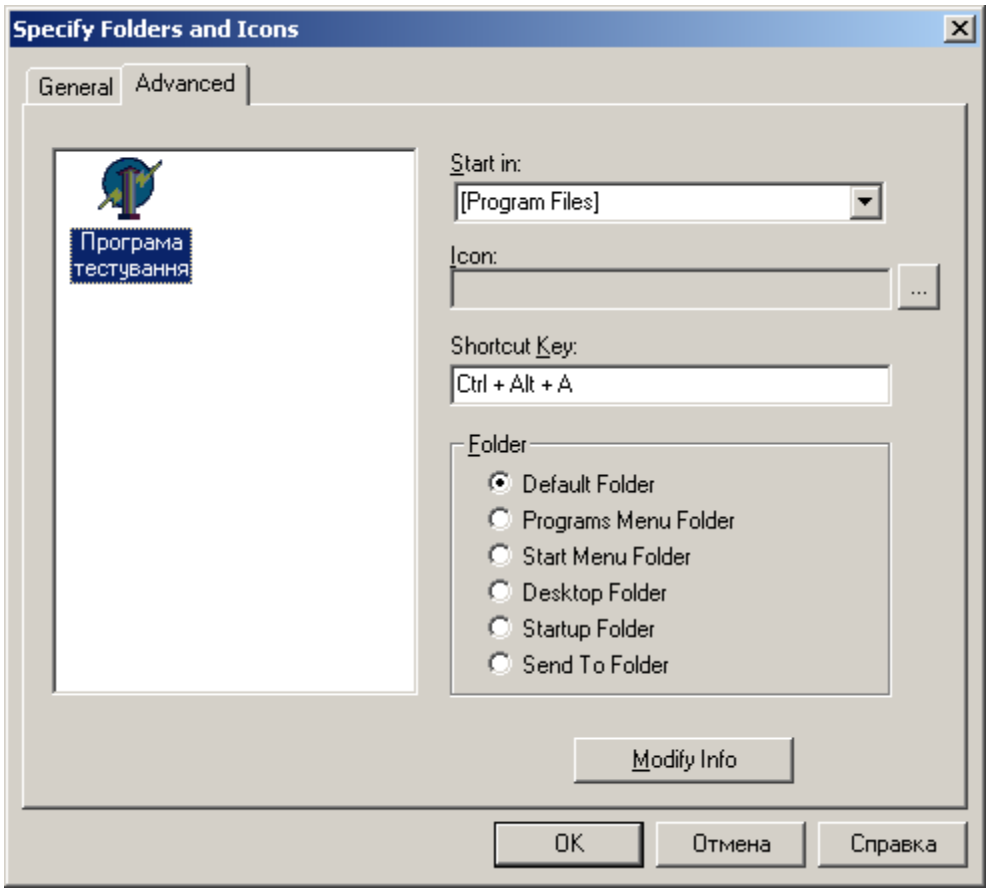
Перемикачі групи **Folder** дозволяють задати, де буде розташований значок установлюваного додатка:

- у меню Програми (**Programs Menu Folder**);
- у меню Пуск (**Start Menu Folder**);
- на робочому столі (**Desktop Folder**);
- у меню Автозавантаження (**Startup Folder**);
- у меню Послати (**Send To Folder**).

3.2.3. Створення установочної дискети

Після визначення всіх характеристик програми установки можна приступити до створення настановної дискети. Цей процес складається з наступних кроків:

- Створення образа настановної дискети
- Тестування процесу установки
- Копіювання образа настановної дискети на дискету



Малюнок 3.2.15 - Вкладка Advanced діалогового вікна Specify Folders and Icons

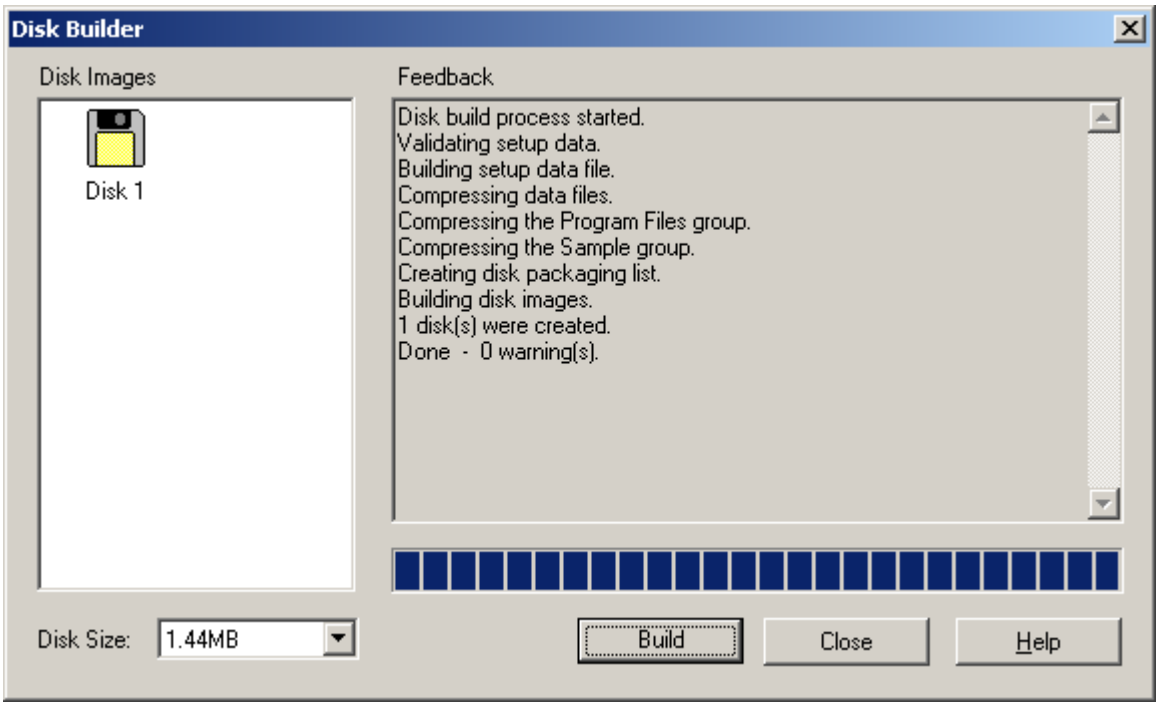
Для того щоб створити образ настановної дискети, що програма InstallShield Express розміщає на твердому диску комп'ютера в каталозі проекту програми установки, необхідно в розділі **Run Disk Builder** вибрати команду **Disk Builder** (див. мал. 3.2.3) або вибрати пункт меню Checklist⇨Run Disk Builder.

У діалоговому вікні, що **відкрилося, Disk** Builder (мал. 3.2.16), у списку, що **розкривається, Disk Size**:, треба вибрати обсяг диска, що буде використовуватися як носія інсталяційної програми (для невеликих програм як носія звичайно використовуються дискети обсягом 1,44 Мбайт) і натиснути кнопку **Build**.

Після того як інсталяційна програма буде створена, у вікні **Disk Images** можна побачити, скільки дискет вона буде займати (мал. 3.2.16).

Після створення образа настановної дискети можна подивитися, як буде працювати програма установки. Для цього необхідно в розділі **Test the Installation** вибрати команду **Test Run** (див. мал. 3.2.3) або вибрати пункт меню Checklist⇨Test the Installation.

Якщо програма установки працює правильно, то можна виконати копіювання програми установки на дискети, для чого треба в розділі **Create Distribution Media** вибрати команду **Copy to Floppy** (див. мал. 3.2.3) або вибрати пункт меню Checklist⇨Create Distribution Media.



Малюнок 3.2.16 - Діалогове вікно Disk Builder після завершення процесу створення образа настановної дискети

Література [3].

ЛІТЕРАТУРА

Основна література

1 Фленов М.Е. Библия Delphi. – 3-е узд. перераб. и доп. –СПБ.: БХВ-Петербург, 2011. – 688 с.: ил. + CD-ROM

2 Гофман В.Э., Хоменко А.Д. Delphi. Быстрый старт. –СПБ.: БХВ-Петербург, 2003. – 288 с.: ил.

3 Миронченко А.С. Императивное и объектно-ориентированное программирование на Turbo Pascal и Delphi. Глубокое погружение. – Одесса: ВМВ, 2007. - 408 с.: ил.

4 Смирнов С.И. Уроки программирования Pascal - Delphi. Часть 1. – Красноярск, 2011. – 100 с.

5 Архангельский А.Я. Программирование в Delphi для Windows. Версии 2006, 2007, Turbo Delphi. – М.: ООО «Бином-Пресс», 2007. - 1248 с.: ил.

6 Фаронов В.В. Программирование баз данных в Delphi 7. Учебный курс. – СПб.: Питер, 2006. – 459 с.: ил.

7 Сорокин А.В. Delphi. Разработка баз данных. – СПб.: Питер, 2005. – 477 с.: ил.

Додаткова література

8 Культин Н.Б. Программирование на Object Pascal в Delphi 5. СПб.: БХВ – Санкт-Петербург, 1999. –464 с., ил. ISBN 5-8206-0079-7

9 Архангельский А.Я. Delphi 5. Справочное пособие – М.: ЗАО “Издательство БИНОМ”, 2001 г. – 768 с.: ил. ISBN 5-7989-0203-X

10 Калаверт Черльз. Delphi 4. Энциклопедия пользователя: Пер. с англ. – К.: Издательство «ДиаСофт», 1998. – 800 с. ISBN 5-900570-16-9

11 Р. Баас, М. Фервай, Х. Гюнтер. Delphi 4: полное руководство: пер. с нем. – К.: Издательская группа BHV, 1999. – 800 с.: ил. ISBN 5-7315-0036-3

Додаток А

Стандартні процедури та функції Object Pascal

Таблица А.1 - Список категорий стандартных процедур и функций Object Pascal

Список категорий
Арифметические и математические
Управление файлами
Дата и время
Операции с динамическими переменными
Ввод/вывод
Исключительные ситуации
Финансовые и математические
Операции с действительными числами
Меню
Нетипизированные файлы
Выполнение программы
Статистика
Текстовые файлы
Строки Unicode
Операции с переменными типа variant
Различные стандартные процедуры и функции
Операции с переменными типа Currency (деньги)
Указатели и адреса

Таблица А.2 - Список арифметических и математических стандартных процедур и функций

Арифметические и математические стандартные процедуры и функции	
Abs	Определяет абсолютное значение аргумента
ArcCos	Определяет значение арккосинуса x
ArcCosh	Определяет значение гиперболического арккосинуса x
ArcSin	Определяет значение арксинуса x
ArcSinh	Определяет значение гиперболическое арксинуса x
ArcTan	Определяет арктангенс переданного аргумента
ArcTan2	Вычисляет ArcTan(y/x) и возвращает угол с учетом полученного квадранта
ArcTanh	Определяет значение гиперболического арктангенса x
Ceil	Округляет значение в положительную сторону
Cos	Определяет косинус x
Cosh	Определяет гиперболический косинус x
Cotan	Определяет котангенс x
CycleToRoad	Преобразовывает угол, заданный в количествах оборотов (циклов), в угол, заданный в радианах
Dec	Уменьшает значение переменной
DegToRoad	Преобразовывает угол из градусов в радианы
Exp	Вычисляет экспоненциальное значение передаваемого аргумента
Floor	Округляет переменную в сторону ближайшего меньшего целого числа
FPower	Умножает аргумент на 10 в заданной степени (Val * 10**Pow)
Frac	Возвращает дробную часть аргумента

Арифметические и математические стандартные процедуры и функции	
Frexp	Возвращает мантиссу и экспонент переданного значения
GradToRad	Преобразовывает угол, заданный в десятичных градусах (Grad), в радианы
High	Возвращает наибольшее значение в области значений аргумента
Hypot	Определяет длину гипотенузы прямоугольного треугольника
Inc	Увеличивает значение переменной
Int	Возвращает целую часть передаваемого аргумента
IntPower	Определяет показатель степени исходя из значений основания и целочисленной экспоненты
Ldexp	Определяет значение x* (2P)
Ln	Определяет натуральный логарифм передаваемого аргумента
LnXP1	Определяет натуральный логарифм (X+1)
Log10	Определяет десятичный логарифм
Log2	Определяет двоичный логарифм
LogN	Определяет логарифм с основанием N
Low	Возвращает наименьшее значение в области значений аргумента
Mean	Определяет среднее арифметическое всех значений массива
MaxIntValue	Возвращает наибольшее значение со знаком из ряда содержащихся в массиве целых чисел
MaxValue	Возвращает наибольшее значение со знаком из ряда содержащихся в массиве действительных чисел
MinIntValue	Возвращает наименьшее значение со знаком из ряда содержащихся в массиве целых чисел
MinValue	Возвращает наименьшее значение со знаком из ряда содержащихся в массиве действительных чисел
Norm	Определяет Евклидову норму
Odd	Проверяет, является ли аргумент нечетным числом
Ord	Возвращает порядковый номер значения
Pi	Возвращает значение π
Poly	Вычисляет стандартный полином для параметра x
Power	Возводит число в степень
Pred	Возвращает предыдущее значение порядковой переменной
RadToDeg	Преобразовывает угол, указанный в радианах, в угол в градусах
RadToGrad	Преобразовывает угол, указанный в радианах, в угол в десятичных градусах (Grad)
Round	Округляет значение действительного типа до целочисленного значения
Sin	Возвращает синус передаваемого аргумента
SinCos	Определяет синус и косинус угла
Sinh	Возвращает гиперболический синус угла
Sqr	Возводит число в квадрат
Sqrt	Вычисляет корень квадратный
Succ	Возвращает последующее значение порядковой переменной
Sum	Определяет сумму всех элементов массива
SumInt	Определяет сумму всех элементов массива целых чисел
SumOfSquares	Определяет сумму квадратов всех элементов массива
SumAndSquares	Определяет сумму и сумму квадратов всех элементов массива
Tan	Определяет тангенс аргумента
Tanh	Определяет гиперболический тангенс аргумента
Trunc	Отсекает дробную часть действительного числа

Таблица А.3 - Обзор стандартных процедур и функций, используемых для управления файлами

Стандартные процедуры и функции, используемые при работе с файлами	
ChangeFileExt	Изменяет расширение файла
Close	Закрывает файл
DeleteFile	Удаляет файл
DiskFree	Возвращает количество свободного дискового пространства, имеющегося на дискете/жестком диске
DiskSize	Возвращает общий объем дискового пространства на дискете/жестком диске
DosPathToUnixPath	Преобразовывает указанный в формате DOS путь в путь формата
ExpandFileName	Возвращает строку, которая содержит имя файла
ExpandUNCFileName	Возвращает строку, которая содержит полностью определенное имя файла в сети
ExtractFileDir	Возвращает имя дисковода и каталога для указанного имени файла
ExtractFileDrive	Возвращает имя дисковода для указанного имени файла
ExtractFileExt	Возвращает расширение файла
ExtractFileName	Возвращает имя файла
ExtractFilePath	Возвращает путь к файлу
FileAge	Возвращает в DOS-формате дату и время создания или последнего изменения файла
FileClose	Закрывает указанный файл
FileCreate	Создает файл с указанным именем
FileDateToDateTime	Преобразовывает данные из формата даты DOS в формат Delphi
FileExists	Возвращает значение True, если файл существует
FileGetAttr	Возвращает атрибуты файла
FileGetDate	Возвращает (в DOS-формате) дату и время создания или последнего изменения файла
FileOpen	Открывает указанный файл
FileSearch	Осуществляет поиск указанного файла в заданном каталоге
FileSeek	Изменяет текущую позицию внутри файла
FileSetAttr	Изменяет атрибуты файла
FileSetDate	Изменяет дату и время создания или последнего изменения файла
FileWrite	Записывает данные в указанный файл
FindClose	Завершает последовательность вызовов функций FindFir3t/FindNext
FindFirst	Производит поиск файла с определенными атрибутами в заданном каталоге
FindNext	Производит поиск файла после вызова процедуры FindFirst
RenameFile	Переименовывает файл
UnixPathToDosPath	Преобразовывает указание пути формата UNIX в путь формата DOS

Таблица А.4 - . Список стандартных процедур и функций, предназначенных для проведения операций с датой и временем

Стандартные процедуры и функции, выполняющие операции с датой и временем	
Date	Возвращает системную дату
DateTimeToFileDate	Преобразовывает формат даты Delphi в формат даты DOS
DateTimeToStr	Преобразовывает дату в формате TDateTime в строку
DateTimeToString	Преобразовывает дату в формате TDateTime в строку
DateTimeToSystemTime	Преобразовывает время из формата Delphi TDateTime в формат Win32 API TSystemTime
DateTimeToTimeStamp	Преобразовывает значение типа TDateTime в соответствующее значение типа TTimeStamp

Стандартные процедуры и функции, выполняющие операции с датой и временем	
DateToStr	Преобразовывает дату в формате TDateTime в строку
DayOfWeek	Возвращает номер дня недели
DecodeDate	Расшифровывает указанную дату
DecodeTime	Расшифровывает значение времени
EncodeDate	Преобразовывает дату, заданную числом, месяцем и годом, в формат TDateTime
EncodeTime	Преобразовывает время, заданное часами, минутами и секундами, в формат TDateTime
FormatDateTime	Преобразовывает значения даты и времени согласно заданному формату
GetFormatSettings	Устанавливает все параметры формата даты и времени в исходное состояние
IncMonth	Определяет дату, которая была задана определенным количеством месяцев до или после исходной даты
IsLeapYear	Определяет, является ли указанный год високосным.
MsecsToTimeStamp	Преобразовывает количество миллисекунд в значение TTimeStamp
Now	Возвращает системную дату и системное время
StrToDate	Преобразовывает строку даты в формат TDateTime
StrToDateTime	Преобразовывает строку даты и времени в формат TDateTime
StrToTime	Преобразовывает строку времени в формат TDateTime
Time	Возвращает системное время
TimeStampToDateTime	Преобразовывает значение TTimeStamp в соответствующее значение TDateTime
TimeStampToMSecs	Преобразовывает значение TTimeStamp в абсолютное количество миллисекунд
TimeToStr	Преобразовывает значения в формате TDateTime в строку
SystemTimeToDateTime	Преобразовывает значение TSystemTime в формат Delphi TDateTime

Таблица А.5 - Список стандартных процедур и функций, предназначенных для работы *с* динамическими переменными

Стандартные процедуры и функции, предназначенные для работы с динамическими переменными	
Dispose	Удаляет динамическую переменную
Finalize	Деинициализирует динамическую переменную
FreeMem	Удаляет динамическую переменную указанного размера
GetMem	Создает динамическую переменную указанного размера и присваивает начальный адрес области памяти, выделенной переменной-указателю
Initialize	Инициализирует динамическую переменную
New	Создает новую динамическую переменную
ReallocMem	Перераспределяет память для динамической переменной

Таблица А.6 - Список стандартных процедур и функций ввода/вывода

Стандартные процедуры и функции ввода/вывода	
GraphicFilter	Возвращает строку фильтра, совместимую со свойством Filter диалоговых окон открытия и сохранения графических файлов
InputBox	Выводит диалоговое окно с полем ввода
InputQuery	Выводит диалоговое окно с полем ввода
IOResult	Возвращает статус последней выполненной операции ввода/вывода

Стандартные процедуры и функции ввода/вывода	
MatchesMask	Определяет строку, соответствующую формату, указанному в строке фильтра
MkDir	Создает новый подкаталог
LoginDialog	Выводит диалоговое окно регистрации, предназначенное для работы с базой данных
RemoveDir	Удаляет пустой каталог
Rename	Переименовывает внешний файл
Reset	Открывает файл для чтения
Rewrite	Создает и открывает новый файл для записи
RmDir	Удаляет пустой подкаталог
Seek	Перемещает позиционный указатель в открытом файле
SelectDirectory	Выводит диалоговое окно выбора каталога
SetCurrentDir	Задает текущий каталог
Truncate	Удаляет все данные в файле, начиная с текущей позиции и до конца файла

Таблица А.7 - Список стандартных процедур и функций, применяемых для обработки исключительных ситуаций

Стандартные процедуры и функции, применяемые для обработки исключительных ситуаций	
DatabaseError	Вызывает исключительную ситуацию EDatabaseError
DBIError	Вызывает исключительную ситуацию EDBEngineError для переданного из BDE кода ошибки
ExceptAddr	Возвращает адрес фрагмента программного кода, в котором была вызвана исключительная ситуация
ExceptionErrorMessage	Форматирует стандартное сообщение об ошибке
ExceptObject	Возвращает ссылку на объект текущей исключительной ситуации
OutOfMemoryError	Вызывает исключительную ситуацию EOutOfMemoryError
RaiseLastwin32Error	Вызывает исключительную ситуацию для ошибки, возникшей в результате вызова функции Win32 API
ShowException	Выводит об ошибке и адрес фрагмента программного кода, в котором была вызвана исключительная ситуация
Win32Check	Проверяет возвращаемое значение вызова функции Windows API и в случае ошибки вызывает исключительную ситуацию

Таблица А.8 - Список стандартных процедур и функций, используемых для выполнения финансово-математических операций

Стандартные процедуры и функции, используемые для выполнения финансово-математических операций	
DoubleDecliningBalance	Определяет сумму амортизационных отчислений за определенный период
FutureValue	Определяет повышение стоимости вклада
InterestPayrnent	Определяет кредитную процентную ставку
InterestRate	Определяет процентную ставку инвестиций
IntemalRateOfRetum	Определяет начальную процентную ставку инвестиций
NetPresentValue	Определяет текущее значение из массива с расчетными значениями текущих платежей
NumberOfPeriods	Определяет количество периодов погашения кредита
Payment	Определяет общую сумму погашения кредита
PeriodPayment	Определяет долю суммы для регулярного погашения кредита
PresentValue	Определяет текущее значение вклада

Стандартные процедуры и функции, используемые для выполнения финансово-математических операций	
SLNDepreciation	Определяет сумму линейных амортизационных отчислений за имущество
SYDDepreciation	Определяет сумму дигрессивных амортизационных отчислений

Таблица А.9 - Список стандартных процедур и функций, предназначенных для выполнения операций с действительными числами

Стандартные процедуры и функции, предназначенные для выполнения операций с действительными числами	
FloatToDecimal	Преобразовывает значение с плавающей запятой в значение в десятичном представлении
FloatToStr	Преобразовывает значение с плавающей запятой в строку
FloatToStrF	Преобразовывает значение с плавающей запятой в строку с использованием заданного формата представления
FloatToText	Преобразовывает значение с плавающей запятой в строку
FloatToTextFmt	Преобразовывает значение с плавающей запятой в строку, содержащую число в десятичном представлении с использованием заданного формата
FormatFloat	Форматирует строку с действительным числом с использованием указанного формата
StrToFloat	Преобразовывает строку в значение с плавающей запятой
TextToFloat	Преобразовывает строку с завершающим нулем в значение с плавающей запятой

Таблица А.10 - Список стандартных процедур и функций, применяемых для проведения операций над меню

Стандартные процедуры и функции, применяемые для проведения операции над меню	
NewMenu	Создает и инициализирует главное меню
NewPopupMenu	Создает и инициализирует контекстное меню
Shortcut	Создает числовое значение для заданного сочетания клавиш
ShortCutToKey	Возвращает код виртуальной клавиши и Shift-статус для заданного сочетания клавиш меню
ShortCutToText	Преобразовывает числовое значение сочетания клавиш в строку, описывающую данное сочетание клавиш
TextToShortCut	Возвращает числовое значение сочетания клавиш для пункта меню по данным из текстовой строки

Таблица А.11 - Обзор стандартных процедур и функций, используемых для проведения операций над нетипизированными файлами

Стандартные процедуры и функции, используемые для проведения операций над нетипизированными файлами	
BlockRead	Считывает из файла одну или несколько записей
BlockWrite	Записывает в файл одну или несколько записей

Таблица А.12 - Список стандартных процедур и функций, используемых для управления ходом выполнения программы

Стандартные процедуры и функции, используемые для управления ходом выполнения программы	
Abort	Прерывает процесс, не выдавая сообщение об ошибке
AddTerrainateProc	Добавляет процедуру к списку процедур выхода
Break	Завершает действие оператора for, while или repeat
CallTerminateProcs	Вызывает все процедуры из списка завершаемых процедур

Стандартные процедуры и функции, используемые для управления ходом выполнения программы	
Continue	Осуществляет переход на новый цикл в операторах for, while или repeat
Exit	Осуществляет выход из текущего блока
Halt	Останавливает выполнение программы и передает управление операционной системе
RunError	Завершает выполнение программы и генерирует сообщение об ошибке времени выполнения (runtime error)

Таблица А.13 - Обзор стандартных процедур и функций, выполняющих статистические операции

Стандартные процедуры и функции, выполняющие статистические операции	
MeanAndStdDev	Определяет среднее и стандартное отклонения
MomentSkewKurtosis	Определяет среднее значение, дисперсию, отклонение и периодичность
PopnStdDev	Определяет стандартное отклонение для совокупности данных
PopnVariance	Определяет дисперсию (TotalVariance/n)
RandG	Генерирует случайное число
StdDev	Определяет стандартное отклонение
TotalVariance	Определяет статистическую дисперсию
Variance	Определяет статистическую дисперсию

Таблица А.14 - Список стандартных процедур и функций, выполняющих операции над строками

Стандартные процедуры и функции, выполняющие операции над строками	
AdjustLineBreaks	Преобразовывает символ перевода строки в последовательность CR/LF (возврат каретки, перевод строки)
AnsiCompareFileName	Сравнивает имена файлов
AnsiCompareStr	Сравнивает две строки указанной длины, учитывая регистр
AnsiCompareText	Сравнивает две строки указанной длины, не учитывая регистр
AnsiExtractQuotedStr	Преобразовывает строку, заключенную в кавычки, в строку без кавычек
AnsiLastChar	Возвращает указатель на последний символ в строке
AnsiLowerCase	Преобразовывает строку в строку со строчными буквами
AnsiLowerCaseFileName	Преобразовывает строку в строку со строчными буквами
AnsiPos	Возвращает позицию заданной подстроки в строке
AnsiQuotedStr	Преобразует строку в строку, заключенную в кавычки
AnsiStrComp	Сравнивает две строки с завершающим нулем, учитывая регистр
AnsiStrIComp	Сравнивает две строки с завершающим нулем, не учитывай регистр
AnsiStrLastChar	Возвращает указатель на последний символ в строке с завершающим нулем
AnsiStrLComp	Сравнивает две строки, учитывая регистр
AnsiStrLIComp	Сравнивает две строки с завершающим нулем, не учитывая регистр
AnsiStrLower	Преобразовывает все символы в строке с завершающим нулем в строчные буквы
AnsiStrPos	Возвращает указатель на первое появление одной строки в другой строке
AnsiStrRScan	Возвращает указатель на последнее появление заданного символа в строке
AnsiStrScan	Возвращает указатель на первое появление заданного символа в строке

Стандартные процедуры и функции, выполняющие операции над строками	
AnsiToNative	Преобразовывает строку символов ANSI в набор символов драйвера другого языка
AnsiToNativeBuf	Преобразовывает строку символов ANSI в набор символов драйвера другого языка
AnsiUpperCase	Преобразовывает все символы строки в прописные буквы
AnsiUpperCaseFileName	Преобразовывает все символы строки в прописные буквы
AppendStr	Присоединяет строку к имеющейся строке
AssignStr	Выделяет память для новой Pascal-строки
Chr	Возвращает символ, соответствующий определенному значению кода ASCII
CompareStr	Сравнивает две строки указанной длины, учитывая регистр
CompareText	Сравнивает две строки указанной длины, не учитывая регистр
Concat	Выполняет операцию конкатенации нескольких строк
Copy	Возвращает из строки фрагмент строки
Delete	Удаляет из строки фрагмент строки
DisposeStr	Освобождает память для строки,
FmtLoadStr	Загружает строку из строкового ресурса программы
FmtStr	Форматирует ряд аргументов
Format	Форматирует ряд аргументов
FormatBuf	Форматирует ряд аргументов
FormatMaskText	Возвращает строку формата, которая используется в маске
GetLongHint	Возвращает вторую часть строки-подсказки (Hint)
GetShortHint	Возвращает первую часть строки-подсказки (Hint)
Insert	Вставляет фрагмент строки в строку
IntToHex	Преобразовывает целое число в строку, содержащую число в шестнадцатеричном представлении
IntToStr	Преобразовывает целое число в строку
IsDelimiter	Указывает, соответствует ли символ в строке символу из набора разделительных символов
IsPathDelimiter	Указывает, является ли байт в определенной позиции строки символом обратной наклонной черты
IsValidIdent	Возвращает значение True при условии, что указанная строка представляет собой допустимый идентификатор Object Pascal
LastDelirniter	Возвращает индекс последнего символа в строке, который соответствует символу из набора разделительных символов
Length	Возвращает динамическую длину строки
LoadStr	Загружает строку из ресурса EXE-файла
Lowercase	Преобразовывает все символы указанной строки в строчные буквы
NativeToAnsi	Преобразовывает строку в символы набора ANSI
NativeToAnsiBuf	Преобразовывает строку в символы набора ANSI
NewStr	Выделяет новой строке память в "куче"
OleStrToString	Копирует данные, полученные посредством OLE, в указанную Delphi-строку
OleStrToStrVar	Копирует OLE-строку в указанную Delphi-строку
Pos	Осуществляет поиск фрагмента строки в другой строке
QuotedStr	Возвращает строку, заключенную в кавычки
SetLength	Назначает строковой переменной динамическую длину
SetString	Устанавливает содержимое и длину строки
Str	Преобразовывает числовое значение в строку
StrAlloc	Выделяет память строке с завершающим нулем

Стандартные процедуры и функции, выполняющие операции над строками	
StrBufSize	Возвращает максимальное количество символов, которые могут быть записаны в строку с завершающим нулем, созданную посредством StrAlloc
StrCat	Копирует одну строку в конец другой строки
StrComp	Сравнивает две строки
StrCopy	Копирует одну строку в другую строку
StrDispose	Удаляет строку из "кучи"
StrECopy	Копирует строку и возвращает указатель на конец строки
StrEnd	Возвращает указатель на конец строки
StrFmt	Форматирует ряд аргументов
StrLCat	Присоединяет одну строку в конец другой строки и возвращает указатель на строку результата
StrLComp	Сравнивает две строки до определенного количества символов
StrIComp	Сравнивает две строки, не учитывая регистр
StrLCopy	Копирует символы из одной строки в другую
StrLen	Возвращает количество символов в строке
StrLFmt	Форматирует ряд аргументов и возвращает указатель на результат
StrLIComp	Сравнивает две строки до определенного количества символов, не учитывая регистр
StrLower	Преобразовывает все символы строки в строчные буквы
StrMove	Копирует символы из одной строки в другую
StrNew	Выделяет память в "куче" для указанной строки
StringOfChar	Возвращает строку с указанным количеством определенных символов
StrPas	Преобразовывает строку с завершающим нулем в Pascal-строку
StrPCopy	Преобразовывает Pascal-строку в строку с завершающим нулем
StrPLCopy	Копирует указанное количество символов Pascal-строки в строку с завершающим нулем
StrPos	Возвращает указатель на первое появление заданной подстроки в строке
StrRScan	Возвращает указатель на последнее появление заданного символа в строке
StrScan	Возвращает указатель на первое появление заданного символа в строке
StringToGUID	Преобразовывает строку в значение GUID
StrToInt	Преобразовывает строку в целое число
StrToIntDef	Преобразовывает строку в целое число. В случае ошибки возвращает значение по умолчанию
StrUpper	Преобразовывает все символы строки в прописные буквы
Trim	Удаляет все символы пробела и управляющие символы в начале и конце строки
TrimLeft	Удаляет все символы пробела и управляющие символы в начале строки
TrimRight	Удаляет все символы пробела и управляющие символы в конце строки
UniqueString	Делает строку уникальной
UpperCase	Преобразовывает все символы строки в строчные буквы
Val	Преобразовывает строку в число

Таблица А.15 - Список стандартных процедур и функций, выполняющих операции над текстовыми файлами

Стандартные процедуры и функции, выполняющие операции над файлами	
Append	Открывает файл для добавления данных
AssignPrn	Связывает файловую переменную с принтером
Eoln	Проверяет, является ли текущий символ в текстовом файле символом конца строки
Flush	Высвобождает буфер открытого для вывода файла
Read	В случае типизированных файлов считывает одну или несколько записей в переменную; в случае текстовых файлов считывает одно или несколько значений в одну или несколько переменных
Readln	Вызывает процедуру Read и переходит к началу следующей строки
SeekEof	Проверяет, является ли текущий символ символом конца файла
SeekEoin	Проверяет, является ли текущий символ символом конца строки
SetTextBuf	Выделяет текстовому файлу буфер ввода-вывода
Write	Для нетипизированных файлов записывает переменную в файл; для текстовых файлов записывает одно или несколько значений в файл
Writeln	Вызывает процедуру Write, после чего записывает в текущую позицию управляющий символ перевода строки

Таблица А.16 - Обзор стандартных процедур и функций, выполняющих операции над Unicode-строками

Стандартные процедуры и функции, выполняющие операции над Unicode-строками	
StringToWideChar	Преобразовывает ANSI строку в Unicode
WideCharLenToString	Преобразовывает последовательность Unicode символов в ANSI
WideCharToString	Преобразовывает Unicode-строку с завершающим нулем в ANSI-строку
WideCharToStrVar	Преобразовывает Unicode-строку в строку с однобайтовыми символами

Таблица А.17 - Список стандартных процедур и функций, выполняющих операции с переменными типа Variant

Стандартные процедуры и функции, выполняющие операции с переменными типа Variant	
VarArrayCreate	Создает массив переменных типа Variant
VarArrayDimCount	Возвращает размер массива переменных типа Variant
VarArrayHighBound	Возвращает наибольший индекс массива переменных типа Variant
VarArrayLock	Блокирует массив переменных типа variant и возвращает указатель на содержащиеся в этом массиве данные
VarArrayLowBound	Возвращает наименьший индекс массива переменных типа Variant
VarArrayOf	Создает одномерный массив переменных типа Variant и заполняет его
VarArrayRedim	Изменяет размер массива переменных типа Variant
VarArrayRef	Возвращает ссылку на массив переменных типа Variant
VarArrayUnlock	Разблокирует массив переменных типа Variant
VarAsType	Преобразовывает тип Variant в другой тип данных
VarCast	Осуществляет приведение переменной Variant к типу данных и сохранение результата в переменной
VarClear	Очищает значение переменной типа Variant
VarCopy	Копирует переменную Variant
VarFromDateTime	Возвращает переменную типа Variant со значением даты и времени
VarIsArray	Проверяет, является ли указанная переменная типа variant массивом
VarIsEmpty	Проверяет, имеет ли переменная типа Variant значение Unassigned
VarIsNull	Проверяет, имеет ли переменная типа Variant нулевое значение

Стандартные процедуры и функции, выполняющие операции с переменными типа Variant	
VarToDateTime	Преобразовывает значения переменной типа Variant в значение даты/времени
VarToStr	Преобразовывает значение переменной типа Variant в строку
VarType	Возвращает код типа переменной Variant

Таблица А.18 - Обзор различных стандартных процедур и функций

Различные стандартные процедуры и функции	
AddExitProc	Добавляет процедуру в список процедур выхода
Assert	Проверяет, возвращает ли булево выражение значение True
Beep	Выдает звуковой сигнал
BeginThread	Порождает новый процесс
CancelDrag	Прерывает текущую операцию Drag & Drop
CharsetToIdent	Определяет имя набора символов
Check	Определяет, указывает ли на ошибку передаваемое из BDE значение
CopyPalette	Возвращает новую цветовую палитру Windows, соответствующую переданной палитре
ColorToIdent	Возвращает символьное имя значения типа TColor
ColorToRGB	Преобразовывает значение типа TColor в RGB-значение цвета
ColorToString	Возвращает строку с именем значения TColor
EndThread	Завершает процесс
Exclude	Удаляет элемент из множества
FillChar	Заполняет заданное количество следующих друг за другом символов (байтов) переменной определенным значением
Hi	Возвращает старший байт переменной типа word или Smallint
Include	Вставляет элемент в множество
Lo	Возвращает младший байт переменной типа Word или Smallint
MessageDIg	Выводит диалоговое окно с сообщением
MessageDIgPos	Выводит диалоговое окно с сообщением в позицию с указанными координатами
Move	Копирует определенное количество байтов из одной области памяти в другую
ParamCount	Возвращает количество параметров командной строки, которые были указаны при вызове программы
ParamStr	Возвращает указанный параметр командной строки
Point	Создает структуру TPoint по заданной паре координат
Random	Возвращает случайное число
Randomize	Инициализирует генератор случайных чисел посредством задания значения переменной RandSeed (по умолчанию используется системное время)
Rect	Создает структуру TRect по заданному набору координат
ShowMessage	Выводит окно сообщения с кнопкой OK
ShowMessageFmt	Выводит окно форматированного сообщения с кнопкой OK
Slice	Возвращает часть массива
SizeOf	Возвращает количество байтов, которые занимает в памяти аргумент
Swap	Осуществляет обмен значений младшего значащего байта и старшего значащего байта переменной типа Word или Smallint
SysErrorMessage	Преобразовывает код ошибки Win32 API в строку
TypeInfo	Возвращает указатель на адрес, где содержится генерированная компилятором информация о типе (RTTI — Runtime type information)
UpCase	Преобразовывает строчные буквы в прописные

Таблица А.19 - Список стандартных процедур и функций, выполняющих операции с переменными типа Currency

Стандартные процедуры и функции, выполняющие операции с переменными типа Currency (денежные единицы)	
CurrToFMTBCD	Преобразовывает значение переменной типа currency в соответствующее значение формата BCD
CurrToStr	Преобразовывает значение переменной типа Currency в строку
CurrToStrF	Преобразовывает значение переменной типа Currency в форматированную строку
FMTBCDToCurr	Преобразовывает значение BCD в значение переменной типа Currency
FormatCurr	Форматирует значение переменной типа Currency
StrToCurr	Преобразовывает строку, представленную в виде числа с плавающей запятой, в значение переменной типа Currency

Таблица А.20 - Обзор стандартных процедур и функций, выполняющих операции с указателем и адресом

Стандартные процедуры и функции, выполняющие операции с указателем и адресом	
Addr	Возвращает адрес указанного объекта
Assigned	Проверяет, имеет ли указатель или процедурная переменная значение nil
Ptr	Преобразовывает адрес в значение типа Pointer

ДОДАТОК Б

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Использование объектно-ориентированного программирования являются хорошим решением при разработке крупных программных проектов. Чем проект объемнее и сложнее, тем больше выгоды вы получите при использовании объектно-ориентированной технологии программирования. Одним из наибольших преимуществ объектно-ориентированного программирования (ООП) является возможность многократного использования программного кода. Если вы, к примеру, создали класс, то можете порождать от него новые классы и изменять их свойства и функциональное назначение.

Класс-предок при этом остается без изменений, а соответствующий исходный код изменять не придется. Более того, свойства и методы объекта инкапсулированы в нем. Это значит, что никто и ничто извне не может ничего изменить в объекте, если такое изменение является недопустимым. В результате разработка приложения облегчается, а программисты могут использовать результаты работы коллег, не вникая в подробности.

Второе преимущество состоит в том, что приложение построено из объектов, которые являются отображениями реально существующих предметов или процессов. Рассмотрим компоненты Delphi еще раз под этим углом зрения. Если концепция объектно-ориентированного программирования нова для вас, вы столкнетесь с некоторыми незнакомыми понятиями. Пусть это вас не пугает. Чтобы изучить технологию ООП, надо прежде всего понять ее основные принципы.

В данной главе рассмотрены основные понятия объектно-ориентированного программирования. Разобравшись в них, вы сможете составить себе ясное представление о данной технологии.

В начале главы рассматривается операционная система Windows как среда разработки и выполнения программ, а также влияние, которое она оказывает на разработку приложений. В конце главы рассматриваются некоторые особенности реализации ООП в среде Delphi.

Б.1 Windows як середовище розробки та виконання програм

Программист, привыкший работать в MS-DOS, должен перестроиться, чтобы начать создавать программы для Windows.

В среде MS-DOS вы, наверное, привыкли к следующей ситуации.

- Программный код вашего приложения состоит из операторов, которые выполняются один за другим.
- Одновременно может выполняться только одна программа.
- В работающей программе что-то постоянно выполняется.
- Существует прямая связь между вашей программой и компьютером. Это значит, что управление клавиатурой, экраном и т.д. осуществляется непосредственно вашей программой. В среде Windows вы должны настроиться на следующую ситуацию.
- Код, который вы пишете, состоит из процедур обработки сообщений, которые Windows посылает вашему приложению.
- Windows фиксирует возникающие в программах и аппаратуре события и затем посылает соответствующие сообщения в вашу программу.
- Одновременно могут выполняться несколько программ. Эти программы делят между собой рабочую область памяти, процессор и другие ресурсы системы.
- Работающая программа находится в рабочей области памяти и ожидает сообщений от Windows, на которые она должна реагировать.
- Взаимодействие с аппаратным обеспечением происходит через графический интерфейс устройств (так называемый GDI). Программиста не должны заботить особенности устройств компьютера, этим занимается Windows.

Б.1.1 Керування програмою на основі повідомлень про події

Разработка приложений для среды Windows осуществляется иначе, чем создание программ для MS-DOS. Windows предлагает вам так называемую *управляемую событиями среду*, когда код программы исполняется как реакция на события. Управление программой на основе сообщений о событиях не ново, но не каждый программист знаком с этой концепцией. Она является неотъемлемой составной частью программирования для Windows. В основном понятие программирования по сообщениям используется для обозначения процесса взаимодействия между различными приложениями. Это означает, что Windows в состоянии управлять многозадачной системой таким образом, чтобы несколько программ могли выполняться одновременно. Иначе говоря, они могут совместно использовать память, процессор и другие ресурсы компьютера.

В качестве примера подобного процесса представьте, что созданная вами программа для работы с базой данных работает одновременно с текстовым редактором. Каждое из этих приложений выполняется в своем собственном окне. Представьте также, что пользователь работает с документом в текстовом редакторе и использует данные, которые он хочет импортировать из базы данных. Для этого ему необходимо в системе управления базами данных открыть файл, хранящий информацию, например, в dBASE-формате, откуда можно импортировать данные.

Пользователь производит щелчок мышью на значке приложения базы данных и этим самым порождает событие. Windows распознает это событие и запускает приложение, которое представлено данным значком.

Приложение реагирует на это сообщение и открывает файл dBASE, откуда пользователь может импортировать данные.

После того как сообщение будет обработано приложением, Windows вновь получит возможность управлять процессором и информацию о том, что сообщение обработано — или о том, что Windows должна сделать еще что-то.

Если пользователь, например, введет данные в приложение базы данных, они должны отобразиться на экране. Приложение передаст необходимые сообщения и инструкции Windows, а Windows отобразит данные. Таким образом, программа работает независимо от аппаратного обеспечения.

Это — главный принцип, на котором основана система сообщений. Как видите, это система с обратной связью. Приложение Windows должно быть в состоянии обрабатывать сообщения от Windows, а также генерировать и посылать свои сообщения.

При этом Windows является как бы посредником между пользователем, программой и устройствами системы (Hardware). Все это приводит к тому, что программирование в Windows заключается главным образом в организации обработки сообщений.

Delphi — это среда разработки, которая до известной степени освобождает программиста от скучной, рутинной работы по управлению обработкой сообщений и оставляет ему больше возможностей для творчества.

Б.1.1.1 Створення та обробка повідомлень

Windows создает входное сообщение для каждого входного события, генерируемого пользователем с помощью мыши ли клавиатуры. Windows сохраняет входные данные в *очереди системных сообщений*. Затем эти сообщения посылаются в *очередь сообщений приложения*.

Сообщение приложению от Windows формируется путем создания *записи сообщения* (Message Record) в очереди сообщений. При этом соблюдается принцип FIFO — First In, First Out (первый зашел — первый вышел). Некоторые сообщения от Windows посылаются непосредственно в окно приложения и не ставятся в очередь.

Это так называемые *внеочередные сообщения* (UM, Unqueued Messages). Типичное UM — это сообщение, которое касается только окна приложения. Хотя большинство сообщений

порождается Windows, приложение также может создавать собственные сообщения, помещать их в свою очередь сообщений и посылать другим приложениям.

Для обработки сообщений главная программа приложения использует непрерывный цикл, так называемый *главный цикл сообщений*.

Этот цикл содержит некоторое количество функций, предназначенных для обработки сообщений. Как правило, выход из этого цикла совершается лишь тогда, когда поступает сообщение, которое должно завершить программу.

Главный цикл сообщений начинается с вызова функции, которая просматривает очередь сообщений (GetMessage()).

В случае, когда пользователь нажимает клавишу, необходимо послать еще одно сообщение. Это сообщение содержит *виртуальный код клавиши* (Virtual Key Code), который, хотя и констатирует нажатие клавиши, но не сообщает непосредственно значение символа клавиши. Поэтому для определения символа сначала вызывается функция (TranslateMessage()). Затем вызывается функция, которая классифицирует принятое сообщение (DispatchMessage()). Каждый объект имеет свою процедуру, в которой определены действия для любого из возможных сообщений. Когда эта процедура выполнена, управление передается в начало цикла. Таким образом, сообщения в этом цикле выбираются и обрабатываются последовательно.

Если очередь пуста, ожидается новое сообщение. В этом случае говорят, что приложение находится в режиме ожидания. По-английски это состояние называется Idle. Во время ожидания контроль над системой передается в Windows, благодаря чему другие приложения получают возможность обрабатывать сообщения.

Б.1.1.2 Зв'язок між подіями та додатками

Каким образом Windows узнает, для какого приложения предназначено данное сообщение о событии? Каждое приложение выполняется в своем собственном окне, имеющем уникальный дескриптор — Handle, а т.к. в записи сообщения содержится информация о дескрипторе окна, которому предназначено данное сообщение, то Windows "знает", по какому "адресу" его следует отослать. Поэтому в Delphi не надо заботиться об этом; Windows самостоятельно осуществляет управление формами и компонентами.

Б.1.1.3 Обробка стандартних повідомлень

Приложение должно быть в состоянии обработать любое сообщение. Практически для каждого события Windows имеет стандартную процедуру обработки. В приложении должно быть указано, что при возникновении события XY, для которого в программе нет специального текста, должна выполняться стандартная процедура Windows.

Большинство стандартных сообщений программы обрабатываются автоматически, поскольку все объекты Delphi имеют встроенный механизм — *процедуру обработки сообщений* (Message Handler).

В каждом из компонентов Delphi определен механизм управления сообщениями, который переводит все сообщения Windows, посылаемые определенному объекту, в вызовы методов.

Б.2 Основні поняття

Как уже ясно из названия, сущность объектно-ориентированного программирования заключается в использовании объектов. ООП представляет собой расширение традиционных языков программирования (например, C или Pascal) новым структурированным типом данных — классом.

Наиболее существенные отличительные черты ООП — это применение классов, наследования (inheritance), инкапсуляции (encapsulation) и полиморфизма (poly -много, morphe — вид, форма). Ниже эти основные понятия ООП разъясняются подробнее.

Б.2.1 Класи

Класс — это абстрактное понятие, сравнимое с понятием *категория* в его обычном смысле.

По определенным свойствам любого элемента определенной категории можно установить, что он принадлежит к этой категории. Сама категория определяется общими свойствами, которые имеют все экземпляры этой категории.

Это можно пояснить на примере музыкальных инструментов. Музыкальные инструменты делятся на следующие категории: духовые, ударные и струнные.

Все эти категории принадлежат к категории музыкальных инструментов. В свою очередь, категория музыкальных инструментов входит в категорию инструментов. На рис. Б.2.1 эта структура категорий графически представлена в виде дерева.

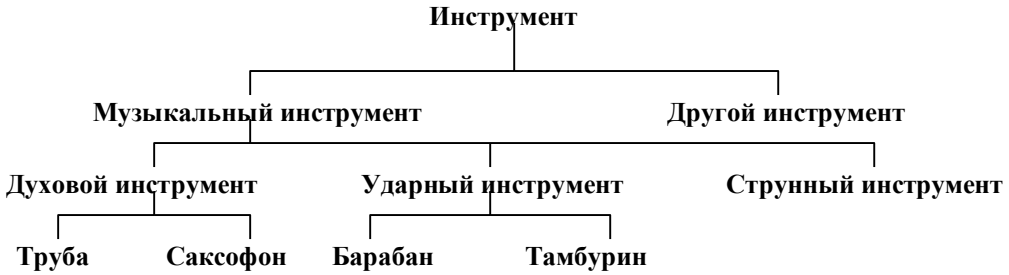


Рисунок Б.2.1 – Дерево категорий

Музыкальные инструменты имеют общие свойства, но каждый инструмент сам по себе обладает особыми свойствами, которые определяют его назначение и отличают его от других инструментов. По тому же принципу можно описать и классы в ООП. Определенный музыкальный инструмент некоторой категории, например моя труба или ваша труба, является объектом. Категория, к которой этот инструмент принадлежит, — это класс.

Класс в ООП — это абстрактный тип данных, который включает не только данные, но и функции и процедуры.

Функции и процедуры класса называются методами и содержат исходный код, предназначенный для обработки внутренних данных объекта данного класса.

Б.2.2 Наслідування

Классы содержат *данные* и *методы*. В ООП методы и данные одного класса могут передаваться другим классам, т.е. объекты могут наследовать свойства друг друга. Класс, наследующий свойства другого класса, обладает теми же возможностями, что и класс, от которого он порожден. Этот принцип называется *наследованием* (inheritance). Порожденный класс называется *потомком* (descendant), а тот, от которого он порожден — *предком* (ancestor). Благодаря новым свойствам, которыми дополняется потомок, порожденный класс может обладать большими возможностями, чем его предок.

В примере дерева категорий музыкальных инструментов класс **Труба** происходит непосредственно от класса **Духовые инструменты**. Таким образом уже определено, что в трубу можно дуть, и это ее свойство не надо переопределять заново.

Тот же принцип соблюдается и для свойства музыкальных инструментов издавать музыкальные звуки. Это свойство класса **Музыкальные инструменты** перенесено на класс **Труба** через его прямого предка, класс **Духовые инструменты.**

Механизм наследования обеспечивает возможность многократного применения программного кода. Таким образом, классы могут быть представлены в виде иерархии. Библиотека VCL (Visual Component Library) в Delphi и является такой иерархической системой классов.

Б.2.3 Інкапсуляція

Совмещение данных и методов их обработки в одном объекте называется *инкапсуляцией* (encapsulation). Методы объекта определяют способы изменения данных.

Способность духового инструмента издавать звуки обусловлена продуванием воздуха через мундштук. Для духового инструмента характерно наличие мундштука, в который надо дуть. Иными словами, мундштук можно использовать только определенным способом, который относится к духовому инструменту. Никакой другой способ не годится.

Представьте, что вам надо написать программу, которая выполняла бы дуэт духового и струнного инструментов. Для этого определите классы **Духовой инструмент** и **Струнный инструмент**. Для класса **Духовой инструмент** определите, что имеется мундштук и что в него надо дуть, чтобы получить звук.

Для класса **Струнный инструмент** определите, что по струнам надо ударять, чтобы получить звук.

Оба класса уже способны играть музыку, но это их свойство было унаследовано от их предка. Они унаследовали метод PlayMusic, который объявлен и реализован как метод класса **Музыкальный инструмент**.

Таким образом, этот метод уже не нужно создавать, и вам не надо знать код реализации метода, чтобы использовать его в двух новых классах. Способ, которым реализована возможность играть музыку, не важен. Этот принцип сокрытия информации (information hiding) характерен для инкапсуляции и существенно облегчает написание больших и стабильно работающих приложений.

Если класс был грамотно сконструирован и тщательно проверен, он может многократно использоваться в различных приложениях. В примере с музыкальными инструментами это означает, что каждый класс обладает свойствами класса Музыкальный инструмент, а способы создания звука заключены внутри него. Эти способы невидимы и недоступны за пределами класса.

Если духовой инструмент получает задание играть, то этот инструмент "знает", что для этого надо дуть в его мундштук, так как это определено в его классе.

Струнный инструмент, который получает то же задание, не может использовать мундштук духового инструмента, чтобы дуть на свои струны. Такого не может случиться еще и потому, что мундштук — это часть класса **Духовой инструмент**, а не класса **Струнный инструмент**. Это подразумевает, что оба класса ничего не знают друг о друге. Они полностью разделены, и им не известны спецификации и свойства друг друга. **Духовой инструмент** закрыт для любых попыток других классов использовать его мундштук. Также и струны **Струнного инструмента** и способ их использования заключены внутри него самого. Объект *закрыт*, т.е. окружение не может случайно изменить этот объект.

Смысл этой закрытости в том, что вы не обязательно должны знать, как, например, труба издает звук. Тот факт, что для получения звука нужен мундштук, скрыт в глубине класса **Духовой инструмент**. Не имеет значения, с каким мундштуком это происходит и как. Это подробности, которые учитываются в самом объекте и не имеют значения для его окружения. Вам необходимо лишь вызвать функцию PlayMusic.

Б.2.4 Поліморфізм

Другая важная концепция ООП — *полиморфизм*. Это означает, что один и тот же метод выполняется по-разному для различных объектов. Например, метод класса **Музыкальный инструмент** — PlayMusicForAnOrchestra — может быть определен как общий метод, который может использоваться с любой категорией музыкальных инструментов. Этот метод написан таким образом, что не важно, какой именно инструмент получает задание играть, однако для классов, описывающих конкретные инструменты, данный метод должен быть *переопределен*

(override), что даст возможность определить конкретные действия, учитывающие особенности данного инструмента

Б.3 Реалізація ООП в Delphi

Преимущество применения объектно-ориентированных методов при разработке приложений для Windows состоит в том, что вы используете предварительно созданные объекты — элементы управления Windows (например, окна, кнопки и т.д.) Кроме того, вы можете использовать их в качестве предков для новых, определенных вами компонентов. Благодаря этому ваши приложения приобретают характерный для Windows вид и вам не нужно заново "изобретать колесо". В то же время вы можете производить изменения по своему вкусу. В Delphi это можно выполнить значительно проще, чем в других языках программирования, например в C++.

Б.3.1 Новий тип даних: клас

Концепция объектно-ориентированного программирования предполагает использование нового типа данных — *класс*. Тип "класс" принадлежит к совокупности известных в Delphi структурированных типов: множество, массив, запись и класс. Особенность типа "класс" состоит в том, что он содержит методы и свойства. Это значит, что класс описывает группу данных и одну (или более) процедуру или функцию, которая имеет доступ к этим данным. Процедуры и функции называются *методами*. Тип "класс" — это структура данных, состоящая из некоторого количества элементов:

- полей;
- методов;
- свойств.

Поля содержат данные определенного типа. Методы — это функции и процедуры, выполняющие определенные действия. Свойства — это поля данных, которые влияют на поведение объекта. Они служат для описания объекта и отличаются от обычных полей тем, что присвоение им значений связано с вызовом методов.

Б.3.2 Об'явлення типів

Каждый новый класс в Delphi должен быть объявлен. Для этого используется зарезервированное слово class. Объявление определяет функциональные возможности класса. В Delphi-версии языка Object Pascal новый класс объявляется следующим образом:

TNew = **class**(TOld);

Для объявления классов в модуле отведен особый раздел, который так и называется: *раздел объявления типов.*

Классы должны быть объявлены на уровне программы или на уровне модуля и не могут быть объявлены внутри процедуры или функции.

Зарезервированное слово class используется для объявления класса или метода класса.

Объявление поля состоит из идентификатора, который обозначает поле, и типа данных поля. Объявление метода состоит из заголовка процедуры или функции. В определении свойства указывается идентификатор свойства и методы его использования.

Б.3.3 Об'єкти, класи та екземпляри

Класс и объект — часто употребляемые термины ООП. Однако иногда эти термины употребляются неправильно. В настоящее время в ООП принято следующее определение этих понятий.

Класс — определенный пользователем тип данных, который обладает внутренними данными и методами в форме процедур или функций и обычно описывает родовые признаки и способы поведения ряда очень похожих объектов. Объект является *экземпляром класса*. Предварительно определенные объекты, используемые в программе (такие как, например, компоненты Delphi), — это в действительности экземпляры классов.

В языке программирования Delphi экземпляр класса реализуется переменной определенного типа класса. Ниже приведен пример объявления объектов.

Пример Б.3.1 – Объявление класса

```
type
TForml = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  CloseBtn: TBitBtn
  OKBtn: TBitBtn
end;
var
  Forml: TForml;
```

В объявлении типа определен новый класс — TForml, наследуемый от класса TForm, содержащегося в VCL. На это указывает зарезервированное слово class. Данный тип содержит указатели на компоненты, которые были помещены в форму: два компонента Label — объекты типа TLabel (или, иначе говоря, экземпляры класса TLabel) и два экземпляра класса TBitBtn.

Для определения экземпляра нового класса объявлена переменная Forml.

После того как сделаны все объявления, можно очень просто создать и инициализировать новые экземпляры классов (или объекты).

Итак, можно сделать следующие выводы.

- ООП отличается от прежних методов программирования возможностью создания новых объектов.
- Экземпляры классов могут вести себя различным образом. Каждый объект располагает копией полей данных типа класса, но разные объекты имеют различные поля и методы.
- Переменная может содержать ссылку на экземпляр класса (объект). Переменная содержит не сам объект, а указывает на зарезервированную для него область памяти. Как и переменные-указатели, несколько переменных-объектов могут ссылаться на один и тот же объект. Переменная-объект может иметь и нулевое значение; это значит, что она в данный момент ни на что не указывает.

Б.3.4 Область видимості

Подробное описание понятия инкапсуляции связано с понятием *области видимости идентификатора*. Область видимости идентификатора (имени переменной, процедуры, функции или типа данных) - это часть программного кода, в которой возможен доступ к этому идентификатору.

Область видимости идентификатора компонента, объявленного в описании класса, простирается от его объявления до конца определения класса, а также распространяется на все потомки этого класса и на все блоки реализации методов класса.

Область видимости идентификатора компонента зависит от *атрибута видимости* раздела, в котором объявлен этот идентификатор. В Delphi используется пять атрибутов видимости (называемых также директивами): published, public, protected, private и automated.

В объявлениях типов классов имеются разделы частных (private) и общих (public) объявлений. В разделе частных объявлений размещаются поля данных и методы, недоступные за пределами модуля, содержащего объявление данного класса. Данные, описанные в этом разделе, могут обрабатываться только путем вызова методов внутри класса, а также внутри

данного модуля. За пределами класса все его частные элементы неизвестны и считаются несуществующими.

Поля данных и методы, объявленные в разделе общих объявлений класса, доступны для всех процедур, программный код которых расположен в области видимости данного объекта. В разделе общих объявлений типа класса должны быть объявлены поля данных и методы, к которым будут иметь доступ методы объектов других модулей.

С атрибутом видимости protected объявляются те элементы, к которым за пределами данного модуля могут иметь доступ только методы классов, порожденных от данного класса.

Директива published похожа на другие атрибуты видимости (private, public и protected) тем, что она может встречаться только в объявлении типа класса. Опубликованное (published) поле или метод может использоваться не только во время выполнения программы, но и во время ее разработки. Все компоненты в палитре компонентов Delphi располагают published-интерфейсом, который используется в первую очередь инспектором объектов. Правила видимости для директивы published — те же, что и для public.

Различие между *общими* (public) и *опубликованными* (published) элементами состоит в том, что во время выполнения программы можно получить информацию о типах (RTTI — Run-time type information) опубликованных элементов класса. С помощью этой информации в приложении можно динамически определить и использовать поля и свойства любого, в том числе и неизвестного, типа класса.

Директива automated введена только для совместимости с Delphi 2.0. Новый класс TAutoObject в Delphi 4 эту директиву уже не использует.

Б.4 Робота з об’єктами, створеними розробником

Б.4.1 Об’явлення нового типу

В приведенном ниже тексте программы определяется новый тип класса, предназначенного для управления и анализа данных, представляющих собой последовательность результатов измерений. Тип получает имя TSeries. Тип TSeries объявляется как непосредственный потомок класса TObject.

Класс может быть объявлен только в разделе объявлений типов модуля, который начинается с зарезервированного слова type и заканчивается на объявлении другого раздела.

Новый класс всегда должен объявляться при помощи слова class.

Так как TObject является предком всех классов в Delphi, его не обязательно указывать при объявлении класса, порожденного от него:

```
type
  TSeries = class
end;
```

Можно использовать и другое объявление:

```
type
  TSeries = class(TObject)
end;
```

На основании вышеизложенного можно сделать следующие выводы.

- TObject является базовым классом для всех классов Delphi.
- TObject содержит конструктор TObject.Create, который создает динамический экземпляр класса и инициализирует все поля данных.
- TObject содержит также деструктор TObject.Destroy, который освобождает память, зарезервированную конструктором Create и таким образом уничтожает экземпляр класса.
- TObject — это класс, содержащий набор методов, которые могут применяться в каждом экземпляре класса.

Б.4.1.1 Об'явлення полів даних нового типу

Пусть по результатам серии экспериментов необходимо получить следующую информацию: число измерений, число определенных материалов в серии, какие результаты лежат в пределах допуска и какие вне его; кроме того, даны названия испытываемых материалов.

Поэтому в качестве полей данных нового объекта определяются следующие переменные: два поля целочисленного типа (NumberOfSamples и NumberOfMaterials), логическое поле (IsStandard или IsNotStandard), поле типа TStringList (MaterialName). В объявлении поля указывается идентификатор, обозначающий поле и его тип.

Объявление типа выглядит следующим образом.

Пример Б.4.1 – Объявление класса

```
type
  TSeries = class(TObject)
public
  {Поля}
  NuraberOfSamples, NumberOfMaterials: Integer;
  IsStandard, IsNotStandard: Boolean;
  MaterialName: TStringList;
end;
```

Эти поля данных объявлены в разделе public объявлений классов, так как они должны быть общедоступны

Б.4.1.2 Об'явлення методів

Теперь следует обеспечить доступ к данным. Чтобы обеспечить доступ к данным, надо написать программный код, обрабатывающей эти данные — так называемые *методы класса*.

Для этого надо знать, сколько результатов измерений входят в серию, сколько разновидностей материалов испытывается и как они называются. Кроме того, надо знать, идет ли речь о допустимых или о недопустимых значениях.

Чтобы определить название того или иного материала, создается процедура (GetNames), а для решения остальных задач — функции. Код реализации этих методов помещается в implementation-секции модуля. В объявлении метода определяется заголовок процедуры или функции, который помещается в объявлении класса.

Теперь описание класса, содержащего названные выше методы, выглядит следующим образом.

Пример Б.4.2 – Объявление класса

```
type
  TSeries = class(TObject);
public
  {Поля данных}
  NumberOfSamples, NumberOfMaterials: Integer;
  IsStandard, IsNotStandard: Boolean;
  MaterialName: TStringList;
  {Методы}
  function CountSamples: Integer;
  function CountMaterials: Integer;
  function Standard: Boolean;
  procedure GetNames;
end;
```

Б.4.1.3 Визначення нового класу

Теперь определим новый класс, порожденный от класса TSeries:

```
type
  TStandard = class (TSeries)
```

end;

В этом объявлении типа используется так называемая *транзитивность* наследования, т.е. свойства предка переходят к потомку. TSeries наследует все свойства TObject, TStandard наследует все элементы TSeries, и, таким образом, класс TStandard наследует элементы класса TObject через TSeries.

Итак, TStandard имеет двух предков и содержит все свойства обоих предков. Как уже было упомянуто при объявлении класса TSeries, в добавок к унаследованным можно определить новые свойства.

При этом надо учесть, что потомок не может удалить объявление какого-либо поля, свойства или метода своего предка.

То, что наследует потомок, можно только редактировать и расширять, но не удалять.