Тема 9

Наследяване в програмен език C#.

1. Механизъм "наследяване на класовете"

В обектно-ориентираното програмиране се използва един мощен механизъм за построяване на отношения между класовете, известен като **наследяване** (***inheritance***). Този механизъм позволява даден клас да наследи друг. Класът, който наследява се нарича **наследник** или **производен клас**. Класът, от който се наследява се нарича **предшественик** или **базов (основен) клас**.

Процесът на наследяване по отношение на базовите и производните класове се изразява в следното:

- производният клас наследява структурата на основния т.е. неговите полета методи и свойства;
- производният получава достъп до наследените от основния клас членове;
- освен наследените, производният клас може да има свои собствени членове (полета, свойства и методи);
- достъпът на методите на производния клас до наследените членове от основния се регламентират от модификаторите за достъп ***public***, ***private*** и ***prortected, internal***;
- производният клас, от своя страна може да е базов за други класове;
- един клас може да е основен за няколко производни. По този начин се получава йерархична структура, породена от механизма на наследяване;
- в C#, за разлика от други програмни езици (например, C++), се допуска един производен клас да наследи само един основен клас т.е. език C# не поддържа **множествено** наследяване на класове.
- в език C# всеки клас може да бъде наследен, стига това да не е изрично забранено с ключова дума ***sealed***.

Тъй като наследяването се разглежда като еволюционен процес, производният клас допълнително може да дефинира свои собствени полета, нови методи, както и да предефинира методи, принадлежащи на базовия клас. Чрез наследяването производният клас не копира конкретни данни от основния, а наследява и доразвива неговия модел.

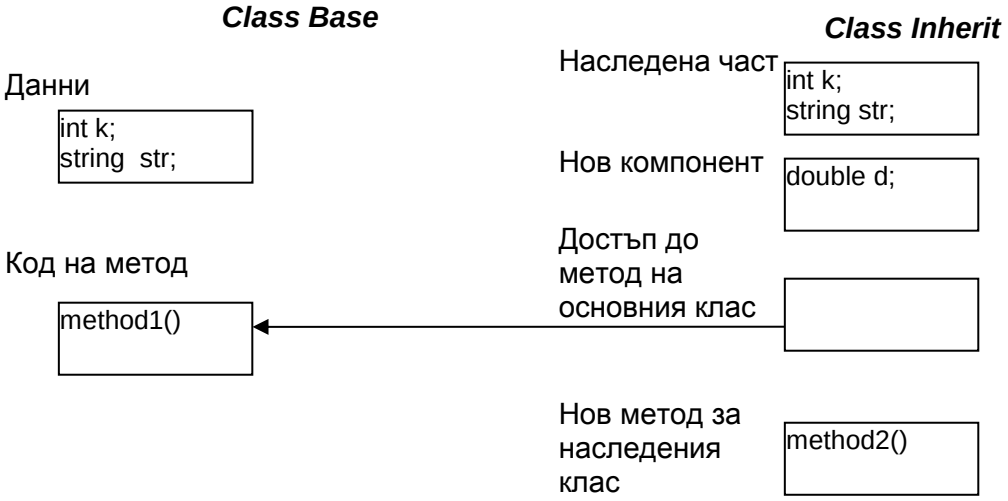
Например, нека клас ***Base*** е основен, клас ***Inherit*** е производен от него. Декларациите на класовете са следните:

```
// декларация на базов клас
class Base
{
      private int k;
```

```
      private string str;
      public int method1();
}
// декларация на производния клас

class Inherit : public Base
{
      private double d;
      public void method2();
}
```

Структурата на данните в паметта може да се представи по начина, показан на фиг. 8.1. Клас ***Base*** съдържа член-данни ***k***, ***st*** и метод ***method1***, които в последствие се наследяват от клас ***Inherit***. От своя страна производният клас има свои собствени членове (***d*** и ***method2***).



*Фиг. 8.1. Структура на класове **Base** и **Inherit***

Важно е да се отбележи, че полето **k** за обект от клас ***Base*** заема памет, напълно различна от паметта, заемана от полето **k** за обект от клас ***Inherit***, както е показано на фиг. 8.1. На практика, стойностите на двете полета нямат нищо общо помежду си.

Производният клас може да използва методи на основния, като едновременно с това може да съдържа и собствени методи. В случая, в производния клас не се създават копия на методите на основния, а се дава достъп на класа до тези методи.

Основните предимства на използването на производни класове са:

- Дефинирането на производни класове е еквивалентно на конструиране на йерархия от класове, която от своя страна е еквивалентна на йерархията от дадена предметна област или на обекти от реалния свят.

- Ако множество класове имат общи данни и методи, тези общи части могат да се обособят като базов клас, а всеки един от множеството класове да бъде дефиниран като производен на базовия. По този начин се избягва многократно описание на едни и същи фрагменти. В някои случаи, това води до икономия на памет.

- При създаването на производни класове е достатъчно да се разполага само с обектните модули на базовите класове, а не с техния програмен текст. Това позволява предварително да бъдат създавани библиотеки от класове, които в последствие да се използват като базови при създаване на различни производни класове за конкретни нужди.

- Производните класове, съчетани с използване на ***виртуални методи*** са средство за реализиране на ***полиморфизъм***. Полиморфизмът дава възможност за еднотипна обработка на информационни структури като списъци, масиви, дървета, графи и др., чиито елементи са обекти на други класове.

2. Декларация на производен клас в C#

Декларацията на производен клас в C# е следната:

```
class ИмеПроизводенКлас : ИмеБазовКлас
{
      // тяло на производен клас
}
```

За да се декларира производен клас, след неговото име се поставя символ двоеточие (**:**), след което следва името на базовия клас. В тялото на производния клас се дефинират само собствените членове на класа – полета, методи и свойства, а производния клас получава от базовия всички негови членове, независимо, че те не са обявени в тялото на производния клас.

Например, нека е деклариран клас Point, който ще бъде използван като базов клас:

```
class Point
    {
        private int x, y;

        public void Input()
        {
            Console.Write("X=");
            string temp = Console.ReadLine();
            x = Int32.Parse(temp);
            Console.Write("Y=");
            temp = Console.ReadLine();
            y = Int32.Parse(temp);
        }
        public void Output()
        {
            Console.WriteLine("( {0}, {1} )", x, y);
        }

    }
```

Нека бъде деклариран производен клас Circle:

```
class Circle : Point
    {
        private int radius;


        public int Radius
        {
            get { return radius; }
            set { radius= value; }
        }
        public double Area
        {
            get
            {
                return radius * radius * Math.PI;
            }
        }
        public void ShowCircle()
        {
            Console.WriteLine("Figure: circle\n Center: ");
            Output();
            Console.WriteLine("Radius: " + radius);
        }
       }
```

В случая, клас Circle наследява клас Point. Когато се дефинира обект от производен клас, той съдържа полета от базовия и полета, които са собствени членове. Така един обект от клас Circle ще съдържа три полета: x, y, radius. Първите две са от наследената част, а третото е собствено за класа. Следващият програмен код демонстрира създаването на обект от производния клас и работата с методите и свойствата на базовия и производния клас:

```
Circle firstCircle = new Circle();
Console.WriteLine("The circle object is created");
Console.WriteLine("Enter a center:");
firstCircle.Input();
Console.WriteLine("Enter a radius:");
firstCircle.Radius = Int32.Parse(Console.ReadLine());
firstCircle.ShowCircle();
Console.WriteLine("The circle area is " + firstCircle.Area);
```

Както се вижда от примера, обект `firstCircle` може да извиква както собствените методи на класа, така и методите на базовия клас.

Трябва да се отбележи, че всеки клас, който бива дефиниран, наследява всички елементи и реализации на клас System.Object. Този клас съответства на тип object и е дефиниран в работното пространство System. На този клас принадлежи и метода ToString, който се използва за конвертиране на обект в низ.

3. Достъп до наследени членове на производния клас. Модификатори на достъп

Производният клас наследява всички членове на базовия с изключение на конструкторите и деструкторите. Членовете на базовия клас, които са със статут ***public, protected, internal*** се наследяват като такива и в производния. Членовете на базовия клас, които са ***private*** са недостъпни на методите на производния и

могат да бъдат достигнати единствено от наследените методи на базовия клас. Например, в базовия клас Point полетата x, y са с модификатор на достъп ***private***. Това означава, че метод на производния клас няма достъп до тези полета, дори и да се касае за обект от производния клас.

Нека програмният код на метод бъде модифициран по следния начин:

```
public void ShowCircle()
 {
      Console.WriteLine("Figure: circle\n Center: {0} , {1}", x, y );
      Console.WriteLine("Radius: " + radius);
  }
```

В първия програмен ред на метода ще бъде генерирана синтактична грешка поради невъзможност за достъп до полета x, y. Достъпът до полетата е възможен само индиректно, като се използва метод на базовия клас. В случая, това е метод Output(), който извежда стойностите на координатите на точката. Този метод може да бъде извикан с програмен ред:

```
Output();
```

или

```
this.Output();
```

или

```
base.Output();
```

Ключова дума `base` указва базовия клас.

Ако даден член на базовия клас е с модификатор на достъп ***protected***, то той е достъпен както от собствените методи на класа, така и от методите на наследените класове. Съответно, за да може да бъде изпълнен програмният ред

```
Console.WriteLine("Figure: circle\n Center: {0} , {1}", x, y );
```

в метод `ShowCircle`, необходимо е модификаторът на достъп на полета x, y в базовия клас Point да бъде сменен от ***private*** на ***protected***:

```
protected int x, y;
```

Така методите на производния клас получават директен достъп до наследените членове, без да се нарушава принципа на капсулиране на данните.

Както и при наследяването в други програмни езици (C++ например), в C# членове на базовия клас, който са обявени като ***protected*** са достъпни от методите на производния клас (както и тези на базовия) и недостъпни за останалите методи. Членовете от базовия клас, които са с модификатор ***protected*** се наследяват като такива и в производния клас. Това означава, че те ще се наследяват като такива във всички преки или не преки наследници на базовия. Методите на наследените класове имат достъп само до ***protected*** членовете, които те самите са наследили. Те нямат достъп до тези членове на базовия клас, ако използват обръщение към базовия клас.

При наследяване на класове в език C# има някои особености като това, че наследяването в C# е винаги със статут ***public***. В език C# не се позволява наследяване със статут ***private*** или ***protected,*** за разлика от C++. Това е и причината, поради която липсва модификато на достъп пред името на базовия клас в декларацията на производния. Този модификатор е ***public*** по подразбиране.

Друга особеност е: Не е възможно клас, който е определен като ***private*** да бъде наследен. Пример:

```
class Example
{
      private class NestedBase
      {
      }
      public class NestedDerived: NestedBase    // error
      {
      }

      ...
}
```

4. Конструктори на базови и производни класове

Конструкторите на класовете отговарят за инициализацията на полетата (данните в класа). Производният клас автоматично получава всички елементи на базовия. Всеки един клас има поне един конструктор. Ако няма такъв компилаторът генерира служебен конструктор по подразбиране.

Инициализирането на производни класове по същество не се различава от това на базовите (основните) класове. То се извършва чрез конструктора на производния клас, но поради взаимоотношенията **основен-производен клас** възникват някои особености. Основният проблем в тези случаи е как да се инициализира наследената част на производния клас. Ако това се изпълнява от конструктора на производния клас, то би трябвало той да има достъп до локалните данни на основния клас. Това противоречи на принципа на скриване на информацията. Затова, функциите по инициализиране са разделени между двата конструктора – на основния и на производния клас. Проблемът за инициализацията при наследените класове се решава по следния начин: **Конструкторът на производния клас инициализира само новите (собствените) членове на производния клас, а наследените членове се инициализират от конструктора на базовия клас.**

Изпълнението на конструкторите на производните класове протича по следните правила: Извикването на конструктор на един производен клас води до извикване на конструктора на базовия клас и след завършване на неговото изпълнение, се изпълнява конструкторът на производния клас.

Пример: Нека клас `DerivedClass` наследява клас `BaseClass`. Двата класа имат по един конструктор без параметри:

```
class BaseClass
    {
        public BaseClass()
        {
            Console.WriteLine("Base class.");
        }
    }

class DerivedClass:BaseClass
    {
        public DerivedClass()
        {
            Console.WriteLine("Derived class.");
```

```
    }
}
```

При дефиниране на обект от производния клас, конструкторът на производния клас ще извика конструктора на базовия клас, след което се изпълнява тялото на конструктора на производния клас.

```
DerivedClass examp = new DerivedClass();
```

Резултатът от изпълнението на програмния ред е последователно извеждане на екрана на:

```
Base class.
Derived class.
```

За да се гарантира извикването на конструктора на базовия клас, в програмния код може да се добави явно обръщението към този конструктор чрез ключова дума ***base***. В случая, програмният код на конструктора на клас `DerivedClass` е:

```
public DerivedClass():base()
{
            Console.WriteLine("Derived class.");
}
```

В посочения пример, независимо дали програмистът се е погрижил за явното извикване на конструктора на базовия клас или не, той винаги се изпълнява. Причината е, че компилаторът се грижи да добави съответния програмен код като прави обръщение към конструктор по подразбиране. Това означава, че базовият клас трябва да разполага с конструктор без параметри. Ако няма такъв или е необходимо да се използва конструктор без параметри, програмистът трябва да се погрижи за явното и извикване като използва ключова дума ***base***.

В програмен език C# извикването на конструктор на базов клас е чрез следния синтаксис:

public ИмеПроизводенКлас(списък формални параметри) : base(списък фактически параметри)
{
// тяло на конструктора на производния клас
}

Особеното в случая е, че в заглавния ред параметри, които са формални за конструктора на производния клас, се явяват фактически за конструктора на базовия.

В следващия пример илюстрира използването на конструктори с и без параметри в базовия и производния класове:

```
class Point
    {
        private int x, y;
        public Point()
        {
            x = y = 0;
        }
        public Point(int x, int y)
        {
```

```
            this.x = x;
            this.y = y;
        }

        public void Input()
        {
            Console.Write("X=");
            string temp = Console.ReadLine();
            x = Int32.Parse(temp);
            Console.Write("Y=");
            temp = Console.ReadLine();
            y = Int32.Parse(temp);
        }
        public void Output()
        {
            Console.WriteLine("( {0}, {1} )", x, y);
        }

    }
    class Circle : Point
    {
        private int radius;
        public Circle() : base()
        {
            radius = 0;
        }
        public Circle(int x, int y, int radius) : base(x, y)
        {
            this.radius = radius;
        }

        public int Radius
        {
            get { return radius; }
            set { radius= value; }
        }
        public double Area
        {
            get
            {
                return radius * radius * Math.PI;
            }
        }
        public void ShowCircle()
        {
            Console.WriteLine("Figure: circle\n Center: " );
            base.Output();
            Console.WriteLine("Radius: " + radius);
        }
    }
```

В примера във всеки един от класовете има по два конструктора: единият е с параметри, а другия – без параметри. Следващите програмни редове демонстрират създаването на обекти от производния клас като се използват и двата конструктора:

```
Circle firstCircle = new Circle();
firstCircle.ShowCircle();

Circle secondCircle = new Circle(100, 200, 4);
secondCircle.ShowCircle();
```

Полетата за обект `firstCircle` са инициализирани с нулеви стойности, докато полетата за обект `secondCircle` са инициализирани със стойности x=100, u=200, radius =4.

В случай, че липсва явното извикване на конструктора на базовия клас, компилаторът се опитва да вмъкне конструктора по подразбиране:

```
public ИмеПроизводенКлас(списък формални параметри) : base()
{
    // тяло на конструктора на производния клас
}
```

В този случай, ако част от параметрите са били предназначени за инициализация на полетата от наследената част, техните стойности се губят и инициализацията е съгласнот това, което е предвидено от конструктора без параметри на базовия клас.

Клас System.Object има подразбиращ се конструктор с модификатор public. Не всички класове обаче имат такъв. Ако конструкторът на такъв базов клас не бъде извикан явно, това ще предизвика грешка по време на компилиране, защото на компилатора се налага да вмъкне конструктор без да знае кой. Ако в даден клас е дефиниран конструктор, то компилаторът не създава служебен конструктор по подразбиране, който няма параметри. И ако подобен клас се използва като базов, тогава конструкторът на производния задължително трябва да извика конструктора на базовия.

Правилата за достъп до конструкторите не се различават от тези до останалите методи. Ако даден конструктор в базовия клас е деклариран като ***private***, това означава, че не е достъпен от производния клас.

Относно въпроси по темата на адрес: ln_zh_st@yahoo.com