МІНІСТЕРСТВО ОСВІТИ і НАУКИ УКРАЇНИ
ХАРКІВСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ МІСЬКОГО ГОСПОДАРСТВА

Мови програмування

Короткий курс програмування в середовищі Delphi

(для студентів заочної форми навчання
спеціальності 7.050201 "Менеджмент організацій"
спеціалізації "Інформаційні системи в менеджменті")

Харків - ХНАМГ - 2004

Короткий курс програмування в середовищі Delphi (для студентів заочної форми навчання спеціальності 7.050201 “Менеджмент організацій” спеціалізації “Інформаційні системи в менеджменті”). Укл. Г.А. Мірошниченко. - Харків: ХНАМГ, 2004. - 81 с.,іл.

Укладач: Г.А. Мірошниченко
Рецензент: к.т.н. І.Т. Карпалюк

Рекомендовано
кафедрою Інформаційних систем та
технологій в міському господарстві
протокол № 11 від 26.03.2004р.

Зміст

Вступ

Даний курс призначений для вивчення дисципліни “Мови програмування” студентами, які навчаються за фахом “Менеджмент організацій” спеціалізації “Інформаційні системи в менеджменті”.

Мета вивчення дисципліни - придбання студентами знань про алгоритмізацію, мови програмування, особливості візуального та об'єктно-орієнтованого програмування, одержання основних практичних навичок створення програмних продуктів.

Після знайомства з теоретичним курсом, студенти мають оволодіти основними принципами алгоритмізації, основними поняттями мови Object Pascal - алфавіт, словник мови, структура програми, типи даних, оператори, основами візуального програмування, основами об'єктно-орієнтованого програмування.

Після оволодіння практичною частиною курса, студенти повинні вміти: використовувати прості й структуровані оператори, застосовувати підпрограми, модулі, візуальні компоненти при створенні програм.

Розділ І
ЗНАЙОМСТВО ІЗ СЕРЕДОВИЩЕМ DELPHI

Середовище Delphi візуально реалізується декількома одночасно розкритими на екрані вікнами. Вікна можна пересувати по екрану таким чином, щоб вони частково або цілком перекривали одне одного. Кожне вікно несе в собі деяку функціональність, тобто призначено для вирішення певних задач.

Після запуску Delphi на екрані з'являються найбільш важливихі вікна Delphi:

1 - головне вікно;

2 - вікно форми;

3 - вікно Дерева об'єктів (Object Tree View);

4 - вікно Інспектора об'єктів;

5 - вікно коду програми.

Розташування і розміри вікон можна змінювати вручну. При роботі зручно використовувати клавішу F12, що працює як перемикач, почергово показуючи вікно форми або вікно коду програми.

Головне вікно

Головне вікно здійснює основні функції керування проектом створюваної програми. Воно завжди присутнє на екрані і займає його верхню частину. Не намагайтеся його розгорнути на весь екран: навіть у максимізованому стані його розміри і положення майже не відрізняються від звичайних. Пов'язано це з функціональністю головного вікна: з одного боку, воно несе в собі елементи, що завжди повинні бути під рукою у програміста, з другого - вікно не повинне віднімати в інших вікон Delphi значного простору екрана. Мінімізація головного вікна призводить до зникнення з екрана інших вікон Delphi (ці вікна з'являться, як тільки будуть відновлені розміри головного вікна), а його закриття означає закінчення роботи програміста із системою програмування.

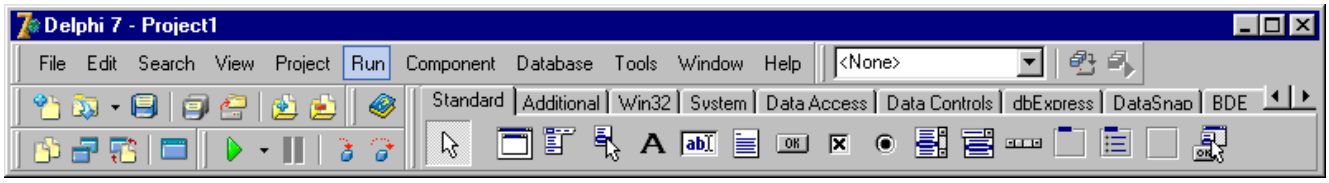

Рис. 1 - Головне вікно Delphi

Всі елементи головного вікна розташовуються на спеціальних панельках, у лівій частині яких знаходяться кнопки керування, що дозволяють за допомогою миші перетягувати панельки з розміщеними на них елементами. Будь-яку панельку (крім головного меню) можна забрати з вікна (зробити її невидимою) або “пустити плавати” по екрану в окремому вікні. Для цього потрібно лише “стягнути” панельку за допомогою миші за межі головного вікна.

Для зміни складу показаних на панельці кнопок потрібно попередньо клацнути по ній правою кнопкою миші. У вікні допоміжного меню, що з'явилося після цього, перераховані назви всіх панельок і вказаний їхній статус (відзначені прапорцями панельки, що видні в головному вікні; якщо позначку прибрати, панелька зникне). Після вибору Customize (Настроювання) з'явиться вікно настроювання). Тепер можна “стягувати” з панельок непотрібні кнопки, вибирати зі списку у вікні commands (закладка Commands) потрібні кнопки і перетаскувати їх на екран.

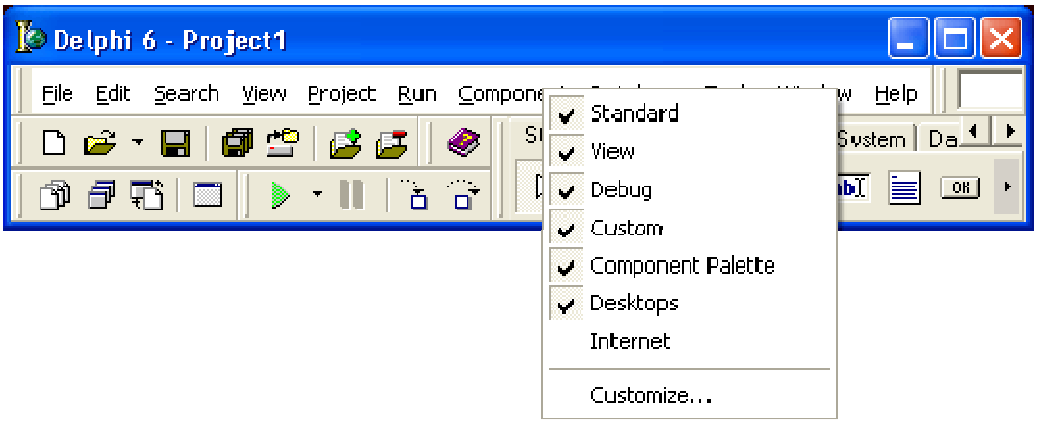

Рис. 2- Настроювання інструментальних панельок: вікно допоміжного меню з назвами всіх панелей.

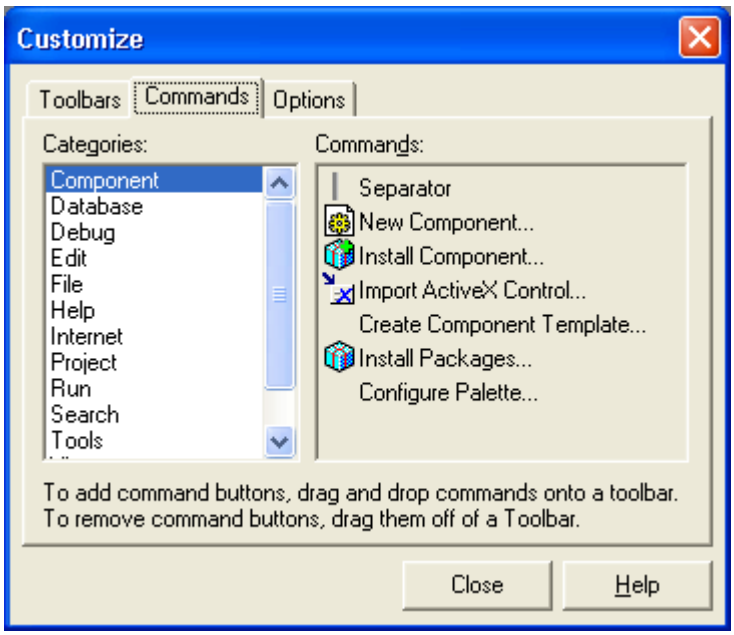

Рис. 3 - Настроювання інструментальних панельок: вікно настроювання з обраною закладкою Command

У головному вікні розташовується:

- головне меню Delphi;
- набір піктографічних командних кнопок;
- палітра компонентів.
- Головне меню

Головне меню містить усі необхідні засоби для керування проектом. Всі опції головного меню являють собою опції-заголовки, що відкривають доступ до випадаючих меню другого рівня.

Рис. 4 - Головне меню

Піктографічні кнопки

Піктографічні кнопки відкривають швидкий доступ до найбільш важливих опцій головного меню. За функціональною ознакою вони розділені на сім груп. Кожна група займає окрему панельку. Розглянемо команди, реалізовані стандартним набором піктографічних кнопок.

Група Standard

 Відкриває доступ до Репозиторія Об'єктів. Еквівалент опції File | New | Other

 Відкриває існуючий файл. Еквівалент опції File | Open File

 Зберігає файл на диску. Еквівалент опції File|Save File (клавіші швидкого доступу Ctrl-S)

 Зберігає усі файли проекту. Еквівалент опції File | Save All

 Відкриває створений раніше проект програми. Еквівалент опції File | Open Project (клавіші швидкого доступу Ctrl-F11)

 Додає новий файл до проекту. Еквівалент опції Project | Add to project (клавіші швидкого доступу Shift-F11)

 Видаляє файл із проекту. Еквівалент опції Project | Remove from Project

Група View

 Вибирає модуль зі списку модулів, зв'язаних з поточним проектом. Еквівалент опції View | units (клавіші швидкого доступу Shift-F12)

 Вибирає форму зі списку форм, зв'язаних з поточним проектом. Еквівалент опції View | Forms (клавіші швидкого доступу Ctrl-F12)

 Переключає активність між вікном форми і вікном коду програми. Еквівалент опції View | Toggle Form/Unit (клавіша швидкого доступу F12)

 Створює нову форму і додає її до проекту. Еквівалент опції File | New | Form

Група Debug

 Компілює і виконує програму. Еквівалент опції Run | Run (клавіша швидкого доступу F9)

 Реалізує паузу в роботі налагоджуваної програми. Еквівалент опції Run | Program Pause

 Здійснює покрокове трасування програми з простежуванням роботи викликаних підпрограм. Еквівалент опції Run | Trace into (клавіша швидкого доступу F7)

Здійснює покрокову трасіровку програми, але не простежує роботу викликаних підпрограм. Еквівалент опції Run | Step Over (клавіша швидкого доступу F8)

Група Custome

Відкриває доступ до вбудованої довідкової служби. Еквівалент опції Help | Delphi Help

Група Desktops

<None> Список вибору можливих варіантів настроювання інших вікон Delphi

Зберігає поточне настроювання вікон Delphi

Вибирає настроювання вікон, що відповідає режиму налагодження

Група Internet

Починає створення нового програмного засобу за технологією WebSnap (для Інтернет)

Створює нову сторінку програмного засобу WebSnap

Створює новий модуль програмного засобу WebSnap

Палітра компонентів

Палітра компонентів займає праву частину головного вікна і має закладки, що забезпечують швидкий пошук потрібного компонента. Під компонентом розуміється якийсь функціональний елемент, що містить визначені властивості і розміщується програмістом у вікні форми. Компоненти являють собою елементи, з яких конструюється видиме зображення, що створюється працюючою програмою.

Рис. 5 - Палітра компонентів.

Як і панель кнопок, палітра компонентів може налагоджуватись. Для цього використовується спеціальний редактор, вікно якого з'являється на екрані

після клацання правою кнопкою миші на будь-якій піктограмі в палітрі компонентів і вибору опції properties (Властивості)

Найчастіше використовувані компоненти розглянуті в Додатку.

Вікно форми

Вікно форми являє собою проект Windows-вікна майбутньої програми. Спочатку це вікно порожнє, точніше, воно містить стандартні для Windows інтерфейсні елементи - кнопки виклику системного меню, максимізації, мінімізації та закриття вікна, смугу заголовка і рамку, що окреслює. Уся робоча зона вікна звичайно заповнена точками координатної сітки, що служить для впорядкування розташованих на формі компонентів.

Значну частину часу програміст зайнятий заповненням вікна форми інтерфейсними елементами. Власне, саме в цьому процесі наповнення форми і полягає головна особливість візуального програмування. Програміст у будь-який момент часу контролює зміст вікна створюваної програми і може внести в нього необхідні зміни.

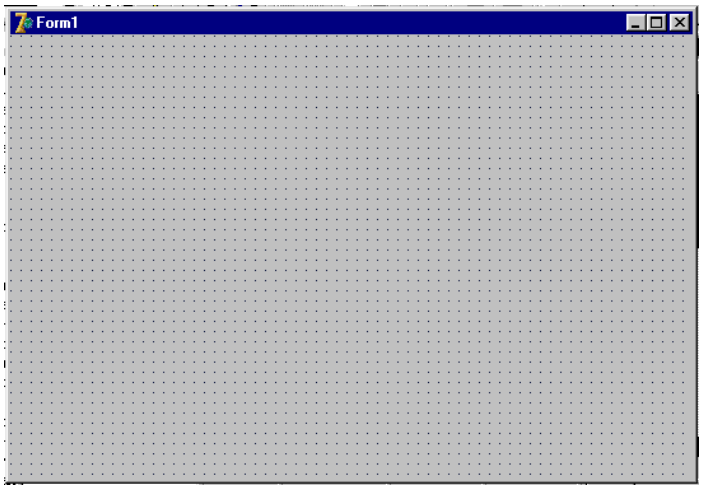

Рис. 6 - Вікно форми

Вікно дерева об'єктів

Це вікно призначене для візуального відображення зв'язків між окремими компонентами, розташованими на активній формі або в активному модулі даних. Клацання по будь-якому компонету в цьому вікні активізує відповідний

компонент у вікні форми і відображає властивості цього компонента у вікні Інспектора об'єктів. Подвійне клацання приводить до спрацьовування механізму Code Insight, що вставляє у вікно коду заготовку для обробника події OnClick. Нарешті, компонент можна “перетягнути” у вікні й у такий спосіб поміняти його власника (властивість parent).

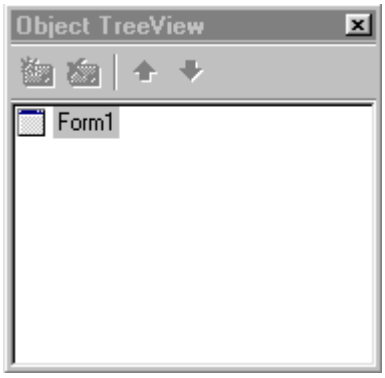

Рис. 7 - Вікно дерева об'єктів

Вікно інспектора об'єктів

Будь-який розташований на формі компонент характеризується деяким набором параметрів: положенням, розміром, кольором і т.д. Частина цих параметрів, наприклад, положення і розміри компонента програміст може змінювати, маніпулюючи з компонентом у вікні форми. Для зміни інших параметрів призначене вікно Інспектора об'єктів. Це вікно містить дві сторінки - Properties (Властивості) і Events (Події). Сторінка properties використовується для встановлення потрібних властивостей компонента, сторінка Events дозволяє визначити реакцію компонента на ту або іншу подію. Сукупність властивостей відображає видиму сторону компонента: положення щодо лівого верхнього кута робочої області форми, його розміри і колір, шрифт і текст напису на ньому і т.п.; сукупність подій - його поведінкову сторону: чи буде компонент реагувати на клацання миші або на натискання клавіш, як він буде поводитися в момент появи на екрані або в момент зміни розмірів вікна і т.п.

Кожна сторінка вікна Інспектора об'єктів являє собою двохколончату таблицю, лівий стовпчик якої містить назву властивості або події, а правий -

конкретне значення властивості або ім'я підпрограми, що обробляє відповідну подію.

Рядки таблиці вибираються клацанням миші і можуть відображати прості або складні властивості. До простих відносяться властивості, обумовлені єдиним значенням - числом, рядком символів, значенням True (Істина) або False (Неправда) і т.п. Наприклад, властивість Caption (Заголовок) представляється рядком символів, властивості Height (Висота) і Width (Ширина) - числами, властивість Enabled (Приступність) - значеннями True або False. Складні властивості визначаються сукупністю значень. Ліворуч від імені таких властивостей вказується значок “+”, а клацання мишею по цьому символу приводить до розкриття списку складової складної властивості. Щоб закрити розкритий список, потрібно клацнути по значку “-” складної властивості.

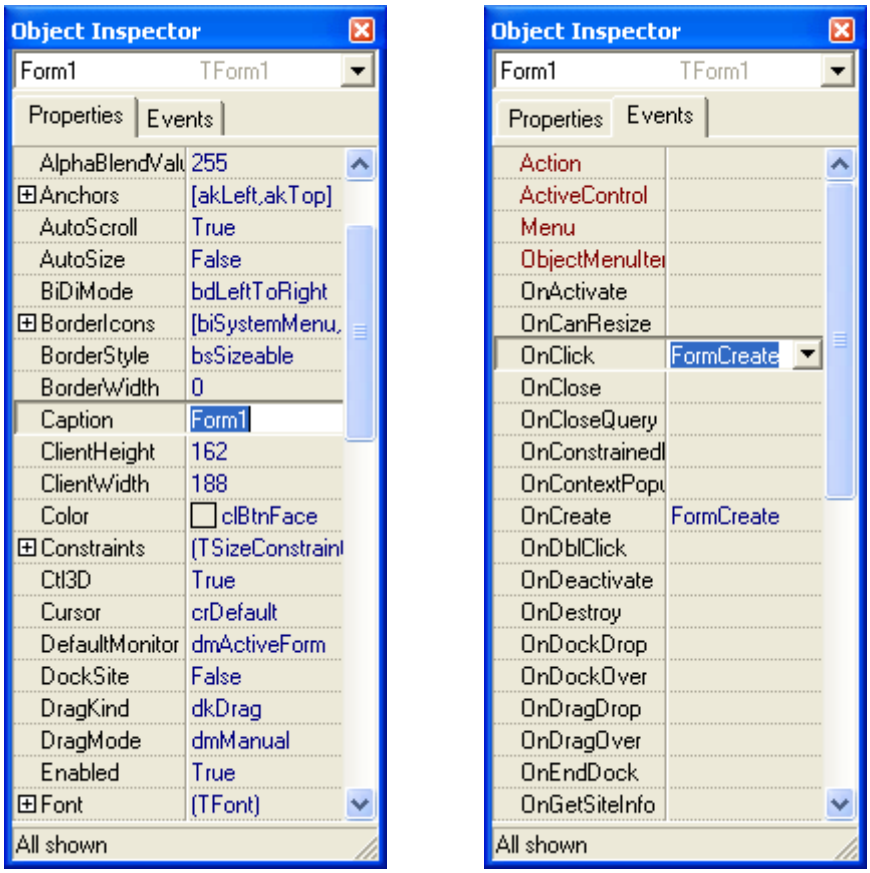

Рис. 8 - Вікно Інспектора об'єктів. Сторінка Properties (Властивості) і сторінка Events (Події)

У верхній частині вікна Інспектора об'єктів розташовується розгортаючийся список усіх вміщених на форму компонентів. Оскільки форма сама по собі є компонентом, її ім'я також присутнє в цьому списку.

У локальному меню вікна, що з'являється після клацання по ньому правою кнопкою миші, є ряд опцій, які дозволяють настроїти вікно.

Якщо Ви випадково або навмисно зробите вікно невидимим, натисніть F11 або виберіть опцію View | Object inspector, щоб воно знову з'явилося на екрані.

Вікно коду програми

Вікно коду призначене для створення і редагування тексту програми. Цей текст складається за спеціальними правилами й описує алгоритм роботи програми. Сукупність правил запису тексту називається мовою програмування. У системі Delphi використовується мова програмування Object Pascal, що являє собою розширену й вдосконалену версію широко розповсюдженої мови Паскаль, вперше запропонованої швейцарським вченим Н. Віртом ще в 1970 р. й удосконаленої співробітниками корпорації Borland (створені ними мови називалися Turbo Pascal, Borland Pascal і Object Pascal). Незважаючи на те, що візуальне середовище Delphi бере на себе багато рутинних аспектів програмування, знання мови Object Pascal є неодмінною умовою для будь-якого програміста, який працює в цьому середовищі.

Первісне вікно коду містить мінімальний вихідний текст, що забезпечує нормальне функціонування порожньої форми як повноцінного Windows-вікна. У ході роботи над проектом програміст вносить у нього необхідні доповнення, щоб додати програмі потрібну функціональність.

Відразу після відкриття нового проекту у вікні коду будуть такі рядки:

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

```
     private
     { Private declarations }
     public
     { Public declarations }
     end;
var
     Form1: TForm1;
implementation
{SR *.DFM}
end.
```

Ці рядки Delphi автоматично вставляє у вікно коду для нової форми. Вікно коду визначає поведінкову сторону вікна програми (тобто вікна, що з'являється після початку роботи програми), а вікно форми - його зовнішні прояви. Обоє вікна тісно пов'язані один з одним, причому Delphi “господарює” у його верхній частині, вставляючи необхідні рядки між

```
unit Unit1;
```

і

```
implementation
```

Ми будемо вставляти текст програми між рядками

```
{$R-*.DFM}
```

і

```
end.
```

у нижній частині вікна.

Щоб вставити у вікно новий рядок (рядки), потрібно спочатку за допомогою клавіш курсора або клацнувши по вікну мишею, поставити курсор на потрібне місце, а потім за допомогою клавіатури ввести текст. Звичайно текст коду програми розташовується в декількох рядках. Для переходу на новий рядок використовуйте клавішу Enter. Якщо в процесі введення Ви помилилися і відразу помітили свою помилку, видаліть помилковий символ клавішею Backspace. Клавіша Backspace видаляє символ ліворуч від курсора, а клавіша Delete - праворуч від нього. Якщо знадобиться видалити відразу весь рядок тексту, поставте в будь-яке місце рядка курсор, натисніть клавішу Ctrl і, не відпускаючи її, клавішу з латинською буквою Y. Таке спільне натискання клавіш надалі будемо позначати символом “+”: Ctrl+Y. Щоб скасувати останню зміну тексту, натисніть Ctrl+Z або виберіть пункт меню Edit | Undo.

Разом з вікном коду звичайно активізується вікно браузера Code Explorer, що полегшує пошук потрібних елементів у випадку, коли у вікні набрано багато рядків коду

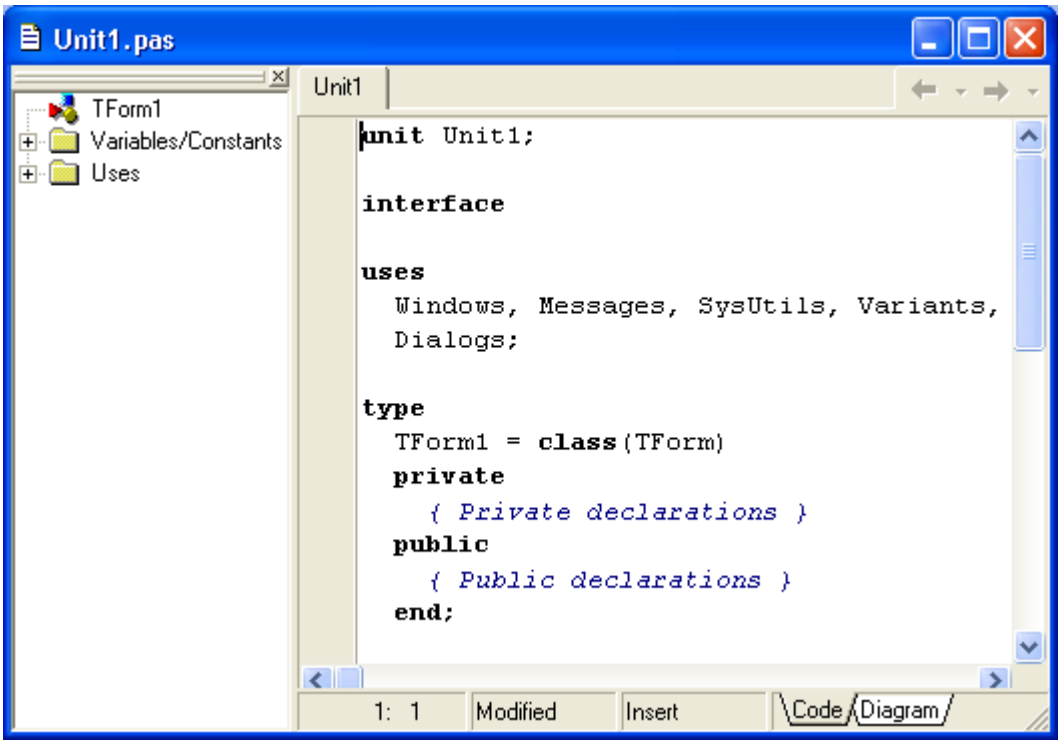

Рис. 9 - Вікно коду з розташованим ліворуч браузером Code Explorer

У нижній частині вікна коду розташовані дві закладки - Code і Diagram. Клацання мишею по останній активізує сторінку діаграм. Спочатку ця сторінка порожня.

Розділ II

ОСНОВИ МОВИ ПРОГРАМУВАННЯ OBJECT PASCAL

Типи даних

Порядкові типи

У цих типах інформація представляється у вигляді окремих елементів. Зв'язок між окремими елементами і їхнім представленням у пам'яті визначає природні відносини порядку між цими елементами. Звідси і назва - порядкові.

У Object Pascal визначені три групи порядкових типів і два типи, обумовлені користувачем. Групи — це цілі, символьні й булеві типи. Порядкові типи, що задаються користувачем, — це перерахування і піддіапазони. Усі значення будь-якого порядкового типу утворюють впорядковану послідовність, і значення змінної порядкового типу визначається його місцем у цій послідовності. За винятком змінних цілих типів, значення яких можуть бути більш або менш нуля, перший елемент будь-якого порядкового типу має номер 0, другий елемент — номер 1 і т.д.

Таблиця 1 - Операції над порядковими типами

Операція	Опис
Low (T)	Мінімальне значення порядкового типу T
High(T)	Максимальне значення порядкового типу T
Ord(X)	Порядковий номер значення порядкового типу. Для цілого виразу — просто його значення. Для інших порядкових типів Ord повертає фізичне представлення результату виразу, що трактується як ціле число. Значення, що повертається, завжди належить одному з цілих типів
Pred(X)	Попереднє значення. Для цілих виразів еквівалентно X-1
Succ(X)	Наступне значення. Для цілих виразів еквівалентно X+1
Dec(V)	Зменшує значення змінної на 1. Еквівалентно V := Pred(V)
Inc(V)	Збільшує значення змінної на 1. Еквівалентно V := Succ(V)

Цілі типи

У змінних цілих типів інформація представляється у вигляді цілих чисел, тобто чисел, що не мають дробової частини. Визначені в Object Pascal цілі типи підрозділяються на фізичні (фундаментальні) і логічні (загальні). У Object Pascal визначені наступні цілі типи:

Integer

Shortint

Smallint

Longint

Byte

Word

Cardinal

Таблиця 2 - Фізичні цілі типи

Тип	Діапазон значення	Фізичний формат
Shortint	Від -128 до 127	8 біт, зі знаком
Smallint	Від -32 768 до 32 767	16 біт, зі знаком
Longint	Від -2 147 483 648 до 2 147 483 647	32 біт, зі знаком
Byte	Від 0 до 255	8 біт, без знака
Word	Від 0 до 65 535	16 біт, без знака

Діапазони значень і формати фізичних цілих типів не залежать від мікропроцесора й операційної системи, в яких виконується програма. Вони не змінюються зі зміною реалізації або версії Object Pascal.

Діапазони значень логічних цілих типів (Integer і Cardinal) визначаються зовсім іншим способом. Вони ніяк не зв'язані з діапазонами відповідних фізичних типів.

Таблиця 3 - Логічні цілі типи

Тип	Діапазон значень	Фізичний формат
Integer	-32 768-32 767	16 біт, зі знаком (SmalInt)
Integer	-2 147 483 648-2 147 483 647	32 біт, зі знаком (Longint)
Cardinal	0-65 535	16 біт, без знака (Word)
Cardinal	0-2 147483647	32 біт, без знака (Longint)

Над цілими даними виконуються всі операції, визначені для порядкових типів, але з ними все-таки зручніше працювати як з числами, а не з "нечисельними порядковими типами". Як і звичайні числа, дані цілих типів можна складати (+), віднімати (-) і множити (*). Однак деякі операції і функції, застосовувані до даних цілих типів, мають дещо інший зміст.

Таблиця 4 - Операції над цілими типами

Операція	Результат
Abs (X)	Повертає абсолютне ціле значення X
X Div Y	Повертає цілу частину результату ділення X на Y
X Mod Y	Повертає остачу ділення X на Y
Odd (X)	Повертає булево True (істина), якщо X — непарне ціле, і False (неправда) — у противному разі
Sqr (X)	Повертає цілий квадрат X (тобто X*X)

Символьні типи

Зміст символьних даних очевидний, коли вони виводяться на екран або принтер. Проте, визначення символьного типу може залежати від того, що мати на увазі під словом символ. У Object Pascal визначені два фізичних символьних типи й один логічний.

Фізичні типи:

AnsiChar - Однобайтові символи, упорядковані відповідно до розширеного набору символів ANSI (American National Standards Institute — Американський національний інститут стандартів)

WideChar - Символи обсягом у слово, впорядковані відповідно до міжнародного набору символів UNICODE. Перші 256 символів збігаються із символами ANSI

Логічний символьний тип називається char.

При написанні програм, що можуть обробляти рядки будь-якого розміру, для вказівки цього розміру рекомендується застосовувати функцію SizeOf.

Застосовувані функції:

Chr (X) - Перетворює цілую змінну в змінну типу char з тим самим порядковим номером. У Delphi це еквівалентно завданню типу Char (X)

UpCase - Перетворює малу літеру в прописну.

Булеві типи

Якщо інформацію про що-небудь можна представити як ІСТИНА (True) або НЕПРАВДА (False), то в Object Pascal вона зберігається в змінних булевих типів. Усього таких типів чотири.

Таблиця 5 - Розміри змінних булевих типів

Тип	Розмір
Boolean	1 байт
ByteBool	1 байт
WordBool	2 байт (обсяг Word)
LongBool	4 байт (обсяг Longint)

Змінним типу Boolean можна привласнювати тільки значення True (істина) і False (неправда). Змінні ByteBool, WordBool і LongBool можуть приймати й інші порядкові значення, що інтерпретуються звичайно як False у випадку нуля і True — при будь-якому ненульовому значенні.

Перелічувальні типи

Опис типів, що перелічуються:

Type NameType = (first value, value2, value3, last value);

Приклад. Можна створити тип MyColor (мій колір) із значеннями myRed, myGreen і myBlue (мій червоний, мій зелений, мій синій). Це робиться так:

Type MyColor = (myRed, myGreen, myBlue);

Подібно до символьних і булевих типів перелічувальні не є числами і використовувати їх на зразок чисел недоцільно. Однак перелічувальні типи відносяться до порядкових, так що значення будь-якого такого типу впорядковані. Ідентифікаторам у списку привласнюються як порядкові номери послідовні числа. Першому імені привласнюється порядковий номер 0, другому — 1 і т.д.

Піддіапазонні типи

Змінні піддіапазонного типу містять інформацію, що відповідає деякому заданому діапазону значень вихідного типу, що представляє будь-який порядковий тип, крім піддіапазонного. Синтаксис визначення піддіапазонного типу має такий вигляд:

Type subrange type = low value...high value;

Піддіапазонні змінні зберігають всі особливості вихідного типу. Єдина відмінність полягає в тому, що змінної піддіапазонного типу можна привласнювати тільки значення, що входять у зазначений піддіапазон.

Дійсні типи

Змінними дійсних типів є числа, що складаються з цілої і дробової частин. У Object Pascal визначено шість дійсних типів. Усі типи можуть представляти число 0, однак вони розрізняються граничним (мінімальним) і максимальним значеннями, а також точністю (кількістю значущих цифр) і обсягом.

Таблиця 6 - Дійсні типи

Тип	Поріг	Максимальне значення	Кількість значащих цифр	Обсяг (байт)
Real	2.9E-39	1.7E38	11-12	6
Single	1.5E-45	3.4E38	7-8	4
Double	5.0E-324	1.7E308	15-16	8
Extended	3.4E-4932	1.IE4932	19-20	10
Comp	1.0	9.2E18	19-20	8
Currency	0.0001	9.2E14	19-20	8

Таблиця 7 - Функції дійсних типів:

Функція	Значення, що повертається
Abs (x)	Абсолютна величина x
ArcTan(x)	Арктангенс x
Cos (x)	Косинус x (x виражається в радіанах, а не в градусах)
Exp (x)	Експонентна функція від x
Frac(x)	Дробова частина x
Int (x)	Ціла частина x. Незважаючи на назву, повертає дійсне значення (з комою, що плаває), тобто просто встановлює нуль у дробовій частині
Ln (x)	Натуральний логарифм від x
Pi	Число Пі (3.1416...)
Round (x)	Найближче до x ціле значення. Повертає значення цілого типу. Умова "найближче до x" не працює, якщо верхнє і нижнє значення виявляються равновіддаленими (наприклад, якщо дробова частина точно дорівнює 0,5). У цих випадках Delphi перекладає рішення на операційну систему. Звичайно процесори Intel вирішують цю задачу відповідно до рекомендації IEEE округляти убік найближчого парного цілого числа. Іноді такий підхід називають "банкірським округленням"
Sin(x)	Синус x
Sqr(x)	Квадрат x, тобто X*X
Sqrt (x)	Квадратний корінь від x
Trunc (x)	Ціла частина x. На відміну від Int, що повертає дійсне значення, Trunc повертає ціле

Строкові типи

У виразах Delphi підтримує три фізичних строкових формати: короткий (ShortString), довгий (LongString) та широкий (WideString).

Таблиця 8 - Строкові функції

Функція	Опис
Concat(sl, s2, s3)	Повертає послідовне з'єднання рядків. Еквівалентна оператору sl+s2+s3
Copy(s, pos, len)	Повертає підстроку довжиною максимум len символів, що починається в позиції pos рядка s
Delete(s, pos, len)	Видаляє максимум len символів із рядка s, починаючи з позиції pos
Insert(sourse, target, pos)	Вставляє рядок source у строкову змінну target, починаючи з позиції pos
Length (s)	Повертає динамічну довжину рядка.
Pos(substring, s)	Повертає місце першого входження підрядка substring у рядок s.
SetLength(s, newlen)	Задає нову динамічну довжину newlen строкової змінний s
SetString	Задає зміст і довжину рядка
Str(x, s)	Перетворить чисельне значення x у строкову змінну s
StringOfChars	Повертає рядок з конкретним числом символів
UniqueString	Робить даний рядок унікальним з рахунком звертань 1
Val (s, v, code)	Перетворить рядок s у відповідне чисельне представлення v
Процедури та функції для роботи з рядками, що утримуються в модулі SisUtils	
IntToStr (Value: Integer): String	Перетворення значення цілочисельного виразу Value у рядок
StrToInt (const S: String): Integer	Перетворення рядка S у ціле число
FloatToStr(Value: Extended): String	Перетворення значення дійсного виразу Value у рядок
StrToFloat (const S: String): Extended	Перетворення рядка S у дійсне число
UpperCase(const S: String): String	Переклад символів рядка S у верхній регістр
LowerCase(const S: String): String	Переклад символів рядка S у нижній регістр
Trim(const S: String): String	Видалення з початку і кінця рядка S пробілів і керуючих символів
TrimLeft(const S: String): String	Видалення з початку рядка S пробілів і керуючих символів
TrimRight(const S: String): String	Видалення наприкінці рядка S пробілів і керуючих символів

Записи

За допомогою зарезервованого слова record (запис) в одному типі можна поєднувати дані різних типів. Загальний синтаксис оголошення цього типу виглядає так:

```
    record
fieldname1: fieldtype1;
fieldname2, fieldname3: fieldtype2;
case optional tagfield: required ordinal type of
1: variantname1: varianttype3;
2, 3: variantname2: varianttype4;
end;
```

Дане оголошення складається з фіксованої і варіантної частин. Однак зовсім не обов'язково вставляти в одне оголошення запису обидві ці частини, звичайно зручніше працювати з кожною з цих частин окремо.

Фіксовані записи

У фіксованій частині запису визначається одне або кілька незалежних полів. Кожному полю обов'язково привласнюється ім'я та тип:

```
    record
fieldname1: fieldtype1;
fieldname2, fieldname3: fieldtype2;
end;
```

Маючи доступ до інформації у запису, можна обробляти весь запис цілком (усі поля одночасно) або тільки окреме поле. Для звертання до окремого поля необхідно набрати ім'я запису, крапку й ідентифікатор поля, наприклад:

```
MyRec.Fieldname1
```

Для доступу до всього запису необхідно вказати його ім'я.

Варіантні записи

Варіантна частина типу record дає можливість по-різному трактувати область пам'яті, спільно займану варіантами поля:

```
     record
case optional tagfield: required ordinal type of
1: variantnamel: varianttype3;
2, 3: variantname2: varianttype4;
end;
```

Усі варіанти займають у пам'яті одне місце. Кожен варіант позначається деякою постійною. При бажанні можна одержувати доступ до всіх полів усіх варіантів одночасно, однак це може мати сенс тільки в найбільш простих випадках, коли точно відомо, як саме інформація кожного варіанта записується в пам'ять. Кожен варіант позначається мінімум однією константою. Усі константи повинні бути порядковими та сумісними за типом з міткою поля.

Масиви

Масивом називається впорядкована індексована сукупність однотипних елементів, що мають загальне ім'я. Елементами масиву можуть бути дані будь-яких типів, включаючи структуровані. Масиви можуть бути одно- або багатомірними. Кожен елемент масиву визначається ім'ям масиву й індексом (номером елемента в масиві) або індексами, якщо масив багатомірний. У програмі масиви описуються за допомогою службового слова `array`.

Наприклад:

```
array [ordinal_type] of type_definition;
array [ordinal_typel, ordinal_type2] of type_definition;
```

Кожен масив містить деяку кількість елементів інформації одного типу. Число елементів масиву в кожнім вимірі задається порядковим типом (ordinal_type). Для цього можна скористатися ідентифікатором деякого типу (наприклад, Boolean або AnsiChar), однак на практиці, звичайно, явно задається піддіапазон цілих. Кількість елементів масиву дорівнює добутку кількостей елементів у всіх вимірах. Для звертання до елемента масиву вказується ім'я цього масиву й індекс елемента у квадратних дужках.

Наприклад:

масив визначений у такий спосіб:

```
var MyArray: Array [1..10] of Integer;
```

Тоді звертання до його третього елемента виглядатиме, як `MyArray[3]`, і виконуватися, як до змінної `Integer`.

Множина

Зарезервоване слово `set` (множина) визначає множину не більше ніж з 256 порядкових значень:

```
Set of ordinal type
```

Мінімальний і максимальний порядкові номери вихідного типу (на основі якого визначається множинний тип) повинні бути в межах між 0 і 255. Змінна множинного типу містить (або не містить) будь-яке значення вихідного порядкового типу. Кожне значення із заданого діапазону може належати або не належати множині.

Приклад:

```
MyAlphaSet := ['A', 'E', 'I', 'O', 'U', 'Y']; // Усі прописні голосні.
```

Порожні квадратні дужки задають порожню множину, що не містить жодного елемента.

Файловий тип

Тип `file` призначений для доступу до лінійної послідовності елементів. Оголошення файлового типу:

```
file of Type1 // Файл визначеного типу, що містить запис фіксованої довжини.
file // Файл без типу.
Text file // Файл із текстовими записами змінної довжини, розділеними
//символами CR і LF ("повернення каретки" і "новий рядок").
```

У Delphi файлова структура — це змінна.

Вказівні типи

Змінна вказівного типу містить значення, що вказує на змінну звичайного типу — адреса цієї змінної

Приклад оголошення змінних-вказівок:

```
Var p1: pointer // Вказівка без типу.
    p2: ^Integer // Вказівка з типом.
```

Таблиця 9 - Засоби роботи з вказівками:

Засіб	Опис
New	Розподіляє нову ділянку динамічної пам'яті і записує її адресу в змінну вказівного типу
Оператор @	Направляє змінну-вказівку на область пам'яті, що містить будь-яку існуючу змінну, процедуру або функцію, включаючи змінні, що мають ідентифікатори
GetMem	Створює нову динамічну змінну заданого обсягу і записує її адресу в змінну вказівного типу

Інформація, що утримується в змінній вказівного типу, — це адреса деякої ділянки в машинній пам'яті. Ці значення задаються під час роботи програми і можуть змінюватись від одного запуску до іншого.

Зарезервоване слово Nil вказує значення вказівки, яка ні на що не вказує. Такі вказівки називають невизначеними.

Структура програм Delphi

Будь-яка програма в Delphi складається з файлу проекту (файл із розширенням dpr) і одного або декількох модулів (файли з розширеннями pas). Кожний з таких файлів описує програмну одиницю Object Pascal.

Структура проекту

Файл проекту являє собою програму, написану мовою Object Pascal і призначену для обробки компілятором. Ця програма автоматично створюється Delphi і містить лише кілька рядків. Щоб побачити їх, необхідно запустити

Delphi і клацнути по опції Project | View Source головного меню. Delphi покаже вікно коду із закладкою Project1, що містить такий текст:

```
program Project1;
uses
     Forms, Unit1 in 'Unit1.pas' {fmExample};
{$R *.RES}
begin
     Application.Initialize;
Application.CreateForm(TfmExample, fmExample);
Application.Run;
end.
```

У вікні коду жирним шрифтом виділяються зарезервовані слова, а курсивом - коментарі. Текст програми починається зарезервованим словом program і закінчується словом end із крапкою за ним. Сполучення end з крапкою називається термінатором програмної одиниці: як тільки в тексті програми зустрінеться такий термінатор, компілятор припиняє аналіз програми й ігнорує частину тексту, що залишилася.

Зарезервовані слова відіграють важливу роль у Object Pascal, додаючи програмі в цілому властивість тексту, написаного на майже природній англійській мові. Кожне зарезервоване слово (а їх в Object Pascal кілька десятків) несе в собі умовне повідомлення для компілятора, що аналізує текст програми так само, як читаємо його і ми: зліва направо і зверху вниз.

Коментарі, навпаки, нічого не значать для компілятора, і він їх ігнорує. Коментарі важливі для програміста, який з їхньою допомогою пояснює ті або інші місця програми. Наявність коментарів у тексті програми робить її зрозумілішою і дозволяє легко згадати особливості реалізації програми, які Ви написали кілька років тому. У Object Pascal коментарем вважається будь-яка послідовність символів, укладена у фігурні дужки. У наведеному вище тексті таких коментарів два, але рядок

```
{$R *.RES}
```

насправді не є коментарем. Цей спеціальним чином написаний фрагмент коду називається директивою компілятора (у нашому випадку - вказівка компілятору на необхідність підключення до програми файлу ресурсів).

Директиви починаються символом $, що розташований відразу за відкриваючою фігурною дужкою.

У Object Pascal як обмежники коментарю можуть також використовуватися пари символів (*, *) і //. Дужки (*...*) використовуються подібно фігурним дужкам, тобто коментарем вважається фрагмент тексту, що знаходиться в них, а символи // вказують компілятору, що коментар розташовується за ними і продовжується до кінця поточного рядка:

Наприклад:

```
{Це коментар}
(*Це теж коментар*)
//Усі символи до кінця цього рядка складають коментар
```

Слово Program із наступним за ним ім'ям програми і крапкою з комою складають заголовок програми. За заголовком йде розділ описів, у якому програміст (або Delphi) описує використовувані в програмі ідентифікатори. Ідентифікатори позначають елементи програми, такі як типи, змінні, процедури, функції. Тут же за допомогою речення, що починається зарезервованим словом uses (використовувати) програміст повідомляє компілятору про ті фрагменти програми (модулі), які необхідно розглядати як невід'ємні складові частини програми і які розташовуються в інших файлах.

Рядки:

```
uses
Forms, Unit1 in 'Unitl.pas' {fmExample};
```

вказують, що крім файлу проекту в програмі повинні використовуватись модулі Forms і Unit1. Модуль Forms є стандартним (тобто уже відомим Delphi), а модуль Unit1 - новим, раніше невідомим, і Delphi у цьому випадку вказує також ім'я файлу з текстом модуля (in 'uniti.pas') і ім'я зв'язаного з модулем файлу опису форми {fmExample}.

Власне тіло програми починається зі слова begin (почати) і обмежується термінатором end із крапкою. Тіло складається з декількох операторів мови Object Pascal. У кожнім операторі реалізується деяка дія - зміна значення

змінної, аналіз результату обчислення, звертання до підпрограми і т.п. У тілі нашої програми - три оператори:

```
Application.Initialize;
Application.CreateForm(TfmExample, fmExample);
Application.Run;
```

Кожний з них реалізує звертання до одного з методів об'єкта Application.

Об'єктом називається спеціальним чином оформлений фрагмент програми, що укладає в собі дані і підпрограми для їхньої обробки. Дані називаються полями об'єкта, а підпрограми - його методами. Об'єкт у цілому призначений для вирішення якої-небудь конкретної задачі і сприймається в програмі як неподільне ціле (іншими словами, не можна з об'єкта “висмикнути” окреме поле або метод). Об'єкти відіграють надзвичайно важливу роль у сучасних мовах програмування. Вони придумані для того, щоб збільшити продуктивність праці програміста й одночасно підвищити якість розроблювальних ним програм. Дві головні властивості об'єкта - функціональність і неподільність - роблять його самостійною або навіть самодостатньою частиною програми і дозволяють легко переносити об'єкт з однієї програми в іншу. Розробники Delphi придумали сотні об'єктів, які можна розглядати як цеглинки, з яких програміст будує багатоповерховий будинок програми. Такий принцип побудови програм називається об’єктно - орієнтованим програмуванням (ООП). В об'єкті Application зібрані дані і підпрограми, необхідні для нормального функціонування Windows-програми в цілому. Delphi автоматично створює об'єкт-програму Application для кожного нового проекту.

Рядок

```
Application.Initialize;
```

означає звертання до методу Initialize об'єкта Application. Прочитавши цей рядок, компілятор створить код, що змусить процесор перейти до виконання деякого фрагмента програми, написаного для нас розробниками Delphi. Після виконання цього фрагмента (програмісти говорять: після виходу з

підпрограми) керування процесором перейде до наступного рядка програми, в якій викликається метод CreateForm і т.д.

Структура модуля

Модулі - це програмні одиниці, призначені для розміщення фрагментів програм. За допомогою програмного коду, що утримується в них, реалізується вся поведінкова сторона програми. Всі модулі мають наступну структуру: заголовок, секція інтерфейсних оголошень, секція реалізації, термінатор.

Заголовок відкривається зарезервованим словом Unit за яким йде ім'я модуля і крапка з комою. Секція інтерфейсних оголошень відкривається зарезервованим словом Interface, а секція реалізації - словом implementation. Термінатором модуля, як і термінатором програми, є end із крапкою. Наступний фрагмент програми є синтаксично вірним варіантом модуля:

```
unit Unit1;
interface
// Секція інтерфейсних оголошень
implementation
// Секція реалізації
end.
```

У секції інтерфейсних оголошень описуються програмні елементи (типи, класи, процедури і функції), що будуть “видні” іншим програмним модулям, а в секції реалізації розкривається механізм роботи цих елементів. Поділ модуля на дві секцій забезпечує зручний механізм обміну алгоритмами між окремими частинами однієї програми. Він також реалізує засіб обміну програмними розробками між окремими програмістами. Одержавши відкомпільований “сторонній” модуль, програміст одержує доступ тільки до його інтерфейсной частини, у якій утримуються оголошення елементів. Деталі реалізації оголошених процедур, функцій, класів сховані в секції реалізації і недоступні іншим модулям.

Клацнувши по закладці Unit1 вікна коду, можна побачити такий текст:

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls,
```

```
Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
type
TfmExample = class (TForm)
Panel1: TPanel;
bbRun: TBitBtn;
bbClose: TBitBtn;
edinput: TEdit;
IbOutput: TLabel;
mmOutput: TMemo;
private
{ Private declarations } public
{ Public declarations } end;
var
fmExample: TfmExample;
implementation
$R *.DFM}
end.
```

Увесь цей текст сформований Delphi, але на відміну від файлу проекту програміст може його змінювати, додаючи програмі потрібну функціональність.

В інтерфейсній секції описані один тип (клас - fmExample) і один об'єкт (змінна fmExample).

Ось опис класу:

```
type
     TfmExample = class(TForm)
     Panell: TPanel;
     bbRun: TBitBtn;
     bbClose: TBitBtn;
     edinput: TEdit;
     IbOutput: TLabel;
     mmOutput: TMemo;
private
     { Private declarations } public
     { Public declarations } end;
```

Класи служать головним інструментом реалізації потужних можливостей Delphi. Клас є зразком, за яким створюються об'єкти, і навпаки, об'єкт - це екземпляр реалізації класу. Зразки для створення елементів програми в Object Pascal називаються типами, таким чином, клас TfmExamplel -це тип. Перед його оголошенням розташоване зарезервоване слово type (тип), що сповіщає компілятор про початок розділу опису типів.

Стандартний клас TForm реалізує все потрібне для створення та функціювання порожнього Windows-вікна. Клас TfmExamplel породжений від цього класу, про що свідчить рядок

```
TfmExample = class (TForm)
```

в якому за зарезервованим словом class у дужках вказується ім'я батьківського класу. Термін “породжений” означає, що клас TfmExample успадкував усі можливості батьківського класу TForm і додав до них власні у вигляді додаткових компонентів, що ми вставили у форму fmExample. Перелік вставлених нами компонентів і складає значну частину опису класу.

Властивість спадкування класами-нащадками усіх властивостей батьківського класу і збагачення їх новими можливостями є одним з фундаментальних принципів об’єктно - орієнтованого програмування. Від спадкоємця може бути породжений новий спадкоємець, що внесе свій вклад у вигляді додаткових програмних заготівель і т.д. У результаті створюється ієрархія класів, на вершині якої розташовується самий простий клас TObject (всі інші класи в Delphi породжені від цього єдиного прабатька), а на самій нижній сходинці ієрархії - потужні класи-нащадки.

Об'єкт fmExampie формально відноситься до елементів програми, що називаються змінними. От чому перед оголошенням об'єкта розташоване зарезервоване слово var (від англ. variables - змінні).

Елементи програми

Елементи програми - це мінімальні неподільні її частини, що несуть у собі визначену значущість для компілятора. До елементів відносяться: зарезервовані слова, ідентифікатори, типи, константи, змінні, мітки, підпрограми, коментарі.

Зарезервовані слова - це англійські слова, що вказують компілятору на необхідність виконання певних дій, наприклад, зарезервовані слова begin, end.

Ідентифікатори в Object Pascal можуть складатися з латинських літер, арабських цифр і знака підкреслення.

Типи - це спеціальні конструкції мови, що розглядаються компілятором як зразки для створення інших елементів програми, таких як змінні, константи і функції.

Константи визначають області пам'яті, що не можуть змінювати свого значення в ході роботи програми. Для оголошення імен констант використовується зарезервоване слово const. Наприклад:

```
const
k = 10;
```

Змінні пов'язані з областями пам'яті, що змінюються. Поперед розділу оголошення змінних повинне стояти слово var. Наприклад:

```
var
j: Integer;
m: Byte;
```

Мітки - це імена операторів програми. Мітки використовуються для того, щоб вказати компілятору, який оператор програми повинен виконуватись наступним. Перед розділом оголошень міток йде зарезервоване слово label. Наприклад:

```
label
Loop;
begin
Goto Loop;
// Програміст вимагає передати керування
// оператору, позначеному міткою Loop. .....
// Ці оператори будуть пропущені
Loop://Оператору, що йде за цією міткою, буде передане керування
...
end;
```

Підпрограми - це спеціальним чином оформлені фрагменти програми.

У Object Pascal є два сорти підпрограм: процедури і функції. Функція відрізняється від процедури тим, що її ідентифікатор можна поряд з константами та змінними використовувати у виразах, тому що функція має вихідний результат визначеного типу.

Отже, структуру програми в загальному випадку можна представити так:

```
Program <Ім'я програми>;
Uses <Список модулів>;
Label <Список міток>;
```

```
Const <Список констант>;
Type <Опис типів>;
Var <Оголошення змінних>;
<Опис процедур>;
<Опис функцій>;
Begin
<Оператори>;
End.
```

Оператори мови

Оператор присвоювання

В Object Pascal для присвоювання змінній визначеного значення або виразу використовується сполучення символів :=, наприклад:

```
X:=123;
Y:=2*X-35;
```

Складений оператор

Складений оператор - це послідовність довільних операторів програми, укладена в операторні дужки - зарезервовані слова begin ... end.

Object Pascal не накладає ніяких обмежень на характер операторів, що входять у складений оператор. Серед них можуть бути й інші складені оператори - мова Object Pascal допускає довільну глибину їхньої вкладеності.

Умовний оператор

Умовний оператор дозволяє перевірити деяку умову й у залежності від результатів перевірки виконати ту або іншу дію. Умовний оператор - це засіб розгалуження обчислювального процесу.

Структура умовного оператора має такий вигляд:

if <умова> then <оператор1> else <оператор2>;

де if/ then/ else - зарезервовані слова (якщо, те, інакше);

<умова> - довільний вираз логічного типу;

<оператор1>, <оператор2> - будь-які оператори мови Object Pascal.

Умовний оператор працює за наступним алгоритмом. Спочатку обчислюється умовний вираз <умова>. Якщо результат є True (істина), то

виконується <оператор1>, а <оператор2> пропускається; якщо результат є False (неправда), навпаки, <оператор1> пропускається, а виконується <оператор2>.

Наприклад:

```
var
X, Y, Max: Integer;
begin .
if X >Y then Max:= X else Max:= Y;
…
end;
```

<оператор1> і <оператор2> можуть бути складеними операторами. Наприклад:

```
var
X, Y, Max, Min: Integer;
begin .
    if X >Y then
      begin
       Max:= X;
       Min:=Y;
      end
    else
      begin
       Max:= Y;
       Min:=X;
      end;
…
end;
```

Крім операцій відносини =, <>, >, >=, <, <= умова в умовному операторі може містити складні логічні вирази, які використовують логічні операції **and** (логічне И), **or** (логічне АБО) і **not** (логічне НЕ). Наприклад:

```
if (a > b) and (b <> 0) then ...
if (l > m) or (m <> 0) then ...
```

У мові програмування Object Pascal пріоритет операцій відносин менше, ніж у логічних операцій, тому окремі складові частини складного логічного виразу обмежують дужками. Наприклад, такий запис попереднього оператора буде невірним:

```
if a>b and b <> 0 then ...// Помилка
```

Частини else <оператор2> умовного оператора може не бути. Тоді при значенні True умовного виразу виконується <оператор1>, у протилежному разі цей оператор пропускається.

Приклад:

```
var
X, Y, Max: Integer;
begin
if X > Y then Max := X;
…
end;
```

Будь-яка частина else, що зустрілася, відповідає найближчій до неї зверху по тексту програми частині then умовного оператора.

Оператор вибору

Оператор вибору дозволяє вибрати одне з декількох можливих продовжень програми. Параметром, за яким здійснюється вибір, є ключ вибору - вираз будь-якого порядкового типу (наприклад, integer, char і т.д.).

Структура оператора вибору:

```
case <ключ_вибору> of
<список_вибору>
else <оператори>
end;
```

Тут `case, of, else, end` - зарезервовані слова.

<ключ_вибору> - ключ вибору (вираз порядкового типу);

<список_вибору> - одна або більш конструкцій виду:

<константа_вибору> : <оператор>;

<константа_вибору> - константа того ж типу, що і вираз <ключ_вибору>;

<оператор> - довільний оператор Object Pascal.

Оператор вибору працює так:

спочатку обчислюється значення виразу <ключ_вибору>, а потім у послідовності операторів <список_вибору> виконується пошук такого, якому передує константа, що дорівнює обчисленому значенню. Знайдений оператор виконується, після чого оператор вибору завершує свою роботу. Якщо в списку вибору не буде знайдена константа, що відповідає обчисленому значенню ключа вибору, то керування передається операторам, що розташовані за словом else. Частина else <оператори> може бути відсутня. Тоді при відсутності в

списку вибору потрібної константи нічого не відбудеться і оператор вибору просто завершить свою роботу.

Кожному з операторів списку вибору може передувати не одна, а кілька констант вибору, розділених комами. Наприклад:

```
var
ch : Char;
begin
case ch of
'n','N','H','H': IbOutput.Caption := 'Немає';
'у','У','д','Д': IbOutput.Caption := 'Так';
else IbOutput.Caption := 'помилка введення';
end
end;
```

Оператори повторень

У мові Object Pascal є три різних оператори, за допомогою яких можна запрограмувати повторювані фрагменти програм.

Рахунковий оператор циклу FOR;

Оператор циклу WHILE із попередньою перевіркою умови;

Оператор циклу REPEAT... UNTIL з постперевіркою умови.

Рахунковий оператор циклу **FOR** має таку структуру:

`for <параметр циклу> := <поч_знач> to <кін знач> do <оператор>;`

Тут `for, to, do` - зарезервовані слова (для, до, виконати);

<параметр_циклу> - змінна типу Integer (або будь-якого іншого порядкового типу);

<поч_знач> - початкове значення – виразу того ж типу;

<кін_знач> - кінцеве значення - виразу того ж типу;

<оператор> - довільний оператор Object Pascal. Якщо оператор складений його необхідно брати в операторні дужки Begin...End.

При виконанні оператора for спочатку обчислюється вираз <поч_знач> і здійснюється присвоювання <параметр_циклу> := <поч_знач>. Після цього циклічно повторюється: перевірка умови <параметр_циклу> <= <кін_знач>;

якщо умова не виконана, оператор for завершує свою роботу; далі здійснюється виконання оператора <оператор>; потім нарощування змінної <параметр_циклу> на одиницю.

Умова, що керує роботою оператора for, перевіряється перед виконанням оператора <оператор>: якщо умова не виконується на самому початку роботи оператора for, оператор, що виконується, не буде виконаний жодного разу.

Крок нарощування параметра циклу строго постійний і дорівнює (+1), однак існує інша форма оператора:

```
for <параметр_цик>: = <поч_знач>downto <кін_знач>do
<оператор>;
```

У цьому випадку крок нарощування параметра циклу дорівнює (-1), а керуюча умова: <параметр__циклу> > = <кін_знач>.

Приклад:

```
Sum := 0;
if N >= 0 then
for i := 1 to N do
Sum := Sum + i
else
for i := -1 downto N do
Sum := Sum + i ;
```

Цей фрагмент програми робить підрахунок додатних або від'ємних чисел у залежності від значення N.

Оператор циклу **WHILE** із попередньою перевіркою умови має таку структуру:

```
while <умова> do <оператор>;
```

Тут while, do - зарезервовані слова (поки виконується умова, робити)

<умова> - вираз логічного типу;

<оператор> - довільний оператор Object Pascal. Якщо оператор складений його необхідно брати в операторні дужки Begin...End.

Якщо вираз <умова> має значення True, то виконується <оператор>, після чого обчислення виразу <умова> і його перевірка повторюються. Якщо <умова> має значення False, оператор while припиняє свою роботу.

Оператор циклу **REPEAT**... **UNTIL** з постперевіркою умови має наступну структуру:

```
repeat <тіло циклу> Until <умова>;
```

Тут repeat, until - зарезервовані слова (повторювати доти, поки не буде виконана умова);

<тіло_циклу> - довільна послідовність операторів Object Pascal;

<умова> - вираз логічного типу.

Оператори <тіло_циклу> виконуються хоча б один раз, після чого обчислюється вираз <умова>: якщо його значення є False, оператори <тіло_циклу> повторюються, у противному разі оператор repeat... until завершує свою роботу.

Зверніть увагу: пара repeat... until подібна операторним дужкам begin ... end, тому перед until ставити крапку з комою необов'язково.

Для гнучкого керування циклічними операторами for, while і repeat до складу Object Pascal включені дві процедури без параметрів:

break - реалізує негайний вихід з циклу; дія процедури полягає в передачі керування оператору, що розташований відразу за кінцем циклічного оператора;

continue - забезпечує дострокове завершення чергового проходу циклу; еквівалент передачі керування в самий кінець циклічного оператора.

Мітки й оператори переходу

Оператор переходу має вигляд:

```
goto <мітка>;
```

Тут `goto` - зарезервоване слово (перейти на мітку);

<мітка> - мітка.

Мітка в Object Pascal - це довільний ідентифікатор, що дозволяє іменувати деякий оператор програми й у такий спосіб посилатися на нього.

Мітка розташовується безпосередньо перед оператором, що позначається, і відокремлюється від нього двокрапкою. Оператор можна позначати декількома мітками, які в цьому випадку відокремлюються одна від одної двокрапкою. Перед тим як з'явитися в програмі, мітка повинна бути описана. Опис міток складається з зарезервованого слова label (мітка), за яким йде перелік міток.

Дія оператора `goto` полягає в передачі керування відповідному наміченому оператору.

Міткою може бути ідентифікатор або ціле число без знака в діапазоні 0..9999. При використанні міток необхідно керуватися наступними правилами: мітка, на яку посилається оператор goto, повинна бути описана в розділі описів, вона обов'язково повинна зустрітися де-небудь у тілі програми; мітки, описані в підпрограмі, локалізуються в ній, тому передача керування ззовні підпрограми на мітку усередині її неможлива.

Підпрограми

Основні відомості про підпрограми

Підпрограма являє собою групу операторів, логічно закінчену і спеціальним чином оформлену. Підпрограму можна викликати необмежене число раз з різних частин програми. Використання підпрограм дозволяє поліпшити структурованість програми і скоротити її розмір.

За структурою підпрограма майже цілком аналогічна програмі і містить заголовок і блок, однак у блоці підпрограми відсутній розділ підключення модулів. Крім того, заголовок підпрограми за своїм оформленням відрізняється від заголовка програми.

Робота з підпрограмою розподіляється на два етапи:

- опис підпрограми;
- виклик підпрограми.

Будь-яка підпрограма повинна бути попередньо описана, після чого допускається її виклик. При описі підпрограми визначаються її ім'я, список параметрів і виконувані підпрограмою дії. При виклику вказуються ім'я підпрограми і список аргументів (фактичних параметрів), переданих підпрограмі для роботи.

У різних модулях Delphi мається велика кількість стандартних підпрограм, які можна викликати без попереднього опису. Деякі з них приведені при описі типів даних і виразів. Крім того, програміст може створювати свої власні підпрограми, що також називаються користувальницькими.

Підпрограми поділяються на процедури і функції, що мають між собою багато спільного. Основне розходження між ними полягає в тому, що функція як результат своєї роботи може повертати під своїм ім'ям деяке значення і, відповідно, може використовуватися як операнд виразу.

Взаємодія з підпрограмою здійснюється по керуванню і за даними. Взаємодія по керуванню полягає в передачі керування з програми в підпрограму й в організації повернення в програму. Взаємодія за даними полягає в передачі підпрограмі даних, над якими вона виконує визначені дії. Цей вид взаємодії може здійснюватися такими основними засобами:

- з використанням файлів;
- за допомогою глобальних змінних;
- за допомогою параметрів.

Найбільш часто зустрічається останній спосіб. При цьому розрізняють параметри й аргументи. Параметри (формальні параметри) є елементами підпрограми і використовуються при описі операцій, які виконує підпрограма.

Аргументи (фактичні параметри) є елементами викликаючої програми. При виклику підпрограми вони заміщають формальні параметри. При цьому проводиться перевірка на відповідність типів і кількості параметрів і

аргументів. Імена параметрів і аргументів можуть розрізнятися, але їхня кількість і порядок проходження повинні збігатися, а типи параметрів і відповідних їм аргументів мають бути сумісними.

Для припинення роботи підпрограми можна використовувати процедуру Exit, що перериває виконання операторів підпрограми і повертає керування викликаючій програмі.

Підпрограми можна викликати не тільки з програми, але і з інших підпрограм.

Процедури

Опис *процедури* містить у собі заголовок і блок, що за винятком розділу підключення модулів не відрізняється від блоку програми. *Заголовок* складається з ключового слова `procedure`, імені процедури і необов'язкового списку параметрів у круглих дужках із вказівкою типу кожного параметра. Заголовок має формат:

```
Procedure <Ім'я> (формальні параметри) ;
```

Для звертання до процедури використовується *оператор виклику* процедури. Він складається з імені процедури і списку аргументів, укладеного в круглі дужки.

Розглянемо як приклад процедуру обробки події натискання кнопки `Button1`, в якій, в свою чергу, викликаються дві процедури `DecodeDate` і `ChangeStr`.

```
     procedure TForm1.Button1Click(Sender: TObject);
     // Опис користувальницької процедури ChangeStr
     procedure ChangeStr(var Source:string; const charl, char2:
char);
     label 10;
     var n: integer;
     begin
     10:
     n:=pos(char1,Source);
     if n > 0 then
     begin
     Source[n]:=char2;
     goto 10;
     end;
```

```
end;
var str1: string;
Year, Month, Day: word;
begin
// Виклик процедури DecodeDate
DecodeDate(Now, Year, Month, Day) ;
Str1:=Edit1.Text;
// Виклик користувальницької процедури ChangeStr
ChangeStr (strl, '1', '*');
Edit1.Text:=str1;
end;
```

Процедура DecodeDate, що служить для розкладання дати на окремі складові (рік, місяць і день), може бути використана без попереднього опису, тому що вона вже описана в модулі SysUtils. Процедура ChangeStr виконує заміну в рядку Source усіх входжень символу, що задає параметр char1, на символ, що задається параметром char2.

Попередній опис користувальницької процедури ChangeStr виконано безпосередньо в обробника події натискання кнопки Button1. Цей опис варто винести за межі оброблювача, у цьому випадку процедуру ChangeStr можна буде викликати не тільки з даного обробника.

Виклик процедури ChangeStr забезпечує заміну всюди в рядку str1 символу 1 на символ *.

Функції

Опис *функції* складається із заголовка та блоку. *Заголовок* включає ключове слово Function, ім'я функції, необов'язковий список формальних параметрів, взятий у круглі дужки, і тип значення, що повертається функцією. Заголовок має формат:

```
Function <Ім'я> (Формальні параметри) : < Тип результату >;
```

Значення, що повертається, може мати будь-як тип, крім файлового.

Блок функції являє собою локальний блок, за структурою аналогічний блоку процедури. У тілі функції повинен бути хоча б один оператор присвоювання, у лівій частині якого розташоване ім'я функції. Саме він і встановлює значення, що повертається функцією. Якщо таких операторів декілька, то результатом функції буде значення останнього виконаного

оператора присвоювання. У цих операторах замість імені функції допускається вказувати змінну Result, що створюється як синонім для імені функції. На відміну від імені функції, змінну Result варто використовувати у виразах блоку функції. За допомогою цієї змінної можна в будь-який момент одержати всередині блоку доступ до поточного значення функції.

Ім'я функції, в принципі, також можна використовувати у виразах блоку функції, однак це приводить до рекурсивного виклику функцією самої себе.

Виклик функції здійснюється по її імені з вказівкою в круглих дужках списку аргументів, якого може і не бути. При цьому аргументи повинні попарно відповідати параметрам, зазначеним у заголовку функції, і мати ті ж типи. На відміну від процедури, ім'я функції може входити у вирази як операнд. Для прикладу розглянемо процедуру обробки події натискання кнопки Button1, в якій викликаються дві функції: Length і ChangeStr2.

```
  procedure TForm1.Button1Click(Sender: TObject);
  // Опис функції ChangeStr2
  function  ChangeStr2(Source:string;  const  char1,char2:char):
string;
  label 10;
  var n: integer;
  begin
  Result:=Source;
  10:
  n:=pos(char1, Result);
  if n > 0 then
    begin
      Result[n]:=char2;
      goto 10;
    end;
    end;

    var str1: string;
        n:    integer;
    begin
    str1:=Edit1.Text;
    // Виклик функції ChangeStr2
    str1:=ChangeStr2(str1, '1', '*');
    Edit1.Text:=str1;
    //  Виклик  функції  Length
    n:=Length(str1) ;
    end;
```

Функція Length повертає довжину рядка і може бути використана без попереднього опису, оскільки вона утримується в модулі System. Функція ChangeStr2 виконує ті ж дії, що і процедура ChangeStr з попереднього прикладу. Виклик функції використовується в операторі присвоювання.

Рекурсивні підпрограми

Іноді потрібно, щоб підпрограма викликала сама себе. Такий спосіб виклику називається *рекурсією*. Рекурсія корисна у випадках, коли основну задачу можна розбити на підзадачі, кожна з яких реалізується за алгоритмом, що збігається з основним.

Розглянемо, наприклад, обчислення факторіала числа:

```
function Fact(n; integer): integer;
begin
if n <= 0 then Fact := 1 else Fact:= n * Fact(n-l);
end;
```

Функція `Fact()` одержує число n і обчислює його факторіал. При цьому, якщо n менше або дорівнює нулю, то функція повертає значення 1, у противному випадку, зменшивши значення n на одиницю, функція викликає сама себе. Подібний рекурсивний виклик виконується доти, поки n не стане рівним одиниці.

Функція `Fact ()` і її параметр мають тип `Integer`, що відповідає типу `Longint` і дозволяє обчислювати факторіал для досить невеликих значень n (не більше 31). Для збільшення діапазону значень можна використовувати дійсні типи, наприклад, `Real.`

Параметри й аргументи

Параметри є елементами підпрограми і використовуються при описі здійснених у ній дій. Аргументи вказуються при виклику підпрограми і заміщують параметри при виконанні підпрограми. Параметри можуть мати будь-як тип, включаючи структурований. Існує кілька видів параметрів:

значення;

константа;

змінна;

нетипізована константа і змінна.

Група параметрів, перед якими в заголовку підпрограми відсутні слова var або const і за якими йде їхній тип, називаються параметрами-значеннями. Наприклад, `a` і `g` є параметрами-значеннями в наступному описі:

```
procedure P1(a: real; g: integer);
```

Параметр-значення обробляється як локальна стосовно підпрограми змінна. У підпрограмі значення таких параметрів можна змінювати, однак ці зміни не впливають на значення відповідних їм аргументів, що при виклику підпрограми були підставлені замість формальних параметрів.

Група параметрів, перед якими в заголовку підпрограми розташоване слово const і за якими йде їхній тип, називаються параметрами-константами. Наприклад, в описі:

```
procedure P2(const x, y : integer);
```

x і y є параметрами-константами. У тілі підпрограми значення параметра-константи змінити не можна. Параметрами-константами можна оформити ті параметри, зміна яких у підпрограмі небажана і повинна бути заборонена. Крім того, для параметрів-констант строкових і структурних типів компілятор створює більш ефективний код.

Група параметрів, перед якими в заголовку підпрограми розташоване слово var і за якими йде їхній тип, називаються *параметрами-змінними*.

Наприклад, параметрами-змінними є d і f у наступному описі:

```
function F1 (var d, f: real; ql7 : integer): real;
```

Параметр-змінна використовується у випадках, коли значення повинне бути передане з підпрограми у викликаючий блок. У цьому випадку при виклику підпрограми параметр заміщається аргументом-змінною, і будь-які зміни формального параметра відбиваються на аргументі. У такий спосіб можна повернути результати з підпрограми по закінченні її роботи.

Для параметрів-констант і параметрів-змінних допускається не вказувати їхній тип, тобто вважати їх *нетипізованими*. У цьому випадку аргументи, що підставляються на їхнє місце можуть бути будь-якого типу, і програміст повинен самостійно інтерпретувати типи параметрів у тілі підпрограми. Прикладом задання нетипізованих параметрів-констант і параметрів-змінних є наступний опис:

```
function F2 (var e1; const t2): integer;
```

Групи параметрів в описі підпрограми розділяються крапкою з комою.

Модулі

Крім програм, структура яких була розглянута, засоби мови дозволяють створювати модулі. На відміну від програми, модуль не може бути автономно запущений на виконання і містить елементи, наприклад, змінні і підпрограми, що допускається використовувати в програмі або в інших модулях. Для того щоб можна було використовувати засоби модуля, його треба підключити, вказавши ім'я цього модуля в розділі uses. Типовими прикладами модулів є System і SysUtils, що містять велику кількість стандартних підпрограм (деякі з них вже були розглянуті). Для кожної форми створюється окремий модуль.

Модуль складається із заголовка, в якому після ключового слова unit вказується ім'я модуля, і чотирьох розділів: інтерфейсу (interface), реалізації (implementation), ініціалізації (initialization) і деініціалізації (Finalization).

Модуль має наступну структуру:

```
Unit <Ім'я модуля>;

// Розділ інтерфейсу
Interface
Uses <Список модулів>;
Const <Список констант>;
Type <Опис типів>;
Var <Оголошення змінних>;
<Заголовки процедур>;
<Заголовки функцій>;

// Розділ реалізації
Implementation
Uses <Список модулів>;
```

```
     Const <Список констант>;
     Type <Опис типів>;
     Var <Оголошення змінних>;
     <Опис процедур>;
     <Опис функцій>;

     // Розділ ініціалізації
Initialization
     <Оператори>

     // Розділ деініціалізації
Finalization
     <Оператори>
     End.
```

У розділі *інтерфейсу* розміщуються описи ідентифікаторів, що повинні бути доступні всім модулям і програмам, які використовують цей модуль і містять його ім'я в списку uses. У розділі інтерфейсу з'являються типи, константи, змінні і підпрограми. При цьому для підпрограм вказуються тільки їхні заголовки. Інші використовувані модулі вказуються в списку uses. Розділ інтерфейсу починається ключовим словом interface.

У розділі *реалізації* розташовується код підпрограм, заголовки яких були наведені в розділі інтерфейсу. Порядок проходження підпрограм може не збігатися з порядком розташування їхніх заголовків, що наводяться в розділі інтерфейсу. Крім того, допускається залишати в заголовку тільки ім'я підпрограми, тому що список параметрів і тип результату функції вже були попередньо зазначені. У розділі реалізації можна також описувати типи, повідомляти константи і змінні й описувати підпрограми, що використовуються тільки в цьому модулі і за його межами не видні. Розділ інтерфейсу починається словом `implementation`.

У розділі ініціалізації розташовуються оператори, виконувані на початку роботи програми, що підключає даний модуль. Розділи ініціалізації модулів виконуються в тому порядку, в якому вони перераховані в списку розділу uses програми. Розділ ініціалізації починається словом `Initialization` і є необов'язковим.

При наявності розділу ініціалізації в модулі можна використовувати розділ деініціалізації, що починається словом `Finalization` і є

необов'язковим. У цьому розділі розташовуються оператори, виконувані при завершенні програми. Розділи деініціалізації модулів виконуються в порядку, зворотному порядку їхнього перерахування в списку uses програми.

Розділ III
ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ

Основні концепції візуального програмування в Delphi

Програмування в Delphi будується на взаємодії двох процесів: процесу конструювання візуального прояву програми (тобто її Windows-вікна) і процесу написання коду, що додає елементам цього вікна і програмі в цілому необхідну функціональність. Для написання коду використовується вікно коду, для конструювання програми - інші вікна Delphi, насамперед - вікно форми.

Між змістом вікон форми і коду існує нерозривний зв'язок, що строго відслідковується Delphi. Це означає, що розміщення на формі компонента приводить до автоматичної зміни коду програми і навпаки - видалення тих або інших автоматично вставлених фрагментів коду може привести до видалення відповідних компонентів. Пам'ятаючи про це, програмісти спочатку конструюють форму, розміщуючи на ній черговий компонент, а вже тільки після цього переходять, якщо це необхідно, до написання фрагмента коду, що забезпечує необхідну поведінку компонента в працюючій програмі.

Щоразу, коли Ви створюєте нову форму, створюється модуль. У програмі може бути і, найчастіше буває, не одна, а декілька - іноді кілька десятків форм і зв'язаних з ними модулів. У першому наближенні модулем можна вважати самостійний розділ програми, в чомусь подібний до глави в книзі. При компіляції програми Delphi створює файли з розширеннями pas, dfm і dcu для кожного модуля. Файл із розширенням pas містить копію тексту з вікна коду програми, у файлі з розширенням dfm зберігається опис змісту вікна форми, а в dcu-файлі - результат перетворення в машинні інструкції тексту з обох файлів. Файли dcu створюються компілятором і дають необхідну базу для роботи

компонувальника, що перетворить їх у єдиний файл, що завантажується, з розширенням exe.

Порожня форма і її модифікація

Як уже говорилося, вікно форми містить проект Windows-вікна програми. Важливо пам'ятати, що з самого початку роботи над новою програмою Delphi створює мінімально необхідний код, який забезпечує її нормальне функціонування в Windows. Таким чином, найпростіша програма готова відразу після вибору опції File | New | Application.

Зміна властивостей форми

Зміна заголовка

Змінимо заголовок вікна програми. За замовчуванням (тобто без будь-яких зусиль з нашої сторони) заголовок вікна збігається із заголовком форми: Form1. Щоб змінити заголовок форми потрібно звернутися до вікна Інспектора об'єктів і клацнути мишею по рядку Caption (Заголовок) сторінки Properties (Властивості) вікна Інспектора об'єктів. Тепер правий стовпчик цього рядка з текстом Form1 виділений кольором і праворуч від виділення бачимо текстовий курсор. Переключіть клавіатуру в режим введення кирилиці і введіть новий заголовок, наприклад МОЯ ПЕРША ПРОГРАМА, потім запустіть програму (натисніть *F9).* Новий прогін програми створить вікно із заголовком МОЯ ПЕРША ПРОГРАМА, що несе в собі набагато більше інформації, ніж просто Form1.

Отже, простою зміною змісту рядка у вікні Інспектора об'єктів можна змінити одну з властивостей вікна.

Зміна кольору форми

Змінимо колір вікна програми. За замовчуванням він збігається з кольором форми. Щоб змінити колір форми потрібно звернутися до вікна Інспектора об'єктів і клацнути мишею по рядку Color (Колір) на сторінці

Properties (Властивості) вікна Інспектора об'єктів, після чого в списку, що випадає (у правому стовпчику цього рядка) вибрати необхідний колір. Наприклад, якщо необхідно зробити форму жовтою, вибираємо clYellow, якщо червоною, тоді – clRed. У такий ж спосіб можна змінювати будь-які інші властивості форми.

Розміщення нового компонента

Для розміщення на формі нового компонента треба спочатку його вибрати (клацнути по ньому мишею) у палітрі компонентів, а потім клацнути мишею по точці робочого простору форми, у тому місці, де повинен розташовуватися лівий верхній кут обраного компонента.

Наприклад, спробуємо таким чином включити у вікно програми компонент Label (Мітка), призначений для розміщення різного роду написів. Для цього в палітрі компонентів необхідно вибрати сторінку Standard і клацнути мишею по кнопці  (ця кнопка відображає компонент Label у палітрі компонентів). Тепер клацніть мишею на формі в тому місці, де повинен розташовуватись лівий верхній кут мітки. Первісні розміри і положення компонента на формі легко змінюються мишею.

Новий компонент має стандартне ім'я Label1 і напис на ньому повторює це ім'я. Змінити цей напис можна за допомогою рядка Caption вікна Інспектория об'єктів. Для цього треба ввести новий напис (наприклад, Delphi) у правий стовпчик цього рядка. Як тільки Ви почнете вводити новий напис, вигляд компонента на формі почне змінюватись, динамічно відбиваючи всі зміни, зроблені Вами у вікні Інспектора об'єктів.

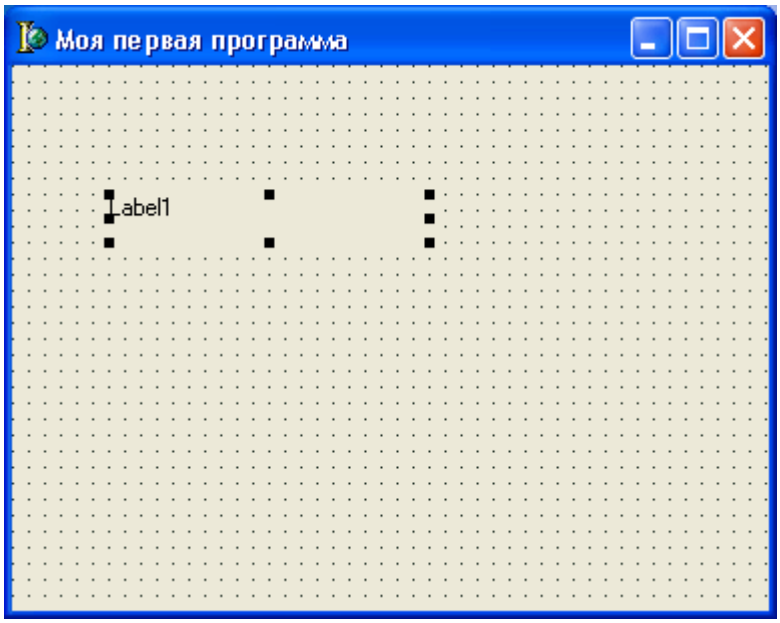

Рис. 10 - Розміщення компонента Label

Виділимо напис кольором і зробимо його шрифт більш великим. Для цього клацнемо мишею по властивості Font вікна Інспектора об'єктів і за допомогою кнопки в правій частині рядка розкриємо діалогове вікно настроювання шрифту. У списку size (Розмір) цього вікна выберемо висоту шрифту 24 пункта (пункт - 1/72 дюйма, тобто приблизно 0,04 мм, таким чином, 24 пункта означає висоту шрифту трохи більше 9 мм), а за допомогою списку Color (Колір) виберемо потрібний колір (наприклад, червоний), після чого закриємо вікно кнопкою ОК.

Напис на компоненті у вікні форми відразу ж відповідним чином змінить свої властивості. Delphi має здатність візуальної реалізації будь-яких змін властивостей компонента не тільки на етапі прогону програми, але й на етапі проектування форми.

Для переміщення компонента за формою необхідно клацнути мишею всередині чорних прямокутників, що обрамляють напис, і, не відпускаючи ліву кнопку миші, змістити її, після чого відпустити кнопку. Таким способом можна змінювати розташування компонента на формі.

За допомогою чорних квадратиків, які обрамляють, можна змінювати розміри компонента. Для цього треба розмістити показник миші над одним з них (у цей момент показник змінює свою форму на двунаправлену стрілку),

потім натиснути ліву кнопку миші і, не відпускаючи її, тягнути сторону або кут компонента в потрібному напрямку, після чого відпустити кнопку.

Усі видимі компоненти Delphi (у Delphi можуть використовуватися як видимі, так і невидимі компоненти. Невидимі компоненти не мають візуального відображення на етапі прогону програми) мають властивості Left (Ліворуч), Top (Зверху), Width (Ширина) і Height (Висота), числові значення яких визначають положення лівого верхнього кута компонента і його розмірів у так званих *пикселях*, тобто в мінімальних за розміром точках екрана, світністю яких може керувати програма. При переміщенні компонента або зміні його розмірів мишею, ці значення автоматично міняються і, навпаки, зміна цих властивостей у вікні Інспектора об'єктів приводить до відповідної зміни положення і розмірів компонента.

Реакція на події

Функціональність програми визначається сукупністю її реакцій на ті або інші події. У зв'язку з цим кожен компонент крім властивостей характеризується також набором подій, на які він може реагувати.

Розглянемо приклад реакції на подію. Для цього проведемо чергову модернізацію нашої першої програми: вставимо в її форму ще один компонент – „кнопку”, і змусимо її реагувати на подію - натискання лівої кнопки миші.

Компонент „кнопка” зображується піктограмою  на сторінці Standard палітри компонентів. Розмістемо цей компонент на формі і розташуємо його нижче мітки і всередині форми.

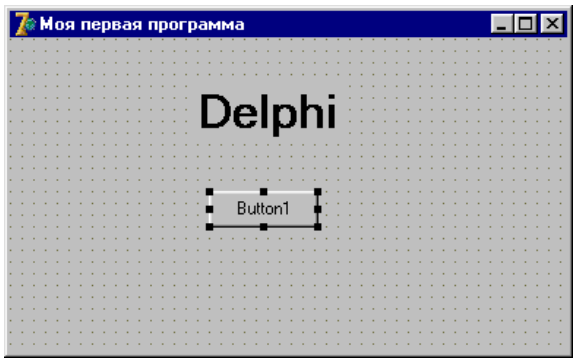

Рис. 11 - Форма з вставленою кнопкою

При клацанні по кнопці мишею в працюючій програмі виникає подія OnClick. Поки ця подія ніяк не обробляється програмою, тому “натискання” кнопки не приведе ні до яких наслідків. Щоб змусити програму реагувати на натискання кнопки, необхідно написати мовою Object Pascal фрагмент програми, що називається обробником події.

Цей фрагмент повинен являти собою послідовність текстових рядків, у яких програміст вказує, що саме повинна робити програма у відповідь на натискання кнопки. Фрагмент оформляється у вигляді спеціальної підпрограми мови Object Pascal - *процедури*.

Щоб змусити Delphi самостійно зробити заготівлю для процедури обробника події OnClick, треба зробити подвійне клацання мишею по компоненті, від якої ми чекаємо реакцію на подію. У відповідь *Delphi* активізує вікно коду, в якому можна побачити такий текстовий фрагмент:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

Слово procedure сповіщає компілятор про початок підпрограми-процедури (у Delphi можуть використовуватись також підпрограми-функції; у цьому випадку замість procedure (процедура) використовується слово function (функція)). За ним йде ім'я процедури TForm1.Button1Click. Це ім'я складене: воно складається з імені класу TForm1 і власне імені процедури Button1Click.

Класами в Delphi називаються функціонально закінчені фрагменти програм, що служать зразками для створення подібних собі екземплярів. Один раз створивши клас, програміст може включати його екземпляри (копії) у різні програми або в різні місця однієї і тієї ж програми. Такий підхід сприяє максимально високої продуктивності програмування за рахунок використання раніше написаних фрагментів програм. До складу Delphi входить кілька сотень класів, створених програмістами корпорації Borland (так званих стандартних класів). Сукупність стандартних класів визначає могутні можливості цієї системи програмування.

Кожен компонент належить до строго визначеного класу, а всі конкретні екземпляри компонентів, що вставляються у форму, одержують ім'я класу з доданим числовим індексом. У Delphi всі імена класів починаються з літери T. Таким чином, ім'я TForm1 означає ім'я класу, створеного за зразком стандартного класу TForm. На початку тексту у вікні коду знаходяться наступні рядки:

```
type
     TForm1 = class(TForm)
     Button1: TButton;
     Label1: TLabel;
     procedure Button1Click(Sender: TObject);
private
     { Private declarations }
     public
     { Public declarations }
     end;
     var Form1: TForm1;
```

Рядок

```
TForm1 = class(TForm)
```

визначає новий клас TForm1, що породжений (створений за зразком) від стандартного класу TForm.

Рядок

```
Form1: TForm1;
```

створює екземпляр цього класу з ім'ям Form1. Стандартний клас TForm описує порожнє Windows-вікно, тоді як клас TForm1 описує вікно з уже вставленими в нього компонентами *мітка* і *кнопка*. Опис цих компонентів містять рядки

```
Button1: TButton;
Label1: TLabel;
```

Вони вказують, що компонент Button1 (Кнопка1) являє собою екземпляр стандартного класу TButton, а компонент Label1 (Мітка 1) - екземпляр класу TLabel.

За ім'ям процедури TForm1.Button1Click у круглих дужках йде опис параметра виклику `Sender: TObject`

Параметр з ім'ям `Sender` належить класу `TObject.` Процедури можуть мати не один, а кілька параметрів виклику або не мати їх зовсім. Параметри

виклику (якщо вони є) служать для настроювання реалізованого в процедурі алгоритму на виконання конкретної роботи. За допомогою параметра `Sender` підпрограма `Button1Click` може при бажанні визначити, який саме компонент створив подію `OnClick.`

Весь рядок

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

у цілому називається заголовком процедури. його завершує символ “`;`”. Цей символ відіграє важливу роль у Object Pascal, тому що показує компілятору на кінець речення мови. З окремих речень складається весь текст програми. Наприкінці кожного речення треба ставити крапку з комою - це обов'язкова вимога синтаксису мови.

Наступні рядки визначають тіло процедури:

```
begin
end;
```

Слово `begin` (початок) сигналізує компілятору про початок послідовності речень, що описують алгоритм роботи процедури, а слово `end` (кінець) - про кінець цієї послідовності. У нашому випадку тіло процедури поки ще не містить опису будь-яких дій, що і не дивно: Delphi тільки створює заготівлю для процедури, і нічого “не знає” про те, для чого ця процедура призначена. Наповнити тіло процедури -задача програміста.

Тепер при натисканні кнопки `Button1` керування буде передаватися в тіло процедури. Між словами `begin i end` можна написати фрагмент програми, що буде виконуватися у відповідь на цю подію.

Наприклад, зробимо нашу кнопку “звучащою”: напишемо в порожньому рядку між словами `begin...end` наступне речення:

```
MessageBeep (0);
```

і зробимо прогін програми, попередньо включивши звукову систему комп'ютера. Тепер у відповідь на натискання кнопки Button1 у динаміку комп'ютера лунатиме звуковий сигнал, тому що вставлений рядок реалізує

звертання до стандартної процедури, що вміє витягати з динаміка різні стандартні для Windows звуки.

Динамічна зміна властивостей компонента

Оскільки кнопка `Button1` у нашій програмі здатна “звучати”, корисно змінити її напис: замість напису `Button1`, що автоматично формує Delphi за ім‘ям компонента, назвемо кнопку, наприклад, “Звук”. Простіше це зробити за допомогою вікна форми й Інспектора об'єктів, тобто на етапі конструювання форми (для цього потрібно просто змінити властивість `Caption` компонента Button1 у вікні Інспектора об'єктів), але ми розглянемо інший спосіб – динамічна зміна напису на етапі прогону програми. Зміни на етапі конструювання називаються статичними, а в ході прогону програми - динамічними. Для динамічної зміни напису створимо оброблювач події `OnCreate` (По створенню) для форми і змінимо в ньому цю властивість.

Подія `OnCreate` виникає після створення windows-вікна, але до появи цього вікна на екрані. Щоб створити обробник цієї події, необхідно розкрити список компонентів у верхній частині вікна Інспектора об'єктів, вибрати компонент `Form1` і двічі клацнути по властивості `OnCreate` на сторінці `Events` цього компонента (клацати потрібно по правій частині рядка `onCreate`). У відповідь Delphi знову активізує вікно коду і покаже заготівлю для процедури `TForm1.FormCreate`. Необхідно відредагувати її в такий спосіб:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Button1.Caption:= 'Звук';
end;
```

Єдиний вставлений рядок являє собою оператор присвоювання мови Object Pascal. У лівій частині оператора вказується властивість `Button1.Caption`, а в правій частині - значення `'звук'`, що ми хочемо дати цій властивості. Зв'язує обидві частин комбінація символів “: =”, що читається як “привласнити значення”. Символи “:=” завжди пишуться разом, без

поділяючих пробілів, хоча перед двокрапкою і після знака рівняння можна для кращої читаності програми вставляти пробіли. Як і будь-яке інше речення мови, оператор присвоювання завершується крапкою з комою.

Складене ім'я `Button1.Caption` необхідне для точної вказівки компілятору, про яку властивість йде мова: у нашій програмі використовуються три компоненти (включаючи саму форму), кожний з яких має властивість `Caption`; уточнюючий префікс `Button1` змусить змінити цю властивість у кнопки, а не в мітки або форми. Значення, що привласнюється властивості, є текстовим рядком. За правилами Object Pascal текстовий рядок повинен полягати в апострофи, що обрамляють. Всередині апострофів можна написати довільні символи - саме вони (без апострофів, що обрамляють) будуть визначати новий напис на кнопці.

Після чергового прогону програми Ви побачите змінений напис на кнопці.

Отже,

- Процес створення Delphi-програми розбивається на дві фази: фазу конструювання форми і фазу кодування.
- Конструювання форми здійснюється за допомогою вибору компонентів з палітри і розміщення їх на формі.
- Програміст може переміщувати будь-який розміщений на формі компонент і змінювати його розміри за допомогою миші.
- Щоб надати компоненту потрібні властивості, використовується сторінка Properties Інспектора об'єктів.
- Щоб компонент міг відгукуватись на ту чи іншу подію, програміст повинен створити обробник події і вказати його ім'я на сторінці Events Інспектора об'єктів.
- Обробник події оформляється у вигляді процедури, що має складене ім'я. Перша частина імені являє собою ім'я класу для форми, друга частина відокремлюється від першою крапкою і може бути довільною. Якщо Delphi автоматично формує заготівлю для обробника, то друга

частина імені являє собою об'єднання імені компонента й імені події без прийменника On.

- Тіло процедури обмежене словами begin ... end і складається з речень (операторів) мови Object Pascal. Наприкінці кожного речення ставиться крапка з комою.
- Для надання компоненту належної функціональності треба написати відповідний код у вікні кода.
- Будь-яку властивість будь-якого компонента можна змінювати статично, тобто на етапі конструювання або динамічно, тобто в ході виконання програми.

Розділ IV
ОСОБЛИВОСТІ ОБ‘ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Основні концепції ООП

Мова `Object Pascal` є об'єктно-орієнтованим розширенням мови `Pascal` і реалізує концепцію об‘єктно-орієнтованого програмування (ООП). Це означає, що функціональність додатка визначається набором зв'язаних між собою задач, кожна з яких стає самостійним об'єктом. Об'єкт має свої властивості (тобто характеристики, або атрибути), методи, що визначають його поводження, і події, на які він реагує. Одним з найбільш важливих понять ООП є клас. Клас являє собою подальший розвиток концепції типу і поєднує в собі завдання не тільки структури і розміру змінних, але і виконуваних над ними операцій. Об'єкти в програмі завжди є екземплярами того або іншого класу (аналогічно змінним визначеного типу).

Базою ООП є:

- інкапсуляція;
- спадкування;

- поліморфізм.

Інкапсуляція являє собою об'єднання даних і методів (підпрограм), що їх обробляють, всередині класу (об'єкта). Це означає, що в класі інкапсулюються (поєднуються і містяться всередину) поля, властивості і методи. При цьому клас одержує визначену функціональність, наприклад, забезпечуючи повний набір засобів для створення програми підтримки деякого елемента інтерфейсу (вікна `Windows,` редактора і т.п.) або прикладної обробки.

Спадкування — це процес породження нових об'єктів-нащадків від існуючих об'єктів-батьків, при цьому нащадок бере від батька всі його поля, властивості і методи. Надалі наслідувані поля, властивості і методи можна використовувати в незмінному вигляді або перевизначати (модифікувати).

Просте спадкування великого змісту не має, тому в об'єкт-нащадок додаються нові елементи, що визначають його особливість і функціональність. Видалити будь-які елементи батька в нащадку не можна. У свою чергу, від нового об'єкта можна породити наступний об'єкт, у результаті утвориться дерево об'єктів (ієрархія класів). На початку цього дерева знаходиться базовий клас `TObject,` що реалізує найбільш загальні для всіх класів елементи, наприклад, дії по створенню і видаленню об'єкта. Чим далі той або інший клас знаходиться в дереві від базового класу, тим більшою специфічністю він володіє.

Приклад оголошення нового класу:

```
TAnyClass = class (TParentClass)
//Додавання до класу TParentClass нових і перевизначення існуючих
//елементів
end;
```

Сутність поліморфізму полягає в тому, що методи різних класів можуть мати однакові імена, але різний зміст. Це досягається перевизначенням батьківського методу в класі-нащадку. У результаті батько і нащадок поводяться по-різному. При цьому звертання до однойменних методів різних об'єктів виконується аналогічно.

Класи й об'єкти

У мові `Object Pascal` класи — це спеціальні типи даних, використані для опису об'єктів. Відповідно *об'єкт*, що має тип будь-якого класу, є *екземпляром* цього класу або змінною цього типу.

Клас являє собою особливий тип запису, що має у своєму складі такі елементи (члени), як поля, властивості і методи.

Поля класу аналогічні полям запису і служать для збереження інформації про об'єкт. *Методами* називаються процедури і функції, призначені для обробки полів. *Властивості* займають проміжне положення між полями і методами. З одного боку, властивості можна використовувати як поля, наприклад, привласнюючи їм значення за допомогою оператора присвоювання; з іншого боку, всередині класу доступ до значень властивостей виконується методами класу.

Опис класу має наступну структуру:

```
Type <Ім'я класу> = class (<Ім'я класу-батька>)
private
<Приватні описи>;
protected
<Захищені описи>;
public
<загальнодоступні описи>;
published
<Опубліковані описи>;
end;
```

У наведеній структурі описами є оголошення властивостей, методів і подій.

Приклад опису класу:

```
type
TColorCircle = class (TCircle);
FLeft, FTop, FRight, FBottom: Integer;
Color: TColor;
end;
```

Тут клас `TColorCircle` створюється на основі батьківського класу `TCircle`. У порівнянні з батьківським, новий клас додатково містить чотири поля типу `integer` і одне поле типу `TColor`.

Якщо в якості батьківського використовується клас Tobject, що є базовим класом для всіх класів, то його ім'я після слова class можна не вказувати. Тоді перший рядок опису буде виглядати в такий спосіб:

```
type TNewClass = class
```

Для різних елементів класу можна встановлювати різні права доступу (видимості), для чого в описі класу використовуються окремі розділи, позначені спеціальними специфікаторами видимості.

Розділи private і protected містять *захищені* описи, що доступні всередині модуля, в якому вони знаходяться. Описи з розділу protected, крім того, доступні для породжених класів за межами названого модуля.

Розділ public містить *загальнодоступні* описи, що видимі в будь-якому місці програми, де доступний сам клас.

Розділ published містить *опубліковані* описи, що на додаток до загальнодоступних описів породжують динамічну (тобто під час виконання програми) інформацію про тип (Run-Time Type Information, RTTI). За цією інформацією при виконанні додатка виконується перевірка на приналежність елементів об'єкта тому або іншому класу. Одним із призначень розділу published є забезпечення доступу до властивостей об'єктів при конструюванні додатків. В Інспекторі об'єктів видні ті властивості, що є опублікованими. Якщо специфікатор published не зазначений, то він мається на увазі за замовчуванням, тому будь-які описи, розташовані за рядком із вказівкою імені класу, вважаються опублікованими.

Об'єкти як екземпляри класу з'являються в програмі в розділі var як звичайні змінні. Наприклад,

```
var
CCircle1: TColorCircle;
CircleA: TCircle;
```

Як і у випадку записів, для звертання до конкретного елемента об'єкта (полю, властивості або методу) вказується ім'я об'єкта й ім'я елемента, розділені крапкою, тобто ім'я елемента є *складеним*.

Приклад звертання до полів об'єкта:

```
var
CCircle1: TColorCircle;
begin
…
Ccircle1.FLeft:=5;
Ccircle1.FTop:=1;
…
end;
```

Тут наведене безпосереднє звертання до полів об'єкта, звичайно це робиться за допомогою методів і властивостей класу.

Поля

Поле класу являє собою дані, що утримуються в класі. Поле описується як звичайна змінна і може належати до будь-якого типу.

Приклад опису полів:

```
type TNewClass = class(TObject)
private
FCode: integer;
FSign: char;
FNote: string;
end;
```

Тут новий клас `TNewClass` створюється на основі базового класу `TObject` і одержує на додаток три нових поля `FCode, FSign` і `FNote`, що мають відповідно цілочисельний, символьний і строковий типи. Згідно з прийнятою угодою імена полів повинні починатися з префікса `F` (від англ. `Field` – поле).

При створенні нових класів клас-нащадок успадковує всі поля батька, при цьому видалення або перевизначення цих полів неможливе, але допускається додавання нових. Таким чином, чим далі за ієрархією який-небудь клас знаходиться від батьківського класу, тим більше полів він має.

Нагадаємо, що зміна значень полів звичайно виконується за допомогою методів і властивостей об'єкта.

Властивості

Властивості реалізують механізм доступу до полів. Кожній властивості відповідає поле, що містить значення властивості, і два методи, що

забезпечують доступ до цього поля. Опис властивості починається зі слова property, при цьому типи властивості і відповідного поля повинні збігатися. Ключові слова read і write є зарезервованими всередині оголошення властивості і служать для вказівки методів класу, за допомогою яких виконується читання значення поля, зв'язаного з властивістю, або запис нового значення в це поле.

Приклад опису властивостей:

```
type TNewClass = class(TObject)
private
   FCode: integer;
   FSign: char:
        FNote: string;
     published
   property Code: integer read FCode write FCode;
   property Sign: char read FSign write FSign;
        property Note: string read FNote write FNote;
     end;
```

Для доступу до полів FCode, FSign і FNote, що описані в захищеному розділі і недоступні іншим класам, використовуються властивості Code, Sign і Note, відповідно.

Методи

Метод являє собою підпрограму (процедуру або функцію), що є елементом класу. Опис *методу* схож на опис звичайної підпрограми модуля. Заголовок методу розташовується в описі класу, а сам код методу знаходиться в розділі реалізації. Ім'я методу в розділі реалізації є складеним і містить у собі тип класу.

Наприклад, опис методу Button1Click буде виглядати так:

```
    interface
    …
    type
  TForm1 = class(TForm)
         Button1: TButton;
         procedure Button1Click(Sender: TObject);
    end;
    …
    implementation
```

```
    …
    procedure TForm1.Button1Click(Sender:
TObjectit);
    begin
    Close;
    end;
```

Метод, оголошений у класі, може викликатися різними способами, що залежить від виду цього методу. Вид методу визначається модифікатором, що вказується в описі класу після заголовка методу і відокремлюється від заголовка крапкою з комиою. Деякі модифікатори:

`virtual` – віртуальний метод;

`dynamic` – динамічний метод;

`override` – переобумовлений метод;

`message` – обробка повідомлення;

`abstract` — абстрактний метод.

За замовчуванням усі методи, оголошені в класі, є *статичними* і викликаються як звичайні підпрограми.

Методи, призначені для створення або видалення об'єктів, називаються *конструкторами* і *деструкторами*, відповідно. Описи даних методів відрізняються від описів звичайних процедур тільки тим, що в їхніх заголовках є ключові слова `constructor` і `destructor.` Як імена конструкторів і деструкторів у базовому класі `Tobject` і багатьох інших класах використовуються імена `Create` і `Destroy.`

Перш ніж звертатися до елементів об'єкта, його потрібно створити за допомогою конструктора. Наприклад:

```
Object := TOwnClass.Create;
```

Конструктор виділяє пам'ять для нового об'єкта в "купі" `(heap),` задає нульові значення для порядкових полів, значення `nil` – для показника і полів-класів, строкові поля встановлює порожніми, а також повертає показник на створений об'єкт.

При виконанні конструктора часто також здійснюється ініціалізація елементів об'єкта за допомогою значень, переданих як параметри конструктора.

Приклад використання конструктора і деструктора:

```
  type
   TShape = class(TGraphicControl)
   private
FPen: TPen;
     procedure PenChanged(Sender: TObject);
   public
constructor Create(Owner: TComponent); override;
destructor Destroy; override;
…
   end;
  //Опис конструктора Create класу TShape
  constructor TShape.Create (Owner: TComponent);
  begin
inherited Create(Owner) ;// Ініціалізація успадкованих частин
Width := 65;          //Зміна успадкованих властивостей
Height := 65;
FPen := TPen.Create;  // Ініціалізація нових полів
FPen.OnChange := PenChanged;
  end;
```

У конструкторі класу-нащадка спочатку викликається конструктор батька, а потім виконуються інші дії. У класі-нащадку директива `override` (перевизначити) забезпечує можливість батьківському класу використовувати новий метод. Ключове слово `inherited` служить для виклику методів батьківського класу.

Повідомлення і події

В основі операційної системи `Windows` лежить використання механізму повідомлень, що "документують" усі вироблені дії, наприклад, натискання клавіші, пересування миші. Програма одержує повідомлення у виді запису заданого типу, обумовленого як:

```
  Type
  PMsg = TMsg;
  Msg = packed record
hwnd: HWND;
message: UINT;
wParam: WPARAM;
lParam: LPARAM;
time: DWORD;
pt: TPoint
end;
```

Поля цього запису містять наступну інформацію:

hwnd — дескриптор керуючого елемента, якому призначене повідомлення;

message — код повідомлення;

wParam і lParam — додаткова інформація про повідомлення;

time — час обробки повідомлення Windows;

pt — координати показника миші під час генерації повідомлення.

Система Delphi перетворює повідомлення у свій формат, для якого використовується запис наступного типу:

```
PMessage = ^TMessage;
TMessage = record
Msg: Cardinal ;
case Integer of
0: (
WParam: Longint;
LParam: Longint;
Result: Longint);
1: (
WParamLo Word;
WParamHi Word;
LParamLo Word;
LParamHi Word;
ResultLo Word;
ResultHi Word);
end;
```

Типи Msg, TMessage, а також константи, використані при посилці повідомлень, описані у файлах widows.pas і message.pas.

Для обробки повідомлень, що посилаються ядром Windows і різними додатками, використовуються спеціальні методи, описувані за допомогою модифікатора message, після якого вказується ідентифікатор повідомлення.

Метод обробки повідомлення обов'язково повинен бути процедурою, що має один параметр, який при виклику методу містить інформацію про повідомлення, що надійшло. Ім'я методу програміст вибирає самостійно, для компілятора воно не має значення. Метод може цілком або частково перекривати метод-предок, що обробляє це повідомлення. Якщо метод тільки модифікує метод-предок, то для виклику останнього використовується метод

inherited. При цьому не потрібно вказувати назву методу-предка і його параметри, тому що виклик буде виконаний автоматично.

Розглянемо як приклад обробку повідомлення Windows, що посилається при зміні розмірів вікна.

```
type
 TForm1 = class(TForm)
 // Оголошення методу обробки повідомлення
 procedure MyPaint(Var Param); message WM_Size;
 end;
 …
// Код методу обробки повідомлення
procedure TForm1.MyPaint(Var Param) ;
begin
// Виклик методу-предка
inherited;
// Очищення поверхні форми
Form1.Refresh;
// Виведення червоної рамки
Form1.Canvas.Pen.Color:=clRed;
Form1.Canvas.Brush.Style:=bsClear;
Form1.Canvas.Rectangle(0,0,Form1.ClientWidth, Form1.ClientHeight);
end;
```

За периметром форми виводиться червона рамка за допомогою процедури MyPaint, що є обробником повідомлення WM_size. Це повідомлення посилається при зміні розмірів вікна. У даному прикладі рамка перемальовується (разом з формою) тільки при зміні розмірів вікна, але не при його перекритті іншими вікнами, тому що в цьому випадку посилається повідомлення WM_Paint, що тут не аналізується. Параметр Param процедури ніде не використовується, але повинен бути зазначений у заголовку процедури.

Звичайно в Delphi не виникає необхідність обробки безпосередніх повідомлень Windows, тому що в розпорядження програміста надаються *події*, працювати з якими набагато простіше і зручніше.

Подія являє собою властивість процедурного типу, призначена для забезпечення реакції на ті або інші дії. Присвоювання значення цій властивості (події) означає вказівку методу, викликуваного при здійсненні події. Відповідні методи називають *обробниками подій*.

Приклад призначення обробника події:

```
Application. OnIdle: =IdleWork;
```

Як обробника події OnIdle, що виникає при простої програми, об'єкту програми призначається процедура Idlework. Тому що об'єкт Application доступний тільки при виконанні програми, то таке присвоювання не можна виконати через Інспектор об'єктів.

Події Delphi мають різні типи, що залежать від виду цієї події. Найпростішим є тип TNotifyEvent, що характерний для нотифікаційних (повідомляючих) подій. Цей тип описаний у такий спосіб:

```
type TNotifyEvent = procedure (Sender: TObject) of object;
```

і містить один параметр sender, що вказує об'єкт-джерело події. Багато подій більш складного типу, поряд з іншими параметрами, також мають параметр Sender.

Тому що події є властивостями, їхні значення можна змінювати в процесі виконання додатка. При цьому допускається призначати обробник події одного об'єкта іншому об'єкту або його події, якщо збігаються типи подій. Подібна можливість забезпечується за рахунок показника на клас.

Крім параметрів, що явно задаються, наприклад, параметра Sender, методу завжди передається показник на його екземпляр класу, що викликав. Цим показником є параметр Self.

Для посилки повідомлення віконним елементам керування можна використовувати функцію SendMessage. Посилка повідомлення може знадобитися у випадку, коли компонент через свої властивості не надає усіх своїх можливостей. Наприклад, список ListBox не має властивостей, що прямо керують горизонтальною смугою прокручування. Тому для відображення і приховання горизонтальної смуги прокручування можна послати списку відповідне повідомлення.

Функція

```
SendMessage(hwnd:HWND; Msg:Cardinal; WParam, LParam:Longint): Longint
```

посилає повідомлення віконному елементу керування, посилання (дескриптор) на який задана параметром hwnd. У Delphi дескриптор віконного елемента містить властивість Handle. Параметр Msg вказує код повідомлення, а параметри WParam і LParam містять додаткову інформацію про повідомлення, і їхні значення залежать від конкретного повідомлення.

Приклад:

```
Label1.Caption:=IntToStr(SendMessage(ListBox1.Handle,
LB_GetCount, 0, 0));
```

Тут списку ListBox1 посилається повідомлення LB_GetCount, що наказує списку видати число його елементів. Для доступу до списку використовується його дескриптор, значення якого містить властивість Handle. Оскільки для даного повідомлення додаткова інформація не потрібна, то значення двох останніх параметрів дорівнюють нулю.

Список літератури

1. Архангельский А.Я.Приемы программирования в Delphi (Версu 5-7).- М.:Бином, 2003.- 784с.
2. Архангельский А.Я.Программирование в Delphi 6. -М.:Бином, 2003.- 1120с.
3. Гофман В.Э., Хомоненко А.Д. Delphi 5.- Спб.: БХВ - Санкт-Петербург, 1999. - 800с.
4. Гофман В.Э., Хомоненко А.Д. Delphi 6.- Спб.: БХВ - Санкт-Петербург, 2001. - 1152с.:ил.
5. Гусева А. И.Учимся программировать Pascal 7.0. Задачи и методы их решения. - М.: Диалог МИФИ, 2003.- 256с.
6. Культин Н. Основы программирования Delphi 7. -СПб.:ВНУ-Санкт-Петербург, 2003.- 608с.
7. Понамарев В.Самоучитель Delphi 7 Studio - Спб.:ВНУ-Санкт-Петербург, 2003.- 512с.
8. Фаронов В.В. Delphi 3: Учебный курс. - М.: "Нолидж", 1998. -526с.

Додаток.

Палітра компонентів

У Палітру компонентів входять наступні сторінки:
Сторінка STANDARD
Сторінка ADDITIONAL
Сторінка WIN32
Сторінка SYSTEM
Сторінка DIALOGS
Сторінка WIN 3.1
Сторінка SAMPLES
Сторінка ACTIVEX
Компоненти для роботи з базами даних:
Сторінка DATA ACCESS
Сторінка DATA CONTROLS
Сторінка DBEXPRESS
Сторінка DATASNAP
Сторінка BDE
Сторінка ADO
Сторінка INTERBASE
Сторінка DECISION CUBE
Сторінка QREPORT
Компоненти для доступу до Інтернет:
Сторінка INTERNET
Сторінка FASTNET
Сторінка WEBSERVICES
Сторінка WEBSNAP
Сторінки INDY CLIENTS, INDY SERVERS, INDY MISC

СТОРІНКА STANDARD

На сторінці Standard палітри компонентів розташовані стандартні для Windows інтерфейсні елементи, без яких не обходиться практично жодна програма.

Frame - рамка. Нарівні з формою служить контейнером для розміщення інших компонентів. На відміну від форми може розміщуватися в палітрі компонентів, створюючи заготівлі компонентів.

MainMenu - головне меню програми. Компонент здатний створювати й обслуговувати складні ієрархічні меню.

PopupMenu - допоміжне або локальне меню. Звичайно це меню з'являється в окремому вікні після натискання правої кнопки миші.

Label - мітка. Цей компонент використовується для розміщення у вікні не дуже довгих однорядкових написів.

Edit - рядок уведення. Призначена для введення, відображення або редагування одного текстового рядка.

Memo - багаторядковий текстовий редактор. Використовується для введення і/або відображення багаторядкового тексту.

Button - командна кнопка. Оброблювач події OnClick цього компонента звичайно використовується для реалізації деякої команди.

CheckBox - незалежний перемикач. Клацання мишею на цьому компоненті в працюючій програмі змінює його логічну властивість Checked.

RadioButton - залежний перемикач. Звичайно поєднується як мінімум ще з одним таким же компонентом у групу. Клацання по перемикачу приводить до автоматичного звільнення раніше обраного перемикача в тій же групі.

ListBox - список вибору. Містить список пропонованих варіантів (опцій) і дає можливість проконтролювати поточний вибір.

ComboBox - комбінований список вибору. Являє собою комбінацію списку вибору і текстового редактора.

ScrollBar - смуга керування. Являє собою вертикальну або горизонтальну смугу, що нагадує смуги прокручування з боків

Windows-вікна.

GroupBox - група елементів. Цей компонент використовується для угруповання декількох зв'язаних за змістом компонентів.

RadioGroup - група залежних перемикачів. Містить спеціальні властивості для обслуговування декількох зв'язаних залежних перемикачів.

Panel - панель. Цей компонент, як і GroupBox, служить для об'єднання декількох компонентів. Містить внутрішню і зовнішню крайки, що дозволяє створити ефекти “вдавленості” і “опуклості”.

Actiontist - список дій. Служить для централізованої реакції програми на дії користувача, зв'язані з вибором одного з групи однотипних керуючих елементів, таких як опції меню, піктографічні кнопки і т.п.

СТОРІНКА ADDITIONAL

BitBtn - командна кнопка з написом і піктограмою.

SpeedButton - піктографічна кнопка. Звичайно використовується для швидкого доступу до тих або інших опцій головного меню.

MaskEdit - спеціальний текстовий редактор. Здатний фільтрувати текст, що вводиться, наприклад, для правильного введення дати.

StringGrid - таблиця рядків. Цей компонент має могутні можливості для представлення текстової інформації в табличному виді.

DrawGrid - довільна таблиця. На відміну від StringGrid осередку цього компонента можуть містити довільну інформацію, у тому числі малюнки.

Image - малюнок. Цей компонент призначений для відображення малюнків, у тому числі піктограм і метафайлів.

Shape - фігура. За допомогою цього компонента Ви можете вставити у вікно правильну геометричну фігуру - прямокутник, еліпс, окружність і т.п.

Bevel - крайка. Служить для виділення окремих частин вікна тривимірними рамками або смугами.

ScrollBox - панель зі смугами прокручування. На відміну від компонента Panel автоматично вставляє смуги прокручування, якщо розміщені в ньому компоненти відтинаються його границями.

CheckListBox - список множинного вибору. Відрізняється від стандартного компонента ListBox наявністю поруч з кожною опцією незалежного перемикача типу CheckBox, що полегшує вибір відразу декількох опцій.

Splitter - границя. Цей компонент розміщається на формі між двома іншими видимими компонентами і дає можливість користувачу під час прогону програми переміщати границю, що відокремлює компоненти один від одного.

StaticText - статичний текст. Відрізняється від стандартного компонента Label наявністю власного windows-вікна, що дозволяє обводити текст рамкою або виділяти його у виглядіі “утисненої” частини форми.

ControiBar - смуга керування.

ApplicationEvents - одержувач події. Якщо цей компонент поміщений на форму, він буде одержувати всі призначені для програми повідомлення Windows (без цього компонента повідомлення приймає глобальний об'єкт-програма Application).

ValueListEditor - редактор рядків, що містять пари ім'я - значення. Пари такого типу широко використовуються в Windows, наприклад, у файлах ініціації, у системному реєстрі і т. п.

LabeledEdit - комбінація однорядкового редактора і мітки.

ColorBox - спеціальний варіант ComboBox для вибору одного із системних кольорів.

Chart - діаграма. Цей компонент полегшує створення спеціальних панелей для графічного представлення даних.

ActionManager - менеджер дій. Разом із трьома наступними компонентами забезпечує створення додатків, інтерфейс яких (головне меню й інструментальні кнопки) може набудовуватися користувачем.

ActionMainMenuBar - смуга меню, опції якого створюються за допомогою компонента ActionManager.

ActionToolBar - смуга для розміщення піктографічних кнопок, створюваних за допомогою компонента ActionManager.

CustomizeDig - діалог настроювання. За допомогою цього компонента користувач може по своєму смаку настроїти інтерфейс працюючої програми.

СТОРІНКА WIN32

Сторінка Win32 містить інтерфейсні елементи для 32-розрядних операційних систем Windows 95/98/NT/2000.

TabControl - набір закладок. Кожна закладка являє собою прямокутне поле з написом і/або малюнком. Вибір тієї або іншої закладки розпізнається програмою і використовується для керування умістом вікна компонента.

PageControl - набір панелей із закладками. Кожна панель може містити свій набір інтерфейсних елементів і вибирається клацанням по зв'язаній з нею закладці.

ImageList - набір малюнків. Являє собою сховище для декількох малюнків однакового розміру.

RichEdit - багатостроковий редактор форматованого тексту. На відміну від компонента Memo сторінки Standard текст у компоненті RichEdit підкоряється правилам Розширеного Текстового Формату (RTF - Rich Text Format) і може змінювати такі свої характеристики, як шрифт, колір, вирівнювання і т.д.

TrackBar - регулятор. Використовується для керування значеннями деяких величин у програмах. Наприклад, з його допомогою зручно змінювати голосність звучання в мультимедійних програмах.

ProgressBar - індикатор процесу. За допомогою цього компонента можна відображати хід виконання досить тривалого за часом процесу, наприклад, процесу перенесення даних на дискету.

UpDown - цифровий регулятор. Дві кнопки цього компонента призначені для збільшення (верхня) або зменшення (нижня) зв'язаної з компонентом числової величини.

HotKey - керуюча клавіша. Компонент використовується для введення керуючих клавіш, таких як F1, Alt+A, Ctrl+Shift+1 і т.п.

Animate - мультиплікатор. Призначений для відображення послідовно змінюючих один одного кадрів зображень, що рухаються, (відеокліпів). Компонент не може супроводжувати відеокліп звуком.

DateTimePicker - селектор часу/дати. Цей компонент призначений для введення і відображення дати або часу.

TreeView - дерево вибору. Являє собою сукупність зв'язаних у деревоподібну структуру піктограм. Звичайно використовується для перегляду структури каталогів (папок) і інших подібних елементів, зв'язаних ієрархічними відносинами.

ListView - панель піктограм. Організує перегляд декількох піктограм і вибір потрібної. Цей компонент здатний розташовувати піктограми в горизонтальних або вертикальних рядах і показувати їх у великому або дрібному маштабі.

HeaderControl- керуючий заголовок. Являє собою горизонтальну або вертикальну смугу, розділену на ряд суміжних секцій з написами. Розміри секцій можна змінювати мишею на етапі роботи програми. Звичайно використовується для зміни розмірів стовпців або рядків у різного роду таблицях.

StatusBar - панель статусу. Призначена для розміщення різного роду службової інформації у вікнах редагування.

ToolBar - інструментальна панель. Цей компонент служить контейнером для командних кнопок BitBtn і здатний автоматично змінювати їхні розміри і положення при видаленні кнопок або при додаванні нових.

CoolBar - інструментальна панель. На відміну від ToolBar використовується як контейнер для розміщення стандартних інтерфейсних компонентів Windows, таких як Edit, ListBox, ComdoBox і т. д

PageScroller – панель, що прокручується. Служить для розміщення вузьких інструментальних панелей. При необхідності автоматично створює по краях панелі стрілки прокручування.

ComboBoxEx - компонент у функціональному відношенні подібний comboBox (сторінка standard), але може відображати в списку, що випадає, невеликі зображення.

СТОРІНКА SYSTEM

На цій сторінці представлені компоненти, що мають різне функціональне призначення, у тому числі компоненти, що підтримують стандартні для Windows технології міжпрограмного обміну даними OLE (Object Linking and Embedding -зв'язування і впровадження об'єктів) і DDE (Dynamic Data Exchange -динамічний обмін даними).

Timer - таймер. Цей компонент служить для відліку інтервалів реального часу.

PaintBox - вікно для малювання. Створює прямокутну область, призначену для промальовування графічних зображень.

MediaPlayer - мультимедійний програвач. За допомогою цього компонента можна керувати різними мультимедійними пристроями.

OleContainer - OLE-контейнер. Служить приймачем впроваджуваних об'єктів.

СТОРІНКА DIALOGS

Компоненти сторінки Dialogs реалізують стандартні для Windows діалогові вікна.

OpenDialog - відкрити. Реалізує стандартне діалогове вікно “Відкрити файл”.

SaveDialog - зберегти. Реалізує стандартне діалогове вікно “Зберегти файл”.

OpenPictureDialog - відкрити малюнок. Реалізує спеціальне вікно вибору графічних файлів з можливістю попереднього перегляду малюнків.

SavePictureDialog - зберегти малюнок. Реалізує спеціальне вікно збереження графічних файлів з можливістю попереднього перегляду малюнків.

І інші компоненти.

СТОРІНКА SAMPLES

Ця сторінка містить компоненти різного призначення.

Gauge - індикатор стану. Подібний компоненту ProgressBar (сторінка Win32), але відрізняється великою розмаїтістю форм.

ColorGrid - таблиця кольорів. Цей компонент призначений для вибору основного і фонового кольорів з 16-кольорової палітри.

SpinButton - подвійна кнопка. Дає зручний спосіб керування деякою числовою величиною.

SpinEdit - редактор числа. Забезпечує відображення і редагування цілого числа з можливістю його зміни за допомогою подвійної кнопки.

DirectoryOutLine - список каталогів. Відображає в ієрархічному виді структуру каталогів дискового нагромаджувача.

Calendar - календар. Призначений для показу і вибору дня в місяці.

СТОРІНКА ACTIVE X

Компоненти Active є “чужими” для Delphi: вони створюються іншими інструментальними засобами розробки програм (наприклад, C++ або Visual Basic) і впроваджуються в Delphi за допомогою технології OLE.

КОМПОНЕНТИ ДЛЯ РОБОТИ З БАЗАМИ ДАНИХ

У Delphi розвиті засоби побудови додатків, розрахованих на роботу з електронними архівами (базами даних).

Сторінка Data Access

На цій сторінці зібрані компоненти, що не залежать від використовуваного доступу до бази даних.

Сторінка Data Controls

Компоненти цієї сторінки призначені для візуалізації даних, їхнього введення і редагування.

Сторінка dbExpress

Компоненти представлені на цій сторінці, підтримують технологію dbExpress прямого доступу до деяких промислових серверів баз даних.

Присутні також і інші сторінки: DataSnap (компоненти, що реалізують взаємодію машин у локальній мережі або Інтернет у типовому для БД випадку, коли клієнт працює з вилученими даними), BDE, ADO, InterBase, Decision Cube, QReport

КОМПОНЕНТИ ДЛЯ ДОСТУПУ ДО ІНТЕРНЕТ

Сторінка Internet

Компоненти цієї сторінки забезпечують засоби зв'язку програми з глобальною комп'ютерною мережею Інтернет.

Сторінка FastNet

Компоненти цієї сторінки надають програмісту можливість використання різних протоколів для передачі ділових повідомлень і даних по локальним і/або глобальним мережам, у тому числі і по Інтернет.

Сторінка WebServices

Компоненти цієї сторінки підтримують технологію SOAP (Simple Object Access Protocol) для створення служб Web (це програма, що запускається сервером Web у відповідь на клієнтську вимогу).

Присутні також інші сторінки: Indy Clients, Indy Servers, Indy Misc. Розташовані на цих сторінках компоненти у функціональному плані дублюють компоненти сторінки FastNet, але мають деякі додаткові можливості.

Навчальне видання

Короткий курс програмування в середовищі Delphi (для студентів заочної форми навчання спеціальності 7.050201 “Менеджмент організацій” спеціалізації “Інформаційні системи в менеджменті”).

Укладач: Мірошниченко Ганна Анатоліївна

Редактор: М.З. Аляб‘єв

План 2004, поз. 66

Підп. до друку 05.07.04.	Формат 60х84 1/16	Папір офісний.
Друк на ризографі.	Умовн.-друк.арк.3,5	Обл.-вид.арк.4,5.
Замовл. №_____	Тираж 50 прим.	Ціна договірна

61002, Харків, ХНАМГ, вул. Революції, 12

Сектор оперативної поліграфії ІОЦ ХНАМГ

61002, Харків, вул. Революції, 12