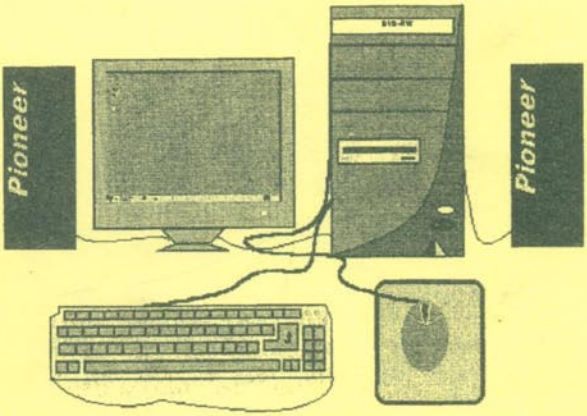
Л.М. Круподьорова

# Програмування мовою Delphi



Міністерство освіти і науки України
Вінницький національний технічний університет

Л.М. Круподьорова

# Програмування мовою Dephi

Затверджено Вченою радою Вінницького національного технічного університету як навчальний посібник для студентів напряму підготовки 0804–"Комп'ютерні науки" спеціальностей "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем". Протокол № 6 від 27 січня 2005 р.

Вінниця ВНТУ 2005

**УДК 681.31**
**К 84**

*Рецензенти:*

***С.В.Юхимчук***, доктор технічних наук
***О.М.Роік***, доктор технічних наук
***О.В.Сілагін***, к. т. н., директор ТОВ " І Т І "

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

**Круподьорова Л.М.**
К 84 **Програмування мовою Delphi.**
Навчальний посібник. - Вінниця: ВНТУ, 2005. - 153 с.

В навчальному посібнику розглянуті основи програмування мовою Delphi, історія виникнення цієї потужної системи програмування, викладені основні положення для освоєння програмування мовою Delphi, наведено багато прикладів і рекомендацій для їх засвоєння. Приділена увага різноманітним задачам з поясненням їх створення. Приведені варіанти індивідуальних завдань і приклади подібних програм. Навчальний посібник може бути використаний як при вивченні теоретичного матеріалу, так і при проведенні практичних і лабораторних робіт.

Навчальний посібник призначений для студентів напряму підготовки 0804–"Комп'ютерні науки" спеціальностей "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем" для оволодіння ними дисципліни " Основи програмування і алгоритмічні мови".

УДК 681.31

ЗМІСТ

# ВСТУП

Даний посібник є логічним продовженням раніше виданого посібника "Алгоритмізація і програмування", в якому розглядались основи програмування мовою Паскаль під керівництвом операційної системи MS DOS. Система програмування мовою DELPHI є продовженням нової версії для програмуваня під керуванням системи Windows. В зв'язку з цим, розглянемо, що привело до її виникнення.

Удосконалюючи Turbo Pascal, фірма Borland розробляла нові версії пакета. Спочатку мова Паскаль була створена як мова для навчання програмуванню. Згодом у систему програмування були внесені доповнення, що дозволяють створювати великі програмні проекти, що зробило її привабливою для професійних програмістів. Пізніше в Turbo Pascal з'явилися засоби, що забезпечують підтримку концепції об'єктно-орієнтованого програмування.

Еволюція технічних засобів персональних комп'ютерів привела до повсюдного витиснення ОС MS-DOS більш могутніми системами Windows, програмування для яких істотно складніше, ніж програмування для MS-DOS. З випуском системи Borland Pascal with Objects корпорація Borland надала в розпорядження програмістів дуже ефективні засоби розробки і налагодження програм, розрахованих для роботи під керуванням операційної оболонки Windows. Найбільш розповсюдженим інструментом розробки програм, орієнтованих на роботу в Windows, був C++ for Windows, призначений для професіоналів. У 1993 році фірма Microsoft випустила перше візуальне середовище програмування Visual Basic, і програмування для Windows стало простіше, ніж програмування для MS-DOS. Фірма Borland випустила аналогічний продукт, що одержав назву Delphi.

*Delphi* – це середовище розробки програм, орієнтованих на роботу в Windows. В основі ідеології Delphi лежить технологія візуального проектування і методологія об'єктно-орієнтованого програмування. Для подання програм у Delphi використовується розроблена Borland мова Object Pascal, в основі якої лежить Turbo Pascal. Слово "Object" особливо підкреслює, що мова підтримує концепцію об'єктно-орієнтованого програмування.

Перша версія, Delphi 1, працювала в середовищі Windows 3.1. Після появи Windows 95 Borland випустила спочатку 16-розрядну версію Delphi 2, потім більш досконалу 32-розрядну – Delphi 3 і її подальший розвиток – Delphi 4. На сучасному етапі вже існують версії Delphi 6 і Delphi 7. Всі вони мають одну основу, але з кожною новою версією можливості Delphi збільшуються.

В основі Delphi лежить концепція швидкого створення додатків (RAD - Rapid Application Development). Основною складовою середовища швидкого створення додатків є технологія, що одержала назву Two Ways Tools. Це значить, що при розміщенні чи зміні компонента в якій-небудь формі, відповідна програма автоматично доповнюється і модифікується. І

навпаки, усі зміни, що вносяться в програму при розробці додатка, автоматично відбиваються на функціональних властивостях компонентів форми.

Delphi являє собою складне середовище програмування, що містить безліч різних елементів. Середовище Delphi розроблене, насамперед, для професійних програмістів. Даний курс включає тільки початок програмування в середовищі Delphi. Перед студентами ставиться задача вивчити принципи побудови додатків для Windows за допомогою середовища візуального програмування,

Жирним шрифтом виділені нові поняття, які необхідно засвоїти. Знання цих понять буде перевірятися при тестуванні.

## 1 ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ

Основою Delphi є графічне середовище розробки додатків, назване інтегрованим середовищем розробки (Integrated Development Environment, IDE). Широковідомі в наш час додатки для Windows мають MDI (Multiple Document Interface), що визначає особливий спосіб керування декількох дочірніх вікон усередині одного великого вікна. Представником MDI-додатка є текстовий редактор Wore. Середовище Delphi є представником специфікації, названої Single Document Interface (SDI), і складається з декількох окремо розташованих вікон. SDI ближче до тієї моделі додатків, що використовується в Windows 95. Вікна можуть переміщатися по екрану, частково чи цілком перекривати один одного. Кожне вікно призначене для розв'язування визначених задач. На рис. 1 показаний вигляд екрана після запуску Delphi. На рис. 1 приведене вікно Delphi 6. Для інших версій вікна будуть мати незначні відмінності.

Після запуску Delphi на екрані відкриваються чотири вікна інтегрованого середовища розробки додатків.

Головні складові частини середовища програмування.

1. Головне вікно.
2. Вікно форми.
3. Вікно редактора коду програми.
4. Інспектор об'єктів.

У верхній частині екрана розташоване головне вікно (рис. 2). Воно має заголовок Delphi 6 – Projectl. У рядку заголовка головного вікна відображається ім'я відкритого в даний момент проекту: Projectl. У головному вікні знаходиться рядок головного меню, панелі інструментів і панелі компонентів. Головне вікно програми залишається відкритим увесь час, поки залишається завантаженою Delphi. Закрити головне вікно – означає закінчити роботу із системою програмування. В правому кутку знаходяться стандартні кнопки керуванням роботою вікна.

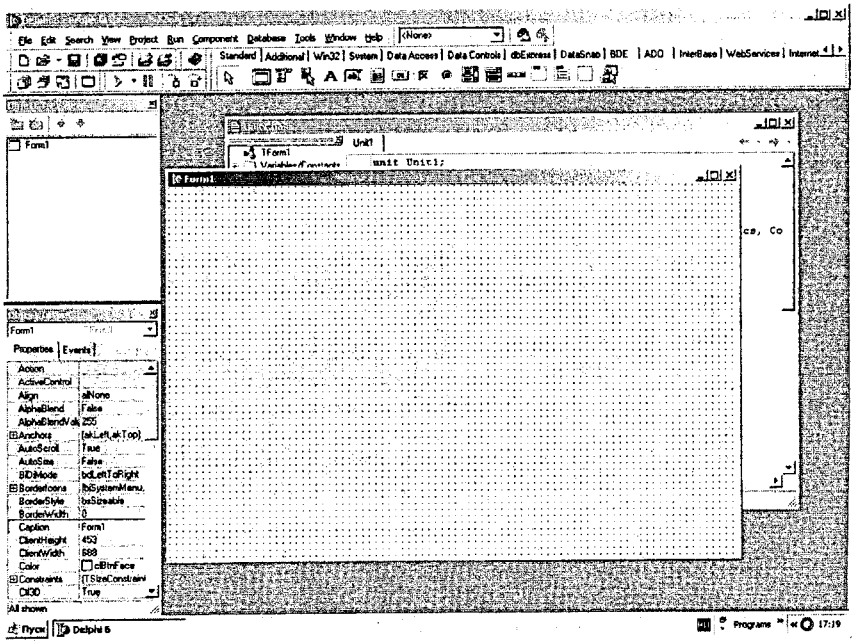


Рисунок 1 – Вигляд екрана комп'ютера після запуску Delphi

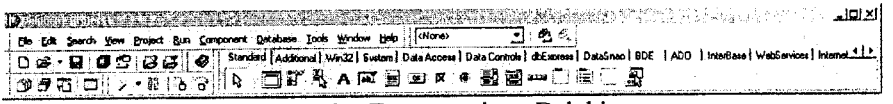


Рисунок 2 – Головне вікно Delphi

Рядок меню містить команди, необхідні для розробки і тестування додатків, а також керування ними.

Панель інструментів (рис. 3) містить кнопки, що відповідають визначеним командам меню, наприклад, командам File, View і ін. Натискання на яку-небудь із кнопок приводить до того ж результату, що і вибір відповідної команди в головному меню.

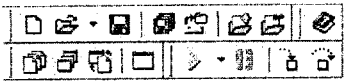


Рисунок 3 – Панель інструментів

У палітрі компонентів (рис. 4) відображаються компоненти, за допомогою яких користувач створює свої додатки. Компоненти є основними елементами кожного Delphi-додатка і, одночасно, основою **бібліотеки візуальних компонентів** – Visual Component Library (VCL). Вони дозволяють створювати інтерфейс користувача прикладних програм.

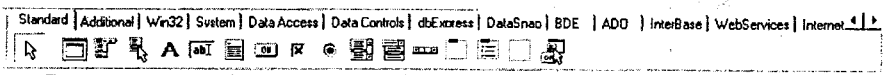



Рисунок 4 – Палітра компонентів з активною сторінкою Standard

**Під компонентом** розуміється деякий функціональний елемент, що має визначені властивості і який програміст розміщає у вікні форми. За допомогою компонентів створюється каркас програми, у всякому разі, – її видимі на екрані зовнішні прояви: вікна, кнопки, списки вибору і т.д. Після запуску Delphi активною є сторінка Standard палітри компонентів. Вибрати іншу сторінку можна клацанням на вкладці з відповідною назвою.

Вікно форми (рис. 5) є проектом Windows-вікна майбутньої програми. Форма – це вікно додатка на етапі розробки.

Спочатку вікно форми містить тільки стандартні для Windows інтерфейсні елементи – кнопки виклику системного меню, максимізації, мінімізації і закриття вікна, смугу заголовка і рамку, що окреслює вікно.

Уся робоча область заповнена точками координатної сітки, що служить для впорядкування розташовуваних на формі компонентів. Для кожного нового проекту автоматично створюється головне вікно майбутнього додатка і за замовчуванням воно має ім'я Forml.

Робота з формою в Delphi інтуїтивно зрозуміла. Створення інтерфейсу перетворюється в дитячу гру. Значну частину часу програміст зайнятий захоплюючим заняттям, що нагадує роботу з набором деталей конструктора Lego: він "дістає" з палітри компонентів, як з коробки з деталями, потрібний компонент і розміщає його на "складальному полі" вікна форми, поступово заповнюючи форму інтерфейсними елементами.

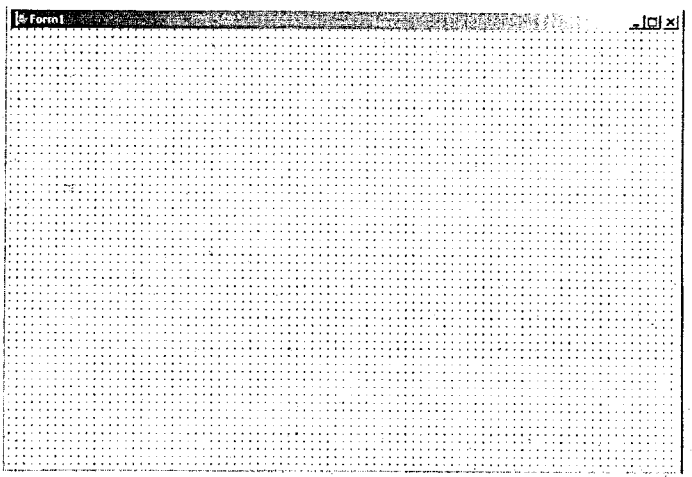



Рисунок 5 – Вікно форми

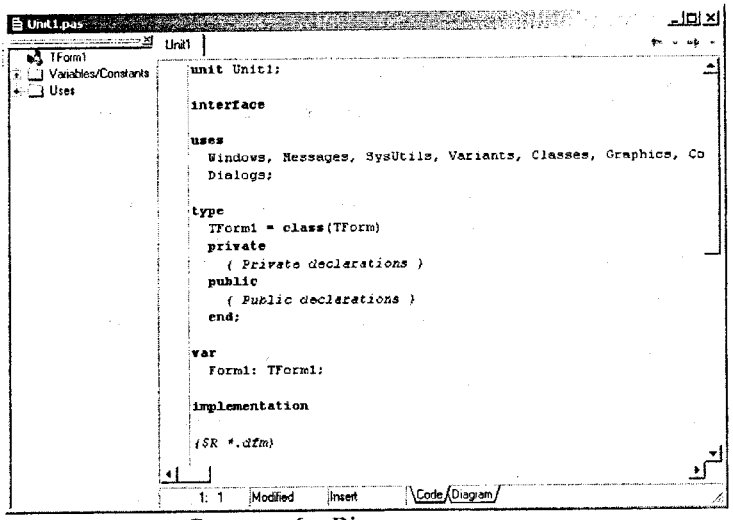



Рисунок 6 – Вікно коду програми

Вікно редактора тексту програми на рис. 1 розташовано за вікном форми. Воно призначено для створення і редагування тексту програми. Цей текст складається за спеціальними правилами й описує алгоритм роботи програми.

Створюючи програми на Delphi, програмісти постійно переходять від вікна форми до вікна програми і назад. Первісне вікно програми містить мінімальний вихідний текст, що забезпечує нормальне функціонування порожньої форми як повноцінне Windows-вікно. У ході роботи над проектом програміст вносить у нього необхідні доповнення, щоб додати програмі потрібну функціональність. Між умістом вікон форми і програми існує нерозривний зв'язок, що строго відслідковується Delphi. Розміщення на формі компонента приводить до автоматичної зміни тексту програми. Видалення тих чи інших автоматично вставлених фрагментів тексту програми може привести до видалення відповідних компонентів.

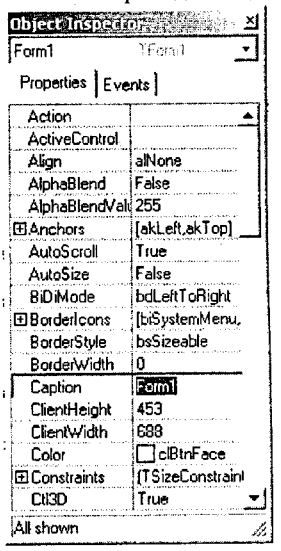



Рисунок 7 – Інспектор об'єктів

За допомогою вікна інспектора об'єктів (рис. 7) задаються і редагуються властивості й обробники подій компонентів. **Інспектор об'єктів** є інструмент, що використовується для формування зовнішнього вигляду і функціональних можливостей форми і компонентів у процесі розробки додатка. Любий 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 компонентом у вікні форми.

Вікно інспектора об'єктів містить дві сторінки, кожну з яких можна активізувати, виконавши клацання на вкладці з відповідною назвою.

Перша сторінка має назву Properties (властивості). Лівий стовпчик цієї сторінки містить список усіх властивостей компонента, що редагується, які доступні під час проектування. Друга сторінка називається Events (події). У її лівому стовпчику перераховані всі наявні оброблювачі подій компонента. Сукупність подій визначає поведінкову сторону компонента, тобто чи буде компонент реагувати на клацання миші чи натискання клавіші, як він буде поводитися в момент появи на екрані. У правих колонках обох сторінок можуть установлюватися значення відповідних властивостей чи оброблювачів подій.

Деякі властивості, відображені на сторінці Properties, мають початкові значення. Рядки таблиці вибираються клацанням миші і можуть відображати прості чи складні властивості. До простих властивостей відносяться властивості, обумовлені єдиним значенням – числом, рядком символів, значенням True чи False і т.д. Складні властивості визначаються сукупністю значень. Ліворуч від імені таких властивостей указується значок "+", а подвійне клацання мишею на імені властивості приводить до розкриття списку складових складної властивості. Для ряду властивостей компонентів у правому кінці рядка може з'явитися одна з кнопок: . Натискання кнопки приводить до появи на екрані діалогового чи вікна списку можливих властивостей.

В інспекторі об'єктів приведені тільки ті властивості, які має даний компонент під час проектування додатка. Повний список властивостей, які даний компонент має під час виконання додатка, можна одержати за допомогою системи підказок Delphi.

У верхній частині вікна інспектора об'єктів розташовується список, що розкривається і містить в собі всі поміщені на форму компоненти. Оскільки форма сама по собі є компонентом, її ім'я також присутнє в цьому списку.

## 2 ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

### 2.1 Розробка додатка в середовищі Delphi

Створення нової програми на Delphi починається з вибору опції File/ New Application. Це означає, що починається робота над новим додатком для Windows. У цьому випадку з'являється проект Windows-вікна програми (див. рис. 1). У вікні тексту програми поданий мінімально необхідний код, що забезпечує функціонування вікна в Windows. Найпростіша програма вже готова. Для того, щоб запустити першу програму, потрібно натиснути клавішу F9. Ця клавіша відповідає команді Run з пункту меню Run головного меню Delphi. При виконанні цієї команди програма послідовно проходить три головних етапи свого життєвого циклу – етапи компіляції, компонування і виконання. На етапі компіляції здійснюється перетворення підготовленого тексту програми в послідовність машинних інструкцій, на

етапі компонування до неї підключаються необхідні допоміжні підпрограми, а на етапі виконання готова програма завантажується в оперативну пам'ять і їй передається виконання.

Отримана програма нічого не вміє робити, крім як реагувати на натискання стандартних кнопок мінімізації, максимізації вікна і виведення системного меню. У заголовку отриманого вікна висвітиться ім'я вікна – Form 1. За замовчуванням заголовок вікна збігається з заголовком форми. Для того, щоб закрити працюючий додаток, необхідно натиснути стандартну кнопку закриття вікна Windows.

Звернувшись до інспектора об'єктів, можна змінити властивість Caption форми. Caption означає заголовок. Клацнувши мишею по рядку Caption вікна інспектора об'єктів, активізуємо цей рядок властивостей і в правому стовпчику запишемо новий заголовок "Моя перша програма на Delphi". Новий прогін програми створить вікно з заголовком "Моя перша програма на Delphi". За допомогою вікна інспектора об'єктів ми змінили одне з властивостей вікна програми – його заголовок.

Процес створення Delphi-програми розбивається на дві фази: фазу конструювання форми і фазу кодування.

Розмістимо на формі два компоненти з панелі Standard: мітку і кнопку. Для того щоб помістити компонент на форму, необхідно виконати такі дії:

1.Вибрати сторінку, на якій знаходиться потрібний компонент.

2.Вибрати компонент, якому необхідно розмістити у формі, клацнувши по ньому мишею.

3.Виконати клацання мишею в проектувальнику форми в тому місці, де потрібно даний компонент розмістити.

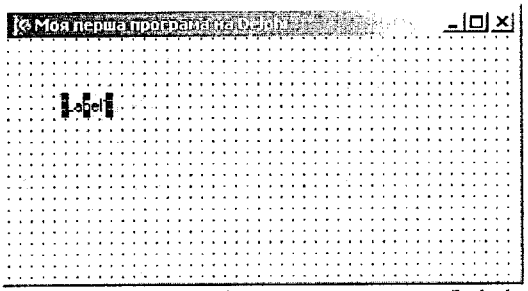



Рисунок 8 – Розміщення компонента Label

Компонент Label (мітка) призначений для розміщення різного роду написів. Компонент Label на панелі Standard відбиває кнопка **A** .Зробимо послідовно два клацання: на компоненті на панелі і на формі. Тепер форма містить компонент Label (рис. 8). Новий компонент має стандартне ім'я Label і напис на ньому повторює це ім'я. Змінити це ім'я можна за допомогою рядка Caption вікна інспектора об'єктів. Як тільки ви почнете вводити новий напис, вигляд компонента на формі почне змінюватись, динамічно відбиваючи всі зміни, вироблені у вікні інспектора об'єктів.

Виділимо напис кольором і зробимо її шрифт більш великим. Для цього необхідно клацнути мишею по властивості Font (шрифт) вікна інспектора об'єктів і відкрити в правій частині рядка діалогову властивість настроювання шрифту. У цьому діалоговому вікні можна змінити розмір, стиль і колір шрифту. Напис на компоненті у вікні форми відповідним чином змінить свої властивості.

За допомогою чорних квадратиків, що обрамляють, можна змінювати розміри компонента. Для цього варто помістити вістря покажчика миші над одним з них (у цей момент покажчик змінює свою форму на двонапрямлену стрілку), потім натиснути ліву кнопку миші і, не відпускаючи її, буксирувати сторону чи кут компонента в потрібному напрямку, після чого відпустити кнопку. Усі видимі компоненти мають властивості Left (лівий), Top (нижній), Width;(ширина), Height (висота), числові значення яких визначають положення лівого верхнього кутка компонента і його розмірів у пікселах. При буксируванні компонента чи зміні його розмірів мишею, ці значення автоматично зміняться. І навпаки - зміна цих властивостей у вікні інспектора об'єктів приводить до відповідної зміни положення і розмірів компонента. У Delphi значення Left і Top автоматично з'являються в невеликому вікні поруч з покажчиком миші при буксируванні компонента за формою.

Компонент кнопка зображується піктограмою  на сторінці Standard палітри компонентів і має назву TButton. Перша встановлювана кнопка за замовчуванням має назву Button 1 (рис. 9).

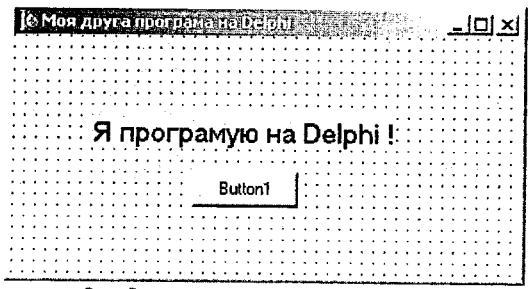



Рисунок 9 – Форма для другої програми на Delphi

При клацанні на кнопці мишею в працюючій програмі виникає подія OnClick (натискання). Щоб змусити додаток реагувати на натискання кнопки, потрібно написати мовою Object Pascal фрагмент програми, що називається оброблювачем події. **Оброблювач події** – фрагмент програми, що виконується у відповідь на визначену зміну в програмі чи в Windows. Цей фрагмент повинен являти собою послідовність операторів мови Object Pascal, оформлених у вигляді процедури. У вікні інспектора об'єктів на сторінці Events варто знайти подію OnClick і двічі клацнути мишею по правій частині рядка. Delphi самостійно зробить заготівку для оброблювача події Onclick. Активізується вікно коду з заголовком

процедури й операторними дужками:

```
Procedure TForm1. Button 1Click(Sender: TObject);
 Begin

 End;
```

Процедура має складене ім'я. Воно складається з імені класу TForml і власного імені процедури Button1Click. Процедури можуть мати параметри.

Клас у Object Pascal – це подальший розвиток об'єктного типу Турбо Паскаля. Написавши оброблювач події, ми додаємо ще один метод до методів, описаних у стандартних класах.

У тексті процедури запишемо такий оператор присвоювання:

```
Label1.Caption := 'Hi, тільки вчуся!';
```

Звертання до властивості компонента в програмі здійснюється так само, як звертання до поля чи до методу об'єкта за допомогою складеного імені.

Тепер, якщо запустити програму, то вона буде реагувати на натискання кнопки. Замість напису "Я програмую на Delphi!" з'явиться повідомлення "Hi, тільки вчуся!".

Текст програми у вікні коду програми:

```
UnitUnit1;
   interface
   uses
       Windows, Messages, SysUtils, Classes, Graphics,
       Controls, Forms, Dialogs;
   type
      TForm1 = class(TForm)
      Label1: TLabel;
      Button1: TButton;
      procedure Button1Click(Sender: TObiect);
      private
           { Private declarations }
      public
           { Public declarations }
   end;
   var
       Form1: TForm1;
       Implementation
      {$R*. DFM}
      procedure TForm1.Button1Click(Sender: TObiect);
          begin
              Label1.Caption := 'Hi, тільки вчуся!';
          end;
    end.
```

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 використаній в Delphi угоді, усі імена класів починаються з букви T. Таким чином, ім'я `TForm1` означає

ім'я класу, створеного за зразком стандартного класу TForm.

Рядок

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

визначає новий клас TForm1, що породжений від стандартного класу TForm1. Рядок

```
Form1 :TForm1;
```

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

```
Button1: TButton;
Label1: TLabel 1;
```

Компонент Button1 є екземпляром стандартного класу TButton, а компонент Label1 – екземпляр класу Tlabel.

Параметр процедури Sender належить класу TObiect. Він може використовуватися для визначення джерела події. У нашому прикладі він ніяк не використовується. Параметр Sender мають всі оброблювачі подій.

### 2.2 Введення – виведення даних у Delphi

Програма може одержувати вихідні дані з вікна введення, поля введення компонента, що має фокус уведення або з файлу.

Введення даних з вікна введення здійснюється викликом функції InputBox, що повертає значення рядка, введеного користувачем. У загальному вигляді інструкція для введення даних з використанням функції InputBox виглядає так:

```
Змінна := InputBox (заголовок, підказка, значення);
```

де: змінна – ім'я змінної рядкового типу, значення якої повинно бути отримане з вікна введення; заголовок – текст заголовка вікна введення; підказка – текст повідомлення, що пояснює; значення – текст, що знаходиться в полі введення в момент появи вікна введення на екрані.

Наприклад, у програмі перерахування ваги з фунтів у кілограми інструкція введення кількості фунтів може виглядати так:

```
S:=Inputbox('Фунти-кілограми','Введіть вагу в фунтах',' ');
```

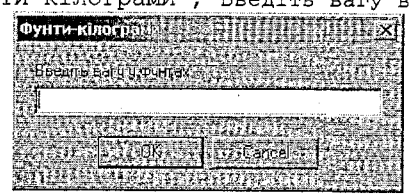



Рисунок 10 – Приклад вікна введення

Виконується ця інструкція в такий спосіб. На екран виводиться вікно введення (рис. 10).

Після того, як у полі введення буде введено число і виконане натискання по кнопці ОК, змінна S одержить значення. Значення функції InputBox рядкового типу. Якщо програмі треба одержати значення числового типу, то введений рядок повинен бути перетворений у число за допомогою відповідної функції перетворення.

Функції перетворення типів найбільше часто використовуються в інструкціях, що забезпечують введення і виведення інформації. Вони відрізняються від функцій перетворення стандартного Паскаля. У табл. 1 приведені функції перетворення, що найбільш часто зустрічаються.

У Delphi є компоненти, що мають фокус уведення, наприклад, компонент `Edit` і компонент `Memo`. У компонентів, що мають фокус уведення, є спеціальне поле для введення даних. Уведення даних з поля введення компонента `Edit` (рис. 11) здійснюється звертанням до властивості `Text` (текст) цього компонента. У цьому випадку введення даних з цього компонента може бути таким:

```
a := StrToFloat(Edit1.Text);
```

До вмісту компонента `TMemo` звертаються за допомогою властивостей цього компонента `Text` і `Lines` (рядка). Компонент `TMemo` є багаторядковим редактором. Текстова інформація в багаторядкових редакторах розташовується в декількох рядках. До рядка тексту звертаються, використовуючи властивість `Lines`. Властивість `Text` містить весь текст цілком.

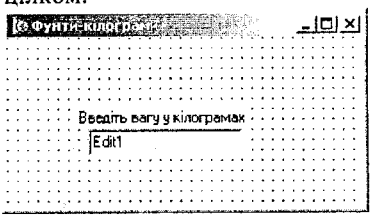



Рисунок 11 – Форма з одним компонентом – полем редагування

Таблиця 1 – Функції перетворення типів

| Функція | Значення |
|---|---|
| `IntToStr (n)` | Рядок, який є зображенням значення цілого n |
| `FloatToStr(n)` | Рядок, який є зображенням значення дійсного n |
| `FloatToStrF (n,f,l,m)` | Рядок, який є зображенням значення дійсного n. При виклику функції вказуються:<br>F –формат(спосіб зображення);<br>L – точність(потрібна загальна кількість цифр);<br>M–кількість цифр після десяткової точки; |
| `StrToInt (s)` | Ціле, зображенням якого є рядок s; |

Вивести результат програма може у вікно повідомлень, у поле виведення діалогового вікна, файл чи на принтер.

Виведення у вікно повідомлення може бути виконане викликом процедури `ShowMessage` чи функції `MessageDlg`.

Процедура `ShowMessage` дозволяє вивести на екран просте діалогове вікно з текстом і однією командною кнопкою. У загальному вигляді інструкція виклику процедури `ShowMessage` виглядає так:

`ShowMessage` (повідомлення);

де повідомлення – вираз рядкового типу.

Функція `MessageDlg` дозволяє забезпечити повідомлення однією зі стандартних позначок Windows, наприклад позначкою "Увага", задати кількість і тип командних кнопок. На рис. 12, наведений приклад вікна повідомлення, отриманого в результаті виконання інструкції

```
r := MessageDlg(' Дискримінант дорівнює нулю' + #13
+ 'Рівняння не має дійсних коренів.', mtInformation, [mbOK], 0);
```


Рисунок 12 – Приклад вікна повідомлення

Функція `MessageDlg` повертає результат – число, перевіривши значення якого можна визначити, вибором якої командної кнопки був завершений діалог. У загальному вигляді звертання до функції `MessageDlg` виглядає так:

Вибір := `MessageDlg` (повідомлення, тип, кнопки, контекст довідки);

Повідомлення – це текст виведеного повідомлення. Тип – це вид повідомлення. Повідомлення може бути інформаційним, попереджуючим чи повідомленням про помилку. Текст повідомлення задається іменованою константою. Іменовані константи, що визначають тип повідомлення, приводяться в таблиці 2.

Таблиця 2 – Іменовані константи функції MessageDlg

| Константа | Тип повідомлення і позначка |
|---|---|
| MtWarning | Увага–чорний знак оклику в жовтому трикутнику |
| MtError | Помилка – білий хрест в червоному крузі |
| MtInformation | Інформація – сині латинські букви на фоні білої хмаринки |
| MtConfirmation | Запит підтвердження – знак питання на фоні білої хмаринки |
| MtCustom | Звичайне повідомлення користувача без позначки |

Число і вид кнопок задається списком "кнопки", що може складатися з декількох розділених комами іменованих констант, записаних у квадратних дужках. Параметр "контекстсправки" визначає номер екрана довідкової системи, що з'явиться при натисканні на клавішу <F1> у той момент, коли вікно повідомлення знаходиться на екрані. Якщо використання довідкової системи непередбачено, то при виклику функції MessageDlg як параметр повинен бути зазначений нуль.

Інформацію можна вивести безпосередньо на форму, у компонент Label, діалогові вікна інших компонентів. Для виведення інформації використовуються відповідні властивості компонентів. Як було показано раніше (п.2.1), для виведення інформації за допомогою мітки використовується властивість Caption. У компоненті Edit для виведення інформації використовується властивість Text, у компоненті Memo – властивості Text і Lines.

Створимо додаток для переведення значень температури з градусів Цельсія в градуси Фаренгейта.

Побудову інтерфейсної частини почнемо зі зміни властивості Caption нової форми. Потім помістимо компонент TPanel на форму. Компонент TPanel являє собою контейнер загального призначення.

**Контейнером** називається компонент, призначений для розміщення на ньому інших компонентів. Контейнери призначені для рівномірного розміщення компонентів у вікні при зміні розмірів вікна. В інспекторі об'єктів на сторінці Properties властивості Align (вирівнювання) задамо значення alButtom, а рядок властивості Caption очистимо. Властивість Align вказує спосіб вирівнювання розташовуваного компонента щодо того контейнера, у якому він розміщається.

Порожня форма являє собою контейнер. Значення властивості означає, що панель притискається до нижньої границі форми і розтягується по всій її довжині. Висоту панелі змінимо таким чином, щоб на ній помістилися три компоненти: мітка TLabel, однорядковий редактор TEdit, кнопка TButton.

Компонент TLabel призначений для виведення різних повідомлень Властивості Caption дамо значення, яке коментує дії користувача. Для компонента TButton цій властивості задамо значення «ОК». У редакторі TEdit очистимо властивість Text і змінимо ширину цього компонента за своїм розсудом.

На форму помістимо багаторядковий редактор – компонентів TMemo. Його властивості Align задамо значення alClient, що означає що компонент заповнює усю вільну область контейнера (у даному випадку усю вільну область форми). В багаторядковому редакторі TMemo є інформація, що складається з одного слова Memo1. Можна очистити вікно редактора, якщо викликати діалогове вікно зміни властивості, двічі клацнувши мишею в рядку Lines інспектора об'єктів. Звернувшись до властивості Font цього компонента, змінимо розмір і колір шрифту тексту.

На цьому етап конструювання форми закінчується. Для того щоб додати формі функціональність, необхідно написати оброблювач події. При натисканні на клавішу «ОК», число з вікна однорядкового редактора TEdit повинне бути перераховане за формулою F=1.8C + 32, а результат перерахування відобразитися у TMemo. Оброблювач події OnClick (натискання лівої клавіші миші) може мати такий вигляд:

```
procedure TForm1.Button1Click(Sender: TObject);
    var x,y: real;
    begin
        x := StrToFloat(Edit1.Text);
        y := 1.8 *x  + 32;
Memo1.Lines.Add(Edit1.Text+'C='+FormatFloat('####.##',y)
+'F');
        Edit1. Clear;
        Edit1.SetFocus
    end;
```

У першому операторі присвоювання вміст вікна редактора перетвориться в дійсне число. Другий оператор присвоювання – формула перерахування. Для відображення результату розрахунку у вікні TMemo використовується метод Add цього компонента. Метод Add класу TStringList додає новий рядок до наявного в Lines набору рядків. Доданий рядок відображається на екрані. Параметром методу є рядкова змінна. Рядкова змінна складається з вмісту вікна редактора Edit1, рядкового подання дійсної змінної y і констант ' C= ' і ' F '. Функція FormatFloat форматує значення y за допомогою описаного формату (див. додаток 1 файлу матеріалів). Методи Clear і SetFocus дозволяють очистити вміст вікна редактора Edit1 і встановити маркер у вікно редактора.

Вікно додатка для переведення значень температури подане на рис. 13.

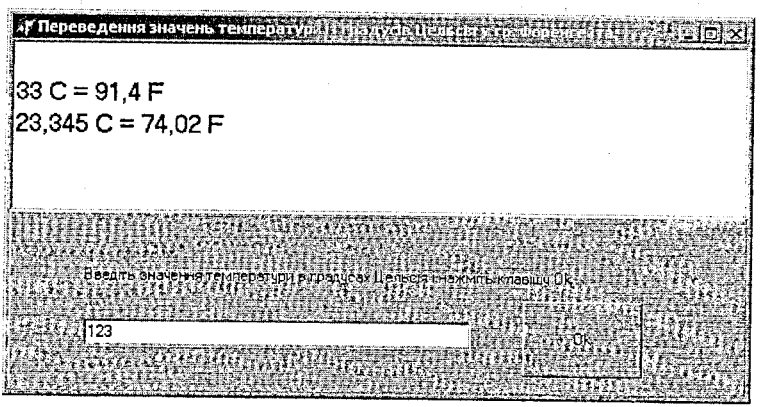



Рисунок 13 – Вікно додатка для переведення значень температури

# 3 СТРУКТУРА ПРОГРАМИ НА DELPHI

## 3.1 Windows як середовище розробки і виконання програм

У середовищі MS-DOS програмний код додатка складається з операторів, які виконуються один за одним. Код програми для Windows складається з процедур обробки повідомлень, що Windows посилає додатку. Windows фіксує події, що виникають в програмах і апаратурі, і потім посилає відповідні повідомлення в програму.

З позицій прикладної програми Windows сприймається як набір системних функцій, що утворить так званий **API – інтерфейс прикладних програм** (API-Application Programming Interface). Щоб виконати ту чи іншу дію (наприклад, відкрити вікно чи накреслити на екрані лінію), програма звертається до відповідної функції API.

До складу Windows входить більш 600 функцій. Код, який їх реалізує, зберігається в оперативній пам'яті і на дисках. Кожна API-функція має ім'я, подібно іменам підпрограм у бібліотеках компіляторів, і програма викликає відповідну функцію, посилаючись на її ім'я. Посилання на ім'я API - функції дозволяється не на етапі компонування програми, а безпосередньо в ході виконання програми.

Windows створює вхідне повідомлення для кожної вхідної події, яка генерується користувачем за допомогою миші або клавіатури. Windows зберігає вхідні дані в черзі системних повідомлень. Потім ці повідомлення посилаються в чергу повідомлень додатка. Додаток також може створювати власні повідомлення і поміщати їх у свою чергу повідомлень. Для обробки повідомлень головна програма додатка використовує безупинний цикл, так званий **головний цикл повідомлень**. Цей цикл містить деяку кількість функцій, призначених для обробки повідомлень. Вихід з цього циклу відбувається тоді, коли надходить повідомлення, що повинне завершити програму.

При користуванні тільки функціями Windows API мінімальна програма буде займати близько 60 рядків, складатися з декількох процедур і функцій.

```
Program WinMin;
   Uses WinTypes, WinProcs;
   Const AppName = 'WinMin';
   // Віконна функція
   Function WindowProc(Window:HWnd;Message,Wparam : Word;
      Lparam : LongInt) : LongInt; EXPORT;
      Begin
             WindowProc := 0;
             Case Message of
          // Обробка повідомлень
                    wm_Destroy :
                       Begin
                           PostQuitMessage(0);
                           Exit;
```

```
                    end;
        end;
WindowProc:=DefWindowProc(Window,Message,WParam, LParam);
  End;
  // Точка входу в програму
   procedure WinMain
   Var Window : HWnd; Message : TMsg;
   // Опис класу
   WindowClass : TWndGlass;
   Begin
          If HPrevInst = 0 Then
     // Тільки для першого класу програми
                  begin
     //Опис атрибутів класу вікна
           With WindowClass do
                Begin
                    Style := cs_HRedraw OR cs_WRedraw;
                    IpfnWndProc := @WindowProc;
                    cbClsExtra := 0;
                    cbWndExtra := 0;
                    hinstance := HInstance;
                   hicon:=Loadicon(0,idi_Application);
                   hCursor:= LoadCursor(0, idc_Arrow);
           brBackground:=GetStockObject(White_Brush);
                    IpszMenuName := ' ';
                    IpszClassName := AppName;
                end;
     // Реєстрація класу
          if not RegisterClass(WindowClass) then
          Halt(255);
       End;
      // Створення класу з визначеними атрибутами
        Window:=CreateWindow(AppName, 'WinMin',
        ws_OverlappedWindow,w_UseDefault,
       cw_UseDefault,cw_UseDefault,0,0,Hinstance,Nil);
          ShowWindow(Window,CmdShow);
          UpdateWindow(Window);
      // Цикл обробки повідомлень
            While GetMessage(Message, 0, 0, 0) do
                Begin
                    TranslateMessage(Message);
                    DispatchMessage(Message);
                End;
  Begin
       WinMain
  End.
```

Програма WinMin складається з функції WindowProc, процедури WinMain і викликів одинадцяти функцій Windows API. Процедура WinMain – це точка входу в програму, яка одержує керування від ядра Windows. Кожна Windows-програма (створювана з використанням тільки функцій Windows API) повинна мати таку процедуру. Функція

`WindowProc` – це спеціальна "віконна" функція, що обробляє повідомлення, що посилаються вікну. Ця функція викликається безпосередньо ядром Windows. Функції Windows API, які використані в цій невеликій програмі, виконують такі дії: для реєстрації класу викликається функція `RegisterClass`, функція `CreateWindow` створює вікно на базі зареєстрованого класу, функції `ShowWindow` і `UpdateWindow` призначені для відображення вікна, а цикл обробки повідомлень містить у собі виклики функцій `GetMessage`, `TranslateMessage` і `DispatchMessage`.

Windows-програма складається з трьох важливих частин:

- ініціалізація – реєстрація класу вікна, створення і відображення вікна;
- виконання – цикл обробки повідомлень;
- завершення – закриття вікна і повернення в середовище Windows.

У Delphi більшість стандартних повідомлень обробляються автоматично, оскільки всі об'єкти Delphi мають вбудований механізм – процедуру обробки повідомлень. У кожному з компонентів Delphi визначений механізм керування повідомленнями, який переводить усі повідомлення Windows, що посилаються визначеному об'єкту, у виклики методів.

Працюючи із середовищем Delphi, досить мати уявлення про те, як влаштована Windows-програма, а писати цикли обробки повідомлень, викликати функції реєстрації віконних класів і створення вікна немає необхідності. Усе це відбувається автоматично і приховане від програміста.

### 3.2 Структура простого проекту Delphi

Програма на Delphi – це сукупність файлів, яка названа проектом.

**Проект** – це сукупність файлів, що забезпечують розробку додатка в Delphi. Проект Delphi складається з файлу проекту, з файлів форм, модулів, установок параметрів проекту, ресурсів і т.д. Багато які з цих файлів створюються Delphi, коли тільки починається робота над проектом. Для розміщення кожного додатка, що створюється мовою Delphi, рекомендується створювати окремий каталог.

На рис. 14 представлене вікно, що містить файли одного простого додатка. **Файл проекту** має розширення `.dpr`. Він містить програму, написану мовою `Object Pascal`. Для кожного проекту може бути тільки один такий файл. У файлі проекту містяться посилання на всі форми проекту і модулі, що до них відносяться. Він зв'язує разом усі файли, з яких складається додаток, і, таким чином, середовище Delphi "знає", які файли необхідні для створення додатка. У файлі проекту також міститься код ініціалізації додатка. Цей файл створюється автоматично.

DPR-файл може бути відкритий для перегляду чи редагування за допомогою команди `View/Source` меню `Project`. Цей файл звичайно не редагується.

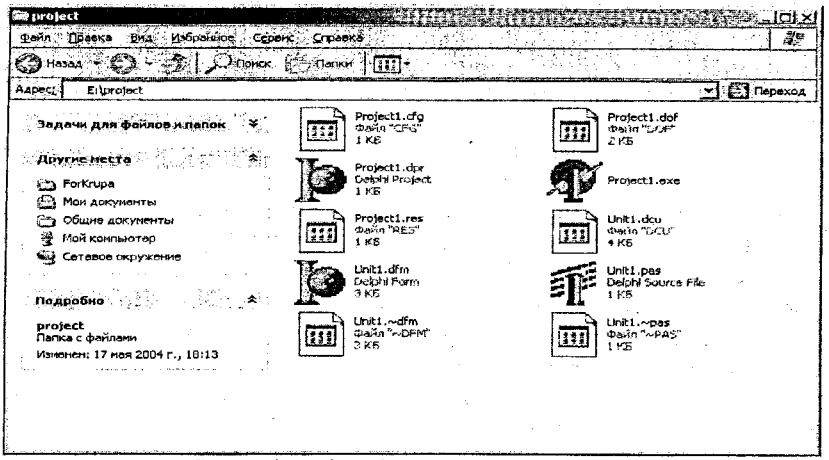



Рисунок 14 – Сукупність файлів одного з простих додатків на Delphi

Приклад програми:

```
program Calcul1;
     uses
        Forms,
           Calcu in 'Calcu.pas' {fmCalcu};
        {$R *.RES}
     begin
          Application.Initialize;
         Application.CreateForm(TfmCalcu, fmCalcu);
         Application. Run;
   End.
```

Рядки

```
uses
Forms,
Calcu in 'Catcu.pas' {fmCalcu};
```

указують, що крім файлу проекту в програмі повинні використовуватися модулі `Forms` і `Calcu`. Модуль `Forms` є стандартним, а модуль `Calcu` – новим, раніше невідомим, і Delphi у цьому випадку вказує також ім'я файлу з текстом модуля. Щоразу, коли до проекту додається нова форма чи новий модуль, Delphi автоматично додає директиву `Uses` у файл проекту.

Директива компілятору

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

містить вказівку компілятору на необхідність підключення до програми файлу ресурсів.

Тіло програми містить три виконуваних оператора:

```
Application.Initialize;
  Application.CreateForm(TfmCalcu, fmCalcu);
  Applicatfon.Run;
```

Кожний з них реалізує звертання до одного з методів об'єкта TApplication. В об'єкті TApplication зібрані дані і підпрограми, необхідні для нормального функціонування Windows-програми в цілому.

**Файл програмного модуля** (файл, що містить опис зв'язаної з модулем форми, процедур і функцій програмної логіки функціонування вікна додатка в синтаксисі Object Pascal) має розширення .Pas. Для кожної форми, що включається в проект, створюється окремий модуль. Саме в цьому файлі зберігається програма: оголошення змінних, типів, код оброблювачів повідомлень для інтерфейсних елементів, додатковий код і т.п. У проект можна включати і модулі, не зв'язані з формами.

**Файл форми** має розширення .Dfm. Він підключається безпосередньо до файлу, що виконується, у момент компіляції програми. Файл форми – це список властивостей усіх компонентів, включених у форму, значення яких були змінені в порівнянні зі значеннями, заданими за замовчуванням. Файл форми зв'язує графічне подання форми з оброблювачами повідомлень і використовується методом TApplication.CreateForm для початкового створення форми.

Файл із розширенням .Dfm є двійковим файлом, але його вміст може бути відображений на екрані у вигляді тексту. Для цього необхідно відкрити такий файл за допомогою команди Open меню File:

```
Object Form1: TForm1
        Left = 200
        Top = 108
        Width = 544
        Height = 375
        Caption ='Form1'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = ' MS Sans Serif '
        Font.Styte = []
        PixelsPerInch = 96
        TextHeight =13
    End
```

**Файл параметрів** проекту має розширення .Dof. Це текстові файли, що містять поточні установки проекту: настроювання компілятора і компоновщика, імена службових каталогів, умовні директиви і параметри командного рядка.

Dsk-файл містить Desktop-настроювання проекту. У цьому текстовому файлі зберігається інформація про те, які вікна відкриті й у яких позиціях вони розташовані. Цей файл дозволяє відновити зовнішній вигляд робочого середовища проекту.

Cfg-файл містить установки конфігурації проекту. Цей файл використовується компілятором при трансляції і має таке ж ім'я, як і файл проекту.

**Файл ресурсів** має розширення .Res. Це двійковий файл, що містить

усі необхідні для проекту ресурси, такі як, наприклад, піктограми, графічні зображення, курсори миші чи рядка. Цей файл створює і модифікує Delphi. Він не повинен змінюватися чи створюватися користувачем.

Розширення `. ~dp, ~df, ~pa` мають файли резервних копій.

Файл, що виконується, має розширення `.Exe`. Це автономний файл, що виконується, для якого більше нічого не потрібно, якщо тільки не використовуються бібліотеки, що динамічно підключаються.

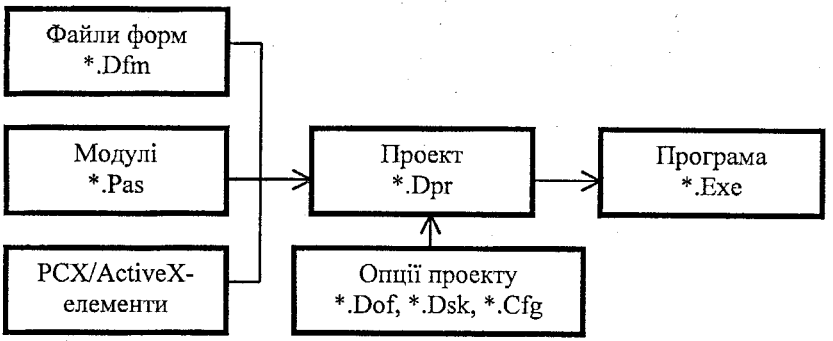



Рисунок 15 – Процес створення файлу, що виконується

Розширення `.Dcu` мають об'єктні файли модулів. Ці файли є відкомпільованою версією файлів програмних модулів.

Крім цих основних файлів проект Delphi може містити: бібліотеку, що динамічно приєднується, (`.Dll`); файли довідки (`.Hlp`); файли зображень (`.Wmf, .Bmp, .Ico`); анімаційні (файли (`.Avi`); файли, написані мовою Pascal більш ранніх версій або на інших мовах програмування й ін.

Процес створення файлу, що виконується, схематично представлений на рис. 15.

Компонент `OCX/Active` – це автономний, заздалегідь розроблений орган керування Windows

## 4 КЕРУВАННЯ ПРОЕКТАМИ

При завантаженні Delphi автоматично створюється новий проект. Можна використовувати цей проект для створення нового додатка чи відкрити вже існуючий проект або використовувати один з наданих середовищем шаблонів як основу для нового додатка. Для того щоб створити новий проект чи відкрити вже існуючий, зберегти проект, чи закрити його, необхідно скористатися командами, розташованими в меню `File: New, Open, Save, Save As, Save Project As, Save All, Close i Close All` (див. додаток В – файли матеріалів).

### 4.1 Менеджер проектів

**Менеджер проектів** – інструмент, що забезпечує доступ до усіх файлів, що містяться в проекті. Він дозволяє переключатися між файлами,

додавати чи видаляти модулі з проекту, при необхідності викликати файли в редактор для внесення змін. Менеджер проектів викликається командою `View/Project Manager` (рис. 16).

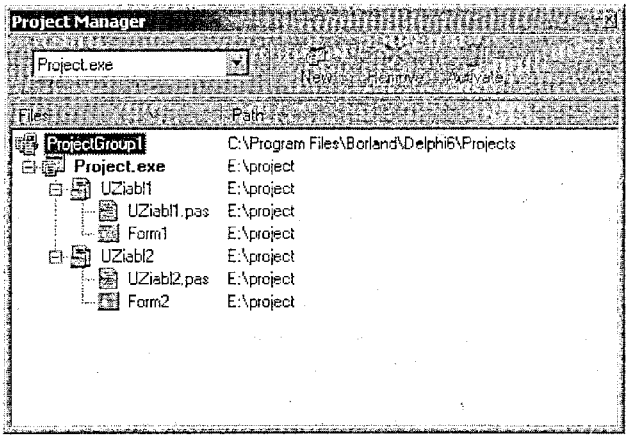



Рисунок 16 – Менеджер проектів Delphi

Смуга кнопок менеджера проекту надає доступ до таких команд:

`Add` – додати модуль до проекту;

`Remove` – видалити модуль із проекту;

`Unit` – додати новий модуль у проект;

`Form` – додати нову форму в проект;

`Options` – викликає діалогове вікно установки параметрів проекту;

Усі кнопки у вікні менеджера проекту дублюють пункти основного меню.

### 4.2 Компіляція, збирання і виконання програм

Результатом компіляції всіх Delphi-проектів є файл, що виконується. Для компіляції вихідних файлів, що входять у проект, використовується команда `Project/Compile` головного меню інтегрованого середовища розроблювача або комбінація клавіш `Ctrl+F9`. При цьому виконуються такі дії:

– компілюються файли з вихідним текстом усіх модулів, вміст яких змінився після останньої компіляції. У результаті для кожного файлу з вихідним текстом модуля створюється файл із розширенням `*.Dcu`. Якщо були внесені зміни в інтерфейсну частину модуля, то перекомпілюється не тільки цей модуль, але і модулі, що використовують його (через директиву `Uses`);

– після того, як відкомпільовані всі модулі, що входять у проект, Delphi компілює файл проекту і створює `*.Exe` файл з ім'ям, еквівалентним імені

проекту.

При збиранні проекту (на відміну від компіляції) компілюються усі файли, що входять у проект, незалежно від того, були в них внесені зміни після попередньої компіляції чи ні. Для збирання проекту використовується команда Project/Build All головного меню інтегрованого середовища розроблювача.

Для виконання програми використовується команда Run/Run головного меню інтегрованого середовища розроблювача чи клавіша F9. При виклику цієї команди відбуваються ті ж дії, що і при виклику команди Project/Compile і Project/ Build All, але після компіляції і зборки програма запускається на виконання.

### 4.3 Встановлення параметрів проекту

Існує два способи керування параметрами проекту. Можна встановити параметри за допомогою діалогової панелі Project Options, яка викликається за допомогою команди Project/Options головного меню інтегрованого середовища розроблювача, а можна скористатися директивами компілятора, розташованими у вихідному тексті програми.

Діалогова панель параметрів проекту (рис. 17) складається з декількох сторінок:

Forms – визначає головну форму, порядок створення інших форм;

Application – задає назву додатка, довідковий файл, іконку;

Compiler – визначає параметри компілятора;

Linker – визначає параметри компоновщика;

Directories/Conditions – задає робочі каталоги;

VersionInfo – установка параметрів версії проекту;

Packages – дозволяє визначити, які пакети доступні для використання при розробці додатка і прилінкувати їх при створенні файла результату.

У проекті Delphi може бути кілька форм, але одна з форм є головною. Головна форма є вихідною формою при запуску додатка і, як правило, першою відбивається на екрані. На сторінці Forms можна призначити кожну з існуючих у проекті форм головною. У списку Autocreate forms вказуються назви форм, що будуть створюватися одночасно з головною формою. Форми можуть створюватися й у процесі виконання програми.

Для того, щоб задати назву додатка, що буде відображатися разом з іконкою, коли вікно буде мінімізоване, необхідно вибрати параметр Title на сторінці Application. Тут же можна задати ім'я файлу, що містить довідкову систему для створюваного додатка, і іконку, що буде символізувати додаток.

На сторінці Compiler задаються параметри компілятора. Найбільш важливим з них є використання або скасування використання налагодженої інформації – цей параметр дуже впливає на розмір результуючого *.Exe-файла.

Призначення пунктів головного меню приводиться в додатку В файла матеріалів.

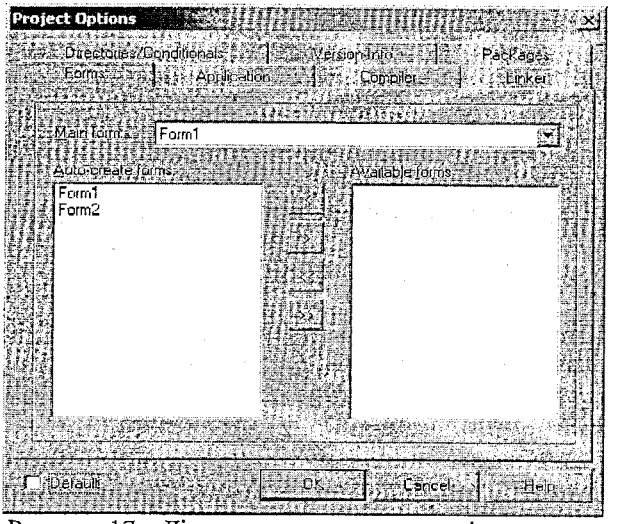



Рисунок 17 – Діалогова панель параметрів проекту

## 5 МОВА OBJECT PASCAL

Мовою програмування Delphi є `Object Pascal`. У тім, що стосується основних програмних конструкцій, ця мова дуже нагадує Pascal сьомої версії середовища Borland Pascal. Під програмними конструкціями варто розуміти структури, що визначають, у якій послідовності виконуються інструкції в програмі. Прикладами програмних конструкцій можуть служити умовні оператори `If-Then-Else`, оператори циклу `Repeat-Until`, а також механізми виклику методів. Перенос у Delphi програм, написаних мовою Pascal, виконати не завжди просто. Причина полягає в численних змінах, внесених у мову, яка використовується в середовищі розробки додатків Borland Pascal.

`Object Pascal` є об'єктно-орієнтованою мовою програмування. Додаток Delphi складається з об'єктів. Всі елементи керування інтерфейсу програми (кнопки, списки, панелі і т.д.) є об'єктами. Об'єкти забезпечують виконання і стандартних, і специфічних функцій додатка.

Удосконалену модель об'єктного типу Turbo Pascal являють собою класи `Object Pascal`. Класи `Object Pascal` мають багато загального з типом Object мови Turbo Pascal.

### 5.1 Класи

**Клас –** це визначений користувачем тип даних, що має внутрішні дані і методи у формі процедур чи функцій і звичайно описує родові ознаки і способи поводження ряду схожих об'єктів. Екземпляр типу "клас" називається **об'єктом.** Об'єкти класу завжди розподіляються разом на

відміну від екземплярів об'єктного типу. Попередньо визначені об'єкти, використовувані в програмі (такі як, наприклад, компоненти Delphi), – це в дійсності екземпляри класів.

У Object Pascal існує також тип Object. До введення терміна "клас" у мові Rascal існувала двозначність визначення "об'єкт", що міг позначати і тип і змінну цього типу. У Object Pascal існує чітка межа: клас – це опис, об'єкт – те, що створено відповідно до цього опису.

Тип "клас" – це структура даних, що складається з полів, методів, властивостей. Поля містять дані визначеного типу. Методи – це функції і процедури, описані всередині класу і призначені для операцій над його полями. **Властивості** – це спеціальний механізм класів, що регулює доступ до полів. Властивість описує один чи два методи, що здійснюють деякі дії над даними того ж типу, що і властивість. Наприклад, звичайна кнопка у вікні додатка має такі властивості, як колір, розміри, положення. Для екземпляра класу "кнопка" значення цих атрибутів задаються за допомогою властивостей – спеціальних змінних, обумовлених ключовим словом `Property`. Колір може задаватися властивістю `Color`, розміри – властивостями `Width` і `Height` і т.ін.

Оскільки властивість забезпечує обмін даними з зовнішнім середовищем, то для доступу до її значеннь використовуються спеціальні методи класу.

Наприклад:

```
Type
TAnObject = Class(TObject)
  Function GetColor: TSomeType;
  Procedure SetColor (aNewValue : TSomeType);
  Property AColor : TSomeType Read GetColor Write SetColor;
End;
```

У даному прикладі доступ до значення властивості `AColor` здійснюється через виклики методів `GetColor` і `SetColor`. До методів у явному вигляді не звертаються. Досить записати:

```
AnObject.Acolor := Avalue;
    Available := AnObject.Acolor;
```

і компілятор відтранслює ці оператори у виклики методів.

Особливим видом властивостей є події. Події для середовища Windows – це специфічні повідомлення про виниклу ситуацію, що перехоплюються й обробляються Windows, щоб забезпечити функціональні можливості інтерфейсу.

У `Object Pascal` **подія** – це властивість процедурного типу, призначена для створення реакції користувача на ту чи іншу вхідну дію:

```
Property OnMyEvent : TMyEvent Read FOnMyEvent Write
FOnMyEvent;
```

де `FOnMyEvent` – поле процедурного типу, що містить адресу деякого методу. Привласнити такій властивості значення – значить вказати об'єкту адресу методу, що буде викликатися в момент настання події. Такі методи називають оброблювачами подій (**оброблювач події** – фрагмент програми,

що виконується у відповідь на визначену зміну в програмі чи в Windows).

Кожен новий клас у Delphi повинен бути оголошений. Для цього використовується зарезервоване слово `Class`. На відміну від інших типів, тип `Class` можна повідомляти тільки глобально. Оголошення визначає функціональні можливості класу. Оголошення класів у модулі виробляється в розділі оголошення типів. Приклад оголошення класу й об'єкта в програмі на Delphi:

```
Type
   TForm1 = Class(TForm)
   Label1 : TLabel;
   Label2 : TLabel;
   CloseBtn : TBitBtn;
   OkBtn : TBitBtn;
End;
Var Form1 : TForm1;
```

В оголошенні типу визначено новий клас – `TForm1`, наслідуваний від класу `TForm`, що міститься в бібліотеці візуальних компонентів. На це вказує зарезервоване слово `Class`. Даний тип містить покажчики на компоненти, що були вміщені у форму: два компоненти `Label` –об'єкти типу `TLabel` (інакше кажучи, екземпляри класу `TLabel` ) і два екземпляри класу `TBitBtn`. Для визначення екземпляра нового класу оголошена змінна `Form1`.

Область видимості ідентифікатора компонента, оголошеного в описі класу, простягається від його оголошення до кінця визначення класу, а також поширюється на всі нащадки цього класу і на всі блоки реалізації методів класу. Область видимості ідентифікатора компонент залежить від атрибута видимості розділу, в якому оголошений цей ідентифікатор.

В оголошеннях типів класів є розділи приватних (`private`), загальних (`public`), захищених (`protected`) і опублікованих (`published`) оголошень.

У розділі приватних оголошень розміщаються поля даних і методи, недоступні за межами модуля, що містить оголошення даного класу. Дані, описані в цьому розділі, можуть оброблятися тільки шляхом виклику методів усередині класу, а також усередині даного модуля.

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

Поля властивості і методи секції protected також доступні тільки усередині модуля з описуваним класом, але вони доступні в класах, що є нащадками даного класу, у тому числі й в інших модулях.

Усі класи `Object Pascal` породжені від єдиного батька – класу `TObject`. Цей клас не має полів і властивостей, але містить у собі методи самого загального призначення, що забезпечують весь життєвий цикл

будь-яких об'єктів – від їхнього створення до знищення. Новий клас можна створити на основі цього базового класу.

У приведеному нижче тексті програми визначається новий тип класу, призначеного для аналізу даних, що представляють собою послідовність результатів вимірювань.

```
Type
   TSeries = Class(TObject);
       Public
       {Поля даних}
        NumberOf Samples, NumberOfMaterlals : Integer;
        IsStandard, IsNoStandard : Boolean;
        MaterialName: TStringList;
        {Методи}
         Function CountSamples : Integer;
         Function CountMaterials : Integer;
         Function Standard : Boolean;
         Procedure GetNames;
   End;
```

За результатами серії експериментів необхідно одержати таку інформацію: число вимірювань, число визначених матеріалів у серії, які результати лежать у межах допуску і які поза ним. Існують також назви випробовуваних матеріалів. Як поля даних нового об'єкта визначаються такі змінні: два поля цілочисельного типу; логічне поле; поле довгого рядка для назви.

Для забезпечення доступу до даних пишеться програмний код, що обробляє ці дані. Код реалізації цих методів міститься в implementation-секції модуля. В оголошенні класу міститься тільки заголовок процедури чи функції. Процедура `GetNames` призначена для визначення назви матеріалу. Для розв'язування інших задач описуються функції.

Принцип спадкування приводить до створення розгалуженого дерева класів, що поступово розростається при переміщенні від `TObject` до його нащадків. Кожен нащадок доповнює можливості свого батька новими властивостями і передає їх своїм нащадкам.

Для прикладу на рис. 18 показаний невеликий фрагмент дерева Delphi.

Клас `TPersistent` збагачує можливості свого батька `TObject`: він уміє зберігати дані у файлі й одержувати їх з нього, у результаті це вміють робити і його нащадки.

Клас `TComponent`, у свою чергу, уміє взаємодіяти із середовищем розроблювача і передає це уміння своїм нащадкам. `TControl` не тільки здатний працювати з файлами і середовищем розроблювача, але він уже вміє створювати й обслуговувати видимі на екрані зображення, а його нащадок `TWinControl` може створювати Windows-вікна і т.ін. До складу Delphi входить більш 300 різних класів.

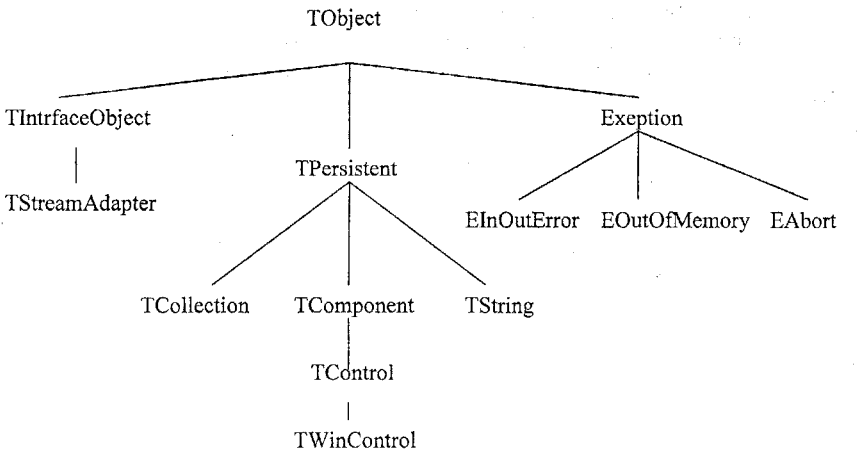



Рисунок 18 – Фрагмент дерева класів Object Pascal

### 5.2 Типи даних у Object Pascal

Object Pascal успадкував від базової мови Borland Pascal усі його особливості і достоїнства.

Так само, як і в Borland Pascal, у Delphi усі використовувані типи даних можна розділити на дві основні групи. До першого відносяться стандартні типи, визначені специфікацією мови Pascal. На основі цих типів розроблювач може описувати власні типи даних, що відносяться до другої групи.

До простих числових типів відносяться: Integer, Cardinal, ShortInt, SmallInt, LongInt, Int64, Byte, Word, LongWord, Real, Real48, Single, Double, Extended, Comp, Currency. Для процесу навчання досить використання Integer для збереження цілочисельних значень і Real для дійсних значень.

Для символьних даних базовим є тип Char. На його основі створені два додаткових типи – AnsiChar і WideChar для використання двох кодувань символів ANSI і Unicode.

У Delphi 6 розширені можливості логічного типу даних. Крім базового типу Boolean можна використовувати додаткові типи ByteBool, WordBool, LongBool. Ці типи необхідні для забезпечення сумісності з функціями Windows і іншими середовищами розробки.

Уведено нові типи для збереження рядкової інформації. Тип AnsiString призначений для створення змінних, що зберігають тексти довільної довжини. Тип WideString здійснює динамічне виділення пам'яті для збереження тексту. Текст повинен бути поданий у кодуванні Unicode. Багато функцій і процедур при роботі з рядковими даними

відрізняються від функцій і процедур Турбо Паскаля (див. додаток А файлу матеріалів).

Масиви стали відкритими, їхній розмір можна задавати динамічно. У четвертій версії Delphi з'явилися багатовимірні динамічні масиви.

Статичні масиви можна описувати за старими правилами, наприклад:

```
Var a : Array [1 .. 5] Of Real;
```

При описі параметрів функції чи процедури допускається такий варіант:

```
Function Prim (a : Array Of Real) : Real
```

Масиви у функції передаються як динамічні, тобто разом з адресою масиву передається і число елементів масиву.

У Object Pascal довжина динамічних масивів визначається під час виконання програми. Описавши змінну як `a1 : Array Of Real;`, ми маємо покажчик, що буде відповідати масиву дійсних чисел. Пам'ять під нього виділяється процедурою `SetLength`:

```
SetLength(a,5);
```

У динамічних масивах нумерація починається тільки з 0. Хоч це дійсно покажчики, знак розіменування ( ^ ) не застосовується.

Подальшим розвитком ідеї динамічних масивів є багатовимірні динамічні масиви:

```
Var aa : Array Of Array Of Real;
```

У функції `SetLength` після імені змінної потрібного типу можна задати відразу усі розмірності масиву, наприклад:

```
SetLength(aa, 10, 5).
```

У практиці програмування зустрічаються ситуації, коли передане змінній значення може мати невідомий заздалегідь тип даних. Для таких випадків в Delphi існує тип **Variant. Варіант** – тип даних, що дозволяє зберігати дані будь-яких типів, крім структурних. Змінна-варіант займає в пам'яті додаткові два байти, у яких міститься інформація про дійсний тип змінної. Ця інформація дозволяє компіляторові створити код, що буде здійснювати необхідне перетворення типів на етапі переробки програми. У змінній-варіант можна помістити: ціле або дійсне число; логічний вираз, рядок; час або дату; OLE-об'єкт; масив довільної розмірності і довжини. Наступний приклад ілюструє використання варіантів.

```
Var V1, V2, V3,V4, V5: Variant;
  i:Integer; s:String;
  Begin

V1:=1;V2:=1234.5678;v3:='Рядок_символів';V4:='1000';
      V5:=V1+V2+V3    // отримаємо число 2234.5678
       i:=V1;         // i=1
      S:=V3;          // s='Рядок_символів';
       i:=V4;         // i=1000;
       s:=V5;         //s='2235.5678'
End;
```

**Тип дата-час** визначається ідентифікатором `TDateTime` і призначений для зберігання дати і часу. Тип дата-час займає 8 байт і

представляє собою дійсне число з фіксованою дробовою частиною: в цілій частині числа зберігається дата, у дробовій – час. Над даними цього типу визначені такі ж операції, що і над дійсними числами, а у виразах цього типу можуть брати участь константи і змінні цілого і дійсного типів. Існують стандартні процедури і функції по роботі з даними дата-час. Ось деякі з них:

`Date` – повертає поточну дату;

`DateToStr (d : TDateTime)` – перетворює дату в рядок символів;

`DateTimeToStr(d : TDateTime)` – перетворює дату і час у рядок символів;

`Time` – повертає поточний час;

`TimeToStr(T : TDateTime)` – перетворить час у рядок.

### 5. 3 Виняткові ситуації

Теоретично, можна створити додаток, у якому помилки відсутні. На практиці імовірність виникнення помилки при виконанні додатка прямо залежить від його складності. Виняткові ситуації виникають при некоректних діях програми: при діленні на нуль, при спробі відкрити неіснуючий файл, при виході за межі виділеної області динамічної пам'яті і т.д.

Крім проблем із самою програмою, збій може виникнути й в операційній системі. Він може бути викликаний програмними або апаратними причинами.

При розробці додатків у Delphi програміст має можливість використовувати кілька механізмів, що забезпечують обробку виняткових ситуацій. Це і спеціальні оператори мови Object Pascal, і класи, призначені для програмування реакції на помилки.

**Виняткова ситуація** – це позаштатна подія, що може вплинути на подальше виконання програми. Сучасний компілятор видає код, що перехоплює таку подію, запобігає небажаним наслідкам, зберігає необхідні дані про стан програми. З точки зору Object Pascal виняткова ситуація – це об'єкт.

Для обробки виняткових ситуацій призначений клас `Exception`.

Тип `Exception` породжує численні дочірні типи, що відповідають випадкам помилок введення / виведення, розподілу пам'яті і т.п., які часто зустрічаються. `Exception` і його нащадки становлять собою виняток із правила, що пропонує всі об'єктні типи іменувати з букви "T". Імена нащадків типу `Exception` починаються з "E".

Приведемо приклади класів, що обробляють виняткові ситуації:

`EOutOfMemory` – недостатньо місця разом;

`EOutOfResources` – недостача системних ресурсів;

`EInvalidPointer` – неприпустимий покажчик;

`EDivByZero` – помилка цілочисельного ділення на нуль;

`ERangeError` – число або вираз виходить за допустимий діапазон;

EIntError – будь-яка помилка в цілочисельних обчисленнях;

EInvalidOp – неправильна операція;

EZeroDivide – спроба ділення на нуль;

EOverflow – переповнення з плаваючою точкою;

EConvertError – помилка перетворення у функціях StrToInt або StrToFloat;

З переліком стандартних класів виключень можна познайомитися в довідковій літературі або в довідковій системі, вбудованій у Delphi.

Для обробки виключень у Object Pascal передбачений **механізм захищеного блоку:**

```
Try
        <оператори>
    Except
        <оброблювачі виключень>
   Else
        <оператори>
End;
```

Порядок виконання операторів:

- спочатку виконуються оператори секції Try ... Except;
- якщо оператори виконані без виникнення виняткової ситуації, робота захищеного блоку на цьому припиняється і керування одержує оператор, що стоїть за End;
- якщо при виконанні частини Try виникло виключення, керування одержує відповідний оброблювач у секції Except, а якщо такий не знайдений – перший з операторів, що стоять за словом Else. Є можливість використовувати цю конструкцію без частини Else.

Наприклад:

```
Try
      Z := X/Y;
  Except
      Z := 1.1e+38;
  End;
```

У блоці Except можуть розташовуватися оброблювачі виключень:

On <клас виключення> Do <оператор>;

Тут On, Do – ключові слова; <клас виключення> – клас обробки виключення; <оператор> – будь-який оператор Object Pascal, крім оператора передачі керування Goto на мітку поза блоком Except. Ім'я класу служить своєрідним ключем вибору, а власне обробка здійснюється оператором, що стоїть за Do . **Оброблювач виключення** – сукупність операторів On <клас виключення> Do <оператор>, що визначає дії програми в залежності від класу виключення.

Пошук потрібного оброблювача здійснюється з початку списку вниз доти, поки не зустрінеться клас, здатний обробляти виключення даного типу. Якщо підходящого класу не виявлено, керування передається

операторам, що стоять за словом Else, а якщо таких немає, виконується обробка того виключення, що є за замовчуванням.

```
Try
   .......
   Except
   On ERangeError Do . . .;
   On EDivByZero Do . . .;
   On EIntError Do . . .;
   End;
```

Середовище Delphi при кожній винятковій ситуації перехоплює керування програмою. Таке поводження середовища скасовується за допомогою опції Tools/Environment Options, де на сторінці Preference потрібно скасувати перемикач Break on exception.

Якщо для програміста важливий лише сам факт виникнення виключення і не важливий тип зв'язаної з ним помилки, він може опустити в секції Except ... End оброблювачі разом зі словом Else, залишивши в ній лише необхідний код реакції на будь-яку помилку:

```
Try
   .....
   Except
   ShowMessage ( 'Помилка' );
   End;
```

### 5.4 Приведення типів даних

На операції із змінними визначеного типу компілятор звичайно накладає обмеження, дозволяючи виконання тільки тих операцій, що характерні для зазначеного типу даних. Іноді компілятор автоматично здійснює перетворення типів, наприклад при присвоєнні цілого значення дійсній змінній. У Delphi існують операції приведення типів IS і AS.

В операції IS визначається чи належить даний об'єкт зазначеному типові або одному з його нащадків. Вираз

```
AnObject IS TMyClass
```

повертає True, якщо змінна AnObject посилається на зразок об'єктного типу TMyClass або одного з його нащадків.

Сама по собі IS не є операцією визначення типу. У ній лише перевіряється сумісність об'єктних типів. Для коректного приведення типу об'єкта застосовується операція **AS:**

```
With AnObject AS TMyClass Do ...
```

В операції AS спочатку перевіряється сумісність типів за допомогою операції IS. Якщо типи несумісні, запускається оброблювач виняткової ситуації. Таким чином, у конструкції AS операція явного приведення типу виявляється заключеною в безпечну оболонку.

## 6 ЗАГАЛЬНІ ВЛАСТИВОСТІ КОМПОНЕНТІВ

Середовище розробки Delphi забезпечують розроблювача засобами і інструментарієм для створення програм методами об'єктно-орієнтованого програмування. Для цього створена розгалужена ієрархія класів, що

дозволяє створювати додатки на будь-який смак і для будь-якої прикладної області. Усі класи цієї ієрархії об'єднані в спеціальну бібліотеку компонентів.

*Компонент* – функціональний елемент, що має визначені властивості і розміщається програмістом у вікні форми. Компоненти для розробки інтерфейсу і системних компонентів називаються відповідно візуальними і невізуальними.

*Візуальні компоненти* видно як під час розробки, так і під час виконання. Вони можуть відображати дані і сприймати введення (координати миші). Більш складні візуальні компоненти дозволяють уводити текст, вибирати елементи списку і виконувати досить складну обробку подій, зв'язаних з мишею.

*Невізуальні* об'єкти доступні винятково через програмний код, і їх властивості можна редагувати тільки за допомогою програмного коду. Невізуальні компоненти містять у собі складні функції системи і компоненти Delphi, які прискорюють процес розробки. Існує дві категорії невізуальних компонентів – діалогові і системні. Категорія діалогу надає прямий доступ до загальних діалогів Windows таким, як стандартний діалог відкриття файлу або діалог вибору кольору. Категорія системних компонентів забезпечує підтримку функціональних можливостей візуальних компонентів. Компоненти `TMainMenu` і `TPopupMenu` спрощують розробку меню прикладної програми. Компонент `TTimer` служить генератором подій.

Прикладами невізуальних об'єктів також є:

`TClipboard` – призначений для організації буфера обміну;

`TScreen` – керування екраном під час роботи комп'ютера;

`Tprinter` – інкапсуляційні функції Windows з обслуговування принтера;

`TApplication` – керування проектом в цілому і ін.

### 6.1 Базові класи VCL

В основі ієрархії об'єктів бібліотеки VCL лежить клас `TObject`. Він забезпечує виконання найважливіших функцій "життєдіяльності" будь-якого об'єкта. Завдяки йому кожен клас одержує в спадщину механізми створення екземпляра об'єкта і його знищення.

Звичайно розроблювач не задумується про те, як об'єкт буде створений, і що необхідно зробити для його коректного знищення. Компоненти VCL створюються і звільняють займані ресурси автоматично. Іноді розроблювачеві приходиться створювати і видаляти об'єкти самостійно. Але навіть у цьому випадку йому досить викликати відповідний конструктор і деструктор:

```
Var SomeList : TStrings;
    ...
    SomeList := TStrings.Create;
    ...
```

```
SomeList.Free;
```

За удаваною простотою цих операцій ховається досить складна реалізація зазначених процесів. Практично весь вихідний код класу `TObject` написаний на асемблері для забезпечення найбільшої ефективності операцій, що будуть виконуватися в кожнім його нащадку. Крім того, клас `TObject` забезпечує створення і збереження інформації про екземпляр об'єкта й обслуговування черги повідомлень.

Клас `TPersistent` породжується безпосередньо від класу `TObject`. Він забезпечує своїх нащадків можливістю взаємодіяти з іншими об'єктами і процесами на рівні даних. Його методи дозволяють передавати дані в потоки, а також забезпечують взаємодію об'єкта з інспектором об'єктів.

Клас `TComponent` є основою для усіх компонентів Delphi. Цей клас задає базову поведінку всіх компонентів – їх основні властивості і методи. До них відносяться:

- можливість відображення компонента в палітрі компонентів і керування ним у дизайнері форм;
- можливість виступати контейнером для інших компонентів;
- можливість виступати як оболонки навколо компонентів `ActiveX` і інших об'єктів, що реалізують інтерфейси.

Клас `TComponent` є базовим для створення невізуальних компонентів, що можуть розташовуватися в палітрі компонентів і використовуватися в дизайнері форм.

Для створення візуальних компонентів базовим є `TControl`, а для створення компонентів, що мають вікна – клас `TWinControl`. Візуальні компоненти є елементами керування. ***Елементи керування*** – це варіанти стандартних елементів керування Windows. Прикладом елемента керування є "кнопка". Користувач може зробити деяку дію, виконавши клацання на цій кнопці. Компонент "Меню" видимий і доступний для редагування тільки розроблювачеві додатка. Для нього цей компонент є інструментом, який використовується для створення меню. Під час роботи програми користувач бачить тільки результат роботи "Меню", а не сам елемент.

Надалі викладені ці два поняття компонентів і елемент керування є синонімами. Всі елементи керування є компонентами, але не усі компоненти – елементи керування.

Загальні предки обумовлюють загальні властивості компонентів. Розглянемо тільки ті, котрі мають найбільше поширення.

### 6.1.1 Клас TObject

Клас `TObject` реалізує функції, що обов'язково буде виконувати будь-який об'єкт, що може бути створений у середовищі розробки. У першу чергу це створення і знищення екземпляра об'єкта.

Процес створення об'єкта включає виділення області адресного простору, встановлення покажчика на екземпляр об'єкта, завдання початкових значень властивостей і виконання встановлювальних дій,

зв'язаних із призначенням об'єкта. Оголошення конструктора виглядає в такий спосіб:

```
Constructor Create;
```

Для знищення екземпляра об'єкта в TObject призначені методи Destroy і Free:

```
Destructor Destroy; Virtual;
    Procedure Free
```

Destroy забезпечує звільнення всіх займаних екземпляром об'єкта ресурсів. При знищенні об'єктів рекомендується замість деструктора викликати **метод Free**, що просто викликає деструктор, але перед цим перевіряє, щоб покажчик на экземпляр об'єкта був не порожнім (не дорівнював Nil ). Це дозволяє уникнути серйозних помилок.

Якщо об'єкт є власником інших об'єктів (наприклад, форма має всі розміщені на ній компоненти), то його метод Free автоматично викличе ті ж методи для всіх об'єктів. Тому при закритті форми розробник позбавлений від необхідності піклуватися про знищення усіх компонентів.

Кожен об'єкт повинен нести якусь інформацію про себе, яка використовується додатком і середовищем розробки. Тому клас TObject містить ряд методів, які забезпечують подання цієї інформації в нащадках.

Функція *ClassName:* ShortString; повертає ім'я об'єкта, що може бути використане для ідентифікації. Наприклад, один метод-оброблювач клацання на кнопці може працювати з декількома типами компонентів кнопок:

```
Procedure TForm1. Button1Click( Sender : Tobject );
  Begin
 If Sender.ClassName='TButton' Then (Sender AS
          TButton).Enabled := False;
 If Sender.ClassName = 'TSpeedButton' Then
         (Sender AS TSpeedButton).Down := True;
  End;
```

Метод ClassNamesIs(const Name:string):Boolean; дозволяє визначити чи належить даний об'єкт тому типові, ім'я якого передане в параметрі Name. У випадку позитивної відповіді функція повертає True.

### 6.1.2 Клас TPersistent

Клас TPersistent призначений для організації взаємодії своїх нащадків з іншими об'єктами і нащадками.

Метод **Assign**(Source:TPersistent) здійснює копіювання вмісту одного об'єкта в інший. При цьому об'єкт-одержувач залишається самим собою, чого не можна досягти, використовуючи просте присвоєння змінних об'єктного типу:

```
FirstObject := SecondObject;
```

У цьому випадку покажчик на одну область адресного простору, що містить об'єкт, заміщається покажчиком на іншу область адресного

простору, що містить інший об'єкт. Метод Assign дозволяє зберегти самі об'єкти незмінними, заміняючи тільки значення їхніх властивостей.

Клас TPersistent не використовується прямо, від нього породжуються нащадки, що повинні вміти передавати іншим об'єктам значення своїх властивостей.

### 6.1.3 Клас TComponent

Клас TComponent є предком усіх компонентів VCL. Він використовується як основа для створення невізуальних компонентів і реалізує основні механізми, що забезпечують функціонування будь-якого компонента. У ньому з'являються перші властивості, що відображаються в інспекторі об'єктів.

Властивість ***Name*** визначає ім'я компонента. Ім'я компонента будується за тими ж правилами, що й імена будь-яких інших об'єктів програмування – констант, змінних підпрограм і т.ін. Оскільки компоненти поміщаються на форму середовищем Delphi, кожен компонент автоматично одержує створюване середовищем ім'я, що збігається з ім'ям свого класу (без початкової букви T) і доповнене числовим суфіксом: Form1, Button2, Edit4 і т.ін. Згодом програміст може перейменувати компонент, щоб зробити текст програми більш читабельним.

Властивість ***Tag*** типу Longint визначає довільний цілочисельний параметр, що не використовується Delphi і яким програміст може розпоряджатися за своїм розсудом.

Властивість ***Components*** – містить список усіх компонентів, власником яких є даний компонент. Властивість Components зручно використовувати в тому випадку коли необхідно звернутися до компонентів, які має даний компонент, використовуючи їхні порядкові імена, а не самі імена.

Властивість ***Owner*** – вказує на власника компонента.

Властивість ***ComponentCount*** – визначає кількість компонентів, власником яких є даний компонент.

Властивість ***ComponentIndex*** – містить індекс даного компонента в списку Components його власника.

### 6.1.4 Клас TControl

Слідом за класом TComponent в ієрархії базових класів розташовується група з трьох класів, що забезпечують створення різних візуальних компонентів. Існує кілька типів елементів керування, що істотно відрізняються за своїми можливостями і поводженням. Кожному типові відповідає власний клас ієрархії.

Клас TControl є базовим для усіх візуальних компонентів і інкапсулює механізми відображення компонента на екрані.

Поточний стан елемента керування визначається властивістю ControlState, тип якого:

```
Type TControlState=Set Of (csButtonDown, csCticked,
```

```
csPalette, csReadingState, csAlingnmentNeeded,
csFocusing,csCreating, csCustomPaint,
csDestroyingHandle, csDocking);
```

Доступність елемента керування в цілому визначається властивістю ***Enabled.*** Властивість `Enabled` визначає чи повинен елемент керування реагувати на події миші, клавіатури або таймера. Якщо властивість `Enabled` має значення `True`, то елемент реагує на події, інакше ці події ігноруються. Відключений елемент керування при стандартному встановленні кольорів Windows зображується сірим кольором.

Властивість ***Height*** задає вертикальний розмір компонента або форми в пікселах.

Властивість ***Width*** визначає ширину елемента керування або форми в пікселах.

Властивості ***Left*** і ***Top*** визначають вертикальну і горизонтальну координати верхнього лівого кутка елемента керування щодо форми або батьківського елемента. Для форм ці значення вказуються щодо екрана.

Координати будь-якої точки елемента керування можна перерахувати в екранні за допомогою методу

```
ClientToScreen (const Point : TPoint) : TPoint;
```

І навпаки

```
ScreenToClient (const Point : TPoint) : TPoint;
```

Параметри робочої області компонента визначаються властивістю ***ClientHeight:*** Integer, що визначає висоту робочої області в пікселах, і властивістю `ClientWidth : Integer`, що визначає ширину робочої області в пікселах.

Властивість ***`ClientOrigin`*** `: TPoint` містить координати лівого верхнього кутка елемента керування в системі координат робочої області власника.

Тип `TPoint` визначений у такий спосіб:

```
Type TPoint = Record X:Longint; Y : Longint; End;
```

Функція `GetClientOrigin:TPoint;Virtual` повертає координати верхнього кутка робочої області.

Властивість ***Align*** – визначає як розташовуються елементи керування усередині батьківського елемента. Властивість може мати одне зі значень: `alNone` – вирівнювання не використовується; `alTop` – компонент притискається до верхньої границі свого батька; `alBottom`, `alLeft`, `alRight` – до нижньої, лівої і правої границь відповідно. Властивість Align необхідно застосовувати, якщо елемент керування повинен залишатися у визначеному положенні на формі при зміні її розмірів.

Властивість ***`Anchors:`***`TAnchors` забезпечує фіксацію елемента керування по сторонах власника. Тип `TAnchors` визначений:

```
Type TAnchors = Set Of TAnchorsKind;
Type TAnchorsKind=(akTop, akLeft, akRight, akBottom);
```

Якщо по вертикалі або горизонталі встановлені обидва якорі, то при зміні розмірів власника розмір елемента керування змінюється таким

чином, щоб відстані до сторін власника залишилися незмінними.

Властивість **AutoSize**:Boolean забезпечує зміну розмірів компонента відповідно до розмірів його вмісту (тексту, зображення, списку, ієрархічного дерева і т.д.).

Властивість **Color** визначає колір елемента керування. Ця властивість має набір значень, наприклад: clBlack, clGreen, clBlue, clRed і т.д.

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

Властивість **Font** визначає шрифт текстового рядка, його колір (Color), розмір (Size), стиль (Style) і ін.

Властивість **Hint** задає текст, що буде відображатися при обробці події OnHint, що відбувається, якщо курсор знаходиться в області компонента. При затримці курсору миші на компоненті спливає невелике вікно з повідомленням, заданим у цій властивості.

Для керування ярликом використовується властивість **ShowHint** : Boolean. При значенні True ярлик починає працювати, при значенні False ярлик виключається.

Властивість **Visible** визначає чи буде даний компонент відображатися на екрані. Якщо властивість Visible має значення True, то компонент видимий користувачеві, у противному випадку – ні.

Властивість **Caption** – текстовий рядок, зв'язаний з компонентом. Текстовий рядок є заголовком для форми. Для мітки властивість Caption – це той текст, що виводиться в положенні мітки.

Властивість **Text** – це теж текстовий рядок. Наприклад, для компонентів TMemo і TEdit текстова константа зберігається у властивості Text.

Властивість **Parent** містить посилання на батьківський елемент. Властивість Parent на відміну від Owner відноситься до елементів керування. Якщо помістити три опції в рамку, що групує, то власником цих опцій буде форма, де вони знаходяться, а батьківським елементом – рамка, що групує.

Властивості **ParentColor, ParentFont і ParentShowHint** указують, що елемент керування повинен брати значення для властивостей Color (Колір), Font (Шрифт) і ShowHint (показувати підказку) з батьківського елемента керування. Механізм зв'язування візуального компонента з батьківським компонентом дозволяє автоматично задавати для нового елемента керування деякі властивості, що відповідають за його зовнішній вигляд. У результаті всі елементи керування, дочірні для однієї батьківського (форми, панелі) будуть виглядати однаково.

Властивість **PopupMenu** задає назву локального меню, що буде відображатися при натисканні правої кнопки миші. Локальне меню відображається тільки у випадку, коли властивість AutoPopup має значення True.

У класі TControl уперше з'являються методи-оброблювачі подій, що забезпечують передачу в елемент подій миші, техніку drag-and-dock і перетягування(drag-and-drop).

### 6.1.5 Клас TWinControl

Нові механізми, інкапсульовані в класі, забезпечують виконання характерних для віконних елементів функцій: прийом і передачу фокуса, відгук на дії мишею і введення з клавіатури.

Властивість ***TabOrder*** визначає положення компонента в так званій послідовності табулятора. Це послідовність, у якій компоненти стають активними, коли користувач натискає клавішу [Tab].

**Метод SetFocus** установлює фокус для даного компонента. Застосовується для компонентів, що мають фокус уведення.

Щоб довідатися чи має елемент керування фокус, використовується метод **Focused** : Boolean; Dynamic.

Властивість ***Brush*** визначає колір і зразок заливання елемента керування.

Властивість ***Controls*** – це масив покажчиків на всі дочірні компоненти даного елемента керування. За допомогою цього масиву можна звернутися до дочірнього елемента не за іменем, а за порядковим номером. Масив містить покажчики на всі дочірні елементи керування.

Властивість ***ControlCount*** – визначає кількість компонентів керування, власником яких є даний керуючий компонент.

Зовнішній вигляд віконного елемента визначається властивістю ***Ctl3D:***Boolean. При значенні True елемент керування має тривимірний вигляд.

### 6.1.6 Клас TGraphicControl

Клас TGraphicControl призначений для створення на його основі візуальних компонентів, що не одержують фокус у процесі виконання додатка. Так як безпосереднім предком класу є клас TControl, то нащадки TGraphicControl уміють реагувати на керуючі дії мишею.

Наочний приклад елемента керування, якому не потрібно одержувати фокус – це компонент TLabel, призначений для відображення тексту або компонент TImage, призначений для візуалізації зображень.

Для візуалізації елементів керування на основі цього класу використовується канва, інкапсульована в класі TCanvas.

Доступ до канви здійснюється через властивість Canvas.

Властивість **Canvas** надає кодові Delphi можливість маніпуляції областю рисунка під час виконання. Основна особливість властивості Canvas полягає в тому, що вона містить властивості і методи, що спрощують графіку. За допомогою методів властивості Canvas можна легко рисувати лінії, дуги, прямокутники й еліпси, а також визначати пріоритетні і фонові кольори і розміщати текст на поверхні об'єкта.

## 6.2 Програми, керовані подіями

Об'єктно-орієнтовані бібліотеки класів типу `ObjectWindows` фірми Borland надають більш простий спосіб обробки повідомлень. Усі класи мають ряд визначених обробників і вміють реагувати на стандартні події. Одним із призначень засобів візуального програмування, до яких відноситься Delphi, є спрощення створення Windows-додатків. Delphi дає розробнику повний доступ до подій моделі роботи Windows, максимально спрощуючи процес обробки тієї або іншої події. Природно, неможливо передбачити усі випадки і цілком інкапсулювати усі повідомлення Windows, яких більше двохсот. Візуальні компоненти містять обробники найбільш загальних (з точки зору ядра Windows) повідомлень. При необхідності створення обробників інших повідомлень можна скористатися об'єктно-орієнтованою технологією. У цьому випадку створюється об'єкт-спадкоємець того компонента, що повинен обробляти подію, і спеціальний метод-обробник цієї події. Але в більшості випадків можна обійтися тими обробниками, що надають стандартні компоненти, включені до складу Delphi.

### 6.2.1 Події, які обробляються формою

Базовий інтерфейсний елемент форми має двадцять обробників подій. Форма одержує подію *OnActivate* при її активізації. Активізація форми може відбутися при одержанні формою фокуса, наприклад, коли користувач натиснув кнопку миші в робочій області форми.

*Подія OnClose* настає при закритті форми. Форма одержує цю подію перед закриттям форми, що може настати або при виклику методу `Close`, або при виборі команди `Close` із системного меню.

За допомогою *події OnCloseQuery* можна дозволити або скасувати закриття форми. Ця подія може настати або при виклику методу `Close`, або при виборі команди `Close` із системного меню. В обробнику цієї події змінюється значення параметра `CanClose` повідомлення `OnCloseQuery`.

*Подія OnCreate* виникає при початковому створенні форми. В обробнику даної події можна, наприклад, задавати початкові значення властивостям форми і передбачати інші різні дії, що повинні відбуватися в момент створення форми. Форма створюється при запуску додатка або при виклику методу `Create`.

*Подія OnDestroy* виникає на фінальній стадії закриття форми і може бути викликана за допомогою методів `Destroy` або `Free`, або закриттям головної форми додатка.

*Подія OnHide* виникає при "прихованні" форми, тобто коли її властивість `Visible` приймає значення `False`.

*Подія OnShow* виникає, коли форма відображається (тобто коли її властивість `Visible` приймає значення `True`).

*Подія OnPaint* виникає при необхідності перерисовування вмісту форми. Наприклад, воно може виникнути при одержанні формою фокуса.

*Подія OnResize* виникає при зміні розмірів форми під час роботи додатка. Обробник цієї події необхідний тільки в тому випадку, якщо плануються які-небудь дії при зміні розмірів форми.

### 6.2.2 Події від клавіатури і миші

Для більшості видимих елементів визначений набір обробників подій, зв'язаних з мишею і клавіатурою. Споконвічно джерелом цих подій є драйвер клавіатури або драйвер миші, потім ядро Windows перетворює їх у стандартний вигляд і пересилає вікну програми. Перш ніж послати повідомлення про введення з клавіатури, ядро Windows визначає, яке вікно є активним. У Delphi-програмах активними елементами будуть одна з форм (або головна форма) і елемент, визначений властивістю `ActiveControl`.

Події від клавіатури одержують тільки деякі віконні компоненти.

*Подія OnKeyPress* виникає при натисканні клавіші на клавіатурі. Звичайно ця подія обробляється в тому випадку, коли необхідна реакція на натискання однієї клавіші на клавіатурі. Параметр `Key` має тип `Char` і містить ASCII-код натиснутої клавіші. Для клавіш, що не мають ASCII-кодів (відповідних символів) таких, як `Shift` або `F1`, подія `OnKeyPress` не виникає. При використанні комбінацій клавіш, наприклад `Shift+A`, виникає тільки одна подія `OnKeyPress`. Так, при натисканні комбінації `Shift+A` параметр `Key` буде мати значення "A" (залежить від стану перемикача `Caps Lock`). Для обробки натискань клавіш, що не мають ASCII-еквівалентів, і комбінацій клавіш необхідно використовувати події `OnKeyDown` і `OnKeyUp`.

Подія `OnKeyPress` має тип `TKeyPressEvent` і описаний у такий спосіб:

```
TKeyPressEvent=procedure(Sender:TObject;
                         var Key:Char)of Object;
```

Подія `OnKeyDown` відбувається при натисканні клавіші на клавіатурі. Обробник цієї події одержує інформацію про натиснуту клавішу і стан клавіш `Shift`, `Alt` і `Ctrl`, а також про натиснуту кнопку миші. Інформація про клавішу передається параметром `Key`, що має тип `Word`. Для визначення того, яка саме клавіша була натиснута, необхідно використовувати коди віртуальних клавіш.

```
OnKeyDown(Sender:TObject;
          var Key:Word; Shift:TShiftState);
```

Параметр `Key` містить код натиснутої клавіші, а параметр `Shift` може мати одне з таких значень:

`ssShift` – натиснута клавіша `Shift`;
`ssAlt` – натиснута клавіша `Alt`;
`ssCtrl` – натиснута клавіша `Ctrl`;
`ssLeft`, `ssMiddle`, `ssRight` – натиснуті ліва, середня і права кнопки миші;
`ssDouble` – натиснуті права і ліва кнопки миші.

Подія ***OnKeyUp*** є парною подією для `OnKeyDown` і виникає, коли користувач відпускає натиснуту раніше клавішу. Як і для події `OnKeyDown`, можна розпізнати клавіші типу `Shift`, `Alt` і `Ctrl` і кнопки маніпулятора миші.

Для більшості видимих елементів визначений набір обробників подій, зв'язаних з мишею.

Подія ***OnClick*** виникає при натисканні кнопки миші в області компонента. Це подія також відбувається, коли користувач:

– вибравши елемент таблиці (grid), деревоподібного списку (outline), списку або комбінованого списку натисканням однієї з клавіш керування курсором;

– натиснувши клавішу Enter, коли активним елементом форми була кнопка зі значенням за замовчуванням (задана властивістю `Default`);

– натиснувши клавішу `Esc`, коли в активній формі була кнопка `Cancel` (задана властивістю `Cancel` );

– натиснувши клавішу для виклику кнопки. Наприклад, якщо кнопка мала заголовок '`&Bold`', клавішею для виклику кнопки буде клавіша "B".

Подія ***OnDblClick*** виникає при подвійному натисканні кнопки миші в області компонента.

Подія ***OnMouseDown*** відбувається при натисканні кнопки миші, коли курсор знаходиться в області компонента. Обробник цієї події використовується в тих випадках, коли необхідно почати які-небудь дії при натисканні кнопки миші в області компонента. Параметр Button цієї події дозволяє визначити, яка кнопка була натиснута, а параметр Shift – чи були натиснуті клавіші `Shift`, `Ctrl` або `Alt` при натисканні кнопки миші. Подія `OnMouseDown` має тип `TMouseEvent`, описаний у такий спосіб:

```
TMouseEvent=procedure(Sender:TObject;Button:TMouseButton;
      Shift:TShiftState;  X, Y: Integer) of Object;
```

Параметри X і Y містять координати курсору миші в момент натискання клавіші.

Подія ***OnMouseMove*** відбувається при переміщенні маніпулятора миші. Параметр Shift дозволяє визначити, чи були натиснуті клавіші `Shift,Ctrl` або `Alt` при натисканні кнопки миші. Подія `OnMouseMove` має тип *`TMouseMoveEvent`*, описаний у такий спосіб:

```
TMouseMoveEvent = procedure(Sender; TObject;
     Shift: TShiftState; X, Y: Integer) of Object;
```

Подія ***OnMouseUp*** є парною події `OnMouseDown` і виникає в тому випадку, коли користувач відпустив раніше натиснуту кнопку миші. Параметр `Shift` дозволяє визначити чи були при цьому натиснуті клавіші `Shift,Ctrl` або `Alt`. Параметри X і Y містять координати курсору миші в момент натискання клавіші. Параметр `Button` має тип `TMouseButton`.

У подій системи Delphi існує пріоритет. Подія `OnClick` є більш важливою ніж `OnMouseUp` і повинна оброблятися першою. Пріоритет події `OnDblClick` вище ніж пріоритет подій `OnMouseDown` і `OnMouseUp`.

Подія ***OnEnter*** виникає, коли компонент одержує фокус введення.

Подія ***OnExit*** виникає, коли компонент втрачає фокус.

### 6.2.3 Події протоколу Drag&Drop

Операційна система Windows широко використовує спеціальний прийом зв'язування програм з даними, що називається Drag&Drop (перетягни і відпусти). Розробник може передбачити можливість перетягування цілих компонентів, а також обміну вмістом між компонентами. У Delphi цей протокол базується на двох властивостях і трьох подіях.

Властивість ***DragMode*** визначає, як буде виконуватися весь комплекс дій, зв'язаних з Drag&Drop: `dmManual` – вручну; `dmAutomatic` – автоматично (властивостями і методами компонентів).

Властивість ***DragCursor*** визначає вигляд покажчика миші в момент, коли над компонентом "перетягуються дані".

Подія ***OnDragOver*** виникає в момент переміщення покажчика миші з "вантажем" над компонентом. Заголовок процедури:

```
Procedure (Sender,Source:TObject;X,Y:Integer;
           State:TDragState;
         Var Accept: Boolean);
```

де `Sender` – компонент, що збуджує подію; `Source` – компонент-відправник "вантажу"; X, Y – поточні координати покажчика миші в пікселах клієнтської області компонента; `State` – стан покажчика (`dsDragEnter` –тільки що з'явився над компонентом; `dsDragLeave` – тільки що залишив компонент або була відпущена кнопка миші; `dsDragMove` – переміщається над компонентом). У параметрі `Accept` обробник повідомляє чи готовий компонент прийняти дані.

Подія ***OnDragDrop*** виникає у випадку, коли користувач "відпустив" об'єкт, що перетягується. Параметри обробника збігаються за призначенням з одноіменними параметрами `OnDragOver`. В обробнику `OnDragDrop` повинен утримуватися код, що буде виконуватися, коли користувач "відпустив" об'єкт. Параметр `Source` указує на об'єкт, що був "відпущений", а параметр `Sender` – на об'єкт, що прийняв перетягнений об'єкт.

При завершенні перетягування виникає подія ***OnEndDrag***. Заголовок процедури:

```
Procedure (Sender, Target : TObject; X, Y : Integer),
```

де `State` – відправник даних; `Targer` – одержувач даних або `Nil`, якщо ніхто не прийняв посилку; X, Y – координати миші в момент відпускання лівої кнопки.

## 7 ФОРМА

Стандартний проект у Delphi складається з форм. Кожну форму описують два файли – файл форми, що описує властивості форми і розміщених на ній компонентів, і модуль, що містить опис екземпляра класу форми і вихідний код.

Будь-яка програма має, як мінімум, одну зв'язану з нею форму, що називається **головною.** Ця форма з'являється на екрані в момент старту програми. Однак програма може мати скільки завгодно форм, кожна з яких розв'язує якусь локальну задачу і з'являється на екрані з потребою.

Будь-який додаток Windows повинен мати хоча б одне вікно. Навіть якщо додаток при запуску нічого не показує на екрані, для нього все одно автоматично створюється дескриптор головного вікна – покажчик на область пам'яті, відведену для його розміщення.

Форма вміє взаємодіяти з інструментами середовища розробки і розміщеними на ній компонентами. Крім того, форма забезпечує ще цілий ряд зручних і корисних для розробника функцій. Це розмітка робочої області, можливість вирівнювання компонентів і т.д.

### 7.1 Створення і знищення форми

У стандартному додатку задача створення і видалення форм покладається на сам додаток. Якщо розробник залишив усі форми в списку створюваних автоматично (див. розділ 4), то після ініціалізації програми будуть виконані конструктори усіх форм проекту. При цьому першою створюється головна форма. Після закінчення роботи програми усі форми також автоматично знищуються.

При необхідності створити форму самостійно розробник може використовувати її конструктор:

```
If Form1=Nil Then Form1:= TForm1.Create(Application);
```

Або:

```
If not Assigned (Form2) then    // Перевіряємо: віконний об'єкт
                                // створений?
   Form2 := TForm2.Create(Self); // Якщо ні – створюємо його
```

При необхідності знищити форму в додатку, який ще працює, використовується метод `Release`:

```
If Form1 < > Nil Then Release;
```

При створенні і знищенні форми, відповідно, викликаються методи-обробники `OnCreate` і `OnDestroy`.

### 7.2 Візуалізація форми

Кожне наступне вікно стає видно тільки після звертання до його методу `Show` або `ShowModal`.

При виклику методу **Show** друге вікно з'являється на екрані і працює одночасно з першим, тому керування відразу передається операторові, який стоїть після звертання до цього методу. Такі вікна називаються немодальними. Метод **ShowModal** створює модальне вікно, що цілком

бере на себе подальше керування програмою, тому оператор за звертанням до ShowModal у частині програми, що викликається, одержить керування тільки після закриття модального вікна. Модальні вікна завжди чекають від користувача прийняття якого-небудь рішення. З їхньою допомогою реалізується діалог з користувачем або створюється інформаційне вікно, що користувач повинен закрити після ознайомлення з інформацією, що міститься в ньому.

Якщо від користувача потрібне прийняття рішення, у модальне вікно вставляються залежні або незалежні перемикачі, кнопки й інші інтерфейсні елементи, за допомогою яких користувач може повідомити програмі про прийняте рішення. У момент закриття діалогу модальне вікно повинне помістити число, що відповідає рішенню користувача, у свою властивість ***ModalResult***. Деякі стандартні кнопки (Ok, Yes, No, Cancel і т.ін.) автоматично виконують ці дії. Поміщають потрібне число в ModalResult і закривають вікно. В інших випадках про це повинен подбати програміст.

Закриття форми забезпечує метод ***Close***, що спочатку викликає метод ***CloseQuery***, що визначає можливість виконання операції. Якщо він повертає True, форма закривається.

При закритті форми можна використовувати метод-обробник ***OnCloseQuery***. У ньому на основі дій користувача або аналізу стану додатка приймається рішення про закриття форми. Для закриття форми параметр CanClose повинен мати значення True.

Видимістю форми можна керувати за допомогою властивості Visible. Метод **Hide** змінює значення властивості Visible на False.

При відображенні форми можна використовувати метод-обробник ***OnShow***, що викликається при використанні методів Show або ShowModal.

Положення форми на екрані і її розміри визначає властивість ***Position***. За замовчуванням значення цієї властивості дорівнює poDesigned. При цьому форма під час виконання додатка з'являється на тім же місці і має той же розмір, що і під час розробки. У звичайних додатках зручно використовувати значення poScreenCenter, що визначає розміщення форми в центрі екрана без зміни розмірів.

Кожна форма має властивість ***Canvas***, що забезпечує рисування на її поверхні. Розміри клієнтської (робочої) області форми, у якій можна розміщати компоненти, визначаються властивостями ***ClientHeight, ClientWidth, ClientRect***.

При одержанні повідомлення про необхідність перерисовування форма викликає метод-обробник OnPaint. Нарисувати лінію на форму дозволяють такі оператори:

```
With Canvas Do
     Begin
        Pen.Color := clRed;
        Pen.Width := 10;
```

```
    MoveTo(10, 10);
    LineTo(ClientWidth - 10, ClientHeight - 10);
End;
```

Якщо цей фрагмент програми розмістити в методі-обробнику `OnPaint`, то лінія буде відображатися так само надійно, як і будь-який інший компонент.

### 7.3 Атрибути і стилі форми

Форми можуть мати вигляд MDI-додатка і SDI-додатка. Вигляд форми задається у властивості ***FormStyle***. Стиль fsNormal визначає звичайну форму, що використовується для розв'язування всіляких задач, у тому числі – для загального керування всією програмою (головна форма). Стилі `fsMDIChild` і `fsMDIForm` використовуються при створенні так званих багатодокументових додатків у стилі MDI. Стиль `fsStayOnTop` використовується для вікон, що завжди повинні розташовуватися над всіма іншими вікнами програми.

Будь-яке вікно повинно мати визначений набір стандартних елементів керування. Це кнопка системного меню, кнопки зміни розмірів і т.д. Наявність цих елементів на формі визначається властивістю `BorderIcons`.

Існує кілька стилів форм, що використовуються в різних ситуаціях. Стиль форми визначається властивістю ***BorderStyle***. Ця властивість може приймати такі значення:

`bsNone` – на формі цілком відсутні всі системні елементи керування, розміри такої форми не можна змінити;

`bsSingle` – набір системних кнопок залежить від властивості `BorderIcon`;

`bsSizeable` – стандартна форма з можливістю зміни розмірів;

`bsDialog` – стандартне діалогове вікно;

`bsToolWindow` – панель інструментів;

`bsSizeToolWin` – ідентична `bsSingle`, але зі зменшеними системними елементами.

### 7.4 Керування компонентами форми

Якщо форма додатка має фокус уведення, то властивість *Active* має значення True. При цьому активний (даний фокус уведення) елемент керування визначається властивістю ***ActiveControl***.

Якщо форма не має фокуса, то його можна одержати за допомогою методу ***SetFocus***.

За допомогою властивості ***Controls***, успадкованого від класу `TWinControl`, з форми можна одержати доступ до будь-якого елемента керування:

```
Controls[Index : Integer] : TControl;
```

Від класу TWinControl успадковані методи, що дозволяють додавати і видаляти компоненти форми програмно під час виконання додатка:

```
InsertConfrol(AControl: TControl);
RemoveConfrol(AControl: TControl);
```

### 7.5 Шаблони форм

Багато додатків у процесі роботи виконують деякі стандартні операції. Для цього вони мають форми, що містять приблизно однакові набори елементів керування.

Шаблоном називається форма, що поміщена в репозиторій (рис. 19) і використовується як основа для створення нових форм додатка. У результаті отримані в такий спосіб форми будуть мати єдиний стиль оформлення і деякі загальні для усіх функцій.

***Репозиторій*** – це сховище Delphi, у якому зберігаються заготовки (шаблони) проектів і їхніх складових частин. Тут можна знайти стандартний додаток, динамічну бібліотеку, форму, модуль і т.ін. Доступ до репозиторію відкриває опція меню File/New. Репозиторій складається з п'яти сторінок – New, Forms, Dialogs, Data, Modules і Projects.

Сторінка New містить шаблони для створення додатка (Application), об'єкта OLE-автоматизації (Automation Object), нового компонента (Component), модуля даних (Data Module), бібліотеки, що динамічно завантажується, (DLL), форми (Form), текстового файлу (Text), програмного модуля й інші.

Сторінка Forms служить для вибору визначених форм. Серед них – About Box, Database Form, Dual List Box, Quick Report Labels, Quick Report List, Quick Report Master/Detail і Tabbed Pages.

Для підключення нової форми до проекту досить звернутися до репозиторію і вибрати потрібний різновид форми. Менеджер проекту автоматично підключає нову форму до списку використовуваних форм і забезпечує всі необхідні дії для її ініціації. Найперша підключена до проекту форма (стандартне ім'я форми – Forml) стає головним вікном програми. Вікно цієї форми автоматично з'являється на екрані в момент старту програми.

Корисну форму можна включити в репозиторій для постійного використання в будь-якому проекті. Для цього використовується команда Add to Repository меню Project головного вікна Delphi.

### 7. 6 Використання декількох форм у додатку

Головне вікно повинне знати про існування іншого вікна, що досягається посиланням на модуль вікна в пропозиції Uses.

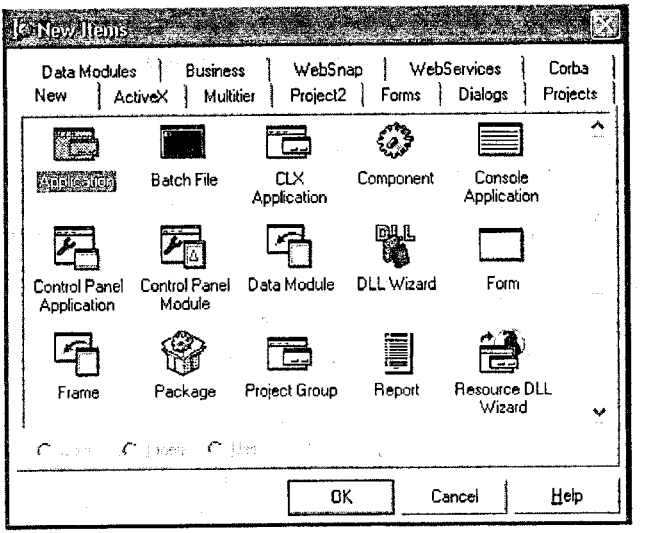



Рисунок 19 – Вікно репозиторія

Якщо, наприклад, у ході виконання одного з методів головного вікна програміст захоче викликати вікно з ім'ям `Form2`, зв'язане з модулем `Unit2`, він повинен послатися на цей модуль у пропозиції Uses головного вікна:

```
Implementation
 Uses Unit2;
```

після чого викликати вікно на екран:

```
Form2. Show;
```

Для використання властивостей і методів компонентів з модуля форми власника можна звертатися до цих компонентів за іменем об'єктної змінної. Для звертання з інших модулів необхідно додавати ім'я форми-власника, наприклад,

```
Form2.Label1.Caption :=...
```

Для переходу до наступної форми в програмі необхідно використовувати послідовність операторів:

```
Form1.Hide; Form2.Show;
```

Якщо Form1 не відображується на екрані, то закриття другої форми не приведе до закінчення роботи додатка. В обробниках подій потрібно передбачити або візуалізацію першої форми, або закінчення роботи додатка.

Робота додатка закінчиться, якщо звернутися до методу `Terminate` класу TApplication:

```
Application.Terminate;
```

# 8 БІБЛІОТЕКА ВІЗУАЛЬНИХ КОМПОНЕНТІВ

Палітра компонентів Delphi містить велике число візуальних компонентів, призначених для конструювання інтерфейсу користувача. Бібліотеки компонентів для різних версій Delphi будуються за принципом розширення: у першій версії було близько 70 компонентів, у третій більше 150, а до складу Delphi 6 входить більше 200 компонентів. Бібліотека VCL у Delphi розбита на тринадцять згрупованих за логічною ознакою сторінок. Вибравши відповідну закладку, одержуємо доступ до компонентів сторінки. Для визначення призначення окремих компонентів широко використовується оперативна довідка.

Стандартні елементи керування розташовуються на сторінках `Standard` і `Additional`. Нові елементи керування в стилі Windows 98 на сторінці Win32. Для сумісності зі старими прикладаннями на сторінці Win 3.1 є відповідні компоненти.

## 8.1 Стандартні візуальні компоненти

На сторінці Standard палітри компонентів Delphi розташовані стандартні компоненти, що відповідають інтерфейсним елементам Windows.

### TMainMenu

Інтерфейсний елемент, названий меню, передбачений стандартними угодами за вимогами до інтерфейс них прикладних програм і використовується практично у всіх Windows-додатках. `TMainMenu` визначає головне меню форми (див. рис. 20). На форму можна помістити скільки завгодно об'єктів цього класу, але відображатися в смузі меню у верхній частині форми буде той з них, що зазначений у властивості `Menu` форми.

Основні можливості, що забезпечують функціонування меню, реалізовані в базовому класі `TMenu`. В основі даного класу лежить властивість, що надає собою колекцію елементів меню. Кожен елемент подає собою окрему команду, що може бути кінцевою (викликати функцію додатка) або розкривати меню більш низького рівня.

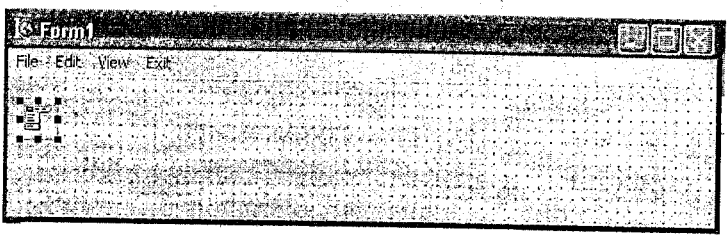



Рисунок 20 – Головне меню програми і невізуальний елемент `TMainMenu`

До окремого елемента меню можна звернутися в такий спосіб:

```
SomeMenu.Items[1].Caption := 'Edit';
```

Кожному елементу меню можна додати власне зображення. Для цього використовується властивість `Images:TImageList`. Картинки повинні утримуватися в компоненті `TImageList`.

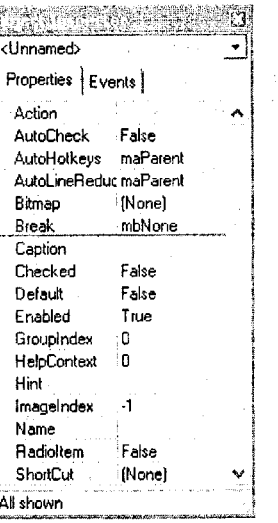



Рисунок 21 – Вигляд інспектора

Для створення опцій меню необхідно клацнути в правій половині рядка `Items` інспектора об'єктів. У рядку `Caption` вводиться текст опції (рис. 21). `Enter` – опція готова і можна переходити до наступної.

Для створення підопцій потрібно клацнути мишею по рядку нижче опції і ввести першу підопцію (рис. 22).

Введення продовжується, поки не буде створений весь список підопцій.

Клацнувши по порожньому прямокутнику праворуч від першої опції, можна ввести другу опцію.

У назвах опцій можна вказати символ "&" перед тим символом, що визначить клавіші швидкого вибору опції.

Для створення розгалужених меню, тобто таких, у яких підопції викликають нові списки підопцій, необхідно клацнути по підопції і натиснути `Ctrl + →`.

Властивість `Break` дозволяє створити багатоярусний список підменю. За замовчуванням має значення `mbNone`. Два інших можливих значення цієї властивості використовуються для створення багатоярусних списків підменю.

`Count` – кількість опцій у підлеглому меню.

`Default` визначає, чи є дана опція підменю тою, що замовчується. Якщо у властивості `Default` зазначене значення `True`, така опція виділяється кольором і вибирається подвійним клацанням миші на батьківській опції.

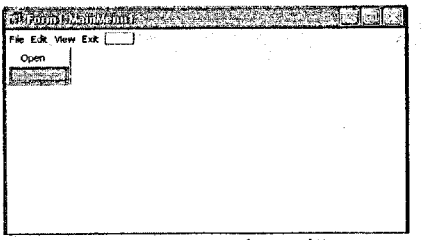



Рисунок 22 – Створення підопцій меню

Рядок `Hint` для опцій меню задає тільки розширене повідомлення, що відображається на панелі статусу.

Якщо Checked = True, поруч з опцією з'являється галочка.

RadioItem – визначає, чи залежить дана опція від вибору інших опцій у тій же групі GroupIndex. Тільки одна опція групи може мати True у властивості Checked. Поруч з такою опцією замість галочки зображується коло.

З активізацією елементів меню зв'язана подія OnClick. Код, розташований в обробникові цієї події, буде виконуватися, коли користувач вибирає елемент меню або за допомогою миші, або за допомогою клавіші. В останньому випадку доступ до рядкового меню здійснюється натисканням клавіші F10 (стандартна клавіша середовища Windows).

Змінювати вміст пунктів меню можна програмним способом. Створимо процедуру, у якій буде додаватися елемент в існуюче меню. Це може відбуватися, наприклад, при натисканні кнопки:

```
Procedure TForm1.Button1Click(Sender : TObject);
  Begin
        NewMenuItem := TMenuItem.Create(File 1);
        NewMenuItem.Caption := 'Новий елемент меню';
        File1.Insert(0, NewMenuItem);
  End;
```

При виклику конструктора Create указується той елемент меню (File 1), що буде власником нового елемента. Далі задається назва елемента. Метод Insert додає новий елемент у зазначеній позиції меню. При необхідності додати елемент у кінець меню, можна використовувати метод Add.

Змінна NewMenuItem повинна мати тип TMenuItem:

```
Var
     NewMenuItem: TMenuItem
```

**TPopupMenu**

Локальне меню – це меню, яке стає доступним, коли користувач натискає праву кнопку миші в робочій області форми або компонента. Може бути створене для будь-якого віконного компонента. Щоб зв'язати клацання правої клавіші миші на компоненті з розкриттям допоміжного меню, у властивість PopupMenu компонента необхідно помістити ім'я меню. Властивість AutoPopup:Boolean забезпечує розгортання меню при клацанні правою кнопкою миші на елементі керування.

Процеси створення і властивості аналогічні TMainMenu.

На рис. 23 реалізоване локальне меню для компонента "мітка". Команди, що містяться в цьому меню, дозволяють указати спосіб вирівнювання тексту мітки – по центру, по лівій або по правій межі.

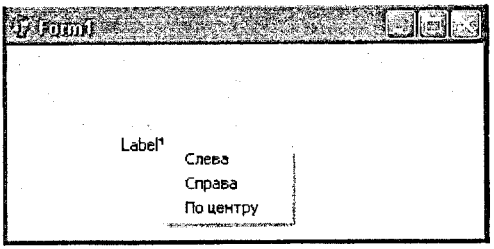



Рисунок 23 – Приклад локального меню для компонента TLabel

Алгоритм дій такий:

Помістимо у форму компонентів типу TLabel. Установимо значення властивості AutoSize в False. Це дає можливість позиціонувати текст мітки. Помістимо на форму компонентів TPopupMenu і в ньому створимо три пункти меню: N1,N2,N3. Назви N1,N2,N3 створюються за замовчуванням у властивості Name редактора меню. Для кожної команди меню необхідно визначити дії при її виборі. Це будуть зміни значення поля Alignment елемента Label1:

```
// Обрана команда "По центру"
Procedure TForm1 . N1Click(Sender : TObject);
    Begin
        Label1.Aligment := taCenter;
    End;
   // Обрана команда "Ліворуч"
   Procedure TForm1.N2Click(Sender : TObject);
   Begin
       Label1. Aligment := taLeftJustify;
    End;
   // Обрана команда "Праворуч"
    Procedure TForm1.N3Click(Sender : TObject);
    Begin
        Label1. Aligment := taRightJustify;
    End;
```

**TLabel**

**A** За допомогою статичного інтерфейсного елемента створюються заголовки для інших інтерфейсних елементів.

Елемент цього типу звичайно використовується, коли необхідно відобразити текст, що не може бути відредагований користувачем, наприклад заголовки компонентів, які не мають власної властивості Caption. Для відображення тексту, який можна змінювати, використовуються компоненти TEdit або TMemo.

Властивість Autosize указує, чи буде мітка змінювати свої розміри в залежності від поміщеного в її властивість Caption тексту: True – буде.

Layout визначає вирівнювання тексту по вертикалі відносно межі мітки (tlTop, tlCenter, tlBottom).

Transparent визначає прозорість фону мітки.

WordWrap – дозволяє/забороняє розрив рядка на межі слова. Для виведення багаторядкових написів: AutoSize=False, WordWrap=True.

### TEdit

Рядок редагування – це прямокутне вікно, у якому можливе введення і редагування тексту. За допомогою компонента TEdit можна відображати і редагувати текст.

Для цього необхідно присвоїти властивості ReadOnly значення True.

*Властивості:*

AutoSelect – указує, чи буде виділятися весь текст у момент одержання компонентом фокуса введення.

AutoSize = True і Border Style = bsSingle, висота компонента автоматично змінюється при зміні властивості Font.Size.

BorderStyle – визначає стиль обрамлення компонента.

MaxLength – визначає максимальну довжину текстового рядка. Якщо 0, довжина рядка не обмежена.

OEMConvert – містить True, якщо необхідно перекодувати текст із кодування MS-DOS у кодування Windows і назад.

PasswordChar – визначає символ, що замінює собою будь-який символ тексту при відображенні у вікні. Використовується для введення паролів.

ReadOnly = True, текст не може змінюватися.

Подія OnChange виникає після будь-якої зміни тексту.

Для компонента TEdit визначені такі методи:

Clear – видаляє весь текст.

ClearSelection – видаляє виділений текст.

CopyToClipboard – копіює виділений текст у Clipboard.

CutToClipboard – копіює виділений текст у Clipboard, після чого видаляє виділений текст із компонента.

PasteFromClipboard – заміняє виділений текст умістом Clipboard, а якщо немає виділеного тексту, копіює вміст Clipboard у позицію текстового курсора.

SelectAll – виділяє весь текст.

Інформація, що вводиться користувачем, може підрозділятися на обов'язкову і додаткову. Обов'язкова інформація повинна вводитися завжди, і поле редагування не повинне залишатися порожнім. У цьому випадку необхідно передбачити захист. Тут можливі варіанти. Наприклад, можна перед використанням вмісту вікна зробити перевірку

```
If Edit1.Text=' ' Then Begin Edit1.SetFocus; Exit End;
```

Якщо вікно редактора Edit1 пусте, то в вікні встановлюється фокус введення і здійснюється вихід із процедури, в якій розміщений даний оператор.

В порядку випадків буває зручно задати тип символів, які можуть бути введені в рядку редагування. Для цього можна використати обробник подій OnKeyPress. Подія OnKeyPress виконується при кожному натисканні клавіші. Можна дозволити введення тільки цифр таким способом:

```
Procedure TForm1.Edit1KeyPress(Sender:TObject;
          Var Key :Char);
   Begin
        If Not (Key in ['0' .. '9'] ) Then Key #27;
   End;
```

Якщо введений символ не входить у задану множину, то клавіша переводиться до стану "нуль".

### TMemo

Компонент TMemo призначений для введення, редагування і/або відображення досить довгого тексту.

Текст зберігається в полі класу TStrings і, таким чином, являє собою набір рядків. До вмісту компонента звертаються, використовуючи властивості Text або Lines. Властивість Text використовується для доступу до усього вмісту компонента, а властивість Lines – для порядкового доступу.

Багато властивостей аналогічні відповідним властивостям класу TEdit. Властивість Wordwrap, як у TLabel.

Специфічні властивості: ScrollBars – смуги прокручування; якщо WantReturns = True, то натискання Enter викликає перехід на новий рядок, у противному випадку – обробляється системою; аналогічно WantTabs для клавіші Tab.

Методи Add, Delete і Insert використовуються для додавання, видалення і вставки рядків. Для роботи з областю обміну використовуються методи CopyToClipboard, CutToClipboard і PasteFromClipboard. Для того щоб виділити весь текст, використовується метод SelectAll, виділений текст доступний через властивість SelText. Для того щоб заповнити Memo вмістом текстового файлу, можна скористатися методом LoadFromFile:

```
If FileExists('c:\autoexec.bat') Then
        Memo1. Lines . LoadFromFile('c:\autoexec.bat');
```

Написавши спеціальну процедуру, можна організувати введення інформації з компонента Memo.

Створимо додаток, що демонструє введення інформації з багаторядкового редактора.

На форму помістимо: компонент TLabel для виведення коментарів; TMemo для введення інформації; TButton для фіксації закінчення введення (рис. 24).

Допоміжна функція:

```
{ Функція повертає підрядок з зазначеним номером }
  Function GetLine( st:string;                { рядок }
                    n:integer)                { номер підрядка }
                     :string;                 { підрядок або ' ' }
  var p:integer;
   begin
     {якщо на початку рядка є проміжки, то видалимо їх }
        while (pos (' ', st)=1) and (lengtn(st)>0) do
         delete(st, 1, 1);
         if n>1 then
              repeat
                p:=pos(#13, st);
                if p <> 0 then
                begin
                    st:=copy(st,p+2,Length(st)-p);
//розділювач-два символи:з кодом  13 і 10; якщо на початку
//частини, що залишилася є проміжки, то стерти їх
                while(pos('',st)=1)and (length(st)>0) do
                  delete(st, 1, 1); n:=n-1;
                  end;
                until (n=1)or(p=0);
  { тут st починається з потрібного підрядка }
   if n>1 then result:=' '
         else
            begin
              p:=pos(#13, st);
               if p<>0 then result:=copy(st,1,p-1)
                  else  result:=st;
            end;
       end;
```

Обробник події:

```
    procedure Tform1. Button1Click(Sender: TObject);
    const
        SIZE=5;   { розмір масиву }
    Var a:array[1..SIZE]of string[30];
        i: integer; st:string;
    begin
   {введення елементів масиву з поля редагування форми }
        For i:=1 to SIZE do
             a[i]:=GetLine(Memo1.Text, i);
    { виведення введеного масиву у вікно повідомлення }
     st:=' ';
     for i: = 1 to SIZE do
          st:=st+IntToStr(i)+'', + a[i] + #13;
     ShowMessage(st);
```

```
end;
```

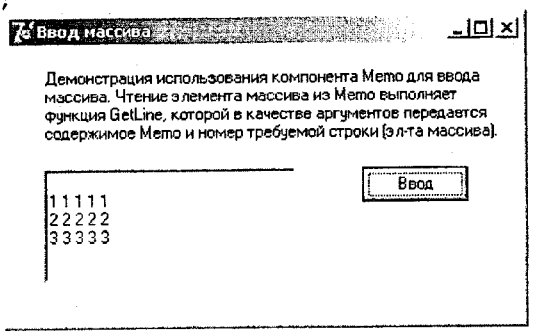



Рисунок 24 – Вікно додатку, яке демонструє введення інформації із Memo.

**TButton**


Компонент TButton є стандартною кнопкою. Кнопка може містити текст, який описує дії, що виконуються при її натисканні. Звичайно кнопки використовуються як керуючі елементи в діалогових панелях.

Кнопкою за замовчуванням вважається кнопка, що посилає подію OnClick при натисканні клавіші Enter. Щоб створити кнопку за замовчуванням, необхідно привласнити властивості кнопки Default значення True. Кнопкою "Cancel" вважається кнопка, що посилає подію OnClick при натисканні клавіші Esc. Для того щоб кнопка стала кнопкою " Cancel ", необхідно присвоїти властивості Cancel значення True.

**Модальними вікнами** називаються такі спеціальні вікна, що, раз з'явившись на екрані, блокують роботу користувача з іншими вікнами аж до свого закриття. Звичайно з їхньою допомогою реалізується діалог, що вимагає від користувача прийняття деякого рішення. Для цього до складу модального вікна включається кілька кнопок. Якщо в кнопки визначена властивість ModalResult, натискання на неї приводить до закриття модального вікна і повертає в програму значення ModalResult як результат діалогу з користувачем. У Delphi визначені такі стандартні значення ModalResult:

mrNone – модальне вікно не закривається;
mrOk – була натиснута кнопка Ok;
mrCancel – була натиснута кнопка Cancel;
mrAbort – була натиснута кнопка Abort;
mrRetry – була натиснута кнопка Retry;
mrIgnore – була натиснута кнопка Ignore ;
mrYes – була натиснута кнопка Yes;
mrNo – була натиснута кнопка No;
mrAll – була натиснута кнопка All;

### TCheckBox

Кнопка з незалежною фіксацією TcheckBox використовується для того, щоб користувач міг вказати своє рішення типу Так / Ні або Так / Ні / Не знаю. Це рішення відбиває у властивості State компонента (містить стан компонента: cbUnchecked–ні; cbChecked – так; cbGrayed– не знаю), доступному як для читання, так і для запису. Перемикачі не залежать один від одного.

Checked – містить вибір користувача типу Так / Ні.

Наступний приклад демонструє застосування цього компонента.

Кнопки з незалежною фіксацією дозволяють змінити колір і стиль шрифту в компоненті Memo (рис. 25).


Рисунок 25 – Використання кнопки з незалежною фіксацією

```
procedure TForm1.CheckBox1Click(Sender:TObject);
 begin
   If CheckBox1.State=cbChecked Then // Якщо кнопка обрана,
   Memo1.Font.Style := [fsItalic] // то стиль – курсив,
    Else Memo1.Font.Style := [ ]; // інакше за замовчуванням.
 end;
   procedure TForm1.CheckBox2Click(Sender: TObject);
   begin
        If CheckBox2.State = cbChecked Then
                    Memo1.Font.Color := clBlue
                    Else Memo1.Font.Color := clBlack;
   end;
```

### TRadioButton

На відміну від TCheckBox компоненти TRadioButton є залежними перемикачами, які призначені для вибору одного з декількох взаємовиключних рішень.

На форму поміщається, щонайменше, два таких компоненти. Якщо в одному компоненті властивість Checked приймає значення True, у всіх інших компонентах, розташованих у тому ж контейнері, ця властивість автоматично приймає значення False.

### TListBox

Інтерфейсний елемент цього типу містить список елементів, які можуть бути обрані за допомогою клавіатури або миші. В

компоненті передбачена можливість програмного прорисування елементів, тому список може містити не тільки рядки, але і довільні зображення.

Визначені події:

OnDrawItem – виникає в момент, коли програма повинна нарисувати черговий елемент. Обробник одержує посилання на список вибору Control, індекс зображуваного елемента Index, границі елемента Rect і його стан State. Прорисування ведеться за допомогою властивості Canvas.

OnMeasureItem – виникає тільки для Style=lbOwnerDrawVariable. Воно передує події OnDrawItem і в ході його обробки програма повинна встановити потрібну висоту чергового елемента.

Список елементів задається властивістю Items. Методи Add, Delete і Insert використовуються для додавання і вставки рядків. Ці методи маніпулюють об'єктом Items (клас TString), що зберігає рядки, що знаходяться в списку. Для того, щоб визначити який елемент списку обраний, використовується значення властивості ItemIndex, а щоб визначити, чи обраний який-небудь елемент – властивість Selected. Сортування елементів списку виробляється, якщо значення властивості Sorted дорівнює True. Вертикальний розмір елементів задається значенням властивості ItemHeight. Для того, щоб всі елементи списку відображалися повністю, властивість ItemHeight повинна бути рівна True. Властивість MultiSelect дозволяє задати можливість одночасного вибору декількох елементів. Значення SelCount дозволяє довідатися, скільки елементів обрано. Число стовпчиків у списку задається значенням властивості Columns.

Canvas – канва для програмного прорисовування елементів.

Style – визначає спосіб прорисовування елементів:
lbStandart – елементи рисує Windows; для інших програмне прорисовування. (Fixed – елементи мають однакову висоту).

При виборі елемента в списку відбувається подія OnClick. Наступний приклад демонструє вибір елемента списку (рис. 26).

```
procedure TForm1.ListBox1 Click (Sender: TObject);
  Var F : String;
   begin
     F := ListBox1.Items[ListBox1.ItemIndex];
     Label1.caption := F;
 end;
```

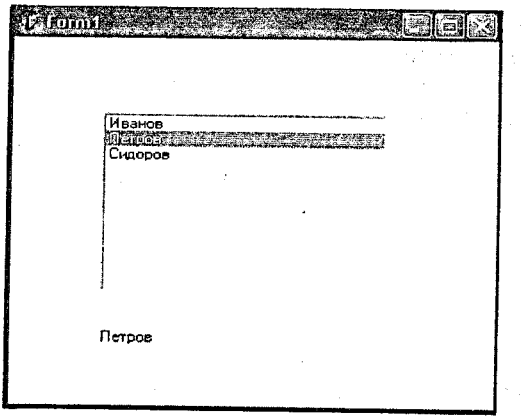



Рисунок 26 – Приклад використання `TListBox`.

**TComboBox**

Комбінований список – це інтерфейсний елемент, що включає у себе список, статичний текст або рядок редагування. Він є комбінацією списку `TListBox` і редактора `TEdit`. Більшість його властивостей і методів запозичені в цих компонентів.

Існує п'ять модифікацій компонента (див. рис. 27), обумовлені його властивістю `Style`:

`csSimple, csDropDown, csDropDownList, csOwnerDrawFixed` і `csOwnerDrawVariable`.

У першому випадку список завжди розкритий, в інших він розкривається після натискання кнопки праворуч від редактора. У модифікації `csDropDownList` редактор працює в режимі відображення вибору і його не можна використовувати для введення нового рядка (в інших модифікаціях це можливо).

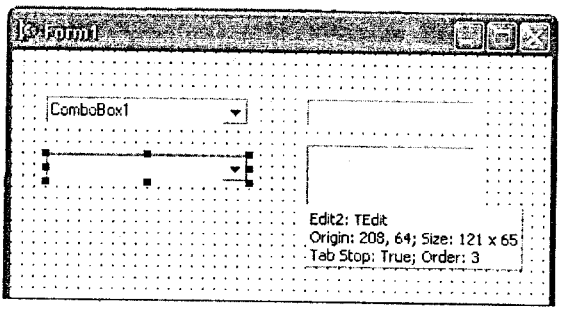



Рисунок 27 – Модифікація `TComboBox`.

csOwnerDrawFixed і csOwnerDrawVariable – для програмного прорисовування елементів.

DropDownCount – кількість елементів списку, поява яких ще не приводить до необхідності прокручування списку.

DroppedDown – визначає, чи розкритий у даний момент список. Ця властивість доступна також для запису, що дозволяє програмно керувати станом списку.

Подія OnDropDown відбувається при зміні стану списку.

### TScrollBar

Керуючий елемент, схожий на смугу прокручування вікна. Використовується для візуального керування значенням числової величини.

Властивості:

Kind – орієнтація компонента.

LargeChange – "велике" зрушення бігунка.

Max – максимальне значення діапазону зміни числової величини.

Min – мінімальне значення діапазону зміни числової величини.

Position – поточне значення числової величини.

SmallChange – "мале" зрушення бігунка.

За допомогою методу SetParams(Aposition,Amax,Amin:integer) можна відразу встановити властивості Position, Max і Min.

Подія OnScroll– виникає при будь-якій зміні властивості Position.

Подія OnChange – при зміні параметрів методом SetParams.

Приклад використання компонента (рис. 28):

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
   Var x: integer;
   Begin
        x :=ScrollBar1.Position;
        Label1.Caption := IntToStr(x);
end;
```

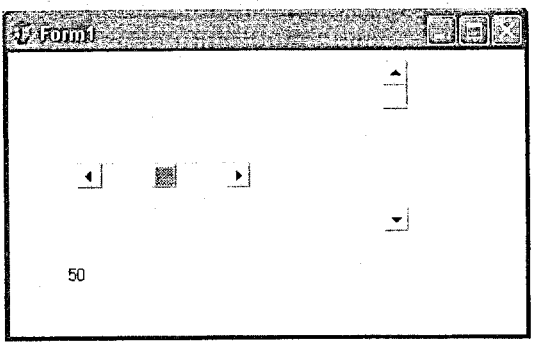



Рисунок 28 – Компонент TScrollBar

**TGroupBox**

Компонент є стандартною групою елементів.

Група елементів – це прямокутник який обрамлює декілька інтерфейсних елементів (зазвичай кнопок з залежною або незалежною фіксацією). Заголовок групи (який відображається в лівому верхньому куті прямокутника) задається за допомогою властивості `Caption`.

**TRadioGroup**

Цей компонент є комбінацією `GroupBox` з набором `RadioButton`. `TRadioGroup` – спеціальний контейнер для розміщення залежних перемикачів класу `TRadioButton`. Кожен розташований у ньому перемикач міститься в спеціальний список `Items` і доступний за індексом, що спрощує обслуговування групи.

У приведеному нижче прикладі (рис. 29) у залежності від обраного перемикача змінюється стан мітки `Label1`, розташованої на панелі `Panel1`. Властивість `Align` у панелі дорівнює `alTop`, у `RadioGroup1` ця властивість дорівнює `alClient`.

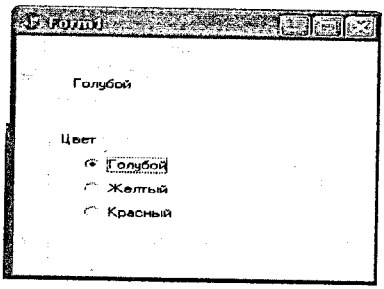



Рисунок 29 – Використання TRadioGroup

```
procedure TForm1.RadioGroup1Click(Sender:TObject);
 begin
  With RadioGroup1 Do
    Begin
    if ItemIndex = 0 then Label1. Caption := 'Голубий';
    if ItemIndex = 1 then Label1. Caption := 'Жовтий';
    if ItemIndex = 2 then Label1. Caption := 'Червоний';
    End;
 end;
```

*Властивості:*

`Columns` – кількість стовпців перемикачів;

`ItemIndex` – індекс обраного перемикача;

`Items` – список рядків із заголовками елементів.

**TPanel**

Панель служить для групування елементів керування і менших контейнерів. За допомогою властивості панелі `Align` можна добитися пропорційного розташування компонентів при зміні розмірів вікна.

Властивості:

`BevelInner` – стиль внутрішньої кромки;
`BevelOuter` – стиль зовнішньої кромки;
`BevelWidth` – ширина кромок у пікселах;
`BorderStyle` – стиль рамки;
`BorderWidtn` – відстань у пікселах від зовнішньої кромки до внутрішньої;
`FullRepaint` – дозволяє/забороняє перерисовування панелі і всіх її дочірніх елементів при зміні її розмірів.

Подія `OnResize` виникає при зміні розмірів компонента.

### 8.2 Додаткові візуальні компоненти

Компоненти, розташовані на сторінці `Additional` (рис. 30) є доповненням до стандартних візуальних інтерфейсних елементів Windows.


Рисунок 30 – Панель `Additional`

**TBitBtn**

Цей компонент є різновидом стандартної кнопки `TButton`. Відмінна риса – властивість `Glyph`, за допомогою якого визначається растрове зображення, що рисується на поверхні кнопки. До складу постачання Delphi входить безліч рисунків, розроблених для розміщення в цих кнопках (каталог `Images/Buttons`).

Частину властивостей кнопка `TBitBtn` успадкувала у свого батьківського класу `TButton`.

Властивості:

`Glyph` – визначає від одного до чотирьох, зв'язаних із кнопкою растрових зображень;
`Kind` – визначає різновид кнопки;
`Layout` – визначає край кнопки, до якої притискається піктограма;
`NumGlyphs` – визначає кількість растрових зображень;
`Style` – визначає стиль оформлення кнопки, в залежності від ОС.

Кнопки `BitBtn` є розвитком концепції, вперше реалізованої у бібліотеці `Borland Windows Custom Controls (BWCC)`. Можливий вибір з восьми визначених типів кнопок (рис. 31), обумовлених значенням властивості `Kind`, і створення власних шляхом вказівки відповідного набору графічних зображень (властивість `Glyph`).

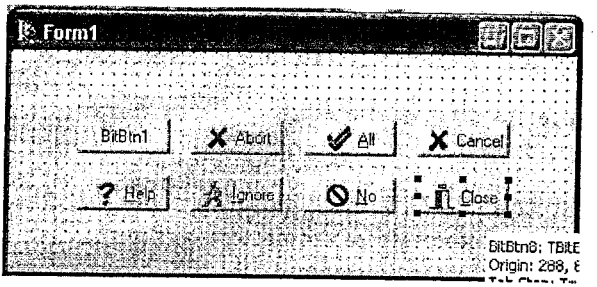



Рисунок 31 – Різновиди кнопки BitBtn

Натискання кожної з поданих на рис. 31 кнопок, крім bkCustom і bkHelp, закриває модальне вікно і повертає в програму результат mrXXX: bkOk – mrOk, bkCancel – mrCancel і т.д. Кнопка bkClose для модального вікна повертає mrCancel, а для головного вікна програми – закриває його і завершує роботу програми. Кнопка bkHelp автоматично викликає розділ довідкової служби, зв'язаної з HelpContext форми, на яку вона поміщена.

**TSpeedButton**

Компонент SpeedButton – спеціалізована кнопка для використання компонентом Panel. Застосовується для створення лінійок інструментів і спеціальних наборів, що включають у себе постійно натиснуті кнопки. Швидкі кнопки дозволяють реалізувати функції, що з іншими кнопками одержати не вдається.

Наприклад, можна створювати кнопки, що залишаються натиснутими, можна групувати кнопки, можна робити кнопки, що змінюють своє гравірування при зміні стану. SpeedButton може знаходитися в станах Up (верхнє), Disabled (відключене), Down (нижнє) і StayDown (притиснуте). Для кожного стану відображається відповідний фрагмент гравірування.

Для фіксації кнопка повинна бути віднесена до якої-небудь групи кнопок (ця група може складатися з неї однієї – варіант одиночної фіксованої кнопки). Для цього використовується властивість GroupIndex: Integer, що не повинна дорівнювати 0. Поведінка кнопки визначається логічною властивістю AllowAllUp: якщо ця властивість має значення True, утоплена кнопка відпускається тільки при натисканні будь-якої іншої кнопки, що входить у ту ж групу; якщо AllowAllUp=False, кнопку можна звільнити повторним клацанням. Індикатором стану кнопки служить логічна властивість Down, що має значення True, якщо кнопка утоплена. Властивість доступна для запису, що дозволяє змінювати стан програмно.

**TMaskEdit**

Компонент призначений для введення тексту, що відповідає деякому шаблону, що задається у властивості `String`. Якщо це властивість не задана, то працює як `Edit`. Властивість `EditText` містить текст до накладення на нього маски шаблону, а властивість `Text` – може містити або вихідний текст, або результат накладення на нього маски шаблону.

Для зручності введення шаблонів у Delphi є спеціальний редактор, що містить кілька наборів найбільш розповсюджених шаблонів. Він запускається при натисканні кнопки в поле властивості `EditMask` в інспекторі об'єктів.

**TStringGrid**

Компонент призначений для подання даних у табличному вигляді. Він організує подання фрагментів тексту в прямокутних комірках по рядках і стовпцях, додає і видаляє рядки і стовпці, організує їхнє прокручування. В верхній і лівій частини таблиці є фіксована область, призначена для розміщення заголовків. Ця область не бере участь у прокручуванні. Кожна окрема комірка таблиці є рядком тексту типу `String`.

Клас `TStringGrid` є нащадком класу `TCustomGrid` і успадковує від нього більшість властивостей і методів.

Властивості компонента:

`BorderStyle` – визначає наявність або відсутність зовнішньої рамки таблиці;

`ColWidths[Index]` – ширина стовпця з індексом `Index`;

`DefaultColWidth` – ширина стовпця за замовчуванням;

`DefaultRowHeight`– містить висоту рядів за замовчуванням;

`EditorMode` – дозволяє/забороняє редагування комірок. Ігнорується, якщо властивість `Options` включає `goAlwaysShowEditor` або не включає `goEditing`.

`FixedColor` – визначає колір фіксованої зони;

`FixedCols` – визначає кількість стовпців фіксованої зони;

`FixedRows` – визначає кількість рядків фіксованої зони;

`GridHeight` – висота таблиці;

`GridLineWidth` – товщина ліній, що накреслюють таблицю;

`GridWidth` – ширина таблиці;

`LeftCol` – лівий стовпець, видимий у зоні прокручування;

`Options` – містить параметри таблиці;

`RowCount` – кількість рядів таблиці;

`RowHeights[Index]` – висота ряду з індексом `Index`;

`Selection` – визначає групу виділених комірок;

`TabStops[Index]`– дозволяє/забороняє вибирати стовпець з індексом `Index` при обході комірок клавішею `Tab`.

Ігнорується, якщо Options не містить goTabs;

TopRow – номер самого верхнього рядка, видимого в зоні комірок, які прокручуються;

VisibleColCount – кількість стовпців, цілком видимих у зоні прокручування;

VisibleRowCount – кількість рядів, цілком видимих у зоні прокручування.

Вміст таблиці задається двовимірним масивом рядків Cells. Вміст окремої комірки доступний при вказівці номера стовпчика і номера рядка, наприклад: Cells[0,0] := 'Ім'я'.

Число стовпчиків задається властивістю ColCount, число рядів – властивістю RowCount. Зміною значення цих властивостей можна додавати або видаляти рядки і стовпчики таблиці. Властивості Col і Row дозволяють визначити поточну комірку.

Objects[ACol, ARow : Integer] забезпечує доступ до списку об'єктів, зв'язаних з комірками (ACol, ARow).

Методи:

CellRect(ACol,ARow)–повертає прямокутник комірки за номерами стовпця Acol і рядка ARow.

MouseToCell(X, Y, ACol, ARow) – повертає табличні координати комірки Acol і ARow по екранних координатах (X, Y) точки.

Події:

OnColumnMoved і OnRowMoved виникають при переміщенні стовпця або рядка.

OnDrawCell виникає при необхідності перерисувати комірки з табличними координатами (Col,Row).

OnGetEditMask і OnGetEditText виникають при редагуванні тексту. У параметрі Value обробник повинен повернути шаблон або текст для редактора TEditMask.

OnSelectCell виникає при виділенні комірки.

OnSetEditText виникає при завершенні редагування.

OnTopLeftChanged виникає після зміни значення TopRow або LeftCol у результаті прокручування робочої зони.

Створимо додаток з використанням компонента TstringGrid (рис.32). Початкова інформація вводиться в табличному вигляді.

При натисканні клавіші "Підсумки" програма розраховує кількість медалей, завойованих спортсменами кожної країни, кількість набраних балів і впорядковує список команд у співвідношенні з кількістю набраних балів.

Обробники подій:

```
procedure TForm1.FormActivate(Sender:TObject);
  // Заповнюється шапка таблиці
  Const Medal:array[0..4] of String [20] =
('Країна','Золотих','Срібних','Бронзових','Усього','Балів');
```

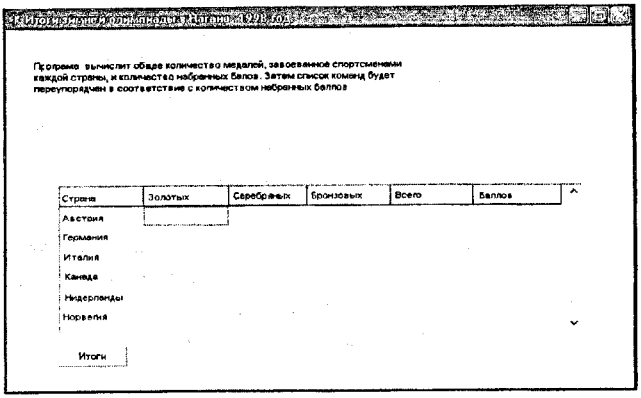



Рисунок 32 – Використання TstringGrid

```
  Country:array[0..11] of String [20] =
    ('Австрія','Німеччина','Італія','Канада','Нідерланди',
     'Норвегія',Росія', 'США','Фінляндія','Швейцарія',
     'Україна');
  Var i :integer;
  begin
      For i:= 0 to 5 Do
      Tabl.Cells[i,0]:=Medal[i];
      For i := 0 To 11 Do
      Tabl.Cells [0,i]:=Country [i];
  end;

procedure TForm1.Button1Click(Sender: TObject:);
  var
      c, r:integer {номер кнопки і рядка таблиці }
      s: integer;  { всього медалей у команди }
      p: integer;  { очків у команди }
      m: integer;  {номер рядка з максимальною кількістю
                    очків}
      buf:array[0..6] of string {буфер для обміну рядків}
      i:integer; {номер рядка,що використовується під
                  час сортування  }
  begin
      for r:=1 to tabl.rowcount do {обробити всі рядки }
          begin
        s: = 0;
    { обчислюємо загальну кількість медалей }
    for c:=1 to 3 do
         if tabl.cells[c,r] <> ' '
             then  s:=s+StrToInt(tabl.cells[c,r])
                 else tabl.cells[c,r]:='0';
    { обчислюємо кількість очків}
```

```
      p:=7*StrToInt(tabl.cells[1,r])+
                 6*StrToInt(tabl.cells[2, r])+
                 5*StrToInt(tabl.cells[3, r ]);
    { виведення результату}
      tabl.cells[4, r]:=IntToStr(s);
      tabl.cells[5, r]:=IntToStr(p);
 end;

      { сортування таблиці }
for r:=1 to tabl.rowcount-1 do
  begin
      m:=r; { максимальний елемент - у  r-ому рядку }
      for i:=r to tabl.rowcount-1 do
   if StrToInt(tabl.cells[5,i])>StrToInt(tabl.cells[5, m])
                              then m:=i;
     if r < > m then
         begin { обміняємо r-ий і m-ий рядок таблиці }
            for c:=0 to 5 do
              begin
                buf[c]:=tabl.Cells[c, r];
                tabl.Cells[c, r]:=tabl.Cells[c, m];
                tabl.Cells[c, m]:=buf[c];
              end;
         end;
   end;
end;
```

**TDrawGrid**

Компонент створює й обслуговує табличні структури. На відміну від `TSringGrid` може містити графічні зображення. Властивості аналогічні властивостям `TSringGrid`.

**TImage**

Служить для розміщення на формі одного з трьох типів зображень, які підтримує Delphi: растрової картинки, піктограми або метафайлу. У властивості `Canvas` компонента міститься канва, за допомогою якої при необхідності програма може відредагувати растрове зображення.

Можливе завдання файлу, зображення якого буде відображатися в формі як під час створення програми, так і під час її виконання. Для завдання файлу під час створення програми використовується властивість `Picture`. При вибиранні цієї властивості з'являється графічний редактор за допомогою якого завантажується графічне зображення з файлу.

Для завдання імені файлу під час виконання програми також необхідно присвоїти властивості `Picture` нове значення. Властивість `Picture` має тип `TPicture`. Допоміжний об'єкт `TPicture` володіє методом `LoadFromFile`, що використовується для завантаження графічного зображення з вказаного файлу.

При необхідності виведення яких-небудь графічних примітивів варто використовувати властивість Canvas графічного зображення, що знаходиться в компонентові Image.

Подія:

OnProgress – дозволяє одержувати зворотний зв'язок під час виконання тривалої операції завантаження зображення. Параметр Stage містить стан процесу завантаження (psStarting – початок, psRunning – йде завантаження, psEnding – процес завершений). Параметр PercentDone приблизно вказує відсоток виконаної роботи. Звичайно в обробнику події за сигналом psStarting створюється індикатор процесу типу TProgressBar, за сигналом psRunning змінюється позиція індикатора, а в момент psEnding індикатор знищується.

**TShape**

Компонент рисує одну з найпростіших геометричних фігур, (рис.33), обумовлених такою множиною: stRectangle, stSquare, stRoundRect, stRoundSquare, stEllipse, stCircle – прямокутник, квадрат, округлений прямокутник, округлений квадрат, еліпс, коло. Фігура цілком займає весь простір компонента. Вигляд геометричної фігури визначається властивістю Shape.

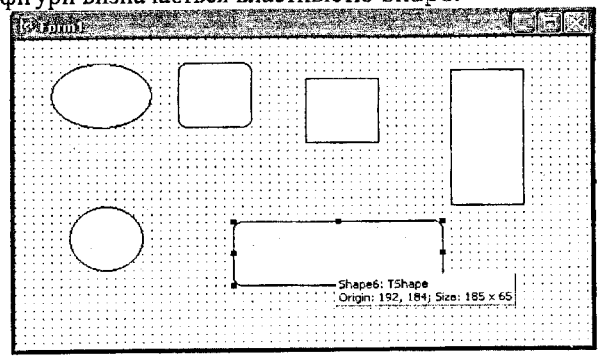



Рисунок 33 – Компонент Shape

Можна керувати властивостями геометричних фігур через властивість Brush – Color і Style. Властивість Color дозволяє задавати колір геометричної фігури, а властивість Style – стиль її заповнення. Наприклад, для того щоб відобразити прямокутник, заповнений жовтим кольором, необхідно виконати такі дії:

```
Procedure TForm1.FormCreate(Sender : TObject);
  Begin
    With Shape1 Do
      Begin
        Shape := StRectangle;          //Прямокутник
        Brush.Color := clYellow; // Колір - жовтий
```

```
        Brush.Style := bsSolid; // Тип заповнення -
                                // повне заповнення
    End;
End;
```

**TBevel**

Призначений для оформлення, наприклад для виділення групи елементів. Властивість `Style` визначає вигляд компонента: прямокутник, рамка, верхня лінія, нижня лінія, ліва лінія, права лінія. `Style` задає стиль компонента.

**TScrollBox**

Служить контейнером для розміщення інших компонентів, її відмінна особливість можливість прокручування і, отже, економія простору форми при необхідності розміщення на ній великої кількості керуючих елементів.

**TCheckListBox**

Групує незалежні перемикачі, які дозволяють звернутися до будь-якого з них за індексом.

Властивість `ChecKed[Index]` містить вибір користувача типу Так/Ні для перемикача з індексом `Index`. `State[Index]` – стан перемикача з індексом `Index`:`cbUnchecked` – ні, `cbChaked` – так; `cbGrayed` – не знаю.

Подія `OnClickCheck` настає при зміні стану будь-якого перемикача.

**TSplitter**

Компонент `TSplitter` використовується для розподілу клієнтської області форми на панелі, розмір якої можна змінювати. Розглянемо приклад. Розташуємо у формі два компоненти `Panel` і компонент `Splitter` між ними (рис. 34). Змінимо значення властивості Cursor компонента `Splitter` на `crVSplit`, а властивості `Align` другої панелі привласнимо значення `alClient`. Властивості `Align` компонентів `Splitter` і першої панелі привласнимо значення `alTop`.

Для компонента визначена подія `OnMoved`, що викликається при будь-якому переміщення компонента мишею.

Якщо курсор миші потрапляє в область компонента `Splitter`, він змінює свою форму. Можна змінювати розмір панелей, пересувати компонент `Splitter`. `Splitter` зв'язаний з одним з компонентів, розміщених в формі (панеллю в нашому прикладі), правилом – обоє мають однакове значення властивості `Align`(`alTop` у нашому прикладі). Тому що друга панель займає всю клієнтську область, не зайняту першою панеллю, при зміні розміру першої панелі змінюється і розмір другої панелі.

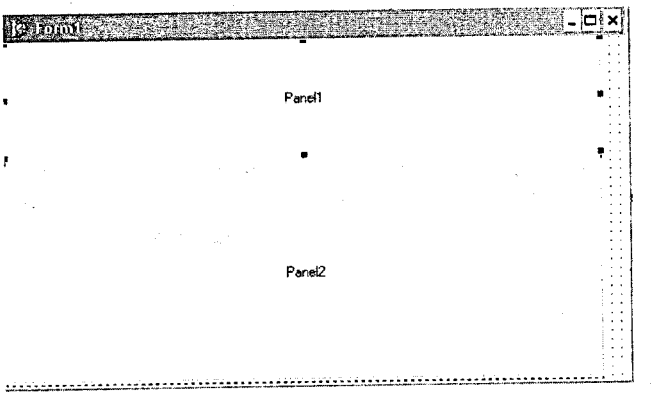



Рисунок 34 – Компонент `Splitter1` зв'язаний з компонентом `Panel1`

### TStaticText

Компонент `StaticText` має ту ж функціональність, що і компонент `Label`. На відміну від `Label` компонент `StaticText` є спадкоємцем класу `TWinControl` і завдяки цьому має посилання на вікно.

### TChart

Цей компонент призначений для графічного подання числових даних. Компонент містить велику кількість специфічних властивостей.

Загальна схема його використання така.

Компонент поміщається на форму. Подвійним натисканням миші на компоненті викликається багатолисткове вікно редактора компонента. Закладка `Series` цього вікна відкриває доступ до так званих серій – об'єктам класу `TChartSeries`, що відображають набори чисел у графічному вигляді. Щоб відобразити дані, потрібно створити як мінімум одну серію – для цього потрібно у вікні редактора натиснути кнопку Add і вибрати підходящий тип графіка.

Після закриття редактора компонент буде містити зразковий вигляд графіка. Його реальний вигляд залежить від фактичних даних, що створюються в працюючій програмі і додаються до серії за допомогою методів `Addx`, `Addy`, `AddXY` об'єкта `TChartSeries`.

Наприклад, такий обробник події `OnActivate` форми створив графік, який показаний на рис. 35.

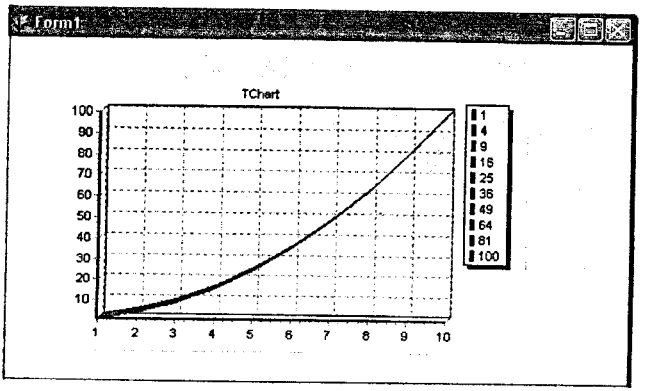



Рисунок 35 – Використання компонента TChar

```
procedure TForm1.FormActivate(Sender:TObject);
Var y:array[1..10] Of Integer; x:integer;
begin
   For x:= 1 To 10 Do
   y[x]= sqr(x);
   For x:= 1 To 10 Do
   Chart1.Series[0].addxy(x, y[x],' ', clRed);
end;
```

### 8.3 Візуальні компоненти інтерфейсу Win32

На сторінці Win32 (рис.36) розташовані компоненти інтерфейсних елементів Windows 98.

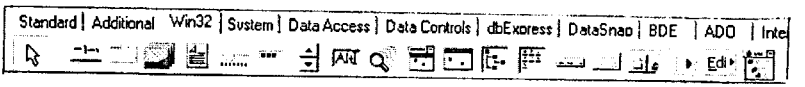



Рисунок 36 – Сторінка Win32

Перелічимо компоненти, розташовані на цій сторінці, а більш докладний опис дамо тільки деяким з них.

**TTabControl**

Набір закладок. Кожна закладка є прямокутним полем з написом і/або текстом. Вибір тієї або іншої закладки розпізнається програмою і використовується для керування вмістом вікна.

Властивість Tabs визначає назви і кількість закладок. Подія OnChange виникає при виборі нової закладки і дозволяє керувати вмістом вікна компонента. Властивість TabIndex визначає індекс обраної закладки.

Як приклад приводиться додаток "Короткий словник іноземних слів"(рис.37). Компонент TabControl займає всю клієнтську частину форми (Align=alClient) і є контейнером для компонента TMemo. У

залежності від обраної закладки в компонент TMemo завантажується той або інший файл, що містить текстову інформацію.

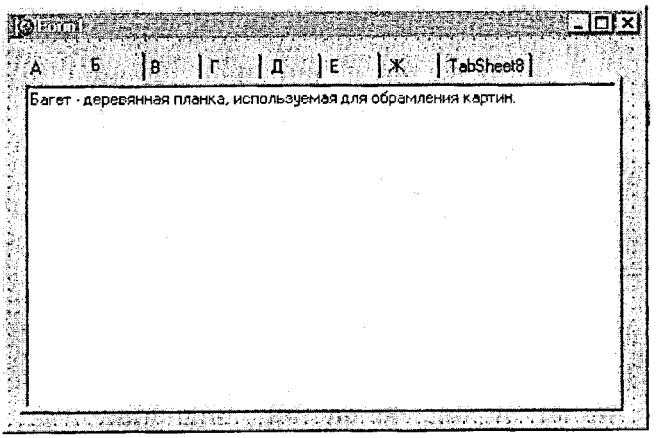



Рисунок 37 – Використання компонента TabControl

Обробник події OnChange.

```
procedure TForm1.TabControl1Change(Sender:TObject);
  Var FileName:String[10];
Begin
  //Формується ім'я файлу відповідно до номеру закладки
  FileName:='FFF'+IntToStr(TabControl1.TabIndex)+'.txt';
  Memo1.Lines.LoadFromFile( FileName);
end;
```

**TPageControl**

Набір панелей із закладками. Кожна панель може містити свій набір інтерфейсних елементів і вибирається натисненням на зв'язаній з нею закладці.

**TImageList**

Контейнер для збереження набору графічних зображень одного розміру. Дозволяє витягати зображення за індексом.

**TRichEdit**

Повнофункціональний текстовий редактор. Цей компонент є оболонкою навколо потужного редактора, що лежить в основі утиліти WordPad. Завдяки використанню цього компонента можна додати в додаток текстовий редактор, що має можливості керування атрибутами всього тексту, виділеного тексту, форматування, зміни шрифтів і т.ін.

**TTrackBar**

Використовується для плавної зміни значень.

**TProgressBar**

Використовується для відображення процесу виконання операції. Властивість `Step` дозволяє задати крок позиціювання індикатора всередині компонента, а властивість `Position` – поточне положення індикатора.

Властивості:

`Max` – максимальне значення діапазону зміни властивості `Position`;
`Min` – мінімальне значення діапазону зміни властивості `Position`;
`Position` – поточне значення величини, що відображується;
`Step` – крок нарощування властивості `Position` методом `StepIt`.

Методи:

`StepBy(Delta)` – нарощує значення властивості `Position` на величину `Delta`;
`StepIt` – нарощує значення властивості `Position` на величину `Step`.

**TUpDown**

Цифровий регулятор. Дві кнопки цього компонента призначені для збільшення або зменшення зв'язаної з компонентом числової величини. Асоціація з будь-яким іншим компонентом задається за допомогою властивості `Associate`.

**HotKey**

Реалізується для завдання клавіші активізації ("гарячої" клавіші). Клавіша задається через властивість `HotKey` того або іншого інтерфейсного елемента.

**TAnimate**

Є стандартним елементом Windows, в якому відбивається вміст AVI-файлу. AVI-файлом може бути невелика анімація, що пояснює дії, що відбуваються. Компонент відтворює відеочастину файлу AVI і ігнорує його звуковий супровід.

Властивості:

`Active` дозволяє/забороняє демонстрацію кліпу.
Якщо `AutoSize = True`, розміри автоматично встановлюються так, щоб цілком розмістити зображення кадру;
`Center` – центрує зображення в межах компонента;

CommonAvi – задає один зі стандартних відеокліпів, що входять у бібліотеку SHELL32.DLL;

FileName – зв'язує компонент із AVI-файлом.

Методи:

Play(Count,FromFrame,ToFrame) демонструє Count раз підряд фрагмент кліпу, починаючи з FromFrame до ToFrame. Stop припиняє показ кліпу.

Події:

OnStart – виникає в момент початку демонстрації;

OnStop – виникає в момент припинення демонстрації.

**TDateTimePicker**

Є списком для введення дати і часу. Цей елемент нагадує список або комбінований список. Список, що випадає, у ньому замінений на календар, з якого користувач може вибрати дату.. Дата і час також можуть вводитись за допомогою стрілок "вверх" і "вниз" або простим набором у рядку введення.

**TTreeView**

Цей компонент є вікном, яке використовується для відображення ієрархічних списків, наприклад, каталогів на диску або рівнів заголовків у документі. Кожен елемент списку складається з опису і набору необов'язкових графічних зображень.

**TListView**

Панель піктограм. Організовує перегляд декількох піктограм і вибір потрібної. Безпосередній спосіб відображення – стовпчиком, вертикально, горизонтально, з іконками та ін. – задається значенням властивості ViewStyle.

**THeaderControl**

Керуючий заголовок. Представляє собою горизонтальну або вертикальну смугу, розділену на ряд суміжних секцій з написами. Розміри секцій можна змінювати мишею на етапі роботи програми. Звичайно використовується для зміні розмірів стовбців чи рядків у різного роду таблицях.

**TStatusBar**

Використовується для створення рядків стану – рядків, у яких указується статус виконання операцій, поточні дата і час, положення маніпулятора миша і т.ін.

**TToolBar**

Інструментальна панель. Застосовується для організації групи керуючих елементів. Здатна автоматично змінювати розміри і

положення командних кнопок.

**TCoolBar**

Інструментальна панель. На відміну від ToolBar, використовується як контейнер для розміщення стандартних інтерфейсних компонентів Windows, таких, як `Edit`, `ListBox`, `ComboBox` і т.д.

## 8.4 Компоненти системних інтерфейсів Windows

На цій панелі представлені компоненти, що мають різне функціональне призначення, у тому числі компоненти, що підтримують стандартні для Windows технології міжпрограмного обміну даними OLE і DDE.

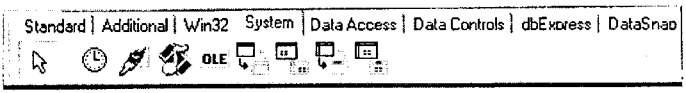



Рисунок 38 – Компоненти панелі `System`

**TTimer**

Таймер служить для відліку інтервалів реального часу.

Властивість `Interval` визначає інтервал часу в мілісекундах, що повинен пройти від включення таймера до настання події `OnTimer`. Таймер включається при установці значення `True` у його властивість `Enabled`. Раз ввімкнений таймер увесь час буде збуджувати події `OnTimer`, поки його властивість `Enabled` не прийме значення `False`.

**TPaintBox**

Вікно для малювання. Створює прямокутну область, призначену для промальовки графічних зображень.

**TmediaPlayer**

Мультимедійний програвач. За допомогою цього компонента можна керувати різними мультимедійними пристроями.

Приклад простого додатка з використанням `MediaPlayer` показано на рис. 39. Помістимо на форму панель `Panel1` і компонент `MediaPlayer1` на панель. Дамо властивості `FileName` значення `Cool.Avi`. Файл `Cool.Avi` повинен розташовуватися в поточному каталозі. Властивості `AutoOpen` присвоїмо значення `True`. Установлюючи значення властивостей `MediaPlayer`, можна змінювати характер взаємодії компонента з файлами-носіями. Наприклад, якщо програвач відтворює відеофайл, він показує його за замовчуванням у окремому вікні; таке поводження компонента можна змінити, установивши його властивість `Display`. Ця властивість говорить програвачеві де потрібно показувати відеофайл.

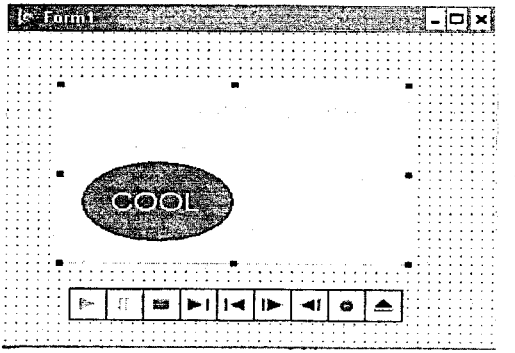



Рисунок 39 – Компонент MediaPlayer

Обробник події:

```
procedure TForm1.FormActivate(Sender:TObjoct);
   begin
      MediaPlayer1.Display:=Panel1;
      MediaPlayer1.DisplayRect:=RECT(40,40,100,100);
   end;
```

Властивість DisplayRect визначає область для показу відеокліпу.

**TOleContainer**

OLE-контейнер служить приймачем об'єктів, що зв'язуються або впроваджуються. Особливістю OLE-об'єкта є те, що подвійне натиснення на цьому компоненті приводить до активізації зв'язаної з об'єктом програми.

Створимо додаток, що містить компоненти OLE-контейнер і кнопку. При натисканні кнопки в контейнер буде завантажуватися графічний файл (рис. 40).

Для поміщеного в контейнер зображення є можливість виклику графічного редактора Paint подвійним натисненням миші.

```
procedure TForm1.Button1Click(Sender: TObject);
   begin
OleContainer1.CreateObjectFromFile('Skyline.bmp',False);
  end;
```

**TDDEClientConv, TDDEClientItem, TDDEServerConv, TDDEServerItem**

Компонент використовується для організації динамічного обміну даними між додатками або для створення сервісної програми в DDE-зв'язку.

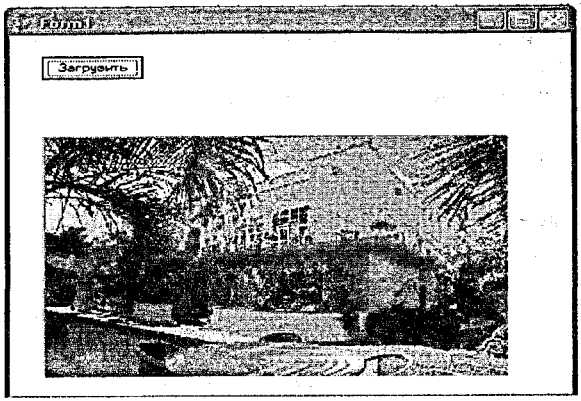



Рисунок 40 – Компонент `ToleContainer` з завантаженим графічним файлом

Створимо додаток з використанням деяких компонентів панелі Win32 і `System` ( рис.41).

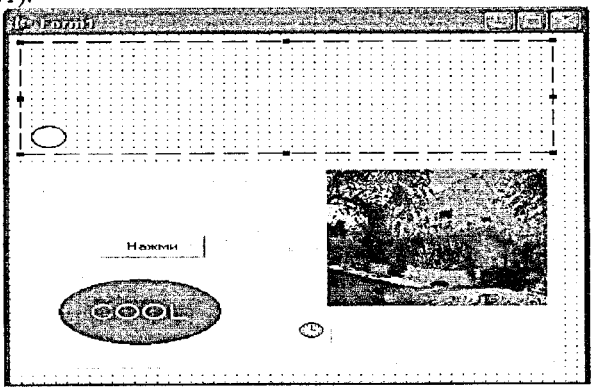



Рисунок 41 – Конструювання вікна додатку, яке демонструє роботу компонентів `TTimer, TProgressBar, TAnimate, TImages, TShape`

Умова задачі.

Тенісний м'яч переміщається в замкнутому об'ємі в горизонтальному і вертикальному напрямках, відбиваючись від верхнього і нижнього рівнів. Досягши правого або лівого краю замкнутого об'єму, м'яч змінює напрямок руху на протилежний. М'яч починає рухатися при натисканні кнопки "Натисни". Одночасно на екрані з'являється відеокліп і компонент `Progressbar`, що відбиває хід виконання процесу. Інтервал часу виконання відеоефектів визначає компонент Timer. Додаток повинен містити графічне зображення.

Порядок виконання.

На форму помістити компонент `Image1`. Встановити властивість

Align=alTop.

На форму помістити компонент Panel1. Установити властивість Align=alClient.

На панель помістити компоненти Progressbar1, Animate1, Timer1, Button1, Image2.

На компонент Image1 помістити компонент Shape1. Установити властивість Shape = stCircle.

У компонент Image2 помістити один із графічних файлів, розташованих у каталозі \delphi6\demos\data\graphex. Для цього використовувати властивість Picture цього компонента.

Установити у властивість Enabled компонента Timer1 значення False, а в його ж властивість Interval – значення 100.

Задати властивість Visible компонента Progressbar1, яке дорівнює False. У поточний каталог помістити файл COOL.AVI.

Обробник події OnClick кнопки Button1:

```
procedure TForm1.Button1Click(Sender: TObject);
  begin
  animate1.FileName:='cool.avi';//Указуємо файл із кліпом
      animate1.active := true;//Запускаємо кліп
      progressbar1.Show;      //Показуємо ProgressBar
      timer1.Enabled := true; //Включаємо таймер
  end;
```

Обробник події OnTimer компонента Timer1:

```
procedure TForm1.Timer1Timer(Sender: TObject);
  begin
       // Показуємо час у компоненті ProgressBar
    Progressbar1.Position: = Progressbar1.Position + 1;
     if progressbar1.Position >= progressbar1.max then
      begin
         Timer1.enabled:=false;  //вимикаємо таймер
           progressbar1.Position:=0//Готуємо новий запуск
                 Progressbar1.Hide;//Ховаємо Progressbar
                 Animate1.stop:  //Зупиняємо кліп
        End;
        With image1 do  // Для компонента Image
         begin
          if x > width -2*shape1.Width  then  xh:=-1;
            if x < 1  then  xh :=1;
              x:=x+xh*2 // Зміна горизонтальної складової
                        //  положення м'яча
                 shape1.left:=x;
            if y> top+height-2*shape1.Width then yh:= -1;
                if y<top   then  yh:=1;
```

```
    y:=y+yh*10;          //Зміна вертикальної складової
                         //положення м'яча
    shape1.top:=y;
  end;
end;
```

Початкові значення координат м'яча, що рухається, встановлюються в події OnCreate форми:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
      x:=0; y:=96; xh:=1; yh:=-1;
end;
```

Додати такий опис: `x,xh,y,yh,i:integer;`

### 8.5 Приклади візуальних компонентів

Компоненти сторінки `Samples` (рис. 42) є прикладами розробки VCL-компонентів, пропонованими фірмою Borland. Це додаткові компоненти різного призначення (рис. 43).


Рисунок 42 – Сторінка Samples палітри компонентів

До їх числа відносяться.

`Tgauge` – індикатор стану. Призначений для відображення ходу виконання якої-небудь операції, наприклад, копіювання групи файлів або читання полів з таблиці даних. Компонент має декілька виглядів. Використовується аналогічно компоненту `ProgressBar`.

`TColorGrid` – таблиця кольорів. Це компонент призначений для вибору основного і фонового кольорів з 16-колірної палітри.

`TSpinButton` – подвійна кнопка. Клацання на кнопці з трикутником, направленим вгору, приводить до виникнення події `OnUpClick`. Клацання на іншій кнопці – до виникнення події `OnDownClick`.

`TSpinEdit` – редактор числа. Він об'єднує в собі три компоненти – дві кнопки із зображеними на них трикутниками, направленими вгору і вниз, і компонент рядок введення.

`TDirectoryOutline` – список каталогів. Призначений для відображення деревовидної структури каталогів. Властивість `Drives` задає ім'я диска, каталог якого повинен відображати даний компонент. Властивість `Directory` містить поточний каталог. При зміні каталогу відбувається подія `OnChange`.

`TCalendar` – є прикладом використовування компоненту `Grid` і є календарем з можливістю вибору дати і місяця.

`IBEventAlerter` – компонент, що реагує на зміни в базах даних, пов'язаних з додатком.

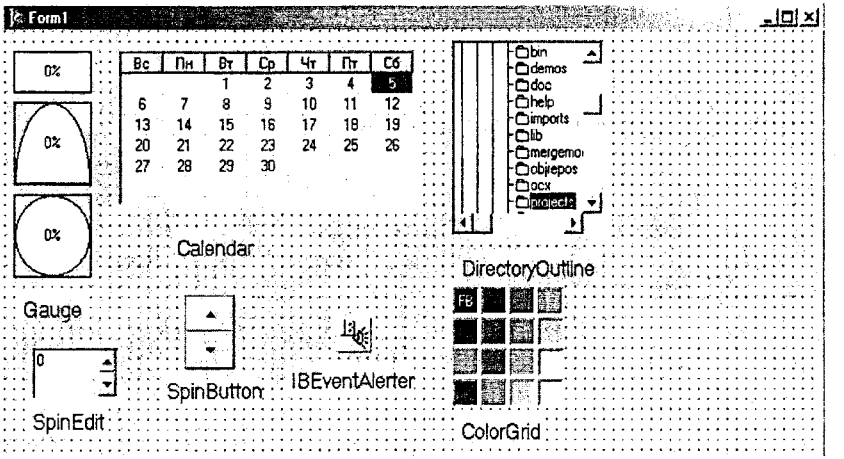



Рисунок 43 – Компоненти панелі `Samples` на формі

### 8.6 Візуальні компоненти Internet

Компоненти цієї сторінки (рис. 44) забезпечують засоби зв'язку програми з глобальною комп'ютерною мережею Internet.

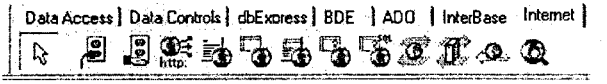



Рисунок 44 – Панель Internet

### 8.7 Компоненти доступу до баз даних

В Delphi розвинуті засоби побудови клієнт-серверних додатків, розрахованих на роботу з базами даних. Ці компоненти зібрані на сторінці `Data Access`. Панель містить невізуальні компоненти.

На панелі `DataControls` розташовані компоненти, що є розширеннями стандартних інтерфейсних елементів Windows, призначеними для сумісного використовування з компонентами для управління базами даних.

На панелі `QReport` розташовані компоненти, що забезпечують простий спосіб створення звітів за наслідками вибірки даних з бази даних.

## 8.8 Візуальні компоненти стандартних діалогів Windows-інтерфейсу

До складу Delphi 6 входить десять компонентів, що реалізовують стандартні діалогові панелі, що використовуються багатьма Windows-додатками (рис. 45).

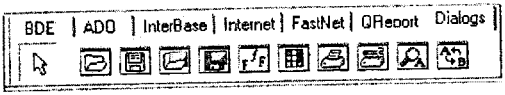



Рисунок 45 – Панель Dialogs

Компоненти панелі:

`TOpenDialog` – діалогова панель вибору файлу, що відкривається, за шаблоном;

`TSaveDialog` – діалогова панель створення файлу;

`TFontDialog` – діалогова панель вибору шрифту і його характеристик;

`TColorDialog` – діалогова панель вибору кольору;

`TPrintDialog` – діалогова панель виведення на пристрій друку;

`TPrinterSetupDialog` – діалогова панель настройки пристрою друку;

`TFindDialog` – діалогова панель пошуку;

`TReplaceDialog` – діалогова панель заміни;

`TOpenPictureDialog` – діалогова панель вибору графічного зображення з переглядом;

`TSavePictureDialog` – діалогова панель збереження графічного зображення з переглядом;

**Основні правила використовування діалогових панелей**

Після того, як компонент поміщений на форму, необхідно зв'язати відображення цієї панелі з якою-небудь подією. Частіше всього такою подією є вибір команди меню або натиснення певної кнопки. В обробнику події необхідно використовувати метод `Execute`. При зверненні до `Execute` на екрані з'являється відповідне діалогове вікно. Вікно діалогу є модальним. Модальне вікно, як відомо, припиняє виконання програми до тих пір, поки користувач не закриє вікно. `Execute` – логічна функція. Проаналізувавши результат `Execute`, програма може присвоїти значення, що повертаються, властивостям тих компонентів, на які вони впливають.

Наприклад:

```
Procedure TForm1.Button1Click(Sender:TObject);
  Begin
      If ColorDialog1.Execute then
      Form1.Color:=ColorDialog1.Color;
  End;
```

В першому рядку цього методу викликається стандартна діалогова панель вибору кольору (рис. 46), а в другому – значення, повернене цією діалоговою панеллю, привласнюється властивості Color форми.

Компоненти TOpenDialog (рис. 47) і TSaveDialog мають ідентичні властивості.

Властивість FileName містить маршрут пошуку і вибраний файл при успішному завершенні діалогу. Програма може використовувати цю властивість для доступу до файла з метою читати з нього дані або записувати в нього.

Властивість Filter використовується для фільтрації файлів, що показуються в діалоговому вікні. Цю властивість можна встановлювати за допомогою спеціального редактора або програмно. При програмному введенні фільтри задаються одним довгим рядком, в якому символи "|" служать для розділення фільтрів один від одного, а також для розділення опису фільтрованих файлів від відповідної маски вибору.

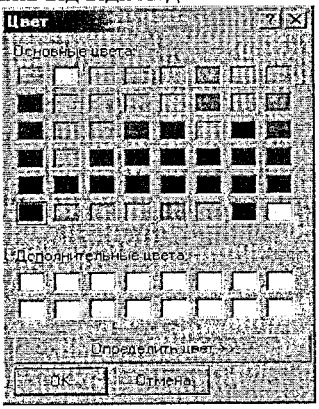



Рисунок 46 – Стандартне вікно вибору кольору

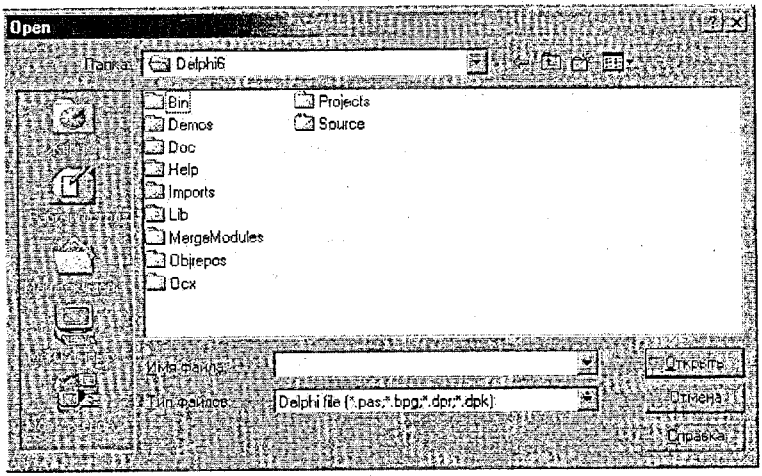



Рисунок 47 – Діалогова панель OpenDialog

Для того щоб відобразити в діалоговій панелі всі файли з розширенням .TXT, необхідно вказати значення цієї властивості як

```
Filter := '*.TXT';
```

або

```
Filter:='Текстові файли|.TXT|DOC-файли|.DOC|Wri-файли|*.WRI;'
```

Встановити початковий каталог дозволяє властивість `InitialDir`.

Створимо просту програму для переглядання вмісту текстового файлу. Для цього на порожню форму поміщаємо компонент `TOpenDialog`, кнопку `TButton` і редактор `TMemo`. При роботі програми клацання на кнопці буде сигналом про необхідність завантажити в редактор новий файл.

Обробник події.

```
Procedure TForm1.Button1Click(Sender :TObject);
  Var S : String; F : TextFile;
  Begin
OpenDialog1.Filter:='Текстові файли|*.TXT|ФайлиПаскаля|*.PAS';
 If OpenDialog1.Execute and FileExists(OpenDialog1.FileName)
       Then
           Begin
                 AssignFile(F, OpenDialog1.FileName);
                 Reset(F);
                 Memo1.Lines.Clear;
                 While not EOF(f) Do
                       Begin
                           ReadLn(F, S);
                           Memo1.Lines.Add(S)
                       End;
                 CloseFile(F)
             End;
  End;
```

Компонент `FontDialog` (рис. 48) має властивість `Device`, яка дозволяє вказати тип пристрою. Властивість може мати такі значення: `fdScreen` – екран, `fdPrinter` – принтер, `fdBoth` – екран і принтер.

В наступному прикладі показано, як відобразити панель вибору шрифтів для принтера.

```
Procedure TForm1.Button1Click(Sender : TObject);
Var FontName : TFont;
Begin
    FontDialog 1.Device := fdPrinter;
    FontDialog 1 .Execute;
    FontName := FontDialog1.Font;
End;
```

## 8.9 Компоненти ActiveX

Компоненти `ActiveX` є чужими для Delphi. Вони створюються іншими інструментальними засобами розробки програм і впроваджуються в Delphi за допомогою технології OLE.

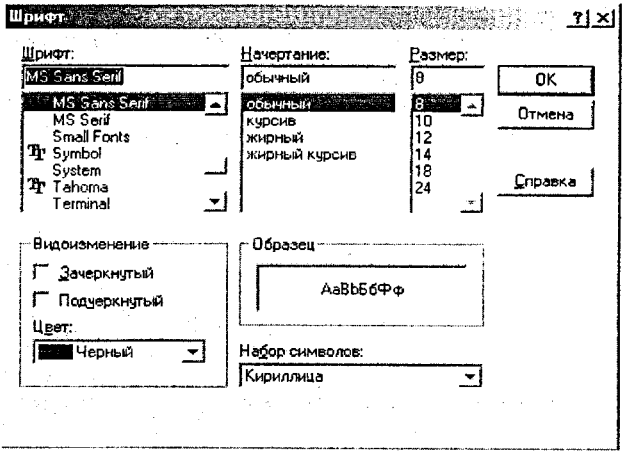



Рисунок 48 – Стандартна діалогова панель вибору шрифту

# 9 СПИСКИ І КОЛЕКЦІЇ

В процесі розробки програмного коду додатка часто буває необхідно зберегти набори різних констант або змінних. Це може бути список днів тижня або група допоміжних об'єктів, створених в процесі роботи додатка для зберігання проміжних результатів розрахунку. Для роботи з рядковими списками призначені класи `TStrings` і `TStringList`. Компоненти VCL `TCombobox`, `TList`, `TDBComboBox` є оболонками для списків рядків. В інших компонентах (`TMemo`, `TStringGrid`) списки забезпечують подання даних. Будь-які типи даних можна заносити в список покажчиків, який реалізований в класі `TList`. Використовування наборів об'єктів, які називаються колекціями, здійснюється за допомогою класів `TCollection` і `TCollectionItem`.

Клас TStrings є базовим абстрактним класом, який забезпечує нащадків основними властивостями і методами, що дозволяють створювати працездатні списки рядків. Його прямим предком є клас `TPersistent`.

Основою механізму списку є властивість, призначена для зберігання елементів списку:

```
Strings[Index : Integer]: String;
```

Звернення до окремого елемента списку може здійснюватися через цю властивість:

```
SomeStrings.Strings[i] := Edit1.Text; або
SomeStrings[i] := Edit1.Text
```

З кожним елементом списку можна зв'язати будь-який об'єкт. Для цього використовується властивість

```
Objects[Index : Integer] : TObject
```

Частіше всього об'єкти потрібні, щоб зберігати для кожного елемента додаткову інформацію. Наприклад, в списку міст для кожного елемента можна додатково зберігати населення, площу, адміністративний статус і т.д. Для цього можна створити такий клас:

```
TSityProps = Class (TObject)
       Square : LongInt;
       Population : LongInt;
       Status : String;
   End;
```

Для додавання до рядка об'єкта із списку використовується метод AddObject, наприклад:

```
SomeStrings.AddObject ('SomeItem', TSityProps.Create);
```

Допоміжні властивості класу містять інформацію про стан списку. Додаткові методи здійснюють пошук в списку і взаємодію з файлами і потоками.

Клас TStrings є абстрактним, і більшість його методів перекриті в нащадку TStringList, а реальний механізм заповнення списку не існує взагалі.

Клас TSringList забезпечує реальне використовування списків рядків в додатку. Властивості і методи класу можна подивитися в довідковій літературі, тут розглянемо тільки деякі з них.

Крім властивості Strings, вміст списку можна отримати за допомогою властивостей Text і CommaText. Вони подають всі рядки списку у вигляді одного довгого рядка. В другій властивості рядки укладені в лапки і розділені комами або пропусками.

Додавання нових рядків здійснюється за допомогою методів Add і Insert. Перший метод додає рядок в кінець списку, а другий вставляє його у вказане місце.

В список можна додати відразу декілька записів. Для цього застосовуються методи AddStrings і Assing. Причому другий метод разом з рядками додає і пов'язані з ним об'єкти.

Для додавання одного об'єкта можна використовувати метод AddObject. Для того щоб додати об'єкт до існуючого рядка, необхідно спочатку створити об'єкт, а потім присвоїти його властивості Object.

Для пошуку необхідного рядка створені методи, що здійснюють пошук за різними параметрами:

IndexOf – пошук за значенням рядка;

IndexOfName – пошук лівої частини рівності;

IndexOfObject – пошук за об'єктом.

Рядки списку можна відсортувати в порядку зростання за допомогою властивості Sorted або методу Sort.

Список можна завантажити з файлу або потоку. Для цього використовуються методи **LoadFromFile** і **LoadFromStream**. Збереження списку виконується методами **SaveToFile** і **SaveToStream**.

Як приклад використовування списків розглянемо таку навчальну задачу. Необхідно розробити додаток, що є фрагментом роботи з базою даних. База даних складається із списку товарів. В самому списку зберігається тільки номер товару по порядку. З кожним рядком зв'язаний об'єкт, в якому міститься більш повна інформація про товар: найменування, вартість, країна-імпортер, об'єм партії. Список можна редагувати, зберігати списки у файлах, завантажувати з файлів.

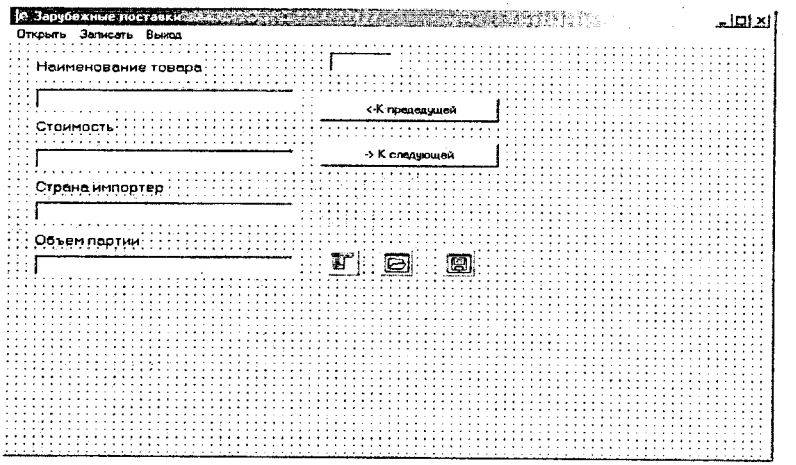



Рисунок 49 – Форма додатку, що демонструє роботу із списками

Проектування вікна додатку.

Для формування головного меню вікна використовуємо компонент TMainMenu, в якому визначимо три пункти із заголовками "Відкрити", "Записати", "Вихід" і назвами N1, N2, N3. Чотири редактори Edit1...Edit4 призначені для введення і виведення інформації. В редакторі Edit5 організовано виведення номера рядка списку. Кнопки BitBtn1 і BitBtn2 здійснюють переміщення за елементами списку. Для організації взаємодії програми з файлами на дисках використовуються діалогові панелі OpenDialog1 і SaveDialog1.

Додаємо опис:

```
Type TTovar = Class     // Оголошуємо новий клас.
```

```
      Name, Cost, Country, Volume:String[20];
      Constructor Create (a:String);
      End;        //Об'єкти цього класу будуть пов'язані
                  //з елементами списку.
  var
    Form1:TForm1; Num, Numk:Integer;
    List1:TStringList; Tovar:TTovar;
```

Конструктор для створення нового об'єкта:

```
Constructor TTovar.Create;
  Begin
     Inherited Create;
     Name := a;
  End;
```

Процедура для додавання елемента списку:

```
Procedure Mem;
 Begin
  List1.add(IntToStr(Num));
  Tovar:=TTovar.Create(' ');//Виділяємо пам'ять під
                            // об'єкт
     List1.Objects[Num]:=Tovar;
  End;
```

Обробники подій:

```
procedure TForm1.FormCreate(Sender: TObject);
  begin
    List1:=TStringList.Create; // Створюємо список
    Num:=0; Mem; Edit5.Text:='0';//Створили 0-й елемент
                                 // списку
  end;
procedure TForm1.BitBtn1Click(Sender:TObject);
  Begin
     Edit5.Text:=IntTostr(Num);
     With List1.Objects[Num] as TTovar do
     Begin   // Записали інформацію в список
          Name:=edit1.text;
          Cost:=edit2.text;
          Country:=edit3.text;
          Volume:=edit4.text;
      End;
      If (Sender AS TBitBtn).Name = 'BitBtn1'Then
      Begin   // Переміщення до початку списку
          If Num=0 Then exit;
          Num:=Num - 1;
      End;
      If (Sender AS TBitBtn).Name='BitBtn2' Then
      Begin    // Переміщення до кінця списку
          Num:=Num + 1;
```

```
            if Num > List1.Count Then Mem;
          End;
    // Зчитуємо інформацію зі списку
      With List1.Objects [Num] as TTovar do
        Begin
           edit1.text := Name;
           edit2.text := Cost;
           edit3.Text := Country;
           edit4.text := Volume;
       End;
      End;
      // Прочитуємо інформацію з файлу
      procedure TForm1.N1Click(Sender: TObject);
      Var F : TextFile;
      Begin
        Try
         With OpenDialog1 Do
         Begin
           If Not Execute Then Exit;
List1.LoadFromFile(FileName);//Завантажуємо список з файлу
  //Готуємо файл для читання інформації в об'єкт Tovar.
AssignFile(F,Copy(FileName,1,Length(FileName)-4)+ '.ooo');
     Reset(F);
     Num:=0
    While Not Eof(F) Do  //Цикл для читання інформації
                         // полів об'єкта Tovar
         With List1 Do
         Begin
              If List1.Objects[Num] = Nil Then
                 Begin
                      Tovar := TTovar.Create(' ');
                      List1.Objects[Num] := Tovar;
                 End;
              Readln(F,(Objects[Num] AS TTovar).Name);
              Readln(F,(Objects[Num] AS TTovar).Cost);
              Readln(F,(Objects[Numj AS TTovar).Country);
              Readln(F,(Objects[Num] AS TTovar).Volume);
              Inc(Num);
         End;
       CloseFile(F);
   End;
   Except
   ShowMessage('Ошибка відкриття файлу');
   End;
   Num:=0;  // Виводимо інформацію в вікно додатку
   With List1.Objects [Num] as TTovar do
   Begin
        edit1.text := Name;
        edit2.text := Cost;
```

```
        edit3.Text := Country;
        edit4.text := Volume;
   End;
   Edit5.Text := IntToStr(Num);
 end;
   // Записуємо інформацію у файл.
   procedure TForm1.N2Click(Sender: TObject);
   Var F: TextFile;   i : Integer;
   begin
   Try
   With SaveDialog1, List1 Do
   Begin
    If Not Execute Then Exit;
    SaveToFile(FileName);  // Записали елементи списку
 AssignFile(F,Copy(FileName,1,Length(FileName)-4)+'.ooo');
    Rewrite(F);
    For i:=0 To Count - 1 Do // Цикл за кількістю записів
    Begin
       If Objects[i] < > Nil Then
             Writeln(F, (Objects[i] AS  TTovar). Name);
             Writeln(F, (Objects[i] AS  TTovar). Cost);
             Writeln(F, (Objects[i] AS  TTovar). Country);
             Writeln(F, (Objects[i] AS  TTovar). Volume);
    End;
   CloseFile(F);
  End;
  Except
  ShowMessage('Помилка відкриття файлу');
 End;
End;
```

Об'єкт `Tovar` забезпечує зберігання для кожного елемента списку додаткових атрибутів. В найпростішому конструкторі об'єкта задається значення для властивості `Name`.

Обробник `BitBtn1Click` здійснює переміщення за рядками списку в двох напрямках, виведення інформації у вікна редакторів `Edit` і записує ті зміни, які можуть проводитися в цих вікнах. `BitBtn1Click` обробляє події, які виникають при натисненні двох кнопок `BitBtn1` і `BitBtn2`. Ідентифікація натиснутої кнопки відбувається за параметром `Sender`.

Збереження і завантаження елементів списку здійснюється використовуванням методів `SaveToFile` і `LoadFromFile`. Для зберігання інформації з об'єктів організований допоміжний файл з розширенням `.ooo`.

Основою класу `TList` є список покажчиків. Сам список є динамічним масивом покажчиків, до якого можна звернутися через індексовану властивість

```
Items[Index : Integer] : Pointer;
```

Властивість **ItemIndex** визначає індекс поточного елемента списку.

Реалізовані в класі `TList` операції із списком забезпечують потреби розробника і збігається з операціями списку рядків.

Для додавання до кінця списку нового покажчика використовується метод

```
Add (Item:Point):Integer;
```

Пряме присвоєння значення елементу, який ще не створений за допомогою методу `Add`, викличе помилку часу виконання.

Новий покажчик можна додати в потрібне місце списку. Для цього використовується метод

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

Можна поміняти місцями два елементи, які визначаються параметрами `Index1` і `Index2`:

```
Exchange (Index1, Index2);
```

Для видалення покажчиків із списку використовуються два методи. Якщо відомий індекс, використовується метод

```
Delete (Index:Integer);
```

Якщо відомий сам покажчик, використовується метод

```
Remove (Item:Pointer):Integer;
```

Повністю всі методи і властивості класу `TList` можна подивитися в довідковій літературі і довідковій системі Delphi.

**Колекція** є різновидом списку покажчиків, оптимізованим для роботи з об'єктами певного вигляду. Сама колекція інкапсульована в класі `TCollection`. Елемент колекції повинен бути екземпляром класу, успадковуваного від класу `TCollectionItem`. Це полегшує програмування і дозволяє звертатися до властивостей і методів об'єктів напряму.

Наприклад, панелі компонента `TCoolBar` з'єднані в колекцію. Клас `TCoolBands`, який об'єднує панелі, є спадкоємцем класу `TCollection`. Окрема панель є екземпляром класу `TCoolBar`, що походить від класу `TCollectionItem`.

## 10 ГРАФІЧНИЙ ІНТЕРФЕЙС

До складу VCL Delphi входять об'єктно-орієнтовані надбудови над об'єктами GUI (Graphic User Interface), призначенням яких є забезпечення зручного доступу до властивостей інструментів і прозора для програміста обробка всіх їх змін.

## 10.1 Клас TCanvas

Клас TCanvas ще називають канвою. Він об'єднує в собі і "полотно", "робочі інструменти" (перо, пензель, шрифт) і "підмайстрів" – набір функцій для рисування типових геометричних фігур.

Канва не є компонентом, але вона входить в якості властивості в ті з них, які повинні вміти нарисувати себе і відобразити яку-небудь інформацію. На канві компонента можна рисувати геометричні фігури, тексти, складати з окремих точок різні узори і відображати растрові зображення.

Властивість Canvas мають багато компонентів VCL і форма у тому числі.

Для рисування канва включає властивості – шрифт, перо і пензель:

```
Font: TFont;
Pen : TPen;
Brush : TBrush.
```

На канві можна рисувати і поточково, отримавши доступ до кожного піксела: властивість Pixels[X, Y:Integer]:TColor.

Початок координат канви розташований в лівому верхньому кутку. Відлік координати X по горизонталі зліва направо. Відлік координати Y по вертикалі зверху вниз.

Властивість PenPos:TPoint містить координати поточної точки пера канви.

Канва містить методи рисування геометричних фігур і зафарбовування їх за допомогою поточного пензля. Ось деякі з них:

Arc(X1,Y1,X2,Y2,X3,Y3,X4,Y4:Integer) креслить дугу еліпса в прямокутнику (X1,Y1) - (X2,Y2), який її охоплює.

Ellipse(X1,Y1,X2,Y2:Integer) креслить еліпс в прямокутнику (X1,Y1) - (X2,Y2), який його охоплює.

LineTo(X,Y:Integer) креслить лінію від поточного положення пера до точки (X,Y).

FrameRect(const Root:TRect) обкреслює межі прямокутника Rect поточним пензлем товщиною в 1 піксел без заповнення внутрішньої частини прямокутника.

MoveTo(X,Y:Integer) переміщає перо в положення (X,Y) без викреслювання лінії.

Polygon(Points:array TPoint) викреслює пером багатокутник за точками, заданими в масиві Points.

Rectangle(X1,Y1,X2,Y2:Integer)-викреслює і заповнює прямокутник (X1,Y1,X2,Y2).

FloodFill(X,Y:Integer;Color:TColor;FillStyle:TFillStyle)-проводить заливку канви поточним пензлем.

TextOut(X,Y:Integer; const Text:String) виводить текстовий рядок Text так, щоб лівий верхній кут прямокутника, що охоплює текст, розташовувався в точці (X,Y).

Draw(X,Y:Integer;Graphic:TGraphic) здійснює прорисовування графічного об'єкта Graphic.

StretchDraw(const Rect:TRect:Graphic:TGraphic) здійснює рисування об'єкта Graphic в заданому прямокутнику Rect. Якщо їх розміри не збігаються, Graphic масштабується.

CopyRect(Dest:TRect; Canvas:TCanvas; Source:TRect)- копіює зображення Source канви Canvas в ділянку Dest поточної канви.

Як приклад, побудуємо зображення, подане на рис. 51.

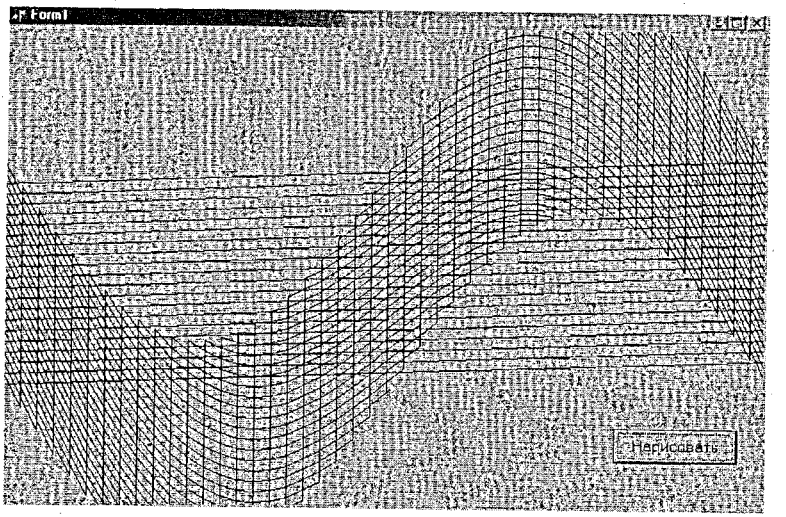



Рисунок 51 – Приклад використовування властивості Canvas

Обробник події:

```
procedure TForm1.Button1Click(Sender:Tobject);
  const pi = 3.14159;
  Var x,y:Real;    px,py,offset,halfheight:LongInt;
  begin
  // Визначаються координати половини сторони внизу форми
  halfheight:=Form1.Height div 2;
  offset:=0;
  For offset:=-10 To 10 Do
  Begin
      px:=0;
      While px < Form1.Width Do
       Begin
            //Масштаб x приводиться до 2 ПІ
            //щоб зробити одну повну синусоїду
             x:=px * (2 * pi/Form1.Width);
             y:=Sin(x);
```

```
      py:=Trunc(0.7*y*halfheight)+halfheignt+(offset*10);
            If (px=0) Then Canvas.LineTo (px,py);
            Canvas.LineTo (px,py);
  py:=Trunc(0.7*y*halfheight)+halfheight+((offset-1)*10);
            Canvas.LineTo (px,py);
            px:=px +15;
        End;
    End;
  end;
```

### 10.2 Клас TFont

За допомогою цього класу створюється об'єкт-шрифт для будь-якого графічного пристрою.

Властивість `Color:TColor` задає колір шрифту.

Властивість `Size:Integer` – висота шрифту в пунктах (1/72 дюйма).

Властивість `Style:TFontStyles` – стиль шрифту.

`fsBold` – жирний,

`fsItalic` – курсив,

`fsUnderline` – підкреслений,

`fsStrikeOut` – перекреслений.

Встановлення властивостей класу вручну здійснюється в основному на етапі проектування. Щоб змінити шрифт для якогось компонента під час виконання, використовують компонент `TFontDialog`.

Приклад використовування властивостей `TFont`:

```
Var F : TFont;
  Begin
      F := TFont.Create;
      With F Do
      Begin
           Name := 'Arial';    Height := 40;
           Color := clBlue;    Size := 40;
           Style := [fsBold];
      End;
      Canvas.Font := F;
      Canvas.TextOut(10, 10, 'Шрифт');
      F.Free;
  End;
```

### 10.3 Клас TPen

Цей клас інкапсулює властивість пера. В конструкторі за замовчуванням створюється безперервне чорне перо вширшки в один піксел. Клас `TPen` містить властивості:

`Color:TColor` – колір викреслюваних пером ліній;

`Style:TPenStyle` визначає стиль ліній (`psSolid` – суцільна, `psDash` – пунктирна і т.ін.);

`Width:Integer` – товщина ліній в пікселах;

Mode:TPenMode – визначає спосіб взаємодії ліній з фоном (переліковний тип TPenMode див. в довідковій системі Delphi).

### 10.4 Клас TBrush

Об'єкти цього класу служать для заповнення внутрішнього простору замкнутих фігур.

Властивості:

**Bitmap**:TBitmap містить растрове зображення, яке буде використовуватися пензлем для заповнення;

Color:TColor – колір пензля;

Style:TBrushStyle – стиль пензля (bsSolid – суцільна;
bsBDiagonal –заштрихованна по діагоналі;
bsCross – заштрихована в клітинку;
bsVertical –заштрихованна вертикальними лініями і т. ін).

Приклад побудови заштрихованого об'єкта на компоненті TPaintBox:

```
procedure TForm1.Button1Click(Sender:TObject);
   Var w,h:Integer;
   begin
      With PaintBox1 Do
      Begin
            W:=Width Div 6;    h:=Height Div 6;
             Canvas.Brush.Style:=bsCross;
             Canvas.Pen.Color:=clBlue;
             Canvas.Ellipse(w,h,5*w,5*h);
       End;
   end;
```

### 10.5 Класи TGraphic і TPicture

Абстрактний клас TGraphic є батьківським для трьох виглядів зображень, загальноприйнятих в графіці Windows: позначки (клас TIcon), метафайлу (клас TMetafile) і растрової картинки (клас TBitmap). Четвертим нащадком TGraphic є клас TJPEGImage – інкапсулюючий стислу растрову картинку у форматі JPEG.

Загальною особливістю нащадків TGraphic є те, що звичайно вони зберігаються у файлах певного формату. Піктограми є невеликими растровими зображеннями, які забезпечені спеціальними засобами, які регулюють їх прозорість. Метафайл – це зображення, побудоване на графічному пристрої за допомогою спеціальних команд. Растрові зображення – це довільні графічні зображення у файлах із стандартним розширенням BMP.

Змінній типу TGraphic можна присвоювати покажчик на будь-який з перерахованих класів-нащадків.

Класи TIcon і TMetafile вміють тільки відображати себе і не призначені для рисування. Клас TBitmap має свою канву для рисування.

Метод Assign(Source:TPersistetm) перевизначає одноіменний метод предка, який допускає поліморфне присвоєння графічних об'єктів.

Для класу TGraphic визначені також методи LoadFromFile, SaveToFile, LoadFromClipboadFormat, SaveToClipboardFormat.

Подія OnProgress дозволяє відображати час завантаження графічних об'єктів, який може бути значним.

Висота і ширина графічного об'єкта задаються властивостями Height і Width. Властивість Modified:Boolean показує чи модифікувався даний графічний об'єкт.

Клас TPicture інкапсулює в собі все необхідне для роботи з готовими графічними зображеннями. Призначення класу – керувати викликами відповідних методів і не звертати увагу на визначення типу графічного об'єкта.

Доступ до графічного об'єкта здійснюється за допомогою властивості Graphic:TGraphic, наприклад Image1.Picture.Graphic:=MyBitmap. Якщо графічний об'єкт має один з трьох перевизначених типів, то до нього можна звернутися і як до одного з властивостей: Bitmap:TBitmap, Icon:TIcon, Metafile:TMetafile.

Якщо в полі Graphic зберігався об'єкт одного класу, а забажали об'єкт іншого класу, то колишній об'єкт знищується, а замість нього створюється пустий об'єкт необхідного класу. Наприклад

```
Image1.Picture.LoadFromFle('myicon.ico');
MyBitmap:=Image1.Picture.Bitmap;
```

Приклад збереження графічного об'єкта в буфері інформаційного обміну Clipboard:

```
Procedure TForm.Buttin1Click(Sender:TObject);
Begin
     Clipboard.Assign(Image1.Picture);
End;
```

Приклад завантаження графічного об'єкта з буфера обміну:

```
Procedure TForm.Buttin1Click(Sender:TObject);
Begin
     Image1.Picture.Assign(Clipboard);
End;
```

Передати властивість об'єкта TPicture Graphic як параметр методам канви Draw( ) або StrechDraw ( ) для виведення картинки на канву можна таким чином:

```
Procedure TForm.Buttin1Click(Sender:TObject);
  Begin
    Form1.Canvas.Draw(300,100,Image1.Picture.Graphic);
  End;
```

Методи класу `TPicture` аналогічні методам `TGraphic`. В цих класах не існує оголошених методів рисування. Пояснення просте – в процесі рисування вони грають пасивну роль; рисують не вони – рисують їх. Все рисування повинно здійснюватися через виклики методів `Draw` і `StretchDraw` канви.

## 11 ФУНКЦІЇ ДРУКУ

Для обслуговування принтера створений спеціальний клас `TPrinter`, реалізований в модулі `Printers`. Для того щоб програма могла взаємодіяти з принтером, необхідно додати модуль `Printers` в секцію `Uses` додатка.

Клас `TPrinters` інкапсулює функції Windows з обслуговування принтера.

Властивості.

Властивість `Canvas` є дисплейним контекстом принтера, в якому відбувається виведення інформації.

Властивість `Orientation` дозволяє задати тип розташування інформації на листі паперу – горизонтально або вертикально (`poLandscape, poPortrait`).

Властивості `PageHeight` і `PageWidht` містять розміри сторінки принтера в пікселах.

Властивість `PageNumber` містить номер поточної сторінки принтера.

Властивість `Printers` містить список всіх описаних в системі принтерів, доступних Windows.

Властивість `PrinterIndex` вказує, який з принтерів в списку, визначений властивістю `Printers`, є поточним вибраним принтером.

Властивість `Printing` має значення `True` до тих пір, поки відбувається процес друку.

Властивість `Title` дозволяє задати заголовок при друці.

Методи.

**Abort** аварійно завершує виведення на пристрій друку.

**BeginDoc** посилає інформацію на принтер.

**End Doc** указує на необхідність початку друку.

**SetPrinter** дозволяє задати поточний пристрій друку.

Для виведення інформації на пристрій друку спочатку викликається метод **BeginDoc**. Потім за допомогою методу TextOut контексту (Canvas) принтера виводиться інформація, її безпосередній роздрук починається після виклику методу **EndDoc**.

```
Begin
      Printer.BeginDoc;
      Printer.Canvas.TextOut(x,y,' ');
      Label1.Caption:='Друкуємо сторінку ' +
                      IntToStr(Printer.PageNumber);
      Printer.EndDoc;
 End;
```

Для виведення тексту в Delphi використовуються оператори `Write` і `Writeln`, як і в Турбо Паскалі.

Приклад виведення вмісту редактора `Memo1`:

```
Var  Pr : TextFiie;
     i : Integer;
   Begin
        AssignPrn(Pr);
        Rewrite(Pr);
        For i := 0 To Memo1.Lines.Count - 1 Do
              Writeln(Pr, Memo1.Lines[i]);
        CloseFile(Pr);
   End;
```

Процедура `AssignPrn` з модуля `Printers` пов'язує текстову файлову змінну з поточним принтером системи і створює в пам'яті буфер виведення. Оператор `Rewrite` відкриває пристрій виведення. Процедура `Writeln` здійснює друк рядка і переводить позицію друку на новий рядок.

Для друку растрового зображення необхідно відрекомендувати його в додатку у вигляді екземпляра класу `TGraphic` або його спадкоємця. Простіше всього використовувати в додатку компонент `TImage` або створений самостійно об'єкт типу `FPicture`. Потім зображення передається на канву принтера за допомогою стандартних методів, і клас `TPrinter` забезпечує його друк.

```
Begin
        With Printer Do
        Begin
            BeginDoc;
            Canvas.Draw(0,0,Image1);
            EndDoc;
        End;
 End;
```

## 12 ІНШІ МОЖЛИВОСТІ DELPHI

Delphi – складне середовище програмування, яке не можна вивчити за один вечір, оскільки воно містить безліч різних елементів.

Delphi є середовищем розробки, що використовується перш за все для підтримки і розробки додатків, призначених як для окремих робочих

станцій, так і для серверів. Delphi може функціонувати під керуванням операційної системи Windows 98, Windows 2000 або Windows NT.

Більшість створених за допомогою Delphi додатків направлені головним чином на розв'язування задач, пов'язаних з виробництвом і бізнесом. Забезпечення функціонування баз даних і створення звітів – найбільш часто розв'язувані задачі.

При роботі в середовищі Delphi за допомогою BDE (Borland Database Engine) можна діставати прямий доступ до dBASE-, Paradox-, FoxPro-, Access- і ASCII- таблиць баз даних. Набір драйверів Borland SQL Links for Windows забезпечує всі необхідні з'єднання з SQL-серверами.

З Delphi поставляється СУБД InterBase, яка звичайно використовується для тестування додатків, що розробляються для роботи з видаленими SQL-серверами Oracle, Sybase, Informix, InterBase NT і DB2. Для того, щоб отримати доступ до інших баз даних або інших форматів даних за допомогою BDE, потрібно скористатися ODBC-драйвером. Таким чином, Delphi можна використовувати при створенні додатків типу клієнт/сервер будь-якого масштабу.

Для створення звітів Delphi містить набір інтегрованих компонентів `TQuickReport`.

Для програмування безпосереднього доступу до апаратного за безпечення Delphi надає в розпорядження програміста інтегрований асемблер. Завдяки цьому Intel-Assembler-код може безпосередньо включаться в код програм Object Pascal. Інтегрований асемблер значною мірою використовує синтаксис, підтримуваний компіляторами Borland Turbo Assembler і Macro Assembler фірми Microsoft. Інтегрований асемблер Delphi є різновидом асемблер-компілятора, що використовує синтаксис мови Object Pascal.

Delphi надає можливості для розробки власних компонентів.

Сучасний ринок програмних продуктів надає широкий вибір готових до роботи компонентів, що мають найрізноманітніше застосування.

## 13 Завдання для індивідуального програмуванняня на DELPHI

1. Створіть обробник події, що буде обчислювати корені квадратного рівняння. Конструкція форми складається з трьох вікон і трьох кнопок.
2. Створити додаток, що дозволяє перевірити, чи є число, введене користувачем, простим. Просте число відобразити в компоненті `Tmemo`.
3. Побудувати додаток, у якому ширина компонента `TEdit` змінюється за допомогою компонента `TScrollBar`.
4. Створити головне меню додатка, що складається з трьох пунктів: "Відкрити", "Зберегти", "Вихід".
5. Створити додаток для введення тексту і записування його на диск. Використовувати компонент `TOpenDialog`.
6. Створити локальне меню для компонента "мітка", що дозволяє змінювати колір виведеної в мітці інформації.
7. Створити додаток для розрахунку характеристик вектора: сума

елементів вектора, добуток елементів вектора, сума квадратів елементів вектора. Вибір характеристики здійснити за допомогою компонента `TRadioGroup`.

8. Створити додаток, що дозволяє завантажувати зображення виборів імені зі списку компонента `TListBox`.
9. Побудувати таблицю чемпіонату з футболу, використовуючи компонент `TstringGrid`. Організувати підрахунок результатів при кожному введенні інформації в таблицю.
10. Створити додаток для подання даних у графічному вигляді. Для побудови графіка використовувати компонент `TChart`. Вихідну інформацію завантажити з файлу за допомогою компонента `TOpenDialog`.

## 14 Приклади программ з різних тем

### 14.1 Приклад виконання вправи з теми 1

**Завдання**

Розробити конструкцію форми, поданої на рис. 52.

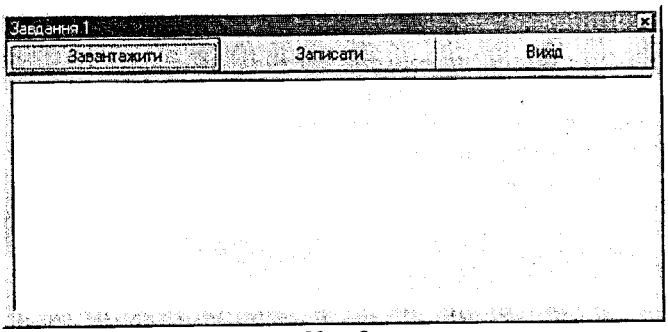



Рисунок 52 – Форма до завдання

Таблиця 3 – Виконання завдання 1

| Алгоритм | Конкретна дія, відповідно запропонованому алгоритму |
|---|---|
| 1 | 2 |
| 1. Установити властивості форми | Переміщаючи курсором миші краї форми, встановлюємо потрібні розміри форми. Властивості форми `Caption` задаємо значення «Завдання 1». Властивості форми `BorderStyle` задаємо значення `bsSizeToolWin` для того, щоб виключити кнопки мінімізації і максимізації вікна |

Продовження таблиці 3

| 1 | 2 |
|---|---|
| 2. Вибрати компоненти палітри і розмістити на формі | На панелі Standard натисненням миші вибираємо компонент TButton. Виконуємо клацання мишею для розміщення компонента у вікні проектувальника форм. Подібні дії робимо для двох наступних кнопок і для компонента TMemo |
| 3. Установити положення і розміри компонентів за допомогою миші. | Пересуваємо компонент мишею у верхню частину вікна.Захопивши курсором миші край кнопки, збільшимо її ширину. Аналогічні дії зробимо з двома наступними кнопками. Захопивши курсором миші верхній край компонента Memo1, збільшимо висоту компонента |
| 4. Додати компонентам потрібні властивості, використовуючи сторінку Properties інспектора об'єктів | Змінимо послідовно значення властивості Caption для кнопок на «Завантажити», «Відкрити», «Вихід». Очистимо рядок редактора Memo1, вибравши подвійним клацанням властивість Lines цього компонента |

### *Завдання для самостійного виконання з теми 1*

### *Завдання 1.1*

Сконструювати форму, подану на рис.53. Форма додатка повинна бути розташована у центрі. При максимізації форми положення вікна редактора змінюється разом з формою.

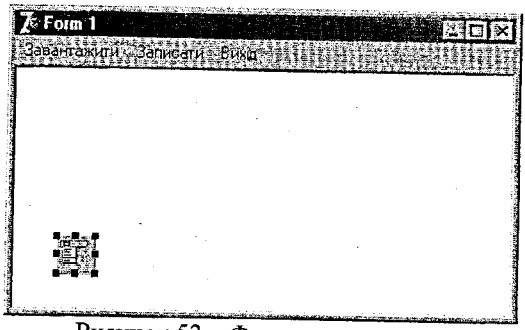



Рисунок 53 – Форма до завдання

### *Завдання 1.2*

Створити форму, подану на рис. 54

### *Завдання 1.3*

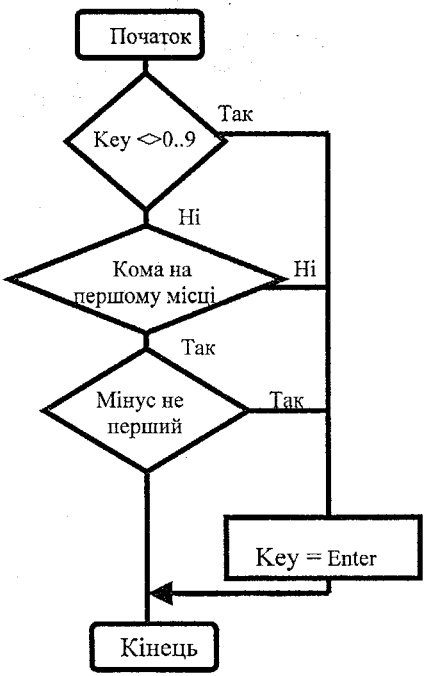
Створити форму, що відповідає за виглядом мікрокалькуляру (рис.4). Розмір форми не повинен змінюватися.

## 14.2 Приклад виконання вправи з теми 2

Завдання

На рис. 54 наведена конструкція форми для розв'язування квадратного рівняння. Створити обробник події, що надасть захист від уведення недопустимих символів у редактори `Edit1, Edit2, Edit3`.

Таблиця 4 – Виконання завдання 2

| Алгоритм | Конкретна дія, яка відповідає поданому алгоритму |
|---|---|
| 1. Вибрати компонент, для якого створюється подія | Натиснення мишею на компоненті `Edit1` |
| 1 | 2 |
| 2. Вибрати ім'я обробника події на сторінці `Events` інспектора подій | На сторінці `Events` інспектора подій вибираємо подію `OnKeyPress` |
| 3. Викликати заготівлю процедури у вікні коду програми | Подвійне натиснення мишею в рядку вибраної події |
| 4. Скласти алгоритм дій, що повинні бути виконані в обробнику події |  |

Продовження таблиці 4

| 1 | 2 |
|---|---|
| 5. Увести текст програми обробника події | Procedure TForm1. EditKeyPress<br>(Sender:TObject; var Key:Char);<br>Var buf:string[20];<br>Begin buf:=(Sender AS TEdit).Text;<br>If Not (Key in ['0'..'9',',','-'])<br>Then Key:=chr(0);<br>If(Key = ',')And(pos(',',buf)<>0)<br>then key:=Chr(0);<br>If (Key='-')And(length(buf)<>0)<br>then key:=Chr(0);end; |

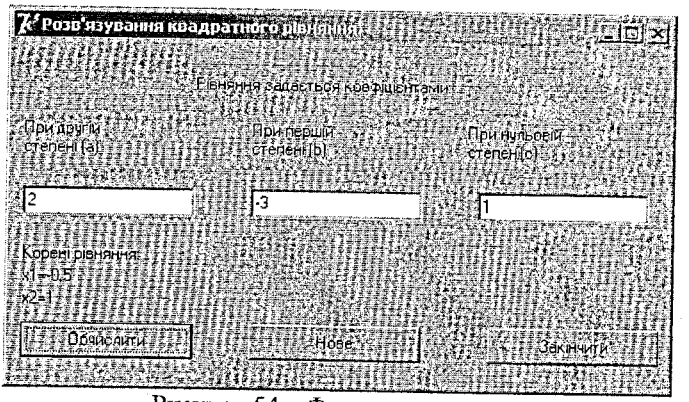



Рисунок 54 – Форма до завдання

***Завдання для самостійного виконання з теми 2***

***Завдання 2.1***

Створіть обробник події, що буде обчислювати корені квадратного рівняння. Конструкція форми подана на рис. 3.

***Завдання 2.2***

Створіть обробник події, що буде підготовлювати вікно до нових розрахунків. Конструкція форми приведена на рис. 54.

***Завдання 2.3***

Створіть обробник події для кнопок `Button1...Button18` у додатку "Калькулятор". Для ідентифікації кнопки використовувати параметр `Sender`.

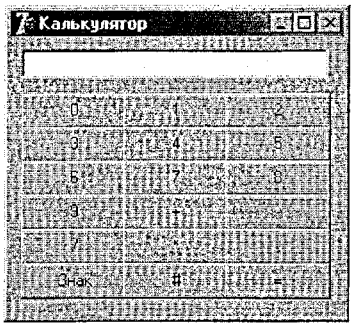



Рисунок 55 – Додаток "Калькулятор"

### 14.3 Приклад виконання вправи з теми 3

**Завдання**

Розробити додаток для тестування знань з алгоритмічної мови Turbo Pascal. Тестування проводиться за п'ятьма розділами курсу. Послідовність виконання тестування така.

Вибирається розділ предмета тестування. На екран послідовно виводяться питання і чотири альтернативних відповіді у випадковому порядку. Студент вибирає одну з відповідей. Програма підраховує кількість правильних відповідей і видає кінцеву оцінку за кожним розділом курсу.

Тестування повинно проводитися в двох режимах: без обліку часу тестування і з урахуванням часу тестування.

У програмі необхідно передбачити запис результатів тестування на диск. Для вибору розділу тестування використовувати компонент `TComboBox`.

Питання і відповіді тестів зберігаються в текстових файлах `Ftext1`, `Ftext2`, `Ftext3`, `Ftext4`, `Ftext5` відповідно до розділу тестування. Питання відділене від відповідей рядком з п'яти зірочок, оскільки питання може розташовуватися в декількох рядках.

Таблиця 5 – Виконання завдання 3

| Алгоритм | Конкретна дія, що відповідає запропонованому алгоритму |
|---|---|
| 1 | 2 |
| 1. Створити заготовку для нового додатка | Вибираємо подопцію "New Application" пункту меню "File" головного вікна Delphi.. |
| 2. Розробити конструкцію форми | Розташування і розміри на рис. 4.<br>**Форма Form1:**<br>BorderStyle – bsSingle;<br>BorderIcon – [biMinimize];<br>Position – poScreenCenter<br>**Комбінований список Combo Box 1:**<br>Text – Розділи тестування; |

Продовження таблиці 5

| 1 | 2 |
|---|---|
| | Items – Основні алгоритмічні структури,<br>Користувальницькі типи,<br>Процедури, модулі,<br>Динамічні структури,<br>Об'єкти<br>**Багаторядковий редактор Memo 1:**<br>Lines – очистити<br>**Редактор Edit 1:**<br>Text – оцінка<br>**Список ListBox.**<br>**Мітки Label 1, Label 2:**<br>Caption – очистити<br>**Панель Panel 1:**<br>Caption – очистити<br><br>**Редактор Edit 1:**<br>Text – оцінка<br>**Кнопка Bitbtn 1:**<br>Caption – далі;<br>Glyph – файл arrow3r.bmp з каталогу<br>Images\Buttons<br>**Контейнер Image 1:**<br>Picture – файл handshak.bmp з каталогу<br>Splash\256color<br>**Компонент TmainMenu:**<br>N2 – Почати;<br>N4 – Тестування на час;<br>N6 – Записати;<br>N7 – Вихід<br>**Компонент Timer:**<br>Enabled – False;<br>Interval – 180000<br>**Компонент TsaveDialog** |
| 3. Скласти перелік подій, на які повинен відгукуватися додаток | ListBox 1 Click – вибір елемента списку (правильної відповіді);<br>ComboBox 1 Click – вибір розділу тестування;<br>BitBtnClick – перехід до наступного питання розділу;<br>FormCreate - для введення прізвища тестованого;<br>N4 Click – для вимикання таймера;<br>N7 Click – кінець роботи додатка;<br>N2 Click – відновлення вихідного режиму роботи;<br>Timer 1 Timer – визначення часу закінчення тестування;<br>N6 Click – запис результату тестування на диск |
| 4.Створити обробники подій | Обробники подій приводяться нижче по тексту |
| 5. Налагодити програму | Клавіша F9 для запуску програми |

Опис змінних:

```
// Кількість питань у розділах курсу
Const count.array [1..5] of Byte=(60,90,100,120,40);
```

```
Type ar = Array[1..4] of Byte;  // Опис для масиву 4-х випадкових
                                                        чисел
Var
   f:text;              // Файлова змінна для зв'язку з файлом тестів
  otv:Array[1..4] of String;        // Масив для відповідей
  a:ar;                          // Масив для чотирьох випадкових чисел

// Змінні для рядка з файлу, імені файлу, прізвища тестованого
   s,filename,fio:String;
   Sav:Array [1..5] Of Integer;     // Масив оцінок

   // Номер питання, номер розділу, сума балів
    ivopr,  // Номер питання
     kvopr,// номер розділу
     sum:Byte;  // сума балів
```

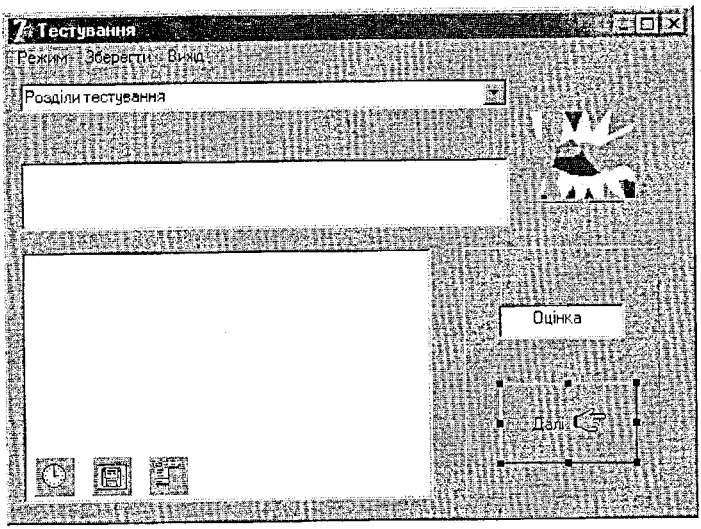



Рисунок 56 – Розробка форми простого додатка на Delphi

Допоміжна процедура для генерування масиву з чотирьох цілих випадкових чисел в інтервалі від 1 до 4:

```
  Procedure Forty(Vara:ar);
  Var j:integer; m:Boolean; a1:Set Of 1..4;
  Begin
      a1:=[];
      For j := 1 To 4 Do
       Begin
          m:=False;
```

```
        While  Not m Do
        Begin
            a[j]:=(Random(4)+1);
            If Not (a[j] In a1)Then
              Begin m:= True; a1:=a1+[a[j]] End;
        End;
    End;
End;
```

Вибір відповіді:

```
procedure TForm1.ListBox1Click(Sender:TObject);
var k:Integer;
Begin
   k:=Listbox1.itemindex; //Зафіксували номер відповіді
   Llstbox1.Itemindex:=-1;
  //Виведення правильної відповіді в положенні мітки Label 1
   Label1.Caption:='Правильна відповідь'+otv[1];
  //Якщо обрана правильна відповідь
   If a[1]=k Then sum:=sum+1;
end;
```

Вибір розділу:

```
procedure TForm1.ComboBox1Click(Sender:TObject);
Var j:integer;
begin
   Label1.Caption:='Питання 1'; Edit1.Text:='Оцінка';
   Labe3.Caption := 'Вибери відповідь';
   Randomize;
   sum:=0;
   kvopr:=ComboBox1.Itemindex+1;//Номер розділу тестування
   Filename:='Ftext'+inttostr(kvopr)+'.txt';
   Assignfile(f, Filename);
   Reset(f);
   Ivopr := 1;
// Блок зчитування питань з файлу
   s:='';
   While s<>'*****' do
   Begin
       Readln(f,s);
       If s<>'*****' Then Memo1.Lines.Add(s);
   end;
// Зчитування відповідей з файлу
For j:=1 to 4 do Begin Readln(f,s); otv[j]:=s; End;
forty(a); // Згенерували випадковий масив
// Створили відповіді в ListBox1 у випадковому порядку
For j:=1 To 4 Do Listbox1.Items[a[j]]:=otv[j];
End;
```

За допомогою кнопки переходимо до наступного питання:

```
procedure TForm1.BitBtn1Click(Sender:TObject);
Var j:Integer;
Begin      // Очистили вміст вікон
   For j:=0 To 4 Do
   Listbox1.Items[j]:=";
   Memo1. Clear;
   Label3.Caption:='Вибери відповідь';
   ivopr:=ivopr+1;
// Якщо питання ще не закінчилися
If ivopr <= count[kvopr] Then
Begin
Label1.Caption:='Питання'+ IntToStr(ivopr);
//Зчитуємо питання
s:=";
While s<>'*****' Do
Begin
  Readln (f,s);
  If s <>'*****' Then Memo1.Text:=Memo1.Text+s;     End;
// Рахували відповіді
For j:=1 To 4 Do
  Begin Readln(f,s); otv[j]:=s; End;
     forty(a);
     For j:=1 To 4 Do Listbox1.Items[a[j]]:=otv[j];
  End
  Else    // Якщо питання з теми закінчилися
  Begin  // Обробляємо результати
If (Sum / Count[kvopr] > 0.85) Then ss := 5;
If(Sum/Count[kvopr]<=0.85)And(Sum/Count[kvopr]>0.6)
      Then ss:=4;
If (Sum/Count[kvopr]<=0.6)And(Sum/Count[kvopr]>0.4)
      Then ss:=3;
If (Sum / Count[kvopr] <= 0.4) Then ss:= 2;
Label3.caption:='Питання по даній темі закінчилися!
                  Сума' + inttostr(sum);
Edit1.text:='Оцінка=' + IntToStr(ss);
Sav[Kvopr]:=ss;
End;
End;
```

Процедура для введення прізвища тестованого:

```
procedure TFormt.FormCreate(Sender:TObject);
Begin
  fio:=Inputbox('Прізвище','Введіть прізвище',");
End;
```

Обробник вибору пункту меню "Тестування на час", що є підопцією

пункту "Режим" головного меню додатка:

```
procedure TForm1.N4Click(Sender: TObject);
Begin
   Timer1.Enabled:=True; // Включили таймер
End;
```

Обробник для пункту меню "Вихід" головного меню додатка:

```
procedure TForm1.N7Click(Sender:TObject);
Begin
   Form 1.Close;
   Application.Terminate;
End;
```

Обробник для підопції "Почати" пункту меню "Режим" головного меню додатка:

```
procedure TForm1.N2Click(Sender:TObject);
begin
   fio:=InputBox('Прізвище','Введіть прізвище',");
End;
```

Обробник події OnTimer (закінчення часу, встановленого у властивості Interval компонента Timer1):

```
procedure TForm1.Timer1Timer(Sender:TObject);
Var j:Integer;
Begin   // Обробка результатів тестування
  If(Sum/Countywopr]>0.85) Then ss:=5;
  If(Sum/Count[kvopr]<=0.85)And(Sum/Count[kvopr]>0.6)
     Then ss:=4;
  If(Sum/Count[kvopr]<=0.6)And(Sum/Count[kvopr]>0.4)
     Then ss:=3;
  If(Sum/Count[kvopr] <= 0.4) Then ss:=2;
  // Виведення результатів тестування
Label3.caption:='Час! Сума = '+ IntToStr(sum);
Edit1.text := 'Оцінка = ' + IntToStr(ss);
//Зберегли результат
  Sav[Kvopr]:=ss;
   // Очистили лічильник питань і вікна додатка
  ivopr:=0;
  Memo1.Clear;
  For j:=0 To 4 Do Listbox1.Items[j]:= ";
   // Виключили таймер
  Timer1.Enabled:=False;
end;
```

Обробник підопції "Зберегти результати тестування" опції "Зберегти" головного меню додатка:

```
procedure TForm1.N6Click(Sender:TObject);
Var ff:TextFile; i:Integer;
```

```
  Begin
   With SaveDialog1 Do // Використовуємо компонент SaveDialog1
   If Execute Then     // і його метод Execute
   Begin
      AssignFile(ff,FileName); Rewrite(ff);
      Writeln(ff,fio);
      For i:=1 To 5 Do Writeln(ff,sav[i]);
      CloseFile(ff);
   End;
End;
```

### *Завдання для самостійного виконання з теми 3*

#### *Завдання 3.1*

Розробити додаток для переведення величини тиску з однієї системи вимірювання в іншу. Вибір системи вимірювання здійснити за допомогою компонента TComboBox.

– 1 дин/см$^2$ = 0.1 Па;
– 1 кгс/м$^2$ = 9.81 Па;
– 1 мм. рт. ст. = 133.0 Па;
– 1 фіз. атм. = 1. 013 × 10$^5$.

Передбачити два варіанти розрахунку: для одиничного значення і для інтервалу значень (вибір варіанта можна організувати за допомогою TRadioGroup).

#### *Завдання 3.2*

Розробити додаток для введення і виведення результатів сесії навчальної групи. Вивести список двійочників (1,2,3); список студентів, що здали сесію без трійок (4,5,6); список студентів, що здали сесію на одні п'ятірки (10,11,12). Введення початкової інформації подати в табличній формі (використовувати компонент TStringGrid). Вибір типу виведення здійснити за допомогою компонента TListBox. Передбачити виведення на друкувальний пристрій.

#### *Завдання 3.3*

Розробити довідкову систему за компонентами Delphi. Вибір сторінки бібліотеки компонентів здійснити за допомогою TTabControl.

## 14.4 Приклад виконання вправи з теми 4

### Завдання

Створити додаток, що забезпечує роботу з базою даних. База даних являє собою адресну книгу. В адресній книзі зберігаються прізвища, адреси і номери телефонів. Організувати введення інформації в базу даних, переміщення за записами бази даних і перегляд інформації вибором зі списку прізвищ.

**Виконання**

У програмі використаємо стандартний клас TStringList для збереження прізвищ адресатів. Прізвища адресатів будемо виводити в компоненті TListBox. Вся інша інформація буде виводитися в однорядкові редактори. Тут інформацію можна відкорегувати. Переміщення за списком організуємо вибором елемента списку в компоненті TListBox і за допомогою кнопок TBitBtn.

У списку рядків AdressList типу TStringList зберігаються тільки прізвища. З кожним рядком списку зв'язаний об'єкт, у якому зберігаються адреси і телефон.

Таблиця 6 – Виконання завдання 4

| Алгоритм | Конкретна дія, що відповідає запропонованому алгоритму |
|---|---|
| 1.Описати новий клас | Назвемо новий клас ім'ям TAdress, опишемо поля і методи нового класу:<br>`Type TAdress=Class Name, Street,Numb, Appart, Tel:String[20]; Constructor Create(a:string); End;`<br>У розділі implementation помістимо реалізацію методу, описаного в новому класі:<br>`Constructor TAdress.Create; Begrin Inherited Create; Name:=a; End;` |
| 2. Описати об'єкт створеного класу | Опишемо об'єкт нового класу:<br>`Var Adress:TAdress;` |
| 3. Використовувати в програмі властивості і методи нового класу | Звернемося до методу Create класу TAdress:<br>`AdressList.AddObject('',TAdress.Create(''));`<br>Звернемося до полів об'єкта Adress, зв'язаного з рядком списку AdressList:<br>`With(AdressList.Objects[n] AS TAdress) Do Begin Street:=Edit2.Text; Numb:=Edit3.Text; Appart:=Edit4.Text; Tel:=Edit5.Text; End;` |

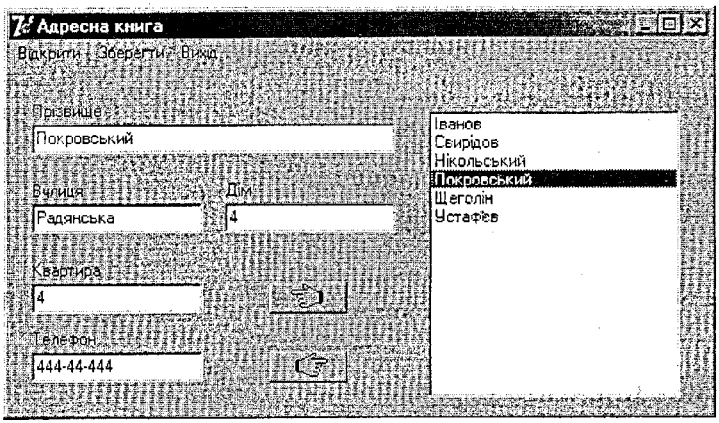



Рисунок 57 – Форма додатка для роботи з адресною книгою

У програмний модуль, зв'язаний з формою, представленою на рис. 6 додаємо такий опис:

```
Type TAdress=Class
  Name,Street,Numb,Appart,Tel:String[20];
  Constructor Create(a:String);
End;
var
  Form1:TForm1;
  AdressList:TStringList; Adress:TAdress;
      n:Integer;
```

Обробники подій.

При створенні форми створюється перший елемент списку рядків і об'єкт з ним зв'язаний:

```
procedure TForm1.FormCreate(Sender:TObject);
begin
 n:=0;
//Створюємо об'єкт типу список рядків
 AdressList:=TStringList.Create;
//Створюємо рядок з об'єктом
AdressList.AddObject('',TAdress.Create(''));
//Створюємо рядок у компоненті ListBox1
ListBox1.Items.Add(AdressList.Strings[n]);
 end;
```

При натисканні на кнопки BitBtn1 і BitBtn2 записуємо вміст екрана в список і переміщаємося в потрібному напрямку. BitBtn1 – до початку списку, BitBtn2 – до кінця списку.

```
  procedure TForm1.BitBtn1Click(Sender:TObject);
  begin
      //*** Записуємо вміст вікон у список
   AdressList-StringsIn]:=Editl.Text;
  ListBox1.ltems[n]:=AdressList.Stringst[n];
  With (AdressList.Objects[n] AS TAdress) Do
  Begin
    Street:=Edit2.Text;
    Numb:=Edit3.Text;
    Appart:=Edit4.Text;
    Tel:=Edit5.Text;
  End;
 //***
//Стрілка вліво
If (Sender As TBitBtn).Name='BitBtnl' Then
 Begin
   n:=n-1; ListBoxl.Itemindex:=n;
   If n < 0 Then Begin inc(n); Exit; End;
 End;
//Стрілка вправо
If (Sender As TBitBtn).Name='BitBtn2' Then
 Begin
   n:=n+1;
   If n+1 > AdressList.Count Then
   Begin
       //Якщо список закінчився, то додаємо новий елемент
  AdressList.AddObject('',TAdress.Create(''));
   ListBoxl.Items.Add(AdressList.Strings[n]);
   ListBoxl.Items[n]:='*';
       //Відзначаємо останній елемент списку
    AdressList.Strings[n]:='*';
   End;
 End;
//Виводимо інформацію у вікна однорядкових редакторів
Editl.Text := AdressList.Strings[n];
 With (AdressList.Objects[n] AS TAdress) Do
 Begin
   Edit2.Text:=Street;
   Edit3.Text:=Numb;
   Edit4.Text:=Appart;
   Edit5.Text:=Tel;
 End;
End;
```

При виборі елемента списку ListBoxl з'являється додаткова інформація про обраного адресата:

```
procedure TForml.ListBoxlClick(Sender:TObject);
```

```
begin
  n:=ListBox1.ItemIndex;
  Edit1.Text:=AdressList.Strings[n];
  With (AdressList.Objects[n] AS TAdress) Do
  Begin
    Edit2.Text:=Street;
    Edit3.Text:=Numb;
    Edtt4.Text:=Appart;
    Edit5.Text:=Tel;
  End;
End;
```

### *Завдання для самостійного виконання з теми 4*

#### *Завдання 4.1*

Створити додаток для роботи з базою даних. База даних зберігає таку інформацію: континент; чисельність населення; кількість держав, розташованих на континенті; назву найвищої вершини континенту; назву самої багатоводної ріки. Для збереження списку континентів використовувати тип `TListString`. Найменування континентів задати за допомогою типізованої константи-масиву.

#### *Завдання 4.2*

Нехай при клацанні на формі додатка відображається точка, якій присвоюється порядковий номер. Координати і номер точки записуються у відповідні властивості створюваного екземпляра класу `TPixel`. Після очищення форми всю послідовність точок можна відновити. Точки повинні уміти відображати себе на екрані.

## 14.5 Приклад виконання вправи з теми 5

### Завдання

Використовуючи компоненти палітри, побудувати додаток, що дозволяє відображати графічний файл `Cus5.bmp` і авторський портрет, зроблений методами класу `Canvas`. Зробити копіювання графічного зображення.

### Виконання

Вікно додатка зображене на рис. 58.

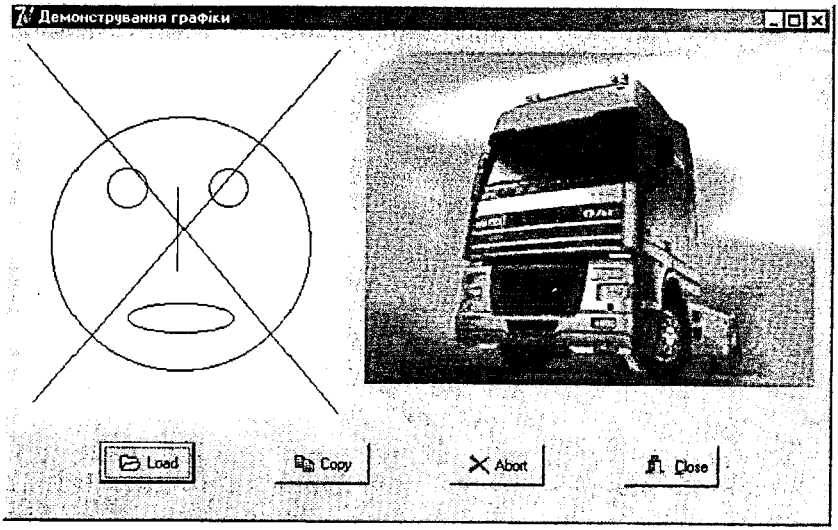



Рисунок 58 – Додаток для побудови і копіювання графічних зображень.

Таблиця 7 – Виконання завдання 5

| Алгоритм | Конкретна дія, що відповідає запропонованому алгоритму |
|---|---|
| 1 | 2 |
| 1. Визначити контейнер для збереження графічного зображення | `TImage` – для збереження графічного файлу `Cus5.bmp`.<br>`Form1` – для побудови графічного зображення |
| 2.Описати об'єкт для збереження графічного зображення | `Var MyBitmap:Bitmap;` |
| 3. Визначити методи для рисування і роботи з графічними файлами | `LoadFromFile` – для завантаження файлу `.bmp`.<br>`Arc,Ellipse,Pie` – для рисування еліпса<br>`MoveTo` – для переміщення поточного положення пера<br>`LineTo` – для зображення лінії<br>`CopyRect` – для копіювання зображення<br>`Stretch` – для заповнення зображенням усієї клієнтської області<br>`Pen.Color` – для задання кольору пера<br>`Draw` – для прорисовування графічного об'єкта `Create` – для створення об'єкта |

Продовження таблиці 7

| 1 | 2 |
|---|---|
| 4. Використовувати в програмі властивості і методи класів TCanvas і Tpicture | Використання методів в обробниках подій нижче за текстом |

Обробник події, що виникає при натисканні на кнопку "Copy", здійснює копіювання графічного зображення з об'єкта Mybitmap у компонент Image1.

```
procedure TForm1.BitBtn2Click(Sender:TObject);
begin
  Image1.Canvas.CopyRect(ClientRect,Canvas,ClientRect);
end;
```

Обробник події, що виникає при натисканні на кнопку "Load", здійснює завантаження графічного файлу в компонент Image1 і побудову авторського рисунка методами класу TCanvas.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  MyBitmap:=TBitmap.Create;
  MyBitMap.Height:=190; MyBitMap.Width:=160;
  Pen.Color:=clRed;
  Image1 .Stretch:=True;
  Image1 .Picture.LoadFromFile('Cus5.bmp');
  With MyBitmap.Canvas Do
  Begin
    Arc(5,10,150,150,150,150,150,150);
    Ellipse(20,40,40,60);
    Ellipse(100,40,120,60);
    MoveTo(70,50);
    LineTo(70,100);
    Pie(50,110,100,130,50,115,100,115);
  End;
// Рисуємо на канві форми свою бітову матрицю
  Form1.Canvas.Draw(5, 5, MyBitMap);
  end;
```

Обробник для перекреслення авторського рисунка

```
procedure TForm1.BitBtn4Click(Sender:TObject);
Var x,y:Integer;
begin
  x:=MyBitMap.Width; y:=MyBitMap.Height;
With MyBitMap . Canvas Do
  Begin
```

```
    MoveTo(0, 0);  LineTo(x, y);
    MoveTo(x, 0);  LineTo(0,y)
  End;
Form1.Canvas.Draw(5, 5, MyBitMap);
end;
```

***Завдання для самостійного виконання з теми 5***

***Завдання 5.1***

Створити додаток для зображення багатокутника. Кількість сторін багатокутника задається у вікні редактора.

***Завдання 5.2***

Створити додаток, що демонструє вигляд геометричних фігур компонента TShape. Вигляд фігури задається кнопкою TRadioGroup.

***Завдання 5.3***

У таблиці TDrawGrid відобразити картинки, що зберігаються у файлах bmp. Імена файлів задати у вигляді типізованої константи-масиву.

## 14.6 Приклад виконання вправи з теми 6

**Завдання**

Авторський рисунок, створений у попередньому додатку, роздрукувати на принтері.

Таблиця 8 – Виконання завдання 6

| Алгоритм | Конкретна дія, що відповідає запропонованому алгоритму |
|---|---|
| 1. Підключити модуль принтера до модуля форми | `Uses Printers.` |
| 2. Зв'язати файлову змінну з поточним принтером системи (для виведення тексту) або використовувати спеціальний метод принтера | `Printer.BeginDoc` |
| 3. Дати команду виведення інформації на друк | `Printer.Canvas.Draw(0,0, MyBitmap); EndDoc;` |

***Завдання для самостійного виконання з теми 6***

***Завдання 6.1***

Вивести на пристрій друку зображення багатокутника, даного в завданні 5.1.

***Завдання 6.2***

Вивести на пристрій друку результати тестування (див 6.1).

***Завдання 6.3***

У завданні 3.2 організувати виведення результатів розрахунку (список двійочників; список студентів, що здали сесію без трійок; список студентів, що здали сесію на одні п'ятірок).

## 15 Завдання для додаткової самостійної роботи

1. Створіть кнопку піднесення до квадрату числа з верхнього текстового поля.
2. Створіть програму елементарних дій калькулятора і додайте до неї кнопку "сброс", яка б звільняла всі три текстових поля.
3. Написати з використанням змінних програму для розв'язання такої задачі: автомобіль три години їхав зі швидкістю 80 кілометрів за годину і дві години зі швидкістю 90 км за годину. Обчислити середню швидкість автомобіля (вона дорівнює сумарному шляху поділеному на сумарний час). Значення змінних задати операторами присвоювання, результат надрукувати.
4. Написати з використанням змінних програму для розв'язання такої задачі: в самому кутку прямокутного подвір'я стоїть прямокутний будинок. Розрахувати площу будинку, вільну площу подвір'я і довжину паркану ( в кутку, де стоїть будинок, паркану звичайно нема). Всі числа цілі.
5. Напишіть програму для такої задачі: комп'ютер запитує назву двох планет, радіуси їх орбіт (в мільйонах км) і швидкість руху по орбіті (в мільйонах км за добу). Після цього він розраховує тривалість року на планетах і подає результат в такому вигляді: тривалість року на планеті Земля – 365 діб, а на планеті Еоелла – 12 діб. Результат в двох варіантах: виведення оператором друку і виведення в текстове поле. Вказівка: для тих, хто не знає фізики і геометрії. Рік дорівнює часу одного оберту планети по орбіті, а він дорівнює довжині орбіти, поділеної на швидкість руху по орбіті. Довжина орбіти дорівнює $2\pi R$, де R – радіус орбіти.
6. В комп'ютер вводяться два числа. Якщо перше число більше другого, то надрукувати їх суму, в протилежному випадку – добуток. Після цього комп'ютер повинен надрукувати текст: ЗАДАЧА РОЗВ'ЯЗАНА.
7. В комп'ютер вводяться довжини трьох відрізків. Комп'ютер повинен відповісти на питання чи правильно, що перший відрізок достатньо малий, щоб створити з двома іншими відрізками трикутник. Вказівка: для цього його довжина повинна бути менша суми довжин двох інших відрізків.
8. Учень вводить з клавіатури літеру з алфавіту. Комп'ютер повинен сказати, який звук позначається літерою – голосний, приголосний дзвінкий, приголосний глухий або будь-який інший (можна і НЕ ЗНАЮ).
9. Написати програму для виконання таких завдань:
   – Безконечно друкувати літеру A..AAAA....
   – Безконечно друкувати 10 000 9999 9998....
   – Безконечно друкувати квадрати натуральних чисел.

10. Якщо камінь кинути горизонтально зі стометрової башти зі швидкістю V=20 м/с, то відстань від башти до нього по горизонталі S буде виражатися формулою S=V·t, де t – час польоту каменя в секундах. Висота над Землею h буде виражатися формулою $h=100-9,8\times t^2/2$. Розрахувати і друкувати t, S і h для t=0; 0,2; 0,4; 0,6 і так далі до тих пір, поки камінь не впаде на Землю.
11. На площині задано чотирикутник координатами своїх вершин. Визначити найбільшу з його сторін і надрукувати її довжину. Передбачити можливість виведення відповіді на екран за допомогою кнопки "Результат".
12. Трикутник на площині задано координатами своїх вершин $A(X_1,Y_1)$, $B(X_2,Y_2)$, $C(X_3,Y_3)$. Визначити чи трикутник гострокутний.
13. Дано чотири числа a, b, c, d. Визначити чи є серед них від'ємні.
14. На прямій задані три точки $M_1(X_1,Y_1)$, $M_2(X_2,Y_2)$ і $M_3(X_3,Y3)$. Визначити, яка з них виявиться розміщеною між тими, що залишилися.
15. На площині задані коло $(x-x_0)^2+(y-y_0)^2=R^2$ і пряма ax+by+c=0, що перетинає коло в двох точках. Визначити чи розміщені ці точки в одній чверті, чи ні.

## 16. Приклади задач

### 16.1 Динамічні графічні об'єкти

Розглянемо приклад задачі, яка зображує більярдний стіл з одним шаром. Задається напрям руху та початкова швидкість шару після удару кием. Можна врахувати силу тертя і висвітлювати слід руху шару до попадання його в лузу обо до переривання з клавіатури.

Таблиця 9 – Компоненти, які використані в програмі

| Назва компонента | Функція (в цьому проекті) |
|---|---|
| Timage | Зображення стола |
| Tbutton | Створення удару |
| TspinEdit | Завдання сили тертя і сили удару |
| Ttimer | Періодичне оновлення стану стола |
| Tlabel | Надписи на формі |
| TstatusBar | Відображення стану програми |

**Модель більярдного стола**

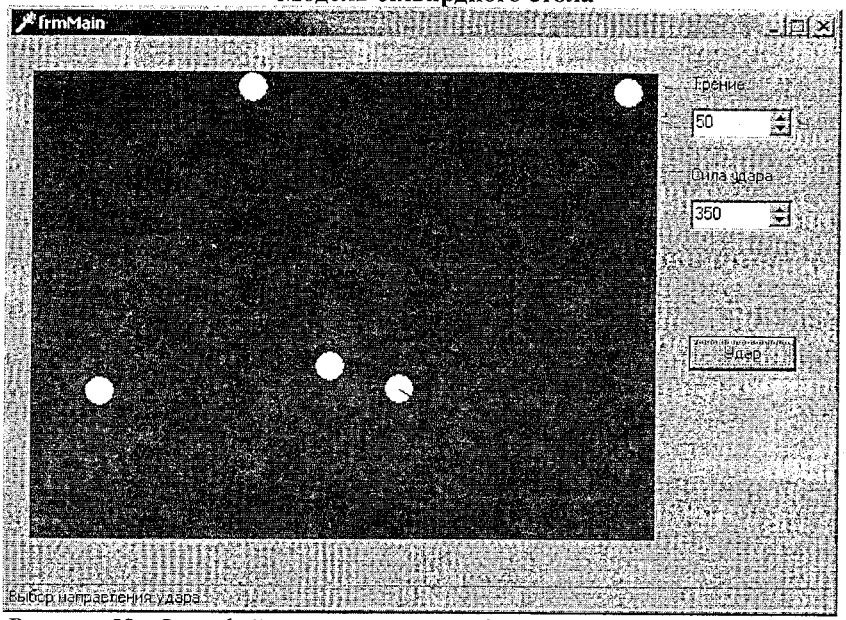



Рисунок 59 – Інтерфейс програми з динамічними об'єктами

**Глобальні визначення змінних, констант і типів даних:**

**Типи даних**

```
TCueState=(stBallSelect,stXYSelect,stPassive);
```

Використовується для змінної стану програми

```
TBall=class(TObject)
```

Клас об'єкта типу "шар"

```
public
   x, y: real; //координати шару
   dx, dy: real; //швидкість шару
   collided: TBall; //шар, з яким було останнє зіткнення
procedure Draw(canvas: TCanvas); //Відображує шар на канві
constructor Create; //Створює об'єкт класу
procedure Move(passed_time: integer); //Обробляє рух шару
function doCollisionWith(ball: TBall):boolean; //Обробляє
                                        зіткнення з іншим шаром
end;
```

**Константи**

```
Radius = 10;
```

Радіус всіх шарів

**Змінні**

```
ball_count: integer;
```
Кількість шарів

```
last_time: integer;
```
Час попереднього перерисовування

```
cue_x, cue_y: real;
```
Координати кия

```
cue_ball: TBall;
```
Шар, по якому буде проведено удар

```
cue_state: TCueState;
```
Стан кия

```
balls: array[1..100] of TBall;
```
Масив для зберігання шарів

**Реалізація методів класу TBall**

```
constructor TBall.Create; //створює об'єкт
begin
  dx:=0;
  dy:=0;
  Collided:=nil;
end;

function TBall.doCollisionWith(ball: TBall):boolean;
                          //обробляє зіткнення з іншим шаром
var
  r, v1r, v2r, u1r, u2r: real;
  rx, ry, n, nx, ny: real;
  v1n, v2n: real;
begin
   Result:=false; //вважаємо, що зіткнення немає
if (ball = self) //виключаємо можливість зіткнення з самим собою
then
      exit;
   r := sqr(x-ball.x) + sqr(y-ball.y);
   if r<4*Sqr(Radius) //перевіряємо чи відбувається зіткнення
then
   begin
        if (collided = ball) //зіткнення вже оброблено
    then exit;
        collided := ball;
```

```
            Result:=true; //зіткнення відбувається, далі розраховуємо
                                                    швидкості шарів
    r:=sqrt(r);
            rx:=ball.x-x;
            ry:=ball.y-y;
            v1r:=(rx*dx+ry*dy)/r;
            v2r:=(rx*ball.dx+ry*ball.dy)/r;
            if abs(rx)>0
            then
            begin
                ny:=1;
                nx:=-ry*ny/rx;
            end
            else
            begin
                 ny:=0;
                 nx:=ry;
            end;
            n:=sqrt(nx*nx+ny*ny);
            v1n:=(nx*dx+ny*dy)/n;
            v2n:=(nx*ball.dx+ny*ball.dy)/n;
            u1r:=v2r;
            u2r:=v1r;
            dx:=rx*u1r/r+nx*v1n/n;
            dy:=ry*u1r/r+ny*v1n/n;
            ball.dx:=rx*u2r/r+nx*v2n/n;
            ball.dy:=ry*u2r/r+ny*v2n/n;
    end
    else //зіткнення немає
        collided:=nil;
end;

procedure TBall.Draw(canvas:TCanvas);//рисуємо шар на канві
begin
    canvas.Brush.Color:=clWhite;
    canvas.Pen.Color:=clWhite;
    canvas.Pie(Integer(Round(x-Radius)),
       Integer(Round(y-Radius)),Integer(Round(x+Radius)),
       Integer(Round(y+Radius)), 0, 0, 0, 0);
end;
```

```
procedure TBall.Move(passed_time: integer); //обробка
                                          переміщення шару
var
 friction: real; //тертя
 i: integer;
begin
   friction := frmMain.speFriction.Value / 10000;
   dx := dx - dx*friction;
   dy := dy - dy*friction; //враховуємо силу тертя
  for i:=1 to ball_count do //перевірка зіткнень
  if doCollisionWith(balls[i])
       then
           break;
   x := x + dx * passed_time;
   y := y + dy * passed_time; //переміщуємо шар
if (x-Radius<0)or(x+Radius>frmMain.imgTable.Width)
   then
       dx:= -dx;
   if (y-Radius<0)or(y+Radius>frmMain.imgTable.Height)
   then
       dy:= -dy; //обробка зіткнень зі стінами
end;
```

**Глобальні процедури**

```
procedure ReDraw(time: integer); //перерисовування і обробка
```
всіх об'єктів в відповідності з пройденим часом time

```
var
 i: integer;
begin
   frmMain.imgTable.Canvas.Brush.Color:=clGreen;
   frmMain.imgTable.Canvas.FillRect( Rect(0, 0,
   frmMain.imgTable.width, frmMain.imgTable.height));
                                          //рисуємо сукно стола
for i:=1 to ball_count do
   begin
        balls[i].Move(time);
        balls[i].Draw(frmMain.imgTable.Canvas);
   end; //обробляємо в циклі всі шари
if cue_state = stXYSelect //рисуємо напрям кия, якщо
                          статус програми – вибір напряму
```

```
  then
     begin
          frmMain.imgTable.Canvas.Pen.Color:=clBlack;
frmMain.imgTable.Canvas.MoveTo(Integer(Round(cue_ball.x)),
       Integer(Round(cue_ball.y)));
frmMain.imgTable.Canvas.LineTo(Integer(Round(cue_x)),
       Integer(Round(cue_y)));
    end;
    case cue_state of //відображуємо в рядку статусу статус
                                                    програми
  stBallSelect: frmMain.stbMain.SimpleText:=Вибір шару';
  stXYSelect: frmMain.stbMain.SimpleText:='Вибір напряму
                                                    удару';
         else
  frmMain.stbMain.SimpleText:=''+IntToStr(time);
  end;
  end;
```

**Обробник подій**

```
  procedure TfrmMain.tmrDrawTimer(Sender: TObject);
        //запускаємо перерисовування з відповідним значенням часу
  var
     now: integer;
  begin
     now:=Integer(Round(Time()*24*3600*1000));//розраховуємо
                                          поточний час в мілісекундах
    ReDraw(now-last_time); //запускаємо перерисовування
    last_time:=now;
  end;

  procedure TfrmMain.imgTableClick(Sender: TObject);
                                  //Обробка клацання  на рисунку стола
  var
    i: integer;
    l,lx,ly: real;
  begin
     case cue_state of
     stBallSelect: //якщо статус – вибір шару, то знаходимо вибраний
                                                          шар
    for i:=1 to ball_count do
```

```
    if (sqr(balls[i].x-cue_x)+sqr(balls[i].y -
                              cue_y))<sqr(Radius)
            then
            begin
                 cue_ball := balls[i];
                 cue_state := stXYSelect;
            end;
     stXYSelect: //якщо статус –вибір напряму удару, то проводимо
                                                          удар
    begin
              lx:=cue_ball.x-cue_x;
              ly:=cue_ball.y-cue_y;
              l:=sqrt(sqr(lx)+sqr(ly));
cue_ball.dx:=cue_ball.dx+lx/l*frmMain.speStrength.Value/1000;
cue_ball.dy:=cue_ball.dy+ly/l*frmMain.speStrength.Value/1000;
cue_state := stPassive;
      end;
    end;
  end;

  procedure TfrmMain.imgTableMouseMove(Sender: TObject;
    Shift: TShiftState; X, Y: Integer); // Обробка руху миші
  begin
     cue_x := x;
     cue_y := y;
  end;

  procedure TfrmMain.btnCueClick(Sender: TObject);//Обробка
                                                          удару
  begin
     cue_state := stBallSelect; //Змінюємо статус на вибір шару
  end;
```

### 16.2 Приклад програми "Розв'язання нелінійних рівнянь"

Для кращого оволодіння прийомами програмування Delphi наведемо ще один корисний і цікавий приклад програми, яка на основі введених даних розраховує значення кореня будь-якого нелінійного рівняння третього порядку, використовуючи для цього відомі математичні алгоритми – метод половинного ділення, метод хорд, метод дотичних і метод ітерацій.

Демонстраційне вікно програми подане на рис.60.


Рисунок 60 – Вікно вибору методу

Розглянемо як створювалась дана програма. Пояснення дамо в описовій формі, щоб мати змогу пояснити всі складні моменти.

Запускаємо Delphi, створюємо головну форму (для цього заходимо в `File->New->Application`). У нас з'являється головна форма на панелі інструментів. На закладці `Standard` вибираємо об'єкт `Button`. Розташовуємо його на головній формі. Потім виділяємо дану кнопку і в інспекторі об'єктів (`Object Inspector` в нижньому лівому кутку) в рядку з написом `Caption` пишемо назву кнопки. Аналогічно робимо з іншими трьома кнопками.

Для того, щоб при натисканні на кнопку виконувались певні дії, потрібно щоб була деяка процедура, яка б викликалась при натисканні на цю кнопку. Такі процедури називаються методами обробниками. В Delphi є готові порожні заготовки методів, які повинні викликатися при дії користувача на інтерфейс програми або при іншій події. Для того, щоб створити такий метод потрібно в інспекторі об'єктів (спочатку потрібно виділити потрібний об'єкт) на закладці `Events` клацнути мишею двічі навпроти заготовки, яка нас цікавить, в нашому випадку це `OnClick`. Зразу після цього відкриється вікно де ми побачимо наш метод обробник (тобто порожню процедуру де ми повинні ввести код, який повинен виконуватися при натисканні на кнопку). Можна створити цю саму заготовку методу обробника просто клацнувши двічі мишею на відповідний кнопці. Такі самі дії робимо і з іншими кнопками.

Тепер розглянемо як створити головне меню (той рядок з написами вверху вікна при натисканні на відповідні пункти якого виїжджає декілька підпунктів). Для цього на панелі інструментів на закладці `Standard` вибираємо об'єкт `MainMenu` і розташовуємо його в будь-якому місці нашої форми. Далі в інспекторі об'єктів (спочатку виділяємо щойно створений об'єкт) на закладці `Properties` вибираємо пункт `Items`, клацаємо на маленькій кнопочці, яка знаходиться в цьому ж рядку. Далі відкривається нове вікно в якому ми будемо створювати пункти меню і методи, які будуть викликатись при натисканні на відповідний пункт меню. В відкритому вікні на синьому прямокутнику клацаємо правою кнопкою мищі і вибираємо пункт `Insert`. Далі знов клацаємо на синій прямокутник

і в інспекторі об'єктів в рядку Caption вводимо назву пункту меню. Аналогічно створюємо інші пункти. В цей же момент часу з'являється пустий підпункт, клацаємо на ньому і в інспекторі об'єктів в рядку Caption вводимо назву підпункту. Так робимо для створення кожного підпункту. Далі двічі клацнувши на потрібному підпункті, відкривається вікно в якому знаходиться текст головної програми з порожньою заготовкою методу обробника, який викликається при натисканні на цей пункт меню. Вводимо потрібний код програми.

Створюємо ще чотири форми, для цього клацаємо мишею на кнопці New Form (вона знаходиться над Object TreeView). На кожній формі створюємо шість об'єктів типу TEdit, вони знаходяться на панелі інструментів закладка Standard, і необхідну кількість об'єктів типу TLabel, а також одну кнопку з написом «Розв'язати». Об'єкти типу TEdit потрібні нам для введення коефіцієнтів кубічного рівняння і інтервалу розрахунку коренів, а об'єкти типу TLabel для створення вигляду кубічного рівняння (тобто для покращення наочності), при натисканні кнопки буде викликатись метод обробник в якому буде виконуватись алгоритм знаходження кореня. Такі дії робимо на кожній щойно створеній формі. Кожну форму в інспекторі об'єктів в рядку Caption на закладці Properties називаємо відповідно до назви відповідного методу знаходження кореня.

Створюємо ще одну форму на якій розташуємо об'єкт типу TrichEdit, він також знаходиться на закладці Standard. В інспекторі об'єктів в рядку Lines натискаємо на кнопку з трьома точками і в вікно, що з'явилось, вводимо потрібний текст, натискаємо на кнопку ОК. Потім в Object TreeView вибираємо рядок з словом form6 і в інспекторі об'єктів в рядку BorderStyle вибираємо пункт bsNone. Створюємо ще одну форму на якій розташовуємо об'єкт типу TImage, він знаходиться на закладці Additional. В інспекторі об'єктів в рядку Picture клацаємо на кнопці з трьома точками і в вікні, що з'явилось, натискаємо на кнопку Load і вибираємо потрібний файл з рисунком. Також на даній формі розташовуємо об'єкт типу TTimer він знаходиться на закладці System. Він потрібний для затримки заставки. Потім в Object TreeView вибираємо рядок з словом form7 і в інспекторі об'єктів в рядку BorderStyle вибираємо пункт bsNone.

Тепер заходимо в меню File->New->Unit, де створюється новий модуль, в ньому ми створюємо процедуру (obchislenie) в якій пишемо алгоритм знаходження коренів рівняння. Створюємо ще три таких модуля і пишемо в них відповідні алгоритми пошуку коренів.

Отже, маємо практично всі потрібні форми і об'єкти, залишилось наповнити їх потрібними алгоритмами і отримаємо повноцінну програму.

На головній формі маємо чотири кнопки і головне меню, яке складається з двох пунктів (program і help). Пункт Program має в собі п'ять підпунктів, перші чотири відповідають назвам чотирьох методів, а

п'ятий відповідає за вихід з програми. Меню Help містить в собі одне підменю About, там міститься інформація про програму і її розробника.

При натисканні на певну кнопку потрібно, щоб завантажувалась форма де розраховувались корені рівняння відповідним методом. Для цього в порожній заготовці методу обробника ми пишемо такий рядок:

```
form2.Visible := true;
```

цей рядок означає таке: ми викликаємо метод Visible об'єкта типу TForm (form2), який дозволяє відкрити форму, якщо присвоїти true і не відкривати – якщо false. Аналогічно написано в методах обробниках інших кнопок. Розглянемо тепер пункти меню. Нехай у нас є деякий пункт меню, що повинен відкривати певну форму, для цього в порожньому методі обробнику, який ми створили раніше, пишемо такий рядок:

```
button1.Click;
```

вона означає, що за допомогою методу Click об'єкта типу TButton (button1) ми викликаємо метод обробник цього об'єкта, який повинен активуватися при натисканні на відповідну кнопку, (тобто ми запускаємо його при іншій події, події натискання на пункт меню). Аналогічно робимо для всіх інших підпунктів меню.

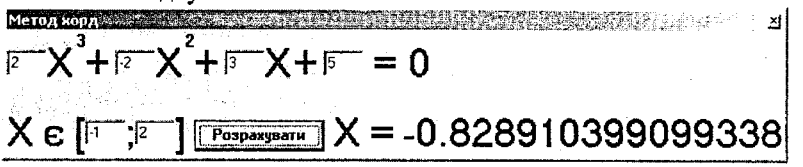



Рисунок 61 – Розрахування значень кореня

Отже при натисканні на деяку кнопку у нас завантажується певна форма на якій ми вводимо коефіцієнти кубічного рівняння. На цій формі також є кнопка «Розрахувати» (рис.61). Нам потрібно зробити так, щоб при натисканні на неї викликалась певна процедура в яку б передавались певні значення для розрахунку кореня і значення цього кореня вона б повертала в форму, яка її викликала. В нас є чотири модуля в яких знаходяться різні алгоритми пошуку коренів. Кожен з модулів ми підключаємо до відповідної форми де потрібно розрахувати корінь за певним алгоритмом. В методі обробнику кнопки, який викликається при натисканні на кнопку, пишемо такі рядки:

```
x1 := StrToFloat(Edit5.Text);
x2 := StrToFloat(Edit6.Text);
koef1 := strtofloat(edit1.Text);
koef2 := strtofloat(edit2.Text);
koef3 := strtofloat(edit3.Text);
freech := StrToFloat(Edit4.Text);
obchislenie(x,x1,x2,f,f1,f2,freech,koef1,koef2,koef3);
```

```
x1 := StrToFloat(Edit5.Text);
```
←Цей рядок за допомогою методу Text об'єкта типу TEdit отримує текстове значення цього рядка (даний

об'єкт на формі має вигляд рядка в який можна вводити текст) далі за допомогою функції StrToFloat цей текстовий рядок перетворюється в число, якщо даний рядок не є числом, то буде видана помилка, і це число записується в змінну x1 ;

```
x2 := StrToFloat(Edit6.Text);
koef1 := strtofloat(edit1.Text);
koef2 := strtofloat(edit2.Text);
koef3 := strtofloat(edit3.Text);
freech := StrToFloat(Edit4.Text);
```

Вищеподані рядки роблять аналогічні дії, як і перший. Тобто ми отримали коефіцієнти і межі між якими потрібно визначити корінь. Далі значення цих змінних передаються в процедуру, яка написана нижче.

```
obchislenie(x,x1,x2,f,f1,f2,freech,koef1,koef2,koef3);
```

– даний рядок викликає з підключеного модуля (з алгоритмом знаходження кореня) процедуру, яка шукає корінь на певному проміжку. Сама процедура зразу звертається до викликаної форми, і змінює значення одного об'єкта типу TLabel в якому знаходиться значення кореня. Сама зміна відбувається за допомогою рядка:

```
Form2.Label14.Caption := FloatToStr(x2);
```

Даний рядок означає, ми на другій формі беремо об'єкт під назвою Label14 і за допомогою методу Caption (ця процедура є методом об'єкта типу TLabel, яка дозволяє змінювати текстовий рядок даного об'єкта, даний об'єкт призначений для відображення текстового рядка на формі) ми записуємо в нього значення правої частини. В правій частині за допомогою процедури FloatToStr значення числової змінної перетворюється в текстовий рядок.

Розглянемо останню форму, це форма № 7. На ній розташований рисунок і таймер (рис. 62). Таймер потрібний для деякої затримки рисунка на екрані. Таймер є особливим об'єктом, оскільки має небагато властивостей, він завжди активізується через певний проміжок часу і при своїй активізації викличе певну процедуру. У цього об'єкта, як і більшості є змінна Tag, в основному вона ніколи не використовується, але ми її використаємо як лічильник. Ця змінна характеризується тим, що зберігає своє значення впродовж роботи програми. Цим ми і скористаємось. Оскільки таймер при кожній активації викликає лише одну процедуру або метод, то можна сказати що ця процедура знаходиться в циклі, який закінчиться лише після дезактивації таймера. Отже у нас таймер активується кожні 1000 одиниць часу (цей інтервал можна змінити, якщо скористатися методом Interval цього таймера і вказати нове значення) це відбувається 4 рази поки змінна Tag не стане рівною чотирьом.

Розглянемо детальніше такі рядки з даної форми:


Рисунок 62 – Демонстраційне вікно програми

```
  procedure TForm7.Timer1Timer(Sender: TObject);
  begin
     timer1.Tag:=timer1.Tag+1;{<-даний рядок збільшує
                                значення змінної Tag на 1}
     if timer1.Tag = 4 then
     begin
     form7.Visible:=false;{<- даний рядок робить форму сім
                невидимою за допомогою методу Visible }
     form1.Enabled:=true;{даний рядок робить форму 1
                                                активною}
  end;
end;
```

Початкові тексти програм і модулів виглядають таким чином:

**1. Модуль допомоги;**

```
  unit Help;
       interface

  uses
  Windows,   Messages,   SysUtils,   Variants,   Classes,
Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls, ComCtrls;
  type
    TForm6 = class(TForm)
    Image1: TImage;
    RichEdit1: TRichEdit;
    procedure Image1Click(Sender: TObject);
    private { Private declarations }
    public  { Public declarations }
  end;
  var
  Form6: TForm6;
  implementation
    {$R *.dfm}
  procedure TForm6.Image1Click(Sender: TObject);
  begin
       close;
  end;
  end.
```

**2. Модуль, що створює форму 7;**

```
unit Intro;
interface
uses
Windows,   Messages,   SysUtils,   Variants,   Classes,
Graphics,
Controls, Forms, Dialogs, ExtCtrls;
type
TForm7 = class(TForm)
  Timer1: TTimer;
  Image2: TImage;
  procedure Timer1Timer(Sender: TObject);
  private { Private declarations }
  public  { Public declarations }
end;
var
   Form7: TForm7;
   implementation
   uses main;
  {$R *.dfm}
procedure TForm7.Timer1Timer(Sender:TObject);
begin
   timer1.Tag := timer1.Tag+1;
   if timer1.Tag = 4 then
    begin
        form7.Visible:=false;
        form1.Enabled:=true;
    end;
end;
end.
```

**3. Модуль роботи з формами**

```
unit Main;
interface
uses
Windows,   Messages,   SysUtils,   Variants,   Classes,
Graphics,
Controls, Forms, Dialogs, Menus, StdCtrls, ExtCtrls;
type
TForm1 = class(TForm)
  Button1: TButton;
  Button2: TButton;
  Button3: TButton;
  Button4: TButton;
  MainMenu1: TMainMenu;
  Program1: TMenuItem;
  N1: TMenuItem;
  N2: TMenuItem;
  N3: TMenuItem;
  Exit1: TMenuItem;
```

```
    Help1: TMenuItem;
    About1: TMenuItem;
    N4: TMenuItem;
    procedure Exit1Click(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
      private  { Private declarations }
      public   { Public declarations }
  end;
  var
  Form1: TForm1;
  implementation
     uses method2,method3,method4,method5,help; {$R *.dfm}
  procedure TForm1.Exit1Click(Sender: TObject);
     begin  close; end;
  procedure TForm1.N1Click(Sender: TObject);
    begin  button1.Click;  end;
  procedure TForm1.Button1Click(Sender: TObject);
    begin  form2.Visible := true;  end;
  procedure TForm1.N2Click(Sender: TObject);
    begin   button2.Click; end;
  procedure TForm1.Button3Click(Sender: TObject);
    begin   form4.Visible := true; end;
  procedure TForm1.Button2Click(Sender: TObject);
    begin  form3.Visible := true;  end;
  procedure TForm1.N3Click(Sender: TObject);
    begin  button3.Click;  end;
  procedure TForm1.N4Click(Sender: TObject);
    begin  button4.Click; end;
  procedure TForm1.Button4Click(Sender: TObject);
    begin  form5.visible := true;  end;
  procedure TForm1.About1Click(Sender: TObject);
    begin   help.Form6.Visible := true;  end;
end.
```

**4. Модуль обробки методів (аналогічно** Method3,Method4, Method5**);**

```
  unit Method2;
  interface
  uses
```

```
  Windows,    Messages,    SysUtils,    Variants,    Classes,
Graphics,
  Controls, Forms,  Dialogs,StdCtrls,unit1;
  type
  TForm2 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit3: TEdit;
    Label7: TLabel;
    Label8: TLabel;
    Edit4: TEdit;
    Label9: TLabel;
    Label10: TLabel;
    Edit5: TEdit;
    Label11: TLabel;
    Edit6: TEdit;
    Label12: TLabel;
    Button1: TButton;
    Label13: TLabel;
    Label14: TLabel;
    procedure Button1Click(Sender: TObject);
    private { Private declarations }
    public  { Public declarations }
  end;
  var
     Form2: TForm2;
     implementation
     {$R *.dfm}
  procedure TForm2.Button1Click(Sender: TObject);
  var x,x1,x2,f,f1,f2,freech,koef1,koef2,koef3:real;
  begin
    x1 := StrToFloat(Edit5.Text);
    x2 := StrToFloat(Edit6.Text);
    koef1 := strtofloat(edit1.Text);
    koef2 := strtofloat(edit2.Text);
    koef3 := strtofloat(edit3.Text);
    freech := StrToFloat(Edit4.Text);
    obchislenie(x,x1,x2,f,f1,f2,freech,koef1,koef2,koef3);
  end;
 end.
```

**5.Модуль обчислення коренів(аналогічно Unit2,Unit3,Unit4)**

```
  unit Unit1;
```

```
  interface
    procedure obchislenie(var x:real; x1:real; x2:real;
           f:real; f1:real; f2:real; freech:real;
            koef1:real; koef2:real; koef3:real);
  implementation
  uses          method2,Windows,Messages,SysUtils,Variants,
Classes,
       Graphics,Controls, Forms,Dialogs,Menus,StdCtrls;
  procedure obchislenie(var x:real;x1:real;x2:real;f:real;

f1:real;f2:real;freech:real;koef1:real;koef2:real;
         koef3:real);
  var j:real; a:string;
  begin
     f1:=koef1*x1*x1*x1+koef2*x1*x1+koef3*x1+freech;
     f2:=koef1*x2*x2*x2+koef2*x2*x2+koef3*x2+freech;
     if f1*f2>0 then ShowMessage('На даному проміжку
                                          коренів немає')
       else
         begin
              if f1<0 then
                  begin
                       j := x2;
                      x2 := x1;
                      x1 := j;
                  end;
              repeat
       begin
    x := (x1+x2)/2;
    f1 := koef1*x1*x1*x1+koef2*x1*x1+koef3*x1+freech;
    f2 := koef1*x2*x2*x2+koef2*x2*x2+koef3*x2+freech;
    f := koef1*x*x*x+koef2*x*x+koef3*x+freech;
    if f = 0 then
    begin
      Form2.Label14.Caption := FloatToStr(x); Break;
    end;
    if f1 = 0 then
    begin
      Form2.Label14.Caption := FloatToStr(x1); Break;
    end;
    if f2 = 0 then
    begin
      Form2.Label14.Caption := FloatToStr(x2); Break;
    end;
    if f1*f>0 then x1 := x
    else if f2*f>0 then x2 := x;
    end;
    until ((f<=0.0001) and (f>=-0.0001));
    Form2.Label14.Caption := FloatToStr(x);
```

```
  end;
end;
end.
```

## 17 Питання для самоконтролю

### 17.1 Розділ „Структура програми на Delphi"

1. Після якого рядка ви можете розмістити опис констант, змінних, функцій, процедур?
2. Що означає рядок {$R * . RES}?
3. Що описується в розділі uses?
4. Що означає команда implementation?
5. Що таке об'єктно - орієнтоване програмування?
6. Яка різниця між розділами private і public?

### 17.2 Розділ „Керування проектами. Менеджер. Компілятор"

1. Які функції менеджера проектів?
2. Для чого використовують список TO-DO List?
3. Що таке компілятор та його директиви?
4. Перелік основних файлів проекту Delphi.
5. Яка різниця між функціями та процедурами?
6. Які функції вікна редактора коду?
7. Які функції вікна „інспектор об'єктів"?
8. Для чого підключають бібліотеки DLL?

### 17.3 Розділ „Класи. Типи даних"

1. Що таке порядкові типи даних?
2. Для чого використовують цілі типи даних?
3. Назвіть кілька цілих типів даних та вкажіть діапазон їх значень.
4. Для чого використовують дійсні типи даних?
5. Що таке булевий тип даних?
6. Що таке символьні типи даних? Для чого вони призначені?
7. Що собою представляють типи рядків?

### 17.4 Розділ „Загальні властивості компонентів. Базові класи. Події"

1. Що таке палітра компонентів VCL?
2. Для чого використовуються компоненти Label, Edit, Button?
3. Які компоненти використовуються для відображення дат, часу?
4. Яка різниця між компонентами Richedit та Memo?
5. Для чого призначений компонент F1Book?

6. Назвіть означення і клас властивості Left.

### 17.5 Розділ „Форма. Створення. Знищення"

1. Що таке форма і для чого вона призначена?
2. Що таке OnCreate форми?
3. OnDestroy?
4. OnClick, OnDobleclick?
5. Які основні властивості форми?
6. Для чого призначена властивість форми Align?

### 17.6 Розділ „Стандартні компоненти"

1. Назвіть декілька компонентів зі сторінки Win 32.
2. Яка різниця між компонентами TabControl і PageControl?
3. Для чого призначений компонент DataSource?
4. Яка різниця між компонентом сторінки DataControls „Мітка" і сторінки Standart Label?
5. Для чого призначений компонент Table сторінки BDE?
6. Нарисуйте схему зв'язку компонентів для взаємодії з БД.
7. Назвіть кілька компонентів сторінки Internet.

### 17.7 Розділ „Робота з графічним пакетом"

1. Які компоненти використовуються для роботи з графікою?
2. Які існують режими рисування в Delphi?
3. Які можливості класу TCanvas?
4. Які компоненти використовуються для відображення анімації?
5. Що таке MediaPlayer?
6. Базові компоненти для відображення графічної та мультимедіа інформації.

## 18 ЛІТЕРАТУРА


1. Фаронов В.В. Delphi 4, Учебный курс.- М.:Издательство "Нолидж", 1988.
2. Дарахвелидзе П., Марков Е. Программирование в Delphi 4.- СПб. БХВ-Санкт-Петербург,1999г.
3. Фаронов В.В. Delphi. Программирование на языке высокого уровня.- Питер, 2003г.
4. Джефф Дантеман, Программирование в среде Delphi, DiaSoft Ltd.- Киев, 1995г.
5. Федоров А.Г. Delphi 3.0 для всех.-М.: КомпьютерПресс, 1998 г.
6. Культин Н. Delphi 3. Программирование на Object Pascal. –СПб. БХВ. Санкт-Петербург,1998г.
7. Роб Баас, Майк Фервай, Хайдемария Гюнтер. Delphi 4. Полное руководство.- К.: Издательская группа BHV, 1999г.
8. Йенсен К. Руководство для пользователя и описание языка Паскаль. Йенсен, Н. Вирт: Пер. с англ. — М.: Финансы и статистика. 1982.- 150с.
9. Культин Н. Б. Программирование в Turbo Pascal 7. 0 и Delphi. 2-е изд., перераб. и доп. / Н. Б. Культин — СПб.: БХВ — Санкт-Петербург, 1999.- 416 с.
10. Мизрохи С. В. TURBO PASCAL и объектно ориентированное программирование/ С. В. Мизрохи.- М.: Финансы и статистика, 1992. - 192с.
11. Немнюгин С. А. Turbo Pascal: Практикум / С. А. Немнюгин. — СПб: Питер, 2001. — 256 с.
12. Стивенс Р. Delphi. Готовые алгоритмы / Р. Стивенс: Пер. с англ. — М.: ДМК Пресс, 2001. - 384 с.
13. Фаронов В. В. Программирование на персональных ЭВМ в среде Турбо Паскаль. 2-е изд. / В. В. Фаронов. — М.: Изд-во МГТУ, 1992. — 448 с.

# Додаток А
ФАЙЛ МАТЕРІАЛІВ
## Деякі стандартні процедури Object Pascal

**Procedure Abort;**

Викликає сховану виняткову ситуацію. Створюється виняткова ситуація, але на екран комп'ютера не виводяться ніякі повідомлення про помилку і виконання поточного блоку програми завершується.

**Function AnsiCompareText (const S1, S2 : String): Integer;**

Порівнює рядки S1 і S2. Верхній і нижній регістр при цьому не розрізняються. Якщо S1 і S2 ідентичні, то повертається значення 0. Якщо S1 менше ніж S2, повертається від'ємне значення, у протилежному випадку – значення більше 0.

**Function AnsiStrLower (Str : PChar) : PChar;**

Перетворює рядок із завершальним нулем у рядок з малими літерами.

**Function AnsiStrPos(Strl, Str2: PChar) : PChar;**

Повертає покажчик на початок першої знайденої послідовності символів (підрядка) Str2 у рядку Strl. Якщо підрядок Str2 не міститься в Strl, то функція повертає значення nil.

**Procedure AppendStr (var Dest: string; const S: string);**

Приєднує рядок S до кінця рядка Dest.

**Function ArcCos (X: Extended): Extended;**

Повертає значення арккосинуса аргументу X. Значення X повинно перебувати в діапазоні від -1 до 1. Результат повертається в радіанах.

**Function ArcSin (X: Extended): Extended;**

Повертає значення арксинуса аргументу X. Значення X повинно перебувати в діапазоні від -1 до 1. Результат повертається в радіанах.

**Function ArcTan (X: Extended): Extended;**

Повертає арктангенс аргументу X. Результат являє собою абсолютне значення арктангенса X у радіанах.

**Procedure AssignFile (var F; FileName: string) ;**

Щоб уникнути конфлікту областей (scope conflicts) у Delphi 6 Assign заміняється процедурою AssignFile. Призначення даної функції полягає в тому, щоб зв'язувати внутрішню файлову змінну з зовнішнім файлом.

**Procedure AssignPrn (var F: Text);**

Процедура AssignPrn посилає всі дані, що містяться у файлі зв'язаному зі змінною F, на принтер.

**Procedure AssignStr (var PrPstring; const S: string);**

Привласнює покажчику P адресу рядка S. У Delphi дана процедура

використовується тільки для забезпечення сумісності зверху вниз.

**Procedure Break;**

Переривае цикл під час виконання операторів for, while чи repeat. З відповідного циклу буде здійснений вихід і керування буде передано першому оператору, що стоїть за циклом.

**Function ChangeFileExt (const FileName,Extension:string): string;**

Заміняє розширення зазначеного в параметрі FileName імені файлу на розширення, що зазначено в параметрі extension. Результат містить нове ім'я файлу, включаючи шлях.

**Function Clipboard : TClipboard;**

Повертає екземпляр класу TClipboard. Якщо кишеня (буфер) Windows додатком ще не використовувався, то за допомогою виклику Clipboard породжується новий екземпляр TClipboard. У протилежному випадку функція Clipboard повертає наявний об'єкт TClipboard.

**Function CompareStr(const S1, S2 : String) : Integer;**

Порівнює рядки S1 і S2. Якщо S1 і S2 ідентичні, то повертається значення 0.

**Function Concat (s1 [, s2, . . . ,sn] : String) : String;**

Призначенням даної функції є злиття двох чи більшої кількості рядків. Результат повертається у вигляді одного рядка.

**Procedure Continue;**

Ініціює наступний цикл для оператора for, while чи repeat.

**Function** DateTimeToFileDate (DateTime : TDateTime) : **Integer;**

Перетворює значення типу TDateTime у дату і час у форматі DOS.

**Function DateTimeToStr (DateTime: TDateTime) : Integer;**

Перетворює значення типу TDateTime у рядок.

**Function DateTimeToString (Var NewString : String: Const Format: String; DateTime : TDateTime) : Integer;**

Перетворить дату і час змінної DateTime у рядок, використовуючи для цього формат, визначений у параметрі Format. Кінцевий результат передається через параметр NewString. Параметр Format розглядається в описі функції FormatDateTime.

**Function DateToStr (Date : TDateTime) : String;**

Перетворює дату в рядок.

**Function DayOfWeek (Date : TDateTime) : Integer;**

Повертає порядковий номер поточного робочого дня.

**Function DeleteFile (Const FileName : String) : Boolean;**
Видаляє із жорсткого диска файл з ім'ям FileName

**Function DiskFree (Drive : Byte) : Int64;**
Повертає розмір вільного дискового простору на зазначеній дискеті або жорсткому диску.

**Function DiskSize (Drive : Byte) : Int64;**
Повертає загальний об'єм пам'яті зазначеної дискети чи жорсткого диска.

**Function EncodeDate (Year, Month, Day: Word) : TDateTime;**
Перетворить отриману з параметрів Year, Month, Day дату в значення типу TDateTime.

**Function ExpandFileName (Const FileName : String) : String;**
Повертає розширене ім'я файлу (ім'я дисковода і шлях включається в ім'я).

**Function Extract File Ext (Const FileName : String) : String;**
Повертає розширення імені файлу.

**Function FileAge (Const FileName : String) : Integer;**
Повертає дату чи створення останньої зміни файлу.

**Function FileDateToDateTime (FileDate : Integer) : TDateTime;**
Являє собою підпрограму, у якій дата і час останньої зміни файлу, що відповідають формату DOS, перетворяться в значення дати і часу у форматі TDateTime.

**Function FiteGetAttr (Const FileName : String) : Integer;**
Повертає атрибути файлу FileName.

**Function FileGetDate (Handle : Integer) : Integer;**
Повертає у внутрішньому форматі DOS дату створення чи останньої зміни файлу.

**Function FileSearch (Const Name, DirList : String) : String;**
Дозволяє виконувати пошук файлу в одному чи декількох каталогах. Параметр Name містить шуканий файл, у параметр DirList передаються каталоги. Список каталогів повинен мати такий же формат, як і список, зазначений в операторі PATH файлу Autoexec.Bat.

**Function FileSeek (Handle, Offset, Origin : Integer) : Integer;**
У відкритому файлі позиціонує покажчик файлу.

**Function FileSetAttr (Const FileName: String; Attr: Integer) : Integer;**
Встановлює або змінює атрибути зазначеного через FileName файлу.

**Function FileSetDate (Handle : Integer; Age : Integer) : Integer;**
Установлює дату/час створення чи останньої зміни файлу.

**Function FileSize (Var F) : Integer;**
Повертає розмір файлу в байтах.

**Function FindFirst (Const Path: String; Attr: Integer;**
**Var F : TSearchRec): Integer;**
Здійснює пошук першого файлу з заданими атрибутами і у зазначеному каталозі.

**Procedure FloatToDecimal(Var DecVal:TFloatRec;**
**Const Value; ValueType:TFloatValue;**
**Precision, Decimal : Integer);**
Перетворює значення з плаваючою комою, подаючи його в десятковому вигляді. Параметр Precision вказує необхідну точність, допустимі значення даного параметра перебувають між 1 і 18. Параметр Precimals вказує максимальну кількість цифр перед десятковим роздільником.

**Function FloatToStrF (Value : Extended; Format: TFloatFormat;**
**Precision, Digits : Integer) String;**
Перетворює значення з плаваючою комою, передане через параметр Value, в рядкове подання. Формат рядка задається параметром Format. Параметр Precision вказує точність подання значення. Значення Digits залежить від обраного формату. Формат задається параметром Format і є типом TFloatFormat. Визначення даного типу виглядає в такий спосіб:

```
Type TfloatFormat=(ffGeneral,ffExponent,ffFixed,ffNumber,
ffCurency);
```

Можливі значення даного типу можуть бути такими:

`ffGeneral` Загальний формат подання чисел. Значення перетворюється, по можливості, у найкоротший десятковий рядок. При цьому може використовуватися подання з фіксованою чи плаваючою комою.

`ffExponent` У цьому випадку використовується формат з плаваючою комою. Перетворення числа в рядок відбувається відповідно загальному формату: -D,DDD...E+DDDD. Загальна кількість знаків у рядку, включаючи десятковий роздільник, задається параметром Precision. Параметр Digits визначає мінімальну кількість цифр в експоненті.

`ffFixed` Використовується формат з фіксованою комою. Значення перетвориться в рядок формату – DDD.DDD. Параметр Digits визначає кількість розрядів після десяткового роздільника.

| | |
|---|---|
| ffNumber | Це значення задає перетворення значення числа в рядок з використанням для його подання фінансового числового формату. Значення перетворюється в рядок формату – D.DDD.DDD.DDD. |
| ffCurrency | Це значення задає перетворення значення в рядок з використанням формату Currency. Перетворення визначається за допомогою глобальних змінних. |

**Function FloatToTextFmt (Buffer : Pchar; Const Value; ValueType : TfloatValue; Format : Pchar) : Integer;**

Перетворює значення з плаваючої коми у десяткове подання числа відповідно до зазначеного формату.

**Function Floor (X : Extended) : Integer;**

Округляє зазначену в X змінну до найближчого меншого цілого числа.

**Procedure FmtStr (Var StrResult : String; Const Format : String; Const Args : Array Of Const);**

Форматує рядок StrResult, використовуючи аргументи масиву Args і тип формату, переданий через параметр Format. Результат повертається через параметр StrResult. Рядок Format може містити об'єкти двох різних видів: просто символи і команди форматування. Просто символи копіюються в рядок, що є результатом. Команди форматування використовують аргументи зі списку аргументів і застосовують до них форматування.

Команди форматування мають такий синтаксис:
`"%"[Index ":"] ["-"] [Width] ["." Prec] Type`

Команда форматування починається із символа %. За ним йдуть:

1. Ідентифікатор індексу аргумента, необов'язковий ([`Index ":"`]).
2. Ідентифікатор вирівнювання символів за лівою межею, необов'язковий (["-"]).
3. Ідентифікатор ширини поля, необов'язковий ([`Width`]).
4. Ідентифікатор точності, необов'язковий (["." `Prec`]).
5. Символ перетворення `Type`.

**Символи перетворення**:

D – десяткове подання (Decimal): аргумент повинен являти собою ціле число.

E – наукове подання: аргумент повинен бути числом з плаваючою комою. Значення перетворюється в рядок формату –D.DDD...E+DDD.

F – подання числа з фіксованою комою: аргумент повинен бути числом з плаваючою комою. Значення перетворюється в рядок формату –DD.DDDD...

G – загальне подання: аргумент повинен бути числом з плаваючою комою, що перетворюється якщо можна в рядок мінімальної довжини. Це може бути формат Fixed чи науковий формат. Кількість значущих розрядів у рядку задається параметром Prec.

N – числове подання: аргумент повинен бути числом з плаваючою комою. Даний формат аналогічний формату подання числа з фіксованою комою з тим розходженням, що рядок результату тепер також має, при необхідності, роздільники розрядів.

M – подання у вигляді грошової одиниці: аргумент повинний бути числом з плаваючою комою. Значення перетворюється в рядок, що містить знак грошової одиниці.

P – покажчик: аргумент повинен бути змінною типу Pointer. Значення перетворюється в рядок формату: XXXX-.YYYY. Тут XXXX представляє адреса сегмента, а YYYY – зсув.

S – подання в рядковому форматі: аргумент у даному випадку повинний бути символом або рядком. Рядок чи символ вставляється замість ідентифікатора формату. Якщо ідентифікатор Prec заданий, то він призначає довжину рядка результату.

X – шістнадцяткове подання: аргумент повинний бути цілим числом. Значення перетворюється в рядок із шістнадцятковими цифрами.

**Function Format (Const Format : String;**
**Const Args : Array Of Const) : String;**

Форматує аргументи, що містяться в списку Args. Форматування здійснюється відповідно до Format-рядка Pascal. Для одержання відомостей про синтаксис рядка форматування див. опис стандартної процедури FmtStr.

**Function FormatFloat(Const Format: String; Value:Extended):String;**

Форматує значення з плаваючою комою, передане через параметр Value, за допомогою рядка Format. У цьому випадку підтримуються такі ідентифікатори:

| Ідентифікатор | Опис |
|---|---|
| 0 | Мітка-заповнювач для цифри. Якщо цифра є, то виводиться цифра. Якщо ні, то 0. |
| # | Мітка-заповнювач для цифри. Якщо цифра є, то виводиться цифра. Якщо ні, то нічого не виводиться. |

| | |
|---|---|
| . | Десятковий роздільник. Перша поява даного символа в рядку Format визначає позицію десятичного роздільника у виведеному рядку. |
| , | Роздільник розрядів. Якщо рядок Format містить одну чи кілька ком, то при виведенні роздільник розрядів буде розділяти цифри на групи по три цифри в кожній. |
| E+ | Цей символ вказує, що число повинно бути представлено в експонентному видгляді. |
| 'XX'/"XX" | Символи, взяті в лапки, просто виводяться на екран і не впливають на форматування рядка. |
| ; | Цей знак служить для розподілу сегментів опису формату додатних, від'ємних і нульових значень чисел у рядку Format. |

**Procedure Hatt[ (Exitcode : Integer)];**

Процедура Halt переривае виконання програми і повертає керування операційній системі.

**Function MinIntValue (const Data : Array Of Integer) : Integer;**

Повертає найменше значення в масиві Data з урахуванням знака.

**Function NewMenu(Owner:TComponent; Const Aname:String; Items: array Of TMenuItem) : TMainMenu;**

Створює і ініціалізує нове головне меню, що містить пункти, зазначені в Items. Через параметр Owner передається ідентифікатор форми, який належить меню. Параметр Aname визначає ім'я нового меню.

**Function Now : TDateTime;**

Повертає поточні дату і час.

**Procedure Run Error [(ErrorCode : Byte)];**

Завершує виконання програми і генерує повідомлення про помилку часу виконання, заданої параметром ErrorCode.

**Function StrUpper (Str : PChar) : PChar;**

Перетворює всі малі літери рядка Str у великі.

**Function Surn(Const Data:Array Of Double):Extended register;**

Підсумовує всі значення масиву Data.

# Додаток В
# Середовище Delphi

Система меню

**Опція File**

| | |
|---|---|
| New | Відкриває доступ до репозиторію Delphi. |
| New Application | Створює нову програму. |
| New Form | Створює нову форму і підключає її до проекту. |
| New Data Module | Створює новий модуль даних і підключає його до проекту. |
| Open | Відкриває раніше створену форму. |
| Reopen | Викликає список раніше завантажених проектів і форм для вибору і повторного завантаження. |
| Save | Зберігає активну форму. |
| Save As | Зберігає активну форму під іншим ім'ям. |
| Save Project As | Зберігає файл проекту під іншим ім'ям. |
| Save All | Зберігає файл проекту і усі відкриті модулі. |
| Close | Закриває поточну форму. |
| Close All | Закриває усі відкриті файли. |
| Use Unit | Вставляє в поточну форму посилання на інший модуль. |
| Add to Project | Додає до проекту раніше створену форму. |
| Remove from Project | Видаляє з проекту форму. |
| Print | Друкує активну форму чи модуль. |
| Exit | Припиняє роботу Delphi. |

**Опція Edit**

| | |
|---|---|
| Undelete (Undo) | Скасовує останню змінну проекту. |
| Redo | Відновлює останню змінну проекту. |
| Cut | Вирізує обраний компонент форми чи фрагмент тексту і поміщає його в буфер Clipboard. |
| Copy | Копіює в Clipboard виділені компоненти форми. чи фрагмент тексту модуля. |
| Paste | Витягає з буфера і копіює компоненти на форму чи текст у модуль. |
| Delete | Видаляє виділені компоненти або фрагмент тексту. |
| Select All | Виділяє усі компоненти форми чи весь текст модуля. |
| Align To Grid | Прив'язує виділені компоненти до сітки. |
| Bring To Front | Переміщає виділені компоненти на передній план. |
| Send To Back | Переміщає виділені компоненти на задній план. |
| Align | Викликає вікно вирівнювання виділених компонентів. |
| Size | Викликає вікно зміни розмірів виділених ком- |

понентів

| | |
|---|---|
| Scale | Масштабує виділені компоненти. |
| Tab Order | Змінює порядок обходу компонентів клавішею Tab. |
| Creation Order | Змінює порядок створення невізуальних компонентів. |
| Lock Controls | Блокує можливість переміщення компонентів на формі. |
| Add to Interface | Визначає нові властивості, методи і події для компонентів Active. |

**Опція Search**

| | |
|---|---|
| Find | Шукає фрагмент тексту і подсвічує його, якщо він знайдений. |
| Find In File | Шукає фрагмент тексту у всіх файлах проекту або тільки у відкритих файлах чи, нарешті, у всіх файлах поточного каталогу. |
| Replace | Шукає і замінює фрагмент тексту. |
| Search Again | Повторює пошук або пошук і заміну. |
| Incremental | Шукає текст за часом його введення – спочатку першу букву, потім Search дві перших букви і т.д. |
| Go to Line Number | Переміщає курсор на рядок із зазначеним номером від початку файлу. |
| Find Error | За адресою помилки періоду прогону програми відшукує фрагмент коду, зв'язаний з її виникненням. |
| Browse Symbol | Показує характеристики символа програми за його іменем (опція доступна тільки після успішного прогону програми). |

**Опція View**

| | |
|---|---|
| Project Manager | Показує вікно менеджера проекту. |
| Project Source | Показує текст файлу проекту. |
| Object Inspector | Показує вікно інспектора об'єктів. |
| Alignment Palette | Показує вікно палітри вирівнювання компонентів. |
| Browser | Показує вікно броузера об'єктів. |
| Using breakpoints | Показує вікно точок зупинки. |
| Call Stack | Показує вікно стека. |
| Watching expressions | Показує вікно спостереження за змінними/виразами. |
| Threads | Показує вікно статусу потоків команд. |
| Modules | Показує вікно модулів проекту. |
| Component List | Показує вікно для вибору компонентів. |
| Window List | Показує вікно відкритих вікон проекту. |
| Toggle Form/Unit | Переключає активність з вікна форми у вікно коду програми і назад. |

| | |
|---|---|
| Units | Показує вікно модулів. |
| Forms | Показує вікно форм. |
| Type Library | Показує вікно бібліотеки типів (використовується при розробці компонентів для застосування поза Delphi). |
| New Edit Window | Відкриває нове вікно з кодом поточного модуля. |
| SpeedBar | Ховає/показує панель інструментальних кнопок. |
| Component Palette | Ховає/показує палітру компонентів. |

**Опція Project**

| | |
|---|---|
| Add To Project | Додає файл до проекту. |
| Remove From Project | Видаляє файл із проекту. |
| Import Type Library | Імпортує в проект бібліотеку типів елементів Active. |
| Add To Repository | Поміщає проект у репозиторій. |
| Compile | Компілює модулі, що змінилися з моменту попередньої компіляції. |
| Build All | Компілює всі модулі проекту і створює програму, що виконується. |
| Syntax Check | Перевіряє синтаксичну правильність програми. |
| Information | Показує інформацію про вашу програму. |
| Web Deployment | Установлює ActiveX компонент ActiveForm на вашому Web-сервері. |
| Options | Викликається перед компіляцією проекту. |
| Web Deploy | Установлює AcliveX компонент чи ActiveForm на вашому Web-сервері. Викликається після компіляції проекту. |
| Options | Показує діалогове вікно установлення параметрів проекту. |

**Опція Run**

| | |
|---|---|
| Run | Компілює програму і здійснює її прогін. |
| Parameters | Указує командний рядок запуску вашої програми. |
| Register Active | Реєструє ваш проект у реєстрі Windows. Опція доступна для ActiveX-проектів. |
| Unregister Active | Видаляє ваш проект із реєстрів Windows. Опція Servers доступна для ActiveX-проектів. |
| Step Over | В режимі налагодження виконує поточний рядок коду і не простежує роботу підпрограм, які викликаються. |
| Trace Into | В режимі налагодження виконує поточний рядок коду і простежує роботу підпрограм, які викликаються. |
| Trace To Next Source | Програма виконується до найближчого від Line |

| | |
|---|---|
| | поточного положення курсора оператора, що виконується. |
| Run To Cursor | В режимі налагодження виконує програму і зупиняється перед виконанням коду в рядку з текстовим курсором. |
| Show Execution Point | Відображає у вікні коду оператор, на якому було перерване виконання програми. |
| Program Pause | Припиняє прогін відкладеної програми. |
| Program Reset | Припиняє прогін програми і відновлює режим конструювання програми. |
| Add Watch | Додає змінну чи вираз у вікно спостереження. |
| Add Breakpoint | Додає точку зупину. |
| Eevaluate/Modify | Відкриває вікно перевірки/зміни змінних. |

**Опція Component**

| | |
|---|---|
| New Component | Відкриває вікно експерта компонентів. |
| Install Component | Поміщає компонент в існуючий чи новий пакет. |
| Import Active Control | Додає до проекту бібліотеку типів Active-компонентів. |
| Create Component | Зберігає компонент із потрібним набором Template властивостей як шаблон для створення подібних компонентів. |
| Install Packages | Вказує пакети, необхідні на етапі конструювання і прогону програми. |
| Configure Palette | Викликає діалогове вікно настроювання палітри компонентів. |

**Опція Database**

| | |
|---|---|
| Explore | Викликає інструмент дослідження баз даних – Database, Explorer чи SQL Explorer (у залежності від версії Delphi). |
| SQL Monitor | Викликає інструмент запитів до БД – SQL Monitor. |
| Form Wizard | Викликає вікно експерта форм для створення форми, що відображає набори данних з вилучених чи локальних БД. |

**Опція Tools**

| | |
|---|---|
| Environment | Викликає вікно настроювання параметрів Delphi і її Options інструментів – палітри компонентів і броузера об'єктів. |
| Repository | Показує вікно репозиторія Delphi. |
| Configure Tools | Показує діалогове вікно редагування опції Tools. |
| Package Collection Editor | Викликає вікно редактора пакетів. |

| | |
|---|---|
| Image Editor | Викликає вікно редактора графіки. |
| Database Desktop | Викликає інструмент обслуговування БД - Database Desktop. |
| **Опція Workgroups** | |
| Browse P VCS Projects | Показує вікно колективної роботи деяких програмістів над одним проектом програми. |
| Manage Archive Directories | Показує діалогове вікно керування архівом колективного проекту програми. |
| Add Project to Version Control | Зберігає поточну версію колективного проекту |
| Set Data Directories | Показує діалогове вікно вибору каталогів для розміщення версій колективного проекту. |
| **Опція Help** | |
| Contents | Показує вміст довідкової служби Delphi. |
| Keyword Search | Відкриває діалогове вікно пошуку ключових слів. |
| What's New | Показує розділ, що описує відмінності версії 3.0 від попередніх версій Delphi. |
| Getting Started | Дає початкове знайомство з Delphi. |
| Using Object Pascal | Описує синтаксис і семантику мови Object Pascal. |
| Developing Applications | Відкриває доступ до численних розділів повного посібника з Delphi. |
| Object and Component Reference | Повний довідник з компонентів VCL. |
| Borland Home Page | Відкриває Web-броузер з доступом до сторінки Borland. |
| Delphi Home Page | Відкриває WeB-броузер з доступом до сторінки Delphi. |
| Borland Programs and Services | Відкриває Web-броузер з доступом до сторінки Programs and Services корпорації Borland. |
| About.... | Показує вікно про програму з оголошенням авторських прав і вказівкою електронних адрес доступу до сторінок Borland у мережі Internet. |

*Навчальне видання*

Людмила Михайлівна Круподьорова

**Програмування мовою Delphi**

Навчальний посібник

Оригінал-макет підготовлено автором

Редактор О.Д.Скалоцька

Навчально-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 25.11.05р.
Формат 29,7х42 1/4
Друк різографічний
Тираж 75 прим.
Зам. № 2005-195

Гарнітура Times New Roman
Папір офсетний
Ум.друк.арк. 8.48

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м.Вінниця, Хмельницьке шосе, 95, ВНТУ