Міністерство освіти і науки України
Державний вищий навчальний заклад
“Ужгородський національний університет”
Математичний факультет
Кафедра системного аналізу і теорії оптимізації

І.В. Семйон, С.В. Чупов, А.Ю. Брила, Н.І. Апшай

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Навчальний посібник

Ужгород – 2011

УДК 004.421 (075.8)
ББК З 973.2-018я73
0-75

Основи об’єктно-орієнтованого програмування. Навчальний посібник// І.В. Семйон, С.В. Чупов, А.Ю. Брила, Н.І. Апшай – Ужгород, 2011. – 141 с.

Рецензенти: д. ф.-м. н., професор, Гупал А.М.,
д. ф.-м. н., професор, Цегелик Г.Г.,
к. ф.-м. н., професор, Бунда В.В.

У навчальному посібнику розглядаються передумови виникнення та основні поняття і принципи об’єктно-орієнтованого програмування реалізовані у середовищі Delphi. Кожна тема супроводжується запитаннями для самоконтролю, завданнями для самостійної роботи та зразком виконання одного завдання.

Адресовано студентам вищих навчальних закладів спеціальності «Прикладна математика» при вивченні дисципліни «Програмування».

Передумови виникнення об’єктно-орієнтованого програмування

Історія розвитку мов програмування почалася з виникненням електронних обчислювальних засобів. Перші програми рідко займали більше 1Кб і являли собою просту послідовність команд, які виконувалися в тому порядку, в якому вони були записані. Єдине, що було доступним для програміста при побудові логіки програми, – це умовні переходи, які дозволяли перейти до тієї чи іншої команди в залежності від виконання чи невиконання деякої умови. З розвитком технічних засобів зростала і складність програм. Стало очевидним, що даний підхід є неприйнятним при розв’язанні складних задач. У результаті було запропоновано *структурний підхід*, суть якого полягає у тому, що складна задача розбивається на простіші підзадачі, до розв’язання яких зводиться початкова задача. Цей підхід реалізується за допомогою *технології процедурного програмування*, яка передбачає розбиття основної задачі на підзадачі та оформлення останніх у вигляді підпрограм (процедур та функцій). Мови програмування, які підтримують технологію процедурного програмування, називають *процедурно-орієнтованими*. Було створено велику кількість бібліотек, які містили порівняно невеликі підпрограми, з яких, немов з цеглин, можна було будувати програми. У подальшому акцент у програмуванні почав робитися на організацію даних. Виявилося, що велика кількість помилок виникає внаслідок неправильного використання даних. До того ж контроль за використанням даних повинен робитися не лише на стадії компіляції програми, а й під час її виконання. Було розроблено ряд структур даних, які частково вирішували дану проблему. Подальший розвиток структурного підходу призвів до появи *модульного програмування*, суть якого полягає у намаганні “сховати” дані і процедури їх обробки всередині модуля. Модуль, що містить дані та процедури їх обробки, виявився зручним засобом для

автономної розробки та при багаторазовому використанні програмного коду. Зараз при розв’язанні прикладної задачі вибудовувалася ціла ієрархія модулів, які могли взаємодіяти між собою. Цей підхід набув широкого поширення саме завдяки автономності модулів та можливості їх багаторазового використання. Практика показала, що в більшості випадків такі модулі відображають властивості реальних об’єктів. Об’єднання в одному модулі процедур та даних, які відображають характерні властивості об’єктів та їх поведінку, привело до введення типу даних клас. Починаючи з мови програмування Simula-67, у програмуванні започатковано новий підхід, який називають *об’єктно-орієнтованим програмуванням (ООП)*. Основою ООП є тип даних клас (*class*), суть якого полягає в тому, що він об’єднує у собі описи структур даних і методи їх обробки. Таким чином, ООП – це методика розробки програм, в основі якої лежить поняття класу як деякої структури, що описує об’єкт реального світу та його поведінку. Задача, яку можна розв’язати за допомогою ООП, описується в термінах класів.

Слід зазначити, що переваги ООП в повній мірі проявляються при розробці достатньо складних програм. Спроби застосування ООП для розв’язання нескладних задач, наприклад, для обрахунків по заданих формулах призводить до штучного нагромадження мовних конструкцій. Зрозуміло, що такі програми не потребують структуризації (поділу на ряд відносно незалежних частин) і їх доцільно створювати, використовуючи традиційні засоби.

РОЗДІЛ 1

КЛАСИ І ОБ'ЄКТИ

1.1. Поняття класу та об'єкта

Будь-яка діяльність людини так чи інакше пов'язана з дослідженням та маніпуляцією різного роду об'єктами. Широко вживаним методом дослідження є метод *моделювання*, який ґрунтується на розробці та дослідженні моделей об'єктів реальної дійсності. *Модель* – це матеріальний або уявний об'єкт, що відображає важливі для дослідження властивості об'єкта-оригінала і використовується для дослідження об'єкта-оригінала. В обчислювальних системах широко використовується *інформаційна модель* – сукупність даних, що відображають основні властивості об'єкта-оригінала, його структуру та зв'язок з оточуючим середовищем. Для збереження даних в обчислювальних системах використовують ту чи іншу *структуру даних* – спосіб організації даних, що дозволяє відобразити зв'язки та відношення між даними.

У програмуванні сукупність даних, що описують властивості деякого об'єкта реальної дійсності, може бути представлена, наприклад, за допомогою структури даних, яку називають *записом* (Record). *Запис* – це складна структура даних, яка може містити поля різного типу. Розглянемо найпростіший вигляд описання записів у системі програмування Delphi, коли кожна із властивостей реального об'єкта, що описується, представляється окремим полем.

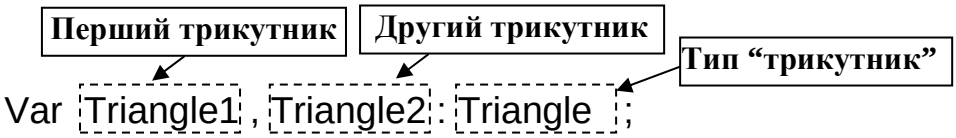
```
Type
    <ім'я типу>=Record
        <ім'я поля 1>:<тип поля>;
        <ім'я поля 2>:<тип поля>;
        ……..
    End;
```

Приклад. Опишемо структуру, яка характеризує трикутник.

Структура, яка описується	Відповідна програмна структура
Трикутник Сторона *a* Сторона *b* } Дані (поля) Сторона *c*	Type Triangle = Record A : Real; B : Real; C : Real; End;

Перевагою такої структури даних є те, що всі розглядувані властивості деякого окремого об’єкта містяться в одній структурі даних. У цьому випадку об’єкт реальної дійсності представляється за допомогою змінної типу Record.

Приклад. Два трикутники можуть бути представлені за допомогою двох змінних типу запис.

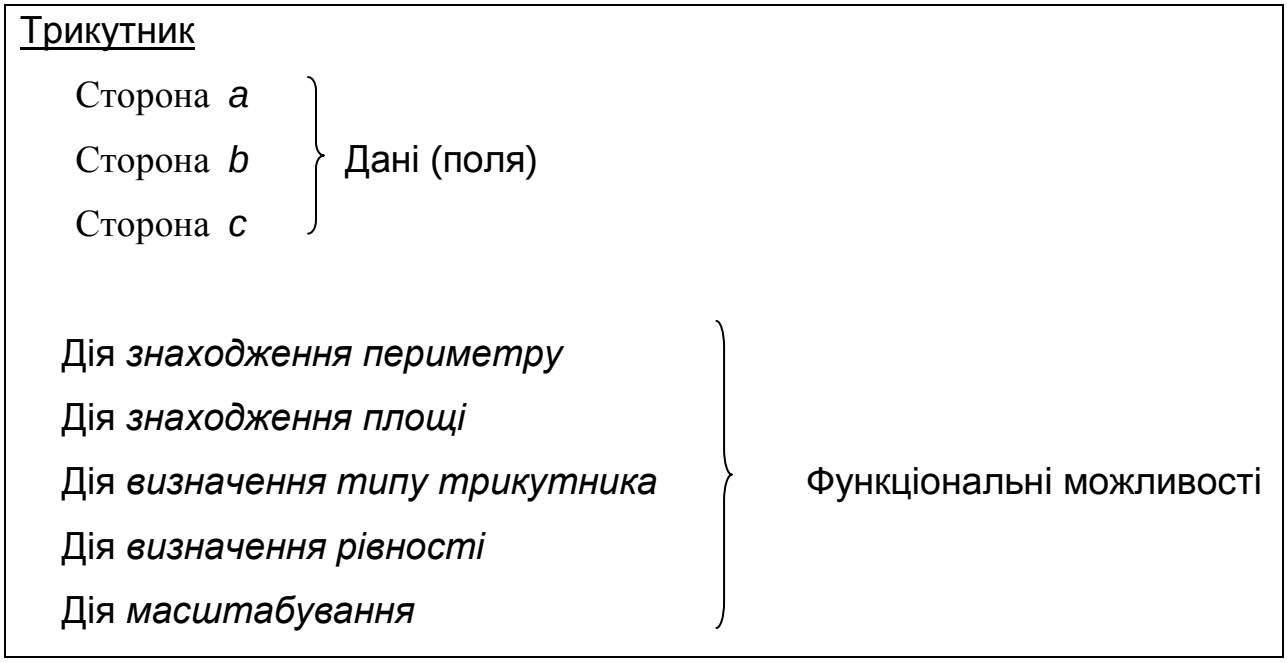


Але ж більшість реальних об’єктів характеризується не тільки певними кількісними характеристиками, а й набором функціональних можливостей. Іншими словами, об’єкти можуть характеризуватися властивостями та діями, що можуть виконувати самі об’єкти або які можна виконувати над ними.

Приклад. Для трикутника можна визначити наступні дії:

1) знаходження периметру;
2) знаходження площі;
3) визначення типу трикутника (рівнобедрений, рівносторонній, довільний);
4) визначення рівності з іншим трикутником, який задається за допомогою своїх сторін;
5) масштабування (збільшення або зменшення всіх сторін трикутника у певну кількість разів).

Отже, необхідно описати наступну логічну структуру



Зрозуміло, що в даному випадку структури даних запис вже недостатньо. Для повноцінного описання об’єктів реальної дійсності використовується тип даних, який має назву *клас* і який дозволяє описувати не тільки властивості об’єкту, а і його поведінку або дії. Слід зазначити, що при визначенні класу описуються властивості й методи, які характерні для всіх об’єктів відповідного класу (групи) реальних об’єктів. Як довільний тип даних, клас визначає певну множину елементів або екземплярів цього типу. Окремо взятий екземпляр деякого класу називають *об’єктом*. Прийнято вважати, що *клас* – це шаблон, на основі якого може бути створено конкретний програмний *об’єкт*, що моделює реальний об’єкт певної предметної області. З точки зору мови програмування, *клас* – це тип даних, а *об’єкт* – це змінна типу клас. *Клас* і *об’єкт* є фундаментальними поняттями технології об’єктно-орієнтованого програмування. Зауважимо, що об‘єкт представляє собою динамічну змінну, тому перед першим використанням його потрібно створити, а після завершення роботи з ним – знищити. Для створення та знищення об‘єктів в описі класу передбачені спеціальні методи, які називаються відповідно конструктором (*constructor*) та деструктором (*destructor*).

1.2. Описання класів

Загальний вигляд описання структури даних клас у системі програмування Delphi наступний:

```
Type
 <ім'я класу> = Class (<ім'я класу-предка>)
            <ім'я поля 1>:<тип поля>;
            <ім'я поля 2>:<тип поля>;                    } Поля класу
            ……..
 Constructor <ім'я конструктора> (<формальні параметри>);
            ……..
 Destructor <ім'я деструктора>(<формальні параметри>); override;
            ……..
   procedure <ім'я процедури 1>(<формальні параметри>);
   procedure <ім'я процедури 2>(<формальні параметри>);
            ……..
   function <ім'я функції 1>(<формальні параметри>):<тип функції>;
   function <ім'я функції 2>(<формальні параметри>):<тип функції>;
            ……..
 End;
```

Методи класу — } — *Члени класу*

При цьому дані, що описано в класі, називають *полями*, а процедури і функції – *методами*. Разом поля і методи називають *членами класу*. Зазначимо, що спочатку описують поля, а потім методи.

Зауваження. У мові програмування Delphi ім'я класу прийнято починати з великої літери "T", від слова "Type" (TButton, TEdit, TImage), а назви полів прийнято починати з малої літери "f", від слова "field" (fSide, fData, fColor). Методи, що повертають значення полів об'єкта, прийнято починати з приставки "Get" (GetSide, GetColor, GetData), що встановлюють значення полів – "Set" (SetSide, SetColor, SetData), які повертають значення булевого типу – "Is" (IsEqual, IsEmpty).

Приклад. Опишемо розглядувану логічну структуру “Трикутник” за допомогою класу в системі програмування Delphi.

Структура, яка описується	Відповідна програмна структура
Трикутник **Сторона *a*** **Сторона *b*** **Сторона *c*** **Дія створення трикутника** **Дія *знаходження периметру*** **Дія *знаходження площі*** **Дія *визн. типу трикутника*** **Дія *визн. рівності*** **Дія *масштабування***	**Type** **TTriangle = class** **fA: Real;** **fB: Real;** **fB: Real;** **Constructor Create(A,B,C:Real);** **function Perimeter : Real;** **function Square : Real;** **function GetTriangleType : String;** **function IsEqual(fA,fB, fC:Real):Boolean;** **Procedure Scale(k:Real);** **End;**

Зараз розглядувані раніше два трикутники можуть бути представлені за допомогою двох змінних типу клас TTriangle:

Змінні Triangle1 і Triangle2 називають об’єктами класу TTriangle.

Оскільки в описанні класу представлені тільки заголовки методів, то обов’язково необхідно вказати їх реалізацію. Для того, щоб відрізнити реалізацію процедур і функцій, які можуть бути присутні в програмі, від методів, що зазначені в описі класу, перед назвами методів додатково вказують ідентифікатор (ім’я) того класу, в якому вони описані. Загальний вигляд описання реалізації методів наступний:

```
Constructor <ім'я класу>.<ім'я конструктора>(<форм. праметри>);
    Begin
      …….
    End;
Destructor  <ім'я класу>.<ім'я деструктора>;
    Begin
      …….
    End;
Procedure <ім'я класу>.<ім'я методу-процеудри>(<форм. праметри>);
    Begin
      …….
    End;
Function <ім'я класу>.<ім'я методу-функції>(<форм. параметри>):<тип>;
    Begin
      …….
    End;
          ……..
```

Приклад. Наведемо реалізацію методів Perimeter та Scale, що є членами класу TTriangle

```
Function   TTriangle . Perimeter:Real;  {Знаходження периметру}
  Begin
    Result:= fA + fB + fC ;
  End;

Procedure   TTriangle . Scale (k:Real);  {Збільшення сторін у k разів}
  Begin
          fA := k * fA ;
          fB := k * fB ;
          fC := k * fC ;
  End;
```

Поля класу **TTriangle** — fA, fB, fC

Ім'я класу — TTriangle

Імена методів — Perimeter, Scale

1.3. Доступ до полів та методів

У методах, що представлено в класі, доступ до полів та інших методів цього ж класу може бути здійснений безпосередньо за їхніми іменами (як у попередньому прикладі) або за допомогою покажчика на екземпляр класу Self

```
Self . <ім'я поля>;          // Доступ до власного поля
Self . <ім'я методу>;        // Виклик власного методу
```

Приклад. З використанням покажчика на екземпляр класу Self метод **Perimeter** може бути реалізовано наступним чином:

```
Function TTriangle . Perimeter:Real;
 Begin
  Result:= Self . fA+ Self . fB+ Self . fC;
 End;
```

Покажчик Self на екземпляр класу може бути використано у методах для того, щоб відрізнити поля класу від формальних параметрів, імена яких можуть збігатися з іменами полів класу.

Приклад. Розглянемо реалізацію методу **IsEqual**

```
function TTriangle.IsEqual( fA,fB, fC :Real):Boolean;
 Begin
  Result:=  ((Self.fA = fA )AND( Self.fB = fB )AND( Self.fC = fC )) OR
            ((Self.fA = fA )AND( Self.fB = fC )AND( Self.fC = fB )) OR
            ((Self.fA = fB )AND( Self.fB = fA )AND( Self.fC = fC )) OR
            ((Self.fA = fB )AND( Self.fB = fC )AND( Self.fC = fA )) OR
            ((Self.fA = fC )AND( Self.fB = fB )AND( Self.fC = fA )) OR
            ((Self.fA = fC )AND( Self.fB = fA )AND( Self.fC = fB ));
 End;
```

Формальні параметри

Поля класу TTriangle

У програмах-клієнтах, в яких використовується об’єкт деякого класу, доступ до полів та методів об’єкта, як і при роботі з записами, здійснюється за допомогою оператора крапка “ . ” або оператора приєднання With:

(без оператора With)

```
<ім’я об’єкта> . <ім’я поля>          //доступ до поля об’єкта
<ім’я об’єкта> . <ім’я методу>        // виклик методу
```

(з використанням оператора With)

```
With   <ім’я об’єкта> do
  begin
    <ім’я поля>                       //доступ до поля об’єкта
    <ім’я методу>                     // виклик методу
  end;
```

Приклад.

Без оператора With	З використанням оператора With

Triangle1. A := Triangle1. A + 34;
b:= Triangle1. IsEqual(2,3,2);
Triangle1. Scale (5);
	With Triangle1 do
	begin
	A := A + 34;
	b:= IsEqual(2,3,2);
	Scale (5);
	end;

Очевидно, використання оператора приєднання With спрощує написання програми і покращує її читабельність.

1.4. Конструктори. Створення об'єктів

За своєю природою об'єкти є динамічними структурами. Пам'ять для них виділяється за допомогою спеціального метода, який називається конструктором і, як правило, має назву Create. Для того, щоб підкреслити особливе призначення даного метода при його описанні вживається ключове слово Constructor. Даний метод (конструктор) може мати параметри, які дозволяють здійснити *ініціалізацію об'єкта*, тобто надати полям об'єкта початкових значень. У випадку, якщо при описанні класу не описано конструктора, то для створення об'єкта цього класу використовується стандартний конструктор Create без параметрів. Однак цей конструктор тільки виділяє пам'ять для об'єкта і не виконує жодних додаткових дій: ініціалізації полів, виділення додаткової пам'яті та ін. Тому для повноцінного створення об'єкта програміст повинен описувати власні конструктори. Більше того, у одному класі може бути описано декілька конструкторів.

Приклад. Наведемо приклад описання конструктора для класу TTriangle

```
TTriangle = class
      ……..
    Constructor  Create(A, B, C : Real);      Опис конструктора
      ……..
End;
```

```
……..
Constructor TTriangle.Create(A,B,C:Real);
  Begin
    fA := A;
    fB := B;
    fC := C;
  End;
```

Ініціалізація полів об'єкта

Реалізація конструктора

Використання об'єкту можливе тільки після його створення. Виключенням є виклик конструктора класу при створенні об'єкту, тобто екземпляра цього класу. При створенні об'єкту конструктор викликається через ім'я класу. Загальний синтаксис створення об'єкта наступний:

```
Var < об'єкт > : < ім'я класу >;          // Описання об'єкта
    ……..
                                          // Створення об'єкта
<об'єкт>:=<ім'я класу>.<ім'я конструктора>[(<фактичні параметри>)];
```

Прилад. Наведемо два варіанти створення об'єкта Triangle1 класу TTriangle

З описанням власного конструктора з параметрами	З використанням конструктора за замовчуванням
Triangle1 := TTriangle.Create(3,4,2);	Triangle1 := TTriangle.Create; Triangle1. fA := 3; Triangle1. fB := 4; Triangle1. fC := 2;
Тут у конструкторі виділяється пам'ять для об'єкта Triangle1, а також ініціалізуються його поля fA, fB, fC, фактичними параметрами: 3, 4, 2.	*Тут окремо виділяється пам'ять для об'єкта Triangle1, а потім здійснюється ініціалізація полів.*

1.5. Деструктори. Знищення об'єктів

Якщо деякий об'єкт більше не використовується у програмі, то він, як динамічна змінна, може бути видалений з динамічної пам'яті. Ця операція виконується за допомогою спеціального методу, який називають *деструктором*. Якщо в класі не описано власного методу-деструктора, то

знищення об’єкту може бути здійснено за допомогою стандартного деструктора Destroy. Однак цей метод не дозволяє виконати деякі особливі дії перед знищенням об’єкту, наприклад, звільнити ресурси, які використовував об’єкт і т.д.

Опис деструктора відрізняється від опису інших методів у класі службовим словом Destructor.

Приклад. Наведемо приклад описання деструктора для класу TTriangle

```
TTriangle =class
    ……..
  Destructor  Destroy;   override;        Опис деструктора
    ……..
 End;
    ……..
Destructor  TTriangle.Destroy;
 Begin
…….
{Дії, які необхідно виконати                Реалізація деструктора
 перед  знищенням об’єкта}
…….
 End;
```

Виклик деструктора такий же, як і виклик звичайного методу

< ім’я об’єкта > **. <ім’я деструктора>;**

Приклад.

```
Triangle1 := TTriangle.Create(3,4,2);     // Створення об’єкта
……                                        // Використання об’єкта
Triangle1. Desrtroy;                      // Знищення об’єкта
```

При використанні деструктора є небезпека звертання до неіснуючого об’єкту (який не був створений за допомогою конструктора, або вже знищений), що призведе до системної помилки. Тому часто для знищення об’єкту використовують метод-деструктор Free , загальна схема виклику якого

< ім’я об’єкта >**. Free**;

Цей метод самостійно викличе метод деструктор користувача, якщо такий описано (саме тому після опису власного деструктора вказується службове слово override). Але якщо об'єкту не існує, то звертання до методу-деструктора користувача не буде, і тому помилка не генерується. Завдяки цій особливості частіше використовують метод-деструктор Free.

Приклад.

```
Triangle1 := TTriangle.Create(3,4,2);     // Створення об'єкта
        ……                                // Використання об'єкта
Triangle1. Free;                          // Знищення об'єкта
```

Зауважимо, що якщо об'єкт створений програмістом, то його обов'язково потрібно знищити, тому що в іншому випадку деструктор автоматично не викликається і в результаті засмічується оперативна пам'ять.

Наведемо приклад модуля, у якому повністю реалізовано клас TTriangle та програми-клієнта, що використовує цей клас.

Приклад.

Текст модуля

```
Unit TriangleUnit;
INTERFACE
 {- - - - - - - - - - - Опис класу TTriangle - - - - - - - - - - -}
  Type
  TTriangle = class
     fA: Real;
     fB: Real;
     fC: Real;
    Constructor Create(A,B,C:Real);
     function Perimeter : Real;
     function Square : Real;
     function GetTriangleType: String;
     function IsEqual(fA,fB, fC:Real):Boolean;
     Procedure Scale (k:Real);
   End;

IMPLEMENTATION

{- - - - - - - - - - - Реалізація класу TTriangle - - - - - - - - - - -}
```

```
  Constructor TTriangle.Create(A,B,C:Real);            {Конструктор }
   Begin
    fA := A;
    fB := B;
    fC := C;
   End;
  function TTriangle.Perimeter : Real;                 {Знаходження периметру}
   Begin
    Result := fA+fB+fC;
   End;
  function TTriangle.Square : Real;                    {Знаходження площі}
    var p:Real;
   Begin
    p := (fA+fB+fC) / 2;
    Result:= Sqrt(p*(p - fA)*(p – fB)*( p - fC))
   End;
function TTriangle.GetTriangleType:String;  {Визначення типу трикутника}
   Begin
    if (fA=fB) AND (fA=fC) then Result:='Рівносторонній'
    else
       if (fA=fB) OR (fA=fC) OR (fB=fC) then Result:='Рівнобедрений'
         else Result:='Довільний';
   End;
  function TTriangle.IsEqual (fA,fB, fC:Real):Boolean;   {Перевірка на рівність}
   Begin
    Result:=  ((Self.fA = fA)AND(Self.fB = fB)AND(Self.fC = fC)) OR
                  ((Self.fA = fA)AND(Self.fB = fC)AND(Self.fC = fB)) OR
         ((Self.fA = fB)AND(Self.fB = fA)AND(Self.fC = fC)) OR
         ((Self.fA = fB)AND(Self.fB = fC)AND(Self.fC = fA)) OR
                 ((Self.fA = fC)AND(Self.fB = fB)AND(Self.fC = fA)) OR
                 ((Self.fA = fC)AND(Self.fB = fA)AND(Self.fC = fB));
   End;
  Procedure TTriangle.Scale (k:Real);                  {Збільшення сторін у k разів}
   Begin
      fA := k * fA;
      fB := k * fB;
      fC := k * fC;
   End;

END.
```

Текст програми-клієнта

```
PROGRAM Project1;     {$APPTYPE CONSOLE}
  USES  SysUtils, TriangleUnit;
  VAR     Triangle1: TTriangle;                    //Опис об'єкта
BEGIN
   Triangle1 := TTriangle.Create(3,4,2);        // Створення об'єкта
 {-----------------------------------------}
     writeln('Периметр трикутника = ', Triangle1.Perimeter);
     writeln('Площа трикутника = ', Triangle1.Square);
     writeln('Тип трикутника : ', Triangle1.GetTriangleType);
     writeln('Рівність трикутнику A=2,B=3,C=4: ', Triangle1.IsEqual(2,3,4));
writeln('================================');
 Triangle1.fA := 12;          //
 Triangle1.fB := 12;          // Прямий доступ до полів об'єкта
 Triangle1.fC := 10;          //
   writeln('Тип трикутника : ',Triangle1.GetTriangleType);
   writeln('Периметр трикутника = ',Triangle1.Perimeter:6:2);
   writeln('Площа трикутника = ',Triangle1.Square:6:2);
     Triangle1.Scale(2);                    //Збільшення сторін у два рази
   writeln('== Збільшено в 2 рази ==');
   writeln('Периметр трикутника = ',Triangle1.Perimeter:6:2);
   writeln('Площа трикутника = ',Triangle1.Square:6:2);
    {-----------------------------------------}
      Triangle1.Free;                  // Знищення об'єкта
 readln;
END.
```

Результати роботи програми

```
Периметр трикутника =  9.00
Площа трикутника =  2.90
Тип трикутника : Довільний
Рівність трикутнику A=2,B=3,C=4: TRUE
================================
Тип трикутника : Рівнобедрений
Периметр трикутника =    34.00
Площа трикутника =   54.54
== Збільшено в 2 рази ==
Периметр трикутника =    68.00
Площа трикутника = 218.17
```

1.6. Композиція класів. Поля типу клас

Часто при описанні класів доводиться описувати групи полів, які є спорідненими і характеризують деяку властивість об’єкта, що описується. Наприклад, розглянемо клас, що характеризує водія служби таксі.

Структура, яка описується			Відповідна програмна структура Type
Водій *Прізвище та ініціали* *Категорії*			**TDriver =class** fPIP:String; fCategories :String;
Дата народження	{	День Місяць Рік	Birth_Day:integer; Birth_Month:integer; Birth_Year:integer;
Дата видачі посвідчення водія	{	День Місяць Рік	Exam_Day:integer; Exam _Month:integer; Exam _Year:integer;
Дата проходження медогляду	{	День Місяць Рік	Medicine_Day:integer; Medicine_Month:integer; Medicine_Year:integer;
Дата прийому на роботу	{	День Місяць Рік	Start_Day:integer; Start_Month:integer; Start_Year:integer;
Початок зміни	{	Години Хвилини	Begin_Hour:integer; Begin _Minutes:integer;
Кінець зміни	{	Години Хвилини	End_Hour:integer; End _Minutes:integer;
			Constructor Create(PIP:String; Categories :String; B_D, B_M, B_Y, E_D, E_M, E_Y, M_D,M_M,M_Y, S_D,S_M,S_Y, WB_H,WB_M,WE_H,WE_M:integer); Destructor Destroy; override; **End;**

У структурі описано 18 полів. Щоб їх розрізнити, використовуються складні імена. Звичайно, такий підхід є прийнятним, але уявіть собі, якими будуть імена полів, коли доведеться описувати структуру, що має 100 і більше полів.

У даному прикладі чітко можна відділити групи полів, які характеризують дату народження, дату видачі посвідчення, дату проходження медогляду, дату прийому на роботу та групи полів, що характеризуються час початку і кінця зміни. Тому доцільно було б описати окремі класи, які представляють дату та час. Наведемо можливе їх описання.

```
Type
  TMyDate = class
    fDay : integer;
    fMonth : integer;
    fYear : integer;
   Constructor Create(D,M,Y:integer);
   End;
```

Клас, що представляє дату

```
  TMyTime  = class
    fHour : integer;
    fMinutes: integer;
    Constructor Create(H,M:integer);
   End;
```

Клас, що представляє час

Використаємо ці класи при описі класу TDriver. При цьому кожна із розглянутих груп полів буде представлена одним полем типу клас.

Без використання членів типу клас	З використанням членів типу клас
Type	Type
TDriver =class	**TDriver =class**
PIP:String[40];	PIP:String[40];
Categories :String[5];	Categories :String[5];
Birth_Day:integer; Birth_Month:integer; Birth_Year:integer;	**Birth _Date: TMyDate;**
Exam_Day:integer; Exam _Month:integer; Exam _Year:integer;	**Exam _Date: TMyDate;**
Medicine_Day:integer; Medicine_Month:integer; Medicine_Year:integer;	**Medicine_Date: TMyDate;**
Start_Day:integer; Start_Month:integer; Start_Year:integer;	**Start _Date: TMyDate;**
Begin_Hour:integer; Begin _Minutes:integer;	**Begin_Time: TMyTime;**
End_Hour:integer; End _Minutes:integer;	**End_Time: TMyTime;**
Constructor Create(PIP:String; Categories :String;B_D, B_M, B_Y, E_D, E_M, E_Y, M_D,M_M,M_Y, S_D,S_M,S_Y, WB_H,WB_M, WE_H,WE_M :integer); Destructor Destroy; override; **End;**	Constructor Create(PIP:String; Categories :String; B_D, B_M, B_Y, E_D, E_M, E_Y, M_D,M_M,M_Y, S_D,S_M,S_Y, WB_H,WB_M, WE_H,WE_M :integer); Destructor Destroy; override; **End;**

Очевидно, використання членів типу клас значно покращує читабельність програми і спрощує її розробку. Але слід пам’ятати, що клас є динамічною структурою даних, а отже, для членів типу клас також необхідно

викликати конструктори для їх створення та деструктори для знищення. Як правило, створення членів типу клас здійснюється у конструкторі класу, що їх містить.

Приклад. Конструктор класу TDriver може мати наступний вигляд:

```
Constructor  TDriver.Create (PIP:String; Categories :String;
        B_D,B_M,B_Y, E_D,E_M,E_Y,  M_D,M_M,M_Y, S_D,S_M,S_Y,
        WB_H,WB_M, WE_H,WE_M :integer);
Begin
   Birth_Date :=TMyDate.Create(B_D,B_M,B_Y);
   Exam_Date:=TMyDate.Create(E_D,E_M,E_Y);
   Medicine_Date:=TMyDate.Create(M_D,M_M,M_Y);
   Start_Date:=TMyDate.Create(S_D,S_M,S_Y);
    Begin_Time:=TMyTime.Create(WB_H,WB_M);
    End_Time:=TMyTime.Create(WE_H,WE_M);
End;
```

Створення полів типу клас

Якщо у класі не використовуються поля типу клас, то можна використовувати деструктор за замовчуванням, але якщо такі поля присутні, то при знищенні об'єкта обов'язково необхідно викликати деструктори кожного з таких полів.

Приклад. Деструктор класу TDriver може мати наступний вигляд:

```
Destructor  TDriver.Destroy;
  Begin
   Birth_Date.Free;
   Exam_Date.Free;
   Medicine_Date.Free;
   Start_Date.Free;
    Begin_Time.Free;
    End_Time.Free;
  End;
```

Знищення полів типу клас

Наведемо приклад використання класу TDriver.

Текст модуля

(Опис та реалізація класу)

```
UNIT TDriverUnit;
INTERFACE
Type
{- - - - - - - - - - - Клас, що представляє дату - - - - - - - - - - -}
   TMyDate = class
     Day : integer;
     Month : integer;
     Year : integer;
    Constructor Create(D,M,Y:integer);
   End;
{- - - - - - - - - - - Клас, що представляє час - - - - - - - - - - -}
   TMyTime  = class
     Hour : integer;
     Minutes: integer;
    Constructor Create(H,M:integer);
   End;
{- - - - - - - - - - - Клас, що представляє водія - - - - - - - - - - -}
   TDriver =class
        fPIP:String;
        fCategories :String;
        Birth_Date: TMyDate;
        Exam_Date: TMyDate;
        Medicine_Date: TMyDate;
        Start_Date: TMyDate;
          Begin_Time: TMyTime;
          End_Time: TMyTime;
     Constructor Create(PIP:String;
      Categories :String; B_D, B_M, B_Y,
       E_D, E_M, E_Y,  M_D,M_M,M_Y,
       S_D,S_M,S_Y,    WB_H,WB_M,
        WE_H,WE_M :integer);
      Destructor Destroy; override;
   End;
IMPLEMENTATION
  Constructor TMyDate.Create(D,M,Y:integer);
   Begin
    Day:=D;
    Month:=M;
    Year:=Y;
   End;
```

```
    Constructor TMyTime.Create(H,M:integer);
     Begin
      Hour:=H;
      Minutes:=M;
     End;
{-------------------------------------------------------}
 Constructor  TDriver.Create(PIP:String;
       Categories :String;
       B_D, B_M, B_Y,  E_D, E_M, E_Y,
       M_D,M_M,M_Y, S_D,S_M,S_Y,
       WB_H,WB_M, WE_H,WE_M :integer);
 Begin
     fPIP:=PIP;
     fCategories:=Categories;
     Birth_Date:=TMyDate.Create(B_D,B_M,B_Y);
     Exam_Date:=TMyDate.Create(E_D,E_M,E_Y);
     Medicine_Date:=TMyDate.Create(M_D,M_M,M_Y);
     Start_Date:=TMyDate.Create(S_D,S_M,S_Y);
       Begin_Time:=TMyTime.Create(WB_H,WB_M);
       End_Time:=TMyTime.Create(WE_H,WE_M);
 End;
Destructor  TDriver.Destroy;
  Begin
   Birth_Date.Free;
   Exam_Date.Free;
   Medicine_Date.Free;
   Start_Date.Free;
     Begin_Time.Free;
     End_Time.Free;
  End;
end.
```

Текст програми-клієнта

```
program Project1;
{$APPTYPE CONSOLE}
USES  SysUtils,  Windows,
  TDriverUnit;
Var    Driver:TDriver;                    // Опис об'єкта Driver
Time:Integer;
BEGIN                                     // Створення об'єкта Driver
  Driver:=TDriver.Create('Ivanov I.I.',
        'A,B',
        10,4,1980,   11,10,200,
        23,11,2009,  12,3,2000,
        6,20,    13,30);
```

```
  with Driver do
    begin            {обчислення часу роботи водія у хвилинах}
     Time:=((End_Time.Hour*60 + End_Time.Minutes)-
          (Begin_Time.Hour*60+Begin_Time.Minutes));
    end;
    writeln(Driver.fPIP, ' працює  ',Time, ' хвилин.');
    Readln;
  Driver.Free;                              // Знищення об'єкта Driver
END.
```

Результат роботи програми

```
Ivanov I.I. працює 430 хвилин.
```

1.7. Області видимості

Поля об'єкту завжди мають містити коректні дані для нормального функціонування об'єкту. Тому не бажано явно надавати клієнту доступ до них. Крім того, при описанні класів деякі методи можуть використовуватися тільки для внутрішніх потреб класу і доступ до них потрібно обмежити. У специфікації Object Pascal такого роду обмеження доступу здійснюється при описанні класів за допомогою директив, що визначають області видимості, тобто області, в межах яких члени класу є доступними з поза меж даного класу.

При описанні членів класу можуть використовуватися наступні директиви: Private, Protected, Public, Published. Ці директиви розділяють члени класу при описанні на розділи.

```
Type
    <ім'я класу>=class
      Private
          <члени класу>
      Protected
          <члени класу>
      Public
          <члени класу>
      Published
          <члени класу>
    End;
```

Розглянемо детальніше області видимості, що визначаються наведеними директивами.

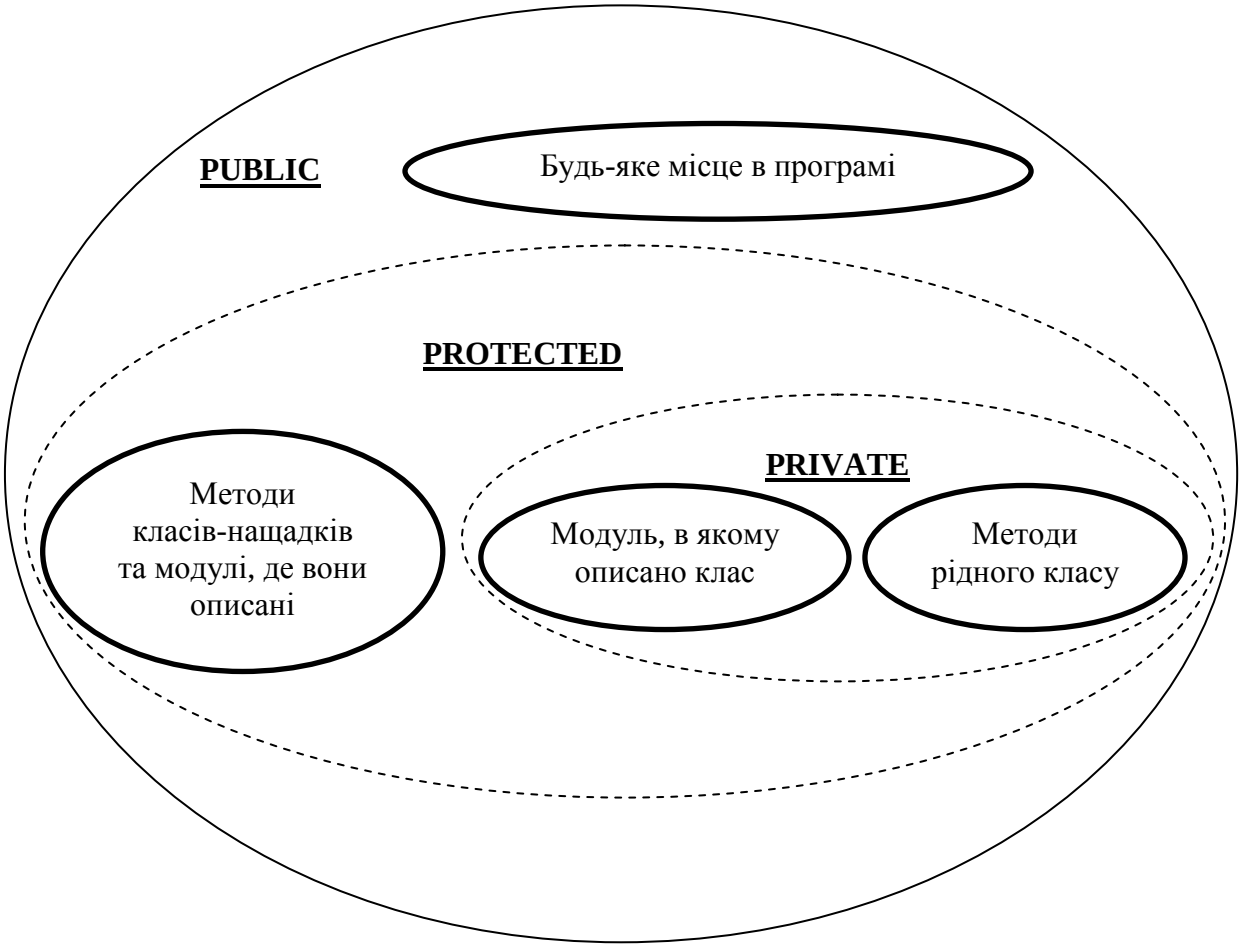
Private. Члени класу (поля, властивості, методи), які описані в розділі Private (після директиви Private), доступні тільки в методах цього ж класу та модулі, де описано даний клас. Дана директива дозволяє повністю “сховати” внутрішню реалізацію класу від користувача, заборонивши доступ до певної частини членів цього класу.

Protected. Описані в розділі Protected члени класу, як і члени класу, що описані в розділі Private, доступні в методах цього ж класу та в модулі, де описано даний клас. Але, крім цього, члени класу, що описані в розділі Protected, доступні в методах класів-нащадків та в модулях, у яких вони описані. Таким чином, члени класу, що описані в розділі Protected, недоступні у програмах-клієнтах, але доступні у методах, що описані у класах-нащадках.

Public. Члени класу, які описані в розділі Public, на відміну від розділу Private, доступні в будь-якому без виключення місці програми. В цьому розділі, як правило, описують “відкриті” члени класу, за допомогою яких здійснюється доступ до внутрішньої структури класу.

Published. У розділі Published описуються члени класу, які можуть бути використані засобами візуального проектування під час розробки програми. Властивості, які описані в розділі Published, відображаються в інспекторі об’єктів. Під час виконання програми члени класу, що описані в цьому розділі, є такими ж доступними, як і члени класу, які описані в розділі Public.

Схематично розглянуті області видимості можна представити наступним чином:



Зауваження. За замовчуванням (якщо не вказано директиви області видимості) вважається, що маємо область видимості Published.

Зауваження. При описанні класу необхідно дотримуватися правила, згідно з яким для членів цього класу визначається якомога менший рівень доступу.

Питання для самоконтролю

1. *Що таке клас?*
2. *Для чого використовують класи?*
3. *Наведіть загальну схему опису класу та його реалізації.*
4. *Що таке поле?*
5. *Що таке метод?*
6. *Як здійснюється доступ до членів класу?*
7. *Що таке об'єкт?*
8. *Що таке конструктор?*
9. *Що таке конструктор за замовчуванням?*
10. *Що таке деструктор?*
11. *Яка різниця між деструкторами* Destroy *та* Free*?*
12. *Як здійснюється доступ до полів та методів об'єкта в програмі-клієнті?*
13. *Як створюються та знищуються поля типу клас?*
14. *Для чого потрібні поля типу клас?*
15. *У яких межах доступні члени класу, описані в області* Private*?*
16. *У яких межах доступні члени класу, описані в області* Protected*?*
17. *У яких межах доступні члени класу, описані в області* Public*?*

Завдання для самостійної роботи

Описати класи, які містять вказані поля і методи та реалізувати програму-клієнт для тестування.

1. Клас “Текст”	
поля	▪ для зберігання послідовності символів; ▪ для зберігання тематики вказаного тексту;
методи	▪ визначення кількості символів у тексті; ▪ визначення кількості пробілів (між словами один пробіл); ▪ заміни кожного входження однієї букви на іншу; ▪ видалення слова з вказаним порядковим номером.
2. **Клас** “Вектор”	
поля	▪ для зберігання координат вектора; ▪ для зберігання розмірності вектора;
методи	▪ введення елементів вектора; ▪ виведення елементів вектора у рядку; ▪ визначення довжини вектора; ▪ нормування вектора.
3. **Клас** “Матриця”	
поля	▪ для зберігання елементів матриці; ▪ для зберігання розмірності матриці;
методи	▪ введення елементів матриці; ▪ виведення елементів матриці; ▪ знаходження найбільшого елемента; ▪ знаходження найменшого елемента.
4. **Клас** “Число”	
поля	▪ для зберігання натурального числа;
методи	▪ введення числа; ▪ виведення числа;

	▪ знаходження кількості цифр; ▪ знаходження суми цифр.
5. **Клас** “Число – масив цифр”	
поля	▪ для зберігання кількості цифр; ▪ для зберігання натурального числа, як масиву цифр;
методи	▪ введення числа; ▪ виведення числа; ▪ знаходження кількості входження деякої цифри; ▪ знаходження суми цифр; ▪ порівняння з іншим числом-масивом.
6. **Клас** “Арифметична прогресія ”	
поля	▪ для зберігання першого члена; ▪ для зберігання різниці;
методи	▪ введення та виведення першого члена; ▪ введення та виведення різниці; ▪ знаходження n-го члена прогресії; ▪ знаходження суми n перших членів прогресії.
7. **Клас** “Геометрична прогресія ”	
поля	▪ для зберігання першого члена; ▪ для зберігання знаменника;
методи	▪ введення та виведення першого члена; ▪ введення та виведення знаменника; ▪ знаходження n-го члена прогресії; ▪ знаходження суми n перших членів прогресії.
8. **Клас** “Текстовий файл ”	
поля	▪ для зберігання імені текстового файлу;
методи	▪ введення та виведення імені текстового файлу; ▪ створення текстового файлу; ▪ виведення текстового файлу на екран;

	▪ визначення кількості рядків; ▪ знаходження кількості входжень деякого слова у файлі.
9. **Клас** “Файл дійсних чисел ”	
поля	▪ для зберігання імені типізованого файлу, що містить дійсні числа;
методи	▪ введення та виведення імені файлу; ▪ створення типізованого файлу; ▪ виведення чисел, які містяться у файлі, на екран; ▪ додавання нового числа у файл; ▪ знаходження суми чисел.
10. **Клас** “Множина цифр”	
поля	▪ для зберігання множини цифр;
методи	▪ додавання нової цифри; ▪ виведення цифр, які входять у множину на екран; ▪ знаходження найбільшої цифри; ▪ знаходження суми цифр.
11. **Клас “**Стек**” (**реалізація стеку за допомогою одновимірного масиву цілих чисел)	
поля	▪ для зберігання вершини стеку (номера останнього доданого елемента); ▪ масив елементів;
методи	▪ виведення на екран; ▪ додавання нового елементу; ▪ видалення елементу; ▪ знаходження суми елементів.
12. **Клас “**Черга**” (**реалізація стеку за допомогою одновимірного масиву цілих чисел)	
поля	▪ для зберігання номерів першого та останнього елементів черги;

	▪ масив елементів;
методи	▪ виведення на екран; ▪ додавання нового елементу; ▪ видалення елементу; ▪ знаходження суми елементів.
13. **Клас “**Впорядкований масив**” (**елементами масиву є цілі числа, які завжди впорядковані за зростанням)	
поля	▪ для зберігання кількості елементів масиву; ▪ масив елементів;
методи	▪ виведення на екран; ▪ додавання нового елемента; ▪ видалення вказаного елемента; ▪ знаходження елемента, з використанням бінарного пошуку.
14. **Клас “**Неорієнтований граф**” (**зберігається за допомогою матриці суміжності)	
поля	▪ для зберігання кількості вершин; ▪ матриця суміжності;
методи	▪ виведення на екран кількості вершин; ▪ виведення на екран ребер графа; ▪ додавання нового ребра; ▪ видалення ребра; ▪ пошук шляхів між двома заданими вершинами.
15. **Клас “**Орієнтований граф із зваженими ребрами**” (**задається матриця вагів)	
поля	▪ для зберігання кількості вершин; ▪ матриця вагів;
методи	▪ виведення на екран кількості вершин; ▪ виведення на екран ребер графа;

	▪ додавання нового ребра; ▪ видалення ребра; ▪ пошук найкоротшого шляху між двома заданими вершинами.
16. **Клас** “Працівник” (використати поля типу клас)	
поля	▪ прізвище та ініціали; ▪ дата народження (поле типу клас); ▪ дата прийняття на роботу (поле типу клас); ▪ розмір заробітної плати;
методи	▪ визначення стажу роботи працівника; ▪ визначення віку працівника на даний момент; ▪ визначення загальної виплаченої суми коштів протягом всього періоду роботи.
17. **Клас** “Автомобіль” (використати поля типу клас)	
поля	▪ марка; ▪ відомості про виробника (поле типу клас) (назва фірми, рік заснування, номер телефону, обсяги виробництва); ▪ відомості про продавця (поле типу клас) (назва фірми, рік заснування, номер телефону, обсяги продажу); ▪ відомості про власника (поле типу клас) (прізвище та ініціали, ідентифікаційний код) ▪ колір; ▪ номер; ▪ дата випуску (поле типу клас);
методи	▪ визначення віку автомобіля; ▪ зміни власника.
18. **Об’єкт** “Адреса мандрівника” (використати поля типу клас)	
поля	▪ країна;

	▪ область; ▪ місто/село; ▪ вулиця; ▪ номер будинку; ▪ дата заповнення (поле типу клас); ▪ термін перебування (кількість днів); ▪ відомості про мандрівника (поле типу клас) (прізвище та ініціали, професія, дата народження (поле типу клас));
методи	▪ визначення того, чи не закінчився термін перебування; ▪ визначення кількості днів до закінчення терміну; ▪ визначення кількості місяців перебування (на основі вказаної кількості днів).
19. **Клас** “Камера схову”	
поля	▪ кількість комірок для зберігання багажу; ▪ одновимірний масив комірок, з елементами типу клас, який містять такі поля: ▪ для визначення того чи зайнята комірка; ▪ кількість одиниць багажу; ▪ одновимірний масив з елементами типу клас, який містить інформацію про багаж (дата отримання багажу, вага, термін зберігання, ідентифікаційний номер власника);
методи	▪ виведення на екран кількості вільних та зайнятих комірок; ▪ визначення кількості вільних комірок через вказану кількість днів (за умови, що новий багаж надходити не буде); ▪ заповнення та звільнення комірок; ▪ визначення загальної ваги вантажу; ▪ визначення ваги вантажу, який належить власнику,

	ідентифікаційний номер якого надається.
20. **Клас** “Готель” (використати поля типу клас)	
поля	▪ назва готелю; ▪ тип готелю; ▪ кількість послуг та інформація про послуги як масив (для кожної послуги зберігається: назва послуги, вартість, тривалість); ▪ кількість та масив інформації про кімнати (порядковий номер, кількість місць, вартість кімнати на одну добу, дата заїзду та дата виїзду жильців);
методи	▪ виведення на екран послуг, які можуть бути надані за вказану суму коштів та вказаний термін часу; ▪ виведення на екран вільних номерів з вказаною кількістю місць; ▪ визначення суми коштів, яку повинні сплатити жильці вказаної кімнати при від’їзді; ▪ визначення вказаної кількості кімнат за умови, що нових жильців приймати не будуть.
21. **Клас** “Фірма” (використати поля типу клас)	
поля	▪ назва фірми; ▪ дата заснування (рік, місяць); ▪ послуги (назва послуги, вартість, термін виконання); ▪ адреси філіалів (країна, місто, вулиця, номер будинку);
методи	▪ визначення кількості років існування фірми; ▪ виведення всіх філіалів фірми у вказаному місті; ▪ виведення на екран послуг, що можуть бути надані за вказану суму коштів та вказаний термін часу.

Зразок виконання самостійної роботи

***УМОВА*. Описати клас, який містить вказані поля і методи та реалізувати тестову програму-клієнт.**

Клас "Текст"	
поля	▪ для зберігання послідовності символів; ▪ для зберігання тематики вказаного тексту;
методи	▪ визначення кількості символів у тексті; ▪ визначення кількості пробілів (між словами один пробіл); ▪ заміни кожного входження однієї букви на іншу; ▪ видалення слова з вказаним порядковим номером.

ОПИС КЛАСУ

Логічна структура

Текст
- Послід. символів
- Тематика
- Дія *визн. кількості символів*
- Дія *визн. кількості пробілів*
- Дія *заміна букв*
- Дія *видалення слова*
- Дія *створення*

Структура даних

```
Type
  TText = Class
     fSymbols : String;
     fTheme: String;
    function GetSymbolCount:Byte;
    function GetBlankCount:Byte;
    Procedure SymbolReplace(SFind,
                                SReplace:Char);
    Procedure DeleteWord (Num:Byte)
  Constructor Create(Symbols,
                          Theme: String);
   End;
```

ПРОГРАМНА РЕАЛІЗАЦІЯ

Текст модуля

```
UNIT TTextUnit;
INTERFACE
Type
  TText = Class
     fSymbols : String;
     fTheme: String;
    function GetSymbolCount:Byte;
    function GetBlankCount:Byte;
    Procedure SymbolReplace(SFind,SReplace:Char);
```

```
   Procedure DeleteWord (Num:Byte);
  Constructor Create(Symbols,Theme: String);
   End;

IMPLEMENTATION

  Constructor TText.Create(Symbols,Theme: String);
   Begin
    fSymbols:= Symbols;
    fTheme:=Theme;
   End;
  function TText.GetSymbolCount:Byte;       {Підрахунок кількості символів}
   Begin
     Result := Length(fSymbols);
   End;
  function TText.GetBlankCount:Byte;        {Підрахунок кількості пробілів}
   var temp:String;
       k:integer;
   Begin
     temp:=fSymbols;
     k:=0;
     while pos(' ',temp)<>0 do
      begin
       k:=k+1;
       Delete(temp,pos(' ',temp),1);
      end;
     Result:=k;
   End;
  Procedure TText.SymbolReplace(SFind,SReplace:Char);
   var i,n:integer;                         {Заміна символу SFind  на SReplace }
   Begin
    n:= GetSymbolCount;
    for i:=1 to n do
     if fSymbols[i]=SFind then fSymbols[i]:=SReplace;
   End;
  Procedure TText.DeleteWord (Num:Byte);
   var i,j,l,n:integer;                     {Видалення слова з номером Num }
   Begin
    if (fSymbols<>'')and(Num<=(GetBlankCount+1)) then
    begin
     if Num=1 then i:=1 else
      begin
       i:=pos(' ',fSymbols); l:=2;
       while (l<Num) do
        begin
```

```
        i:=i+1;
        if fSymbols[i]=' ' then l:=l+1;
       end;
     end;
    j:=i+1;  n:=GetSymbolCount;
    while (j<n)and(fSymbols[j]<>' ') do
     j:=j+1;
    if j<>n then j:=j-1;
    Delete(fSymbols,i,j-i+1);
   end;
  End;
end.
```

Програма-клієнт

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils, TTextUnit;
 Var t1:TText;

BEGIN
  t1:=TText.Create('one two three four five','numbers');
  writeln('Symbols              : ',t1.fSymbols);
  writeln('SymbolCount      : ',t1.GetSymbolCount);
  writeln('Blanks               : ',t1.GetBlankCount);
  writeln('---- Replace  o -> a -----');
    t1.SymbolReplace('o','a');
  writeln('After replace        : ',t1.fSymbols);
  writeln('----  Delete  third word ---');
   t1.DeleteWord(3);
  writeln('After delete         : ',t1.fSymbols);
  t1.Free;
  readln;
END.
```

Тестовий приклад

```
Symbols              : one two three four five
SymbolCount        : 23
Blanks               : 4
---- Replace  o -> a -----
After replace        : ane twa three faur five
----  Delete  third word ---
After delete         : ane twa faur five
```

РОЗДІЛ 2

ІНКАПСУЛЯЦІЯ – БАЗОВИЙ ПРИНЦИП ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

2.1. Поняття властивості

Нагадаємо, що клас – це складна структура даних, яка може містити поля різного типу та методи для їх обробки. Згідно з принципом *інкапсуляції* в об'єктно-орієнтованому програмуванні, поля – це внутрішні змінні об'єкта, які використовуються для збереження даних. Безпосередній доступ до полів об'єкта несе у собі небезпеку недопустимого використання полів (так у розглянутому раніше класі "Трикутник" користувач може надати стороні трикутника від'ємного значення). Щоб уникнути недопустимого використання даних, згідно принципу інкапсуляції, доступ до полів об'єкта повинен здійснюватися не безпосередньо, а за допомогою спеціальних методів (процедур і функцій). У цих методах програміст має можливість передбачити аналіз значень і не допустити некоректного використання полів об'єкта, а також зберігати та повертати значення полів у різних форматах.

Приклад. Описати клас, який представляє квадрат. Передбачити метод знаходження площі квадрата. Доступ до поля – сторони квадрата повинен здійснюватися за допомогою спеціальних методів.

Структура, яка описується	Відповідна програмна структура
Квадрат Сторона квадрата Метод *встановлення значення* Side Метод *зчитування значення* Side Метод *знаходження площі квадрата*	**TRect = class** Side:real; Constructor Create(P_Side:Real); Procedure Set_Side(Value:real); Function Get_Side:Real; Function Square:real; **end;**

Текст модуля

```
Unit R_Unit;

INTERFACE
Type
        { - - -  Описання класу TRect  - - - }
    TRect = class
      Side:real;
    Constructor Create(P_Side:Real);
      Procedure Set_Side(Value:real);
      Function Get_Side:Real;
      Function Square:real;
    end;
IMPLEMENTATION
        { - - - Реалізація методів класу TRect - - - }
 Constructor TRect.Create(P_Side:Real);
 Begin
  Set_Side(P_Side);         //Встановлення початкового значення поля Side
 End;
 Procedure TRect.Set_Side(Value:real);
 Begin
  if Value<0 then Side:=0           //Якщо нове значення сторони квадрата
   else Side:=Value;                //Value<0, то новим значенням буде 0
 End;
 Function TRect.Get_Side:Real;
 Begin
  Result:=Side;                 //Метод-функція повертає значення поля Side
 End;
 Function TRect.Square:real;
 Begin
  Result:=sqr(Side);    //Метод-функція повертає значення площі квадрата
 End;
END.
```

Текст програми-клієнта

```
Program Project1;
        {$APPTYPE CONSOLE}
Uses  SysUtils, R_Unit;
Var
  Kv:TRect;                     // Описання об'єкта  Kv класу TRect
  r:real;
BEGIN
  Kv:=TRect.Create(3);          //Створення об'єкта Kv класу TRect
                                //Початкове значення сторони рівне 3
```

```
   Writeln('----- Example 1 ----');
   writeln(' Side     = ',Kv.Get_Side:2:2);
   writeln(' Square   = ',Kv.Square:2:2);

   Writeln('----- Example 2 ----');
   Kv.Set_Side(5);       //Надання полю Side (стороні квадрата) значення 5
   writeln(' Side     = ',Kv.Get_Side:2:2);
   writeln(' Square   = ',Kv.Square:2:2);
   Writeln('----- Example 3 ----');
   Kv.Set_Side(-7);       // Спроба надати полю Side (стороні квадрата)
                  // некоректного значення -7 (в результаті сторона рівна 0)
   writeln(' Side     = ',Kv.Get_Side:2:2);
   writeln(' Square   = ',Kv.Square:2:2);
   Readln;
  Kv.Free;                  // Знищення об'єкта Kv

END.
```

Результати роботи програми

```
----- Example 1 ----
 Side     = 3.00
 Square   = 9.00
----- Example 2 ----
 Side     = 5.00
 Square   = 25.00
----- Example 3 ----
 Side     = 0.00
 Square   = 0.00
```

Наведений у прикладі прийом застосування окремих методів для встановлення та зчитування значення полів пропонувався в ранніх специфікаціях як рекомендація, якої бажано дотримуватись. Але цей прийом не забороняє звертатися безпосередньо до поля з програми-клієнта, якщо поле має область видимості public, тому, наприклад, можливим є наступний оператор присвоєння Kv.Side:=-7 (довжині сторони квадрата присвоєно некоректне від'ємне значення). Для уникнення такої ситуації поле описують в області private, а доступ до поля реалізується за допомогою *властивості*. *Властивість* – це член класу, який, як і поле, асоціюється з певними даними об'єкта, але, на відміну від поля, має відповідні методи для зчитування та встановлення значень. Метод встановлення значення поля називається

методом запису властивості (write), а метод зчитування значення поля – *методом зчитування властивості* (read).

Розглянемо синтаксис опису властивості:

```
Property <ім'я властивості>:<тип властивості> read <ім'я методу зчитування> write <ім'я методу запису>;
```

Приклад.

```
Property Side:Real read Get_Side write Set_Side;
```

Отже, властивості використовуються для доступу до полів класу, які мають бути захищені (включені у розділ Private). Загалом, описання властивості для доступу до захищеного поля здійснюється наступним чином:

```
Type
 <Ім'я класу>=class
   Private                    // Опис закритих членів класу
 <ім'я поля>:<тип поля>;      //Поле, доступ до якого потрібно захистити
 Function <ім'я методу зчитування>:<тип поля>;        //метод зчитування
 Procedure  <ім'я методу запису>(value: <тип поля>);      // метод запису
   Public           //Опис полів та методів, які є загальнодоступними
 Property <ім'я властивості>:<тип поля> read <ім'я методу зчитування>
          write  <ім'я методу запису>;
 End;
```

Зауваження. Оскільки властивість у даному випадку використовується для доступу до захищеного поля, то тип властивості повинен збігатися із типом цього захищеного поля.

Приклад. Опишемо клас **TRect** з використанням властивості. Доступ до поля fSide буде здійснюватися за допомогою властивості Side.

```
Type
   TRect = class
    Private
       fSide:real;
        Procedure Set_Side(Value:real);
        Function Get_Side:Real;
    Public
      Property Side:Real read Get_Side write Set_Side;
   End;
```

Таким чином, властивості дають можливість здійснювати коректне використання полів об’єктів, які, як правило, описуються в розділі Private і доступні тільки для методів цього ж об’єкта та в межах модуля, де описано клас. Характерною рисою властивостей є те, що компілятор дозволяє працювати із властивостями як із звичайними полями об’єкта. В той же час, при наданні властивості деякого значення насправді викликається метод запису, який вказано після службового слова write, а при звертанні до значення властивості викликається метод зчитування, який вказується після службового слова read.

Приклад. Нехай Kv є об’єктом класу TRect.

Звертання до властивості	Дія	Еквівалент з викликом відповідного методу
Kv.Side:=5;	*встановлення значення*	Kv.Set_Side(5);
r:= Kv.Side;	*зчитування значення*	r:=Kv.Get_Side;

Наведемо повний текст програми із використанням описаної властивості.

Текст модуля

```
Unit R_Unit;
INTERFACE
Uses   SysUtils;
Type
    TRect = class
      Private
        fSide:real;              // Доступ до поля fSide буде обмеженим
        Procedure Set_Side(Value:real);
        Function Get_Side:Real;
      Public                     // Опис загальнодоступних методів
       Constructor Create(P_Side:Real);
        Property Side:Real read Get_Side write Set_Side;
        Function Square:real;
    End;
IMPLEMENTATION
 Constructor TRect.Create(P_Side:Real);
 Begin
```

```
 Side:=P_Side;
End;
Procedure TRect.Set_Side(Value:real);
Begin
 if Value<0 then fSide:=0                    //Доступ до закритого поля fSide
  else fSide:=Value;
End;
Function TRect.Get_Side:Real;
Begin
 Result:=fSide;                              //Доступ до закритого поля fSide
End;
Function TRect.Square:real;
Begin
 Result:=sqr(Side);
End;
END.
```

Текст програми-клієнта

```
Program Project1
        {$APPTYPE CONSOLE}
Uses
  SysUtils, R_Unit;
Var       Kv:TRect;
          r:real;
BEGIN

  Kv:=TRect.Create(3);
    Writeln('----- Example 1 ----');
    writeln(' Side     = ',Kv.Side:2:2);       //Виклик еквівалентний Kv.Get_Side
    writeln(' Square   = ',Kv.Square:2:2);

    Writeln('----- Example 2 ----');
    Kv.Side:=5;                                //Еквівалентно Kv.Set_Side(5)
    writeln(' Side     = ',Kv.Side:2:2);       //Виклик еквівалентний Kv.Get_Side
    writeln(' Square   = ',Kv.Square:2:2);

    Writeln('----- Example 3 ----');
    Kv.Side:=-7;                               //Еквівалентно Kv.Set_Side(-7)
    writeln(' Side     = ',Kv.Side:2:2);       //Виклик еквівалентний Kv.Get_Side
    writeln(' Square   = ',Kv.Square:2:2);
    Readln;
  Kv.Free;

END.
```

Результат роботи програми той самий, що і в попередньому прикладі.

2.2. Безпосередній доступ до поля при описі властивості

Якщо обмежень на допустимість значень поля не накладено, то при описанні властивостей не обов'язково створювати методи зчитування і запису значення цього поля. У цьому випадку може надаватися безпосередній доступ до поля через зазначення імені поля замість імені відповідного методу.

Приклад.

Безпосередній доступ до поля при зчитуванні (при цьому оператор r:=Kv.**Side** еквівалентний оператору r:= Kv.**fSide,** якщо не брати до уваги область видимості поля **fSide**):

```
TRect = class
  Private
    fSide:real;
  Public
   Constructor Create(P_Side:Real);
    Procedure Set_Side(Value:real);
  Property Side:Real read fSide write Set_Side;
    Function Square:real;
End;
```

Безпосередній доступ до поля при встановленні значення поля (при цьому оператор Kv.**Side**:=5 еквівалентний оператору Kv.**fSide**:=5 **,** якщо не брати до уваги область видимості поля **fSide**):

```
TRect = class
  Private
    fSide:real;
  Public
   Constructor Create(P_Side:Real);
    Function Get_Side:Real;
   Property Side:Real read Get_Side write fSide;
    Function Square:real;
End;
```

Безпосередній доступ до поля як для зчитування, так і для встановлення значення поля:

```
TRect = class
 Private
   fSide:real;
 Public
  Constructor Create(P_Side:Real);
   Property Side:Real read fSide   write fSide;
    Function Square:real;
End;
```

2.3. Обмеження доступу до властивості. Властивості тільки для читання і тільки для запису

Властивості дають можливість не тільки коректно використовувати поля об'єкта, а й обмежити доступ до поля, тобто описувати властивості тільки для читання чи тільки для запису:

- загальний вигляд опису властивості тільки для читання:

Property <ім'я властивості>:<тип властивості> **read** <ім'я методу зчитування>;

Приклад.

```
Type
  TRect = class
   Private
    fSide:real;
      Function Get_Side:Real;
   Public
     ……….
    Property Side:Real read Get_Side;
     ……….
  End;
```

зараз використання оператора Kv.**Side**:=5 заборонено;

- загальний вигляд опису властивості тільки для запису:

Property <ім'я властивості> :<тип властивості> **write** <ім'я методу запису>;

Приклад.

```
Type
  TRect = class
   Private
    fSide:real;
```

```
    Procedure Set_Side(Value:real);
  Public
    ……….
   Property Side:Real write Set_Side;
    ……….
End;
```

зараз використання оператора r:= Kv.**Side** заборонено.

2.4. Властивості-методи

При описанні класів часто доводиться використовувати методи, які дозволяють знаходити (обчислювати) деякі характеристики реального об'єкта, що описується. Назвемо їх методами-характеристиками.

Приклад. Метод Square дозволяє обчислювати площу квадрата, яка є однією із його характеристик.

```
TRect = class
  Private
    fSide:real;
    Procedure Set_Side(Value:real);
    Function Get_Side:Real;
  Public
   Constructor Create(P_Side:Real);
    Property Side:Real read Get_Side write Set_Side;
    Function Square:real;
End;
```

Але ж доступ до даних (характеристик) об'єкта прийнято організовувати за допомогою властивостей. Тому доцільно було б з метою узагальнення виклик методу-характеристики також організувати за допомогою властивості. Назвемо такі властивості *властивостями-методами*. Загальна схема опису властивості-методу наступна:

```
Type
 <Ім'я класу>=class
   Private                          // Опис закритих членів класу
       Function <ім'я методу-характеристики>:<тип методу>;
       ……..
   Public            //Опис полів та методів, які є загальнодоступними
```

```
Property  <ім'я властивості>:<тип методу> read <ім'я методу-характеристики>;
        ……..
  End;
```

Зауваження. Оскільки властивість буде представляти метод, то тип властивості повинен збігатися з типом методу-характеристики.

Приклад. Опишемо розглянутий клас TRect з використанням властивості-методу. Розглянутий раніше метод Square перейменуємо на Get_Square і опишемо його у розділі Private. Сама ж реалізація методу залишається незмінною.

```
TRect = class
  Private
    fSide:real;          // Доступ до методу Get_Square буде обмеженим
    Procedure Set_Side(Value:real);
    Function Get_Side:Real;
    Function Get_Square:real;
  Public                          // Опис загальнодоступних методів
   Constructor Create(P_Side:Real);
    Property Side:Real read Get_Side write Set_Side;
    Property Square:Real read Get_ Square;
End;
```

Зауваження. Слід зазначити, що такі зміни у класі не потребують змін програми, оскільки звертання до властивості є еквівалентним виклику методу-характеристики.

Запитання для самоконтролю

1. *Що таке інкапсуляція?*
2. *Що визначають області видимості?*
3. *Якою є область видимості за замовчуванням?*
4. *Чому поля класів прийнято включати в область Private?*
5. *Для чого потрібні методи для встановлення та зчитування значень захищених полів?*
6. *Що таке властивість?*

7. *Чи може надаватися безпосередній доступ до захищеного поля при використанні властивостей?*
8. *Чим відрізняються властивості тільки для читання і властивості тільки для запису?*
9. *Що таке методи-характеристики?*
10. *Що таке властивості-методи?*
11. *Чим відрізняються властивості-методи від звичайних властивостей?*

Завдання для самостійної роботи

У кожному із завдань передбачити конструктор класу та метод ToString, який дозволяє одержати рядкове представлення даних об’єкта. Доступ до даних реалізувати за допомогою властивостей. Опис та реалізацію класу розмістити в окремому модулі, а для тестування створити програму-клієнт.

1. Створити клас TMoney для роботи з грошовими сумами. Сума повинна зберігатися у вигляді доларового еквіваленту. Реалізувати методи додавання/вилучення грошової маси, вказуючи необхідну суму у гривнях, та визначення курсу долара, при якому сума у гривнях збільшиться на 100. Курс долара зберігати в окремому полі.
2. Створити клас TAngle для роботи з кутами. Кут зберігається як ціле число у градусній мірі. Реалізувати збільшення та зменшення кута на величину, яка задається у радіанах.
3. Створити клас TRational для роботи з раціональними дробами. Число представляється парою цілих чисел. Реалізувати збільшення, зменшення, множення і ділення числа на число, яке задається як дійсне число.
4. Створити клас TDate для роботи із датами у форматі “день.місяць.рік”. Дата представляється структурою із трьома полями. Реалізувати методи збільшення/зменшення дати на певну кількість днів, місяців чи років.

5. Створити клас TFraction для роботи із десятковими дробами. Число представляється за допомогою двох полів, у яких зберігається ціла та дробова частини числа. Реалізувати методи збільшення та зменшення числа.
6. Створити клас TBankomat, який моделює роботу банкомата. Клас повинен містити поля для зберігання кількості купюр кожного із номіналів від 5 до 200 гривень. Реалізувати методи знаходження максимальної та мінімальної сум, які може видати банкомат, та метод зняття деякої суми.
7. Створити клас TTime для роботи із часом у форматі “години:хвилини”. Час представляється структурою із двома полями. Реалізувати методи збільшення/зменшення часу на певну кількість годин чи хвилин.
8. Створити клас TMan для зберігання інформації про людину з полями: ім’я, вік, стать. Реалізувати методи визначення знаку гороскопа та встановлення того, чи є людина дитиною, юнаком або дорослою людиною.
9. Створити клас TGoods, який характеризує деякий товар на складі. Клас повинен містити поля: назва товару, одиниці виміру, кількість, ціна однієї одиниці. Реалізувати методи визначення сумарної вартості товару та методи збільшення/зменшення кількості товару.
10. Створити клас TBus, який характеризує автобус і містить наступні поля: марка автобуса, вага, кількість місць, кількість пасажирів. Передбачити методи збільшення/зменшення кількості пасажирів та визначення наближеної ваги автобуса з пасажирами без багажу.
11. Створити клас TQuadrangle, який характеризує чотирикутник і містить наступні поля: довжини сторін, кути. Передбачити методи для визначення типу чотирикутника, знаходження периметру та площі.
12. Створити клас TFirm, який характеризує приватне підприємство і містить наступні поля: назва, вид діяльності, кількість працівників,

кількість продукції, що може бути виготовлена за один день. Передбачити методи для визначення кількості продукції, яка може бути виготовлена за вказаний період часу та часу, який необхідно для виготовлення вказаної кількості продукції.

13. Створити клас TElectronicDevice, який характеризує електронний пристрій і містить наступні поля: назва, сфера застосування, енергоспоживання, виробник. Передбачити методи для визначення кількості спожитої енергії при вказаному періоді часу та часу, протягом якого може працювати пристрій при наявному заряді батарей.
14. Створити клас TCharacter, який характеризує символ і містить наступні поля: код символу, назва таблиці кодування, колір та розмір. Передбачити методи для визначення символу, код якого відрізняється від даного на вказане ціле число, переведення символу у верхній регістр та навпаки.
15. Створити клас TDisk, який характеризує носій інформації і містить наступні поля: ємність, гарантій термін експлуатації, надійність, об’єм записаної інформації, дата випуску. Передбачити методи для визначення кількості вільних байт/кілобайт/мегабайт/гігабайт, визначення залишку гарантійного терміну.
16. Створити клас TElectronicDocument, який характеризує електронний документ і містить наступні поля: дата створення, дата останнього редагування, автора. Передбачити методи для визначення “віку” документа, часу від створення до останнього редагування, часу від останнього редагування до вказаної дати.
17. Створити клас TBook, який характеризує книгу і містить наступні поля: відомості про автора (прізвище ім’я по-батькові, дата народження, кількість виданих книжок), кількість сторінок, рік видання кількість екземплярів, ціна. Передбачити методи для визначення “віку” книги, віку автора на час видання, вартості всього видання.

18. Створити клас TPlant, який характеризує рослину і містить наступні поля: назва рослини, вид, вегетаційний період, час посіву. Передбачити методи для визначення віку рослини та часу дозрівання плодів на основі введеної дати посіву.
19. Створити клас TAnimal, який характеризує тварину і містить наступні поля: назва, вид, приріст ваги за добу, кількість корму на одну добу. Передбачити методи для визначення приросту ваги однієї тварини за вказаний період часу та кількість спожитого корму.
20. Створити клас TChemicalMatter, який характеризує хімічну речовину і містить наступні поля: назва, позначення, концентрація, об'єм. Передбачити методи для визначення кількості води, яка необхідна для досягнення необхідної концентрації та концентрації, яку одержимо після додання вказаної кількості води.
21. Створити клас TSchoolSubject, який характеризує шкільний предмет і містить наступні поля: назва, загальна кількість годин, вік дітей, для яких передбачено предмет. Передбачити методи для визначення кількості годин, які необхідно провести на тиждень, кількості годин на місяць.

Зразок виконання самостійної роботи

УМОВА. Створити клас TMoney для роботи з грошовими сумами. Сума повинна зберігатися у вигляді доларового еквіваленту. Реалізувати методи додавання/вилучення грошової маси, вказуючи необхідну суму у гривнях, та визначення курсу долара, при якому сума у гривнях збільшиться на 100. Курс долара зберігати у окремому полі. Передбачити конструктор класу та метод ToString. Доступ до даних реалізувати за допомогою властивостей.

ОПИС КЛАСУ

Логічна структура	Структура даних
Гроші Сума (в доларах) Курс долара Дія *встановлення суми*	TMoney=class Private fSuma:real; fDKurs:real; Procedure Set_Suma(value:real);

Дія *зчитування суми* Дія *встановлення курсу* Дія *зчитування курсу* Дія *знаходження курсу* (*сума + 100*) ***Властивість*** Сума (грн.) ***Властивість*** Курс ***Властивість*** Шуканий курс (*сума + 100*) Дія *збільшення суми* Дія *зменшення суми* Дія *одержання рядкового представлення даних*	Function Get_Suma:Real; Procedure Set_DKurs(value:real); Function Get_DKurs:Real; Function Get_SearchDKurs:Real; Public Constructor Create(P_Suma,P_DKurs:Real); Property Suma:Real read Get_Suma write Set_Suma; Property DKurs:Real read Get_DKurs write Set_DKurs; Property SearchDKurs:Real read Get_SearchDKurs; Procedure Add_Sum(value:real); Function Sub_Sum(value:real):boolean; Function ToString:String; End;

ПРОГРАМНА РЕАЛІЗАЦІЯ

Текст модуля

```
UNIT UnitTMoney;

INTERFACE
 Uses SysUtils;
  {- - - - - - -  Опис класу TMoney  - - - - - - -  }
 Type
  TMoney=class
    Private
      fSuma:real;
      fDKurs:real;
      Procedure Set_Suma(value:real);
      Function Get_Suma:Real;
       Procedure Set_DKurs(value:real);
       Function Get_DKurs:Real;
       Function Get_SearchDKurs:Real;
    Public
       Constructor Create(P_Suma,P_DKurs:Real);
      Property Suma:Real read Get_Suma write Set_Suma;   //власт.-поле
      Property DKurs:Real read Get_DKurs write Set_DKurs; //власт.-поле
      Property SearchDKurs:Real read Get_SearchDKurs;     //власт.-метод
       Procedure Add_Sum(value:real);
       Function Sub_Sum(value:real):boolean;
      Function ToString:String;
```

```
 End;

IMPLEMENTATION
  {- - - - - - -  Реалізація методів класу TMoney  - - - - - - -  }
                    {  Метод встановлення суми value з переведенням у долари  }
 Procedure TMoney.Set_Suma(value:real);
  Begin
   if (value>=0)and(DKurs>=0) then
    fSuma:=value/DKurs
   else
    fSuma:=0;
  End;
                          {  Метод зчитування суми з переведенням у гривні }
 Function TMoney.Get_Suma:Real;
  Begin
   if fDKurs>=0 then
    Result:=fSuma*DKurs
   else
    Result:=0;
  End;
                                              {  Метод встановлення курсу долара  }
 Procedure TMoney.Set_DKurs(value:real);
  Begin
   if value>=0 then
    fDKurs:=value
   else
    fDKurs:=0;
  End;
                                                 {  Метод зчитування курсу долара  }
 Function TMoney.Get_DKurs:Real;
  Begin
   Result:=fDKurs;
  End;
                                     {  Метод визначення курсу долара, при якому }
                                       {  сума у гривнях збільшиться на 100              }
 Function TMoney.Get_SearchDKurs:Real;
  Begin
   Result:=(Suma+100)/fSuma;
  End;
                                                                {  Конструктор класу }
 Constructor TMoney.Create(P_Suma,P_DKurs:Real);
  Begin
   DKurs:=P_DKurs;
   Suma:=P_Suma;
```

```
 End;
                                                { Збільшення суми на value гривень }
Procedure TMoney.Add_Sum(value:real);
 Begin
  if (value>0)and(DKurs<>0) then
   Suma:=Suma+value;
 End;
                                                { Зменшення суми на value гривень }
Function TMoney.Sub_Sum(value:real):boolean;
 Begin
  if (value>0)and(DKurs<>0) then
   begin
    if Suma>=value/DKurs then
     begin
      Suma:=Suma-value;
      Result:=true;       //Якщо сума не менша за  value то сума зменшується
                          // і функція повертає true
     end
    else
     Result:=false;      //Якщо сума менша за  value то функція повертає false
   end
  else
   Result:=False;
 End;
                                      { Одержання рядкового представлення даних }
Function TMoney.ToString:String;
 Begin
  Result:=' Dollars ='+floattostrf(fSuma,ffFixed,6,2)+'; '+
          ' DKurs   ='+floattostrf(DKurs,ffFixed,6,2)+'; '+
          ' Grivna ='+floattostrf(Suma,ffFixed,6,2);
 End;
END.
```

Програма-клієнт

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
USES
  SysUtils,
  UnitTMoney;
VAR
  Money:TMoney;

BEGIN
```

```
  Money:=TMoney.Create(100,8.5);
   writeln('--- Create(100,8.5) ---');
    writeln(Money.ToString);
   writeln('--- Add_Sum(150) ---');
    Money.Add_Sum(150);
    writeln(Money.ToString);
   writeln('--- Sub_Sum(30) ---');
    Money.Sub_Sum(30);
    writeln(Money.ToString);
   writeln('--- Suma:=-300 ---');
    Money.Suma:=-300;
    writeln(Money.ToString);
   writeln('--- Suma:=300 ---');
    Money.Suma:=300;
    writeln(Money.ToString);
   writeln('--- DKurs:=8 ---');
    Money.DKurs:=8;
    writeln(Money.ToString);
   writeln('--- SearchDKurs ---');
    writeln(Money.SearchDKurs:6:2);
   writeln('--- DKurs:=SearchDKurs ---');
    Money.DKurs:=Money.SearchDKurs;
    writeln(Money.ToString);
  readln;
   Money.Free;
end.
```

Тестовий приклад

```
--- Create(100,8.5) ---
 Dollars =11,76;  DKurs   =8,50;  Grivna =100,00
--- Add_Sum(150) ---
 Dollars =29,41;  DKurs   =8,50;  Grivna =250,00
--- Sub_Sum(30) ---
 Dollars =25,88;  DKurs   =8,50;  Grivna =220,00
--- Suma:=-300 ---
 Dollars =0,00;  DKurs   =8,50;  Grivna =0,00
--- Suma:=300 ---
 Dollars =35,29;  DKurs   =8,50;  Grivna =300,00
--- DKurs:=8 ---
 Dollars =35,29;  DKurs   =8,00;  Grivna =282,35
--- SearchDKurs ---
 10.83
--- DKurs:=SearchDKurs ---
 Dollars =35,29;  DKurs   =10,83;  Grivna =382,35
```

РОЗДІЛ 3

УСПАДКУВАННЯ – ДРУГИЙ ОСНОВНИЙ ПРИНЦИП ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ. ВИДИ МЕТОДІВ

3.1. Поняття успадкування

Однією з проблем у програмуванні є повторне використання вже існуючого програмного коду та його модифікація. Як було зазначено раніше, проблема повторного використання окремих процедур і функцій легко вирішується з використанням модулів. Проблема повторного використання методів обробки даних може бути вирішена за допомогою класів. Один раз описаний клас, як незалежна програмна структура (тип даних), може бути з легкістю використаний у будь-якій програмі-клієнті. Труднощі виникають у випадку, коли з'являється необхідність модифікації вже існуючого класу, оскільки зміна існуючого класу може призвести до критичних помилок та втрати цілісності структури. В ООП ця проблема вирішується завдяки успадкуванню. *Успадкування* – це властивість класів, яка дає можливість описувати новий клас (клас-нащадок) на основі вже існуючого класу (класу-предка).

Клас-нащадок характеризується наступними ознаками:

1) він автоматично успадковує від класу-предка всі поля і методи;

2) можуть перевизначатися деякі методи класу-предка, тобто описуватися нові методи з іменами, які використані у класі предка, але знищувати будь-який член класу-предка не дозволяється;

3) може доповнюватися новими полями, властивостями та методами.

Отже, при створенні нового класу можна діяти за наступним алгоритмом:

1) знайти клас, що має багато спільного з тим, який ми хочемо описати, і визначити його як предка;

2) перевизначити методи класу-предка, якщо є така необхідність;

3) доповнити створений клас новими полями, властивостями та методами.

Наведемо загальний вигляд опису класу-нащадка

```
Type
    <ім'я класу-нащадка>=class(<ім'я класу-предка>)
      …….
    End;
```

Приклад. Описати клас, який представляє геометричну фігуру куб. Передбачити методи знаходження площі однієї бічної грані куба, та знаходження об'єму куба. Розмістити опис класу в окремому модулі.

Очевидно, в даному випаду доцільно використати вже створений клас TRect, який представляє геометричну фігуру квадрат.

Нова структура	***Клас-предок***	***Нові методи***
Куб	**TRect = class** Private fSide:real; ProcedureSet_Side(Value:real); Function Get_Side:Real; Public Constructor Create(P_Side:Real);	**Constructor** **Create(P_Side:Real);**
Сторона куба →	**Property Side:Real** read Get_Side write Set_Side;	
Площа бічної грані →	**Function Square:real;**	
Об'єм куба →		**Function Volume:real;**
	End;	

Сторона куба може бути представлена за допомогою властивості Side класу-предка TRect, а його метод Square може бути використаний для знаходження площі однієї бічної грані. Описання класу TCube, що представляє куб, наведено нижче.

```
Type   //Клас, що представляє квадрат
   TRect = class
     Private
       fSide:real;
       Procedure Set_Side(Value:real);
       Function Get_Side:Real;
     Public
      Constructor Create(P_Side:Real);
       Property Side:Real read Get_Side write Set_Side;
       Function Square:real;
   End;
```

```
//Клас, що представляє куб

TCube = class(TRect)
 Public
Constructor Create(P_Side:Real);
   Function Volume:real;
End;
```

Насправді ж клас TCube містить такі поля і методи

(*виділені курсивом успадковано від класу-предка* TRect, *нові методи підкреслено*)

```
   TCube = class(TRect)
     Private
       fSide:real;
       Procedure Set_Side(Value:real);
       Function Get_Side:Real;
     Public
      Constructor Create(P_Side:Real);
        Function Volume:real;
    Property Side:Real
             read Get_Side write Set_Side;
       Function Square:real;
    End;
```

Нехай у програмі-клієнті зроблено опис об’єкта Cube

```
Var Cube:TCube;
```

Схематично структуру об’єкта Cube зображено на рис 3.1 (штриховими лініями виділено недоступні для об’єкта Cube та перевизначені члени класу)

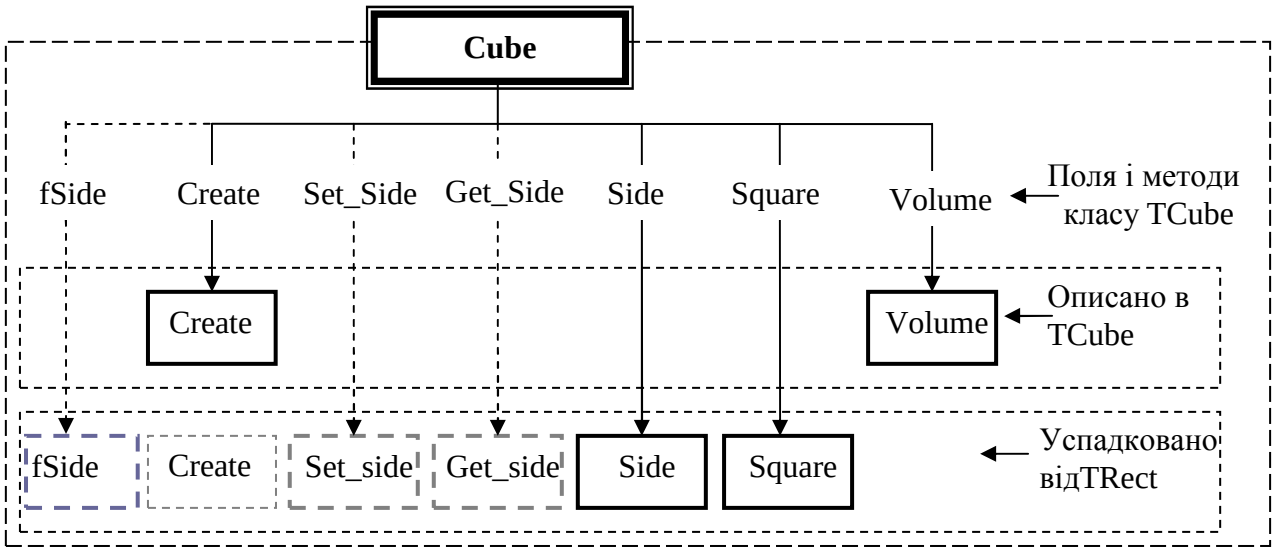


Рис. 3.1

Зауваження. Як зображено на схемі 3.1, конструктор Create класу TRect, є недоступним в класі TCube, бо він є перевизначеним в цьому класі. Поле fSide та методи Set_Side, Get_Side є недоступними, оскільки в класі TRect вони описані в розділі Private. Метод Square не перевизначається оскільки грань куба є квадратом, площа якого визначається в TRect.

Зауваження. У мові програмування Delphi всі класи є нащадками класу TObject. При описанні класів факт успадкування класу TObject можна опускати, тому наступні два описання є ідентичними.

<ім'я класу>=class ……. End;	<ім'я класу>=class(**TObject**) ……. End;

3.2. Вплив області видимості при успадкуванні

Згідно правил описання областей видимості, клас-нащадок, якщо він описаний не в тому модулі, в якому описано клас-предок, має можливість здійснювати доступ лише до тих членів класу-предка, які описані в розділі Public, Protected та Published.

Клас-предок	**Рівень доступу**	**Клас-нащадок**
<ім'я предка>=class **Private** <члени класу>	успадковано, але недоступні	<ім'я нащадка>=class(<ім'я предка>) **Private** ……
Protected <члени класу>	доступні включаються →	**Protected** ……
Public <члени класу>	доступні включаються →	**Public** ……
Published <члени класу> **End;**	доступні включаються →	**Published** …… **End;**

Зауваження. Доступність означає, що вони можуть бути викликані у методах класу-нащадка. При цьому вони включаються у той розділ класу-нащадка, в якому вони описані у класі-предка.

Згідно правил успадкування, член класу-предка не може оголошуватись в області з більш обмеженими правами доступу класу-нащадка. Так, наприклад, якщо член класу-предка описано у Public, то він не може бути включений у область Private класу-нащадка (якщо він не перевизначається). У той же час, член класу-предка може оголошуватись в області з менш обмеженими правами доступу класу-нащадка. Так, наприклад, член класу-предка з областю видимості Protected може бути оголошений в області Public нащадка, без нового описання цієї властивості.

<ім'я предка>=class Protected {*повний опис власт.*} Property <ім'я власт.>:<тип> Read <метод зчитування> Write <метод запису>; End;	<ім'я нащадка>=class(<ім'я предка>) Public {*тільки ім'я власт.*} Property <ім'я власт.>; End;

Приклад. Опишемо клас-предок TC1, клас-нащадок TC2.

Текст модуля класу-предка

```
unit TC1_Unit;
interface
 Type
 TC1=class
  Private
   fp:real;
  Protected
   Property P:real write fp;     //Опис властивості P, як захищеного члена
 end;
implementation
end.
```

Текст модуля класу-нащадка

```
unit TC2_Unit;
interface
 uses TC1_unit;
 Type
 TC2=class(TC1)
  Public
   Property P;                  //Розширення області видимості властивості P
 end;
implementation
end.
```

Текст програми-клієнта

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  TC1_Unit,  TC2_Unit;
var
    oc1:TC1;                //Об'єкт класу TC1
    oc2:TC2;                //Об'єкт класу TC2
begin
 {oc1.P:=2;  – помилка. У  TC1 властивість P описано у Protected}
 oc2.P:=2;       //Доступ можливий (у TC2 власт. P описано у Public)
end.
```

Аналогічні переоголошення можна робити від області Protected до Published, та від Public до Published.

3.3. Перевизначення методів. Конструктор класу-нащадка

Часто при визначенні класу-нащадка доводиться описувати однойменні дії, які також виконуються у класі предка, але мають інший зміст. Наприклад, розглянемо методи знаходження площі квадрата (клас-предок TRect) і площі поверхні куба (клас-нащадок TCube). Як у класі предка, так і у класі нащадка їх доцільно назвати Square. Явище, коли у класі нащадка визначається метод, ім'я якого використане у класі предка, називається *перевизначенням*. При цьому метод, описаний у нащадка, "приховує" однойменний метод класу-предка. Це означає, що об'єкт класу-нащадка має доступ лише до того методу, що описано у класі нащадка, а однойменний метод класу-предка є недоступним.

Слід зазначити, що у випадку успадкування, як правило, перевизначається також конструктор класу.

Приклад. Розглянемо описаний раніше клас-нащадок TCube, в якому метод Square буде визначати площу не однієї з граней, а площу всієї поверхні куба. Зрозуміло, для цього його потрібно перевизначити.

Клас-предок	***Нова структура***	***Нові методи***
TRect = class Private fSide:real; ProcedureSet_Side(Value:real); Function Get_Side:Real; Public Constructor Create(P_Side:Real);	**Куб** **Конструктор** *(перевизначається)*	 **Constructor Create(P_Side:Real);**
Property Side:Real read Get_Side write Set_Side;	**Сторона куба** *(успадковується)*	
Function Square:real; // *Square – площа квадрата*	**Площа поверхні** *(перевизначається)*	**Function Square:real;** *//Square – площа поверхні*
End;	**Об'єм куба**	**Function Volume:real;**

Описання нового класу TCube наведено нижче.

```
Type
                    //Клас, що представляє квадрат
   TRect = class
     Private
       fSide:real;
       Procedure Set_Side(Value:real);
       Function Get_Side:Real;
     Public
      Constructor Create(P_Side:Real);      {Конструктор квадрата}
      Property Side:Real read Get_Side write Set_Side;
       Function Square:real;                {Площа квадрата}
   End;
                    //Клас, що представляє куб
TCube = class(TRect)
  Public
     Constructor Create(P_Side:Real);      {Конструктор куба}
      Function Square:real;                {Площа поверхні куба}
     Function Volume:real;
End;
```

Нехай у програмі-клієнті зроблено опис об'єкта Cube

```
Var Cube:TCube;
```

Зараз виклик Cube.Square означає знаходження площі поверхні куба. Схематично структуру нового об'єкта Cube зображено на рис. 3.2.

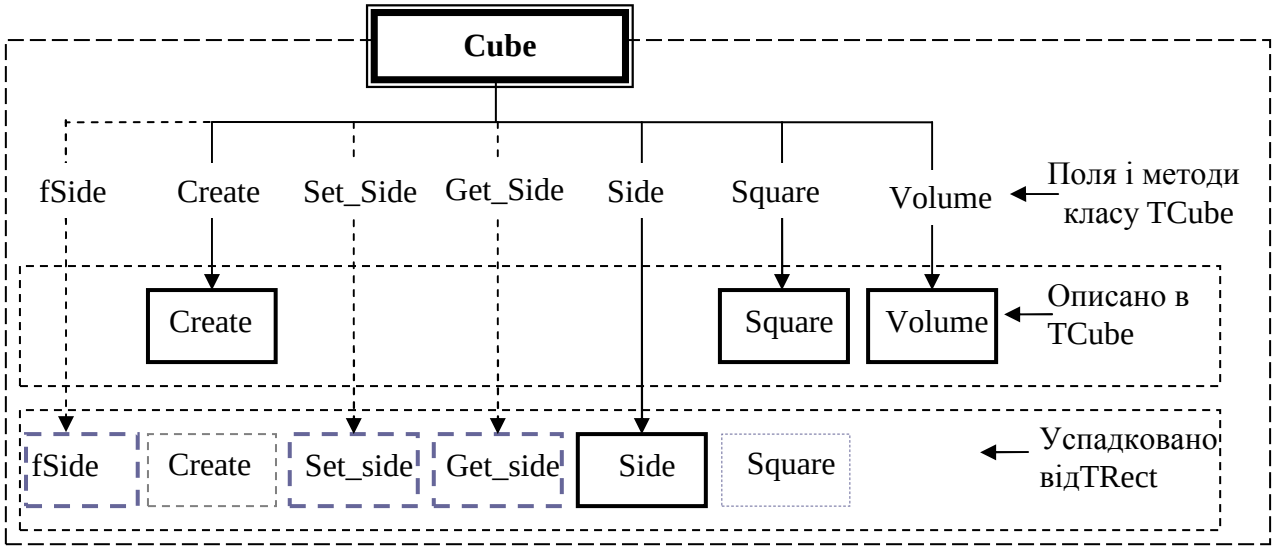


Рис. 3.2

Зазначимо, що навіть у випадку перевизначення методи класу-нащадка, якщо це не заборонено областю видимості, мають можливість звернутися до

перевизначених методів класу-предка. Для цього перед іменем перевизначеного методу записують директиву Inherited.

Приклад. Наведемо можливу реалізацію методу Square класу TCube. Знаходження площі поверхні будемо шукати за формулою

*площа поверхні куба = 6 * площа однієї грані куба,*

де одна грань куба – це квадрат. Оскільки клас TCube є нащадком класу TRect (квадрата), то площу поверхні буде обчислено за формулою

*площа поверхні куба = 6 * площа квадрата.*

```
Function TCube. Square :real;
 Begin

  Result:= 6* ( Inherited  Square );
 End;
```

Метод для знаходження площі поверхні куба

Виклик методу предка для знаходження площі квадрата

Якщо у класі нащадка перевизначено конструктор класу-предка, то першим оператором у конструкторі класу-нащадка повинен бути виклик конструктора класу-предка, з використанням директиви Inherited.

Приклад. Наведемо можливу реалізацію конструктора класу TCube.

```
Constructor TCube. Create  (P_Side:Real);
 Begin

  Inherited Create(P_Side);
 End;
```

Конструктор класу TCube

Виклик конструктора класу-предка TRect

Схему доступу до перевизначених методів класу-предка TRect в методах класу-нащадка TCube, з використанням директиви Inherited, представлено на рис. 3.3.

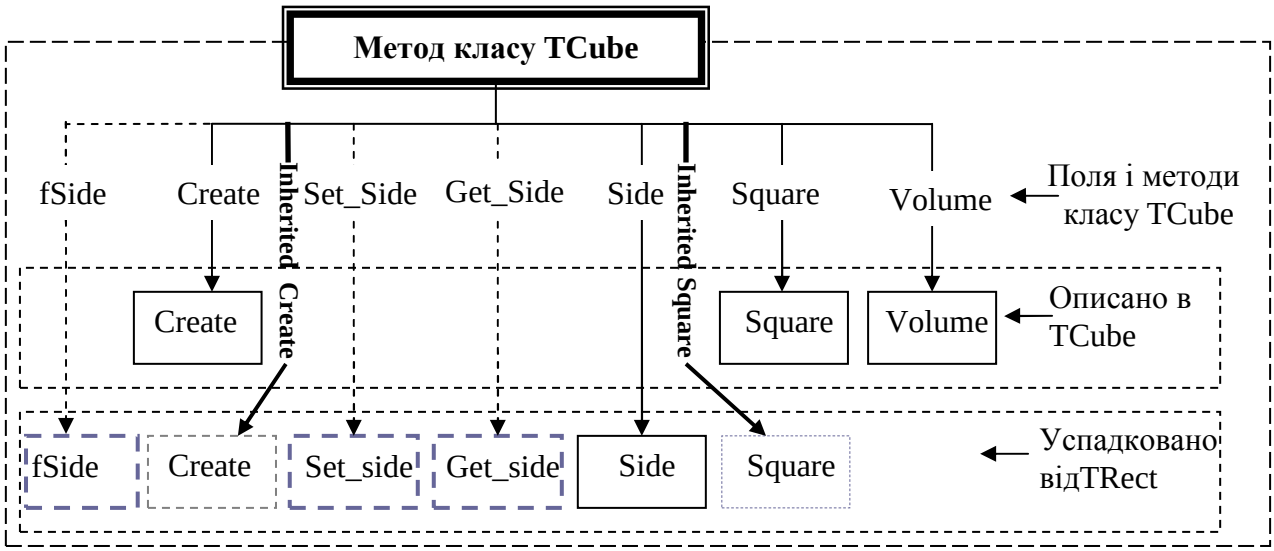


Рис. 3.3

Зауваження. Директива Inherited може бути використана тільки всередині методів класу-нащадка і не може бути застосована для об'єктів класу-нащадка.

Приклад. Наведемо повний текст модулів та програми з використанням описаних класу-предка TRect і класу-нащадка TCube.

Модуль з описанням класу-предка TRect

```
UNIT TRect_Unit;
INTERFACE
 Uses   SysUtils;
 Type
       //Клас, що представляє квадрат
    TRect = class
      Private
       fSide:real;
         Procedure Set_Side(Value:real);
         Function Get_Side:Real;
       Public
        Constructor Create(P_Side:Real);
         Property Side:Real read Get_Side write Set_Side;
         Function Square:real;
    End;
IMPLEMENTATION
    //Реалізація методів класу TRect
 Constructor TRect.Create(P_Side:Real);
 Begin
  Side:=P_Side;
```

```
 End;
 Procedure TRect.Set_Side(Value:real);
 Begin
  if Value<0 then fSide:=0
   else fSide:=Value;
 End;
 Function TRect.Get_Side:Real;
 Begin
  Result:=fSide;
 End;
 Function TRect.Square:real;
 Begin
  Result:=sqr(Side);
 End;
END.
```

Модуль з описанням класу-нащадка TCube

```
UNIT TCube_Unit;
INTERFACE
Uses TRect_Unit;
Type
  TCube = class(TRect)     // Визначення класу TCube як нащадка класу
TRect
     Constructor Create(P_Side:Real);
     Function Volume:real;
     Function Square:real;
  End;
IMPLEMENTATION
         // Реалізація методів класу TCube
 Constructor TCube.Create(P_Side:Real);
 Begin
        inherited Create(P_Side);     //Виклик конструктора предка
 End;
 Function TCube.Volume:real;
 Begin
  Result:=Side*sqr(Side);
 End;
 Function TCube.Square:real;
 Begin
  Result:=6* (inherited Square);
 End;
END.
```

Програма-клієнт

```
USES
  SysUtils,  TCube_Unit;

Var
  Cub:TCub;
BEGIN

  Cub:=TCub.Create(3);
    Writeln('---------');
    writeln(' Side     = ',Cub.Side:2:2);
    writeln(' Square   = ',Cub.Square:2:2);
    writeln(' Volume   = ',Cub.Volume:2:2);
  Readln;
  Cub.Free;

END.
```

Результати роботи програми

```
 Side      = 3.00
 Square   = 54.00
 Volume  = 27.00
```

3.4. Доступ до об’єкта типу класу-нащадка через змінну типу класу-предка

Згідно з відповідністю типів у мові Delphi, *змінній класу-предка, як покажчику, можна надавати адресу довільного об’єкта, який є екземпляром класу-нащадка.* Якщо змінна типу класу-предка містить адресу об’єкта-нащадка, то доступними через цю змінну є тільки ті члени об’єкта-нащадка, які він успадкував від цього предка.

Приклад. Проілюструємо цей прийом на прикладі розглянутих раніше класів TRect і TCube.

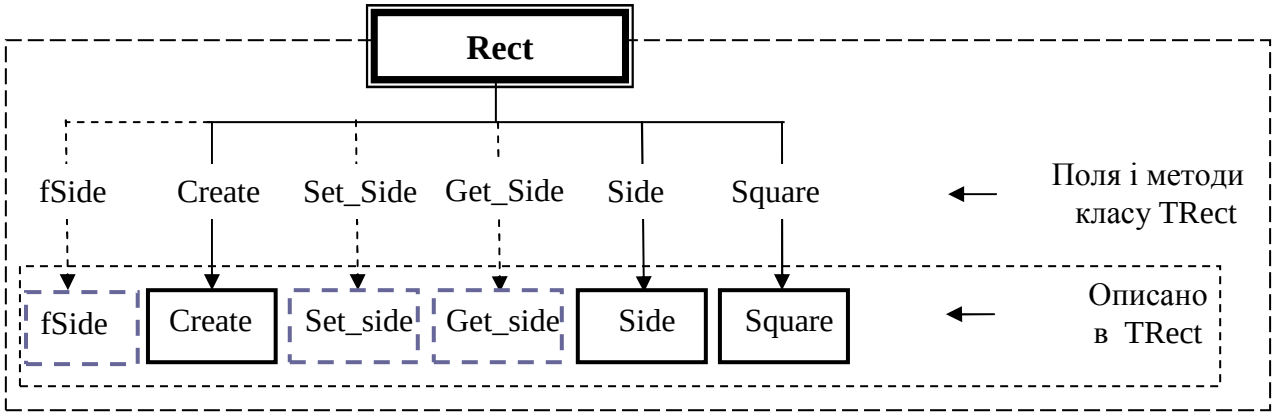
```
    TRect = class
      Private
       fSide:real;
         Procedure Set_Side(Value:real);
         Function Get_Side:Real;
      Public
```

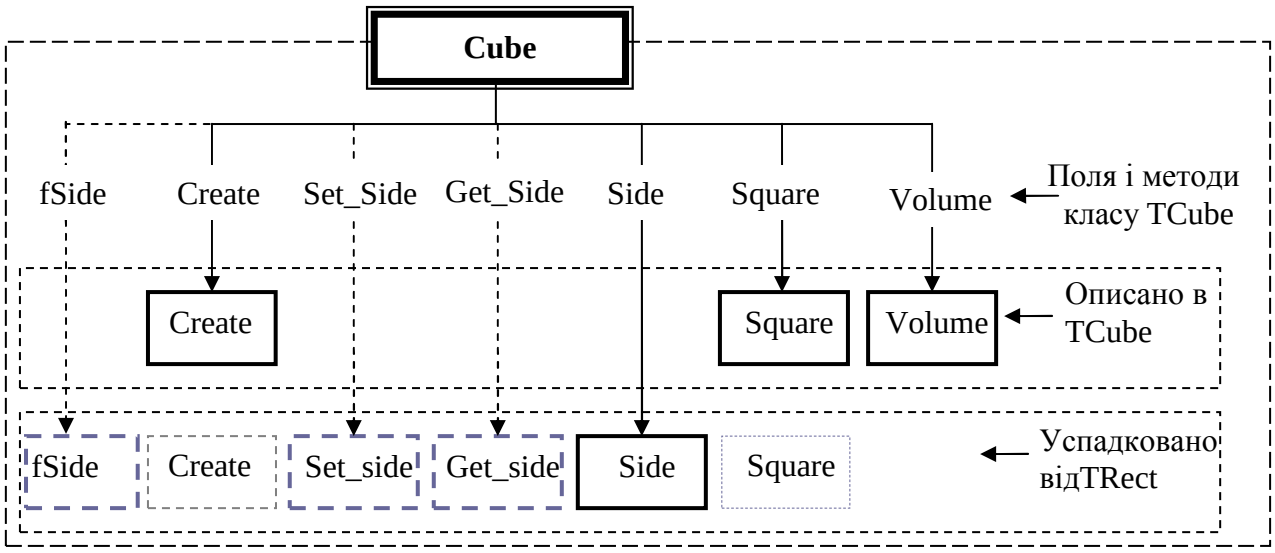
```
      Constructor Create(P_Side:Real);
       Property Side:Real read Get_Side write Set_Side;
        Function Square:real;
   End;
 TCube = class(TRect)
   Constructor Create(P_Side:Real);
   Function Volume:real;
   Function Square:real;
 End;
```

Нехай у програмі-клієнті описано два об’єкти

```
Var
  Rect : TRect;
  Cub  : TCub;
```

Розглянемо схематичне представлення об’єктів Rect і Cub.





Нехай у програмі-клієнті змінній Rect присвоїли значення Cub.

```
      Rect  :=  Cub ;
```

Як зазначалося, змінна класу-предка Rect має доступ лише до тих членів об’єкта-нащадка Cub, які клас-нащадок TCube успадкував від класу-предка TRect. Схематично такий доступ зображено нижче.

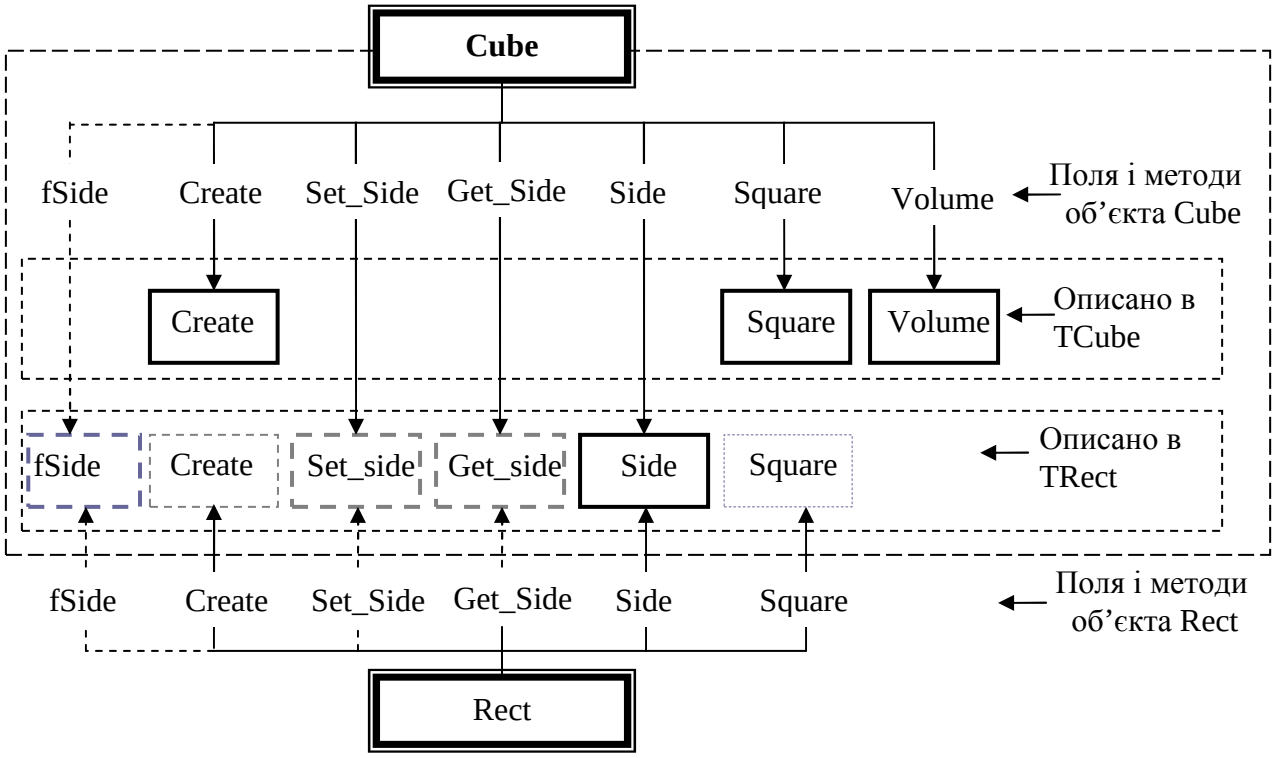


Рис. 3.4

Зауваження. Після виконання описаного оператора присвоєння, об’єкти Rect і Cub спільно використовують властивість Side. Тому наступні оператори присвоєння є еквівалентними.

Cub. Side:=5; еквівалентно Rect. Side:=5;

Але оскільки об’єкт Rect може використовувати лише поля і методи, які клас TCub успадкував від класу TRect, то наступні оператори не є еквівалентними.

Cub. Square ← Обчислення площі поверхні куба (описано в TCub)

Rect. Square ← Обчислення площі квадрата (описано в TRect)

Приклад. Розглянемо текст програми-клієнта, в якій використовується розглянутий спосіб доступу до об’єкта-нащадка.

Програма-клієнт

```
USES
  SysUtils,  TCube_Unit;
Var  Rect:TRect;
       Cub, Cub1:TCub;
BEGIN
  Cub:=TCub.Create(3);
    Writeln('---- Cub -----');
    writeln(' Cub.Side     = ',Cub.Side:2:2);
    writeln(' Cub.Square   = ',Cub.Square:2:2);
    writeln(' Cub.Volume   = ',Cub.Volume:2:2);
  Rect:=Cub;
    Writeln('----  Rect -> Cub  -----');
    writeln(' Rect.Side     = ',Rect.Side:2:2);
    writeln(' Rect.Square   = ',Rect.Square:2:2);
  writeln('--------------------------');

  Cub1:=TCub.Create(5);
    Writeln('---- Cub1 -----');
    writeln(' Cub1.Side     = ',Cub1.Side:2:2);
    writeln(' Cub1.Square   = ',Cub1.Square:2:2);
    writeln(' Cub1.Volume   = ',Cub1.Volume:2:2);
  Rect:=Cub1;
    Writeln('----  Rect -> Cub1  -----');
    writeln(' Rect.Side     = ',Rect.Side:2:2);
    writeln(' Rect.Square   = ',Rect.Square:2:2);
   writeln('--------------------------');

    Readln;
  Cub.Free;
  Cub1.Free;
END.
```

Результати роботи програми

```
---- Cub -----
 Cub.Side     = 3.00
 Cub.Square   = 54.00
 Cub.Volume   = 27.00
----  Rect -> Cub  -----
 Rect.Side     = 3.00
 Rect.Square   = 9.00
--------------------------
---- Cub1 -----
 Cub1.Side     = 5.00
```

```
 Cub1.Square   = 150.00
 Cub1.Volume   = 125.00
----  Rect -> Cub1  -----
 Rect.Side     = 5.00
 Rect.Square   = 25.00
---------------------------
```

Запитання для самоконтролю

1. *У чому полягає суть успадкування?*
2. *Що таке клас-предок?*
3. *Що таке клас-нащадок?*
4. *Для чого використовують класи-предки?*
5. *Скільки класів-предків може мати клас?*
6. *Як описати клас-нащадок?*
7. *Що таке перевизначення методів?*
8. *Як перевизначити метод класу-предка?*
9. *Як здійснити доступ до перевизначених методів класу-предка?*
10. *Чи має об'єкт-нащадок доступ до перевизначених методів класу-предка?*
11. *Які члени класу-предка є доступними для класу-нащадка?*
12. *Чи має можливість клас-нащадок змінювати область видимості членів класу-предка?*
13. *Як описати члени класу-предка, щоб вони були доступними для класу-нащадка і недоступними у програмі-клієнті?*
14. *Як описати члени класу-предка, щоб вони не були доступними як для класу-нащадка, так і для програми-клієнта?*
15. *Чи може змінна типу класу-предка приймати значення об'єкта-нащадка?*
16. *До яких полів і методів об'єкта-нащадка може здійснити доступ об'єкт-предок?*

Завдання для самостійної роботи

Створити клас-нащадок на основі класу, вимоги до якого описано у попередньому розділі. У кожному із завдань передбачити метод ToString, який дозволяє отримати рядкове представлення даних об’єкта. Створити програму-клієнт для тестування.

1. Створити клас на основі класу TMoney для роботи з грошовими сумами. Реалізувати додавання та вилучення грошової маси, вказуючи необхідну кількість у євро. Курси валют зберігаються в окремих полях.
2. Створити клас на основі класу TAngle для роботи з кутами. Реалізувати збільшення та зменшення кута на величину, яка задається у радіанах чи градусах. Передбачити метод для визначення того, чи є кут гострим, тупим або прямим.
3. Створити клас на основі класу TRational для роботи із раціональними дробами. Реалізувати збільшення, зменшення, множення і ділення числа на число, яке задається як дійсне число чи раціональний дріб.
4. Створити клас на основі класу TDate для роботи із датами у форматі “день.місяць.рік”. Реалізувати метод визначення кількості днів між двома датами та метод для порівняння дат (яка з дат іде раніше).
5. Створити клас на основі класу TFraction для роботи із десятковими дробами. Реалізувати методи порівняння чисел, представлення десяткового дробу як раціонального дробу (чисельник і знаменник при цьому не повинні містити спільних дільників).
6. Створити клас на основі класу TBankomat, який моделює роботу банкомата. Передбачити ситуацію, коли наявна кількість купюр не дозволяє видати вказану суму і розробити метод, який знаходить суму, що найменше відрізняється від необхідної і може бути виданою.
7. Створити клас на основі класу TTime для роботи із часом у форматі “години:хвилини” (зберігається київський час). Розробити метод для

знаходження часу у Москві, Токіо, Вашингтоні, Лондоні та метод для визначення того, чи у вказаному місті день або ніч.

8. Створити клас на основі класу TStudent для зберігання інформації про студента (додати поля для зберігання назви факультету, спеціальності та результатів екзаменаційної сесії). Реалізувати метод для визначення середнього балу та розміру стипендії.
9. Створити клас на основі класу TGoods, який характеризує деякий товар на складі. Додати поле для збереження курсу умовної одиниці, і розробити метод для перерахунку вартості товару в умовах інфляції.
10. Створити клас на основі класу TBus. Додати поле для збереження тарифу за проїзд, поле для збереження вартості палива, метод визначення витрат палива на основі ваги автобуса, метод визначення прибутку.
11. Створити клас на основі класу TQuadrangle, який характеризує чотирикутник. Додати поля для збереження координат вершин чотирикутника. Розробити метод для знаходження діагоналей та центру ваги цього чотирикутника.
12. Створити клас на основі класу TFirm, який характеризує приватне підприємство. Додати поля для збереження кількості працівників, розміру їх заробітної плати, вартості одиниці виробленої продукції, кількості виробленої продукції. Розробити методи для знаходження дефіциту при виплаті заробітної плати або прибутку підприємства, розміру заробітної плати у доларах та євро.
13. Створити клас на основі класу TElectronicDevice. Додати поля для зберігання адреси постачальника, допустимого віку користувача, максимально допустимий термін одноразового навантаження. Розробити метод для визначення вартості електроенергії, що споживається при вказаній кількості годин або вказаному місяці (врахувати різну кількість днів) роботи пристрою.

14. Створити клас на основі класу TCharacter, який характеризує символ. Додати поля для зберігання висоти та ширини символу. Розробити методи для визначення того, чи є даний символ цифрою, та метод для визначення регістру символу.
15. Створити клас на основі класу TDisk, який характеризує носій інформації. Додати поля для зберігання вартості, кількості допустимих операцій перезаписування, кількості вже здійснених операцій перезаписування, статусу, що визначає чи можна дописувати на диск (фіналізовано/не фіналізовано). Розробити метод для визначення кількості операцій перезаписування, що залишились, та метод визначення того, чи можна дописати вказану кількість інформації.
16. Створити клас на основі класу TElectronicDocument, який характеризує електронний документ. Додати поля для зберігання шляху до електронного документа (наприклад, текстового файлу), закодованого пароля доступу до документа (пароль закодувати довільним способом) та кількості допустимих невдалих спроб введення пароля. Розробити методи кодування/декодування пароля та перевірки правильності введеного пароля (після вказаної кількості невдалих спроб введення пароля знищити електронний документ).
17. Створити клас на основі класу TBook, який характеризує книгу. Додати поля для зберігання вартості друку одного екземпляра, витрат на рекламу. Розробити метод для визначення прибутку від реалізації всього тиражу, та необхідної кількості екземплярів, які треба продати, щоб компенсувати витрати на друк та рекламу.
18. Створити клас на основі класу TPlant. Додати поля для зберігання вартості одного кілограма плодів, кількості кілограмів, що може дати рослина, відсотку збільшення врожаю при додаванні одиниці добрив, вартості добрив, середньої вартості вирощування рослини на день (з дня посіву), часу, протягом якого рослина плодоносить. Розробити

метод для визначення доходу та прибутку від вирощування рослини (при додаванні та недодаванні добрив).

19. Створити клас на основі класу TAnimal, який характеризує тварину. Додати поля для зберігання початкової ваги та вартості тварини, вартості одного кілограму корму та одного кілограму живої ваги вирощеної тварини. Розробити метод для визначення доходу та прибутку після вказаної кількості діб.
20. Створити клас на основі класу TChemicalMatter, який характеризує хімічну речовину. Додати поля для зберігання часу приготування речовини, вартості виготовлення одного кубометра речовини та вартості одного кубометра води. Розробити метод для визначення вартості суміші при вказаній концентрації і вказаному необхідному об'ємі та визначення концентрації речовини у суміші при вказаному об'ємі і заданій кількості коштів.
21. Створити клас на основі класу TSchoolSubject, який характеризує шкільний предмет. Додати поля для зберігання вартості проведення одного уроку, тем нового матеріалу та відповідної кількості годин, що вимагає дана тема. Розробити метод для визначення кількості нерозглянутих тем, якщо буде оголошено карантин на вказану кількість днів, та метод визначення вартості проведення всього предмета.

Зразок виконання самостійної роботи

УМОВА. Створити клас на основі класу TMoney для роботи з грошовими сумами. Реалізувати додавання та вилучення грошової маси, вказуючи необхідну кількість у євро. Курси валют зберігаються в окремих полях.

Логічна структура **Гроші (додається євро)**	**Клас-нащадок** **TMoneyE**	**Клас-предок** **TMoney**	
Поле *сума*		**fSuma**	*успадковано*
Поле *курс долара*		**fDKurs**	*успадковано*
Дія *встановлення суми*		**Set_Suma**	*успадковано*
Дія *одержання суми*		**Get_Suma**	*успадковано*
Дія *встановлення курсу*		**Set_DKurs**	*успадковано*
Дія *одержання курсу*		**Get_DKurs**	*успадковано*
Поле *курс євро*	**fEKurs**		**нове поле**
Дія *встановлення курсу євро*	**Set_EKurs**		**новий метод**
Дія *одержання курсу євро*	**Get_EKurs**		**новий метод**
	Create	Create	***перевизначається***
Властивість *сума*		**Suma**	*успадковано*
Властивість *курс долара*		**Dkurs**	*успадковано*
Дія *пошук курсу (сума+100)*		**SearchDKurs**	*успадковано*
Властивість *курс євро*	**EKurs**		**нова властивість**
Дія *збільшення суми в євро*	**Add_Sum**	Add_Sum	***перевизначається***
Дія *зменшення суми в євро*	**Sub_Sum**	Sub_Sum	***перевизначається***
Дія *одержання рядкового представлення*	**ToString**	ToString	***перевизначається***

Логічна структура	Структура даних
Гроші (додається євро) Поле *курс євро* Дія *встановлення курсу євро* Дія *зчитування курсу євро* ***Властивість*** Курс євро Дія *збільшення суми (в євро)* Дія *зменшення суми (в євро)* Дія *одержання рядкового представлення даних*	TMoneyE=class(TMoney) Private fEKurs:real; Procedure Set_EKurs(value:real); Function Get_EKurs:Real; Public Constructor Create(P_Suma, P_DKurs, P_EKurs:Real); Property EKurs:Real read Get_EKurs write Set_EKurs; Procedure Add_Sum(value:real); Function Sub_Sum(value:real):boolean; Function ToString:String; End;

ПРОГРАМНА РЕАЛІЗАЦІЯ

Текст модуля

```
UNIT UnitTMoneyE;
INTERFACE
 Uses SysUtils, UnitTMoney;
  {- - - - - - -  Опис класу TMoneyE  - - - - - - -  }
 Type
  TMoneyE=class(TMoney)
    Private
      fEKurs:real;
       Procedure Set_EKurs(value:real);
       Function Get_EKurs:Real;
    Public
       Constructor Create(P_Suma,P_DKurs,P_EKurs:Real);
      Property EKurs:Real read Get_EKurs write Set_EKurs;
       Procedure Add_Sum(value:real);
       Function Sub_Sum(value:real):boolean;
      Function ToString:String;
  End;

IMPLEMENTATION
```

```
                                                    {  Конструктор класу }
Constructor TMoneyE.Create(P_Suma,P_DKurs,P_EKurs:Real);
 Begin
  inherited Create(P_Suma,P_DKurs);
  EKurs:=P_EKurs;
 End;
                                          {  Метод встановлення курсу долара  }
Procedure TMoneyE.Set_EKurs(value:real);
 Begin
  if value>=0 then
   fEKurs:=value
  else
   fEKurs:=0;
 End;
                                            {  Метод зчитування курсу долара  }
Function TMoneyE.Get_EKurs:Real;
 Begin
  Result:=fEKurs;
 End;
                                      {Збільшення суми на value Євро}
Procedure TMoneyE.Add_Sum(value:real);
 Begin
  if (value>0) then
   Suma:=Suma+(value*EKurs);
 End;
                                           {  Зменшення суми на value Євро }
Function TMoneyE.Sub_Sum(value:real):boolean;
 Begin
  if (value>0) then
   begin
    if Suma>=(value*EKurs)/DKurs then
     begin
      Suma:=Suma-(value*EKurs);
      Result:=true;
     end
    else
     Result:=false;
   end
  else
   Result:=False;
 End;
                                    {  Одержання рядкового представлення даних }
Function TMoneyE.ToString:String;
 Begin
```

```
   Result:=(inherited ToString) +
               '; '+ ' Evro   ='+floattostrf(Suma/EKurs,ffFixed,6,2)+
               ';  EKurs   ='+floattostrf(EKurs,ffFixed,6,2);
  End;
END.
```

Програма-клієнт

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
USES
  SysUtils,
  UnitTMoneyE;

VAR
    MoneyE : TMoneyE;
BEGIN
   MoneyE:=TMoneyE.Create(100,8.5,11);
   writeln('--- Create(100,8.5,11) ---');
    writeln(MoneyE.ToString);
   writeln('--- Add_Sum(150) ---');
    MoneyE.Add_Sum(150);
    writeln(MoneyE.ToString);
   writeln('--- Sub_Sum(30) ---');
    MoneyE.Sub_Sum(30);
    writeln(MoneyE.ToString);
   writeln('--- Suma:=-300 ---');
    MoneyE.Suma:=-300;
    writeln(MoneyE.ToString);
   writeln('--- Suma:=300 ---');
    MoneyE.Suma:=300;
    writeln(MoneyE.ToString);
   writeln('--- EKurs:=11.5 ---');
    MoneyE.EKurs:=11.5;
    writeln(MoneyE.ToString);
   writeln('--- SearchDKurs ---');
    writeln(MoneyE.SearchDKurs:6:2);
   writeln('--- DKurs:=SearchDKurs ---');
    MoneyE.DKurs:=MoneyE.SearchDKurs;
    writeln(MoneyE.ToString);
  readln;
   MoneyE.Free;
END.
```

Результат роботи програми

```
--- Create(100,8.5,11) ---
 Dollars =11,76;  DKurs =8,50;  Grivna =100,00;  Evro   =9,09;  EKurs   =11,00

--- Add_Sum(150) ---
Dollars =205,88;  DKurs =8,50;  Grivna =1750,00;  Evro   =159,09;  EKurs=11,00

--- Sub_Sum(30) ---
Dollars =167,06;  DKurs =8,50;  Grivna =1420,00;  Evro  =129,09;  EKurs=11,00

--- Suma:=-300 ---
 Dollars =0,00;  DKurs   =8,50;  Grivna =0,00;  Evro   =0,00;  EKurs   =11,00

--- Suma:=300 ---
Dollars =35,29;  DKurs   =8,50;  Grivna =300,00;  Evro   =27,27;  EKurs   =11,00

--- EKurs:=11.5 ---
Dollars =35,29;  DKurs   =8,50;  Grivna =300,00;  Evro   =26,09;  EKurs   =11,50

--- SearchDKurs ---
 11.33

--- DKurs:=SearchDKurs ---
Dollars =35,29;  DKurs   =11,33;  Grivna =400,00;  Evro   =34,78;  EKurs =11,50
```

РОЗДІЛ 4

ПОЛІМОРФІЗМ – ТРЕТІЙ ОСНОВНИЙ ПРИНЦИП ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ. ВІРТУАЛЬНІ ТА ДИНАМІЧНІ МЕТОДИ

4.1. Поняття поліморфізму. Віртуальні та динамічні методи

Стосовно властивості успадкування всі методи класу можна поділити на такі групи: статичні, віртуальні та динамічні.

Вид методу	**Директива, яка використовується при описанні**	**Приклади заголовків методів**
Статичні	–	Procedure SomeProcedure;
		Function SomeFunc:Real;
Віртуальні	Virtual	Procedure SomeProcedure; Virtual;
		Function SomeFunc:Real; Virtual
Динамічні	Dynamic	Procedure SomeProcedure; Dynamic;
		Function SomeFunc:Real; Dynamic;

Статичні методи – це методи, при описанні яких не використовуються жодні директиви. Саме такі методи ми розглядали до цього часу. При успадкуванні нащадок може повністю перевизначити статичний метод, успадкований від предка, лише визначивши метод з таким самим іменем. Раніше зазначалося, що для змінної типу класу-предка доступними є тільки ті члени об'єкта-нащадка, які клас-нащадок успадкував від класу-предка. Важливим є той факт, що перевизначені у класі-нащадку статичні методи недоступні для об'єкта предка (див. рис. 3.4). Але в мові програмування Delphi передбачено механізм віртуальних та динамічних методів, який дозволяє у класі-нащадка перевизначати методи класу-предка так, що коли змінна типу клас-предок містить значення типу класу-нащадка, тоді при звертанні до віртуального або динамічного методу активується реалізація методу класу-нащадка. Оскільки при успадкуванні віртуальні та динамічні методи принципово нічим не відрізняються, то надалі будемо розглядати

тільки віртуальні методи. Загальна схема використання цих методів наступна.

Описання методу у класі предка	Описання методу у класі нащадка
<**клас-предок**>=class …….... <метод> **Virtual**; …… End;	<**клас-нащадок**>=class(<**клас-предок**>) …….. <метод> **Override**; …….. End;

Отже, при описанні методу у класі-предка метод повинен описуватися з директивою Virtual або Dynamic. При описанні відповідного методу в класі-нащадка описання методу обов'язково повністю має збігатися з описанням методу в класі-предка і доповнюватися директивою Override.

Приклад. Розглянемо описаний раніше клас TRect, який представляє квадрат.

```
TRect = class
 Private
  fSide:real;
    Procedure Set_Side(Value:real);
    Function Get_Side:Real;
 Public
  Constructor Create(P_Side:Real);
   Property Side:Real read Get_Side write Set_Side;
     Function Square:real;
End;
```

Модифікуємо описання класу, описавши метод знаходження площі квадрата Square як віртуальний.

```
TRect = class
 Private
  fSide:real;
    Procedure Set_Side(Value:real);
    Function Get_Side:Real;
 Public
  Constructor Create(P_Side:Real);
   Property Side:Real read Get_Side write Set_Side;
     Function Square:real; Virtual;
End;
```

При описанні класу-нащадка TCube визначимо метод знаходження площі поверхні куба Square, з використанням директиви Override.

```
TCube = class(TRect)
  Constructor Create(P_Side:Real);
  Function Volume:real;
   Function Square:real;      Override;
End;
```

Нехай у програмі-клієнті описано два об’єкти Rect і Cub.

```
Var
 Rect : TRect;
 Cub  : TCub;
```

Після їх створення

Rect. Square ← Обчислення площі квадрата (описано в TRect)

Cub. Square ← Обчислення площі поверхні куба (описано в TCub)

Надамо змінній Rect значення Cub.

```
      Rect   :=   Cub  ;
```

Зараз при виклику методу Square об’єкта Rect буде викликатися метод Square, описаний у класі TCub

Rect.Square ← Обчислення площі поверхні куба(описано в класі-нащадку TCub)

Схематично такий доступ зображено на рис. 4.1.

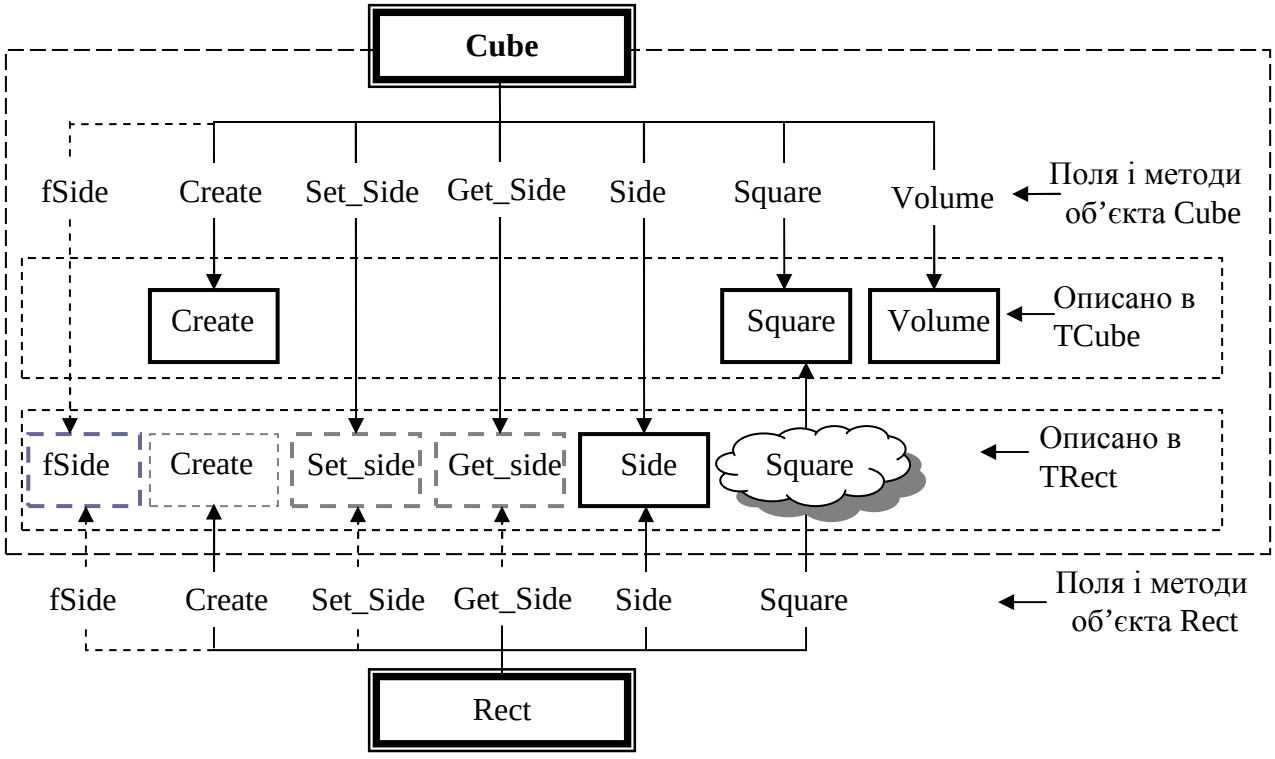


Рис. 4.1

Отже, як показує приклад, коли змінна типу класу-предка містить адресу певного об'єкта, то при виклику віртуального методу його реалізація буде визначатися фактичним типом цього об'єкта, а не типом, що використовувався при описанні цієї змінної. Принцип, згідно якого визначається метод, що відповідає віртуальному методу, називається *поліморфізмом*.

Оскільки визначити реалізацію методу, який відповідає віртуальному, наперед неможливо, то виникає необхідність визначати цей метод під час роботи програми. Механізм вибору методу під час виконання програми називається *пізнім зв'язуванням*. Для цього призначені таблиці віртуальних методів, які автоматично створюються компілятором. Таблиця віртуальних методів створюється для кожного класу. У цій таблиці зберігаються адреси всіх віртуальних методів класу, незалежно від того успадковані вони від предка чи перевизначені в даному класі. Якщо компілятор зустрічає виклик віртуального методу, то замість безпосереднього виклику методу підставляється код, який звертається до таблиці віртуальних методів і відшукує необхідний метод, що потрібно викликати.

Приклад. Наведемо повний текст модулів та програми-клієнта з використанням розглянутих класів TRect і TCube.

Текст модуля, з описанням класу TRect

```
UNIT TRect_Unit;
INTERFACE
 Uses   SysUtils;
 Type
              //Клас, що представляє квадрат
    TRect = class
      Private
       fSide:real;
         Procedure Set_Side(Value:real);
         Function Get_Side:Real;
      Public
       Constructor Create(P_Side:Real);
        Property Side:Real read Get_Side write Set_Side;
          Function Square:real; virtual;       //Описано як віртуальний метод
    End;
```

```
IMPLEMENTATION
    //Реалізація методів класу TRect
 Constructor TRect.Create(P_Side:Real);
 Begin
  Side:=P_Side;
 End;
 Procedure TRect.Set_Side(Value:real);
 Begin
  if Value<0 then fSide:=0
   else fSide:=Value;
 End;
 Function TRect.Get_Side:Real;
 Begin
  Result:=fSide;
 End;
 Function TRect.Square:real;
 Begin
  Result:=sqr(Side);
 End;
END.
```

Текст модуля, з описанням класу TCube

```
UNIT TCube_Unit;
INTERFACE
Uses TRect_Unit;
Type
  TCube = class(TRect)
    Constructor Create(P_Side:Real);
    Function Volume:real;
    Function Square:real; Override;   //Перевизначено віртуальний метод
  End;
IMPLEMENTATION
 // Реалізація методів класу TCube
 Constructor TCube.Create(P_Side:Real);
 Begin
  inherited Create(P_Side);      //Виклик конструктора Create, що описано у
                                 // класі-предка TRect
 End;
 Function TCube.Volume:real;
 Begin
  Result:=Side*sqr(Side);
 End;
 Function TCube.Square:real;
 Begin
  Result:=6* (inherited Square);
```

```
 End;
END.
```

Текст програми-клієнта

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  TRect_Unit,
  TCube_Unit;

Var
  Rect:TRect;
  Cube:TCube;
BEGIN
  Rect:=TRect.Create(2);
    writeln('- - - - Rect : TRect - - - -');
    writeln(' Rect.Side     = ',Rect.Side:2:2);
    writeln(' Rect.Square   = ',Rect.Square:2:2);  //Обч. площа квадрата
    Writeln('---------------------------');
  Cube:=TCube.Create(3);
  Rect := Cube ;
   writeln('- - - - Rect -> Cub - - - -');
   writeln(' Rect.Side   = ',Rect.Side:2:2); //Звертання еквівалентне Cub.Side
   writeln(' Rect.Square   = ',Rect.Square:2:2); //Обч. площа поверхні куба
                                                 ///(еквівалентно виклику Cub.Square
  Readln;
  Cube.Free;
END.
```

Результати роботи програми

```
- - - - Rect : TRect - - - -
 Rect.Side     = 2.00
 Rect.Square   = 4.00   {площа квадрата з сторою 2}
---------------------------
- - - - Rect -> Cub - - - -
 Rect.Side     = 3.00
 Rect.Square   = 54.00  {площа поверхні куба з сторою 3}
```

4.2. Абстрактні методи

Властивості успадкування та поліморфізму дають можливість визначати клас, який є загальним описанням деякої групи подібних об’єктів реальної дійсності. Але дуже часто, описуючи такий клас, недоцільно чи

навіть неможливо вказати реалізацію деяких спільних для всієї групи методів. У цьому випадку методи описують як *абстрактні* і реалізації для них не вказують. Опис абстрактного методу здійснюється з використанням директиви Abstract. Клас, що містить принаймні один абстрактний метод, також називають абстрактним. Характерною особливістю абстрактного класу є те, що для нього не може бути створено об'єкта, тобто він може використовуватися тільки як клас-предок. У класі нащадка абстрактні методи повинні бути перевизначені, і для кожного із них необхідно вказати реалізацію. Загальна схема використання абстрактних методів наступна.

Описання методу у класу-предка	Описання методу у класу-нащадка
<**клас-предок**>=class …….... <метод> **Virtual**; Abstract; {реалізація цього методу не наводиться} …… End;	<**клас-нащадок**>=class(<**клас-предок**>) …….. <метод> **Override**; {реалізація методу наводиться} …….. End;

Приклад. Опишемо клас TFigure, який є предком класів TRect (представляє квадрат) та TCircle (представляє круг). Обидві фігури характеризуються однією величиною Data (сторона квадрата, чи радіус круга) та методом відображення цієї величини Show_Data, однак способи знаходження площі цих фігур відрізняються.

Текст модуля, з описанням абстрактного класу-предка TFigure

```
UNIT TFigure_Unit;
INTERFACE
Type
  TFigure=class
   Private
    fData:Real;
    Procedure Set_Data(Value:real);
    Function Get_Data:Real;
   Public
    Constructor Create(P_Data:Real);
    Property Data:Real read Get_Data write Set_Data;
    Procedure Show_Data;
     Function Square:Real; Virtual; Abstract;      //Метод абстрактний
  End;                                             //(реалізація не наводиться)
```

```
IMPLEMENTATION
 Constructor TFigure.Create(P_Data:Real);
  Begin
    Data:=P_Data;
  End;
 Procedure TFigure.Set_Data(Value:real);
  Begin
   if Value<0 then fData:=0
    else fData:=Value;
  End;
 Function TFigure.Get_Data:Real;
  Begin
   Result:=fData;
  End;
 Procedure TFigure.Show_Data;
  Begin
   writeln('Data = ',Data:8:2);
  End;
END.
```

Текст модуля, з описанням класу-нащадка TRect

```
UNIT TRect_Unit;
INTERFACE
Uses TFigure_Unit;
Type
  TRect=class(TFigure)
   Constructor Create(P_Data:Real);
   Function Square:Real; Override;        //Метод перевизначається
  End;
IMPLEMENTATION
Constructor TRect.Create(P_Data:Real);
 Begin
   inherited Create(P_Data);
 End;
Function TRect.Square:Real;               //Наводиться реалізація методу
 Begin
   Result:=sqr(Data);
 End;
END.
```

Текст модуля, з описанням класу-нащадка TCircle

```
UNIT Tcircle_Unit;
INTERFACE
 Uses Tfigure_Unit;
 Type
  TCircle=class(TFigure)
   Constructor Create(P_Data:Real);
```

```
   Function Square:Real; override;          //Метод перевизначається
  End;
IMPLEMENTATION
 Constructor TCircle.Create(P_Data:Real);
  Begin
    inherited Create(P_Data);
  End;
 Function TCircle.Square:Real;          /Наводиться реалізація методу
  Begin
    Result:=pi*sqr(Data);
  End;
END.
```

Програма-клієнт

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
USES
  SysUtils,
  TCircle_Unit,
  TRect_Unit,
  TFigure_Unit;

VAR  Figure : TFigure;
        Rect    : TRect;
        Circle  : TCircle;
BEGIN
Rect:=TRect.Create(2);    //Створення об'єкта класу TRect(сторона квадрата =2)
Circle:=TCircle.Create(10); // Створення об'єкта класу  TCircle (радіус круга=10)
      {Змінна Figure значення типу класу-нащадка TRect}
  Figure:=Rect;
  writeln('- - - -  Figure -> Rect:TRect - - -');
  writeln(' Figure.Data  =', Figure.Data :8:2);      // Виведення сторони квадрата
  writeln(' Figure.Square =', Figure.Square :8:2);  // Виведення площі квадрата
  writeln('-------------------------');
   {Змінна Figure отримує значення типу класу-нащадка TCircle }
 Figure:=Circle;
  writeln('- - - -  Figure-> Circle:TCircle - - -');
  writeln(' Figure.Data   =', Figure.Data :8:2);          // Виведення радіуса круга
  writeln(' Figure.Square =', Figure.Square :8:2);    // Виведення площі круга
 readln;
 Rect.Free;
 Circle.Free;
END.
```

Результати роботи програми

```
- - - -  Figure -> Rect:TRect - - -
 Figure.Data   =    2.00
 Figure.Square =    4.00  { площа квадрата }
------------------------
- - - -  Figure-> Circle:TCircle - - -
 Figure.Data   =   10.00
 Figure.Square =  314.16    { площа круга }
```

Зауваження. При описі групи споріднених об'єктів можна дати наступні рекомендації. Спільні для всіх об'єктів характеристики описуються за допомогою статичних методів, а ті, які мають індивідуальні особливості в кожного з об'єктів, описуються за допомогою віртуальних методів. Віртуальні методи при цьому можуть бути абстрактними. Успадковуючи цей клас-предок, кожен з класів-нащадків визначає власну реалізацію успадкованих віртуальних методів предка.

4.3. Визначення типу об'єкта. Оператор is

Часто описуючи клас-предок, описують поля, властивості та методи, які можуть перевизначатися у класах-нащадках, і, хоч їхні назви збігаються, вони мають інший зміст. Наприклад, властивість Data у класі TFigure має загальний невизначений зміст, у класі TRect властивість Data – це довжина сторони квадрата, а у класі TCircle властивість Data – це радіус круга. Або ж раніше розглянутий метод Square у класі TRect використовується для знаходження площі квадрата, а перевизначений метод Square у класі нащадка TCube використовується для знаходження площі поверхні куба. Оскільки змінна типу класу-предка може отримувати адресу як екземпляра цього класу, так і будь-якого екземпляра класу-нащадка, то часто виникає необхідність визначати тип об'єкта, адресу якого віна містить. У Delphi для

перевірки належності об’єкта (екземпляра деякого класу) до того чи іншого класу (типу) використовується оператор is

```
<Ім’я об’єкта>  is  <Ім’я класу>
```

Цей оператор повертає значення логічного типу True, якщо об’єкт є екземпляром вказаного класу, і False – в іншому випадку.

Приклад. Розглянемо раніше описані класи TFigure, TRect, TCircle. Нехай у програмі-клієнті описано об’єкти (змінні).

```
VAR  Figure : TFigure;  {предок}
     Rect   : TRect;    {нащадок TFigure }
     Circle : TCircle;  {нащадок TFigure }
```

та виконано фрагмент програми

```
Rect:=TRect.Create(2);
Circle:=TCircle.Create(10);
Figure:=Rect;
```

Розглянемо значення наступних операторів

Оператор	Значення	Примітки
Figure is TRect	True	Бо Figure містить адресу об’єкта Rect типу TRect
Figure is TCircle	False	Бо Figure не містить адресу об’єкта типу TCircle
Figure is TFigure	True	Бо Figure описано як об’єкт типу TFigure

Якщо ж виконати оператор присвоєння

```
Figure:=Circle;
```

то, очевидно, значення операторів будуть такими

Оператор	Значення	Примітки
Figure is TRect	False	Бо Figure не містить адресу об’єкта типу TRect
Figure is TCircle	True	Бо Figure містить адресу об’єкта Circle типу TCircle
Figure is TFigure	True	Бо Figure описано як об’єкт типу TFigure

Зауваження. У розглянутому прикладі недоцільно застосувати оператор is для визначення того, чи є Figure типу TFigure, оскільки TFigure є абстрактним класом, і не можна створювати об’єктів цього класу. Але слід пам’ятати, що хоча змінна типу класу-предка може містити адреси об’єктів-нащадків, вона формально має тип класу-предка. Тому оператор

Figure is TFigure

завжди повертає True. Отже, якщо виникає необхідність у перевірці типу об’єкта, адресу якого містить змінна типу класу-предка, то умова формується так, щоб перевіряти те, чи є об’єкт екземпляром класу-нащадка. У даному випадку це умови (Figure is TCircle) та (Figure is TFigure).

Приклад. На основі описаних раніше класів TFigure, TRect, TCircle, описати масив типу TFigure, який містить два об’єкти-квадрати (з довжинами сторін 2 і 7), та три об’єкти-круга (з радіусами 5, 2, 9). Вивести на екран для об’єктів-квадратів довжини діагоналей, а для об’єктів-кругів – довжини кіл, що їх обмежують.

Текст програми-клієнта

```
PROGRAM Project_Is;
{$APPTYPE CONSOLE}
USES
        SysUtils,
   TCircle_Unit,
   TRect_Unit,
   TFigure_Unit;
VAR
       A:array[1..5] of TFigure;
        i:integer;
        r:real;
BEGIN
 A[1]:=TRect.Create(2);
 A[2]:=TRect.Create(7);    //Створення двох об’єктів-квадратів
 A[3]:=TCircle.Create(5);
 A[4]:=TCircle.Create(2);   //Створення трьох об’єктів-кругів
 A[5]:=TCircle.Create(9);

 for i:=1 to 5 do
  begin
   if (A[i] is TRect) then          {Перевірка того, чи є A[i] об’єктом класу TRect}
    begin
     r:=sqrt(2*sqr(A[i].Data));            //Обчислення довжини діагоналі
     writeln('Діагональ квадрата = ',r:2:2);
    end
   else
    begin
     r:=sqrt(2*pi*A[i].Data);              //Обчислення довжини обмежуючого кола
     writeln('Довжина кола = ',r:2:2);
```

```
   end;
  end;
  Readln;
 for i:=1 to 5 do
   A[i].Free;                          //Знищення об'єктів
END.
```

Результати роботи програми

```
Діагональ квадрата = 2.83
Діагональ квадрата = 9.90
Довжина кола = 5.60
Довжина кола = 3.54
Довжина кола = 7.52
```

4.4. Приведення типів класів. Оператор as

Як було зазначено раніше, при описі групи споріднених об'єктів описується клас-предок, що містить спільні для всієї групи властивості і методи, а потім описуються класи-нащадки, які містять властиві тільки їм властивості та методи. При цьому змінна типу класу-предка може отримувати адреси об'єктів-нащадків, але не може здійснювати доступ до членів об'єктів-нащадків, які не успадковані від предка, а описані у відповідних класах-нащадках. Тим не менше, при розв'язанні прикладних задач іноді виникає необхідність через змінну типу класу-предка отримати доступ до власних членів об'єкта-нащадка, адресу якого вона містить. У Delphi такий доступ здійснюється завдяки приведенню змінної типу класу-предка до типу класу-нащадка з використанням оператора as

```
<Ім'я об'єкта>  as  <Ім'я класу>
```

Приклад. Проілюструємо цей підхід на прикладі розглянутих раніше класів TRect і TCube.

```
  TRect = class
    Private
     fSide:real;
       Procedure Set_Side(Value:real);
       Function Get_Side:Real;
    Public
     Constructor Create(P_Side:Real);
```

```
      Property Side:Real read Get_Side write Set_Side;
       Function Square:real;            //Обчислення площі квадрата
   End;
TCube = class(TRect)
  Constructor Create(P_Side:Real);
  Function Volume:real;
  Function Square:real;                 //Обчислення площі поверхні куба
End;
```

Нехай у програмі-клієнті описано два об'єкти

```
Var
 Rect : TRect;
 Cub  : TCub;
```

Надамо змінній Rect значення Cub

```
Rect := Cub
```

Як зазначалося, після такого присвоєння об'єкт Rect має доступ лише до тих членів об'єкта Cub, які TCub успадкував від TRect. Схематично об'єкти Rect і Cub зображено на рис. 4.2.

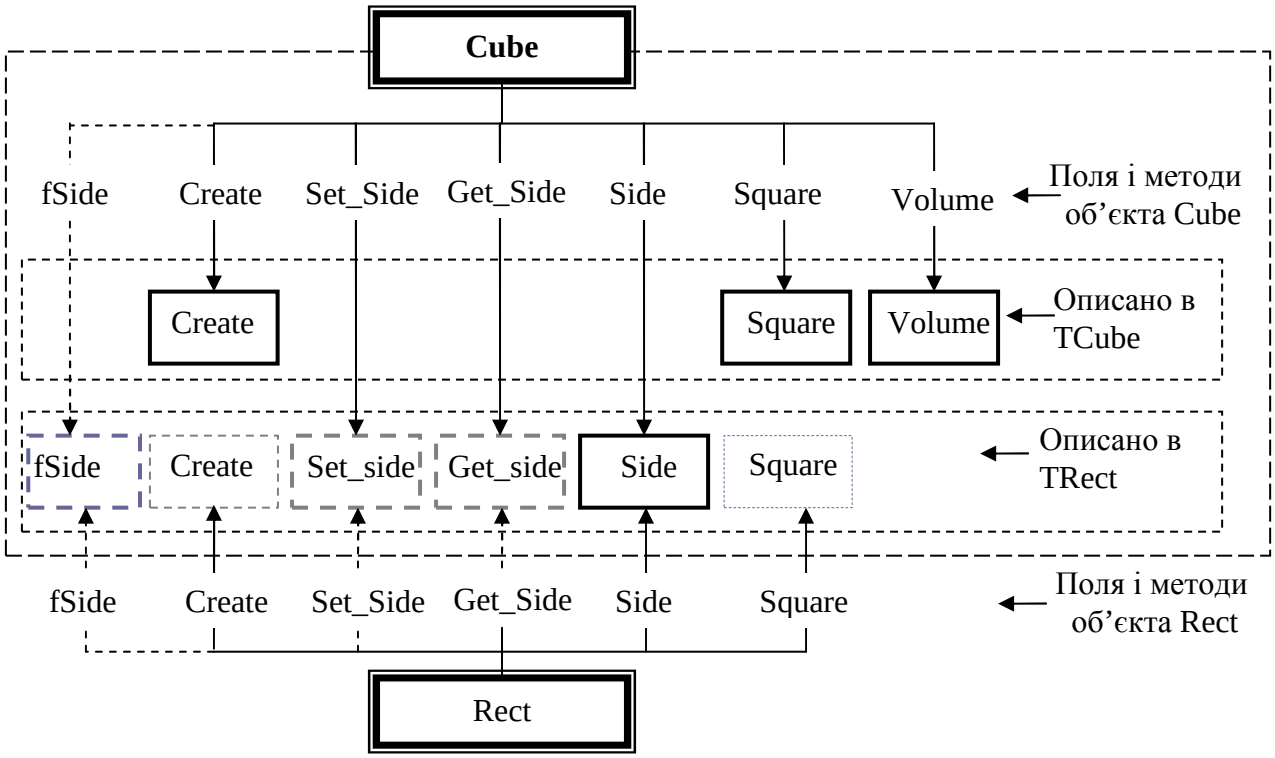


Рис. 4.2

Приведемо тип змінної Rect до типу TCub за допомогою оператора as

```
Rect as TCub
```

Схематично таке приведення представлено на рис. 4.3.

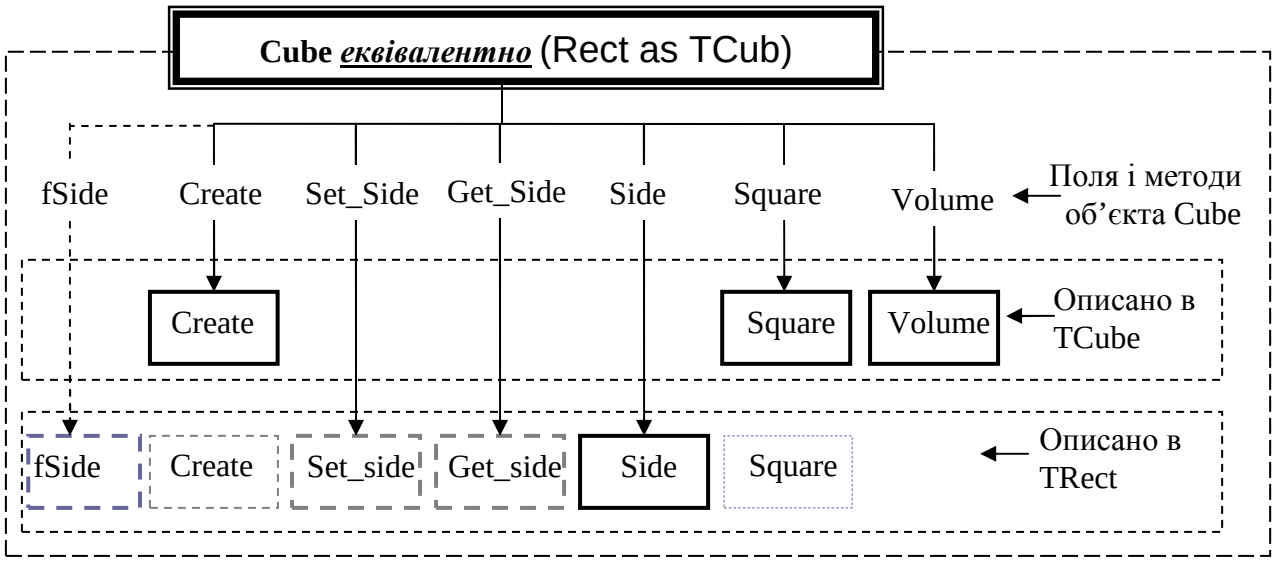


Рис. 4.3

Отже, після такого приведення об'єкт Rect фактично "перетворюється" в об'єкт Cub, а тому має доступ до всіх членів об'єкта Cub. Для порівняння наведемо декілька викликів методів.

Тип об'єкта	**Виклик методу**	**Дія**
TRect	Rect .Square	Обчислення площі квадрата
TCub	(Rect as TCub). Square	Обчислення площі поверхні куба
TRect	Rect .Volume	**Виклик неможливий** (у TRect не описано методу Volume)
TCub	(Rect as TCub).Volume	Знаходження об'єму куба

Приклад. На основі описаних раніше класів TRect і TCube описати масив типу TRect, який містить два об'єкти-квадрати (з довжинами сторін 2 і 7), та три об'єкти-куби (з довжинами ребер 5, 2, 9). Вивести на екран для об'єктів-квадратів площі цих квадратів, а для об'єктів-кубів – їх об'єми.

Текст програми-клієнта

```
PROGRAM Project_array;
{$APPTYPE CONSOLE}

USES
  SysUtils,
```

```
  TCube_Unit,
  TRect_Unit;
VAR
  A:array[1..5] of TRect;  //Описання масиву для збереження як
                           //об'єктів типу TRect, так і об'єктів-нащадків типу
TCircle
         i:integer;
         r:real;
BEGIN
 A[1]:=TRect.Create(2);
 A[2]:=TRect.Create(7);      //Створення двох об'єктів-квадратів
 A[3]:=TCube.Create(5);
 A[4]:=TCube.Create(2);      //Створення трьох об'єктів-кубів
 A[5]:=TCube.Create(9);
 for i:=1 to 5 do
  begin
    if (A[i] is TCube) then
    begin
     r:= ( A[i] as TCube ) .Volume;       //Обчислення об`єму куба

     writeln('Об`єм куба = ',r :2:2);
    end
   else
    begin
     r:=A[i].Square;          //Обчислення  площі квадрата
     writeln('Площа квадрата = ',r:2:2);
    end;
  end;
  Readln;
 for i:=1 to 5 do
   A[i].Free;                             //Знищення всіх об'єктів
END.
```

Об’єкт типу **TRect**

Об’єкт типу **TCube**

Результат роботи програми

```
Площа квадрата = 4.00

Площа квадрата = 49.00

Об`єм куба = 125.00

Об`єм куба = 8.00

Об`єм куба = 729.00
```

Запитання для самоконтролю

1. *Які види методів можна виділити відносно успадкування?*
2. *Що таке віртуальні методи? Як вони описуються?*
3. *Чим відрізняються віртуальні методи від статичних?*
4. *Що таке абстрактні методи?*
5. *Чи можуть віртуальні методи бути абстрактними?*
6. *Як називають клас, що містить абстрактні методи?*
7. *Чи може клас містити крім абстрактних інші методи?*
8. *Чи можна створити об'єкт абстрактного класу?*
9. *У яких випадках використовують абстрактні класи?*
10. *Чи можна визначити тип об'єкта-нащадка, якщо змінна типу клас-предок містить його адресу?*
11. *До яких членів об'єкта-нащадка має доступ змінна типу класу-предка, якщо вона містить його адресу?*
12. *Як можна привести тип змінної класу-предка до типу класу-нащадка?*

Завдання для самостійної роботи

Примітки. *У кожному із завдань при описі класів самостійно визначити необхідні поля, властивості та методи вводу/виводу. Деякі методи класу-предка можуть бути віртуальними і абстрактними. У програмі-клієнті для збереження сукупності об'єктів використати масив.*

1. Створити клас TPrism, який представляє правильну призму і містить методи для знаходження площі поверхні та об'єму. На основі цього класу створити класи-нащадки TPrism3 та TPrism4, які представляють правильну трикутну та чотирикутну призми. З клавіатури вводиться дані для створення правильної трикутної та чотирикутної призми. На їх основі поступово створити *m* правильних призм (трикутних та

чотирикутних), об’єм кожної з яких на 5 більше попередньої. Для трикутних призм знайти сумарний об’єм, а для чотирикутних – суму площ поверхні.

2. Створити клас TPrism, який представляє призму (в основі опуклий багатогранник) і містить методи для знаходження площі поверхні та суми ребер. На основі цього класу створити класи-нащадки, що представляють пряму та похилу призми. Випадковим чином згенерувати дані для n призм обох видів. Визначити, чи є серед прямих призм така, площа поверхні якої дорівнює площі поверхні першої по порядку похилої призми. Для похилих призм знайти суму всіх сум ребер.
3. Створити клас TNumber з віртуальними методами для знаходження суми цифр та знаходження першої/останньої цифри. На основі цього класу створити класи-нащадки TIntNumber та TRealNumber, у яких реалізовано перевизначені віртуальні методи. Створити m об’єктів цілих чисел та n об’єктів дійсних чисел (дані згенерувати випадковим чином). Знайти суму перших цифр цілих чисел та суму останніх цифр дійсних чисел.
4. Створити клас TBody, який представляє просторову геометричну фігуру з методами обчислення площі поверхні та об’єму. На основі цього класу створити класи нащадки TParallelepiped та TBall. Випадковим чином створити певну кількість паралелепіпедів та куль, щоб їх сумарна кількість дорівнювала n. Знайти сумарну площу поверхонь усіх геометричних тіл.
5. Створити клас TTriangle з віртуальними методами для обчислення площі та периметру. На основі цього класу створити класи, які представляють рівносторонні, прямокутні та рівнобедрені трикутники. Випадковим чином створити певну кількість трикутників кожного виду, щоб їх сумарна кількість дорівнювала n. Для рівносторонніх та

прямокутних обчислити суму площ, а для рівнобедрених – суму всіх периметрів.

6. Створити клас TQuadrangle, який представляє чотирикутник і містить віртуальні методи для обчислення площі та периметру. На основі цього класу створити класи, які представляють прямокутник, квадрат, паралелограм (квадрат створити на основі прямокутника). Випадковим чином створити певну кількість чотирикутників кожного виду, щоб їх сумарна кількість дорівнювала n. Обчислити суму площ прямокутників та квадратів і суму периметрів паралелограмів.
7. Створити клас TMatrix, який представляє матрицю і містить методи для обчислення детермінанта та суми елементів матриці. На основі цього класу створити класи, які представляють квадратні матриці порядку 2, та порядку 3. За допомогою цих класів обчислити вираз

$$S = \left(\sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} \right) + |A| + |B|,$$

де $A = \left\| a_{ij} \right\|_1^3$ – матриця порядку 3, а $B = \left\| b_{ij} \right\|_1^2$ – матриця порядку 2.

8. Створити клас TVector, який представляє вектор і місить методи для обчислення довжини вектора та скалярного добутку векторів. На основі цього класу створити класи, які представляють вектори з просторів R^2 та R^3. За допомогою цих класів, обчислити значення виразу

$$S = \langle a, b \rangle + \langle c, d \rangle + |a|,$$

де $a, b \in R^3$, а $c, d \in R^2$.

9. Створити клас TVector, який представляє вектор і місить методи для визначення того, чи є інший вектор паралельним/перпендикулярним до нього та метод знаходження довжини вектора. На основі цього класу створити класи, які представляють вектори з просторів R^2 та R^3. У масиві зберегти 3 двовимірні та 4 тривимірні вектори. Знайти суму

довжин паралельних до першого по порядку двовимірних векторів та суму перпендикулярних до першого по порядку тривимірних векторів.

10. Створити клас TEquation, який представляє рівняння і містить віртуальні методи для знаходження коренів рівняння та перевірки чи є деяке значення коренем рівняння. На основі цього класу створити класи нащадки, які представляють лінійні рівняння та квадратні рівняння. Випадковим чином згенерувати дані для n лінійних рівнянь та m квадратних рівнянь. Знайти суму коренів для кожного із видів рівнянь (за умови, що вони існують).
11. Створити клас TSystemLinearEquation, який представляє систему лінійних алгебраїчних рівнянь і містить методи для введення/виведення коефіцієнтів, знаходження коренів та перевірки того, чи є деякий набір чисел розв'язком системи. На основі цього класу створити класи-нащадки, які представляють системи двох та трьох лінійних рівнянь відповідно з двома та трьома невідомими. Випадковим чином згенерувавши дані, знайти розв'язок систем лінійних алгебраїчних рівнянь обох видів.
12. Створити клас TVSeries, який представляє прогресію і містить методи для знаходження n-го члена прогресії і знаходження суми перших n членів цієї прогресі. На основі цього класу створити класи нащадки, які представляють арифметичні та геометричні прогресії. Випадковим чином згенерувати дані для n прогресій (геометрична, арифметична, геометрична, арифметична, і т.д.). Знайти суму перших m членів прогресії, n-товий член якої є найбільшим.
13. Створити клас TTriad, який представляє трійку цілих чисел і містить методи для їх збільшення/зменшення на 1. Реалізувати класи нащадки TDate ("число.місяць.рік") та TTime ("години.хвилини.секунди"). Випадковим чином згенерувати n дат та m об'єктів-часу. Визначити,

які із дат мають значення, що є допустимими, якщо їх трактувати як час. Всі інші дати зменшити на 1.

14. Створити клас TPair, який представляє пару чисел і містить методи для їх збільшення/зменшення на 1. Реалізувати класи нащадки TTime (“години.хвилини”) та TMoney (“гривні.копійки”). Згенерувати поступово випадковим чином *n* пар (час, гроші), де час – тривалість виконання роботи, а гроші – вартість однієї хвилини роботи працівників. Обчислити витрати на виконання кожної із робіт.
15. Створити клас TArray, який представляє одновимірний масив і містить методи введення/виведення, збільшення/зменшення всіх елементів на 1 та знаходження середнього арифметичного. Реалізувати класи-нащадки, що представляють одновимірні масиви з елементами цілого та дійсного типів. Випадковим чином створивши *m* масивів кожного виду, знайти масив, середнє арифметичне елементів якого є найбільшим.
16. Створити клас TSructureArray, який представляє одновимірний масив як послідовну структуру даних для збереження дійсних чисел і містить методи введення/виведення, додавання/вилучення, впорядкування та пошуку елемента. Реалізувати класи-нащадки, що представляють структуру даних впорядкований масив (можна здійснювати дихотомічний пошук) та невпорядкований масив. Випадковим чином задаючи *n* елементів, зберегти їх одночасно у впорядкованому та невпорядкованому масивах. Визначити кількість порівнянь, які необхідно для знаходження елемента як у впорядкованому, так і невпорядкованому масивах.
17. Створити клас TLine, що представляє пряму і містить методи для визначення того, чи є інша пряма паралельною/перпендикулярною до неї, чи належить вказана точка прямій. Реалізувати класи-нащадки, що представляє пряму на площині і в просторі. Випадковим чином

згенерувати дані для створення n прямих у просторі, та m прямих на площині. Визначити, чи є серед заданих прямих у просторі така, що є перпендикулярною до всіх інших прямих у просторі.

18. Створити клас TGiperPlane, що представляє гіперплощину і містить віртуальні методи для визначення того, чи належить точка гіперплощині, чи є вектор перпендикулярним до гіперплощини, знаходження проекції точки на гіперплощину. Реалізувати класи-нащадки, що представляють гіперплощину в просторі R^2 та R^3. Для кожного виду гіперплощин окремо сформувати m екземплярів, та, випадковим чином задавши точку (з відповідного простору), визначити, чи належить точка перетину деяких гіперплощин.
19. Створити клас TStructure, що представляє лінійний однозв'язний список (поле даних дійсного типу) і містить методи для виведення елементів структури на екран (без їх видалення), знаходження суми елементів та додавання/вилучення елементів. Реалізувати класи-нащадки, що представляють структури даних черга та стек. Випадковим чином задаючи дані, зберегти їх за допомогою структури даних черга, а потім використовуючи стек. Записати елементи у чергу у зворотному порядку до порядку їх введення.
20. Створити клас TFraction, який представляє дріб і містить методи для додавання/віднімання іншого дробу. На основі цього класу створити класи-нащадки TRFraction (раціональний дріб) та TDFraction (десятковий дріб), для яких передбачити метод "інвертування" (для раціональних дробів чисельник і знаменник поміняти місцями, а для десяткового – здійснити обмін цілої і дробової частин). Випадковим чином згенерувати певну кількість дробів обох видів (в одному масиві) та інвертувати дроби, які менші за середнє арифметичне усіх дробів.
21. Створити клас TEmploeer, який представляє працівника (паспортні дані, дата прийняття на роботу, дата вступу на поточну посаду) і

містить методи знаходження стажу, авансу та заробітної плати. На основі цього класу створити класи, для представлення директорів (назва відділення, кількість менеджерів та працівників, що є у підпорядкуванні), менеджерів та операторів заводу. Якщо стаж 5 (10) років, то розмір заробітної плати збільшується на 10% (25%). Визначити розміри заробітної плати працівників, дані яких згенеровано випадковим чином, якщо оклад директора 3000 грн., менеджера – 2500 грн., оператора – 1800 грн. Вивести на екран дані тих працівників, які отримують заробітну плату, вищу ніж середня заробітна плата на заводі.

Зразок виконання самостійної роботи

УМОВА. Створити клас TPrism, який представляє правильну призму і містить методи для знаходження площі поверхні та об’єму. На основі цього класу створити класи-нащадки TPrism3 та TPrism4, які представляють правильну трикутну та чотирикутну призми. З клавіатури вводиться дані для створення правильної трикутної та чотирикутної призми. На їх основі поступово випадковим чином створити m правильних призм (трикутних та чотирикутних). Для трикутних призм знайти сумарний об’єм, а для чотирикутних – суму площ основ.

ОПИС КЛАСІВ

Клас-предок TPrism

Логічна структура	Структура даних
Правильна призма **Поле** *висота* **Поле** *сторона основи* **Дія** *встановлення висоти* **Дія** *зчитування висоти* **Дія** *встановлення сторони* **Дія** *зчитування сторони* ***Властивість*** висота ***Властивість*** сторона осн. **Дія** *знаходження площі поверхні* **Дія** *знаходження об'єму* **Дія** *одержання рядкового представлення даних*	TPrism =class Private fHeight:real; fSide:real; Procedure Set_Height (value:real); Function Get_Height:Real; Procedure Set_Side (value:real); Function Get_Side:Real; Public Constructor Create(P_Height, P_Side:Real); Property Height:Real read Get_Height write Set_Height; Property Side:Real read Get_Side write Set_Side; Function Square:Real; virtual; abstract; Function Volume:Real; Function ToString:String; virtual; End;

Розглянемо структуру класів-нащадків.

Логічна структура **Призма**	**Клас-предок** TPrism	**Клас-нащадок** TPrism3	**Клас-нащадок** TPrism4	
Поле *висота*	**fHeight**			*успадковано*
Поле *сторона основи*	**fSide**			*успадковано*
Дія *встановлення висоти*	**Set_Height**			*успадковано*
Дія *одержання висоти*	**Get_Height**			*успадковано*
Дія *встановлення сторони*	**Set_Sidq**			*успадковано*
Дія *одержання сторони*	**Get_Side**			*успадковано*
	Create	**Create**	**Create**	***перевизначається***
Властивість *висота*	**Height**			*успадковано*
Властивість *сторона основи*	**Side**			*успадковано*
Дія *знаходження площі поверхні*	Square	**Square**	**Square**	***перевизначається***
Дія *знаходження об'єму*	**Volume**			*успадковано*
Дія *одержання рядкового представлення*	ToString	**ToString**	**ToString**	***перевизначається***

Клас-нащадок TPrism3

```
TPrism3 =class(TPrism)
  Constructor Create(P_Height, P_Side:Real);

      Function Square:Real; override;

      Function ToString:String; override;

End;
```

Клас-нащадок TPrism4

```
Tprism4 =class(TPrism)
  Constructor Create(P_Height, P_Side:Real);

      Function Square:Real; override;

      Function ToString:String; override;

End;
```

ПРОГРАМНА РЕАЛІЗАЦІЯ

Текст модуля з описом класу-предка TPrism

```
UNIT TPrism_Unit;
INTERFACE
 Uses SysUtils;
Type
 TPrism =class
  Private
     fHeight:real;
     fSide:real;
      Procedure Set_Height (value:real);
      Function Get_Height:Real;
      Procedure Set_Side (value:real);
      Function Get_Side:Real;
   Public
     Constructor Create(P_Height, P_Side:Real);
     Property Height:Real read Get_Height write Set_Height;
     Property Side:Real read Get_Side write Set_Side;
      Function Square:Real; virtual; abstract;
      Function Volume:Real;
      Function ToString:String; virtual;
 End;
```

```
IMPLEMENTATION
 Constructor TPrism.Create(P_Height, P_Side:Real);
  Begin
   Height:=P_Height;
   Side:=P_Side;
  End;
 procedure TPrism.Set_Height(value:real);
  Begin
   if value>=0 then fHeight:=value;
  End;
 function TPrism.Get_Height:Real;
  Begin
   Result:=fHeight;
  End;
 {----------------------}
 procedure TPrism.Set_Side(value:real);
  Begin
   if value>=0 then fSide:=value;
  End;
 function TPrism.Get_Side:Real;
  Begin
   Result:=fSide;
  End;
 {----------------------}
 Function TPrism.Volume:Real;
  Begin
   Result:=Height*Square;
  End;
 Function TPrism.ToString:String;
  Begin
   Result:='Height= '+floattostr(Height)+'  Side= '+floattostr(Height);
  End;
END.
```

Текст модуля з описом класу-нащадка TPrism3

```
UNIT TPrism3_Unit;
INTERFACE
 Uses TPrism_Unit;
 Type
  TPrism3 =class(TPrism)
   Constructor Create(P_Height, P_Side:Real);
    Function Square:Real; override;
    Function ToString:String; override;
  End;
IMPLEMENTATION
 Constructor TPrism3.Create(P_Height, P_Side:Real);
```

```
  Begin
   inherited Create(P_Height, P_Side);
  End;
 Function TPrism3.Square:Real;
  Begin
   Result:=sqr(Side)*sqrt(3)/4;
  End;
 Function TPrism3.ToString:String;
  Begin
   Result:='Triangle Prism : '+(inherited ToString);
  End;
END.
```

Текст модуля з описом класу-нащадка TPrism4

```
UNIT TPrism4_Unit;
INTERFACE
 Uses TPrism_Unit;
 Type
  TPrism4 =class(TPrism)
    Constructor Create(P_Height, P_Side:Real);
     Function Square:Real; override;
     Function ToString:String; override;
  End;
IMPLEMENTATION
 Constructor TPrism4.Create(P_Height, P_Side:Real);
  Begin
   inherited Create(P_Height, P_Side);
  End;
 Function TPrism4.Square:Real;
  Begin
   Result:=sqr(Side);
  End;
 Function TPrism4.ToString:String;
  Begin
   Result:='Quadrangular Prism : '+(inherited ToString);
  End;
END.
```

Текст програми-клієнта

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
 USES   SysUtils,
   TPrism_Unit,
   TPrism3_Unit,
   TPrism4_Unit;
```

```
 VAR A:array[1..100] of TPrism;
     i,m:integer;
     rh,rs,SumV,SumS:Real;

BEGIN
 write('m= ');readln(m);
 Randomize;
 for i:=1 to m do
  begin
   writeln('--------------------');
    write('Height = ');readln(rh);
    write('Side = ');readln(rs);
    if Random(2)=0 then
     begin
       A[i]:=TPrism3.Create(rh,rs);
       writeln('--- Triangle Prism ---');
     end
    else
     begin
       A[i]:=TPrism4.Create(rh,rs);
       writeln('--- Quadrangular Prism ---');
     end;
  end;
   writeln('--------------------');
  SumV:=0; SumS:=0;
 for i:=1 to m do
  begin
   if (A[i] is TPrism3) then
     SumV:=SumV+A[i].Volume
   else
     SumS:=SumS+2*A[i].Square;
  end;

   writeln('Sum Volume =',SumV:6:2);
   writeln('Sum Square =',SumS:6:2);
   readln;
 for i:=1 to m do
  A[i].Free;

END.
```

Результати роботи програми

```
m= 5
--------------------
Height = 2
Side = 1
```

```
--- Quadrangular Prism ---
-------------------
Height = 2
Side = 2
--- Triangle Prism ---
-------------------
Height = 3
Side = 2
--- Triangle Prism ---
-------------------
Height = 3
Side = 3
--- Quadrangular Prism ---
-------------------
Height = 4
Side = 2
--- Quadrangular Prism ---
-------------------
Sum Volume =  8.66
Sum Square = 28.00
```

4.5. Перевантаження методів

Часто, моделюючи об’єкти реальної дійсності, доводиться описувати деякі однотипні дії, які спрямовані на досягнення єдиної мети, але реалізуються різними способами. Наприклад, площа трикутника може бути знайдена різними способами, в залежності від того, які початкові дані задано. Оскільки при створенні класу не можна описати два різних члени з однаковим іменем, то для кожного з методів знаходження площі доведеться вигадувати нове ім’я, яке би характеризувало цей метод. Зрозуміло, що наявність методів з різними іменами, які виконують схожі дії тільки ускладнюють описання класу і заплутують користувача цього класу. У Delphi проблема описання однотипних методів вирішується за допомогою так званих перевантажених методів. Немає необхідності вигадувати нові імена для цих однотипних методів, достатньо, щоб вони відрізнялися кількістю або типом формальних параметрів. При описанні перевантажених методів використовується директива Overload.

```
Type  <ім'я класу>  = class
        . . . . . .
        <метод>    overload;
        . . . . . .
        <метод >    overload;
      End;
```

Приклад. Описати клас TTriangle, який містить методи для знаходження площі трикутника за формулами:

1) $S = \frac{1}{2}ah$;

2) $S = \sqrt{p(p-a)(p-b)(p-c)}$;

3) $S = \frac{1}{2}ab\sin\alpha$ (кут α задається цілим числом в градусах).

Текст модуля

```
Unit T_Unit;
INTERFACE
Uses
  SysUtils;
Type
        TTriangle=class
           Function Square(a,h:Real):Real; overload;
           Function Square(a,b,c:Real):Real; overload;
           Function Square(a,b:Real; alpha:integer):Real; overload;
        End;
IMPLEMENTATION

Function TTriangle.Square(a,h:Real):Real;
 Begin
  write('1) Square = ');
  Result:=a*h/2;                                          {S = 1/2 ah}
 End;
Function TTriangle.Square(a,b,c:Real):Real;
  var p:real;
 Begin
   write('2) Square = ');
   p:=(a+b+c)/2;
   Result:=sqrt(p*(p-a)*(p-b)*(p-c));                     { S = √(p(p-a)(p-b)(p-c)) }
 End;
Function TTriangle.Square(a,b:Real; alpha:integer):Real;
 Begin
```

```
  write('3) Square = ');
  Result:=(a*b*sin(alpha*pi/180))/2;
 End;
END.
```

$\left\{ S = \frac{1}{2} ab \sin\alpha \right\}$

Текст програми-клієнта

```
Program Project1;
  {$APPTYPE CONSOLE}
Uses
  SysUtils;
Var   tr:TTriangle;
BEGIN
  tr:=TTriangle.Create;
 writeln( tr.Square(1,2) :6:2);      //Виклик  Square(a,h:Real)
 writeln( tr.Square(2.0, 2.0, 2.0) :6:2);   //Виклик  Square(a,b,c:Real)
writeln( tr.Square(2.0, 2.0, 60) :6:2); //Виклик Square(a,b:Real;alpha:integer)
  writeln;
 writeln( tr.Square(2,  2,  2)  :6:2);     //Виклик Square(a,b:Real;alpha:integer)
   tr.Free;
 readln;
END.
```

Результат роботи програми

```
1) Square = 1.00
2) Square = 1.73
3) Square = 1.73
3) Square = 0.07
```

Зауважимо, що в наведеному прикладі не описується власний конструктор. Натомість використовується конструктор за замовчуванням.

Треба пам’ятати, що методи можуть відрізнятися не тільки кількістю формальних параметрів, а і їх типом. Тому при виклику перевантажених методів слід чітко вказувати тип фактичних параметрів. У наведеному прикладі у виклику tr.Square(2.0, 2.0, 2.0) вказано три дійсних параметри і, відповідно, викликається метод Square(a,b,c:Real). У той же час виклик

tr.Square(2,2,2), в якому останній фактичний параметр є літералом цілого типу, призведе до виклику методу Square(a,b:Real;alpha:integer).

Перевантажені методи можуть визначатися не тільки в межах одного класу, а і у класах-нащадках. Для цього у класі нащадка, дотримуючись раніше описаних вимог, описують методи з використанням директиви overload.

Приклад. Описати клас TTriangleC, який є нащадком класу TTriangle та містить метод для знаходження площі трикутника, що заданий за допомогою координат вершин.

Текст модуля з описанням TTriangle_Unit

```
UNIT TTriangle_Unit;
INTERFACE

Type
 TTriangle=class
   Function Square(a,h:Real):Real; overload;
   Function Square(a,b,c:Real):Real; overload;
   Function Square(a,b:Real; alpha:integer):Real; overload;
 End;

IMPLEMENTATION

Function TTriangle.Square(a,h:Real):Real;
 Begin
  write('1) Square = ');
  Result:=a*h/2;
 End;
Function TTriangle.Square(a,b,c:Real):Real;
 var p:real;
 Begin
  write('2) Square = ');
  p:=(a+b+c)/2;
  Result:=sqrt(p*(p-a)*(p-b)*(p-c));
 End;
Function TTriangle.Square(a,b:Real; alpha:integer):Real;
 Begin
  write('3) Square = ');
  Result:=(a*b*sin(alpha*pi/180))/2;
 End;
END.
```

Текст модуля з описанням TTriangleC_Unit

```
UNIT TTriangleC_Unit;
INTERFACE
USES TTriangle_Unit;
 Type

  TTriangleC=class(TTriangle)
     Function Square(x1,y1,x2,y2,x3,y3:Real):Real; overload;
  End;

IMPLEMENTATION

 Function LengthD(x1,y1,x2,y2:Real):Real;
  Begin
   Result:=sqrt(sqr(x2-x1)+sqr(y2-y1));
  End;
 Function TTriangleC.Square(x1,y1,x2,y2,x3,y3:Real):Real;
   Var a,b,c:Real;
  Begin
    write('4) Square = ');
   a:=LengthD(x1,y1,x2,y2);
   b:=LengthD(x1,y1,x3,y3);
   c:=LengthD(x2,y2,x3,y3);
    Result:= Square(a,b,c) ;      //Виклик методу предка Square(a,b,c:Real)
  End;
END.
```

Текст програми-клієнта

```
PROGRAM Project1;
 {$APPTYPE CONSOLE}
USES
  SysUtils,
  TTriangleC_Unit;
  Var tr:TTriangleC;
BEGIN
 tr:=TTriangleC.Create;
writeln( tr.Square(1,2) :6:2);              //Виклик Square(a,h:Real)
writeln( tr.Square(2.0,2.0,2.0) :6:2);   //Виклик Square(a,b,c:Real)
writeln( tr.Square(2.0,2.0,60) :6:2);    //Виклик Square(a,b:Real;alpha:integer)
writeln( tr.Square(1,1, 3,1, 2,2) :6:2); //Виклик Square(x1,y1,x2,y2,x3,y3:Real)
  writeln;
 tr.Free;
  readln;
END.
```

Результат роботи програми

```
1) Square =  1.00
2) Square =  1.73
3) Square =  1.73
4) Square = 2) Square  =  1.00
```

Виводиться внаслідок виклику методу предка в класі TTriangleC

Зауваження. У розглянутому прикладі об'єкт класу **TTriangleC** має доступ до чотирьох методів обчислення площі трикутника.

Перевантажені методи, як і звичайні методи, також можуть бути віртуальними. Наведемо приклад їх описання.

Приклад. Описати клас TTriangleV, який є нащадком класу TTriangle та містить віртуальний метод Square(a,b,c:Real).

Клас-предок

```
TTriangle=class
  Function Square(a,h:Real):Real; overload;
  Function Square(a,b,c:Real):Real; overload; virtual;
  Function Square(a,b:Real; alpha:integer):Real; overload;
End;
```

Клас-нащадок

```
TTriangleV=class(TTriangle)
  Function Square(a,b,c:Real):Real;  overload; override;
End;
```

Запитання для самоконтролю

1. *Що таке перевантажені методи?*
2. *У яких випадках використовують перевантажені методи?*
3. *Як описують перевантажені методи?*
4. *Чим повинні відрізнятися один від одного перевантажені методи?*
5. *Чим відрізняються перевантажені методи від інших методів у класі?*
6. *Чи можуть описуватися перевантажені методи у класі-предка і класі-нащадка?*
7. *Чи можуть бути перевантажені методи віртуальними?*
8. *Чи можна в середні перевантажених методів викликати інші перевантажені методи? Чи не буде такий виклик рекурсивним?*

Завдання для самостійної роботи

Опис та реалізацію класу розмістити в окремому модулі, а для тестування створити програму-клієнт.

1. Створити клас TTriangle, який представляє трикутник і містить перевантажені методи для знаходження радіуса описаного кола.
2. Створити клас TTriangle, який представляє трикутник і містить перевантажені методи для визначення подібності трикутників, які задані різними способами (довжинами сторін, координатами своїх вершин).
3. Створити клас TLine, який містить перевантажені методи визначення того, чи є паралельними прямі, що задаються різними способами.
4. Створити клас TLine, який містить перевантажені методи знаходження точки перетину прямих, заданих різними способами.
5. Створити клас TLine, який містить перевантажені методи визначення того, чи є перпендикулярними прямі, що задаються різними способами.
6. Створити клас TParallelogram, який представляє паралелограм і містить перевантажені методи для знаходження площі різними способами.
7. Створити клас TProgression, який представляє арифметичну прогресію і містить перевантажені методи для знаходження суми перших *n* членів.
8. Створити клас TRhombus, який представляє ромб і містить перевантажені методи для знаходження площі різними способами (задано сторони та кут між ними, задано довжини діагоналей, задано координати вершин).
9. Створити клас TTrapezoid, який представляє трапецію і містить перевантажені методи для знаходження площі різними способами (задано довжини сторін, задано висоту та середню лінію, задано координати вершин).

10. Створити клас TPolygon, який представляє правильний n-кутник і містить перевантажені методи для знаходження довжини сторони на основі радіуса вписаного і описаного кола.
11. Створити клас TPolygon, який представляє правильний n-кутник і містить перевантажені методи для знаходження площі (задаються довжини сторін, координати вершин, кількість та довжина сторони у випадку правильного n-кутника).
12. Створити клас TRectangle, який представляє прямокутник і містить перевантажені методи для знаходження площі різними способами (коли прямокутник задається за допомогою довжин сторін та координат вершин).
13. Створити клас TVector, який представляє n-вимірний вектор і містить перевантажені методи для введення координат вектора з клавіатури та файлу.
14. Створити клас TVector, який представляє n-вимірний вектор і містить перевантажені методи для обчислення скалярного добутку двох векторів.
15. Створити клас TMatrix, який представляє матрицю і містить перевантажені методи для введення елементів матриці з клавіатури та файлу.
16. Створити клас TMatrix, який представляє квадратну матрицю порядку 2 і містить перевантажені методи для визначення детермінанта матриці, що може задаватися різними способами (завантажуватися з файлу, задаватися послідовним перерахуванням елементів (перший рядок, а потім другий) або задаватися у вигляді двовимірного масиву).
17. Створити клас TSequence, який містить перевантажені методи сумування послідовності елементів, що містяться у текстовому файлі та файлі цілих чисел.

18. Створити клас TIntegral, який містить перевантажені методи наближеного обчислення інтегралу методами лівих, правих та середніх прямокутників.
19. Створити клас TFraction, який містить перевантажені методи знаходження суми двох дробів (два десяткові, десятковий і раціональний, два раціональні).
20. Створити клас TEquation, що містить перевантажені методи знаходження коренів рівняння типу $f(x)=0$ на проміжку $[a,b]$.
21. Створити клас TPlane, який містить перевантажені методи для визначення того, чи належить точка площині в R^3, що задається різними способами (точкою та вектором нормалі, канонічним рівнянням, трьома точками).

Розділ 5

ІНТЕРФЕЙСИ

5.1. Поняття інтерфейсу. Інтерфейс як засіб доступу до окремих членів класу

Більшість описуваних об'єктів є громіздкими і описуються за допомогою класів, що об'єднують у собі велику кількість полів, властивостей та методів. На практиці рідко в одній програмі використовуються всі можливі властивості і методи класів, тому така велика різноманітність властивостей і методів тільки заважає, заплутуючи користувача класу. В той же час, можна виділити окремі групи споріднених методів. Тому доцільно було б мати можливість надавати користувачеві доступ до кожної окремої групи за потребою. Це дозволить не тільки спростити доступ до класу, а й обмежити, в разі потреби, доступ клієнта до загальнодоступних методів класу. Такий підхід можна реалізувати за допомогою інтерфейсів. *Інтерфейс* – це абстрактна структура даних, яка використовується для формального опису функціональних можливостей об'єкта, що моделюється. За своєю природою інтерфейси є абстрактними класами, які можуть містити описання тільки абстрактних методів. Для них, як і для класів, визначено поняття успадкування. Наведемо загальну схему описання інтерфейсу.

```
Type
    <ім'я інтерфейсу>= Interface (<назва інтерфейсу-предка>)
        procedure <ім'я процедури 1>(<формальні параметри>);
        procedure <ім'я процедури 2>(<формальні параметри>);
          ……..
        function <ім'я функції 1>(формальні параметри):<тип функції>;
        function <ім'я функції 2>(формальні параметри):<тип функції>;

          ……..

    End;
```

Оскільки всі методи інтерфейсу є абстрактними (реалізації для них не наводиться), то, на відміну від абстрактних класів, при описанні методів службове слово abstract не використовується. Ще однією характерною особливістю інтерфейсів є те, що для них не визначено поняття областей видимості. Можна вважати, що всі методи інтерфейсу відносяться до області видимості Public.

Приклад. Наведемо приклад опису інтерфейсу для доступу до класу, що представляє трикутник.

```
Type
  ITriangle = Interface
    function Perimeter(A,B,C:Real) : Real;
    function Square(A,B,C:Real) : Real;
    function IsEqual(A1,B1,C1, A2,B2,C2:Real):Boolean;
  End;
```

Зауваження. Аналогічно до класів, імена яких прийнято починати з великої літери “T”, імена інтерфейсів прийнято починати з великої літери “I” (ITriangle, IUnknown та інші).

Як і для абстрактного класу, не можна створити екземпляр типу інтерфейсу. Клас-нащадок інтерфейсу має або містити реалізацію методів інтерфейсу-предка (в цьому випадку кажуть, що клас реалізує інтерфейс), або оголосити їх як абстрактні, з використанням директиви abstract. Раніше зазначалося, що клас може мати тільки один клас-предок, у той же час, клас може містити, крім одного класу-предка, довільну кількість інтерфейсів-предків. Тому загальний вигляд описання класу наступний:

```
Type
  <Ім'я класу> = Class (<Ім'я класу-предка>,<Ім'я інтерфейсу-предка1>,
              <Ім'я інтерфейсу-предка2>, … , <Ім'я інтерфейсу-предкаN>)

   procedure <ім'я процедури>(<формальні параметри>);
    ……
   function <ім'я функції>(<формальні параметри>):<тип функції>;
        ……                                                          } Методи інтерфейсів, що будуть реалізовані

procedure <ім'я проц.>(<форм. парам.>);          virtual; abstract;
    ……
function <ім'я функ.>(<форм. парам.>):<тип функції>; virtual; abstract;
        ……                                                          } Методи інтерфейсів, що не будуть реалізовані

                        <Опис інших членів класу>
        ……
   End;
```

Зауваження. Якщо у класі реалізується певний інтерфейс, то при його описі обов'язково вказується клас-предок, навіть якщо це клас TObject.

Приклад.

```
Type
      IInterface1= IInterface
         procedure Proc1;
      End;
      IInterface2= IInterface
         procedure Proc2;
         procedure Proc3;
      End;
      TExampleClass=class(TObject, IInterface1, IInterface2)
         procedure Proc1;                              //Буде реалізовано
         procedure Proc2;                              //Буде реалізовано
         procedure Proc3;      virtual; abstract;      //Не буде реалізовано
             ……
      End;
```

Як клас TObject є предком для усіх класів, так інтерфейс IInterface є інтерфейсом-предком для всіх інтерфейсів. Цей інтерфейс містить опис заголовків методів (_AddRef, _Release, QueryInterface), які використовуються для зв'язку з класами, що реалізують методи інтерфейсів. Оскільки ми розглядаємо тільки загальні питання, пов'язані з інтерфейсами, то детально ці методи не розглядатимемо. Але зауважимо, що клас, для якого предком є інтерфейс, має містити реалізації всіх методів цього інтерфейсу та його предків. А отже, будь-який клас, що реалізовує інтерфейс має містити також реалізації методів інтерфейсу IInterface. У системі програмування Delphi розроблено спеціальний клас TInterfacedObject, який містить реалізації методів цього інтерфейсу. Його використовують як клас-предок для всіх інших класів, у яких реалізовано інтерфейси. Отже, використовуючи цей клас, вказувати реалізації методів інтерфейсу IInterface не потрібно.

Приклад. Розглянемо клас, що представляє операції з трикутниками, які задані як за допомогою сторін, так і за допомогою координат своїх вершин. Для того, щоб була можливість окремо надавати доступ до операцій з трикутниками, які задані за допомогою сторін і окремо до операцій з трикутниками, які задані за допомогою координат вершин, цей клас опишемо як нащадок двох відповідних інтерфейсів.

Модуль інтерфейсу ITriangle

(трикутники задаються за допомогою довжин сторін)

```
UNIT ITriangle_Unit;
INTERFACE
 Type
  ITriangle = interface
     function Perimeter(A,B,C:Real) : Real;
     function Square(A,B,C:Real) : Real;
     function IsEqual(A1,B1,C1, A2,B2,C2 :Real):Boolean;
  End;
IMPLEMENTATION
END.
```

Модуль інтерфейсу ITriangleXY

(трикутники задаються за допомогою координат вершин)

```
UNIT ITriangleXY_Unit;
INTERFACE
 Type
  ITriangleXY = interface
     function PerimeterXY(xa,ya,xb,yb,xc,yc:Real) : Real;
     function SquareXY(xa,ya,xb,yb,xc,yc:Real) : Real;
     function IsEqualXY(xa1,ya1,xb1,yb1,xc1,yc1,
                              xa2,ya2,xb2,yb2,xc2,yc2 :Real):Boolean;
   End;
IMPLEMENTATION
END.
```

Модуль класу TTriangle

```
UNIT Triangle_Unit;
INTERFACE
 Uses   ITriangle_Unit,
          ITriangleXY_Unit;
 Type
  TTriangle = class(TInterfacedObject, ITriangle, ITriangleXY)
   Private
     function LengthXY(x1,y1,x2,y2:Real) : Real;
   Public
                              {Методи інтерфейсу ITriangle }
    function Perimeter(A,B,C:Real) : Real;
    function Square(A,B,C:Real) : Real;
    function IsEqual(A1,B1,C1,A2,B2,C2 :Real):Boolean;
                              {Методи інтерфейсу ITriangleXY }
        function PerimeterXY(xa,ya,xb,yb,xc,yc:Real) : Real;
        function SquareXY(xa,ya,xb,yb,xc,yc:Real) : Real;
        function IsEqualXY(xa1,ya1,xb1,yb1,xc1,yc1,
                                 xa2,ya2,xb2,yb2,xc2,yc2 :Real):Boolean;
```

```
  End;
IMPLEMENTATION
                {Функція для знаходження відстані між точками (x1,y1) та (x2,y2)}
  Function TTriangle.LengthXY(x1,y1,  x2,y2:Real) : Real;
   Begin
    Result:=sqrt(sqr(x2-x1)+sqr(y2-y1));
   End;
  {---------------------- Реалізація методів ITriangle  ------------------------}
  function TTriangle.Perimeter(A,B,C:Real) : Real;
   Begin
    Result:=A+B+C;
   End;
  function TTriangle.Square(A,B,C:Real) : Real;
    var p:Real;
   Begin
    p:=(A+B+C)/2;
    Result:= Sqrt(p*(p-A)*(p-B)*(p-C))
   End;
  function TTriangle.IsEqual(A1,B1,C1,A2,B2,C2 :Real):Boolean;
   Begin
    Result:=((A1=A2) AND (B1=B2) AND (C1=C2)) OR
            ((A1=A2) AND (B1=C2) AND (C1=B2)) OR
            ((A1=B2) AND (B1=A2) AND (C1=C2)) OR
            ((A1=B2) AND (B1=C2) AND (C1=A2)) OR
            ((A1=C2) AND (B1=B2) AND (C1=A2)) OR
            ((A1=C2) AND (B1=A2) AND (C1=B2));
   End;
{---------------------- Реалізація методів ITriangleXY ----------------------}
  function TTriangle.PerimeterXY(xa,ya,xb,yb,xc,yc:Real) : Real;
   Begin
    Result:=LengthXY(xa,ya,xb,yb)+LengthXY(xb,yb,xc,yc)+
            LengthXY(xa,ya,xc,yc);
   End;
  function TTriangle.SquareXY(xa,ya,xb,yb,xc,yc:Real) : Real;
    var a,b,c,p:Real;
   Begin
       a:=LengthXY(xb,yb,xc,yc);
       b:=LengthXY(xa,ya,xc,yc);
       c:=LengthXY(xa,ya,xb,yb);
    p:=(a+b+c)/2;
    Result:= Sqrt(p*(p-a)*(p-b)*(p-c));
   End;
  function TTriangle.IsEqualXY(xa1,ya1,xb1,yb1,xc1,yc1,
                               xa2,ya2,xb2,yb2,xc2,yc2 :Real):Boolean;
    var A1,B1,C1,A2,B2,C2 :Real;
   Begin
```

```
      A1:=LengthXY(xb1,yb1,xc1,yc1);
      B1:=LengthXY(xa1,ya1,xc1,yc1);
      C1:=LengthXY(xa1,ya1,xb1,yb1);
        A2:=LengthXY(xb2,yb2,xc2,yc2);
        B2:=LengthXY(xa2,ya2,xc2,yc2);
        C2:=LengthXY(xa2,ya2,xb2,yb2);
   Result:=((A1=A2) AND (B1=B2) AND (C1=C2)) OR
          ((A1=A2) AND (B1=C2) AND (C1=B2)) OR
          ((A1=B2) AND (B1=A2) AND (C1=C2)) OR
          ((A1=B2) AND (B1=C2) AND (C1=A2)) OR
          ((A1=C2) AND (B1=B2) AND (C1=A2)) OR
          ((A1=C2) AND (B1=A2) AND (C1=B2));
  End;
END.
```

Схематично структуру класу TTriangle (без методів інтерфейсу IInterface, які реалізовані в класі TInterfacedObject) зображено на рис. 5.1.

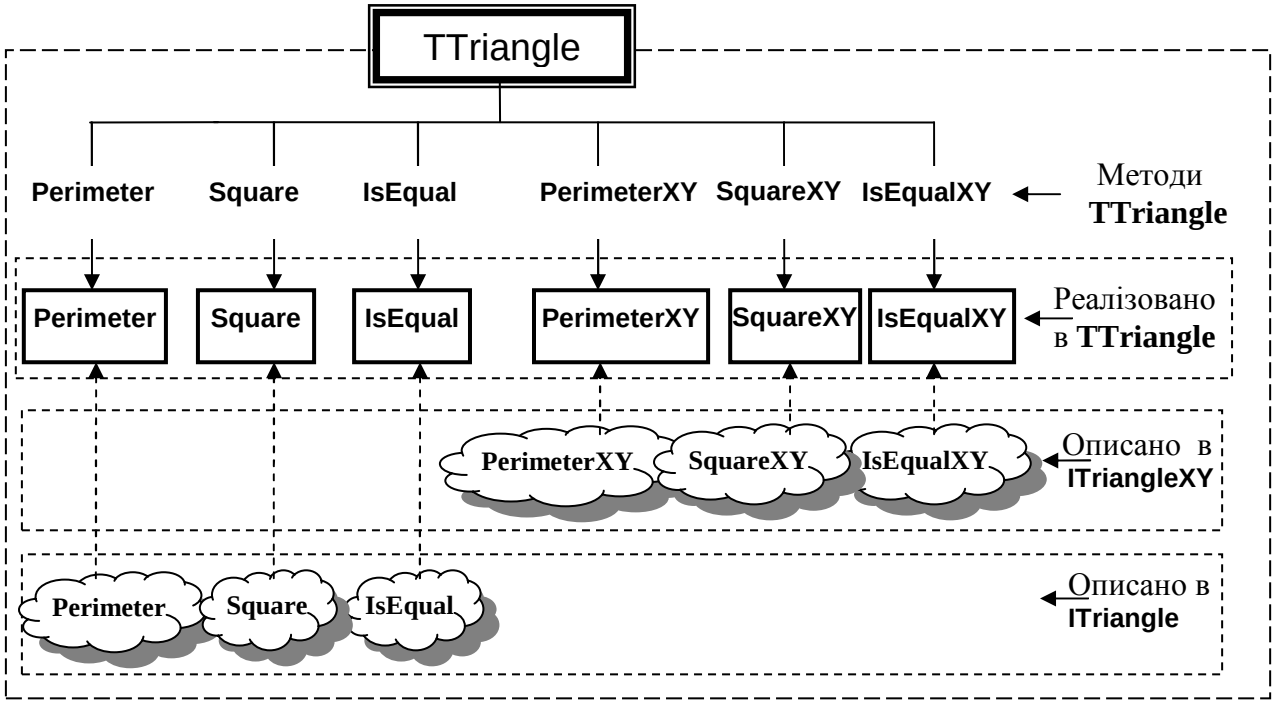


Рис. 5.1

Нехай у програмі-клієнті описано змінні типу інтерфейс ITr та ITrXY.

```
Var   ITr : ITriangle;
      ITrXY : ITriangleXY;
```

Розглянемо випадок, коли доступ до об'єкта класу TTriangle здійснюється через інтерфейс ITriangle.

```
ITr:= TTriangle.Create;      ← Створення об'єкта класу TTriangle
```

Схематично такий доступ зображено на рис. 5.2.

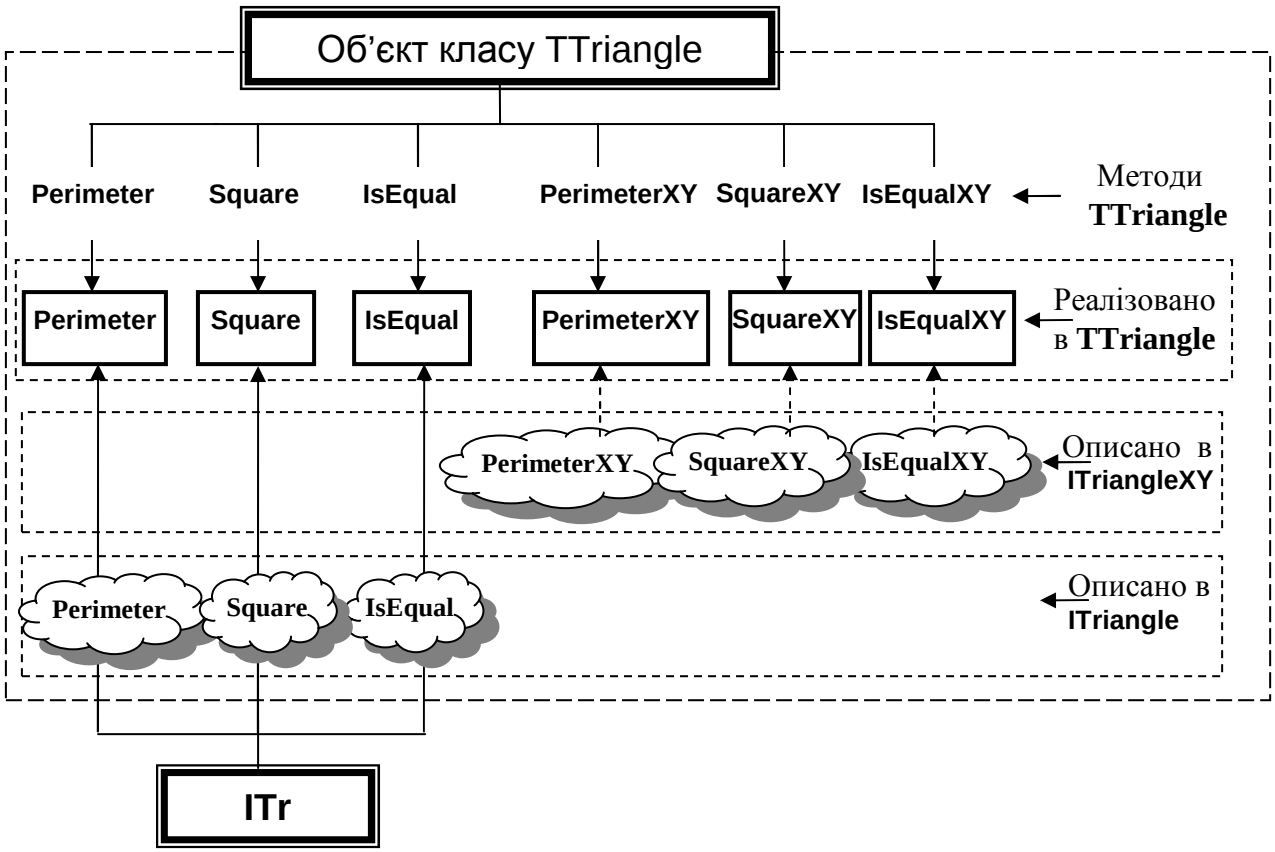


Рис. 5.2

Очевидно, змінна ITr має доступ лише до тих методів, заголовки яких описано в інтерфейсі ITriangle, а до інших методів (PerimeterXY, SquareXY, IsEqualXY) вона доступу не має.

Текст програми-клієнта

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
USES  SysUtils,
  Triangle_Unit,
  ITriangle_Unit;
Var ITr:ITriangle;
BEGIN
  ITr:=TTriangle.Create;
    writeln('Perimeter(3,2,3) = ',ITr.Perimeter(3,2,3):6:2);
    writeln('Square(3,2,3) = ',ITr.Square(3,2,3):6:2);
    writeln('IsEqual(3,2,3, 2,3,3) = ',ITr.IsEqual(3,2,3,  2,3,3));
  readln;
END.
```

Результати роботи програми

```
Perimeter (3,2,3) =   8.00
Square (3,2,3) =   2.83
IsEqual(3,2,3, 2,3,3) = TRUE
```

Розглянемо випадок, коли доступ до об'єкта класу TTriangle здійснюється через інтерфейс ITriangleXY.

```
ITrXY:=TTriangle.Create;
```

Схематично такий доступ зображено на рис. 5.3.

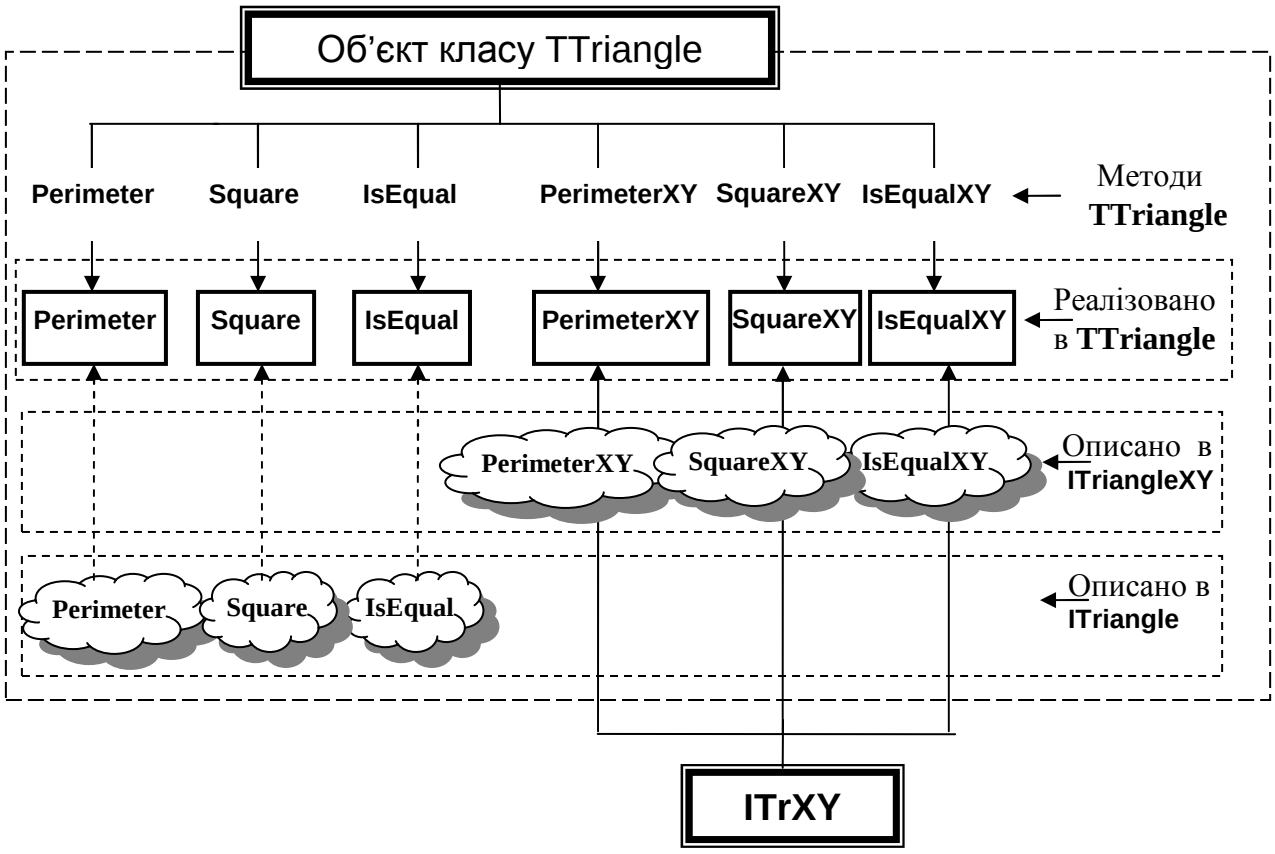


Рис. 5.3

Змінна ITrXY має доступ лише до тих методів, заголовки яких описано в інтерфейсі ITriangleXY.

Текст програми-клієнта

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
USES
  SysUtils,
  Triangle_Unit,
  ITriangleXY_Unit;
 Var
```

```
    ITrXY:ITriangleXY;
BEGIN
  ITrXY:=TTriangle.Create;
   writeln('PerimeterXY(1,1, 3,1, 2,2) = ',ITrXY.PerimeterXY(1,1, 3,1, 2,2):6:2);
   writeln('SquareXY(1,1, 3,1, 2,2) = ',ITrXY.SquareXY(1,1, 3,1, 2,2):6:2);
   writeln('IsEqualXY(1,1, 3,1, 2,2, 1,1, 4,4, 1,3) = ',
                        ITrXY.IsEqualXY(1,1, 3,1, 2,2, 1,1, 4,4, 1,3));
  readln;
END.
```

Результати роботи програми

```
PerimeterXY(1,1, 3,1, 2,2) =   4.83
SquareXY(1,1, 3,1, 2,2) =   1.00
IsEqualXY(1,1, 3,1, 2,2, 1,1, 4,4, 1,3) = FALSE
```

5.2. Багатоваріантність реалізації інтерфейсів

Інтерфейс використовується для формального опису функціональних можливостей об’єкта шляхом опису тільки заголовків методів. Оскільки інтерфейс може бути реалізовано у будь-якому класі, то і реалізація його методів у різних класах може бути різною. У цьому випадку одна і та ж змінна типу інтерфейсу може бути використана для доступу до різних реалізацій цього інтерфейсу.

Приклад. Майже у кожній програмі виникає проблема збереження та пошуку даних. Частково така проблема може бути вирішена за допомогою одновимірних масивів. При цьому зауважимо, що якщо більше буде виконуватись операцій додавання нових елементів, а операція пошуку рідко, то достатньо використати неупорядкований масив даних, якщо ж часто буде виконуватись операція пошуку елементів, то доцільно використати упорядкований масив, для пошуку елементів у якому можна використати бінарний пошук.

Розглянемо інтерфейс IArray та відповідні класи TUnOrderArray і TOrderArray, які представляють, відповідно, невпорядкований і впорядкований масиви. Для спрощення опису цих класів опишемо проміжковий клас TArray з реалізацією інтерфейсу IArray.

Модуль інтерфейсу IArray

(Описуємо заголовки методів, що визначають функціональні можливості об'єкта)

```
UNIT IArray_Unit;
INTERFACE
 Type
  IArray = Interface
     function FindElement(value:Real; var Count:byte):Byte;     {Пошук елемента}
     function AddElement(value:Real):Boolean;        {Додавання нового елемента}
     function DeleteElement(value:Real):Boolean;              {Вилучення елемента}
     Procedure Print;                              {Виведення елементів масиву на екран}
  End;
IMPLEMENTATION
END
```

Модуль класу TArray

```
UNIT TArray_Unit;
INTERFACE
USES IArray_Unit;
 Type
 TArray = class(TInterfacedObject, IArray)
  Protected
      n:Byte;
      A:array[0..100] of real;
  Public
   Constructor Create;
    function FindElement(value:Real; var Count:byte):Byte;  virtual; abstract;
    function AddElement(value:Real):Boolean; virtual; abstract;
    function DeleteElement(value:Real):Boolean;
    Procedure Print;
  END;
IMPLEMENTATION
 Constructor TArray.Create;
  Begin
   n:=0;
  End;
 Procedure TArray.Print;
   var i:Byte;
  Begin
   if n=0 then writeln('No elements.')
   else
    for i:=1 to n do
     write('  ',A[i]:5:2);
    writeln;
  End;
```

```
function TArray.DeleteElement(value:Real):Boolean;
  var d_i,j:byte;
 Begin
   d_i:=FindElement(value,j);
   if d_i<>0 then
    begin
     for j:=d_i to n-1 do
       A[j]:=A[j+1];
     Dec(n);
     Result:=true;
    end
   else
     Result:=false;
 End;
END.
```

Модуль класу TUnOrderArray_Unit

```
UNIT TUnOrderArray_Unit;
INTERFACE
 Uses TArray_Unit;
 Type
  TUnOrderArray = class(TArray)
   Public
    Constructor Create;
     function FindElement(value:Real; var Count:byte):Byte; override;
     function AddElement(value:Real):Boolean; override;
  End;
IMPLEMENTATION
Constructor TUnOrderArray.Create;
  Begin
   Inherited Create;
  end;
function TUnOrderArray.FindElement(value:Real; var Count:byte):Byte;
  var f_i:Byte;
 Begin
  f_i:=1;
  while (f_i<=n) AND (A[f_i]<>value) do Inc(f_i);
    Count :=f_i;
  if f_i>n then f_i:=0;
  Result:=f_i;
 End;
function TUnOrderArray.AddElement(value:Real):Boolean;
 Begin
  If n<100 then
   begin
```

```
    Inc(n);
    A[ n ]:=value;
   end;
 End;
END.
```

Модуль класу TOrderArray_Unit

```
UNIT TOrderArray_Unit;
INTERFACE
 Uses TArray_Unit;
 Type
  TOrderArray = class(TArray)
    Public
     Constructor Create;
       function FindElement(value:Real; var Count:byte):Byte; override;
       function AddElement(value:Real):Boolean; override;
   End;
IMPLEMENTATION
 Constructor TOrderArray.Create;
  Begin
   Inherited Create;
  end;
function TOrderArray.FindElement(value:Real; var Count:byte):Byte;
  var left,right,f_i:Byte;
 Begin
  left := 1; right := n;
  f_i := (left + right) div 2;
  Count:=0;
  while ( left< right) do
   begin
     inc(Count);
     if A[f_i] =value  then
      break
    else
     if A[f_i] > value then
        right := f_i - 1
     else
       left := f_i + 1;
     f_i := (left + right) div 2;
   end;

  if (f_i=0) or (f_i>n) or (A[f_i]<>value) then
    Result:= 0
   else
    Result:=f_i;
```

```
 End;
function TOrderArray.AddElement(value:Real):Boolean;
 var i:integer;
 Begin
  Inc(n);
  i:=n;
  while (i>1)and(A[i-1]>value) do
   begin
    A[i]:=A[i-1];
    dec(i);
   end;
    A[i]:=value;
 End;
END.
```

Текст програми-клієнта

```
PROGRAM Project1;
{$APPTYPE CONSOLE}
USES
  SysUtils,
  IArray_Unit,
  TUnOrderArray_Unit,
  TOrderArray_Unit;
 Var IAr:IArray;
       Count:byte;
BEGIN
  IAr:=TOrderArray.Create;
    IAr.AddElement(1);
    IAr.AddElement(9);
    IAr.AddElement(2);
    IAr.AddElement(7);
    IAr.AddElement(8);
    IAr.AddElement(6);
    IAr.AddElement(3);
    IAr.AddElement(-1);
  writeln('---- Order array ------');
  IAr.Print;
  writeln('Number element (2) :',IAr.FindElement(2,Count));
  writeln('Count = ',Count);
  writeln('Number element (3) :',IAr.FindElement(3,Count));
  writeln('Count = ',Count);
  IAr:=TUnOrderArray.Create;
    IAr.AddElement(1);
    IAr.AddElement(9);
    IAr.AddElement(2);
    IAr.AddElement(7);
```

```
   IAr.AddElement(8);
   IAr.AddElement(6);
   IAr.AddElement(3);
   IAr.AddElement(-1);
  writeln('---- Unorder array ------');
  IAr.Print;
  writeln('Find element (2) :',IAr.FindElement(2,Count));
  writeln('Count = ',Count);
  writeln('Find element (3) :',IAr.FindElement(3,Count));
  writeln('Count = ',Count);
   Readln;
END.
```

Результати роботи програми

```
---- Order array ------
  -1.00   1.00   2.00   3.00   6.00   7.00   8.00   9.00
Number element (2) :3
Count = 2                  ← Кількість перевірок під час пошуку елемента (2)
Number element (3) :4
Count = 1                  ← Кількість перевірок під час пошуку елемента (3)
---- Unorder array ------
   1.00   9.00   2.00   7.00   8.00   6.00   3.00  -1.00
Find element (2) :3
Count = 3                  ← Кількість перевірок під час пошуку елемента (2)
Find element (3) :7
Count = 7                  ← Кількість перевірок під час пошуку елемента (3)
```

Запитання для самоконтролю

1. *Що таке інтерфейс?*
2. *Що спільного між інтерфейсами та абстрактними класами?*
3. *Чим відрізняються інтерфейси від абстрактних класів?*
4. *Який інтерфейс є базовим для всіх інтерфейсів?*
5. *Чи можна визначати клас, який реалізує інтерфейс і для якого явно не вказано класу-предка?*
6. *Що повинен містити клас, який реалізує інтерфейс?*
7. *Чи обов'язково клас, який реалізує інтерфейс має містити реалізації всіх його методів?*
8. *Як клас* TInterfacedObject *пов'язаний з інтерфейсами?*

9. *Чи можна змінній типу інтерфейс надавати значення об'єкта типу класу, в якому реалізовано цей інтерфейс?*
10. *Чи може інтерфейс мати багато реалізацій?*

Завдання для самостійної роботи

Опис та реалізацію класів розмістити в окремому модулі, а для тестування створити програму-клієнт.

1. Створити клас, що містить методи знаходження площі, периметра та типу опуклого чотирикутника, який може задаватися довжинами сторін або координатами вершин. Створити відповідні інтерфейси для випадку задання чотирикутника довжинами сторін та координатами вершин.
2. Створити клас, що містить методи знаходження площі, периметра, радіуса вписаного і описаного кола для рівностороннього, рівнобедреного, різностороннього та прямокутного трикутників. Для різного типу трикутників передбачити відповідні інтерфейси.
3. Створити клас, що містить методи визначення паралельності та перпендикулярності прямих на площі, що задані різними способами. Для різних способів задання прямих передбачити відповідні інтерфейси.
4. Створити клас, що містить методи знаходження розв'язків системи двох лінійних рівнянь з двома невідомими та системи трьох лінійних рівнянь з трьома невідомими (коефіцієнти можуть бути задані як десяткові дробів або як раціональні дроби). Для системи двох лінійних рівнянь з двома невідомими та системи трьох лінійних рівнянь з трьома невідомими передбачити відповідні інтерфейси.
5. Створити клас, що містить методи розв'язання квадратного рівняння (знаходження розв'язків розглядати як у R, так і просторі комплексних чисел). Для різних випадків передбачити відповідні інтерфейси.

6. Створити клас, що містить методи додавання, віднімання, множення та ділення раціональних дробів та такі ж методи для роботи з комплексними числами. Для випадку раціональних дробів та випадку комплексних чисел передбачити відповідні інтерфейси.
7. Створити клас, що містить методи для знаходження відсотку від числа, збільшення, зменшення числа на певну кількість відсотків (число може бути десятковим або раціональним дробом). При створенні класу використати відповідні інтерфейси.
8. Створити клас, що містить методи для знаходження суми і різниці чисел, які можуть бути або раціональними дробами, або числами, записаними за допомогою римських цифр. При створенні класу використати відповідні інтерфейси.
9. Створити клас, що містить методи для знаходження суми і різниці цифр, які можуть бути арабськими, римськими або задані прописом. При створенні класу використати відповідні інтерфейси.
10. Створити клас, що містить методи знаходження суми, добутку та середнього арифметичного елементів масиву, який може бути статичним або динамічно створеним. При створенні класу використати відповідні інтерфейси.
11. Створити інтерфейс, що містить опис методів для роботи з векторами (знаходження довжини вектора, суми векторів, множення вектора на число, скалярного добутку) та класи, які містять реалізації методів інтерфейсу у випадку двовимірних та тривимірних векторів.
12. Створити інтерфейс, що містить опис методів для роботи з двовимірними векторами (знаходження довжини вектора, суми векторів, множення вектора на число, скалярного добутку) та класи, які містять реалізації методів інтерфейсу у випадку, коли вектор задано його координатами та координатами початку і кінця.

13. Створити інтерфейс, що містить опис методів для знаходження n-го члена та суми перших n членів прогресії. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку арифметичної та геометричної прогресій.
14. Створити інтерфейс, що містить опис методів для знаходження суми цифр числа, а визначення кількості нулів у записі числа. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку цілого та дійсного чисел
15. Створити інтерфейс, що містить опис методів додавання, вилучення та пошуку елементів, що є цілими числами типу Byte. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи елементи у множині та одновимірному масиві.
16. Створити інтерфейс, що містить опис методів додавання, вилучення та пошуку елементів, що є цілими числами. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи елементи у стеку та черзі.
17. Створити інтерфейс, що містить опис методів додавання, вилучення та пошуку дійсних чисел, які зберігаються у файлі. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи числа у текстовому та типізованому файлі дійсних чисел.
18. Створити інтерфейс, що містить опис методів роботи з матрицями (знаходження детермінанта, сліду, рангу матриці), та класи, що містять реалізації методів інтерфейсу у випадку квадратних матриць порядку 2 і 3.
19. Створити інтерфейс, що містить опис методів знаходження суми, найбільшого, найменшого елементів. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи елементи у одновимірному та двовимірному масивах.

20. Створити інтерфейс, що містить опис методів визначення відстані від точки до площини в R^3 та метод перевірки належності точки деякій площині. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку задання площини канонічним рівнянням та за допомогою трьох точок.
21. Створити інтерфейс, що містить опис методів знаходження площі поверхні, площі основи та об’єму правильної піраміди. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку трикутної та чотирикутної пірамід.

ЛІТЕРАТУРА

1. Гофман В.Э., Хомоненко А.Д. Delphi 5. – СПб.: БХВ-Петербург, 2001. – 800 с.
2. Лаптев В.В., Морозов А.В., Бокова А.В. C++. Объектно-ориентированное программирование. Задачи и упражнения. – СПб.: Питер, 2007. – 288 с.
3. Пашеку Хавьер Программирование в Borland Delphi 2006 для профессионалов.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2006. – 944 с.
4. Сухарев М.В. Основы Delphi. Профессиональный подход – СПб.: Наука и Техника, 2004. – 600 с.
5. Object Pascal. Language Guide. Borland Software Corporation. 100 Enterprise Way, Scotts Valley, CA 95066-3249, 2002.

ЗМІСТ

Відповідальний за випуск: завідувач кафедрою системного аналізу і теорії оптимізації

к. ф.-м. н., доц. Кузка О.І.

Автори: к. т. н., доц. Семйон І.В.,

к. ф.-м. н., доц. Чупов С.В.,

к. ф.-м. н., Брила А.Ю.,

к. пед. н., Апшай Н.І.

Рецензенти: д. ф.-м. н., професор, Гупал А.М.,

д. ф.-м. н., професор, Цегелик Г.Г.,

к. ф.-м. н., професор, Бунда В.В.

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Навчальний посібник