Лабораторна робота №5.

Тема : Успадкування. Програмування поліморфних методів.

Мета роботи:

Ознайомлення із програмуванням поліморфних методів при об'єктно-орієнтованому підході при використанні мови C#. Доступ до методів базових.

Теоретичні відомості

Ієрархія

Керувати великою кількістю розрізнених класів досить складно. З цією проблемою можна впоратися шляхом упорядкування та ранжирування класів, тобто поєднуючи загальні для декількох класів властивості в одному класі й використовуючи його в якості базового. Цю можливість надає механізм успадкування.

Спадкування застосовується для наступних взаємозалежних цілей:

- виключення із програми повторюваних фрагментів коду;
- спрощення модифікації програми;
- спрощення створення нових програм на основі існуючих.

Спадкування є єдиною можливістю використовувати об'єкти, вихідний код яких недоступний, але в які потрібно внести зміни. За допомогою успадкування створюється ієрархія класів (відношення ‘бути’). Крім того, можна побудувати ще одну структуру – ієрархію об'єктів (тоді, коли один об'єкт є частиною іншого – відношення ‘ частина - ціле ’).

Спадкування

Спадкування класів — ця властивість, за допомогою якого один об'єкт може набувати властивості іншого. При цьому підтримується концепція ієрархічної класифікації, що має напрямок зверху вниз. Використовуючи спадкування, об'єкт повинен визначити тільки ті якості, які роблять його унікальним у межах свого класу. Він може успадковувати загальні атрибути від своїх батьківських класів.

Клас у C# може успадковуватися довільною кількість інших класів та успадковувати тільки один клас та декілька інтерфейсів. Інтерфейс – спеціальний вид класів, які не мають реалізації.

Синтаксис успадкування:

```
[атрибути] [специфікатори] class ім'я_класу [:ім'я_базового_класу]
{ тіло_класу}
```

Клас, який успадковується, називається базовим(предком). Клас, який успадковує базовий клас, називається похідним(дочірнім). При описі класу ім'я його предка записується в заголовку класу після двокрапки. Якщо ім'я предка не зазначене, предком уважається базовий клас усієї ієрархії **`System.Object`**. Похідний клас, успадковує всі змінні, методи, властивості, оператори й індексатори, певні в базовому класі, крім того в похідний клас можуть бути додані унікальні елементи або перевизначені існуючі.

Розглянемо успадкування класів на прикладі геометричних фігур на площині. У якості базового класу створимо клас `Demopoint` (точка на площині), у якості похідного класу від `Demopoint` клас `Demoline` (відрізок на площині):

```
class Demopoint //базовий клас
  {
    public int x;
    public int y;
    public void Show()
    {
      Console.Writeline("({0}, {1})", x, y);
    }
  }
```

```
class Demoline : Demopoint //похідний клас
{
  public int xend;
  public int yend;
  public  void Show()
  {
    Console.Writeline("({0}, {1})-({2}, {3})", x, y, xend, yend);
  }
}

class Program
{
  static void Main()
  {
    Demopoint point = new Demopoint();
    point.x = 0;
    point.y = 0;
    point.Show();
    Demoline line = new Demoline();
    line.x = 2;  line.y = 2;
    line.xend = 10;  line.yend = 10;
    line.Show();
  }
}
```

Екземпляр класу **Demoline** з однаковою легкістю використовує як власні поля й методи, так і успадковані від класу **Demopoint**. При цьому, якщо метод похідного класу називається також як і метод базового класу, то викликається метод похідного. Але компілятором буде генерувати попередження про перевантаження. Щоб уникнути подібного попередження необхідно перед однойменним членом похідного класу, у цьому випадку перед методом **Show** у класі **Demoline**, поставити специфікатор **new**. Даний специфікатор приховує однойменний член базового класу й попереджень видаватися не буде.

Використання захищеного доступу

У нашому прикладі поля **x** і **y** базового класу були відкриті для доступу (**public**). Якщо забрати **public**, то поля автоматично стануть закритими для доступу (**private**), у тому числі й для доступу з похідного класу. Розв'язати проблему доступу до закритих полів базового класу з похідного можна двома способами: використовуючи властивості класу або специфікатор **protected**. При оголошенні якогось члена класу за допомогою специфікатора **protected**, він стає закритим для всіх класів, крім похідних.

```
class Demopoint
  {
     protected int x;
     protected int y;
     public void Show()
     {
       Console.Writeline("({0}, {1})",x, y);
     }
  }

  class Demoline : Demopoint
  {
    public int xend;
    public int yend;
    public new void Show()
    {
       x=2; y=2; //доступ до закритих полів базового класу
      Console.Writeline("({0}, {1})-({2}, {3})", x, y, xend, yend);
    }
```

```
    }

    class Program
    {
      static void Main()
      {
        Demopoint point = new Demopoint();
        point.Show();
        Demoline line = new Demoline();
          //line.x = 2; line.y = 2; //доступ до полів закритий
        line.xend = 10;    line.yend = 10;
        line.Show();
      }
    }
  }
```

Спадкування конструкторів

В ієрархії класів як базові, так і похідні класи можуть мати власні конструктори. При цьому конструктор базового класу створює частина об'єкта, відповідну до базового класу, а конструктор похідного класу - частина об'єкта, відповідну до похідного класу. Тому що базовий клас не має доступу до елементів похідного класу, те їх конструктори повинні бути роздільними.

У попередньому прикладі клас створюється за рахунок автоматичного виклику засобу **C#** конструктора по замовченню. Додамо конструктор тільки в похідний клас **Demoline**:

```
class Demopoint
{
  protected int x;
  protected int y;
  public void Show()
  {
    Console.Writeline("({0}, {1})",x, y);
  }
}

class Demoline : Demopoint
{
  public int xend;
  public int yend;
  new public void Show()
  {
    Console.Writeline("({0}, {1})-({2}, {3})", x, y, xend, yend);
  }

  public Demoline(int x1, int y1, int x2, int y2) //конструктор похідного класу
  {
    x = x1;    y = y1;
    xend = x2; yend = y2;
  }
}

class Program
{
  static void Main()
  {
    Demopoint point = new Demopoint(); //викликається конструктор за замовчуванням
    point.Show();
    Demoline line = new Demoline(2, 2, 10, 10); //викликається власний конструктор
    L ine.Show();
  }
}
```

У цьому випадку конструктор визначається тільки в похідному класі, тому частина об'єкта, відповідна до базового класу, створюється автоматично за допомогою конструктора за замовчуванням, а частина об'єкта, відповідна до похідного класу, створюється власним конструктором.

Якщо ж конструктори визначені у базовому, і в похідному класі, то процес створення об'єктів трохи ускладнюється, тому що повинні виконатися конструктори обох класів. У цьому випадку використовується ключове слово **base**, яке має два призначення:

1) Дозволяє **викликати** конструктор базового класу, формат розширеного оголошення:

```
конструктор_похідного_класу (список_параметрів) : base (список_аргументів)
{ тіло конструктора }
```

де за допомогою елемента списку аргументів передаються параметри конструкторові базового класу. Наприклад:

```
class Demopoint
    {
      protected int x;
      protected int y;
      public void Show()
      {
        Console.Writeline("({0}, {1})",x, y);
}
      public Demopoint (int x, int y)//конструктор базового класу
      {
        this.x=x;  this.y=y;
      }
    }

class Demoline : Demopoint
    {
      public int xend;
      public int yend;
      new public void Show()
      {
        Console.Writeline("({0}, {1})-({2}, {3})", x, y, xend, yend);
      }

      public Demoline(int x1, int y1, int x2, int y2):base(x1, y1) //конструктор
похідного класу
      {
        xend = x2; yend = y2;
      }
    }

class Program
    {
      static void Main()
      {
        Demopoint point= new Demopoint(5, 5);
        point.Show();
        Demoline line = new Demoline( 2, 2, 10, 10);
        line.Show();
      }
}
```

2) У цьому випадку ключове слово **base** діє подібно посиланню **this**, за винятком того, що посилання base завжди вказує на базовий клас для похідного класу, формат її запису виглядає в такий спосіб:

```
base.член_класу
```

Тут у якості елемента **член_класу** можна вказувати або метод, або поле екземпляра. Ця форма посилання **base** найбільш застосовна в тих випадках, коли ім'я члена в похідному класі приховує член з таким же іменем у базовому класі.

```
class Demopoint
  {
          protected int x;
          protected int y;
          public void Show()
          {
            Console.Write("({0}, {1})",x, y);
          }
          public Demopoint (int x, int y)//конструктор базового класу
          {
            this.x=x;  this.y=y;
          }
        }

        class Demoline : Demopoint
        {
          public int xend;
          public int yend;
          new public void Show()
          {
            base.Show(); //виклик члена базового класу
            Console.Writeline("-({0}, {1})", xend, yend);
          }

          public Demoline(int x1, int y1, int x2, int y2):base(x1, y1) //конструктор
      похідного класу
          {
            xend = x2; yend = y2;
          }
        }

        class Program
        {
          static void Main()
          {
            Demoline line = new Demoline( 2, 2, 10, 10);
            line.Show();
          }
        }
```

Незважаючи на те, що метод **Show** у класі **Demoline** приховує однойменний метод у класі **Demopoint**, посилання **base** дозволяє одержати доступ до методу **Show** у базовому класі. Аналогічно за допомогою посилання **base** можна одержати доступ до однойменних полів базового класу.

Змінні базового класу та похідного класу

C# є мовою з строгою типізацією, в ньому потрібно суворе дотримання сумісності типів з урахуванням стандартних перетворень типів. Із чого випливає, що змінна одного типу звичайно не може посилатися на об'єкт іншого посилального типу. За одним невеликим виключенням - посилальна змінна базового класу може посилатися на об'єкт будь-якого похідного класу. Продемонструємо це на прикладі:

```
class Demopoint
  {
    public int x;
```

```
    public int y;
    public void Show()
    {
       Console.Writeline("точка на площині: ({0}, {1})",x, y);
    }
    public Demopoint (int x, int y)
    {
      this.x=x;  this.y=y;
    }
  }

class Demoshape : Demopoint
  {
    public int z;
    new public void Show()
    {
       Console.Writeline("точка в просторі: ({0}, {1}, {2})", x, y, z);
    }
    public Demoshape(int x, int y, int z):base(x, y)
    {
      this.z=z;
    }
}

class Program
  {
    static void Main()
    {
      Demopoint point1 = new Demopoint(0,1);
      Console.Writeline("({0}, {1})",point1.x,point1.y);
      Demoshape pointshape = new Demoshape(2,3,4);
      Console.Writeline("({0}, {1}, {2})",pointshape.x, pointshape.y, pointshape.z);
      Demopoint point2=pointshape; //припустима операція
       //помилка - не відповідність типів посилань
      //pointshape=point1;
      Console.Writeline("({0}, {1})", point2.x, point2.y);
       //помилка, тому що в класі Demopoint немає поля z
      //Console.Writeline("({0}, {1}, {2})", point2.x, point2.y, point2.z);
    }
  }
```

Помилка виникне й при спробі через об'єкт **point2** звернутися до методу **Show**. Наприклад, **point2.Show()**. У цьому випадку компілятор не зможе визначити, який метод **Show** викликати - для базового або для похідного класу. Для розв'язку даної проблеми можна скористатися таким поняттям як **поліморфізм**, який ґрунтується на механізмі віртуальних методів.

Поліморфізм

Поліморфізм – одна з основних складових об’єктно-зорієнтованого програмування, що дозволяє визначати в класі, що успадковуються методи, які будуть загальними для всіх класів, що успадковують, при цьому клас, що успадковує, може визначати специфічну реалізацію деяких або всіх цих методів. Головний принцип поліморфізму: «один інтерфейс, кілька методів». Завдяки ньому, можна користуватися методами, не володіючи точними знаннями про тип об'єктів.
Основним інструментом для реалізації принципу поліморфізму є використання віртуальних методи й абстрактних класів.

Віртуальні методи

Віртуальний метод - це метод, який оголошений у базовому класі з використанням ключового слова **virtual**, і потім перевизначений у похідному класі за допомогою ключового слова **override**. При цьому якщо реалізована багаторівнева ієрархія класів, те кожний похідний клас може

мати свою власну версію віртуального методу. Цей факт особливо корисний у випадку, коли доступ до об'єкта похідного класу здійснюється через посилальну змінну базового класу. У цій ситуації C# сам вибирає яку версію віртуального методу потрібно викликати. Цей вибір проводиться по типу об'єкта, на яку посилається дане посилання. Кожний похідний клас, може мати власну версію віртуального методу. Вибір версії віртуального методу, яку потрібно викликати, здійснюється відповідно до типу об'єкта, на який посилається посилальна змінна, під час виконання програми. Інакше кажучи, саме тип об'єкта, на який указує посилання (а не змінної типу посилання), визначає викликувану версію віртуального методу. Таким чином, якщо клас містить віртуальний метод і від цього класу було успадкування іншими класами, в яких визначені свої версії методу, при змінній типу посилання базового класу на різні типи об'єктів викликаються різні версії віртуального методу.

При визначенні віртуального методу в базовому класі перед типом значення, що повертається, вказується ключове слово *virtual*, а при перевизначені віртуального методу в класі, що успадковує, використовується модифікатор override. Віртуальний метод не може бути визначений з модифікатором static або abstract. Перевизначити віртуальний метод не обов'язково. Якщо клас, що успадковує, не надає власну версію віртуального методу, то використовується метод успадкованого класу.

Перевизначення методу покладене в основу концепції динамічного вибору який здійснюється під час виконання програми, а не під час компіляції.

Синтаксис:

```
virtual тип ім'я (список _ параметрів){тіло _ методу};
```

Наприклад:

```
class Demopoint //базовий клас
  {
    protected int x;
    protected int y;
    public virtual void Show()  //віртуальний метод
    {
       Console.Writeline("точка на площині: ({0}, {1})",x, y);
    }
    public Demopoint (int x, int y)
    {
      this.x=x;  this.y=y;
    }
  }
  class Demoshape : Demopoint //похідний клас
  {
    protected int z;
    public override void Show() //перевантаження віртуального методу
    {
       Console.Writeline("точка в просторі: ({0}, {1}, {2})", x, y, z);
    }

    public Demoshape(int x, int y, int z):base(x, y) //конструктор похідного класу
    {
      this.z=z;
    }

  }
  class Demoline : Demopoint //похідний клас
  {
    protected int x2;
    protected int y2;
```

```
    public override void Show()  //перевантаження віртуального методу
    {
       Console.Writeline("відрізок на площині: ({0}, {1})-({2},{3})",x,y,
x2, y2);
    }
    public Demoline(int x1, int y1, int x2, int y2):base(x1, y1)
    {
      this.x2 = x2; this.y2 = y2;
    }
  }

  class Program
  {
    static void Main()
    {
      Demopoint point1 = new Demopoint(0,1);
      point1.Show();
      Demoshape pointshape = new Demoshape(2,3,4);
      pointshape.Show();
      Demoline line = new Demoline(0,0, 10, 10);
      line.Show();
      Console.Writeline();
       //використання посилання базового класу на об'єкти похідних класів
      Demopoint point2=pointshape;
      point2.Show();
      point2=line;
      point2.Show();
    }
  }
```

Таким чином, завдяки поліморфізму через посилальну змінну можливо звертатися до об'єктів різного типу, а також за допомогою того самого імені виконувати різні дії.

Абстрактні методи та класи

Іноді корисно створити базовий клас, що визначає тільки свого роду "порожній бланк", який успадкують усі похідні класи, причому кожний з них заповнить цей "бланк" власною інформацією. Такий клас визначає структуру методів, які похідні класи повинні реалізувати, але сам при цьому не забезпечує реалізації цих методів. Подібна ситуація може виникнути, коли базовий клас попросту не в змозі реалізувати метод. У даній ситуації розробляються **абстрактні методи** або цілі **абстрактні класи**.

Абстрактний метод створюється за допомогою модифікатора `abstract`. Він не має тіла й, отже, не реалізується базовим класом, а похідні класи повинні його обов'язково перевизначити. Абстрактний метод автоматично є віртуальним, однак використовувати специфікатор **`virtual`** не потрібно. Більше того, якщо ви спробуєте використовувати два специфікатори одночасно, **`abstract`** і **`virtual`**, те компілятор видасть повідомлення про помилку.

Якщо похідний клас успадковує абстрактний, то він повинен повністю перевизначити всі абстрактні методи базового класу або також бути оголошений як абстрактний. Таким чином, специфікатор **`abstract`** успадковується доти, поки в похідному класі не будуть реалізовані всі абстрактні методи.

Розглянемо приклад використання абстрактних методів і класів.

```
abstract class Demo //абстрактний клас
  {
    abstract public void Show();//абстрактний метод
    abstract public double Dovgina();//абстрактний метод
  }
```

```
class Demopoint:Demo //похідний клас від абстрактного
{
  protected int x;
  protected int y;
  public Demopoint (int x, int y)
  {
    this.x=x;  this.y=y;
  }
  public override void Show() //перевизначення абстрактного методу
  {
     Console.Writeline("точка на площині: ({0}, {1})",x, y);
  }
  public override double Dovgina()  //перевизначення абстрактного методу
  {
    return Math.Sqrt(x*x+y*y);
  }
}

class Demoshape : Demopoint //похідний клас
{
  protected int z;
  public Demoshape(int x, int y, int z):base(x, y)
  {
    this.z=z;
  }
  public override void Show()  //перевизначення абстрактного методу
  {
     Console.Writeline("точка в просторі: ({0}, {1}, {2})", x, y, z);
  }
  public override double Dovgina()  //перевизначення абстрактного методу
  {
    return Math.Sqrt(x*x+y*y+z*z);
  }
}

class Demoline : Demopoint //похідний клас
{
  protected int x2;
  protected int y2;
  public Demoline(int x1, int y1, int x2, int y2):base(x1, y1)
  {
    this.x2 = x2; this.y2 = y2;
  }
  public override void Show()  //перевизначення абстрактного методу
  {
     Console.Writeline("відрізок на площині: ({0}, {1})-({2},{3})",x,y,
x2, y2);
  }
  public override double Dovgina()  //перевизначення абстрактного методу
  {
    return Math.Sqrt((x-x2)*(x-x2)+(y-y2)*(y-y2));
  }
}

class Program
{
  static void Main()
  {
     Demo [] Ob=new Demo[5]; //масив посилань
     //заповнення масиву посиланнями на об'єкти похідних класів
```

```
        Ob[0]=new Demopoint(1,1);
        Ob[1]=new Demoshape(1,1,1);
        Ob[2]=new Demoline(0,3,4,0);
        Ob[3]=new Demoline(2,1,2,10);
        Ob[4]=new Demopoint(0,100);
        foreach (Demo a in Ob) //перегляд масиву
        {
          a.Show();
          Console.Writeline("Dovgina: {0:f2}\n", a.Dovgina());
        }
    }
}
```

Заборона успадкування

У C# є ключове слово sealed, що дозволяє описати клас, від якого заборонено успадкування. Наприклад:

```
sealed class Demo {   … }
class newdemo: Demo {   …  }  // помилка
```

Завдання до лабораторної роботи:

Порядок виконання роботи:

1) Клонувати репозиторій **https://classroom.github.com/a/07oeqHdc**
2) Визначити ієрархію класів та класи. Повну структуру класів і їх взаємозв'язок продумати самостійно. У класах реалізувати абстрактні, віртуальні та не віртуальні методи, властивості, деструктори, індексатори та операції згідно з варіантом завдання. Для абстрактного класу визначити які методи повинні бути абстрактними.
3) Розробити програму мовою C# тестування всіх можливосте створених класів із виведення відповідної інформації.
4) Підготувати звіт в електронному виді надіслати мудл(https://moodle.chnu.edu.ua/course/view.php?id=3371) .

Завдання 1. Варіанти задач. Побудувати ієрархію класів відповідно до варіанта завдання. Згідно завдання вибрати базовий клас та похідні. В класах задати поля, які характерні для кожного класу. Для всіх класів розробити метод Show(), який виводить дані про об’єкт класу.

1.1. Студент, викладач, персона, завідувач кафедри.
1.2. Службовець, персона, робітник, інженер.
1.3. Робітник, кадри, інженер, адміністрація.
1.4. Деталь, механізм, виріб, вузол.
1.5. Організація, страхова компанія, нафтогазова компанія, завод.
1.6. Журнал, книга, друковане видання, підручник.
1.7. Тест, іспит, випускний іспит, випробування.
1.8. Місце, область, місто, мегаполіс.
1.9. Іграшка, продукт, товар, молочний продукт.
1.10. Квитанція, накладна, документ, рахунок.

1.11. Автомобіль, поїзд, транспортний засіб, експрес.
1.12. Двигун, двигун внутрішнього згоряння, дизель, реактивний двигун.
1.13. Республіка, монархія, королівство, держава.
1.14. Савець, парнокопитне, птах, тварина.
1.15. Корабель, пароплав, вітрильник, корвет.

Завдання 2. Варіанти задач. До побудованої ієрархії класів з завдання 1 додати конструктори та деструктори в який виводити повідомлення в консоль. Продемонструвати порядок виклику конструкторів та деструкторів. У класах передбачити не менше 3 (трьох) конструкторів.

2.1. Студент, викладач, персона, завідувач кафедри.
2.2. Службовець, персона, робітник, інженер.
2.3. Робітник, кадри, інженер, адміністрація.
2.4. Деталь, механізм, виріб, вузол.
2.5. Організація, страхова компанія, нафтогазова компанія, завод.
2.6. Журнал, книга, друковане видання, підручник.
2.7. Тест, іспит, випускний іспит, випробування.
2.8. Місце, область, місто, мегаполіс.
2.9. Іграшка, продукт, товар, молочний продукт.
2.10. Квитанція, накладна, документ, рахунок.
2.11. Автомобіль, поїзд, транспортний засіб, експрес.
2.12. Двигун, двигун внутрішнього згоряння, дизель, реактивний двигун.
2.13. Республіка, монархія, королівство, держава.
2.14. Савець, парнокопитне, птах, тварина.
2.15. Корабель, пароплав, вітрильник, корвет.

Завдання 3. Варіанти задач. Побудувати ієрархію класів відповідно до варіанта завдання. Згідно завдання вибрати базовий клас та похідні. В класах задати задати вмістиме для кожного класу.

3.1. Створити абстрактний клас **`Figure`** з методами обчислення площі й периметра, а також методом, що виводять інформацію про фігуру на екран. Створити похідні класи: **`Rectangle`** (прямокутник), **`Circle`** (коло), **`Triangle`** (трикутник) зі своїми методами обчислення площі й периметра. Створити масив **`n`** фігур і вивести повну інформацію про фігури на екран.
3.2. Створити абстрактний клас **`Function`** з методом обчислення значення функції **`y=f(x)`** у заданій точці. Створити похідні класи: **`Line (y=ax+b)`**, **`Quadratic (y=ax`**^{2}**`+bx+c)`**, **`Hyperbola (y=k/x)`** зі своїми методами обчислення значення в заданій точці. Створити масив n функцій і вивести повну інформацію про значення даних функцій у точці `x`.
3.3. Створити абстрактний клас **`Видання`** з методами, що дозволяють вивести на екран інформацію про видання, а також визначити чи є дане видання шуканим. Створити похідні класи: **`Книга`** (назва, прізвище автора, рік

видання, видавництво), **Стаття** (назва, прізвище автора, назва журналу, його номер і рік видання), **Електронний ресурс** (назва, прізвище автора, посилання, анотація) зі своїми методами висновку інформації на екран. Створити каталог (масив) з **n** видань, вивести повну інформацію з каталогу, а також організувати пошук видань на прізвище автора.

3.4. Створити абстрактний клас **Trans** з методами, що дозволяють вивести на екран інформацію про транспортний засіб, а також визначити вантажопідйомність транспортного засобу. Створити похідні класи: **Легкова_машина** (марка, номер, швидкість, вантажопідйомність), **Мотоцикл** (марка, номер, швидкість, вантажопідйомність, наявність коляски, при цьому якщо коляска відсутня, то вантажопідйомність рівна 0), Вантажівка (марка, номер, швидкість, вантажопідйомність, наявність причепа, при цьому якщо є причіп, то вантажопідйомність збільшується у два рази) зі своїми методами висновку інформації на екран, і визначення вантажопідйомності. Створити базу (масив) з **n** машин, вивести повну інформацію з бази на екран, а також організувати пошук машин, що задовольняють вимогам вантажопідйомності.

3.5. Створити абстрактний клас **Persona** з методами, що дозволяють вивести на екран інформацію про персону, а також визначити її вік (на момент поточної дати). Створити похідні класи: **Абітурієнт** (прізвище, дата народження, факультет), **Студент** (прізвище, дата народження, факультет, курс), **Викладати** (прізвище, дата народження, факультет, посада, стаж), зі своїми методами висновку інформації на екран, і визначення віку. Створити базу (масив) з **n** персон, вивести повну інформацію з бази на екран, а також організувати пошук персон, чий вік попадає в заданий діапазон.

3.6. Створити абстрактний клас **Товар** з методами, що дозволяють вивести на екран інформацію про товар, а також визначити, чи відповідає вона строку придатності на поточну дату. Створити похідні класи: **Продукт** (назва, ціна, дата виробництва, строк придатності), **Партія** (назва, ціна, кількість шт, дата виробництва, строк придатності), **Комплект** (назви, ціна, перелік продуктів) зі своїми методами висновку інформації на екран, і визначення відповідності строку придатності. Створити базу (масив) з **n** товарів, вивести повну інформацію з бази на екран, а також організувати пошук простроченого товару (на момент поточної дати).

3.7. Створити абстрактний клас **Товар** з методами, що дозволяють вивести на екран інформацію про товар, а також визначити, чи відповідає вона шуканому типу. Створити похідні класи: **Іграшка** (назва, ціна, виробник, матеріал, вік, на який розрахована), **Книга** (назва, автор, ціна, видавництво, вік, на який розрахована), **Спорт-інвертар** (назва, ціна, виробник, вік, на який розрахована), зі своїми методами висновку інформації на екран, і

визначення відповідності шуканому типу. Створити базу (масив) з **n** товарів, вивести повну інформацію з бази на екран, а також організувати пошук товарів певного типу.

3.8. Створити абстрактний клас **Телефонний_довідник** з методами, що дозволяють вивести на екран інформацію про записи в телефонному довіднику, а також визначити відповідність запису критерію пошуку. Створити похідні класи: **Персона** (прізвище, адреса, номер телефону), **Організація** (назва, адреса, телефон, факс, контактна особа), **Друг** (прізвище, адреса, номер телефону, дата народження) зі своїми методами висновку інформації на екран, і визначення відповідності шуканому типу. Створити базу (масив) з **n** товарів, вивести повну інформацію з бази на екран, а також організувати пошук у базі на прізвище.

3.9. Створити абстрактний клас **Клієнт** із методами, що дозволяють вивести на екран інформацію про клієнтів банку, а також визначити відповідність клієнта критерію пошуку. Створити похідні класи: **Вкладник** (прізвище, дата відкриття внеску, розмір внеску, відсоток по внескові), **Кредитор** (прізвище, дата видачі кредиту, розмір кредиту, відсоток по кредиту, остача боргу), **Організація** (назва, дата відкриття рахунку, номер рахунку, сума на рахунку) зі своїми методами висновку інформації на екран, і визначення відповідності даті (відкриття внеску, видачі кредиту, відкриття рахунку). Створити базу (масив) з **n** клієнтів, вивести повну інформацію з бази на екран, а також організувати пошук клієнтів, що почали співробітничати з банком у задану дату.

3.10. Створити абстрактний клас **Програмне_забезпечення** з методами, що дозволяють вивести на екран інформацію про програмне забезпечення, а також визначити відповідність можливості використання (на момент поточної дати). Створити похідні класи: **Вільне** (назва, виробник), **Умовно-безкоштовне** (назва, виробник, дата установки, строк безкоштовного використання), **Комерційне** (назва, виробник, ціна, дата установки, строк використання) зі своїми методами висновку інформації на екран, і визначення можливості використання на поточну дату. Створити базу (масив) з **n** видів програмного забезпечення, вивести повну інформацію з бази на екран, а також організувати пошук програмного забезпечення, яке припустимо використовувати на поточну дату.

Контрольні питання

1) Що розуміється під терміном «поліморфізм»?
2) У чому полягає основний принцип поліморфізму?
3) У чому полягає значення основного принципу поліморфізму?
4) Які механізми використовуються в мові C# для реалізації концепції поліморфізму?
5) Що розуміється під терміном «віртуальний метод»?
6) Яке ключове слово мови C# використовується для визначення віртуального методу?
7) У чому полягає особливість віртуальних методів у похідних (дочірніх) класах?
8) У який момент трансляції програми здійснюється вибір версії віртуального методу?
9) Які умови визначають вибір версії віртуального методу?
10) Яке ключове слово (модифікатор) мови C# використовується для визначення віртуального методу в базовому (батьківському) класі?
11) Яке ключове слово (модифікатор) мови C# використовується для визначення віртуального методу в похідному (дочірньому) класі?
12) Які модифікатори неприпустимі для визначення віртуальних методів?
13) Що означає термін «перевизначений метод»?
14) У який момент трансляції програми здійснюється вибір викликуваного перевизначеного методу?
15) Приведіть синтаксис віртуального методу в загальному виді. Проілюструйте його фрагментом програми мовою C#.
16) Що розуміється під терміном «абстрактний клас»?
17) У чому полягають особливості абстрактних класів?
18) Який модифікатор мови C# використовується при оголошенні абстрактних методів?
19) Чи є абстрактні методи віртуальними?
20) Чи використовується модифікатор virtual мови C# при оголошенні абстрактних методів?
21) Чи можливо створення ієрархії класів за допомогою абстрактного класу?
22) Чи можливо створення об'єктів абстрактного класу?
23) Приведіть синтаксис абстрактного класу в загальному виді.
24) Для чого може створюватися клас, від якого не можна успадковувати?
25) Чи можна одночасно використовувати sealed і abstract при описі класу.