МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

Г.В. Красовська, І.М. Доманецька, О.В. Федусенко

ЗАСТОСУВАННЯ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПІДХОДУ НА ПРИКЛАДІ ЗАДАЧ БУДІВЕЛЬНОЇ ГАЛУЗІ

Рекомендовано науково-методичною радою
Київського національного університету будівництва і архітектури,
як навчальний посібник
для студентів, які навчаються за напрямом підготовки
6.010104 «Професійна освіта. Комп'ютерні технології»

Київ 2015

УДК 681.3.06
ББК 32.937.26-01
К77

Автори: Г.В. Красовська, канд. техн. наук, доцент;
І.М. Доманецька, канд. техн. наук, доцент;
О.В. Федусенко, канд. техн. наук, доцент

Рецензенти: *Я.О. Слободян,* академік академії Будівництва України, д-р тенх. наук, професор;
Ю.М. Тесля, завідувач кафедри технологій управляння Київського національного університету ім. Тараса Шевченка, д-р техн. наук, професор;
С.В. Цюцюра, професор кафедри інформаційних технологій Київського національного університету будівництва і архітектури, д-р техн. наук

Рекомендовано науково-методичною радою Київського національного університету будівництва і архітектури, як навчальний посібник для студентів, які навчаються за напрямом підготовки 6.010104 «Професійна освіта. Комп'ютерні технології»

Красовська Г.В.

К77 Застосування об’єктно-орієнтованого підходу на прикладі задач будівельної галузі: навчальний посібник / Г.В. Красовська, І.М. Доманецька, О.В Федусенко. – К.: КНУБА, 2015. – 112 с

Розглянуто питання застосування об’єктно-орієнтованого підходу під час реалізації прикладних обчислювальних задач будівельної галузі. Приклади програмної реалізації наведено мовою Delphi Pascal. Послідовно викладено практичні прийоми роботи з основними конструкціями мови, графічною підсистемою та невізуальними класами-контейнерами Delphi.

Призначено для студентів, які навчаються за напрямом підготовки 6.010104 «Професійна освіта. Комп'ютерні технології» під час вивчення дисциплін «Об’єктно-орієнтоване програмування» та «Інструментальні програмні засоби розробки інформаційних систем».

УДК 004.7
ББК 32.937.26-01

ЗМІСТ

ВСТУП

Сьогодення характеризується глибоким проникненням комп'ютерних інформаційних технологій у різні процеси будівельної галузі: від роботи над кресленнями будівель і споруд до оперативного контролю й управління ходом їх будівництва.

Складність та ймовірнісний характер будівельного виробництва, взаємозамінність методів виконання робіт, конструкцій, ресурсів, необхідність оперативної адаптації ухвалених рішень до нових умов процесу зведення будівельного об'єкта потребує від інформаційних технологій, що впроваджуються, застосування методологій та технологій, що забезпечать гнучкість та можливість швидкої адаптації до мінливих умов.

Готуючи спеціалістів в галузі комп'ютерних інформаційних технологій у будівельному вузі доцільно основи системних знань давати на прикладах задач, що запозичені з предметної області.

Взаємодія різнопланових технологій, структур, ієрархічна природа будівництва, множинність проявів, індивідуальність вибору рішень дає благодатне підґрунтя для застосування на практиці методології об'єктно-орієнтованого підходу (ООП), що дозволяє розробляти програмне забезпечення підвищеної складності за рахунок поліпшення його технологічності (кращих механізмів розділення даних, збільшення повторюваності кодів, використання стандартизованих інтерфейсів користувача і тощо). Щоб технологічно грамотно застосовувати ООП, необхідно добре розуміти його основні концепції і навчитися мислити в стилі ООП у процесі розробки програми.

Об'єктно-орієнтоване програмування належить до нормативних дисциплін циклу професійної та практичної підготовки. Внаслідок вивчення дисципліни «Об'єктно-орієнтоване програмування» студенти повинні знати основні поняття, терміни та принципи об'єктно-орієнтованої технології розробки програмного забезпечення; вміти застосовувати на практиці засоби реалізації основних принципів об'єктно-орієнтованого підходу в сучасних мовах програмування високого рівня та засоби моделювання об'єктно-орієнтованого програмного забезпечення. Фахівці у галузі комп'ютерних інформаційних технологій повинні володіти сучасними методами та засобами програмування, найбільш перспективними технологіями створення комп'ютерних систем.

Прагнення програмістів мати середовище програмування, в якому б простота і зручність поєднувалися з потужністю і гнучкістю, стала реальністю з появою середовища Delphi. Це середовище забезпечує візуальне проектування користувацького інтерфейсу, об'єктно-орієнтовану мову ObjectPascal (пізніше перейменовану в Delphi) та унікальні за своєю простотою і потужністю засоби, що мають максимально прискорити і спростити створення програмних продуктів. Розвиток середовища розробки Delphi характеризується не тільки нарощуванням нових функціональних можливостей, але і введенням нових наборів компонентів та включенням новітніх інформаційних технологій: практично всі сучасні технології вже увійшли в Delphi в тій чи іншій формі. Середовище Delphi активно розширюється зовнішніми продуктами, які охоплюють все більше число етапів створення програмного продукту.

Цей навчальний посібник складено на основі матеріалів курсу «Об'єктно-орієнтоване програмування», який викладається на факультеті автоматизації інформаційних технологій Київського національного університету будівництва і архітектури.

Розділ 1.
БАЗОВІ ПОНЯТТЯ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Розвиток обчислювальних потужностей комп'ютерів наприкінці XX сторіччя відкрив широкі можливості для нарощування потужностей програм. Але методологія структурного програмування, що панувала на той час, не спроможна була подолати все більш зростаючу складність розроблюваного програмного забезпечення. Об'єктно- орієнтований підхід (ООП) до розробки програм з'явився як відповідь на кризу програмного забезпечення.

За такого підходу програма складається з об'єктів, які є відображенням (програмною моделлю) реальних об'єктів предметної області. Виконання програми зводиться до взаємодії програмних об'єктів, що відображає їх реальну взаємодію в предметній області.

> Предметна область – частина реального світу, яка має суттєве значення або безпосереднє відношення до функціонування програми. Під час розробки програми з предметної області виділяють і розглядають тільки ті об'єкти і зв'язки між ними, які необхідні для опису вимог та умов розв'язування поставлених задач.

Кожен програмний об'єкт є екземпляром (представником) певного класу. Питання розвитку (модифікації) програми зводиться до повторного використання розробленого програмного коду шляхом нарощування ієрархії класів та додавання до них нових властивостей. Все це разом забезпечило об'єктно-орієнтованому підходу беззаперечне лідерство в галузі розробки програм.

Основні риси ООП, втілені в базових принципах: ***інкапсуляція***, ***спадковість***, ***поліморфізм***, ***абстрагування***. Розглянемо основні поняття та принципи ООП детальніше.

Класи та об'єкти

За визначенням Граді Буча, ***об'єкт*** – відчутна сутність, що чітко виявляє свою поведінку [2]. Кожний об'єкт характеризується властивостями (розмір, колір, смак тощо) та діями (рухається, стоїть, дзвонить тощо). Виходячи з цього, все, що нас оточує – стіл та стілець, на якому сидимо, комп'ютер, з яким працюємо, мобільний, що деренчить у

кишені, і ми самі – є об’єктами.

За збігом основних ознак ми можемо виділити групи споріднених об’єктів – класи (люди, меблі, телефони і т. д.). ***Клас*** – це сукупність об’єктів одного типу, тобто схожих за своїми властивостями (атрибутами) та призначенням (поведінкою). Іншими словами, кожен об’єкт є екземпляром певного класу.

Екземпляр класу та об’єкт – слова-синоніми.

З точки зору програмної реалізації, клас – це сукупність полів даних (спільних властивостей екземплярів класу) та методів (процедур або функцій), призначених для обробки цих даних.

У Delphi Pascal у загальному спрощеному випадку клас описується так:

```
type
  TMyClass = class
    field_1: type_1;
    field_2: type_2;
    procedure Metod_1 (список параметрів);
    function Metod _2 (список параметрів): тип_результату;
  end;
```

де field_1, field_2 – поля класу певних типів type_1 та type_2; Metod_1, Metod_2 – методи класу.

Під час опису в тілі класу наводяться лише заголовки методів. Зазвичай тіло класу описується в розділі interface програмного модуля. Реалізація методів класу виконується окремо в розділі implementation:

```
procedure TMyClass. Metod_1 (список параметрів);
 begin
   // тіло методу Metod_1
end;
function TMyClass. Metod_2 (список параметрів): тип_результату;
 begin
   // тіло методу Metod_2
 end;
```

Зверніть увагу на те, що під час реалізації методу перед його іменем через точку вказується ім’я класу: *ім’я_класу.ім’я_методу*.

Оголошення екземплярів класу (об’єктів)

Синтаксис оголошення екземплярів класу (об’єктів) у Delphi Pascal аналогічний до оголошення звичайної змінної. Але на відміну від простої змінної типом змінної-об’єкта є клас:

```
Var
    екземпляр_класу : ім’я_класу;
```

Obj1, Obj2 : TMyClass;

В деяких об’єктно-орієнтованих мовах програмування оголошення змінної, що має тип-клас, автоматично створює екземпляр цього класу. В Delphi Pascal замість цього використовується так звана об’єктна модель посилань (object reference model). Ідея полягає в тому, що кожна змінна типу клас містить не значення об’єкта, а лише посилання (reference) або покажчик (pointer) на область пам’яті, в якій знаходиться об’єкт. Для спрощення синтаксису мови у процесі роботи з покажчиками в Delphi Pascal була вилучена (стала необов’язковою) операція розкриття посилання ^.

Конструктор та деструктор

Конструктор (constructor) – спеціальний метод класу, який в Delphi Pascal має ім’я Create і за замовчанням призначений для:

виділення пам’яті під об’єкт;

ініціалізації полів об’єкта нулями.

В класі для опису конструктора використовується спеціальне ключове слово – constructor.

Найчастіше конструктор використовується для ініціалізації полів об’єкта заданими значеннями, що визначає початковий стан об’єкта.

> Стан об’єкта характеризується складом і поточними значеннями всіх його визначених властивостей (полів, атрибутів). Зміна значення поля (декількох полів) переводить об’єкт в інший стан.

Опис класу набуває такого вигляду:

```
type
  TMyClass = class
    field_1: type_1;
    field_2: type_2;
    procedure Metod_1 (список параметрів);
    function Metod _2 (список параметрів): тип_результату;
    constructor Create(f1: тип_1; f2: тип_2);
  end;
```

Реалізація конструктора у найпростішому випадку буде такою:

```
  constructor TMyClass.Create(f1: тип_1; f2: тип_2);
     begin
        field_1:= f1;
        field_2:= f2;
        // та інші дії у разі необхідності
```

end;

Всі, сподіваємося, пам'ятають жорстке правило роботи з динамічним розподілом пам'яті:

Взяв пам'ять – поклади на місце!

Отже, якщо існує спеціальний метод для виділення пам'яті під об'єкт, має бути спеціальний метод для її звільнення. Такий метод називається деструктором (destructor) і має в Delphi Pascal ім'я Destroy:

destructor Destroy;

Якщо в конструкторі та деструкторі не треба виконувати ніяких особливих дій, окрім виділення та звільнення пам'яті, під час опису класу їх можна не вказувати і не реалізовувати. Чому? Це буде з'ясовано в темі «Спадковість».

В Delphi Pascal всі об'єкти динамічні. Для їх створення та знищення використовуються спеціальні методи класу ***конструктор*** та ***деструктор***.

Створення та знищення екземплярів класу (об'єктів)

Щоб створити екземпляр класу (об'єкт), після його оголошення треба викликати конструктор класу. Загальний синтаксис такий:

екземпляр_класу := ім'я_класу. Create (фактичні_параметри);

Obj1 := TMyClass.Create (a, b);

Obj2 := TMyClass.Create (c, d);

тут a, b, c, d – певні значення, що за типом повинні відповідати параметрам конструктора.

Найчастіше для знищення екземпляра класу (об'єкта) можна використати деструктор Destroy або метод Free (звідки він береться також буде з'ясовано трохи далі). Цей метод спочатку перевіряє, чи не дорівнює покажчик на об'єкт NIL, якщо ні, то викликається деструктор цього класу. Загальний синтаксис руйнування екземпляра класу такий:

екземпляр_класу . Destroy*;*

або

екземпляр_класу . Free;

Виклик методу Free вважається безпечнішим, тому Obj1. Free; Obj2. Free.

Використання полів та методів класу

Але перш ніж зруйнувати екземпляр класу (об'єкт), логічно було б якось скористатися його полями та методами. Для того щоб звернутися до полів або методів об'єкта, використовується звичний синтаксис для

записів:

екземпляр_класу . *поле*

або

екземпляр_класу . *метод (список_фактичних_параметрів)*

Наприклад:

Obj1. field_1 := *значення*; *змінна1* := Obj1. field_2;

Obj1. Metod_1 (*фактичні_параметри*);

змінна2 := Obj1. Metod_2 (*фактичні_параметри*);

> Не можна звертатися до полів та методів екземпляру класу до його створення (тобто до виклику конструктора) або після знищення (тобто після виклику деструктора) !!!

Винятком з цього правила є тільки особливий вид методів класу – *класові методи*.

Класові методи

Класові методи можна викликати без створення екземпляра класу. Класові методи зазвичай модифікують глобальні дані або надають певну інформацію про клас.

Опис класових методів здійснюється за допомогою ключового слова class, що ставиться перед заголовком методу:

```
Type
 TSample = class
   class function GetClassName:string;
End;
class function TSample.GetClassname:string;
 Begin
   Result:='The Sample Class';
 End;
Var
 Obj:TSample;
 S1, S2:string;
```

Виклик класового методу може здійснюватися двома способами:

1. без створення екземпляра класу (це основна перевага класових методів!):

   S1:=TSample.GetClassName;

2. після створення екземпляра класу як звичайний метод класу:

   Obj:= TSample.Create; S2:=Obj.GetClassName;

Під час реалізації класових методів необхідно дотримуватися таких

правил:

- у реалізації класових методів **можна** посилатися на конструктор та на інші класові методи;
- у реалізації класових методів **не можна** посилатися на поля класу або інші некласові методи.

Відношення між класами та об’єктами [1]

Розробка об'єктно-орієнтованого програмного забезпечення зводиться не тільки до визначення класів, які описують предметну область, та створення екземплярів цих класів (об’єктів). Класи та об’єкти не існують автономно – вони взаємодіють між собою. Тому під час розробки ретельно аналізуються зв'язки (відносини), що існують між класами. Найпоширенішими під час опису взаємодії між класами є наступні три типи зв'язків: ***асоціація***, ***узагальнення*** та ***залежність***.

Асоціація – це відношення, яке показує, що об'єкти одного класу певним чином взаємодіють з об'єктами іншого класу. Наприклад, вислів «бригада виконує роботи на будівельному майданчику» визначає асоціацію між об'єктами класів «Бригада» та «Майданчик». Ця асоціація є простою, тобто жоден із класів, що беруть участь в ній, не є важливішим, ніж інший. Відношення асоціації зображено на рис.1.1.

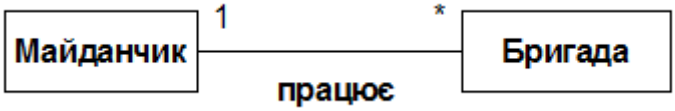


Рис. 1.1. Асоціація «Бригада-Майданчик»

Асоціації зазвичай описуються *ім'ям*, що відображає природу відносин між об'єктами. На рисунку ім'ям асоціації є «виконує роботи на». Під час визначення асоціації вказується, яка кількість екземплярів кожного класу бере участь у відношенні. Ця кількість називається *кратністю асоціації*. Так, у прикладі асоціації «Бригада-Майданчик» кратність характеризується висловом «На одному майданчику працюють багато бригад протягом спорудження об’єкта, але кожна бригада працює тільки на одному майданчику в заданий період часу». Тут слід зауважити, що будь-яка бригада може працювати на майданчиках різних будівельних об’єктів, що споруджуються будівельною організацією, але не на декількох будоб’єктах одночасно.

Особливим видом асоціації є ***агрегування*** – відношення типу *«є частиною»* (*«is-part-of»*), коли об'єкт-ціле складається з декількох об'єктів-частин. Наприклад, вислів «бригада складається з робітників» визначає

ставлення агрегації між об'єктами класів «Бригада» і «Робітник» (рис.1.2).

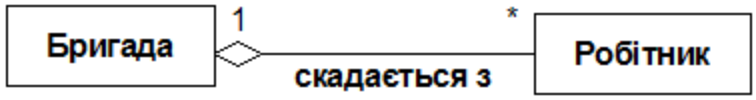


Рис.1.2. Агрегування «Бригада-Робітник»

Окремим випадком агрегування є ***композиція*** – відношення, коли час життя частин і цілого збігаються. Прикладом такого зв'язку є відношення «Будорганізація-Бригада»: після ліквідації організації бригади як самостійні одиниці існувати не можуть (рис. 1.3).

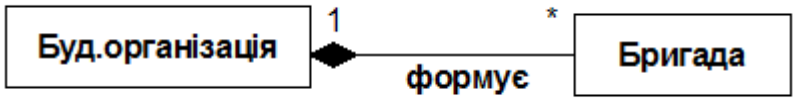


Рис.1.3. Композиція «Будорганізація-Бригада»

На відміну від цього агрегація «Бригада-Робітник» не володіє такою властивістю: при переформуванні бригад колишні бригади скасовуються, нові – формуються, при цьому об'єкти-робітники не знищуються.

Узагальнення (спадковість, генералізація) – це відношення між загальним класом (предком, батьком, суперкласом) і одним або декількома його «особливими випадками» (нащадками, дочірніми класами, підкласами). Наприклад, будівельна організація може проводити будівництво на різних видах будівельних об’єктів (рис. 1.4). Будь-який будівельний об’єкт описується властивостями: назва, замовник, адреса, терміни початку та закінчення будівництва тощо. Але в залежності від типу будівельний об’єкт буде мати свої особисті додаткові характеристики та особисту специфіку реалізації методів.

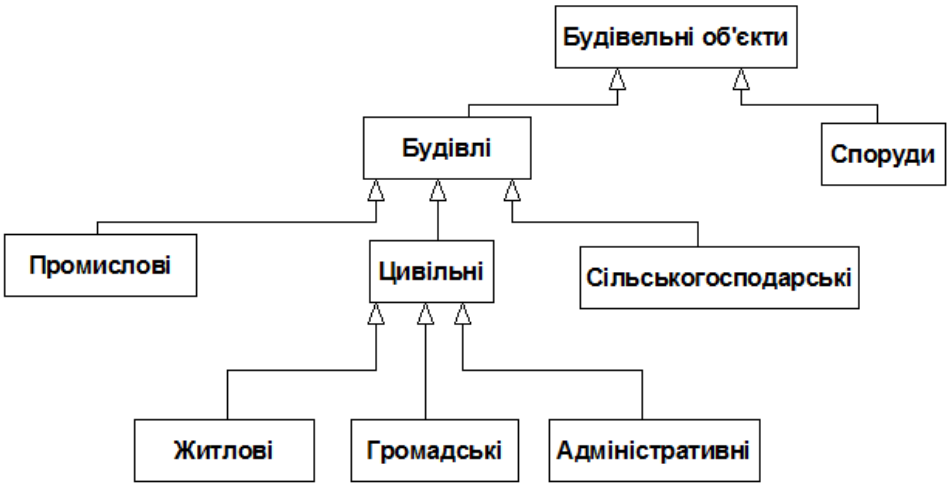


Рис.1.4. Узагальнення між класами будівельних об’єктів

Якщо продовжувати ієрархію узагальнення для житлових будівель, то окремими випадками будуть: житлові будинки, гуртожитки, готелі, пансіонати тощо. Отже узагальнення об'єднує класи за їх спільними властивостями і поведінкою. Узагальнення іноді називають відносинами типу *«є»* (*«is-a»*), вважаючи, що певні сутності (класи «Житловий будинок» і «Гуртожиток») є окремими особливими випадками іншої, загальнішої (клас «Житлові будівлі»). При цьому нащадок успадковує всі властивості предка – його поля і методи. Це дає змогу використовувати об'єкти класу-нащадка всюди, де зустрічаються об'єкти класу-предка (нащадок може бути підставлений замість предка), але не навпаки. Найчастіше, хоча і не завжди, нащадок додає свої власні поля і методи до тих, що існують у предка (детальніше це буде розглянуто в темі «Спадковість»).

У випадках, коли клас-нащадок не додає власних полів і методів, але реалізація деяких успадкованих ним методів відрізняється від батьківських, визначається відношення типу *«є подібним»* (*«is-like-a»*). Прикладом такого відношення є ієрархія класів для робітників, що працюють за різними схемами оплати праці: погодинною та відрядною. Всі працівники отримують заробітну плату. Тому методи розрахунку обсягу заробітної плати матимуть однаковий прототип, але їх реалізація для робітників з різною формою оплати праці буде різною.

Відношення ***залежності*** – це такий тип відношення, за якого зміна у визначенні одного класу призводить до зміни реалізації іншого класу. Наприклад, зміна в класі «Робітник» (додавання нових методів, зміна прототипів існуючих методів та ін.) може призвести до змін у класі «Бригада». Найчастіше такий зв'язок виникає у випадках, коли класи перебувають у відношенні агрегації або коли об'єкти одного класу є параметрами методів іншого класу.

Інкапсуляція

Одним з найважливіших принципів об’єктно-орієнтованого підходу є принцип інкапсуляції.

Інкапсуляція – базовий принцип ООП, згідно з яким поля екземпляра класу повинні оброблятися тільки методами цього класу. Тобто під час роботи з екземплярами класу небажаним є пряме звернення до полів: читання і запис їх значень повинні виконуватися безпосередньо викликом відповідних методів класу [3; 4]. Це зумовлено необхідністю забезпечення надійності

роботи як окремих екземплярів класу так і всієї програми.

Припустимо, що для певного класу TSomeClass певне поле цього класу SomeField може набувати значення у визначеному діапазоні [a, b]. Користувач помилково може ввести дані для цього поля без урахування висунутих обмежень, що призведе до непрацездатності окремих методів класу. Згідно з принципом інкапсуляції, всі перевірки щодо збереження екземпляра класу у працездатному стані необхідно передбачати всередині самого класу, а саме в методах цього класу. Тобто до списку методів TSomeClass необхідно включити метод SetSomeField, в якому буде проводитися перевірка на відповідність введеного значення встановленим вимогам і, у разі відповідності, запис цього значення в поле.

Для того щоб захистити поля та методи класу від неконтрольованого доступу ззовні, необхідно під час опису класу використовувати директиви видимості. Це дозволяє розробникові реалізовувати принцип інкапсуляції на практиці.

Директиви видимості

У процесі опису класів у Delphi Pascal використовуються такі основні директиви видимості:

- private – приватні поля та методи, звернення до яких можливе тільки в методах цього класу. Ця директива дозволяє приховати деталі внутрішньої реалізації класу. Доцільно вносити в область видимості private поля та методи, які ніколи не будуть використовуватись або модифікуватись нащадками цього класу, іншими класами або користувачами програми. Це суто внутрішні поля та методи.
- protected – захищені поля та методи, звернення до яких можливе в методах цього класу і в методах нащадків цього класу.
- public – загальнодоступні поля та методи, які не мають обмежень на доступ.

Загальний опис класів набуває такого вигляду:

```
Type
 Ім'я_класу = class
   private
     приватні поля;
     приватні методи;
   protected
```

```
    захищені поля;
    захищені методи;
  public
    загальнодоступні поля;
    загальнодоступні методи;
end;
```

> Якщо директива видимості під час опису класу не вказана, за замовчуванням приймається директива видимості **public**.

Для організації керованого доступу до захищених полів класу необхідним є створення методу для читання значення поля та методу для запису значення в поле. При цьому такі методи повинні бути описані в секції з менш жорстким обмеженням доступу, ніж саме поле. Наприклад, задано такий фрагмент опису класу:

```
Type
 TSomeClass = class
   private
    SomeField:real;
   protected
    function GetSomeField:real;
    procedure SetSomeField (New_value:real);
   public
    conctructor Create(New_A:real);
end;
function TSomeClass. GetSomeField:real;
begin
  Result:=A;
end;
procedure TSomeClass. SetSomeField (New_value:real);
begin
  if (SomeField>=a) and (SomeField<=b)    // перевірка встановленої умови
        and (SomeField <> New_value)      // перевірка, чи нове значення
                                          // відрізняється від старого
              then SomeField:= New_value;
end;
conctructor TSomeClass.Create(New_A:real);
begin
  SomeField:= New_A; // або так SetSomeField (New_A);
end;
```

Слід звернути увагу, що природним є те, що метод читання є функцією, яка як результат повертає значення поля, а методом запису є процедура з одним параметром – нове значення поля. Але дуже незручним є те, що для кожного поля, до якого регламентується доступ необхідно знати імена методів доступу.

Розширене розуміння інкапсуляції Delphi Pascal знайшло своє відображення в понятті властивості.

Властивості класів (property)

Delphi Pascal надає для захисту полів спеціальний програмний механізм, що називається ***властивістю*** (property). Загальний синтаксис опису властивості такий:

Property ім'я_властивості: тип_значення

read ім'я_методу_для_зчитування

write ім'я_методу_для_запису;

Властивість – це зовнішній шар для обробки поля (рис. 1.5).

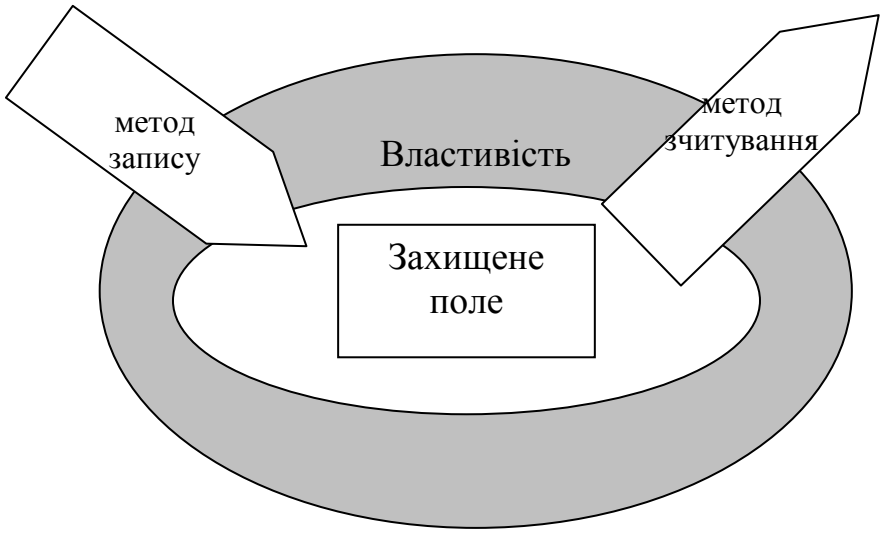


Рис. 1.5. Доступ до поля через властивість

Властивість має ім'я та тип. Тип властивості може збігатися або не збігатися з типом поля, з яким вона пов'язана. Для організації зв'язку властивості з полем використовуються два методи: метод для зчитування значення поля та метод для запису значення поля.

Метод для зчитування повинен бути функцією без параметрів. Тип результату функції повинен співпадати з типом властивості.

Метод для запису повинен бути процедурою з одним параметром.

Тип параметра повинен співпадати з типом властивості.

Якщо тип властивості і поля не збігаються, тоді в методах зчитування та запису необхідно виконувати відповідні перетворення між типом властивості і типом поля:

```
Type
 TSomeClass = class
   private
    FSomeField:real;      // згідно з угорською нотацією
   protected              // імена поля краще починати з літери F
    function GetSomeField:real;
    procedure SetSomeField (New_value:real);
   public
    conctructor Create(New_A:real);
    Property SomeField:real read GetSomeField write SetSomeField;
end;
```

Для тоді щоб записати значення в захищене поле FSomeField необхідно звернутися до властивості-property, що його захищає. Звернення до властивості виконується аналогічно зверненню до полів класу. Якщо SomeObj – екземпляр класу TSomeClass, має місце такий запис:

SomeObj. SomeField:=1,5;

При цьому автоматично буде викликаний метод запису значення SetSomeField, фактичним параметром якого стане значення 1,5. А в записі:

X:= SomeObj. SomeField;

буде викликаний метод читання значення поля GetSomeField, результат роботи якого буде підставлений у вираз.

> Для підтвердження важливості використання властивостей досить навести такий факт: в стандартних класах Delphi 100% полів недоступні для користувачів і замінені на відповідні їм властивості. Більш ніж доцільно дотримуватись цього правила і під час розробки власних класів.

Як вже було сказано, методи, що забезпечують реалізацію властивостей, можуть виконувати перевірку коректності встановлювальних значень або певні перетворення під час зчитування значень полів. Якщо ж потреби в спеціальних методах зчитування /запису полів немає, то можна використовувати замість імен методів імена полів. Наприклад, якщо в класі *TSomeClass* додати інше поле FOtherField, яке не потребує спеціальних перевірок або перетворень, властивість для цього поля може бути описана

таким чином:

property OtherField: TSomeType **read** FOtherField **write** FOtherField;

Можуть бути властивості, які використовуються тільки для зчитування або тільки для запису значення в поле:

property FirstProperty : TSomeType **read** GetFirstField;

property SecondProperty : TSomeType **write** SetSecondField;

Під час опису та використання властивостей треба дотримуватися таких правил (обмежень):

1. Властивості не можуть бути параметрами методів або підпрограм;
2. Якщо властивість описана тільки для читання (або тільки для запису), то спроба записати значення в таку властивість (або прочитати значення) викличе помилку компіляції:

SomeObj.FirstProperty := SomeData; *// ПОМИЛКА!!!*

SomeData := SomeObj.SecondProperty; *// ПОМИЛКА!!!*

3. У тілі методів читання або запису для властивості необхідно звертатися до поля класу, а не до його властивості. Як приклад розглянемо можливу небезпечну реалізацію методу запису для властивості SecondProperty, яка надає доступ до поля FSecondField:

```
procedure TSomeClass.SetSecondField (New_value: TSomeType );
begin
  if (FSecondField <> New_value)
      then SecondProperty:= New_value; // ПОМИЛКА!!!
end;
```

Звернення в методі запису властивості до цієї ж властивості спричиняє виклик цього ж самого методу запису, в якому знову відбувається звернення до властивості і т. д. В результаті утворюється необмежена рекурсія, яка буде перервана апаратно по завершенні обсягу програмного стека (повідомлення про помилку «Stack overflow»). Правильним буде такий опис:

```
procedure TSomeClass.SetSecondField (New_value: TSomeType );
begin
  if (FSecondField <> New_value)
      then FSecondField:= New_value; // ПРАВИЛЬНО!!!
end;
```

Властивість-масив

В Delphi Pascal під час організації доступу до поля класу, яке описане як масив, використовується векторна властивість або **властивість-масив**, опис якої має вигляд:

Property ім’я_властивості [Index : integer]: тип_значення

read ім’я_методу_для_зчитування

write ім’я_методу_для_запису;

Наприклад, у деякому класі описане поле:

FField : ***array*** [1..N] ***of real***;

Для зчитування та запису цього поля необхідно задати два методи:

function GetItem (Index : integer) : real;

procedure SetItem(Index : integer; New_Value : real);

Тоді властивість для доступу до цього поля буде:

property Some_Array : real **read** GetItem **write** SetItem**;**

Реалізація цих методів така:

```
function GetItem  (Index : integer) : real;
begin
    Result:= FField [Index];
end;
procedure SetItem (Index : integer; New_Value : real);
    begin
  if  FField [Index] <> NewValue then FField [Index]:=NewValue;
end;
```

Як стає зрозумілим з наведеного програмного коду, перший параметр цих методів Index використовується для визначення порядкового номера (індексації) елементів поля-масиву FField.

Доступ до властивості-масиву забезпечується за допомогою методів, які повинні обов’язково мати як перший параметр типу integer, який в методі використовується як індекс елемента масиву.

Багатомірні властивості-масиви

Для доступу до багатомірного масиву можна також описати відповідну властивість-масив. Розглянемо опис на прикладі двовимірного масиву:

Property ім’я_властивості [Index1, Index2: integer]: тип_значення

read ім’я_методу_для_зчитування

write ім’я_методу_для_запису ;

Під час опису методів доступу до їх списку формальних параметрів також додаються додаткові параметри-індекси:

function GetItem (Index1, Index2: integer) : real;

procedure SetItem(Index1, Index2: integer; New_Value : real);

Індексовані властивості

Інколи для доступу до елементів поля-масиву зручніше використовувати не індекси цих елементів, а ідентифікатори. Наприклад, у

класі описане поле-масив, в якому зберігаються координати точки. При цьому вважається, що перший елемент масиву задає координату по осі *x*, другий – по *y*, а третій – по *z*:

Coord : array [1..3] of real;

	x	*y*	*z*
	1	2	3
Coord			

Для того щоб зробити доступ до координат точки наочнішим, можна використати індексовану властивість, яка дозволить звертатися до елементів поля-масиву Coord за ідентифікаторами *x*, *y* та *z*.

Під час опису індексованої властивості методи доступу до елементів масиву Coord будуть такі ж як під час опису властивості-масиву:

function GetItem (Index : integer) : real;
procedure SetItem(Index : integer; New_Point : real);

Але для доступу до елементів цього масиву описуються три індексовані властивості (по одній властивості на кожен елемент масиву):

property x : Integer **index** 1 **read** GetItem **write** SetItem;
property y : Integer **index** 2 **read** GetItem **write** SetItem;
property z : Integer **index** 3 **read** GetItem **write** SetItem;

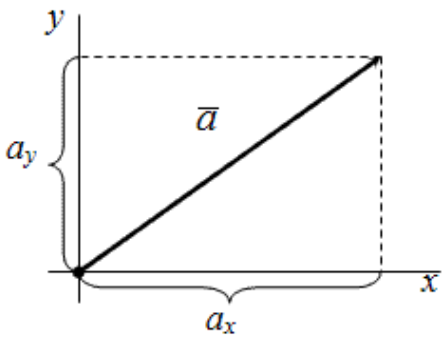
Якщо для певного екземпляра класу MyObject виконати

MyObject .x:=10;

з опису властивості *x* буде визначено індекс елемента масиву (для *x* індекс дорівнює 1) викликано метод SetItem (1, 10), в результаті чого елементу Coord [1] буде присвоєно значення 10.

Приклад опису класу

Розглянемо проектування класу для роботи з математичними векторами. Математичний вектор $\bar{a}$ в декартовій системі координат – це направлений відрізок, початок якого завжди знаходиться в точці (0, 0), а кінець визначається векторними координатами $(a_x; a_y)$.



$$\bar{a} = (a_x; a_y)$$

Рис. 1.6. Визначення вектора через векторні координати

Зі шкільного курсу математики відомо, що над вектором допустимі

такі дії (операції):

- обчислення модуля (довжини вектора);
- перевірка, чи не є вектор нульовим.

Для пари векторів можливі такі операції:

- визначення координат вектора, що є сумою, різницею двох векторів;
- визначення координат вектора, що є добутком іншого вектора на число;
- визначення скалярного добутку двох векторів;
- визначення кута між двома векторами;
- перевірка двох векторів на рівність
- перевірка ортогональності та колінеарності двох векторів.

Таким чином кожен вектор описується двома полями – координатами вектора FaX, FaY та набором операцій, що були визначені вище. Додатково описується функція, що виводить інформацію про вектор у форматі рядка символів. Опис класу для математичного вектора матиме такий вигляд:

```
type TMathVector=class(TObject)
protected                    // ЗАХИЩЕНА СЕКЦІЯ, що є доступною для нащадків
   FaX, FaY: double;  // векторні координати

 public                    // ЗАГАЛЬНО ДОСТУПНА СЕКЦІЯ
   constructor Create;
   property aX: double read FaX write FaX; // доступ до полів
   property aY: double read FaY write FaY; //  FaX, FaY відповідно

   function getAbsolute: double;    // визначення  модуля вектора
   function isZeroVector: boolean; // повернає true якщо вектор нульовий

   function summationVector(v: TMathVector):TMathVector; // сума векторів
   function substractionVector(v: TMathVector):TMathVector; // різниця
   function scalingVector(k: double):TMathVector; // добуток на число
   function multiScalarVector(v: TMathVector):double; // скалярний добуток

   function getAngle(v: TMathVector):double; // кут між векторами

   function IsEqual (v: TMathVector):boolean;       // true якщо рівні
   function IsOrthogonal(v: TMathVector):boolean; // true якщо ортогональні
   function IsCollinear (v: TMathVector):boolean;   // true якщо колінеарні

   function Info:String; // формує рядок з інформацією про вектор
```

```
end;
```

З прикладу видно, в класі є конструктор без параметрів, що ініціює поля об’єкта нулями. В класі не виконуються додаткові маніпуляції з виділення пам’яті, окрім тих, що передбачені в стандартному конструкторі, тому власний деструктор класу непотрібен. Оскільки визначення полів класу здійснене у захищеній секції, потрібно передбачити механізм для доступу до значень полів класу. В нашому випадку – це прості властивості, навіть без використання додаткових методів контролю значень, бо вектор з будь-якими координатами (додатними чи від’ємними, цілими чи дійсними тощо) має право на існування.

У складі класу є методи з префіксом Is-, що реалізовані як логічні функції, які повертають true, якщо відповідна умова виконується. Наприклад, якщо об’єкт-вектор ***v***, який ініціює метод ***v.isZeroVector***, дійсно є нульовим, то результатом виконання методу буде true, інакше false.

Необхідно також звернути увагу на методи, що реалізують двомісні операції (операції для двох векторів на кшталт суми двох векторів). Фактично операція визначення суми є бінарною операцією, тобто вона потребує два операнди:

вектор1+вектор2.

Ця операція реалізується як метод класу TMathVector. Для роботи такого методу потрібні 2 об’єкти: один – лівий операнд (вектор1) – ініціює (викликає) метод, другий – правий операнд (вектор2) – передається в метод як параметр:

вектор1. summationVector (вектор2);

Результатом роботи такого методу є новий вектор:

вектор3= вектор1. summationVector (вектор2);

```
var  v1,v2,v3: TMathVector;

v1:= TMathVector.Create;
v1. aX = 1; v1. aY = 1;

 v2:= TMathVector.Create;
 v2. aX = 2; v1. aY = 3;

v3 := v1. summationVector(v2);
```

```
Label1.Caption:= v3.Info;
```

Спадковість

Зі шкільного курсу біології всім відомо, що класи живих організмів утворюють ієрархію спадковості. При цьому класи нащадки успадковують властивості класів – предків і додають свої властивості. Представники неживої природи також можуть утворювати ієрархію зі спадкуванням властивостей (див. «Відношення класів та об'єктів», рис. 1.4). Відповідно до цього, класи програмних об'єктів, які відображають об'єкти (сутності) предметної області, також повинні утворювати ієрархію спадковості.

> Спадковість – базовий принцип ООП, згідно з яким у процесі створення нового класу, що декількома властивостями відрізняється від вже існуючого, немає необхідності переписувати заново вже існуючі поля та методи, достатньо оголосити новий клас нащадком вже існуючого класу.
> Якщо клас описаний як прямий нащадок лише одного класу – це *одинична спадковість*.
> Якщо клас описаний як прямий нащадок від декількох класів – це *множинна спадковість*.

Відношення спадковості між класами називається зв'язком типу «is a». Тобто під час спадковості має місце таке речення (див. рис. 1.4): «Житлові будинки – це цивільні будівлі», «Житлові будинки – це будівлі», «Житлові будинки – це будівельні об'єкти».

На верхньому рівні ієрархії класів можуть бути класи, які не мають конкретних екземплярів, але вони наділяють своїх нащадків певними спільними властивостями.

> Класи, які не мають екземплярів класу, називаються абстрактними класами.

Принцип спадковості має свою реалізацію і в Delphi Pascal. Узагальнений синтаксис опису класів предка та нащадка такий:

```
type
Клас_предок = class
    опис полів та методів класу_ предка;
end;
Клас_нащадок = class (Клас_предок)
    опис полів та методів класу_нащадка,
            що розширюють властивості класа-предка;
```

```
end;
```

Розглянемо створення класу-нащадка докладніше:

```
type
TParentClass = class
    field1: Type1;
    procedure M1;
end;
TSonClass = class (TParentClass)
    field2: Type2;
    procedure M2;
end;
```

Клас TSonClass описано як нащадок класу TParentClass. У класі TSonClass описано одне поле і один метод, але насправді в класі TSonClass 2 поля і 2 методи:

- поле field1, що успадковане від TParentClass та field2 – власне, що додане до властивостей предка;
- метод M1, успадкований від TParentClass та M2 – власний, доданий до властивостей предка.

> Спадковість дає змогу в програмі замість екземпляра класу-предка використовувати екземпляр класу-нащадка, тому що нащадок має всі властивості предка, але не навпаки.

Під час оголошення та створення екземплярів класу для класів, що пов’язані ієрархією спадковості, можливі два способи, які наведено у табл. 1. Незалежно від способу створення екземплярів класу-нащадка його використання не відрізнятиметься.

Таблиця 1

Створення та використання класів при спадкуванні

Оголошення екземплярів	
Var P : TParentClass; S : TSonClass;	**Var** P : TParentClass; S : TParentClass;
Створення екземплярів	
P := TParentClass.Create; S : =TSonClass.Create;	P := TParentClass.Create; S : =TSonClass.Create;
Використання екземплярів	
x:= P.field1+ S.field1+ S.field2;	
P.M1, S.M1, S.M2.	

P.field2, P.M2; ***Неправильно –*** *предок не має властивостей нащадка!*

За замовчуванням в середовищі Delphi для всіх класів є спільний предок – клас TObject. Якщо після ключового слова **class** у дужках не наведено ім'я класу-предка, тоді ним за замовчуванням стає клас TObject. Наведені нижче описи класу еквівалентні:

TParentClass = **class** …;

TParentClass = **class** (TObject) …;

Отже клас TParentClass є прямим нащадком від класу TObject. Клас TSonClass транзитивно успадковує від свого безпосереднього предка TParentClass усі властивості і методи класу TObject: конструктор Create, деструктор Destroy, метод Free тощо. Детально властивості класу TObject описано в [3; 4; 9].

В Delphi Pascal відсутня реалізація множинної спадковості. Якщо все ж таки необхідно, щоб новий клас був наділений властивостями декількох, можливі такі рішення [3; 4]:

- породити класи-предки один від одного, а потім породити клас-нащадок;
- породити клас-нащадок від одного предка, а інші включити до класу-нащадка як поля (такий спосіб реалізує відношення агрегації між класами).

Приклад реалізації спадковості

Розглянемо задачу визначення класу-нащадка на прикладі векторів. З математики відомо, що поряд з визначенням вектора через векторні координати, існує спосіб визначення вектора координатами початкової та кінцевої точки. Розширимо в новому класі TCoordVector можливості класу TMathVector полями, що зберігатимуть координати початкової (FX1, FY1) та кінцевої (FX2, FY2) точок вектора. При цьому необхідно перевизначити методи обробки векторів з урахуванням специфіки створюваного класу.

Поля векторних координат FaX, FaY, що успадковані від батьківського класу, завжди можна визначити через координати початкової та кінцевої точок вектора. Слід зауважити, що будь-яка зміна координат початку чи кінця вектора потребує перерахунку значень векторних координат:

```
type   TCoordVector = class(TMathVector)
private
   FX1, FY1: double;  // координати початкової точки вектора
   FX2, FY2: double;  // координати кінцевої точки вектора
protected

   procedure setX1(x: double);
   procedure setY1(y: double);
   procedure setX2(x: double);
   procedure setY2(y: double);
public
    constructor Create (x1, y1, x2, y2 : double);

   function Calc_aX():double;
   function Calc_aY():double;

   function summationVector(v: TCoordVector): TCoordVector;
   function substractionVector(v: TCoordVector): TCoordVector;
   function scalingVector(k: double): TCoordVector;

   function Info:String;

   property X1: double read FX1 write setX1;
   property Y1: double read FY1 write setY1;

   property X2: double read FX2 write setX2;
   property Y2: double read FY2 write setY2;

   property aX: double read FaX;
   property aY: double read FaY;

end;
```

У новоствореному класі додано нові поля (FX1, FY1, FX2, FY2), методи (setX1, setY1, setX1, setY1, Calc_aX, Calc_aY) та властивості (X1, Y1, X2, Y2), які забезпечують специфіку функціонування класу. Крім того, клас TCoordVector містить свої власні методи додавання (summationVector), віднімання (summationVector) та добутку вектора на число (summationVector). Методи, що реалізують усі інші операції над векторами будуть успадковані від батьківського класу TMathVector. На рис. 1.7 представлено вікно-підказку, що демонструє, які властивості власні та батьківського класу є доступними для об’єктів класу TCoordVector.

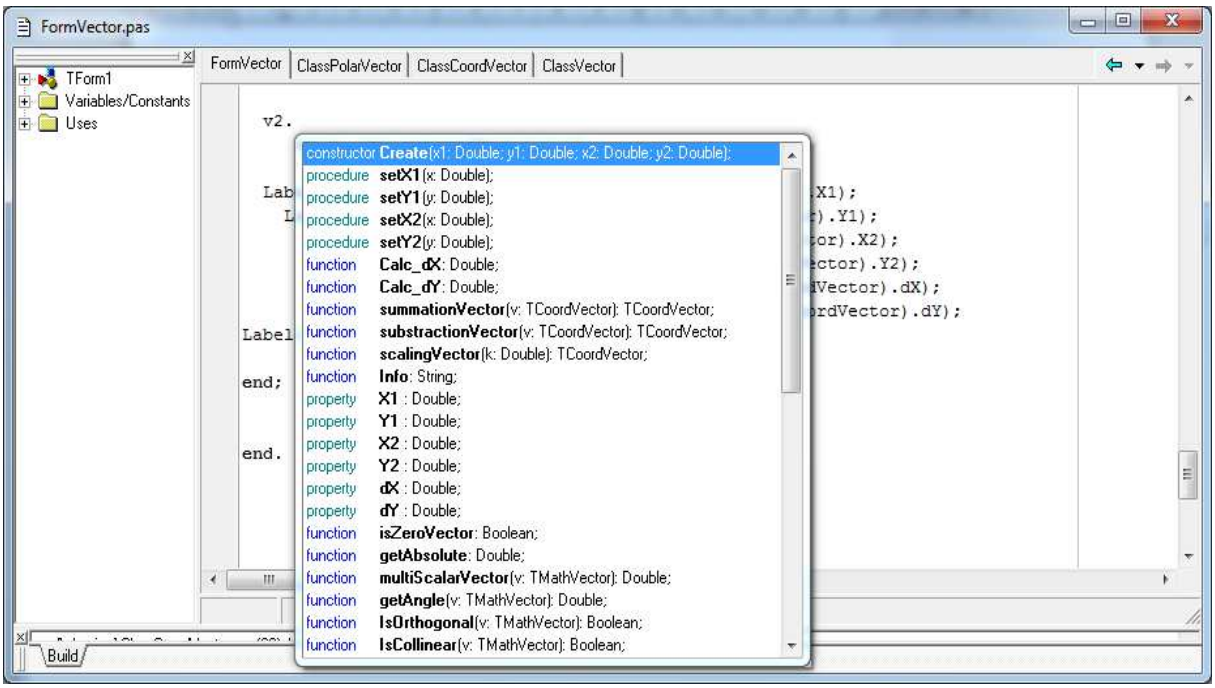


Рис. 1.7. Вікно-підказка, що демонструє доступні поля/методи/властивості об’єктів класу TCoordVector

Слід звернути увагу на властивості aX та aY, що описано в класі TCoordVector. Ці властивості є однойменними батьківських властивостей, але з можливістю тільки читання поля, тому що для TCoordVector запис значення у векторні координати можливий тільки через їх переобчислення. Отже властивості aX та aY, фактично, перекривають для об’єктів класу TCoordVector доступ до відповідних властивостей, що успадковані від предка TMathVector. Про механізм перекриття властивостей предків їх нащадками йтиметься у наступній темі.

Класифікація методів класів

Спадковість дуже зручний механізм, який дозволяє повторно використовувати вже написаний програмний код під час створення нових класів. Але виникає таке питання: що робити в ситуації, коли успадкований метод за своєю функціональність не влаштовує нащадка? Зазвичай можна створити новий метод з новим ім’ям, але це дуже небезпечний шлях, тому що за великої кількості методів і розмаїття імен можна заплутатися. Бажано було б надати можливість класам перевизначити реалізацію методу зі збереженням його імені.

У процесі перевизначення методів можливі три випадки: **перекриття**, **заміщення** та **перевантаження**.

Перекриття методів

Перекриття методів відбувається, коли в класі-нащадку описаний однойменний метод методу предка. Розглянемо приклад:

Опис класів	*Реалізація перекриття методів*

type
TParentClass = **class**
  procedure M;
end;
TSonClass = **class** (TParentClass)
// однойменний до батьківського
  procedure M;
end;
	procedure TParentClass.M;
	begin
	  ShowMessage(‘Hello from Parent!!!’);
	end;
	procedure TSonClass.M;
	begin
	  ShowMessage(‘Hello from Son!!!’);
	end;

Якщо створені екземпляри двох класів: екземпляр класу предка P і екземпляр класу нащадка S, то під час виклику P.M на екрані з’явиться повідомлення ‘Hello from Parent!!!’, а під час виклику S.M – ‘Hello from Son!!!’.

Перекриття методів дуже простий та зручний механізм перевизначення методів, що успадковані від предка, але він має свої недоліки (див. Поліморфізм).

Заміщення методів

Заміщувані методи поділяються на: *віртуальні*, *динамічні* та *абстрактні* (рис. 1.8).

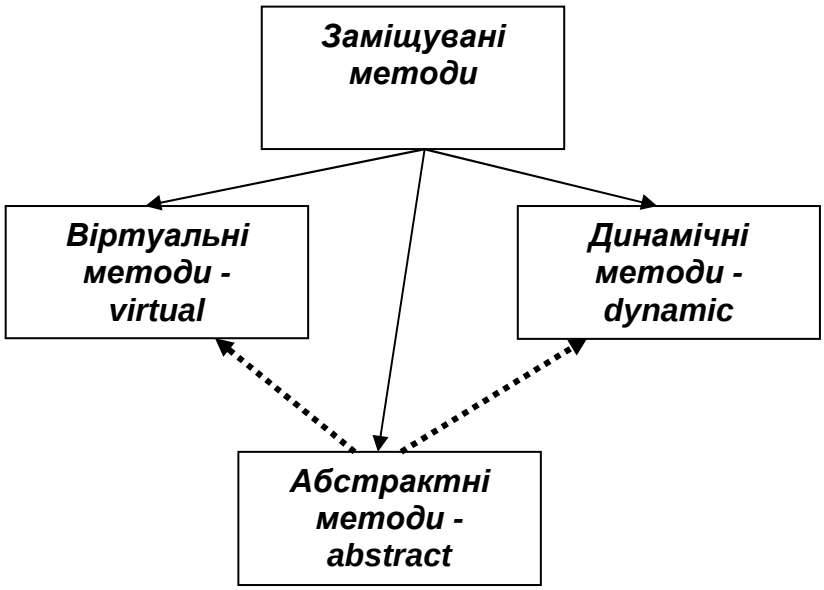


Рис. 1.8. Заміщувані методи класів

Віртуальні та динамічні методи

Віртуальний метод описується з директивою **virtual**, а динамічний – з директивою **dynamic**:

```
type
TParentClass = class
   procedure M;
   procedure VM; virtual;
   procedure DM; dynamic;
   procedure VirtMethod; virtual;
   procedure DynaMethod; dynamic;
end;
```

У класі-нащадку віртуальні та динамічні методи можуть перекриватися або заміщуватися. Під час заміщення методу в класі нащадка використовується директива **override**:

```
TSonClass = class (TParentClass)
   procedure M1;          //перекриття батьківського методу
   procedure VM; virtual;
   procedure DM; dynamic;
   procedure VirtMethod; override; // заміщення батьківського методу
   procedure DynaMethod; override;
end;
```

Реалізація заміщуваних віртуальних і динамічних методів здійснюється аналогічно до перекриття методів класу:

```
{ TParentClass }
procedure TParentClass. VirtMethod;
    begin ShowMessage('Hello from ParentClass virtual method!!!'); end;
procedure TParentClass. DynaMethod;
    begin ShowMessage('Hello from ParentClass dynamic method!!!'); end;
{ TSonClass }
procedure TSonClass. VirtMethod;
    begin ShowMessage('Hello from TSonClass virtual method!!!'); end;
procedure TSonClass. DynaMethod;
    begin ShowMessage('Hello from TSonClass dynamic method!!!'); end;
```

У наведеному прикладі після створення екземплярів для класів предка та нащадка принцип та результат використання заміщуваних та перекритих методів не відрізняється: для кожного з екземплярів буде викликана своя реалізація методів. Але внутрішній механізм роботи перекритих та заміщуваних методів, а також особливості застосування під час розробки ієрархії класів зовсім різні (див. Поліморфізм).

Абстрактні методи

Абстрактні методи описуються директивою abstract та мають бути віртуальними або динамічними (див. рис. 1.8):

```
type
 TParentClass =class
        procedure AbsVirtMethod;     virtual; abstract;
        procedure AbsDynaMethod;    dynamic; abstract;
 end;
```

Директива abstract може використовуватися тільки в класі, де метод оголошується вперше. Її використання робить незаконним визначення тіла методу в даному класі (іншими словами, абстрактний метод не має реалізації в класі). Це унеможливлює виклик такого методу.

Абстрактні методи заміщуються і реалізуються в класах-нащадках:

```
TSonClass = class (TParentClass)
        procedure AbsVirtMethod;     override;
        procedure AbsDynaMethod;    override;
end;
{ TSonClass }
procedure TSonClass. AbsVirtMethod;
    begin ShowMessage('TSonClass virtual non-abstract method!!!'); end;
procedure TSonClass. AbsVirtMethod;
    begin ShowMessage('TSonClass dynamic non-abstract method!!!'); end;
```

> Клас, який має абстрактні методи, називається абстрактним.

Метою введення абстрактних методів є надання родоначальнику ієрархії класів певних властивостей, що дає змогу наділити всіх нащадків спільним набором властивостей, які кожний з нащадків буде реалізовувати по-своєму.

Приклад використання заміщуваних методів та абстрактних класів

Наступним завданням є створення ієрархії класів для опрацювання векторів як у декартовій так і в полярній системі координат. Особливість цієї задачі, з одного боку, полягає в тому, що вектор у полярній системі координат характеризується полярною відстанню та полярним кутом і ніяк не є ані узагальненням, ані деталізацією вектора з декартової системи координат. З іншого боку, над векторами в полярній системі координат можуть виконуватись ті ж операції, що і для векторів з декартової системи координат.

У даному випадку доцільно визначити абстрактний базовий клас TAbstractVector, який міститиме перелік абстрактних методів, обов'язкових до реалізації в обох системах координат. А класи векторів у декартовій системі координат TMathVector та його нащадок TCoordVector, а також новостворюваний клас TPolarVector, будуть породжуватися вже від TAbstractVector. У класах-нащадках TAbstractVector будуть додаватися відповідні поля та буде надана реалізація операцій над векторами у термінах відповідної системи координат. На рис. 1.9 показано узагальнену ієрархію класів з абстрактним класом на найвищому рівні.

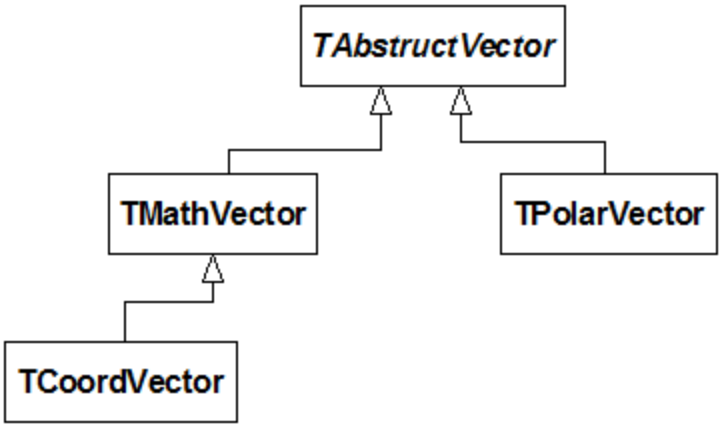


Рис. 1.9. Ієрархія класів задачі «Опрацювання векторів у декартовій та полярній системі координат»

Визначення абстрактного класу має такий вигляд:

```
type
 TAbstractVector=class(TObject)
 public
   function isZeroVector: boolean;  virtual; abstract;
   function getAbsolute: double; virtual; abstract;

   function scalingVector(k: double):TAbstractVector; virtual; abstract;
   function summationVector(v: TAbstractVector):TAbstractVector; virtual;
abstract;
   function substractionVector(v: TAbstractVector):TAbstractVector;virtual;
abstract;
   function multiScalarVector(v: TAbstractVector):double; virtual; abstract;

   function getAngle(v: TAbstractVector):double; virtual; abstract;
   function IsOrthogonal(v: TAbstractVector):boolean; virtual; abstract;
   function IsCollinear (v: TAbstractVector):boolean; virtual; abstract;
   function IsEqual (v: TAbstractVector):boolean; virtual; abstract;
end;
```

Директива inherited

Виникають ситуації, коли з класу-нащадка необхідно викликати метод предка, що був перевизначений (заміщений або перекритий) цим нащадком. Для цього використовується спеціальна директива ***inherited*** (укр. – успадкований). Загальний опис використання цієї директиви такий:

inherited <ім'я методу класа-предка>;

Опис класів	*Реалізація методів*

type
TParentClass = **class**
  procedure M;
end;
TSonClass = **class** (TParentClass)
// перекриття метода
  procedure M;
end;
	procedure TParentClass.M;
	begin
	  ShowMessage('Hello, world!!!');
	end;
	procedure TSonClass.M;
	begin
	  inherited M;
	  ShowMessage('Have a nice day!!!');
	end;

У результаті під час виклику методу M для класу-нащадка буде спочатку викликаний метод предка M. На екран буде виведене повідомлення 'Hello, world!!!', а потім повідомлення 'Have a nice day!!!'.

Приклад використання директиви inherited

Використання директиви inherited розглянемо на прикладі методу знаходження суми векторів, заданих своїми координатами початку і кінця. Задача додавання векторів, заданих своїми координатами, може бути зведена до визначення суми векторів у векторних координатах з подальшим переміщенням результуючого вектора в точку початку вектора, що є першим операндом у дії додавання.

```
function TCoordVector.summationVector(v: TCoordVector):TCoordVector;

 var sumV : TCoordVector;
    sumC    : TMathVector;
begin
  sumC:= inherited summationVector(v); // сума у векторних координатах

  sumV := TCoordVector.Create(0,0,0,0);// створення вектора-результату
  sumV.FX1:=x1; // точка початку результуючого вектора визначається
  sumV.FY1:=y1;// точкою початку вектора - першого операнда

  sumV.FX2:=x1+sumC.dX;// визначення координат точки кінця
  sumV.FY2:=y1+sumC.dY;// результуючого вектора

  sumV.FdX:= sumV.Calc_dX(); //визначення векторних координат
  sumV.FdY:= sumV.Calc_dY();// результуючого вектора
  result := sumV;
end;
```

Перевантаження методів

Перевантажені методи мають *однакове ім'я*, але *різні списки параметрів*, що відрізняються за кількістю параметрів або їх типами.

> Перевантаженими можуть бути не тільки методи класів, а також звичайні підпрограми: процедури або функції, які не є методами класу.

В Delphi Pascal перевантажені методи описуються з директивою **overload**.

Розглянемо приклад опису класу з перевантаженими конструкторами: один конструктор без параметрів, який обнуляє поле, а другий записує в поле задане значення new_f (реалізація цих конструкторів очевидна, тому наводитися не буде):

```
Type
  TAnyClass = class
    f : integer;
    constructor Create; overload;                  // 1-ий конструктор
    constructor Create (new_f : integer); overload;    // 2-ий конструктор
  end;
Var A, B : TAnyClass;
```

Створення екземплярів може бути таким:

```
A := TAnyClass. Create;          // буде викликаний 1-ий конструктор
B := TAnyClass. Create (10);   // буде викликаний 2-ий конструктор
```

Під час перевантаження може виникнути **неоднозначність**, якщо в списку параметрів будуть вказані параметри, що сумісні за типами. Додамо до опису класу TAnyClass ще один третій конструктор:

```
constructor Create (new_f : longint); overload;   // 3-ий конструктор
```

Під час створення екземпляра A : = TAnyClass. Create буде викликаний 1-ий конструктор. Але під час створення екземпляра B : = TAnyClass.Create (10) буде отримане повідомлення про помилку: неоднозначність визначення – 2-ий чи 3-ий конструктор.

> Для уникнення неоднозначності під час перевантаження не рекомендується створювати перевантажені підпрограми, які мають однакову послідовність і кількість параметрів, що сумісні за своїми типами.

Диспетчеризація методів

З курсу програмування відомо, що під час виклику підпрограми визначається адреса оперативної пам'яті, де розташована перша команда цієї підпрограми (точка входу). Здійснюється перехід за визначеною адресою, і розпочинається виконання команд підпрограми. Після завершення підпрограми здійснюється повернення в точку виклику.

Для методів класу також здійснюється визначення адреси методу, команди якого треба виконати під час виклику. Враховуючи те що для ієрархії класів може здійснюватися перевизначення методів у нащадках, внаслідок чого утворюється ланцюжок однойменних методів, виникає задача їх **диспетчеризації**.

Диспетчеризація – механізм визначення адреси для методу екземпляру класу (класу) під час виклику.

Поняття раннього та пізнього зв'язування. Статичні методи

Визначення адреси звичайної підпрограми може відбуватися *статично* під час компонування (лінкування) програми, тобто до початку її роботи, або *динамічно* під час виконання програми, якщо підпрограма викликається з DLL. Для методів класів також характерні два аналогічні способи визначення адрес методів.

Визначення адреси методу на етапі компонування програми називається **раннім зв'язуванням,** а під час виконання програми – **пізнім зв'язуванням**.

Для звичайних методів класу, перекритих та перевантажених методів притаманне раннє зв'язування. Методи класів, для яких здійснюється раннє зв'язування, називаються в Delphi *статичними*.

Для заміщуваних, що перевизначені з директивою **override**, – пізнє.

Механізм диспетчеризації методів класу ґрунтується на таких складових:

- кожен клас містить по три таблиці, в яких знаходяться адреси методів: таблицю для статичних, віртуальних (ТВМ) та динамічних методів (ТДМ);
- кожен клас містить покажчик на клас-предок. Це дозволяє здійснювати пошук адрес методів, що успадковані від предка (див. «Диспетчеризація успадкованих методів»).

Таблиця статичних методів класу містить адреси тільки тих

методів, які введені вперше в даному класі або перекриті в ньому.

Таблиця динамічних методів (ТДМ) містить адреси тільки тих методів, які введені вперше в даному класі або перевизначені (перекриті або заміщені).

Таблиця віртуальних методів (ТВМ) містить адреси *всіх* віртуальних методів класу, незалежно від того успадковані ці методи від предка чи перевизначені в даному класі.

Найшвидше під час виконання програми спрацьовує виклик статичного методу, тому що адреса методу на момент виконання вже визначена раннім зв'язуванням.

Для заміщуваних методів на етапі компіляції в текст програми замість адреси методу підставляється програмний код, який звертається до ТВМ або ТДМ класу. Цей код «спрацює» під час виконання програми (пізнє зв'язування), що трошки уповільнює її роботу. При цьому виклик віртуальних методів здійснюватиметься швидше за виклик динамічних у разі, якщо динамічний метод не перевизначений в даному класі, а спадкується від предка, внаслідок чого його адреси немає в ТДМ даного класу (див. «Диспетчеризація успадкованих методів»). Усі адреси віртуальних методів класу завжди «під рукою» в одній таблиці, але це вимагає більшого об'єму пам'яті для ТВМ.

Диспетчеризація успадкованих методів

Механізм диспетчеризації (виклику) методу, який був успадкований від предків, такий:

- відбувається пошук адреси методу у власній таблиці;
- якщо адресу методу не знайдено через посилання на предка, що міститься в класі, відбувається пошук адреси відповідного методу в таблиці предка;
- якщо не знайдено – пошук в таблицях предків вище за ієрархією (до рівня спільного для всіх предка – класу TObject);
- якщо не знайдено – виведення повідомлення про помилку.

Необхідно ще раз зауважити, що пошук адреси статичного методу відбувається до виконання програми – раннім зв'язуванням, а пошук динамічного – під час виконання програми – пізнім зв'язуванням.

Доцільність використання статичних та заміщуваних методів

Припустимо, що ми маємо справу з деякою сукупністю класів, що утворюють ієрархію спадковості. Ці класи мають набір спільних методів, що передаються від предків нащадкам.

Ті з методів, що не змінюватимуть свого змісту, необхідно реалізувати як статичні методи. Ті, що змінюються у процесі переходу від предка до нащадка, краще реалізувати як заміщувані гнучкіші методи. При цьому методи, що є спільними для всіх класів з ієрархії (основні, родові методи), необхідно описувати в класі-предку, можливо як абстрактні, і потім перевизначити їх в класах нащадках [3; 4].

Виникає запитання: а який з видів заміщуваних методів обирати: віртуальний чи динамічний? Проаналізуємо їх переваги та недоліки.

Диспетчеризація віртуальних методів відбувається швидше ніж динамічних, але для зберігання ТДМ необхідно більше пам'яті. Диспетчеризація динамічних відбувається трошки повільніше за віртуальні у разі звернення до батьківських таблиць, якщо динамічний метод не перевизначений в нащадку. З іншого боку, перевизначення всіх динамічних методів в нащадку потребуватиме не менше пам'яті ніж для віртуальних.

Отже можна сформулювати такі правила [6]:

- якщо метод, скоріше за все, буде перевизначений майже всіма нащадками, краще його зробити віртуальним;
- якщо метод викликатиметься дуже часто, багато разів на секунду, краще зробити його віртуальним;
- якщо для класу планується значна кількість нащадків, а метод, що вимагає пізнього зв'язування, буде перекриватися і використовуватися не дуже часто, краще зробити його динамічним.

В інших випадках відмінність у швидкості виклику віртуальних та динамічних методів не має принципового значення.

Поліморфізм

У попередніх темах дуже багато йшлося про значущість пізнього зв'язування для заміщуваних методів. Для наочної демонстрації цього розглянемо приклад.

Описано клас-предок, що призначений для обробки значення певного типу. Конкретизація типу значення відбувається в нащадках цього класу. На рівні класу-предка введений родовий метод GetData, що повертає значення поля у форматі рядка:

```
type
TField = class
  function GetData:string; virtual; abstract;
end;

TIntegerField = class (TField)
  FData : Integer;
  function GetData: string;override;
end;

TRealField = class (TField)
  FData : real;
  function GetData: string; override;
end;

function TIntegerField.GetData;
 begin   Result := IntToStr(FData);  end;

function TRealField.GetData;
 begin   Result := FloatToStr(FData);   end;

procedure ShowData (AField : TField);
 begin   ShowMessage( AField.GetData );  end;

var   I : TIntegerField;   R: TRealField;
```

Описана процедура ShowData, яка приймає як параметр екземпляр класу-предка і викликає метод GetData. Оголошені екземпляри класів TIntegerField та TRealField. Після створення екземплярів I та R можна здійснити такі виклики:

ShowData(I); ShowData(R);

Але для якого класу буде фактично викликаний метод GetData? Формальний параметр процедури ShowData – екземпляр абстрактного класу TField, який взагалі не має реалізації методу GetData, а під час виклику в процедуру ShowData передаються екземпляри нащадків цього абстрактного класу. Тут вступають в силу правила контролю відповідності типів (typecasting).

Об’єкту як покажчику на екземпляр класу може бути присвоєна адреса будь-якого іншого об’єкта, що є покажчиком на екземпляр дочірнього класу (правило спадковості: замість екземпляру батьківського класу може бути використаний його нащадок).

Отже під час виклику ShowData(I) формальному параметру AField буде присвоєна адреса фактичного параметра I, який є екземпляром дочірнього класу TIntegerField. А під час виклику ShowData(R) формальному параметру AField буде присвоєна адреса фактичного параметра R, який є також екземпляром дочірнього класу TRealField. Тому в результаті першого виклику на екран буде виведене ціле значення, а під час другого – дійсне.

Настроювання екземпляра класу на адреси заміщуваних методів відбувається під час створення екземпляра в конструкторі. Іншими словами: в конструкторі, окрім виділення пам’яті та ініціалізації полів, настроюються таблиці віртуальних і динамічних методів для екземпляра класу.

Можливість створювати ієрархії класів зі спільним набором властивостей, можливість змінювати в нащадках реалізацію спільних властивостей, а також можливість предкам звертатися до методів, що були заміщені в нащадку називається поліморфізмом.

Поліморфізм – базовий принцип ООП згідно з яким:
- реалізується ідея спільних методів для різних класів об’єктів, кожний з яких реалізує ці методи по-своєму;
- батьківські класи (класи-предки) можуть звертатися до методів, що заміщені у нащадків.

Можна дещо ускладнити попередній приклад:

```
type
TField = class
  function GetData:string;  virtual; abstract;
  procedure ShowData;
end;

TIntegerField = class(TField)
  FData : Integer;
```

```
  function GetData: string; override;
end;

TRealField = class(TField)
  FData : real;
  function GetData: string; override;
end;
```

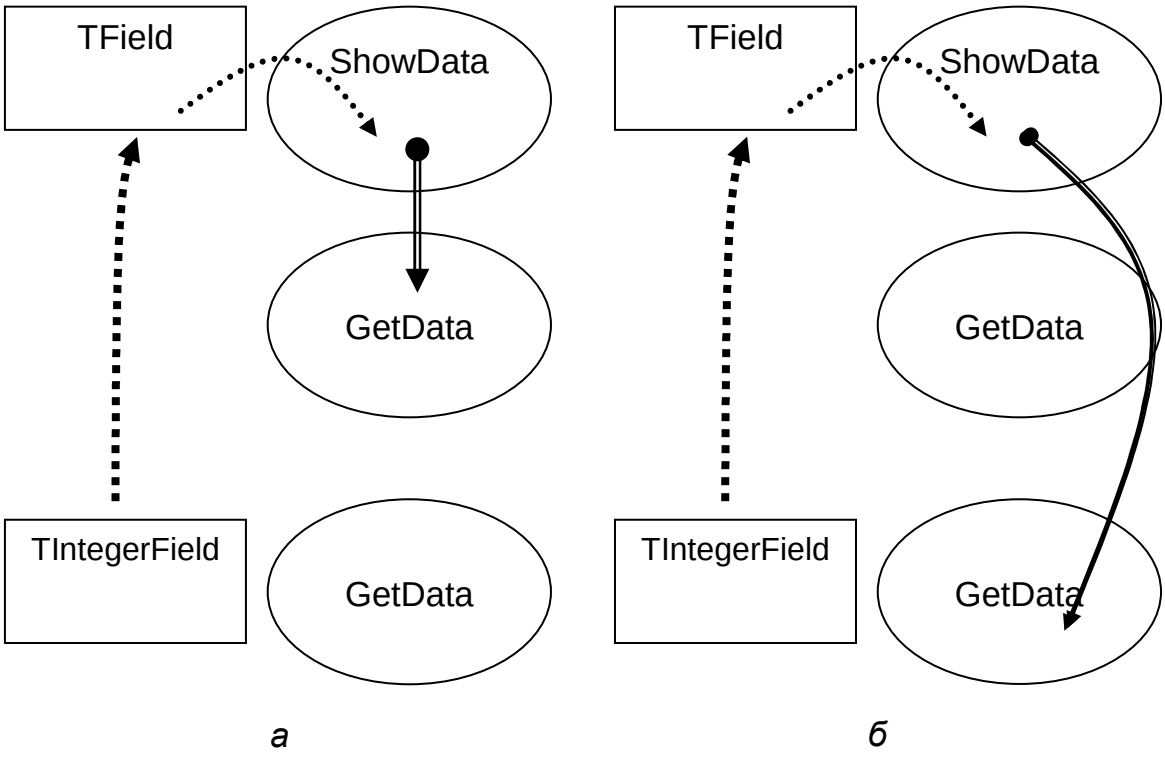


Рис. 1.10. Схеми виклику перекритих *а*
та заміщуваних *б* методів об’єктів:

```
function TIntegerField.GetData;
 begin   Result := IntToStr(FData);  end;

function TRealField.GetData;
 begin   Result := FloatToStr(FData);   end;

procedure TField.ShowData;
begin  Showessage( GetData ); end;
```

Складність цього прикладу полягає в тому, що заміщуваний метод GetData викликається зі статичного методу предка ShowData. Якщо згадати диспетчеризацію успадкованих методів для виклику ShowData необхідно звернутися до класу-предка, тобто піднятися на рівень вище за ієрархію класів. Якби метод GetData не був заміщуваний, а був би перекритий, після виходу на рівень предка здійсниться виклик методу GetData предка (див. рис. 1.10, а). На відміну від цього, для заміщуваних методів буде можливий виклик зі статичного методу предка методу, що був заміщений у нащадку (див. рис. 1.10, б).

Оператори is та as

Маючи справу з поліморфною ієрархією класів, часто виникає необхідність визначити тип екземпляра (його приналежність до класу) [7]. В Delphi Pascal використовуються дві операції, які дозволяють дещо приборкати складність роботи зі «спрутом» класів.

Перша – оператор ***is,*** що забезпечує перевірку того, чи є тип об'єкта типом даного класу, або одним з його нащадків [7]. Друга – оператор ***as*** виконує явне приведення типу об’єкта до даного класу.

екземпляр_класу ***is*** *клас*
is – логічна операція з результатом бульового типу, що набуває значення true, якщо об’єкт є екземпляром даного класу або екземпляром одного з класів-нащадків.

Наприклад:

```
function Division (AField : TField) : string;
begin
  If  Afield is TIntegerField Then Result := IntToStr (AField.FData div 100)
 else
  If  Afield is TRealField Then Result := FloatToStr (AField.FData / 100)
 else Result := ‘В класі TField немає поля!’;
end;
```

У прикладі перевіряється, чи є параметр AField функції Division екземпляром одного з дочірніх класів. Якщо – так, тоді чисельні поля є в складі класу, і відбуваються обчислення відповідно до типу поля. Якщо параметр не є дочірнім класом, виводиться відповідне повідомлення.

екземпляр_класу ***as*** *клас*

as операція приведення класу лівого операнда до класу, що заданий правим операндом. Якщо класи несумісні (правий операнд не є дочірнім класом для лівого операнда) виникає виключна ситуація EInvalidCast.

Фактично оператор ***as*** виконує явне зведення типів. Для класів напис:

клас (екземпляр_класу)

аналогічний напису:

екземпляр_класу ***as*** *клас*,

але на відміну від простого зведення типів оператора ***as*** на початку перевіряє, чи сумісні ці типи під час виконання. Після застосування операції ***as*** сам об'єкт не змінюється, але викликаються ті його методи, якби він належав до зведеного класу.

Висновки

- Об’єктно-орієнтований підхід (ООП) до розробки програм з’явився у 80-х роках минулого сторіччя як відповідь на кризу програмного забезпечення.
- Основними принципами ООП є інкапсуляція, спадковість, поліморфізм, абстрагування.
- В ООП програма складається з об’єктів, які є відображенням (програмною моделлю) реальних об’єктів предметної області.
- Кожний об’єкт характеризується властивостями (розмір, колір, смак тощо) та діями (рухається, стоїть, телефонує тощо).
- Кожен об’єкт має свій стан, який визначається поточними значеннями всіх його визначених властивостей (полів, атрибутів). Зміна значення поля (декількох полів) переводить об’єкт в інший стан.
- За збігом основних ознак можна виділити групи споріднених об’єктів – класи. Клас – це сукупність об’єктів одного типу, тобто схожих за своїми властивостями (атрибутами) та призначенням (поведінкою). З точки зору програмної реалізації, клас – це сукупність полів даних та методів, призначених для обробки цих даних.
- Кожен програмний об’єкт є екземпляром (представником) певного класу, а класи можуть утворювати ієрархію.
- Всі об’єкти Delphi є динамічними. Для створення та знищення

екземплярів класу використовуються спеціальні методи класів – конструктор та деструктор.

- Не можна звертатися до полів та методів екземпляра класу до його створення або після знищення, окрім методів які описані як класові.
- Класи та об’єкти не існують автономно – вони взаємодіють між собою. Найпоширенішими під час опису взаємодії між класами є такі три типи зв'язків: асоціація, узагальнення та залежність. Асоціаційний зв’язок може мати дві особливі форми: агрегація та композиція.
- Директиви видимості private, protected та public дозволяють на практиці реалізувати принцип інкапсуляції. За замовчуванням діє директива public.
- Для забезпечення контрольованого доступу до захищених полів класу необхідно описувати спеціальні методи для читання та запису значення поля.
- Методи читання та запису під час опису класу можуть бути об’єднані в спеціальну синтаксичну конструкцію Delphi – властивість (property).
- Властивості за способом організації доступу до поля можуть бути: тільки для читання, тільки для запису або для читання та запису одночасно. За формою властивості бувають: проста, властивість-масив, індексована.
- Класи можуть утворювати ієрархію спадковості. Розрізняють одиничну та множинну форму спадковості. В Delphi множинна спадковість на пряму не реалізується.
- Класи-нащадки успадковують усі властивості предка та можуть їх розширювати. Це надає змогу в програмі замість екземпляра класу предка використовувати екземпляр класу-нащадка.
- Перевизначення методів дає змогу нащадкам змінювати реалізацію методів, що успадковані від предка. Під час перевизначення методів можливі три випадки: перекриття, заміщення, перевантаження методів.
- Заміщувані методи поділяють на: віртуальні, динамічні та абстрактні. Заміщувані методи перевизначаються директивою override. Перевизначення заміщуваних методів з директивами virtual або dynamic перетворює заміщення на перекриття.
- Абстрактні методи мають бути віртуальними або динамічними і повинні обов’язково заміщуватися в нащадках класу. Клас, у складі якого є абстрактні методи називається абстрактним.

- Директива inherited дозволяє нащадкам викликати методи предків, які були цим нащадком перевизначені.
- Для уникнення неоднозначності під час перевантаження не рекомендується створювати перевантажені підпрограми, які мають однакову послідовність і кількість параметрів, що сумісні за своїми типами.
- Диспетчеризація – механізм визначення адреси для методу екземпляра класу або класу під час виклику.
- Визначення адреси методу на етапі компонування програми називається раннім зв’язуванням, а під час виконання програми – пізнім зв’язуванням.
- Для статичних методів класу, перекритих та перевантажених методів притаманне раннє зв’язування, для заміщуваних – пізнє. Настроювання екземпляра класу на адреси заміщуваних методів відбувається під час створення екземпляра в конструкторі.
- Використання заміщуваних методів дає змогу з класів-предків викликати методи, що були заміщені в класах-нащадках.

Запитання та завдання для самоперевірки

1. У чому полягає об’єктно-орієнтований підхід до створення програм?
2. Що таке предметна область і як програмні об’єкти пов’язані з об’єктами предметної області?
3. Які основні принципи об’єктно-орієнтованого підходу?
4. Дайте визначення класу та об’єкта (екземпляра класу).
5. Який синтаксис опису класів в Delphi? Як задається реалізація методів класу?
6. Дайте визначення поля та методу класу. Наведіть приклад опису класу з полями та методами.
7. Як оголошується та створюється екземпляр класу?
8. Для чого використовуються конструктор та деструктор? Як вони описуються в класі?
9. Як можна знищити екземпляр класу? В чому особливості методу *Free*?
10. Чи можна використовувати поля та методи класу без створення екземпляра класу?
11. Що таке класові методи? В чому полягають особливості їх використання?

12. Які типи відношень є між класами та об’єктами?
13. Яке відношення називається асоціацією? Наведіть приклади об’єктів, які пов’язані цим відношенням.
14. Що спільного і в чому різниця між відношеннями агрегація та композиція? Наведіть приклади.
15. Яке відношення між класами визначає узагальнення? Наведіть приклад.
16. В чому полягає відношення залежності? Наведіть приклад.
17. В чому полягає принцип інкапсуляції? Доведіть на прикладі доцільність його застосування.
18. Які директиви видимості використовуються під час опису класів? Дайте характеристику кожної директиви.
19. Як організувати доступ до захищених полів класу?
20. Що таке властивість (property)? Який синтаксис її опису?
21. Як повинен бути описаний метод читання та запису для властивості?
22. Чи може тип властивості відрізнятися від типу поля, з яким вона пов’язана? Наведіть приклад.
23. Коли доцільно використовувати властивість тільки для читання чи тільки для запису?
24. Яких правил необхідно дотримуватися під час використання властивостей?
25. У яких випадках використовується і як описується властивість-масив? Як організувати властивість для доступу до багатовимірного масиву?
26. У чому особливості опису та використання індексованої властивості? Що спільного і в чому різниця між властивістю масивом та індексованою властивістю?
27. В чому полягає принцип спадковості? Що таке одинична та множинна спадковість?
28. Які класи називаються абстрактними? В чому доцільність їх використання?
29. Чи можуть предки використовувати властивості своїх нащадків? Чи можуть нащадки використовувати властивості, що успадковані від предків?
30. Який клас є спільним предком за замовчуванням для всіх класів Delphi?
31. Чи можуть нащадки змінювати властивості, що успадковані від предків?

32. Як здійснюється перекриття методів. Як викликати метод предка, що був перекритий в нащадку?
33. Яки види заміщуваних методів? З якою директивою нащадок має перевизначити метод, щоби він став заміщуваним?
34. Доведіть доцільність використання абстрактних методів. Який синтаксис їх опису в Delphi?
35. Як утворюються перевантажені методи? Коли може виникнути неоднозначність під час роботи з перевантаженими методами?
36. Поясніть поняття диспетчеризації методів. У чому полягає раннє та пізнє зв'язування?
37. Як реалізується диспетчеризація статичних та заміщуваних методів, а також методів, що успадковані?
38. В чому переваги та недоліки застосування статичних та заміщуваних, динамічних та віртуальних методів?
39. В чому полягає принцип поліморфізму? Поясніть на прикладі переваги заміщення методів над перекриттям.
40. Поясніть використання операторів *is ma as.*

Завдання для практичної реалізації

1. Описати клас для кола заданого радіуса та координатами центра. Описати методи для визначення довжини та площі кола.
2. Описати клас для рядка, який вміє визначати свою довжину та підраховувати кількість слів у цьому рядку.
3. Описати клас для оцінки з певної дисципліни, що містить поля «назва дисципліни» та «оцінка в 100-бальній шкалі», і може формувати значення оцінки в національній шкалі або ECTS.
4. Описати клас для комплексного числа. Визначити, які поля необхідні для роботи цього класу. Створити методи для порівняння, додавання та присвоєння значення двох комплексних чисел.
5. Описати клас для точки, яка визначається координатами (x, y). Створити масив не більше ніж з К точок. Визначити довжину ламаної, вершини якої записані в цьому масиві.
6. Описати клас для студентів денного відділення. При цьому необхідно враховувати, що студентом денного відділення може бути лише людина, що не досягла віку 35-ти років. Створити масив не менш як з К студентів.

7. Описати клас для масиву, що зберігає впорядковані за зростанням числові значення. Описати властивість для запису значень в поле-масив при цьому не порушуючи впорядкованість послідовності.
8. Описати клас для масиву, що зберігає виключно прості числа. Описати властивість для запису значень в масив, якщо під час запису визначається, що число не є простим, виводиться відповідне повідомлення.
9. Описати клас, що містить масив рядків з інформацією про особу: прізвище, ім’я, по батькові, місце народження, стать (кожне з цих значень зберігається в окремому рядку). Описати індексовану властивість для доступу до полів. Створити методи для виведення інформації про особу, пошуку особи за різними ознаками та метод, що виводить прізвище та ініціали особи.
10. Описати ієрархію класів для прямокутника та паралелепіпеда з заміщенням методів для обчислення площі (площі поверхні) та об’єму.
11. Описати ієрархію класів для натурального та комплексного числа з заміщенням методів введення/ виведення даних.
12. Описати клас для прямокутного трикутника. Передбачити, що створення екземпляра класу може здійснюватися під час введення даних в діалоговому режимі або з файлу.
13. Визначити об’єктну модель для транспортних засобів: самокат, велосипед, мотоцикл, автомобіль. Визначте спільні характеристики та особливості кожного з цих класів.
14. Визначити об’єктну модель для споруджень на дачній ділянці житлового, побутового та господарського призначення; приміщень дачного будинку; будівельної конструкції дачного будинку (фундамент, огороджувальні конструкції тощо).
15. Визначити об’єктну модель, що реалізує закріплення студентів (факультет, прізвище, курс) за гуртками в університеті. Гуртки можуть бути: з профілюючих дисциплін та спортивні секції. Для кожного гуртка задається назва, день тижня, в який проводиться гурток, час початку, місце проведення.

Розділ 2.
КОМПОНЕНТНА МОДЕЛЬ DELPHI

Застосування принципів ООП у компонентній моделі VCL Dephi

Згідно з одним із основних положень теорії об'єктно-орієнтованого програмування – спадкування, класи утворюють ієрархію. В середовищі Delphi спільним предком для всіх класів є клас TObject (рис. 2.1). Від нього відходять п'ять основних (базових) класів, від яких породжуються всі інші. В класі TObject введені основні методи, без яких неможливе використання жодного об'єкта (опис властивостей класу наведено у [3; 4; 8]).

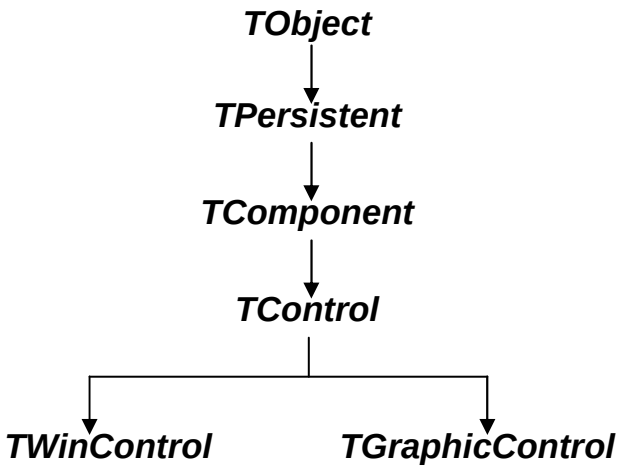


Рис. 2.1. Базові класи компонентів Delphi

Якщо проводити аналогію з будовою людини, перші п'ять класів є скелетом, на якому тримаються м'язи та всі інші органи.

Всі компоненти підрозділяються на декілька основних групи (рис. 2.2):

♦ візуальні компоненти або елементи управління – всі класи-нащадки TControl:

◊ віконні елементи управління – нащадки класу TWinControl;

◊ графічні елементи управління або невіконні елементи управління – нащадки класу TGraphicControl.

♦ невізуальні компоненти – нащадки класу TComponent – це всі компоненти, що невидимі під час виконання застосування в клієнтській області вікна, або відображаються не так як під час проектування форми.

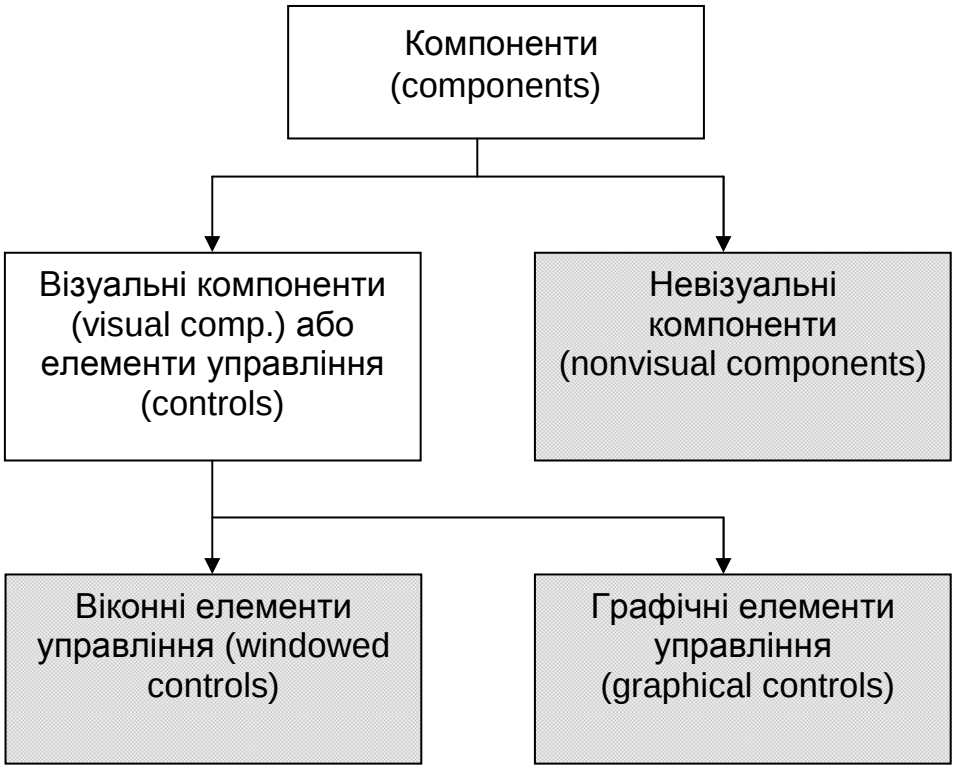


Рис. 2.2. Класифікація компонентів Delphi

Клас TPersistent

TPersistent – абстрактний клас сталих об’єктів, тобто об’єктів, що можуть зберігатися або завантажуватися з потоку, де під потоком розуміємо носій пам’яті: жорсткий диск або оперативну пам’ять.

Найголовнішим надбанням класу TPersistent є метод Assign, який дозволяє копіювати дані об’єктів або присвоєння одного сталого об’єкта іншому:

procedure Assign (Sourse: TPersistent);

Наприклад, Obj2.Assign(Obj1) виконає присвоєння для всіх полів та властивостей Obj2 значення відповідних полів та властивостей Obj1. Якщо вказані об’єкти не сумісні за типом, збуджується виключна ситуація EConvertError.

Треба мати на увазі, що використання операції присвоєння для об’єктів не буде еквівалентним виклику Assign. Об’єкти в Delphi динамічні, тобто Obj1 та Obj2 є покажчики на об’єкти. Таким чином вираз:

Obj2:= Obj1;

встановить Obj2 покажчиком на область пам’яті, на яку посилається Obj1.

Клас TComponent

Як вже було сказано вище, клас TComponent є класом-предком для всіх компонентів VCL. В цьому класі вводяться декілька властивостей і методів, спільних для всіх компонентів.

По-перше, це ім’я компонента Name: TComponentName – рядок до 63 символів довжиною. Ім’я компонента не може бути пустим рядком, в одному модулі не може бути двох компонентів з однаковим ім’ям, воно використовується для доступу до його полів, властивостей, обробників подій і методів.

Кожний компонент має властивість Tag: Integer, яка може використовуватися програмістом на свій розсуд.

Кожен компонент у програмному застосуванні має свого власника (Owner) і сам може бути власником інших компонентів. Ім’я власника для кожного компонента задається у процесі його утворення, тому змінюється визначення конструктора для компонента:

constructor Create (AOwner: TComponent);

Власник компонента відповідає за його створення та руйнування. Під час створення або руйнування власника автоматично створюються або руйнуються всі компоненти, власником яких він є. Наприклад, якщо на формі розташовані декілька компонентів, не треба клопотати про виклик їх конструкторів і деструкторів у програмі. Завдяки чому це можливе?

Всі компоненти, що належать даному заносяться у спеціальний список (масив), який доступний через властивість Components:

Components [Index]:TComponent;

Елементи цього масиву – покажчики на компоненти-власності, індекси елементів починаються з 0. Загальна кількість компонентів у списку Components задається властивістю ComponentCount.

Таким чином, під час створення компонента-власника з його конструктора буде викликано всі конструктори компонентів, що увійшли до масиву Components, а під час руйнування з його деструктора також будуть викликані всі деструктори підлеглих компонентів.

Кожний компонент “знає” свого власника, посилання на якого зберігається властивістю Owner: TComponent, і свій індекс в його списку ComponentIndex: Integer.

Клас TControl

Клас TControl – безпосередній нащадок класу TComponent. Це клас елементів управління Windows.

Особливості класу TControl такі:

1) в ньому вводяться основні обробники подій (див. «Програмування, що орієнноване на події»);
2) в ньому вводяться властивості, що відповідають за розмір, розташування, видимість та активність елементів управління.

Формально елемент управління Windows – це стандартне вікно, спроможне реагувати на спеціальне повідомлення, для якого задані певна поведінка і множина атрибутів. У Windows до стандартних елементів управління відносяться [6]:

- стандартні кнопки (button), кнопки-прапорці (check-box), кнопки-перемикачі (radio-button);
- статичні мітки (static label);
- поля введення (edit);
- списки (list box) та комбіновані списки (combo-box);
- стрічки прокрутки (scrollbar);
- рядок стану (status bar);
- лічильник (spin button);
- індикатор виконання (progress bar);
- набір вкладок (tab control) та ін.

Стандартні системні елементи управління – основні складові будь-якого програмного застосування. Але в бібліотеці є багато елементів управління, які не використовуються в ОС Windows і спадкуються від класу TCustomControl.

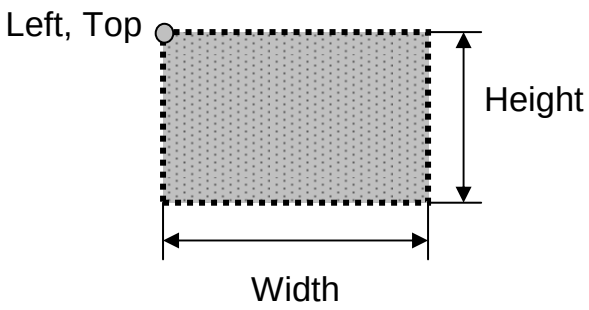


Рис. 2.3. Координати та розмір елемента управління

Розмір та розташування кожного елемента управління визначаються прямокутною областю, яка задається координатами точки прив’язки, довжиною та висотою (рис. 2.3). Відповідно до цього кожен елемент управління має такі властивості:

Left : Integer;

Top : Integer;

Width : Integer;

Height : Integer;

Але для виведення інших елементів управління використовується не вся площа поверхні елемента. Розміри робочої або клієнтської області задаються властивостями:

ClientWith: Integer;

ClientHeight : Integer;

Наприклад, для форми у клієнтську область не включаються заголовок, рядок меню та рамка (border).

Дуже важливу роль, коли задано розміри та розташування має властивість, яка визначає вирівнювання елементу управління відносно його батьківського елемента:

Align : TAlign;

TAlign = (alNone, alTop, alBottom, alLeft, alRight, alClient);

При цьому:

alNone - вирівнювання не здійснюється;

alTop - елемент прижатий до верхнього краю батьківської області,

alBottom - елемент прижатий до нижнього краю батьківської області,

alLeft - елемент прижатий до лівого краю батьківської області;

alRight - елемент прижатий до правого краю батьківської області;

alClient - елемент займає всю батьківську клієнтську область.

Видимість елемента управління визначається властивістю:

Visible : Boolean;

Коли значення Visible := true, то елемент буде видимим на екрані, якщо всі його батьківські елементи також видимі.

Активність елемента управління, тобто можливість реагувати на сигнали від клавіатури та миші, встановлюється властивістю:

Enabled : Boolean;

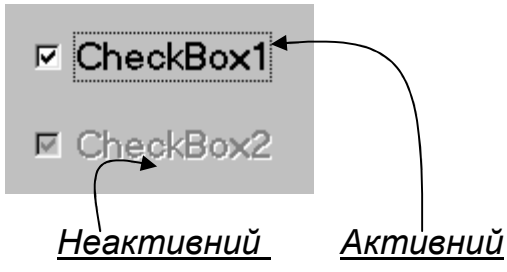


Рис. 2.4. Активність елементів управління

Якщо значення Enabled:= true (і всі батьківськи елементи теж мають Enabled:= true), то елемент буде доступним, інакше він виводиться сірим кольором і не реагує на події (рис. 2.4).

Елемент управління може мати заголовок Caption: TCaption, який виводиться на елементі (наприклад, на рис. 2.4 заголовок першої кнопки-прапорця – CheckBox1), та зберігати якийсь текст у властивості Text:TCaption (тип TCaption = string).

Тип, розмір, колір та стиль шрифту для виведення заголовка визначається складеною властивістю Font : TFont.

Колір елемента управління задається властивістю Color : TColor.

Під час розробки програмного застосування бажано дотримуватись єдиного стилю елементів управління. Тому для всіх елементів управління задаються властивості для спадкування стилю від батьківського елемента:

ParentColor : Boolean;

ParentFont : Boolean;

ParentShowHint : Boolean;

ParentCtl3D : Boolean;

Кожен елемент управління може задавати для себе форму покажчика миші. Це визначається властивістю Cursor : TCursor. Тип курсора задається цілочисельною константою (TCursor = -32768..32767). В Delphi підтримуються стандартні курсори: crDefault, crNone, crArrow, crCross, crIBeam та інші – форму яких можна переглянути і задати через ObjectInspector.

Кожен елемент управління може мати ярлик-підказку, який з’являється, коли покажчик миші просувається над елементом. Текст підказки задається властивістю Hint : string, можливість відображення підказки визначається властивістю ShowHint : Boolean.

Більшість елементів управління може мати своє випливне меню, ім’я якого зберігається у властивості PopupMenu : TPopupMenu.

Клас TWinControl

Клас TWinControl – це предок усіх віконних елементів управління. Віконний елемент управління відрізняється від простого елемента управління трьома рисами:

1. Кожен віконний елемент управління в операційній системі Windows отримує дескриптор (Handle: HWnd) вікна – унікальний ідентифікатор, який використовується ОС для управління цим елементом. Фактично, з точки зору ОС ці елементи є вікнами, хоча ззовні на них не схожі;

2. Можуть бути батьківськими елементами для інших. Батьківський елемент управління – це віконний елемент управління, що містить у собі

інші елементи, які називаються дочірніми. Наприклад, якщо в GroupBox розташовані дві кнопки Button, то GroupBox для них батьківський елемент, а кнопки – дочірні елементи.

3. Отримують фокус введення – аналог курсора, він позначає на формі активний елемент, який на цей момент отримує та обробляє всі вхідні повідомлення. На рис. 2.5 Button1 має фокус введення (обведена прямокутником).

Як було сказано у попередніх розділах, компоненти можуть мають власника (Owner) та батьківський компонент (Parent). Власник відповідає за створення та руйнування елемента. Батьківський елемент відповідає за відображення, розташування, розмір, стиль, видимість та активність дочірнього елемента управління.

Всі дочірні елементи заносяться у список, який доступний через властивість Controls:

Controls [Index: Integer]:TControl;

Елементи цього масиву – покажчики на дочірні компоненти, індекси елементів починаються з 0. Загальна кількість елементів у списку Controls задається властивістю ControlCount.

Таким чином після відображення у вікні батьківського компонента, будуть відображені всі видимі елементи управління, що увійшли до масиву Controls.

На рисунку 2.5 на формі виведено 5 елементів: GroupBox1, Button1, Button2, Label1, Label2. Власником усіх цих компонентів буде форма, до її списку Components буде занесено 5 компонентів. Але дочірніх елементів у форми всього 3: GroupBox1, Button2, Label2.

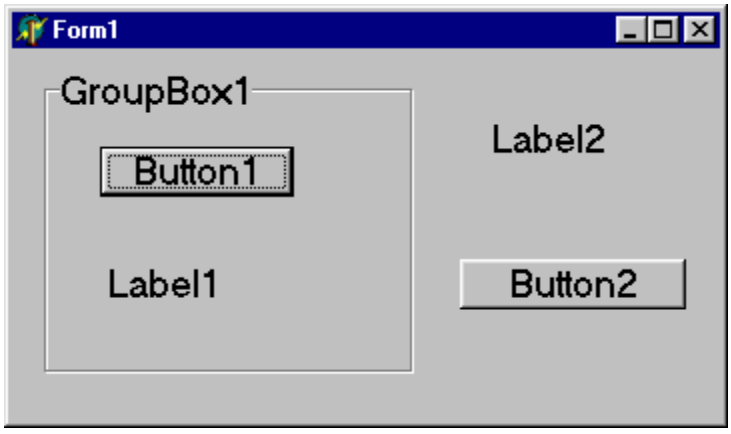


Рис. 2.5. Приклад “Батьки і власники”

Сам по собі GroupBox1 є батьківським елементом для Button1 та Label1, тому що вони розташовані на його поверхні. Для GroupBox1 список Components буде пустий, а список Controls містить 2 елементи: Button1 та Label1.

Управління фокусом введення здійснюється трьома методами:

- function Focused : boolean перевіряє, чи має елемент зараз фокус введення;
- function CanFocus : boolean; перевіряє, чи може взагалі отримати елемент фокус введення;
- procedure SetFocus; встановлює фокус введення на елемент, що викликав цей метод.

Користувач під час роботи програмного застосування може пересувати фокус введення за елементами управління, натискуючи клавішу <Tab>. Порядок обходу елементів встановлюється в батьківському списку TabOrderList. Його можна переглянути під час проектування форми у випливаючому меню кожного батьківського елемента або викликавши метод GetTabOrderList.

Кожен віконний елемент управління, в свою чергу, отримує від батьківського елемента свій номер у списку TabOrderList, який стає доступним через властивість TabOrder: Integer. Якщо встановити значення TabOrder= -1, то цей елемент буде виключений зі списку обходу.

Програмування, що орієнноване на події

Подіє-орієнтоване програмування (англ. event-driven programming) – парадигма програмування, в якій виконання програми визначається подіями: діями користувача (клавіатура, миша), надходження сигналів від апаратури, повідомленнями інших програм, операційної системи або деяких об’єктів самої програми.

Архітектура подіє-орієнтованого програмного забезпечення

Типова сучасна програма – це програма, що керується подіями. Звістка про настання події – це повідомлення, що генерується операційною системою. Це повідомлення одержується програмою через спеціальну функцію вікна програми – диспетчера повідомлень (рис. 2.6). Повідомлень є сотні (але їх перелік фіксований), тому однією з задач програміста під час розробки програми є визначення необхідних повідомлень і опис у складі програми спеціальних підпрограм, в яких задаються реакції на них. Такі підпрограми називаються обробниками подій (повідомлень).

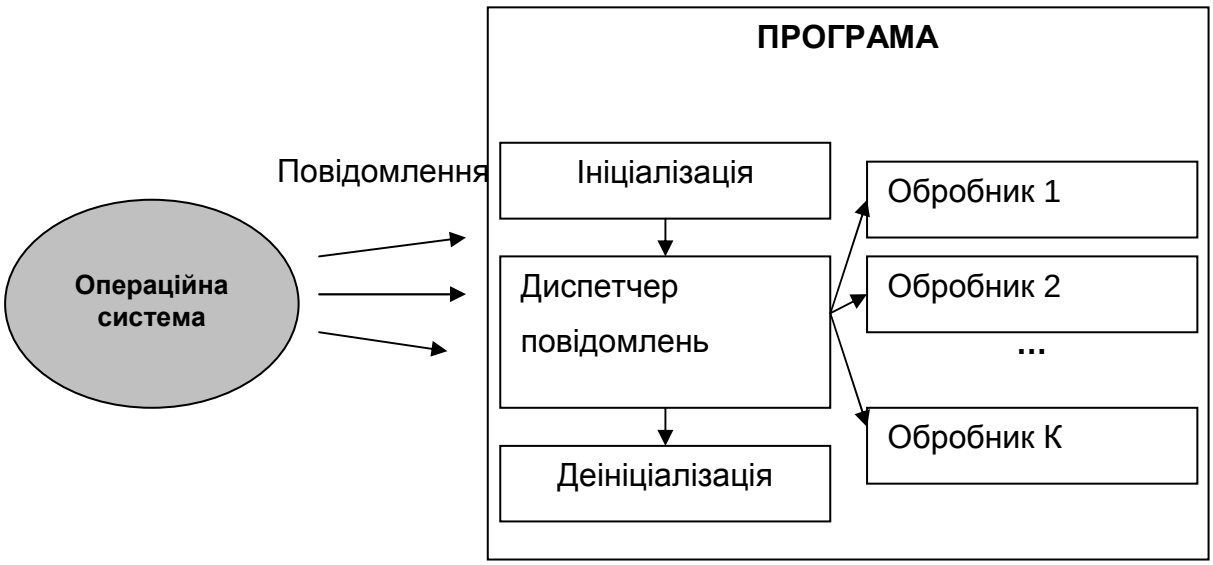


Рис. 2.6. Архітектура програм, керованих подіями

Після того як диспетчер повідомлень отримав повідомлення, він визначає і викликає відповідного обробника події. Якщо такий обробник не передбачений програмістом, спрацьовує стандартний обробник за замовчуванням. Якщо від операційної системи надходить повідомлення про закриття програми, диспетчер повідомлень викликає спеціальний код завершення (деініціалізації) програми. Для цього випадку, наприклад, в класі TForm призначені спеціальні обробники подій OnClose та OnClose Query.

Працювати з сотнями повідомлень Windows, навіть маючи довідник і електронну допомогу, досить важко. Однією з переваг Delphi є те, що програміст може уникнути безпосередньої роботи з ними. В Delphi більшість повідомлень Windows подані як стандартні події (біля двох десятків), використання яких не вимагає глибокого знання самих повідомлень Windows.

Основні обробники подій

Починаючи з класу TControl, елементи управління спроможні обробляти події. В класі TControl вводяться такі групи обробників (спільним параметром для всіх подій завжди є Sender: TObject.):

1. Події від миші:

- OnClick, OnDblClick викликаються, коли відбулося одинарне, або подвійне клацання мишею;
- OnMouseDown, OnMouseUp викликаються, коли кнопка миші натиснута (MouseDown) або відпущена (MouseUp). Параметрами передаються:

- Button: TMouseButton, який набуває одне із значень mbLeft, mbRight, mbMiddle залежно від того, яка кнопка миші була натиснута.
- Shift: TShiftState, який відстежує стан клавіатури і миші. Набуває значення ssShift, ssAlt, ssCtrl, якщо були натиснуті відповідні кнопки на клавіатурі, ssLeft, ssRight, ssMiddle додатково натиснуті кнопки на миші, або ssDouble – обидві кнопки на миші.
- X, Y: Integer – координати клацання.

- OnMouseMove викликаються, коли відбувається просування покажчика миші над елементом. Параметрами є Shift: TShiftState і X, Y: Integer.
- OnContextPopup викликаються, коли відбулося клацання правою кнопкою миші на елементі управління і дозволяє змінювати дії під час виведення випливного меню. Параметрами є MousePos: TPoint – координати миші, Handled: Boolean. За замовчуванням параметр Handled встановлений False, якщо його встановити в True – це припинить обробку події.

2. Події на зміну розміру елемента управління:

- OnResize викликається, коли завершується операція зміни розміру елемента управління;
- OnCanResize викликається перед операцією зміни розміру. Параметрами є NewWidth, NewHeight: Integer – нова ширина та висота елемента; Resize: Boolean, що дозволяє взагалі зміну або ні.
- OnConstrainedResize викликається одразу після OnCanResize. Параметрами є MinWidth, MinHeight, MaxWidth, MaxHeight: Integer – мінімально та максимально припустимі розміри елемента.

3. На події під час перетягування елемента управління:

- OnDragOver виникає при перетаскуванні елемента управління над даним. Параметри:
  - Sender : TObject – елемент, над яким відбувається перетягування;
  - Source : TObject – елемент, який перетяггують;
  - X, Y: Integer – координати курсора миші
  - State: TDragState визначає стан перетягування: dsDragEnter при вході покажчика миші на елемент, dsDragLeave – при виході або якщо кнопка миші була відпущена, dsDragMove – пересування всередині.
  - Accept: Boolean – повідомляє, чи дозволено прийняти (Accept:=true), елемент, що перетаскується. Якщо Accept:=false, то

на цей елемент не можна перетягувати інші.

– OnDragDrop виникає під час перетягування елемента управління (тим, що викликав цей обробник) над даним, коли вже відпущена ліва кнопка миші. Параметрами є вище означені Sender Source : TObject ,X, Y: Integer.

- OnStartDrag, OnEndDrag виникає на початку і в кінці перетягування. Викликається елементом, що перетягують.

Для того щоб перетягування взагалі було можливе для всіх елементів, треба встановити властивість DragMode := dmAutomatic. Властивість DragCursor визначає тип курсора під час перетягування. Обидві властивості встановлюються через Object Inspector або програмно.

4. Події на стикування елементів управління.

Найчастіше стикуються вікна. Наприклад, у робочому середовищі Delphi вікна Code Explore об'єднано з вікном редагування програмного коду. Механізм об'єднання закладений всередині елементів управління. Для того щоб ним скористатися, необхідно для приймача (стикувальна станція – docking site) встановити властивість DockSite:=True. Приймачем може бути будь-який елемент управління, що має цю властивість (Form, Panel, ToolBar і ін.).

Елементи, що стикуються повинні мати властивості DragKind:= dkDock, DragMode:=dmAutomatic.

Нащадки TControl можуть бути тільки елементами, що стикуються. Стикувальними станціями можуть бути тільки нащадки TWinControl.

Обробники OnStartDock і OnEndDock викликаються для стикувальних елементів під час початку та закінчення стикування.

В класі TWinControl вводяться такі групи обробників:

1. Додаткові події від клавіатури:

- OnKeyPress викликаються, коли була натиснута клавіша. Параметр Key: Char – символ, що відповідає клавіші.
- OnKeyDown, OnKeyUp викликаються в момент натискання або відпущення кнопки на клавіатурі. Параметри:
  – Key: Word – код клавіші, що була натиснута:
  – Shift: TShiftState – стан функціональних клавіш на клавіатурі (дивися обробники подій від миші).

2. Додаткові події від миші: OnMouseWheel, OnMouseWheelDown, OnMouseWheelUp викликаються, коли обертається коліща миші або під час клацання коліщам як кнопкою.

3. Події на передачу фокуса введення: OnEnter, OnExit викликається, коли елемент отримує або втрачає фокус введення.
4. Події на стикування елемента управління: обробники OnDockDrop і OnDockOver викликаються для стиковочних станцій під час стикування, а обробник OnUnDock – роз’єднання.

Подія як властивість процедурного типу

В мові Delphi Pascal подія (event) – це властивість процедурного типу, призначена для створення реакції об'єкта на повідомлення, які надходять від операційної системи або інших об’єктів:

```
type
TMyEvent = procedure  (Sender:TObject; MyParameters) of Object ;
TMyClass = class
private
  FOnMyEvent:TMyEvent ;
protected
 procedure  MyEvent  (Sender:TObject ; MyParameters ); dynamic;
public
  property  OnMyEvent:TMyEvent  read  FOnMyEvent write  MyEvent ;
end ;
procedure  TMyClass.MyEvent (Sender:TObject; MyParameters)
begin
  if  Assigned(OnMyEvent)  then OnMyEvent(Sender, MyParameters)
      else . . .  // обробка події  за замовчанням
end ;
```

У наведеному прикладі описано властивість OnMyEvent – властивість процедурного типу TMyEvent, FOnMyEvent – захищене поле процедурного типу TMyEvent, яке фактично буде містити адресу метода-обробника повідомлення.

Процедурний тип метода-обробника повідомлення TMyEvent описується як процедура з набором параметрів, який залежить від події. Але для всіх методів-обробників повідомлення першим параметром має бути параметр Sender типу TObject, що фактично вказує на об'єкт-джерело події.

Для запису властивості описано метод MyEvent. Перевірка умови Assigned(OnMyEvent) є еквівалентною перевірці OnMyEvent <>nil.

Присвоїти значення властивості-події означає вказати об'єкту адресу методу, що викликатиметься в момент настання події. Такі методи називаються **обробниками подій** (Event Handlers).

Найпростіший тип подій не має більше ніяких параметрів, окрім Sender і в Delphi для неї введений спеціальний тип TNotifyEvent:

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

Наприклад, подію на клацання кнопкою миші описано так:

```
property  OnClick : TNotifyEvent;
```

Подія, що виникає під час натискання клавіші на клавіатурі забезпечує передачу програмістові коду натиснутої клавіші:

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

Всі події в Delphi іменують, починаючи з префікса “On”: OnCreate, OnMouseMove і т. д.

Дуже корисно може бути використання оператора as у методах обробки подій. Для забезпечення сумісності джерело події Sender має тип TObject, хоча реально ними можуть бути інші компоненти. Тому, щоб мати можливість користуватися їхніми методами застосовують as [3; 4]:

```
(Sender as TControl).Caption := ‘Thanks’;
```

Поняття делегування подій

Оскільки подія визначається як властивість класу, то її значення можна змінювати не тільки на етапі розробки програми, а і в процесі її виконання. Така можливість зміни обробника подій називається делегуванням подій. Можливо в будь-який момент часу зв’язати обробник події одного об'єкта з обробником події іншого, тим самим призначити (делегувати) подію іншому об’єкту. Наприклад:

```
Button1.OnClick : = Button2.OnClick ;
```

Принцип делегування є найбільш яскравим прикладом інкапсуляції методів у властивостях процедурного типу. Це дозволяє розширювати функціональність об’єкта не шляхом спадкування та перевизначення методів, а через делегування подій.

Використання класів-списків Delphi

Під час створення програм дуже часто доводиться опрацьовувати декілька екземплярів одного класу (або значень одного типу). У таких випадках зазвичай використовуються масиви. Але під час використання масивів є свої недоліки: необхідно зазделегіть визначати кількість елементів у масиві, складності з додаванням та видаленням елементів з середини масиву тощо. Наступним кроком у вирішенні питання обробки набору даних змінного розміру є динамічні структури даних. Але вони не підтримують індексацію елементів.

У Delphi створено декілька класів які поєднують у собі всі надбання масивів і динамічних структур даних. Це класи-списки TList та TStringList.

Клас TList

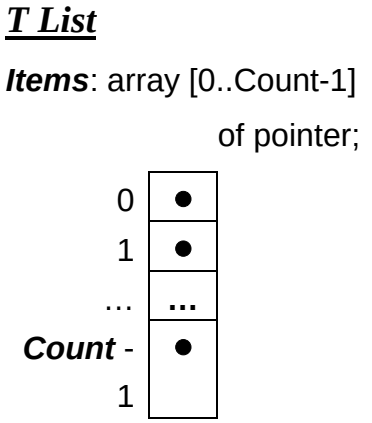


Рис.2.7. Клас TList

Клас TList використовується у програмі для створення списків загального призначення.

Основою класу TList є масив нетипізованих покажчиків Items. Індексація елементів цього масиву починається з 0, поточна кількість елементів у масиві зберігається властивістю Count (рис. 2.7), максимально встановлена кількість елементів задається властивістю Capacity.

Зручним є те, що клас TList надає можливість додавати елементи в кінець списку або всередину, видаляти елементи, міняти їх місцями тощо. Перелік основних властивостей та методів класу наведений в додатку А.

На початку роботи, коли тільки створюється об’єкт-список, кількість елементів у ньому дорівнює 0 (Count = 0). Під час додавання елементів кількість автоматично збільшується, а під час видалення зменшується. Але необхідно звернути увагу на те, що додавання елемента, а особливо видалення, не впливає на область пам’яті, на яку посилається покажчик зі списку. Перш ніж скористатися, наприклад, методом Add необхідно виділити область пам’яті під елемент, що додається в список, а потім додати його до списку (рис. 2.8).

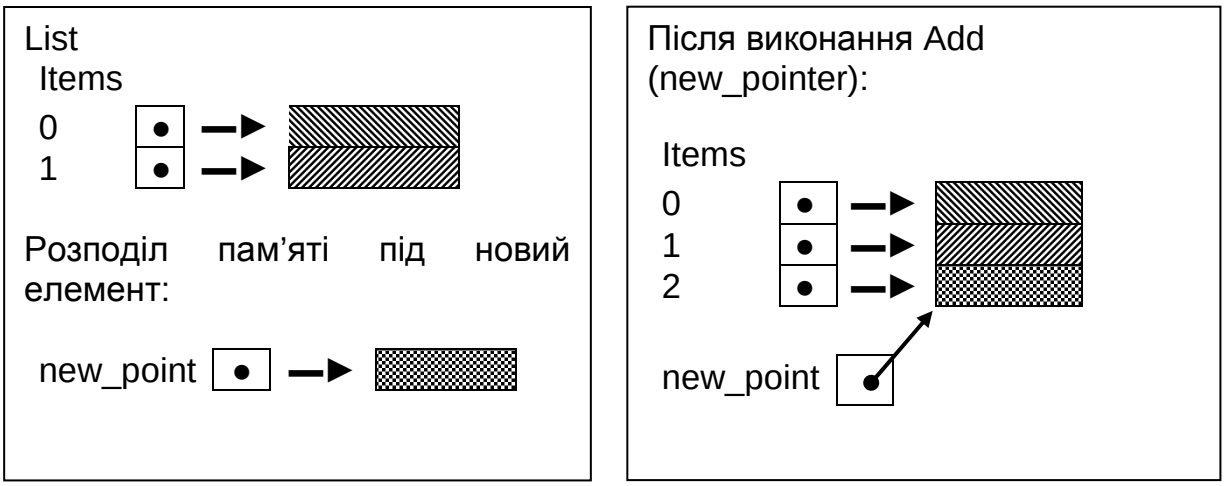


Рис 2.8. Додавання нового елемента до списку.

Аналогічно і для видалення. Метод Delete видаляє тільки покажчик з масиву Items і не звільняє область пам'яті, на яку він посилається (рис. 2.9). Теж саме стосується методів Insert, Remove, Clear.

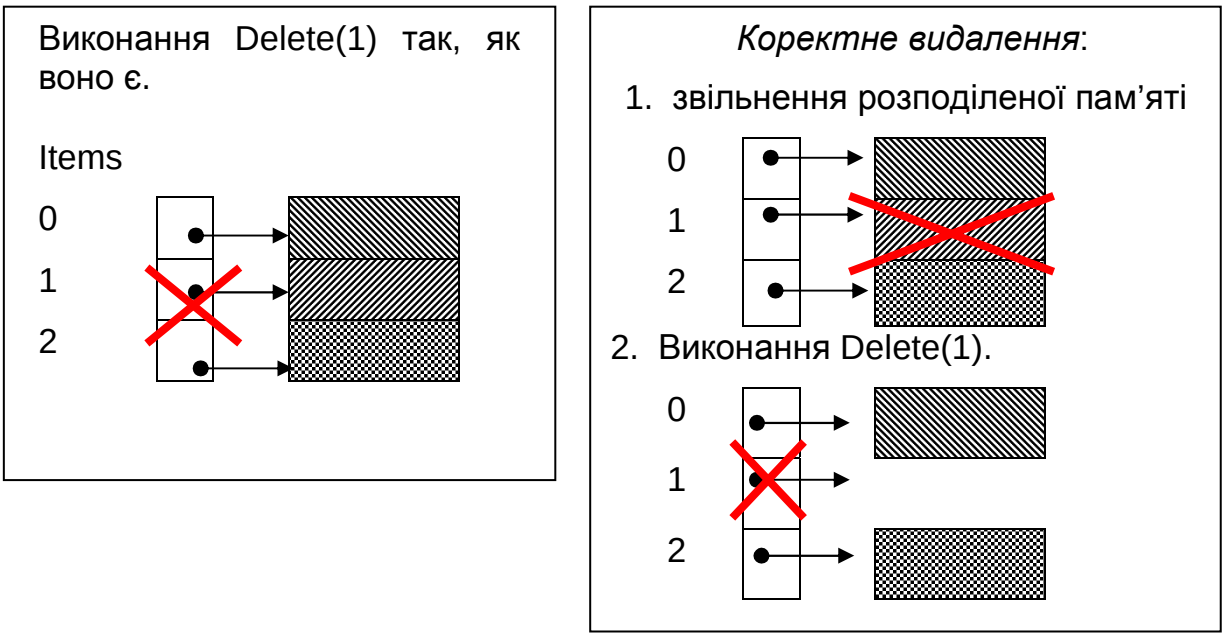


Рис 2.9. Видалення елемента зі списку.

У переважній більшості випадків при використанні класу TList необхідно описати в програмі два класи: один – для об'єктів, що будуть зберігатися в списку, другий – клас-нащадок списку, для якого обов'язковим є перевизначення деструктору. В новому деструкторі спочатку звільняється пам'ять, яка виділена під елементи списку, а потім руйнується він сам. Також бажано розширити функціональність класу для

коректної реалізації методів додавання, видалення елементів та очищення списку.

Класи TStrings та TStringList

Клас TStrings є базовим класом, який забезпечує своїх нащадків основними властивостями та методами, які дозволяють створювати списки рядків. Його прямим предком є клас TPersistent, а нащадком клас TStringList [3, 4].

Класу TStrings складається з двох масивів: Strings та Objects (рис. 2.10). Також як і в класі TList використовується властивість Count – кількість елементів в списку.

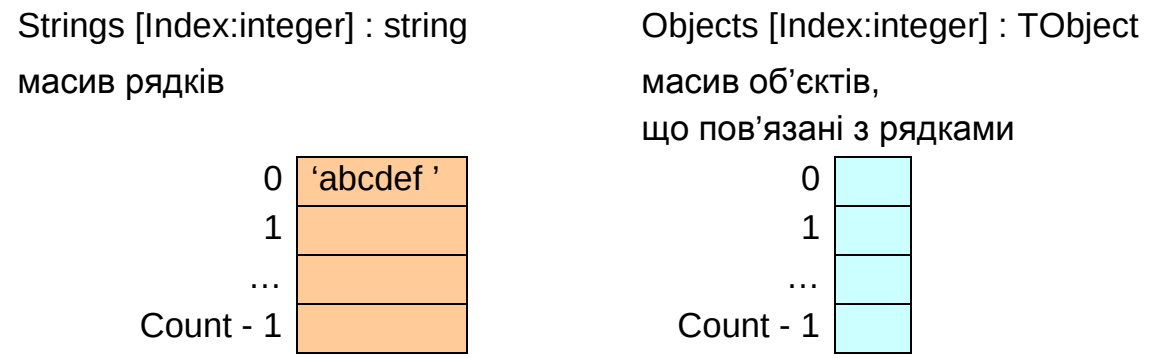


Рис. 2.10 Властивості класу TStrings

Властивість Strings класу TStrings є властивістю за замовчанням, тому для екземпляру цього класу, наприклад, SomeStrings можливий такий синтаксис:

SomeStrings.Strings[0]:= ‘abcd’;

або

SomeStrings[0]:= ‘abcd’;

без явної вказівки властивості Strings.

Клас TStrings дуже широко використовується в компонентах Delphi для збереження списку рядків. Наприклад, компонент TMemo – вдастивість Lines, компонент TListBox, TComboBox, TRadioGroup – вдастивість Items. Такий компонент як TStringGrid також побудований на класі TStrings (рис. 2.11). Властивості Rows (рядки) та Cols (стовпці) класу TStringGrid є списками рядків. Перетин цих списків дає комірку типу string

(властивість Cells[j, i]).

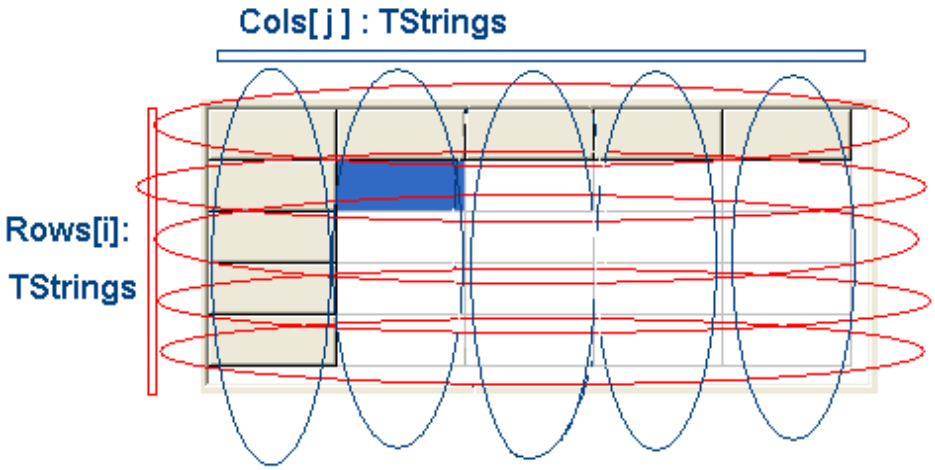


Рис. 2.11. Застосування класу TStrings в класі TStringGrid

Клас TStrings є абстрактним класом, тому небажано створювати екземпляри цього класу. Але можна скористатися конструктором нащадка класу TStrings – конструктором класу TStringList.

```
Var Str:TStrings;
… … …
Str:= TStringList.Create;
```

Після цього можна використовувати список рядків наприклад так:

```
Str.Add('First string');
Str.Add('Second string'); … і так далі.
```

Повний перелік властивостей класів TStrings та TStringList наведений у додатку А.

Сталість об'єктів

Важко уявити собі більш-менш серйозне застосування, яке б не зберігало дані, що обробляються. Об'єкт вважається сталим, якщо його стан (значення полів) зберігається в певному сховищі даних, де під сховищем даних можна розуміти файл, базу даних і т.п. Сталість об'єкту буває внутрішня, коли він сам зберігає свої дані, та зовнішня, коли зовнішні об'єкти організують збереження полів даного об'єкту. Для цього можна використати, наприклад, типізований файл, але корисним буде познайомитися зі спеціальним об'єктом – потоком даних.

Клас TStream

Потік даних – однорідна послідовність байтів (на зразок

нетипізованого Pascal-файлу) – дуже вдалий засіб для уніфікації введення/ виведення для різних носіїв інформації.

Потоки являють собою спеціальні об’єкти-нащадки абстрактного класу TStream. Клас TStream “вміє” відкриватися, читати, писати, змінювати поточне розташування та закриватися. Крім того клас TStream надає набір методів для зберігання та зчитування компонентів – нащадків класу TComponent (властивості класу TStream наведені у додатку А). Оскільки для різних носіїв ці речі відбуваються по-різному, конкретні аспекти реалізовані у його нащадках. Найчастіше використовуються потоки для роботи з файлами на диску та пам’яттю [3, 4].

Клас TFileStream

Клас TFileStream – прямий потомок класу TStream, дозволяє створювати потік для роботи з файлом. Для відкриття файлу-потоку використовується конструктор:

constructor Create (var FileName:string; Mode:Word);

де FileName – повне ім’я файлу, а Mode – режим доступу до файлу, який може задаватися такими константами:

- fmCreate – файл створюється;
- fmOpenRead – файл відкривається для читання;
- fmOpenWrite – файл відкривається для запису;
- fmOpenReadWrite – файл відкривається для читання та запису.

У разі необхідності при сумісному використанні одного файлу декількома програмними застосуваннями при відкритті файлу можуть задаватися прапори сумісного використання [4, с. 216].

Для пересування по файлу використовується процедура:

Seek(Offset:Longint; Origin:Word);

де Offset – кількість байт, на яку треба пересунути файловий покажчик, Origin – напрям, у якому відбувається пересунення, може приймати такі значення:

- soFromBeginning – зміщення задається додатним числом і відраховується від початку потоку;
- soFromCurrent – зміщення задається додатним чи від’ємним числом і відраховується від поточної позиції;
- soFromEnd – зміщення задається від’ємним числом і відраховується від кінця потоку.

Висновки

- Всі компоненти Delphi є класами, що утворюють ієрархію спадковості. Спільним предком всіх компонентів є клас TObject.
- Компоненти можуть бути візуальними та невізуальними.
- Всі нащадки класу TPersistent є сталими об’єктами. На рівні TPersistent введено метод, що реалізує механізм присвоєння одного сталого об’єкта іншому.
- Клас TComponent є класом-предком для всіх компонентів VCL. В цьому класі вводяться найголовніші властивості і методи всіх компонентів, змінює форму конструктора, що успадкований від TObject.
- Компоненти можуть бути власниками інших компонентів, список яких зберігається у властивості-масиві Components. Кожний компонент “знає” свого власника, посилання на якого зберігається в спеціальних властивостях.
- Клас TControl – нащадок класу TComponent, є предком для всіх елементів управління Windows. На рівні TComponent вводяться основні обробники подій та властивості, що відповідають за розмір, розташування, видимість та активність елементів управління.
- Клас TWinControl – предок всіх віконних елементів управління. Віконний елемент управління відрізняється від простого елементу управління трьома рисами: наявність дескриптора вікна; можливість бути батьківським (не плутати зі спадкуванням!) щодо інших елементів; можливість отримання фокусу введення.
- Типова сучасна програма це – програма, що керується подіями. Звістка про настання події – це повідомлення, що генерується операційною системою як реакція на дії користувача, сигнали від апаратури, повідомленнями інших програм та самої операційної системи.
- Обробник подій (повідомлень) – підпрограма, що реалізує реакцію програми на настання події. Обробник повідомлень визначається диспетчером повідомлень. Якщо обробник не заданий під час розробки програми, спрацьовує стандартний обробник за замовчанням.
- На рівні класів – нащадків TControl вводяться обробники подій від миші, клавіатури, на зміну розміру, перетаскування та стикування елементів управління тощо.
- Подія – це властивість процедурного типу, що призначена для створення реакції об'єкту на повідомлення, які надходять від операційної системи або інших об’єктів.

- Принцип делегування дозволяє розширювати функціональність об’єкту не шляхом спадкування та перевизначення методів, а через делегування подій.
- Для створення списків об’єктів призначені класи TList, TStrings, TStringList. Класи TList та TStrings є абстрактними, тому з ними краще працювати через їх нащадків.
- Сталість об’єкту – здібність об’єкту зберігати та відновлювати свій стан. Сталість об’єкту буває внутрішня та зовнішня.
- Потоки, нащадки абстрактного класу TStream, надають можливість уніфікованого введення/ виведення однорідної послідовності байтів для різних носіїв інформації. Клас TStream надає набір методів для зберігання в потік та зчитування з потоку компонентів – нащадків класу TComponent.

Запитання та завдання для самоперевірки

1. Які класи входять до ієрархії базових класів Delphi?
2. Надайте класифікацію класів-компонентів Delphi.
3. Охарактеризуйте клас TPersistent. Як в Delphi здійснюється присвоєння об’єків?
4. Охарактеризуйте клас TComponent. Які компоненти вважаються власниками інших компонентів?
5. Для чого призначені властивості Name і Tag нащадків класу TComponent?
6. Охарактеризуйте клас TControl. Які елементи управління Ви знаєте?
7. Чим характеризуються віконні елементи управління? Який з базових класів є предком всіх віконних елементів управління?
8. Які властивості визначають розташування та розмір елементів управління?
9. Які властивості визначають видимість та активність елементів управління?
10. Для чого призначена властивість Hint? Як визначити можливість її відображення?
11. Охарактеризуйте віконні елементи управління – нащадки класу TWinControl.
12. Що таке фокус введення, як ним керувати?
13. Чим відрізняється батьківський компонент і компонент-власник? Яке їх призначення? Чи завжди списки підлеглих їм елементів ідентичні?
14. Дайте визначення подіє-орієнтованого програмування.

15. Що таке подія? Як виникають події і як програми сповіщаються про настання події?
16. Охарактеризуйте архітектуру програм, керованих подіями.
17. Які групи подій опрацьовуються класом TControl?
18. Які обробники подій від клавіатури, миші та на передачу фокусу введення?
19. Як обробляється перетаскування та стикування елементів управління?
20. Для чого призначені методи-обробники подій? Особливості визначення списку формальних параметрів для обробників подій.
21. Як власним класам надати можливість оброблювати події від ОС або інших об’єктів?
22. Який синтаксис опису обробника подій як властивості процедурного типу?
23. Що означає присвоєння значення властвості-події?
24. Для чого призначений клас TNotifyEvent і як ним користатися?
25. Як Ви розумієте принцип делегування подій?
26. В чому переваги використання класів-списків Delphi?
27. Як організований клас TList? Чи можна в класі TList зберігати базові типи даних або об’єкти?
28. В чому особливості реалізації додавання та видалення елементів зі списку, що зберігається в TList?
29. Як і для чого в програмі створювати нащадків класу TList? Чому вважається доцільним перевизначення деструктору класу, що використовує клас TList?
30. Яке призначення та особливості будови класу TStrings?
31. Які компоненти Delphi мають в своєму складів властивість-екземпляр TStrings?
32. Як доцільно створювати екземпляри класу TStrings?
33. Що таке сталість об’єкту? Нащадки якого класу Delphi набувають сталості?
34. Охарактеризуйте поняття потоку даних та класи, що призначені для їх опрацювання.
35. Які властивості класу TFileStream?

Розділ 3. Побудова графічних зображень засобами графічної підсистеми Delphi

Загальні принципи побудови графічного зображення

Як відомо відеоадаптер, який керує виведенням інформації на екран комп'ютера, може працювати двох режимах: текстовому та графічному. Відмінності в роботі цих режимів розглянемо на прикладі відображення на екрані шрифтових знаків (символів).

Текстовий режим роботи екрану

Текстовий режим – режим відеоадаптера, в якому екран розбитий на рядки та стовпчики, на зразок матриці, в кожну комірку якої можна вивести лише один символ з обмеженого набору (наприклад, з таблиці ASCII-кодів символів).

Кількість рядків та стовпчиків екрана, в які виводяться символи (найчастіше, 25X80), а також кількість кольорів, які можуть використовуватися під час виведення, встановлюється відповідним режимом роботи відеоадаптера. При цьому відлік рядка та стовпчика починається з лівого верхнього кута екрану.

Шрифти, які використовуються в текстовому режимі роботи екрану називаються растровими, тому що кожен шрифтовий знак (символ) описується матрицею (растром) з підсвітлених точок екрану – пікселів (рис. 3.1).

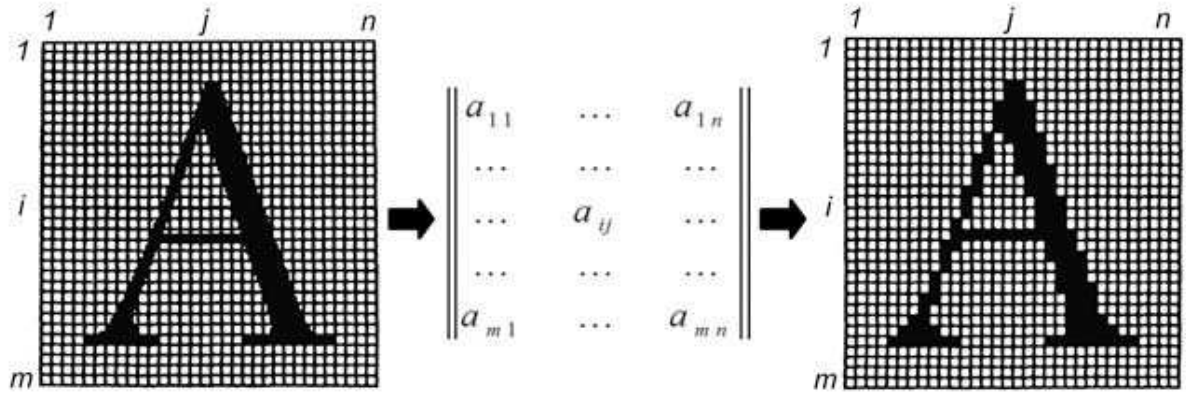


Рис. 3.1 Растрове подання символу [8]

Формування кольору в текстовому режимі

Кольори шрифтових знаків (символів) встановлюються трійкою елементарних кольорів RGB (Red-Green-Blue). Сполучення відповідних складових червоного, зеленого, синього кольору та яскравості визначають інші кольори та відтінки (рис. 3.2) Ознака наявності складової у формуванні кольору позначається: 0-відсутність або 1-наявність (табл. 3.1).

В результаті отримується послідовність нулів та одиниць – код відповідного кольору.

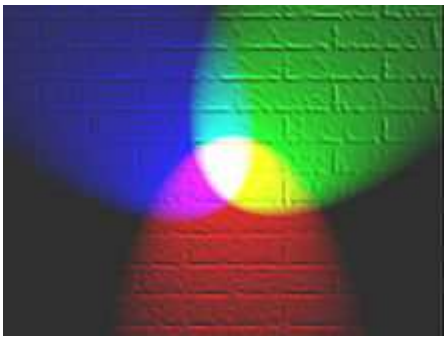


Рис. 3.2 RGB-палітра кольорів

Таблиця 3.1 – Формування RGB-кольору текстового зображення

Код	Яскр.	R	G	B	Колір	Код	Яскр.	Колір
0	0	0	0	0	чорний	8	1	темно-сірий
1	0	0	0	1	синій	9	1	яскраво-синій
2	0	0	1	0	зелений	10	1	яскраво-зелений
3	0	0	1	1	голубий (циан)	11	1	яскраво-голубий
4	0	1	0	0	червоний	12	1	яскраво-червоний
5	0	1	0	1	фіолетовий (магента)	13	1	рожевий
6	0	1	1	0	коричневий	14	1	жовтий
7	0	1	1	1	сірий	1	1	біли

Код	Яскр.	R	G	B	Колір	Код	Яскр.	Колір
						5		й

Кодування кольорів з таблиці 3.1 дає тільки 4 біти для визначення кольору символу. Під час виведення на екран до цих чотирьох бітів додається ще чотири біти, що задають колір фону, на якому виводиться символ. Якщо колір фону заданий з підвищеною яскравістю, тоді встановлюється ефект мерехтіння. Послідовність з восьми бітів, що описує колір символу називається байт-атрибутом символу:

Байт-атрибут символу							
колір фону				колір символу			
мерехт.	Red	Green	Blue	яскрав.	Red	Green	Blue
1	0	0	1	1	1	1	0
синій фон, що мерехтить				*жовта буква*			

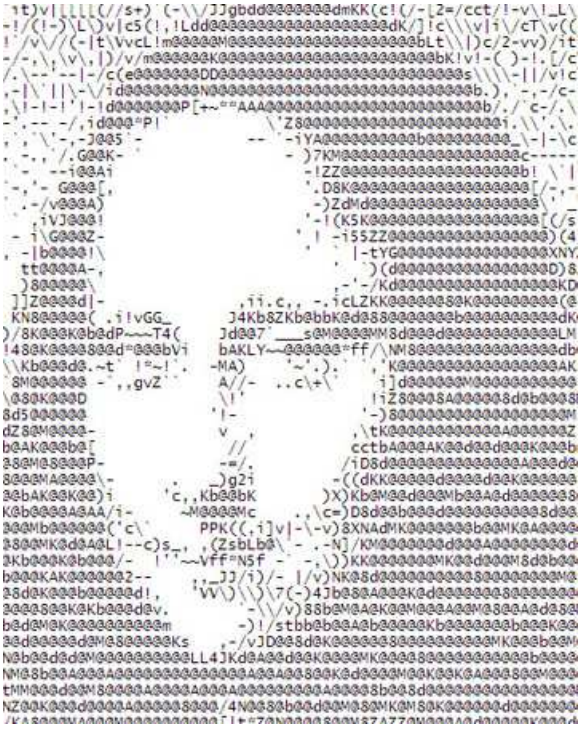
В текстовому режимі роботи екрану побудова графічних зображень можлива тільки з використанням растрів наперед заданих символів:

- символів букв, цифр, дужок тощо;
- символів псевдографіки (символи псевдографіки наведені у таблиці 3.2).

Приклади побудови графічних зображень в текстовому режимі роботи екрану наведені на рисунку 3.3 [10, 11]

Таблиця 3.2 – Символи та коди символів псевдографіки

▒ 176	▓ 177	▓ 178	│ 179	┤ 180	╡ 181	╢ 182	╖ 183	╕ 184	╣ 185	║ 186	╗ 187	╝ 188	╜ 189	╛ 190	┐ 191
└ 192	┴ 193	┬ 194	├ 195	─ 196	┼ 197	╞ 198	╟ 199	╚ 200	╔ 201	╩ 202	╦ 203	╠ 204	═ 205	╬ 206	╧ 207
╨ 208	╤ 209	╥ 210	╙ 211	╘ 212	╒ 213	╓ 214	╫ 215	╪ 216	┘ 217	┌ 218	▉ 219	▄ 220	▌ 221	▐ 222	▀ 223



а)

б)

Рис. 3.3 Приклади побудови графічних зображень з використанням символів в текстовому режимі роботи екрану: а) – букви і цифри; б) - символи псевдографіки.

Багато широковживаних, особливо за часи використання MS-DOS, програм реалізують свій інтерфейс з користувачем шляхом побудови вікон, використовуючи також символи псевдографіки: програми-оболонки Norton Commander, Volkov Commander, FAR; середовища розробки TurboPascal, TurboC, TurboC++ тощо (рис. 3.4).

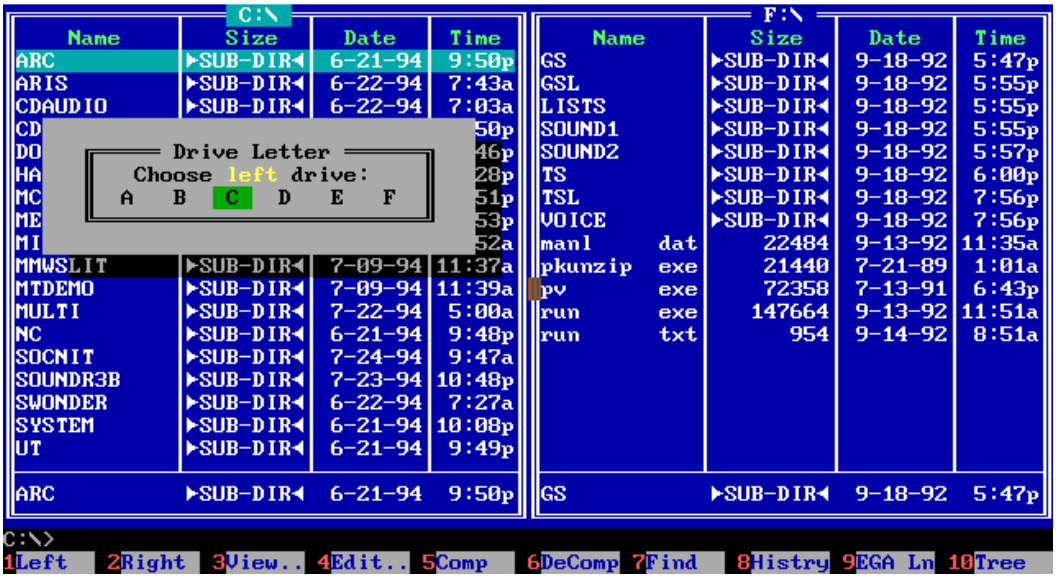


Рис. 3.4 Приклади програмного інтерфейсу, що побудований з

використанням символів псевдографіки [12].

Графічний режим роботи екрану

В графічному режимі екран розбивається не на рядки та стовпчики, а на множину точок – пікселів. Кількість пікселів по горизонталі і вертикалі, а також кількість кольорів, які можуть використовуватися встановлюється також відповідним режимом відеоадаптера.

В графічному режимі на екран виводяться довільні графічні об’єкти, зображення яких задається контуром. До таких об’єктів відносяться також і букви алфавітів, цифри та інші знаки.

Відомі два основних способи опису інформації про контур шрифтового знака [8]:

- контурно-векторний;
- контурний.

При описі знаків в контурно-векторній формі контур знака відтворюється (апроксимується) послідовністю векторів (рис. 3.5, а). В лініях контуру задаються початкові координати першого вектора, відносно обраної точки на лінії шрифту, і прирости координат кінців векторів по осям x та y. [8]

При контурному описі шрифтового знака контур розбивається на окремі ділянки, що представляють собою відрізки прямої, дуги кіл і кривих (рис. 3.5, б). При моделюванні контурів, крім координат початку і кінця векторів, задаються параметри дуг кіл (радіус кола, координати центру кола, координати точок сполучення дуг). [8]

Після побудови моделі контуру проводиться растеризація (формування растру) цього контуру. Фактично, контурний шрифт перетворюється в растровий. [8]

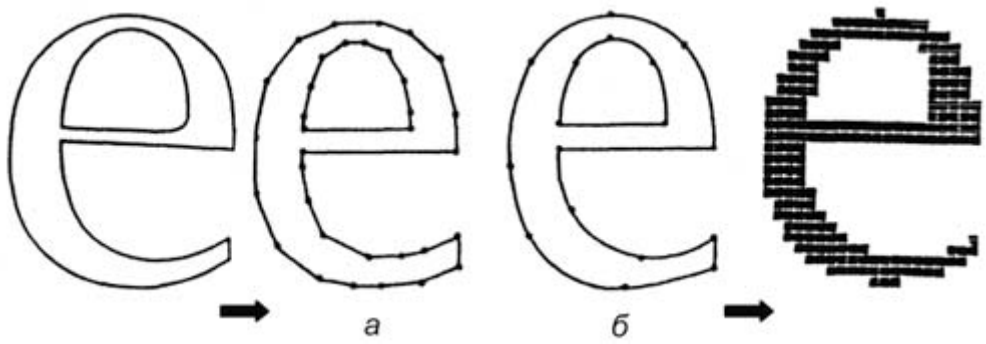


Рис. 3.5 Способи моделювання нарису шрифтових знаків при контурно-векторному (а) і контурному (б) описі зображення знаків [8]

Формування кольору в графічному режимі

В графічному режимі роботи екрану, так само як і в текстовому режимі, для формування кольору пікселів застосовується RGB-модель. Але використання 4-бітного кольору недостатньо для формування реалістичних зображень (рис. 3.6). Найбільш прийнятна кольорова гама, яка, наприклад, використовується для формування зображень на фотографіях, отримується 24-бітним форматом кольору (TrueColor). При цьому на кожну складову RGB відводиться не по одному, а по 8 бітів. Це дає кольорову палітру, що складається з 16 777 216 кольорів.

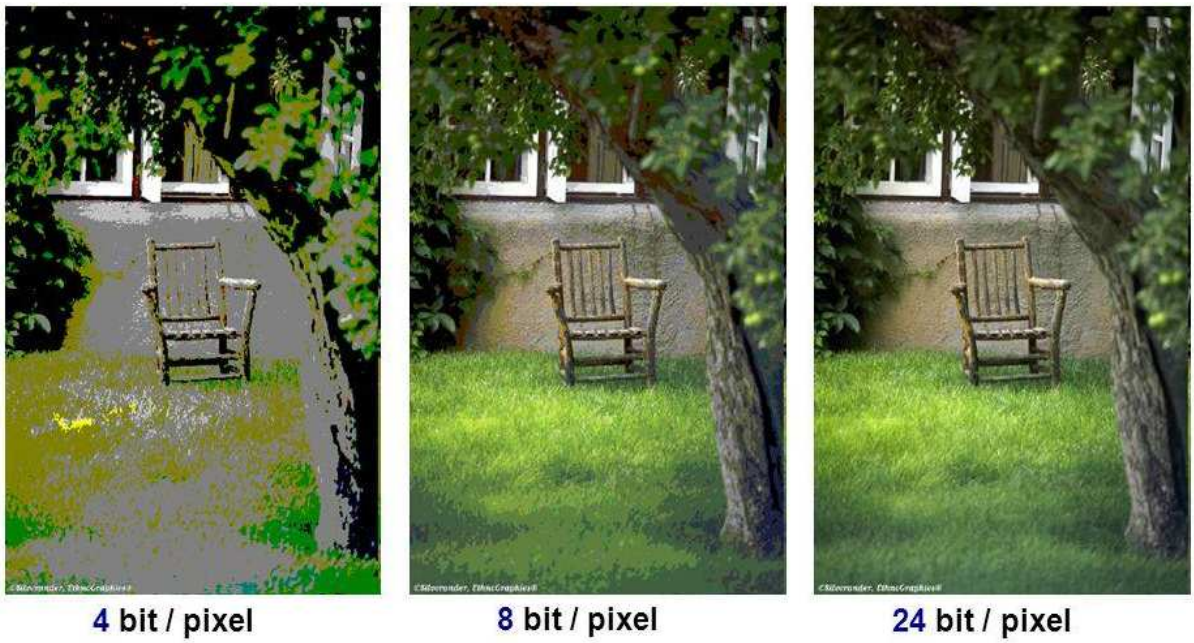


Рис. 3.6 Зміна якості зображення в залежності від глибини кольору [14]

Кількість бітів, що використовується при формуванні кольору одного пікселя називається глибиною кольору і позначається «біт на піксель» (bits per pixel, bpp).

Колір шрифтових знаків (а також інших графічних об’єктів) може визначатися двома складовими (рис. 3.7):

- колір, стиль та товщина контуру;
- колір та стиль заповнення (заливки).

Рис. 3.7. Приклади застосування різних кольорів та стилів контуру та

заповнення шрифтових знаків [8]

Екранна система координат

В графічному режимі екран комп'ютера можна розглядати як декартову площину, яка задається осями абсцис ОХ та ординат ОУ. При цьому вісь ординат спрямована донизу. Координати точок цієї площини вимірюються (задаються) в пікселах. Відлік пікселів починається з лівого верхнього кута екрану.

При побудові на екрані графічних зображень необхідно пам'ятати, що координати об'єктів в природній системі будуть дзеркально обернені в екранній площині (рис. 3.8).

Тому необхідно проводити перетворення з природних координат об'єкта в екранні.

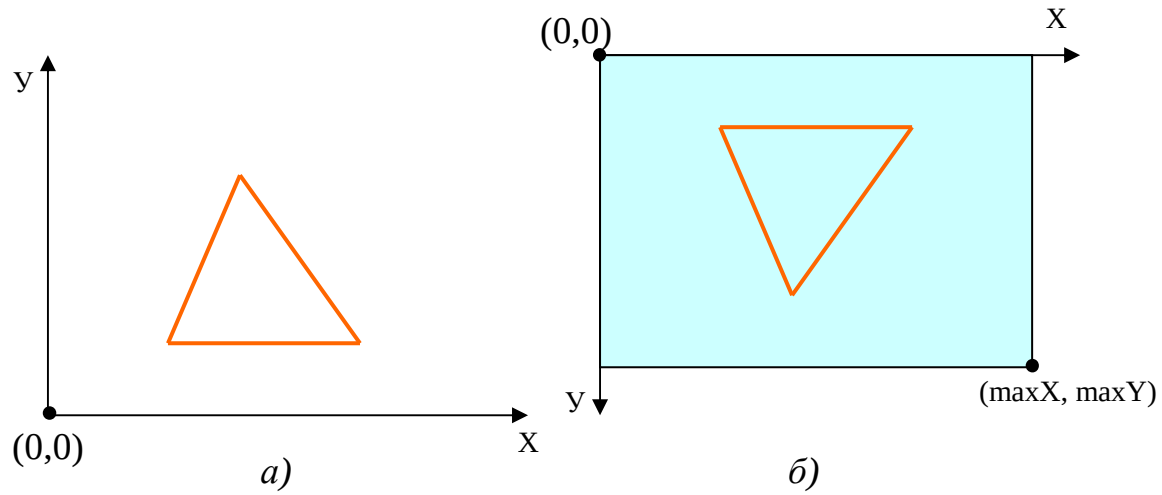


Рис. 3.8 Відмінності природної (а) та екранної (б) системи координат

Якщо побудова зображення здійснюється відносно екранної точки «початку координат» - точки (0, 0) на рисунку 3.8, б, тоді точка А з декартовими координатами (X_A,Y_A) на екрані буде мати координати:

$$X_A^e = [X_A * M_x],$$

$$Y_A^e = \text{maxY} - [Y_A * M_y]$$

де [*вираз*] – позначення цілої частини значення; maxY – кількість пікселів по вісі ОУ; M_x, M_y - масштабні коефіцієнти для переведення величин X_A,Y_A в пікселі. Наприклад, якщо дві точки мають координати (1.5, 2) і (2.5, 3), для відображення їх на екрані необхідно збільшити ці координати на деякі масштабні коефіцієнти, тому що різниця координат в 1-2 пікселі майже не розрізнена. Необхідно зауважити, що для того щоби

графічні об'єкти при відображенні на екрані зберігали свої пропорції, необхідно щоби $M_x=M_y$.

Віднімання при обчисленні У-координати визначено тим, що ось ОУ екрану спрямована донизу (для запобігання дзеркального відображення об'єктів).

Якщо задати на екрані умовний центр координат (C_x^e, C_y^e), тоді точка А з декартовими координатами (X_A, Y_A) на екрані буде мати координати:

$$X_A^e = C_x^e + [X_A * M_x],$$

$$Y_A^e = C_y^e - [Y_A * M_y]$$

де C_x^e, C_y^e – центр координат, обраний на екрані; M_x, M_y - масштабні коефіцієнти для переведення величин X_A, Y_A в пікселі (рис. 3.9).

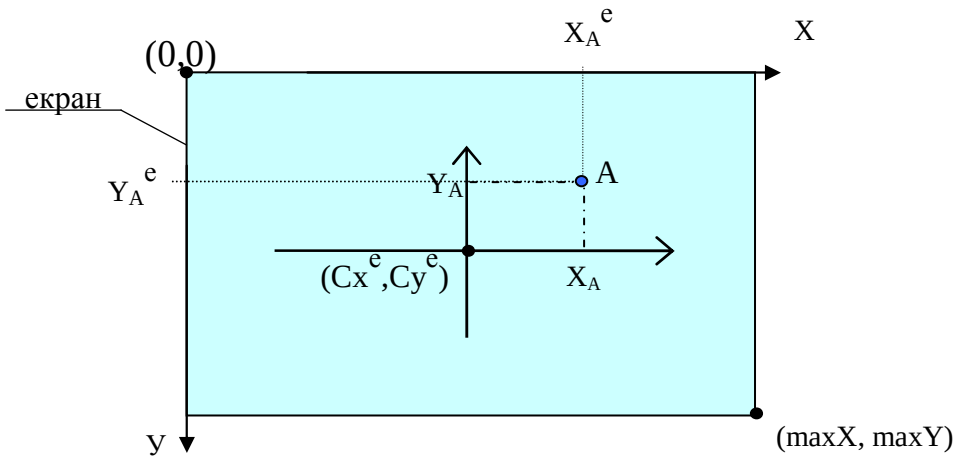


Рис. 3.9 Визначення екранних координат відносно заданого центру

Базові компоненти графічної підсистеми Delphi

В операційній системі (ОС) Windows при побудові графічних зображень використовується спеціалізована складова цієї ОС – підсистема GDI (Graphics Device Interface), яка забезпечує формування графічних зображень і передачу їх на пристрої відображення: монітори і принтери. GDI відповідає за побудову ліній і кривих, формування шрифтових знаків і обробку кольорової палітри.

Використання GDI значно спрощує програмування побудови графічних зображень на пристроях різного типу, тому що забезпечує однакові принципи формування зображень, які не залежать від особливостей організації кожного конкретного пристрою. Використовуючи GDI, можна легко малювати на кількох різних пристроях,

таких як екран або принтер, і досягти практично однакового відображення на них.

Прості ігри, які не потребують швидкої графіки, використовують GDI, але GDI повільний задля швидкої графіки, анімації та 3D. Сучасні ігри використовують DirectX чи OpenGL, що дає програмістам доступ до більш потужних апаратних можливостей.

Кожен тип графічного пристрою (монітор, принтер тощо), на якому виводиться графічне зображення, в ОС Windows описується спеціальним об'єктом GDI, який називається «контекст пристрою» (Device Context, DC). Програма, яка воліє вивести на пристрій графічне зображення, повинна отримати від ОС дескриптор контексту графічного пристрою (handle Device Context**,** hDC) для коректного утворення цього зображення.

Дескриптор – це структура даних ОС Windows, яка надає прикладним програмам доступ до екземпляру базового об'єкта операційної системи, наприклад, файлу, вікна, пристрою. Дескриптор однозначно ідентифікує створений об'єкт і дозволяє читати та змінювати його стан.

Доступ до дескрипторів DC в Delphi надається через графічні елементи управління (рис. 2.1 з попереднього розділу) – нащадки TGraphicControls.

Клас TGraphicControl

Клас TGraphicControl – прямий нащадок від TControl – є основою для невіконних елементів управління (рис.2.1, 2.2). Невіконні елементи управління не мають дескриптору вікна і не можуть отримати фокус введення, але вони можуть обробляти події.

Нащадками цього класу є:

- класи загального призначення TSpeedButton, TBevel, TSplitter і TCustomLabel, від якого породжені TLabel і TDBText;
- класи, що використовуються для побудови та відображення графічних зображень TImage, TPaintBox, TShape.

В доповнення до всіх властивостей, що спадкуються від TControl, клас TGraphicControl дає своїм нащадкам властивість TCanvas (часто використовується назва – *канва*).

Клас TCanvas

Цей клас – серцевина графічної підсистеми Delphi. Він об'єднує у собі полотно – робочу поверхню для малювання, робочі інструменти (перо, пензель, шрифт), а також набір функцій для побудови графічних

примітивів (фігур).

Перо (***Pen***) визначає колір та стиль контуру графічного примітива.

Пензель (***Brush***) визначає колір та стиль заповнення (заливку) графічного примітива.

Шрифт (***Font***) визначає характеристики шрифтових знаків, що можуть використовуватися при побудові зображення.

Клас TCanvas отримує від ОС дескриптор контексту графічного пристрою, який зберігається властивістю **Handle** : hDC. Для кожного hDC операційною системою підтримуються свої налаштування пера, пензля і шрифту. Для доступу до цих інструментів клас TCanvas має властивості **Pen**, **Brush**, **Font**, які є екземплярами відповідних класів TPen, TBrush, TFont. Основні властивості цих класів наведені у додатку Б.

Не зважаючи на те, що візуально в графічному режимі не має поняття курсору, який визначає в текстовому режимі роботи екрану поточну «комірку», в яку буде виводитися символ, існує поняття поточного піксела, з якого розпочинається побудова зображення, якщо не було спеціальних вказівок про його переміщення. Координати поточного піксела знаходяться у властивості **PenPos**: TPoint. Змінити поточну позицію можна за допомогою **MoveTo** (X, Y: Integer), задаючи як параметри нові координати. Отримати доступ до кольору кожного піксела екрану можна через властивість-матрицю **Pixels** [X, Y: Integer]: TColor.

У канви є два обробника подій:

- **OnChange,** який викликається, коли зображення тільки що змінилося;
- **OnChanging**, який викликається, коли будуть відбуватися зміни в зображенні.

Крім того канва надає методи для побудови графічних фігур, аналогічні процедурам модуля Graph Turbo Pascal 7.0.

Графічні примітиви, що реалізує Canvas

Лінії

Побудова прямої лінії (відрізка) з поточної позиції в координати (X,Y):

procedure **LineTo** (X, Y: Integer);

Найчастіше для побудови відрізка з координатами кінців (x1, y1) та (x2, y2) метод LineTo використовують в парі з методом MoveTo: с початку методом MoveTo переводять поточний піксел в координати (x1, y1), а

потім викликається LineTo:

Form1.Canvas.MoveTo (x1, y1); Form1.Canvas.LineTo (x2, y2);

Слід відмітити, що зображення будується поточним кольором пера та пензля. Змінювати поточні налаштування кольорів на нові необхідно до початку виведення зображення.

Круги та еліпси

procedure Ellipse(X1, Y1, X2, Y2: Integer) ;

Рисує та зафарбовує еліпс, що вписаний в прямокутник з координатами (X1, Y1) та (X2,Y2). Для того, щоби нарисувати коло необхідно задати квадрат.

procedure Pie (X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);

Рисує та зафарбовує сектор еліпса, що вписаний в прямокутник з координатами (X1, Y1) та (X2,Y2). Сектор визначається променями, що проходять з центру еліпса через точки (X3,Y3) та (X4,Y4), що лежать на сторонах прямокутника. Сектор рисується проти годинної стрілки від точки (X3,Y3) до (X4,Y4) (рис. 3.10). Слід зауважити, що при використанні методу Pie прямокутник, промені та координати точок на екран не виводяться (рис. 3.11).

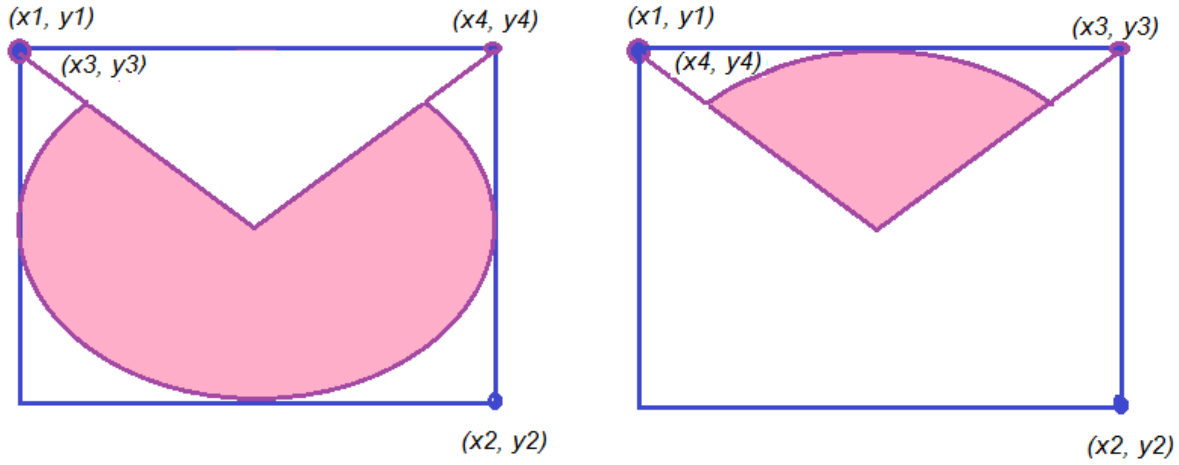


Рис. 3.10 Схема завдання координат для визначення сектора еліпса методом Pie.

procedure Arc (X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer) ;

Метод рисує сегмент еліпса. Початкова та кінцева точки визначаються аналогічно до Pie.

procedure Chord (X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);

Рисує хорду і зафарбовує частину еліпса, що залишається. Точки

задаються аналогічно до Pie та Arc.

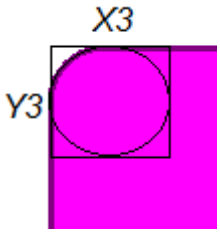
Рис. 3.11 Зображення сектора (Pie), сегмента (Arc) еліпса та частини, що відсікається хордою (Chord).

Прямокутники та багатокутник

procedure Rectangle (X1, Y1, X2, Y2 : Integer) ;

Рисує та зафарбовує прямокутник з лівим верхнім кутом в (X1, Y1) та правим нижнім в (X2.Y2).

procedure RoundRect (X1, Y1, X2, Y2, X3, Y3: Integer);



Рисує прямокутник із заокругленими кутами. Координати вершин аналогічні до методу Rectangle. Заокруглення рисуються як сегмент (відповідна чверть) еліпса з розмірами по горизонталі та вертикалі X3 и Y3.

Ламана лінія, багатокутник

Побудова ламаної, вузли якої задані в масиві точок Points: array of TPoint:

procedure **Polyline** (Points) ;

Тип TPoint є класом з полями (x, y) типу Longint.

Наприклад, сформовано масив координат точок:

Var P : array [1..5] of TPoint = ((x:10; y:10), (x:60; y:200),
(x:110; y:10), (x:160; y:200), (x:210; y:10));

Побудувати таку пилоподібну ламану на канві форми можна так: Form1.Canvas.Polyline (P);

procedure Polygon (const Points: array of TPoint) ;

Рисує та зафарбовує багатокутник аналогічно до ламаної, але при цьому остання точка поєднується з першою.

Діаграми та графіки

Використовуючи властивості та методи класів TCanvas, TPen, TBrush можна самостійно будувати різноманітні графічні зображення. Delphi надає також потужні готові класи формування різноманітних діаграм та

графіків, які дуже часто використовуються в інженерній практиці та в економічних разрахунках.

Компонент Delphi клас – TChart (вкладка Additional VCL) призначений для роботи зі складними діаграмами та графіками різних видів. Цей компонент дозволяє будувати дво- і тривимірні діаграми на основі різноманітних даних.

Робота з даним компонентом виконується за допомогою редактора Editing Chart, який викликається подвійним кліком на компоненті Chart під час проектування форми або через поле значення властивості SeriesList в Objeci Inspector.

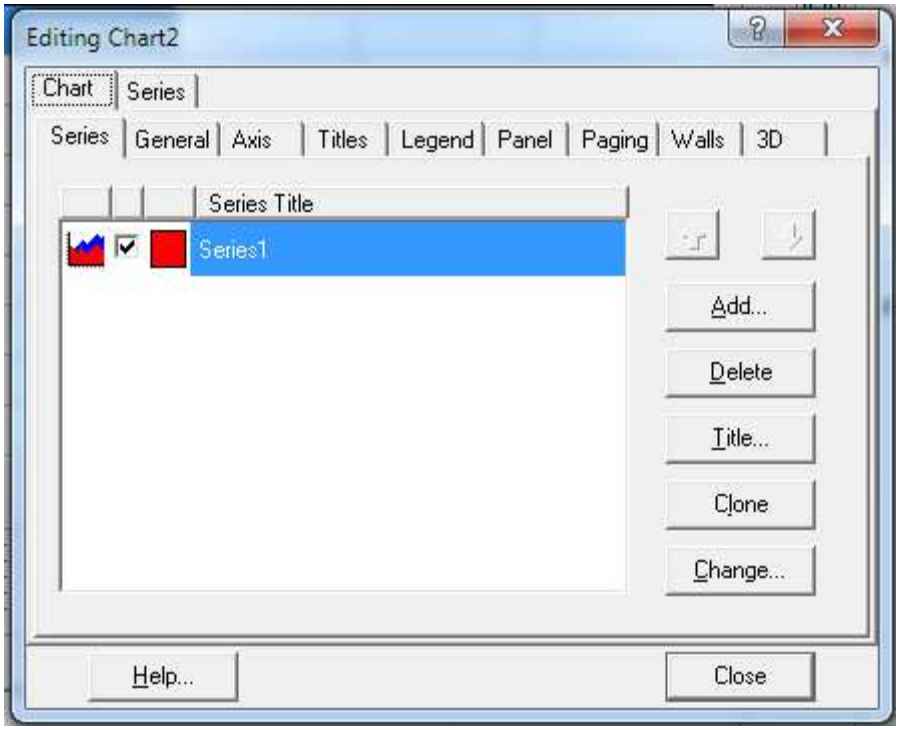


Рис. 3.12 Вікно редактора компонента TChart

Вкладка Chart

Основні параметри діаграми визначаються на вкладці ***Chart*** (діаграма), вона, у свою чергу, складається з набору додаткових вкладок.

Вкладка ***Series*** (ряд даних) призначена для вибору типу (форми) відображення ряду даних на діаграмі. Щоб додати на діаграму серію, слід натиснути кнопку Add. Після цього з'явиться вікно вибору типу серії (рис. 3.13).

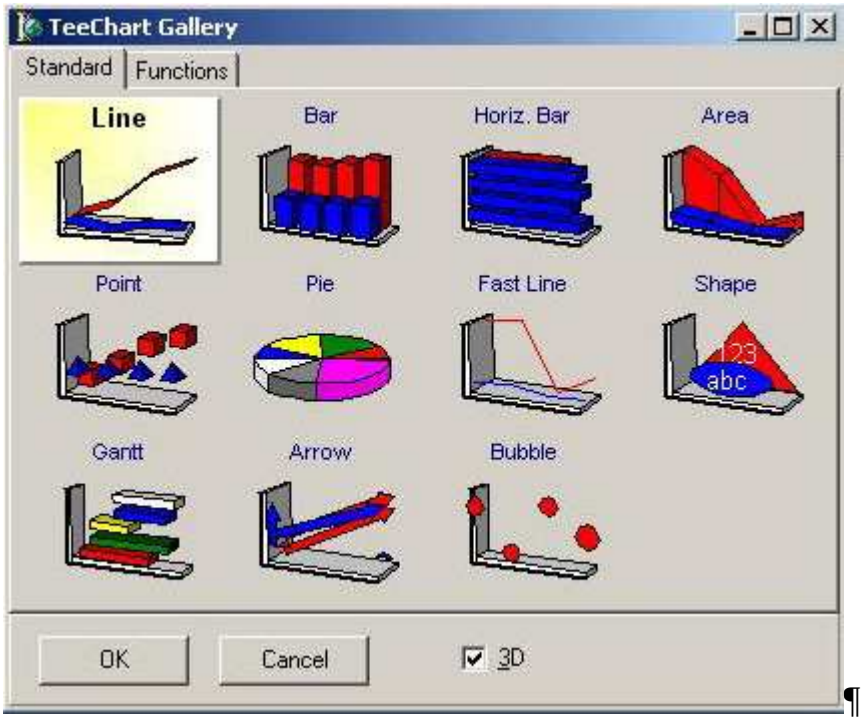


Рис.3.13. Вікно вибору типу серії

Вкладка ***General*** (Загальні*)* встановлює загальні параметри діаграми, такі як об'ємність, відступи від країв, можливість збільшення (Zoom) та ін. Панель General містить такі елементи управління:

• кнопка Export - експорт зображення у файл,

• кнопка Print Preview - попередній перегляд і друк діаграми,

• панель Zoom - масштабування,

• панель AllowScroll - відповідає за прокрутку зображення.

Засоби вкладки ***Axis*** (Осі) регулюють налаштування координатних осей, їх заголовків, їх масштаб, крок пунктирної сітки тощо. В області ShowAxis визначається, для якої осі встановлюються параметри – лівої, правої, верхньої або нижньої. На вкладці *Scales*, встановлюються властивості масштабу значень по осі. *Automatic* встановлює автоматичне масштабування даних по осі - мінімум і максимум обчислюються динамічно, виходячи з поточних значень серії. При скасуванні автоматичного масштабування можна встановити автоматичне масштабування мінімального (Minimum) або максимального (Maximum) значення (відмітка Auto).

Вкладка ***Titles*** (Заголовків) допомагає оформити заголовки.

Вкладка ***Legend*** (Легенда) відповідає за зовнішній вигляд і зміст легенди. Легенда – область графіка, де наводиться інформація для пояснення графіка.

Вкладка ***Panel*** (Панель) задає оформлення основи діаграми (підкладки): колір і форму границь панелі.

Вкладка ***Paging*** (Сторінки), дозволяє розділити діаграми на сторінки. Для цього необхідно в полі Points per Page (точки на сторінці) підібрати відповідне значення.

Вкладка ***Walls*** (Рамка) визначає кольорове та стилістичне вирішення рамки, що міститиме діаграму.

Вкладка ***3D*** дає можливість простим пересуванням повзунків управління налаштовувати 3D ефекти: зміна масштабу, положення в просторі тощо.

Вкладка Series

Після того як на вкладці Chart було обрано тип ряду даних на діаграмі необхідно задати безпосередньо ряди даних (серію) діаграми – набір точок, якому відповідає окремий стовбець або лінія. Кожна серія це компонент – об’єкт класу TChartSeries, точніше одного з його нащадків. Компонент TChartSeries є батьківським типом для TLineSeries, TAreaSeries, TPointSeries, TBarSeries, THorizBarSeries, TPieSeries, TChartShape, TFastLineSeries, TArrowSeries, TGanttSeries, TBubbleSeries.

Для формування та оформлення ряду даних призначена вкладка Series, яка, в свою чергу, також містить з декілька вкладок.

Найбільш важлива з них ***Data Sourece*** (Джерело даних). На ній можна визначається тип джерела даних: No Data - відмова від генерації значень; Random Values - створити випадкові значення; Function сформувати значення, як результат застосування функції до значень інших вже занесених рядів. На діаграмі можуть розміщуватись декілька графіків. Наприклад, для кожного місяця року необхідно побудувати графіки, що відображають витрати цементного розчину для об’єктів будівництва по будівельної організації в цілому. Кількість графіків відповідає кількості об’єктів К, на кожному графіку відображується помісячний обсяг витрат. Використавши функцію ***Average*** і вказавши К вихідних рядів отримаємо графік, що покаже витрату розчину в кожному місяці року окремо для кожного будівельного об’єкта. За замовчанням встановлено тип джерела – Random Values. Коли в режимі проектування програми було визначено форму відображення серії даних, в компоненті відразу відображається довільно сформований ряд даних. Це – результат роботи джерела Random Value.

Інші властивості компонентів TChart та TChartSeries наведені у

додатку Б.

Приклад, створення діаграми під час виконання програми

Крім того, що діаграми можна створювати під час проектування форми, їх також можна створювати під час виконання програми. Приклад, створення діаграми для функції $y = 2x^2$.

```
var chart:TChart;
series: TLineSeries;
i:integer;
begin
 chart:=TChart.Create (Form1);          //Створення діаграми
 With chart do
   begin
     Parent:=Form1;
     Left:=0;                            // Визначення розміру діаграми
     Top:=0;
     Width:=500;
     Height:=400;
     View3D:=False;
   end;
With chart.Title.Text do
   begin
       Clear;
       Add ('Графік функції y=2*x*x');  // заголовок діаграми
   end;

 series:=TLineSeries.Create(chart);  // створення серії у вигляді лінії
 series.Clear;
 for i:=-20 to 20 do
     series.AddXY(i, 2*i*i, '', clRed);  // наповнення серії значеннями

 chart.AddSeries(series);                // додавання серії у діаграму
 Visible:=True;                          // візуалізація графіку
end;
```

Результати роботи представлені на рисунку 3.14.

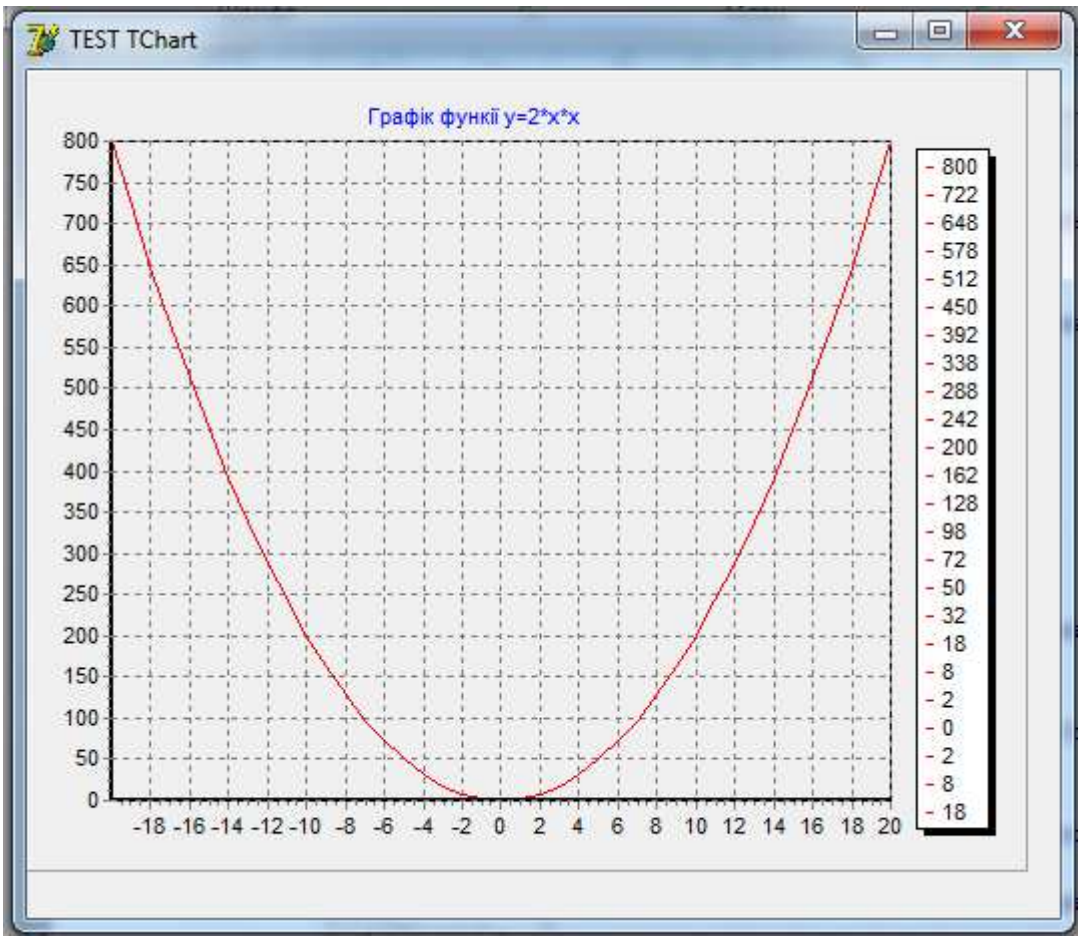


Рис.3.14 Графік функції виконаний з використанням компонента TChart

Висновки

- Відеоадаптер працює в одному з двох режимів: текстовому та графічному.
- В текстовому режимі екран розбитий на рядки та стовпчики, на зразок матриці, в кожну комірку якої можна вивести лише один символ з обмеженого набору.
- Шрифти, які використовуються в текстовому режимі роботи екрану називаються растровими.
- Кольори шрифтових знаків (символів) встановлюються трійкою елементарних кольорів RGB, сполучення яких та біта яскравості (мерехтіння) визначають інші кольори та відтінки.
- В графічному режимі екран розбивається на множину точок – пікселів. Кількість пікселів по горизонталі і вертикалі, а також кількість кольорів, які можуть використовуватися встановлюється також відповідним режимом відеоадаптера.
- Відомі два основних способи опису інформації про контур довільних графічних об’єктів: контурно-векторний, контурний.

- В графічному режимі для формування кольору пікселів застосовується RGB-модель, але на кожну складову RGB відводиться не по одному, а по декілька бітів.
- Колір графічних об’єктів може визначатися двома складовими: колір, стиль та товщина контуру; колір та стиль заповнення (заливки).
- При побудові зображень в екранній системі координат необхідно враховувати три фактори: початок координат (0, 0) знаходиться в лівому верхньому куті екрану; екранна ось ординат ОУ спрямована до низу; координати вимірюються в пікселах.
- В ОС Windows формування графічних зображень і передача їх на пристрої відображення реалізується в підсистемі GDI, яка забезпечує однакові принципи формування зображень для кожного конкретного пристрою виведення (монітор, принтер тощо).
- Кожен тип графічного пристрою описується спеціальним об’єктом GDI, який називається «контекст пристрою» (Device Context, DC). В ОС Windows кожен DC визначається дескриптором (handle Device Context, hDC), за допомогою якого програми можуть управляти роботою пристрою.
- Основою графічної підсистеми Delphi є клас TCanvas, який об’єднує у собі полотно – робочу поверхню для малювання, робочі інструменти (перо Pen, пензель Brush, шрифт Font), а також набір функцій для побудови графічних примітивів (ліній, еліпсів та їх секторів, хорд, сегментів, прямокутників, ламаних, багатокутників).
- Компоненти TChart та TChartSeries є потужним та гнучким інструментом для створення під час проектування форми або під час виконання програми різноманітних за формою та змістом діаграм та графіків.

Запитання та завдання для самоперевірки

1. Які режими роботи екрану комп’ютера ви знаєте?
2. Чим характеризується текстовий режим роботи екрану?
3. Який принцип лежить в основі формування шрифтів в режимі текстового режиму роботи екрану?
4. Який принцип лежить в основі формування кольорів в режимі текстового режиму роботи екрану? Що таке байт-атрибут символу?
5. Як в текстовому режимі формуються графічні зображення?
6. Охарактеризуйте особливості графічного режиму роботи екрану.
7. Які способи опису контуру графічних об’єктів в графічному режимі

роботи екрану ви знаєте? Що в них спільного, а в чому різниця?

8. Що таке растеризація зображення? Для чого вона виконується?
9. Як досягається реалістичність зображення в графічному режимі роботи екрану?
10. Якими складовими визначається колів графічних об’єктів в графічному режимі?
11. Які особливості у екранної системи координат?
12. Як необхідно коригувати природні координати графічних об’єктів при виведенні на екран?
13. Яки принципи побудови зображення відносно екранного початку координат та відносно довільно визначеного?
14. Що таке «дескриптор контексту пристрою» і як він використовується?
15. Охарактеризуйте клас TGraphicControl. Які ви знаєте нащадки цього коасу серед візуальних компонентів VCL?
16. Охарактеризуйте клас TCanvas. Які властивості та методи для оьробки пікселів зображення він надає?
17. Які обробники подій TCanvas?
18. Яке призначення мають класи класів TPen, TBrush, TFont?
19. Як клас TCanvas позиціонує початкову точку виводу зображення?
20. Які графічні примітиви реалізує клас TCanvas?
21. Які методи класу TCanvas призначені для рисування ліній? Наведіть приклад виводу на екран прямого відрізка.
22. Які методи класу TCanvas призначені для рисування еліпсів? Як за допомогою метода Ellipse нарисувати круг?
23. Що спільного та в чому різниця у використанні методів Pie, Arc та Chord?
24. Чим відрізняються методи Rectangle та RoundRect? Як визначається заокруглення на RoundRect?
25. Чим відрізняються принципи роботи методів Polyline та Polygon?
26. Охарактеризуйте призначення та принципи використання компоненту TChart?
27. Які параметри діаграми дозволяє задавати компонент TChart?
28. Що таке серії даних (наведіть приклад), яку форму вони можуть набувати при використанні компоненту TChart?
29. Що таке серія даних? За допомогою яких методів компоненту TChartSeries відбувається наповнення серії даними?
30. Які зміни потрібно внести в наданий лістинг програми, щоб компонент

відобразив графік функції y=sin(x).

Розділ 4. Застосування принципів ООП в прикладних задачах будівельної галузі

Будівельна галузь характеризується розмаїттям інженерних задач, задач планування та управління виробництвом. До розмаїття задач додається розмаїття їх постановок, моделей та методів їх розв'язку.

Визначення координат центра тяжіння складного перерізу

В інженерній практиці виникає необхідність визначати координати центру тяжіння складного перерізу, що складається з простих елементів, для яких розташування центру тяжіння відомо. Така задача є частиною задачі визначення геометричних характеристик складових поперечних перерізів балок і стержнів. З подібними питаннями доводиться стикатися проектувальникам будівельних конструкцій при підборі перерізів елементів, а також студентам при вивченні дисциплін «Теоретична механіка» і «Опір матеріалів».

Координати центру тяжіння складного перерізу X^c, Y^c визначаються за формулами:

$$X^C = \frac{\sum x_i^c \cdot S_i - \sum x_j^c \cdot R_j}{\sum S_i - \sum R_j}, \qquad Y^C = \frac{\sum y_i^c \cdot S_i - \sum y_j^c \cdot R_j}{\sum S_i - \sum R_j}, \qquad (1)$$

де x^c_i, y^c_i, S_i – координати центру тяжіння та площа і-го елементу, що входить до складного перерізу; x^c_i, y^c_i, R_i – координати центру тяжіння та площа j-го елементу, який є розривом або отвором в перерізі (рис. 4.а).

Узагальнений алгоритм визначення координат центру тяжіння складного перерізу буде наступним:

1. розбиття складного перерізу на прості елементи;
2. визначення просторових характеристик кожного простого елемента у перерізі відносно заданого центру координат;
3. визначення геометричних характеристик простих елементів перерізу (площа та координати центру тяжіння);
4. розрахунок координат центру тяжіння всього складного перерізу.

Функціональна декомпозиція процесу визначення координат центру тяжіння складного перерізу подана на рис. 4.1.

Рис. 4.1 «Дерево функцій» визначення центра тяжіння складного перерізу

Аналіз простих елементів, з яких може складатися переріз

Розглянемо, на які прості елементи може розбиватися складний переріз. На рисунку 4.2.а складний переріз утворюють 5 простих елементів – плоскі геометричні фігури (базові графічні примітиви, БГП): прямокутники (1, 2), трикутник (3), круги (4, 5), що визначають отвори на другому прямокутнику. Параметри прямокутників та трикутника необхідно буде додавати, а кругів – віднімати під час обчислення за формулою (1).

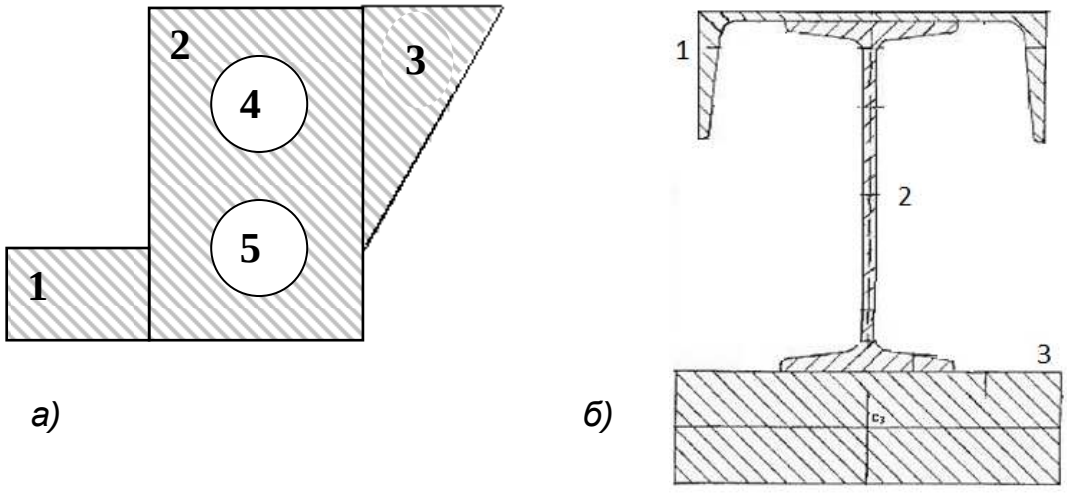


Рис. 4.2 Складній переріз: а) – базові графічні примітиви; б) – БГП та стандартні профілі прокату

В будівельній практиці складний переріз окрім плоских геометричних фігур може складатися зі стандартних профілів прокату

(СПП). На рисунку 4.2.б складний переріз включає швелер (1), двотавр (2) та прямокутну полосу (3). Геометричні характеристики СПП (площа перерізу та координати центра тяжіння) визначаються за відповідними сортаментами.

Отже з точки зору об'єктної декомпозиції предметної області для вирішення задачі визначення координат центру тяжіння складного перерізу можна виділити такі основні сутності – кандидати на створення класів (рис. 4.3):

- складний переріз, який включає список простих елементів і для якого розраховуються координати центру тяжіння за формулами (1).
- прості елементи перерізу, які можуть бути ("is-a") базовим графічним примітивом або стандартним профілем прокату. Кожен простий елемент має свої координати точки прив'язки (x0, y0) та методи для визначення площі Square() та координат центру перерізу Xc(), Yc(). Для БГП площа та координати центру тяжіння розраховується. Для СПЕ ці параметри визначаються за сортаментом.

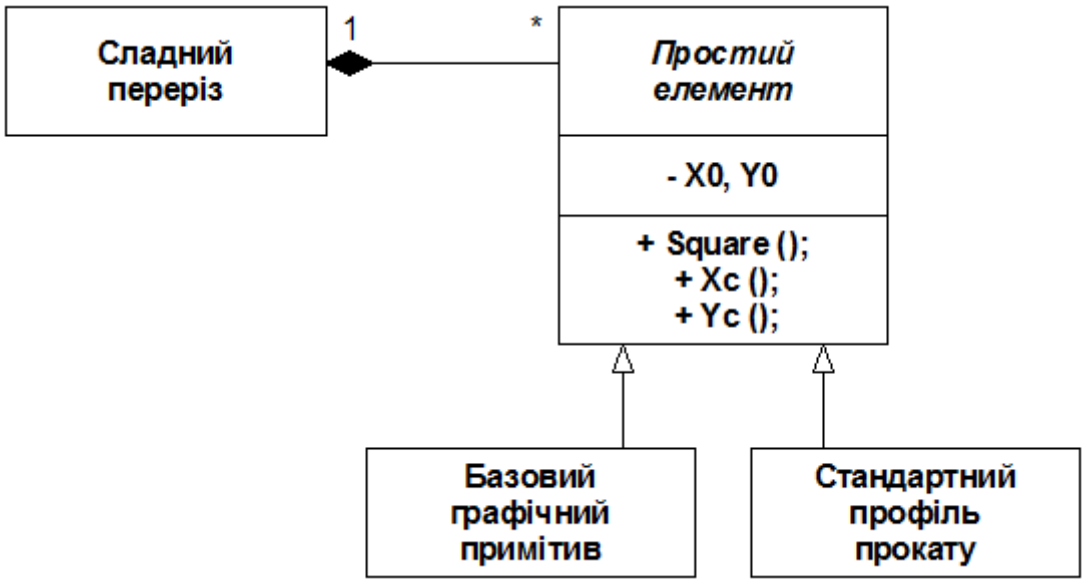


Рис. 4.3 Проект діаграми класів для складного перерізу

Надалі при формуванні складного перерізу будуть розглядатися тільки базові графічні примітиви. До базових графічних примітивів, з яких може складатися переріз, відносяться: прямокутник (квадрат), паралелограм (ромб), трапеція, сектор круга, круг (який можна розглядати як сектор з кутом 360 градусів), еліпс, трикутник, правильний багатокутник. Зі шкільного курсу геометрії відомо, як визначити площі

цих фігур та координати центру тяжіння.

В якості приклада для подальшого розгляду візьмемо круг, прямокутник та трикутник, тому що всі інші БГП можуть бути поділені на ті, що визначені. Наприклад, трапеція або паралелограм можуть бути розбиті на два трикутники або на два трикутники та прямокутник. В таблиці 4.1 наведені геометричні характеристики фігур та формули для розрахунку їх площ та координат центрів тяжіння.

Таблиця 4.1 – Характеристики БГП

Геометричні характеристики	Площа та координати центру тяжіння
Круг (TCircle)	
координати центру: (X_0, Y_0); радіус R.	$S = \pi R^2$ $X_c = X_0 \quad Y_c = Y_0$
Прямокутник (TRect)	
Координати двох протилежних вершин: $(X_0, Y_0), (X_1, Y_1)$	$S = \lvert X_0 - X_1 \rvert \cdot \lvert Y_0 - Y_1 \rvert$ $X_c = \frac{X_0 + X_1}{2} \quad Y_c = \frac{Y_0 + Y_1}{2}$
Трикутник (TTriangle)	
Координати трьох вершин: $(X_0, Y_0), (X_1, Y_1), (X_2, Y_2)$	$S = \sqrt{p(p-a)(p-b)(p-c)},$ $p = \frac{a+b+c}{2},$ де a, b, c – довжини сторін трикутника (необхідний метод для розрахунку довжини сторони за координатами вершин). $X_c = \frac{X_0 + X_1 + X_2}{3} \quad Y_c = \frac{Y_0 + Y_1 + Y_2}{3}$

Аналіз функціональних вимог до програми визначення координат центра тяжіння складного перерізу

Програма для визначення координат центра тяжіння складного перерізу окрім функцій, що були подані на рис. 4.1, повинна проводити графічне відображення складного перерізу, що задав користувач – проектувальник. Функціональна декомпозиція програми наведена на рис.

4.4. Основними функціями є:

1. Формування складного перерізу.

1.1. Введення просторових характеристик простих елементів перерізу (прямокутників, кіл, трикутників) та ознаки чи є елемент отвором (розривом).

РИСУНОК 4.4.

1.2. Додавання елементів до перерізу.

2. Розрахунок координат центру тяжіння (КЦТ) складного перерізу.

2.1. Розрахунок геометричних характеристик елементів перерізу (площі елемента, КЦТ елемента).

2.2. Розрахунок площі та КЦТ всього перерізу.

3. Графічне відображення складного перерізу.

3.1. Відображення осей координат та сітки.

3.2. Відображення перерізу.

3.2.1. Відображення елементів перерізу.

3.2.2. Відображення отворів та розривів.

3.2.3. Відображення центру тяжіння елементів.

3.2.4. Відображення центру тяжіння всього перерізу.

Діаграма прецедентів програми визначення координат центра тяжіння складного перерізу наведена на рис. 4.5.

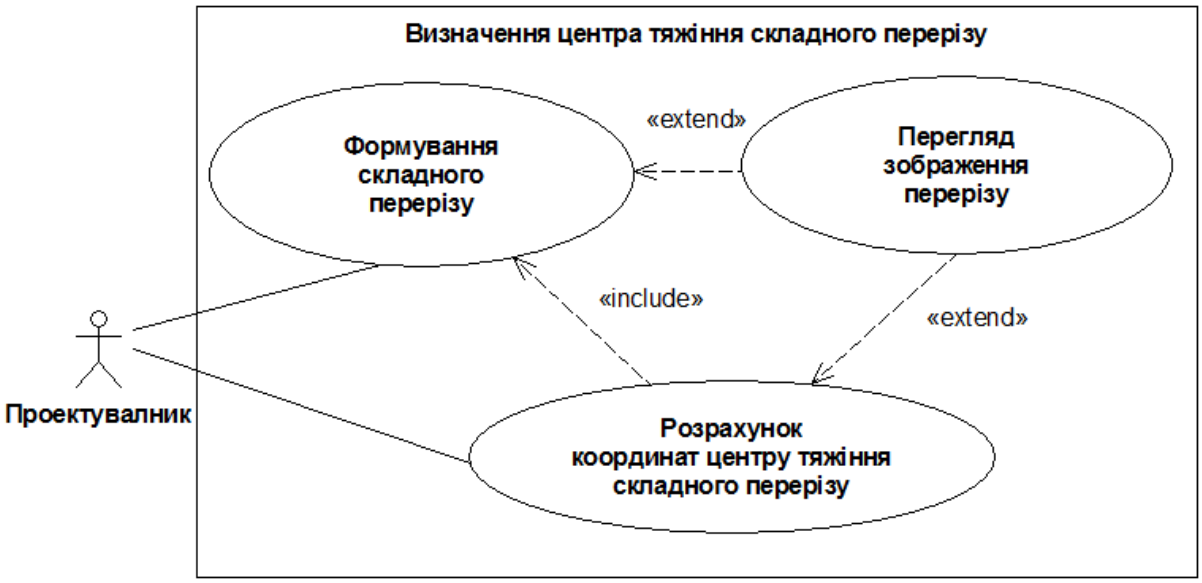


Рис. 4.5. Діаграма прецедентів

Проектування об’єктної моделі задачі визначення КЦТ складного перерізу

Виходячи з аналізу задачі визначення координат центру тяжіння складного перерізу можна запропонувати таку об’єктну модель: клас – складний переріз TCompSection, що складається з простих елементів – базових графічних примітивів TBGP. Характер зв’язку між цими класами може бути визначений як композиція, тому що простий елемент існує тільки як складова всього перерізу.

Фактично клас складного перерізу TCompSection є класом-контейнером для простих елементів перерізу. Крім того для кожного елемента зі списку необхідно зберігати ознаку, в залежності від якої треба буде додавати чи віднімати параметри даного елемента під час обчислення координат центру тяжіння всього перерізу. Для створення класу складного перерізу можна скористатися стандартним класом-контейнером TStringList, утворивши від нього TCompSection як нащадка. При цьому у властивості Objects, що спадкується від TStringList, будуть зберігатися об’єктні посилання на елементи перерізу, а у пов’язаній з Objects властивості Strings може зберігатися ознака щодо того, чи є даний елемент отвором (розривом).

Клас TBGP має «втілення», може бути представлений кругом, прямокутником та трикутником. Отже TBGP є спільним предком для класів TCircle, TRect, TTriangle. Цей предок є абстрактним, тому що не може мати реалізації методів для розрахунку площі та координат центру, кожен нащадок цього класу повинен мати власну реалізацію цих методів.

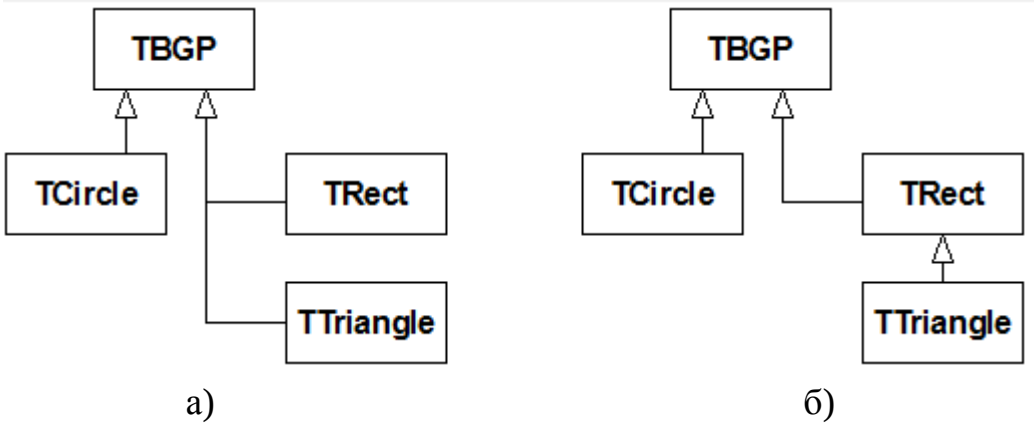
Виходячи з опису властивостей класів, що наведені у таблиці 1 властивостями базового класу TBGP є:

- координати точки прив’язки Fx0, Fy0, які вводяться з директивою видимості private. Для доступу до цих полів призначені відповідні методи читання / запису getX / setX, getY / setY з директивою видимості protected;
- конструктор, що ініціалізує поля класу значеннями nx, ny;
- методи для розрахунку площі та КЦТ.

Аналізуючи зв’язки між класом-предком та його класами-нащадками можна запропонувати два варіанти зв’язків:

- TCircle «is-a» TBGP, TRect «is-a» TBGP, TTriangle «is-a» TBGP (рис. 4.6.а);
- спираючись на принцип розширення спільних властивостей нащадка

відносно предка (таблиця 4.1), з точки зору реалізації класів зручним буде спадкування трикутника від прямокутника: до властивостей, що набув прямокутник додаються координати ще однієї вершини . Але таки підхід дещо суперечить змісту зв'язку: TTriangle «is-a» TBGP, але TTriangle «is-**NOT**-a» TRect (рис. 4.6.б).



а) б)

Рис. 4.6 Варіанти побудови діаграми класів: а) спадкування з дотриманням принципу «is-a»; б) спадкування з порушенням формального принципу «is-a» за для зручності реалізації класів.

Виходячи з функцій системи, програма повинна надавати можливість візуалізації складного перерізу на екрані. При побудові такого зображення необхідно виводити на екран зображення координатних осей та проводити їх розмітку. Клас TCoordField буде мати у своєму складі:

- поля: Picture – об'єктне посилання на графічний компонент – поле виведення, у властивості якого Canvas буде будуватися зображення; Delta – відступ від краю поля виведення до кінців осей; Cx, Cy – координати центру початку координат в полі виведення Picture; Scale – кількість пікселів на одиницю ділення по осі;
- методи: конструктор та рисування Draw.

Слід відмітити, що проектування об'єктної моделі предметної області, що подана на рисунку 4.7 у вигляді UML-діаграми класів, – творчий процес. Запропонована об'єктна модель не є єдиною правильною для даної задачі. Можливо запропонувати ще декілька варіантів реалізації за участю тих класів, що були визначені, або з додаванням інших чи вилученням з існуючих.

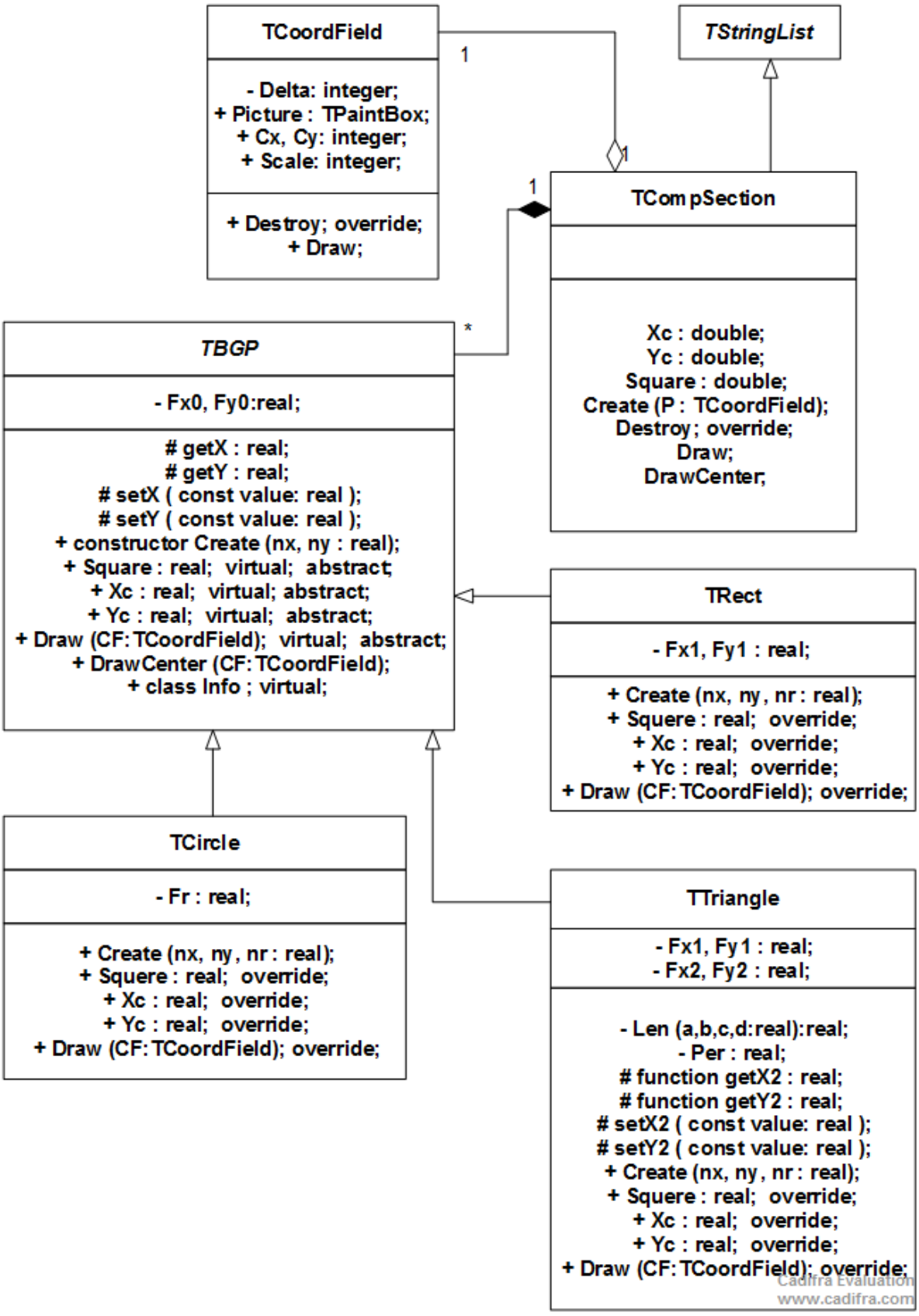


Рис. 4.7 Діаграма класів програми визначення координат центра тяжіння складного перерізу

Приклад роботи програми визначення координат центра тяжіння складного перерізу

Головне вікно програми визначення координат центра тяжіння складного перерізу наведено на рис. 4.8

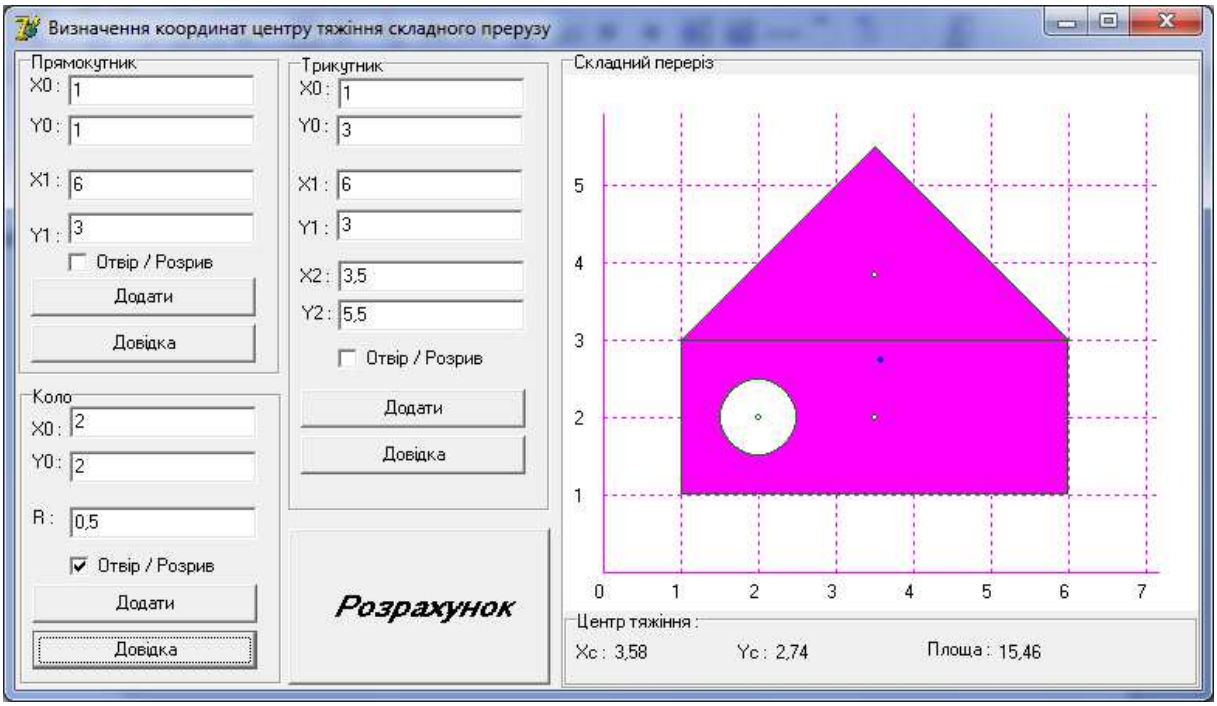


Рис. 4.8. Головне вікно програми визначення координат центра тяжіння складного перерізу.

В якості прикладу складний переріз включає три прості елементи, Координати яких задані в полях робочого вікна. На зображенні перерізу позначені КЦТ елементів (світлі точки) та всього перерізу (синя очка). Числові значення КЦТ та площі всього перерізу виведені під його графічним зображенням.

Календарне планування робіт

Будівництво в сучасному світі – це багатоетапний і складний процес як з технічної, так і з правової сторони, що вимагає багатьох узгоджень, регламентується великою кількістю нормативної документації, потребує участі великої кількості фахівців різних областей та використання різноманітних машин, механізмів, конструкцій та матеріалів.

Одним з основних документів будівельного виробництва є календарний план.

Календарний план - це проектно-технологічний документ, який визначає послідовність, інтенсивність і тривалість виконання робіт на будівельному об’єкті, їх взаємозв'язки (послідовність їх виконання), а також потребу (з розподілом у часі) в матеріальних, технічних, трудових, фінансових та інших ресурсах, що використовуються при виконанні визначених робіт [5].

Головним завданням календарного планування будівництва окремих об'єктів є визначення такої черговості та послідовності виконання робіт, які забезпечують здачу об’єкту замовникам в експлуатацію в договірні чи планові терміни.

Таким чином, при складанні календарного плану будівництва об’єкту виконується порівняння методів виконання робіт і вибирається той метод, який за даних конкретних умовах найбільш прийнятним. Для цього процес будівництва об'єкта представляється у вигляді моделі, за допомогою якої і аналізуються всі можливі виробничі ситуації.

Параметрами календарного плану в найпростішому варіанті є номенклатура (перелік) робіт із зазначенням для кожної роботи дати початку та закінчення, її тривалості та розрахунок необхідних ресурсів. Ці розрахунки виконуються окремо по кожній роботі, для різних періодів будівництва, а також для об’єкту в цілому.

У будівельній практиці для забезпечення ефективного планування та управління використовують лінійні, матричні та сітьові моделі.

Лінійні календарні графіки є найбільш простою та доволі широко застосовуваною в будівництві організаційно-технологічною моделлю календарного плану. Вони можуть бути подані у формі графіків Ганта і циклограм.

Графік/діаграма Ганта - це інструмент управління, який забезпечує графічне відображення плану робіт, зручне для контролю та відстеження прогресу виконаних завдань.

Лінійний календарний план у формі графіка Ганта містить (див. рис. 4.9):

- по осі ординат - перелік видів робіт, розташованих у технологічній послідовності, та, за потреби, їх характеристики (наприклад, обсяги, вартість, трудомісткість, склад виконавців);

- по осі абсцис - прийняті порядкові або календарні одиниці часу (дні, місяці), що охоплюють весь період виконання робіт.

Кожна смуга на діаграмі представляє окрему роботу в складі проекту, її кінці - моменти початку і завершення роботи, її протяжність - тривалість роботи. Крім того, на діаграмі можуть бути відзначені сукупні завдання, відсотки завершення, покажчики послідовності і залежності робіт, мітки ключових моментів (віхи), мітка поточного моменту часу «Сьогодні» та ін.

Лінійний календарний графік у вигляді діаграми Ганта наочний та простий у побудові.

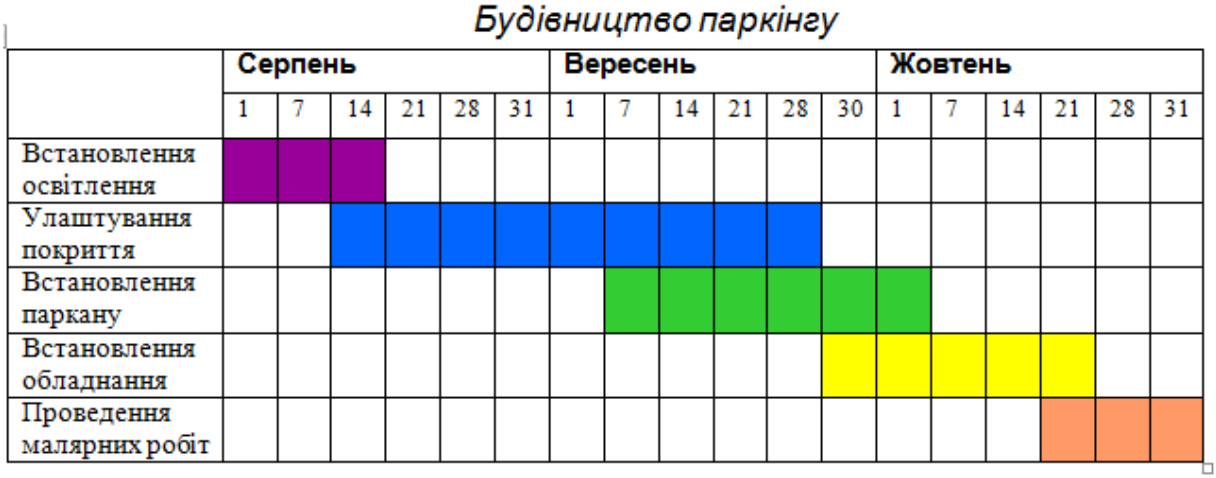


Рис. 4.9. Приклад діаграми Ганта для задачі «Будівництво паркінгу»

Однак, вирішуючи завдання своєчасної здачі об'єктів будівництва замовникам, необхідно також враховувати обмеження за наявними виробничими ресурсами і передбачати найбільш повне і раціональне їх використання. Тому іншими не менш важливими завданнями календарного планування є: раціональне використання наявних виробничих ресурсів, облік наявних обмежень на поставку матеріальних, технічних та інших ресурсів. Тому, одночасно у взаємозв'язку з календарним планом будівництва, формуються і календарні плани (графіки) використання необхідних ресурсів. Це, так звані, епюри потреби в трудових ресурсах, засобах механізації, конструкціях, виробах, матеріалах.

Епюри ресурсів наочно показують рівень потреби, наявності ресурсів, дають уявлення про рівномірність їх споживання.

Ресурси можуть бути відтворювані (поновлювані) та не відтворювані (не поновлювані).

Матеріальні ресурси - предмети і засоби праці: сировина, матеріали, енергія, запасні частини машин і обладнання та інші – все не поновлювані ресурси. До поновлюваних ресурсів відносять людські ресурси, машини і механізми.

Ресурсні графіки/епюри будуються в осях координат X – час, Y - сумарна потреба робіт проекту в ресурсі.

Нехай R_{ki} - інтенсивність споживання k-го ресурсу *i*-ю роботою, або іншими словами потреба *i*–ї роботи в *k*–му ресурсі в кожний момент її виконання.

З огляду на те, що паралельно можуть виконуватись багато робіт, узагальнена потреба проекту в ресурсі *k* на заданий момент часу *t* буде визначатись за формулою:

$$W_{kt} = \sum_{i \in Rt} R_{ki} \qquad (4.1)$$

де, R_t – множина робіт проекту, що виконуються в момент часу *t*.

Загальна потреба всього проекту в k-му не відновлюваному ресурсі дорівнює

$$W_k = \sum_t W_{kt} \qquad (4.2)$$

Для ресурсів відновлюваного типу загальна потреба всього проекту в n-му ресурсі дорівнює

$$W_n = MAX(W_{nt}) \qquad (4.3)$$

Узагальнена технологія опрацювання календарного плану проекту будівництва будь-якого об’єкту в рамках навчального прикладу складається з наступних кроків:

1. Визначення описових характеристик об’єкту:
   - назва будівельного об’єкту;
   - назва провідної будівельної організації.
2. Формування списку робіт проекту.

   Для кожної роботи потрібно зазначити:
   - описову назву роботи;
   - дату початку роботи;

- дату закінчення роботи;
- потребу роботи в ресурсах.

3. Розрахунок загальних параметрів проекту:
   - дати початку проекту;
   - дати закінчення проекту;
   - загальної тривалості проекту;
   - мінімальної потреби в трудових ресурсах;
   - максимальної потреби в трудових ресурсах.
4. Розрахунок даних для побудови епюр потреби в ресурсах (див. формула 4.1).
5. Побудова графіку потреби в трудових ресурсах.
6. Побудова графіку Ганта.
7. Далі на основі аналізу одержаних розрахункових даних може виконуватись корекція характеристик робіт з метою одержання бажаних характеристик проекту, наприклад, більш рівномірної епюри потреби в ресурсах.

Аналіз функціональних вимог до програми опрацювання календарного плану проекту будівництва об'єкту

Виходячи з аналізу вищезазначеної технології та з огляду на зручність і доцільність роботи з даними, програмний продукт опрацювання календарного плану проекту будівництва об'єкту повинен реалізувати у своєму складі такі функції(див. рис. 4.10):

1. Опрацювання інформації про проект:
   1.1. Створення нового проекту/календарного плану та визначення його описових характеристик.
      1.1.1. назви об'єкту, що будується;
      1.1.2. назви організації – головного виконавця будівництва.
   1.2. Введення/коригування описових характеристик проекту.
   1.3. Завантаження вже існуючого проекту з файлу.
   1.4. Збереження даних проекту на зовнішньому носії.
2. Формування списку робіт, що виконуються на об'єкті.
   2.1. Додавання роботи у список
   2.2. Видалення роботи зі списку
   2.3. Корекція параметрів роботи:
      2.3.1. назви роботи;
      2.3.2. дати початку роботи;

2.3.3. дати закінчення роботи;

2.3.4. потреби роботи в ресурсах.

3. Розрахунок параметрів календарного плану.

3.1. розрахунок загальних параметрів.

3.1.1. дати початку будівництва об’єкту;

3.1.2. дати завершення будівництва об’єкту;

РИСУНОК 4.10

3.1.3. загальної тривалості будівництва;

3.1.4. величини максимальної потреби в заданому ресурсі;

3.1.4. величини мінімальної потреби в заданому ресурсі.

3.2. розрахунок даних для епюри потреби в ресурсі.

4. Графічне відображення розрахованих параметрів.

4.1. Відображення загальних параметрів.

4.2. Відображення епюри потреби в ресурсі.

4.3. Відображення лінійного графіку виконання робіт у вигляді діаграми Ганта.

Діаграма прецедентів програми опрацювання календарного плану проекту будівництва об’єкту наведена на рис. 4.11.

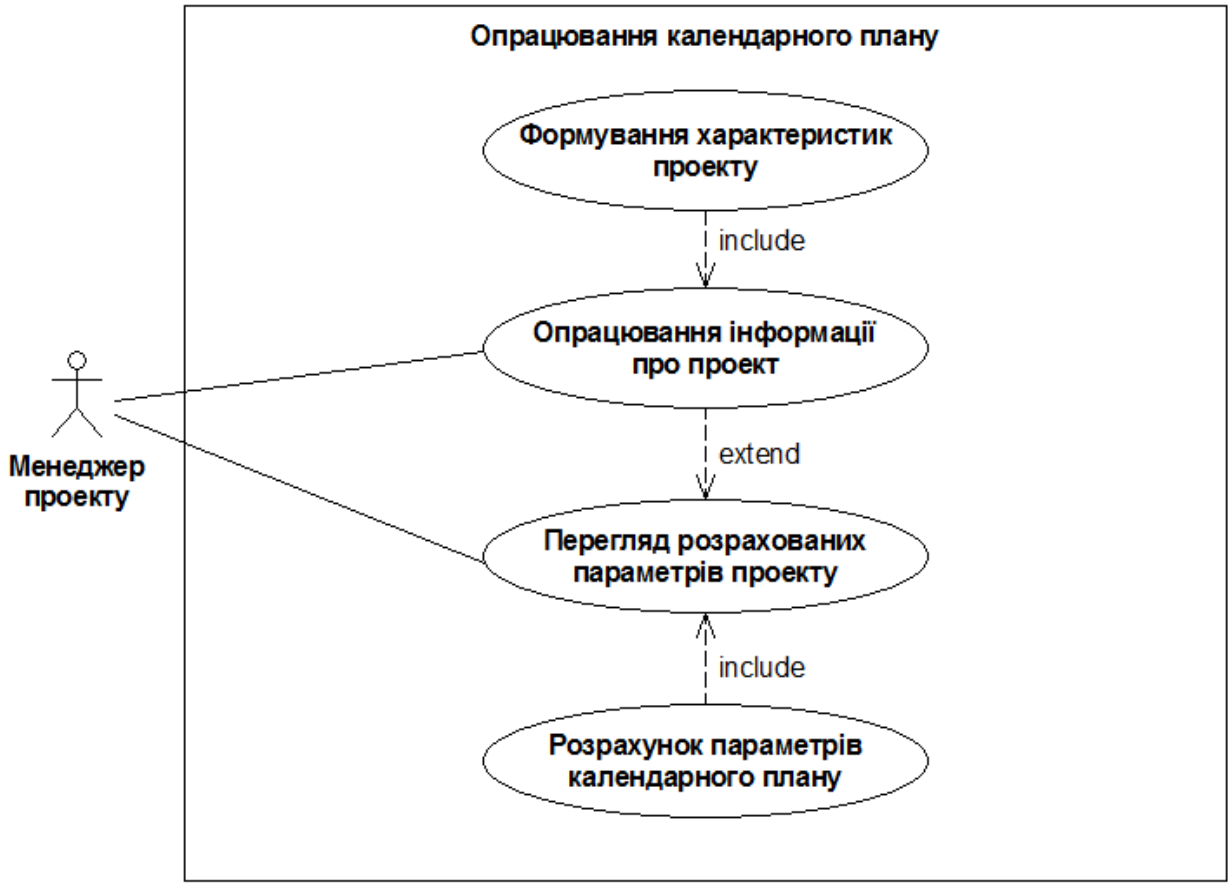


Рис. 4.11. Діаграма прецедентів програми опрацювання календарного плану проекту будівництва об’єкту

Проектування об’єктної моделі задачі опрацювання календарного плану проекту будівництва об’єкту

Проведений аналіз поставленої задачі показав, що первинною інформацією в задачах календарного планування є номенклатура (перелік) робіт, що мають бути виконані для будівництва заданого об’єкту. Тому центральним класом об’єктної моделі задачі буде клас TWork (рис.4.12), поля якого визначають для кожної роботи її назву, дати початку та закінчення, об’єм трудових ресурсів необхідних для її виконання. Для зручності роботи в класі для закритих полів визначені відповідні властивості та методи контролю коректності даних, що вводяться. Наприклад, дата початку роботи не може бути пізнішою дати закінчення роботи.

Оскільки план/проект будівництва, як зазначалось вище, складається з багатьох робіт, то при розробці класу TProject – класу, що описує наш проект, ми повинні подбати про те, щоб у цьому класі крім описових характеристик (назви об’єкту, що будується, та назви провідної будівельної організації) містився список його робіт.

За об’єднання окремих об’єктів робіт в єдиний список робіт проекту відповідає клас TWorkList. Як видно з діаграми, цей клас є нащадком стандартного класу TList об’єктної моделі Delphi (див. рис.4.12 зв’язок узагальнення). Принцип спадковості ООП, описаний у розділі 1, забезпечує нашому класу одержання у спадок від класу TList множини корисних методів роботи зі списком об’єктів.

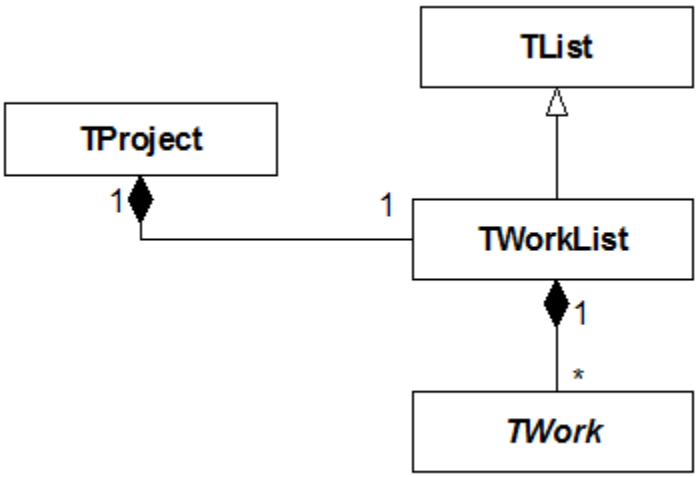


Рис. 4.12. Діаграма класів програми опрацювання календарного плану проекту будівництва об’єкту

Композиційний зв'язок один до багатьох (1 .. *) між класами TWorkList та TWork відповідно, вказує на те, що клас TWorkList забезпечує організацію об’єктів саме класу TWork у єдиний список (один список містить багато робіт) і підкреслює спеціалізовану спрямованість методів класу TWorkList на роботу з об’єктами класу TWork.

Зв'язок композиції між класами TProject та TWorkList (1 .. 1) демонструє включення класу TWorkList в клас TProject в якості невід’ємної частини. В програмному коді цей зв'язок реалізований наступним чином: клас TProject містить поле *FworkList : TWorkList*, що є об’єктом класу TWorkList.

Приклад роботи програми опрацювання календарного плану проекту будівництва об’єкту

Розглянемо роботу застосування на прикладі невеликого проекту з будівництва дачного будиночка. На головному вікні програми (рис.4.13), у лівій його частині розташовані всі характеристики проекту (описові + список робіт) та елементи управління для їх коригування. У правій частині розташовані область відображення розрахованих параметрів: епюра потреби в ресурсі, графік Ганта та загальні розрахункові параметри. Зміна характеристик робіт приводить до зміни виду діаграм та значень розрахованих параметрів.

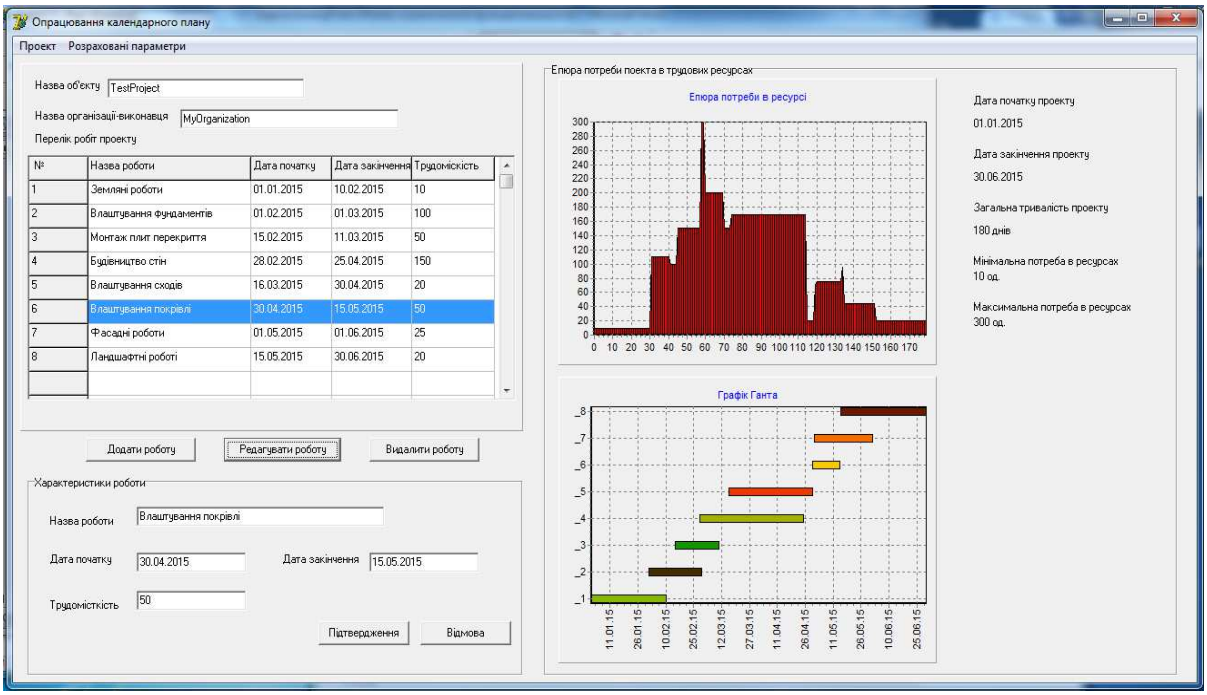


Рис. 4.13. Зовнішній вигляд головного вікна програми

Висновки

Розглянуті нами приклади продемонстрували як реалізуються засади ООП для вирішення конкретних задач. Чітке виокремлення понять предметної області - класів, опис їх взаємодії між собою, обмеження можливостей програміста заздалегідь продуманими реалізаціями методів класу роблять програми більш інтуїтивно зрозумілими, стійкими до появи некоректних спроб обробки інформації, максимізованими з точки зору повторного використання програмного коду.

Запитання та завдання для самоперевірки

В задачі визначення координат центру тяжіння складного перерізу

1. Якими додатковими класами-нащадками, з вашої точки зору, можна розширити TBGP?
2. Чи достатньо з вашої точки зору для одержання об’ємного вирішення задачі (визначення координат центру тяжіння об’ємної фігури) додати координату z в клас TBGP?
3. Прокоментуйте, який узагальнений алгоритм роботи реалізує метод Draw класуTCompSection.

В задачі опрацювання календарного плану проекту будівництва об’єкту:

4. Які зміни доцільно внести в проект, щоб забезпечити можливість визначення потреби роботі не в одному, а в декількох ресурсах?
5. Які зміни мають бути внесені в реалізацію класів, за умови, що на діаграмі класів (рис.4.12) зв’язки композиції будуть замінені на зв’язки агрегації?
6. Прокоментуйте алгоритм, що формує дані для побудови епюри потреби в ресурсі. Які методи яких класів для цього використовувались?

СПИСОК ЛІТЕРАТУРИ

1. Андрианова А.А., Исмагилов Л.Н., Мухтарова Т.М. Объектно-ориентированное программирование на C# : Учебное пособие / А.А. Андрианова, Л.Н. Исмагилов, Т.М. Мухтарова. – Казань: Казанский (Приволжский) федеральный университет, 2012. – 134 с.
2. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. \ Пер. с англ.- М.: “Издательство Вильямс ”, 2008 г. – 721 с. ил.
3. Дарахвелидае П. Г., Марков Е. П. Программирование в Delphi 7. — СПб.: БХВ-Петербург, 2003. — 784 с : ил.
4. Дарахвелидзе П.Г., Марков Е.П. Delphi – среда визуального программирования: СПб.:BHV – Санкт Петербург, 1996. – 352 с.
5. Ізмайлова О. В. Методи прийняття багатокритерійних рішень в інформаційних системах: Навч. посібник.-Київ: КНУБА, 2002. – 111с.
6. Кенту М. Delphi 5 для профессионалов. – СПб: Питер, 2001. – 944 с.
7. Конопка Рэй. Создание оригинальных компонент в Delphi. Пер. с англ. – К.: ДиаСофт, 1996-.512 с., ил.
8. Самарин Ю. Способы представления шрифтов в цифровом виде / Ю. 7 // КомпьюАРТ. - 2012. - №11. - С. 42-47. Електронний ресурс: http://www.compuart.ru/article.aspx?id=23461&iid=1079
9. Красовська Г.В., Демченко В.В. Об’єктно-орієнтоване програмування: Об’єктна модель Delphi: Конспект лекцій. [Електронний варіант] - К.:КНУБА, 2009 – 51 с. Режим доступу: http://org.knuba.edu.ua/mod/resource/view.php?id=5603

Посилання на джерела графічних зображень

10. http://shkamer.ru/wp-content/uploads/2012/02/jokonda.jpg
11. http://www.gamedev.ru/files/images/console_quest.jpg
12. http://upload.wikimedia.org/wikipedia/uk/thumb/b/bf/Norton_commander.png/350px-Norton_commander.png4.
13. https://upload.wikimedia.org/wikipedia/commons/thumb/2/28/RGB_illumination.jpg/330px-RGB_illumination.jpg
14. http://images.myshared.ru/531094/slide_18.jpg

ДОДАТКИ

Додаток А. Властивості невізуальних класів

Таблиця А.1 – Властивості та методи класу TList

Назва	**Призначення**
property Count : Integer;	Кількість елементів в списку
property Capacity;	Максимальна кількість елементів
property Items [Index : integer] : Pointer;	Список елементів
function Add (Item : Pointer) : Integer	Додає в кінець списку новий елемент
procedure Insert (Index : Integer; Item : Pointer);	Вставляє в список новий елемент за заданим індексом
procedure Delete (Index : Integer);	Видаляє елемент зі списку. Необхідно звернути увагу, що цей метод не звільнює пам’ять, що була виділена під елемент, а тільки звільнює покажчик на нього
procedure Remove (Item : Pointer);	Видалення елементу зі списку, якщо відомий покажчик на нього
procedure Move(CurIndex, NewIndex : Integer);	Переміщує елемент з індексом CurIndex на нове місце з індексом NewIndex
procedure Exchange (Index1, Index2 : Integer);	Міняє місцями елементи в списку
procedure Clear ; dynamic;	Видалення всіх елементів списку
function IndexOf (Item:Pointer) : Integer;	Повертає індекс елемента у списку за його адресою, або –1 якщо такого немає
function Expand : TList;	Збільшує максимальну кількість елементів в списку (значення властивості Capacity)
procedure Pack;	Видаляє зі списку всі порожні (Nil) покажчики
function First : Pointer; function Last : Pointer;	Повертає покажчики на перший та останній елементи в списку

Таблиця А.2 – Основні методи класу TStrings

Назва методу	***Призначення***
Insert(Index:integer; const S:string);	Вставляє в список заданий рядок з заданим індексом. Існуючі елементи пересуваються донизу.
InsertObject(Index:integer; const S:string; Obj:TObject);	Вставляє з заданим індексом в список заданий рядок та об'єкт, що з ним пов'язаний
Add(const S:string);	Додає в кінець списку заданий рядок
AddObject(const S:string; Obj:TObject);	Додає в кінець списку рядок та об'єкт, що з ним пов'язаний
procedure Delete (Index : Integer);	Видаляє елемент зі списку.
procedure Move(CurIndex, NewIndex : Integer);	Переміщує елемент з індексом CurIndex на нове місце з індексом NewIndex
procedure Exchange (Index1, Index2 : Integer);	Міняє місцями елементи в списку
procedure Clear ; dynamic;	Видалення всіх елементів списку
function IndexOf (S:string) : Integer;	Повертає індекс рядка у списку, або –1 якщо такого немає
function IndexOfObject (Obj:TObject) : Integer;	Повертає індекс об'єкта у списку, або –1 якщо такого немає
function Equal (Strings : TStrings):boolean;	Перевіряю чи однакові за вмістом список, для якого викликано метод та список-параметр цього метода

Таблиця А.3 – Деякі властивості та методи класу **TStream**

Назва	**Призначення**
property Position : Longint;	Визначає поточну позицію в потоці
property Size : Longint;	Визначаю розмір потоку в байтах
procedure ReadBuffer (var Buffer; Count:Longint);	Зчитує з потоку Count байтів і записує їх у Buffer
procedure WriteBuffer(const Buffer; Count:Longint);	Записує в потік Count байтів з буферу Buffer

Додаток Б. Властивості класів графічної підсистеми

Таблиця Б.1. Властивості класів TPen, TBrush та TFont

Властивість	Призначення
Клас ***TPen***	
Handle: HPen;	Дескриптор пера.
Color: TColor;	Колір пера.
Mode: TPenMode;	Визначає як перо взаємодіє з поверхнею канви, тобто як колір пера буде взаємодіяти (об'єднуватися) з кольором канви.
Style: TPenStyle;	Встановлює стиль лінії, що рисується пером (сполошна, пунктирна, подвійна і т.д.)
Width : Integer;	Товщина пера в пікселах.
Клас ***TBrush***	
Handle: HBrush;	Дескриптор пензлю.
Color: TColor;	Колір пензлю.
Style: TBrushStyle;	Встановлює стиль пензлю (тип зафарбування: суцільна, штрихування і т.д.)
Bitmap: TBitmap	Містить бітову карту, яка визначена користувачем для заповнення поверхні.
Клас ***TFont***	
Handle: TFont;	Дескриптор шрифту.
Name: TFontName;	Ім'я (тип) шрифту, наприклад, TimesNewRoman.
Style: TFontStyle;	Стиль шрифту (жирний, з підкресленням, курсив і т.п.)
Color: TColor;	Колір шрифту.
Pitch: TFontPitch;	Встановлює спосіб встановлення ширини символів шрифту (однакова ширина символу або змінна)
Height: Integer;	Висота шрифту
PixelPerInch: Integer;	Визначає число точок на дюйм. Встановлюється автоматично, програміст не повинен його змінювати!!!
Size: Integer;	Розмір шрифту в пікселах. По замовчанню встановлюється 10.

Таблиця Б.2. Властивості класу TChart

Властивість	Призначення
AllowPanning: TPanningMode;	Визначає можливість користувача прокручувати частину графіка, що розглядається, під час виконання, натискаючи праву кнопку миші. Можливі значення: • *pmNone* - прокрутка заборонена; • *pmHorizontal* - прокрутка дозволена в горизонтальному напрямку; • *pmVertical* - прокрутка дозволена у вертикальному напрямку; • *pmBoth* - прокрутка дозволена в обох напрямках.
AllowZoom: Boolean;	Дозволяє користувачеві змінювати під час виконання масштаб зображення, вирізаючи фрагменти діаграми або графіка курсором миші.
BackWall: TChartWall; **BottomWall**: TChartWall; **LeftWall**: TChartWall;	Визначають характеристики відповідно задньої, нижньої і лівої граней області тривимірного відображення графіка.
BottomAxis: TChartAxis; **LeftAxis**: TChartAxis; **RightAxis**: TChartAxis;	Визначають характеристики відповідно нижньої, лівої і правої осей.
Chart3dPercent: Integer;	Масштаб тривимірного відображення діаграми.
Foot: TChartTitle;	Визначає підпис під діаграмою. За замовчуванням відсутня. Текст підпису визначається під властивістю Text.
Frame: TChartPen;	Визначає рамку навколо діаграми.
Legend: TChartLegend;	Легенда діаграми - список позначень.
MarginLeft: Integer; **MarginRight**: Integer; **MarginTop**: Integer; **MarginBottom**: Integer;	Значення лівого, правого, верхнього і нижнього полів.

SeriesList: TChartSeriesList;	Список серій даних, що відображаються в компоненті.
Title: TChartTitle;	Визначає заголовок діаграми.
View3d: Boolean;	Дозволяє або забороняє тривимірне відображення діаграми.
View3dOptions: TView3dOptions;	Характеристики тривимірного відображення.

Таблиця Б.3. Методи компонента TChartSeries

Метод	Призначення
function **AddXY** (Const AXValue, AYValue: Double; Const AXLabel: String; ACoIor: TColor): Longint;	Додає нову точку в серію. Параметри AXValue і AYValue містять відповідно значення по осях X і Y. Параметр AXLabel містить мітку для додається точки серії. Параметр ACoIor визначає колір. Функція повертає позицію нової точки у серії
function **AddY** (Const AYValue: Double; Const AXLabel: String; AColor: TColor): Longint;	Додає в серію нове значення по осі X. Застосовується для тих серій, в яких графік будується за X і мітками значень по X (наприклад, Pie, Bar). Призначення параметрів таке ж, як у методу AddXY
procedure **AssignValues** (Source: TChartSeries);	Копіює всі точки із серії Source в поточну серію
procedure **Clear**;	Видаляє всі значення із серії; якщо слідом за цим не занести нових точок, буде показуватися порожній графік
procedure **CheckDataSource**;	Оновлює точки в серії, незалежно від того, який компонент є джерелом даних - набір даних або інша серія. Оновлення здійснюється за поточними даними джерела. Метод рекомендується викликати в разі змін даних в джерелі.
procedure **ColorRange** (AValueList: TChartValueList; Const From Value, To Value: Double; AColor: TColor);	Змінює колір зазначеного діапазону точок серії. AValueList - або XValues, або YValues. From Value вказує початкове, а Te Value кінцеве значення у списку AValueList. AColor - новий колір
function **Count**: Longint;	Повертає число точок у серії. Наприклад, помістити всі значення по X і Y точок серії в ListBoxl
procedure **Delete** (ValueIndex: Longint);	Видаляє із серії точку з номером ValueIndex. Графік, до якого належить серія, автоматично перемальовується.
procedure **DoSeriesClick**	Ініціює настання події OnClick

Метод	Призначення
(ValueIndex: LongInt; Button: TMouseButton; Shift:TShiftState; X, Y: Integer); virtual;	
function **GetCursorValueIndex**: Longint;	Повертає індекс точки серії в TChart ValueList, найближче до якої розташований курсор миші. Якщо таку точку визначити не вдається, повертається - 1.
procedure **GetCursorValues** (Var x, y: Double);	Повертає значення по X і Y точки графіка (а не тільки серії), найближче до якої розташований курсор миші
function **GetHorizAxis**: TChartAxis;	Повертає покажчик на призначену серії горизонтальну вісь. Використовуючи даний покажчик, можна викликати методи осі, звертатися до її властивостей
function **GetVertAxis**: TChartAxis;	Повертає покажчик на вертикальну вісь
function **MaxXValue**: Double; virtual;	Повертає максимальне значення по X
function **MinXValue**: Double; virtual;	Повертає мінімальне значення по X
function **MaxYValue**: Double; virtual;	Повертає максимальне значення по Y
function **MinYValue**: Double; virtual;	Повертає мінімальне значення по Y
procedure **RefreshSeries**;	Оновлює значення серії з джерела даних, зазначеного у властивості DataSource
procedure **Repaint**;	Призводить до повного перемальовування всього графіка. Рекомендується викликати цей метод у разі зміни хоча б однієї з основоположних властивостей серії
function **ValuesListCount**: LongInt;	Повертає число списків значень точки, що використовуються в серії. Зазвичай це 2 (XValues і YValues), але деякі серії використовують 3 (BubbleSeries - XValues,

Метод	Призначення
	YValues, Radius; GanttSeries -Y, Start, End)
function **VisibleCount**: Longint;	Повертає число точок серії, видимих на графіку

Додаток В. Опис прецедентів задачі визначення координат центру тяжіння складного перерізу

Опис прецедентів програми визначення координат центра тяжіння складного перерізу

Прецедент	Формування складного перерізу	
Виконавець	Проектувальник	
Ціль	Введення просторових координат простих елементів перерізу	
Короткий опис	Проектувальник на основі даних про прості елементи. що входять до складу перерізу вносить їх координати та ознаку чи є елемент отвором (розривом) в перерізі.	
Типовий хід подій		
Дії виконавців		***Відгук системи***
Проектувальник вводить в поля координати простого елементу перерізу: для прямокутника в поля групи A, кола – поля групи B, трикутник – поля групи C. Задає ознаку отвору D.		Надання вікна для введення даних. Оброблення введення.
Проектувальник повідомляє систему про закінчення введення, натиснувши кнопку E.		Створення екземпляру класу для простого елемента. Додавання елемента до перерізу. Графічне відображення елемента в координатному просторі CF.
Проектувальник ініціює розрахунок координат центру тяжіння всього перерізу, натиснувши кнопку F.		Розрахунок КЦТ та площі перерізу. Виведення результатів на екран (група міток G)
Альтернативи		
1. Проектувальник вводить некоректні дані в поля		2. Якщо неможливо конвертувати рядок з поля в число, виведення повідомлення, надання можливості редагування даних.

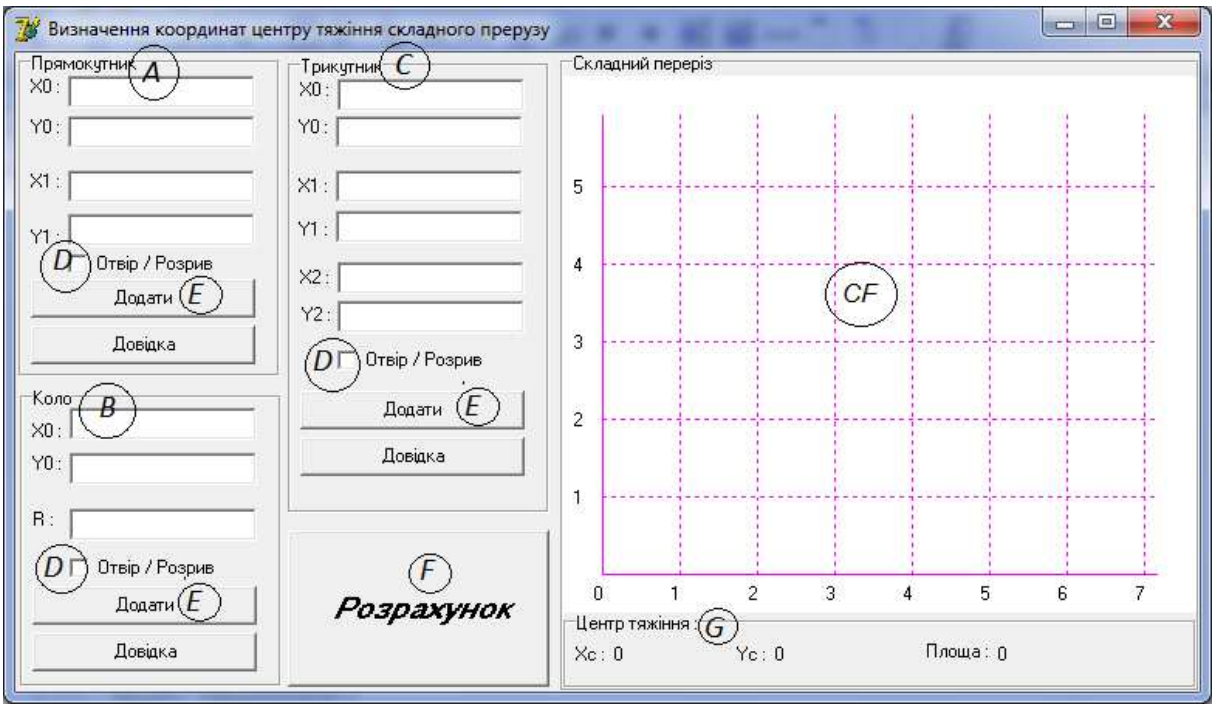


Рис. Д.1 Зовнішній вигляд головного вікна програми

Прецедент	Розрахунок координат центру тяжіння складного перерізу	
Виконавець	Програма	
Ціль	Розрахунок координат центру тяжіння (КЦТ) та площі складного перерізу	
Короткий опис	Розрахунок проводиться за формулою 1.	
Типовий хід подій		
Дії виконавців		***Відгук системи***
Проектувальник ініціює розрахунок координат центру тяжіння всього перерізу, натиснувши кнопку F.		Перевірка коректності формування перерізу (чи немає перетину серед простих елементів, чи всі розриви в перерізі описані).
		Розрахунок КЦТ та площі перерізу. Виведення розрахункових значень на екран (мітки групи G). Графічне позначення центру тяжіння всього перерізу в координатному просторі CF.
Альтернативи		
		2. Якщо переріз сформований некоректно, виведення повідомлення,

	надання можливості редагування даних.

Прецедент	Перегляд зображення перерізу
Виконавець	Програма
Ціль	Візуалізація складного перерізу
Короткий опис	Під час введення простих елементів складного перерізу їх графічне зображення виводиться в координатному просторі.
Типовий хід подій	
Дії виконавців	***Відгук системи***
Проектувальник повідомляє систему про закінчення введення просторових характеристик елемента перерізу, натиснувши кнопку C.	Графічне відображення елемента в координатному просторі CF.
Проектувальник ініціює розрахунок координат центру тяжіння всього перерізу, натиснувши кнопку E.	Графічне позначення центру тяжіння всього перерізу в координатному просторі CF.

Опис системних операцій

	Опис
Ім’я	додавання елементу перерізу
Обов’язки	Додавання простого елемента до списку елементів, що утворюють переріз
Тип	системна
Посилання	Функції системи: 1.2 Прецедент: Формування складного перерізу
Примітки	Ознака отвору зберігається в масиві рядків Strings, що спадкується від предка TStringList. Значення «0» - істина, це елемент перерізу, «-1» - хибно, це отвір.
Винятки	
Вивід	
Передумови	Був створений S – екземпляр класу складного перерізу

	Опис
	TCompSection. Був створений CF – екземпляр класу TCoordField.
Постумови	просторові характеристики елементу перерізу зчитані, перетворені в числовий тип та передані в конструктор елемента. створений F – екземпляр елемента складного перерізу, нащадок TBGP; елемент перерізу F доданий до списку елементів перерізу S; на екрані виведене графічне зображення елемента виведене в поле CF

	Опис
Ім’я	розрахунок КЦТ складного перерізу
Обов’язки	виведення розрахункових значень КЦТ та площі складного перерізу
Тип	системна
Посилання	Функції системи: 2.2 Прецедент: Розрахунок координат центру тяжіння складного перерізу
Примітки	
Винятки	Для коректності розрахунку необхідно перевірити, чи не припущена проектувальником помилка при введенні просторових характеристик елементів перерізу: чи немає перетину елементів, чи немає незафіксованих розривів в перерізі.
Вивід	
Передумови	Був створений S – екземпляр класу складного перерізу TCompSection. Був створений CF – екземпляр класу TCoordField.
Постумови	розрахунок КЦТ виведений на екран. площа перерізу виведена на екран. центр тяжіння позначений графічно в просторі CF.