Тема 11

Абстрактни класове и методи. Програмни интерфейси

1. Абстрактни класове

В програмен език C# е възможно да се декларират класове, които са **абстрактни**. Те служат за наследяване и не е възможно да се дефинира (създаде) обект от абстрактен клас т.е. те не могат да бъдат инстанцирани. Абстрактния клас съдържа абстрактни методи т.е. методи, които нямат имплементация. В производните класове тези методи задължително се предефинират. Абстрактните методи също са виртуални методи. Абстрактният клас, освен абстрактни методи, може да съдържа и методи, който не са абстрактни. Дори е възможно всички методи на даден абстрактен клас да не са абстрактни. Но, ако в даден клас има поне един един абстрактен метод, този клас задължително трябва да бъде абстрактен.

Чрез абстрактните класове и абстрактните методи се реализира т.нар. **принудителен полиморфизъм**. Използва се в случаите когато е необходимо да се реализира полиморфично действие в класове, който нямат общ корен. И тъй като за реализацията на полиморфизъм трябва да има наследствена йерархия, създава се абстрактен базов клас, който да служи като общ родителски клас.

1.1. Деклариране на абстрактен клас и абстрактен метод

За разлика от други програмни езици (например, C++), за да се създаде абстрактен клас, той трябва явно да е обявен като такъв чрез ключова дума ***abstract*** в началото на декларацията. Общият вид на декларацията е следния:

abstract <модификатор> class ИмеКлас

{

// тяло на абстрактен клас

}

В декларацията на класа ключова дума abstract предшества модификатора на достъп.

Пример:

```
abstract public class Figure
{
…
}
```

След като даден клас бъде определен като абстрактен, в него могат да бъдат декларирани неопределен брой абстрактни методи[1]. На практика това са методи без дефиниция т.е. методи, които нямат тяло, а само заглавна част.

[1] Абстрактните методи в C# са аналог на чисто виртуалните функции в C++.

Общият вид на декларацията на абстрактен метод е:

<модификатор> abstract тип ИмеАбстрактен Метод(списък параметри);

В декларацията на абстрактен метод ключова дума ***abstract*** е след модификатора на достъп. Методът няма тяло и след затварящата скоба със списъка параметри се поставя символ **;** (точка и запетая).

Пример:

```
abstract public class Figure
{
        public abstract void Input();
        public abstract void Output();
        public abstract double Area();
}
```

В примера е деклариран абстрактен клас Figure, който съдържа 3 абстрактни метода Input, Output и Area. Предназначението им е за въвеждане и извеждане на данни за фигурата и намиране на нейната площ, съответно.

Абстрактните методи са виртуални по подразбиране и не е необходимо да се обявяват като такива с ключова дума ***virtual***. Ако това бъде направено в програмния код, ще се генерира грешка при компилация.

Абстрактните методи задължително трябва да бъдат предефинирани в производните класове като се обявят като такива с ключова дума ***override***. Пример:

```
public class Rectangle : Figure
    {
        private double height, width;
        public override void Input()
        {
            Console.WriteLine("Object of class Rectangle");
            Console.Write("Height:");
            height = Convert.ToDouble(Console.ReadLine());
            Console.Write("Width:");
            width = Convert.ToDouble(Console.ReadLine());
        }
        public override void Output()
        {
            Console.WriteLine("Object of class Rectangle");
            Console.WriteLine("Height:" + height);
            Console.WriteLine("Width:" + width);

        }
        public override double Area()
        {
            return height * width;
        }
    }
    public class Circle : Figure
    {
        private double radius;
        public override void Input()
        {
```

```
            Console.WriteLine("Object of class Circle");
            Console.Write("Radius:");
            radius = Convert.ToDouble(Console.ReadLine());
        }
        public override void Output()
        {
            Console.WriteLine("Object of class Circle");
            Console.WriteLine("Radius:" + radius);
        }
        public override double Area()
        {
            return radius*radius*Math.PI;
        }
    }
```

В примера са декларирани два класа Rectangle и Circle, които наследяват базовия клас Figure. Производните класове описват фигурите правоъгълник и окръжност. Във всеки един от тези класове трите абстрактни метода са предефинирани т.е. имат имплементация. Ако в тялото на производния клас бъде пропусната имплементация на някои от абстрактните методи, компилаторът ще изведе съобщение за грешка.

1.2. Деклариране на обект от абстрактен клас, реализация на принудителен полиморфизъм

Абстрактните класове имат някои особености, които имат характер на ограничения в сравнение с реалните класове. Те могат да се използват само като базови за други класове. Основното ограничение по отношение на тези класове е, че не могат да бъдат създавани обекти на абстрактни класове. Могат да бъдат създавани референции към абстрактни класове. Например, следният програмен ред ще доведе до грешка по време на компилация:

```
Figure fig= new Figure();
```

Причината е, че се прави опит да бъде дефиниан обект от абстрактен клас. Но може да бъде създадена референция към обект от абстрактен клас:

```
Figure fig;
```

Тази референция се използва при реализацията на полиморфично двействие, в случая се касае за принудителен полимофизъм. Следващият програмен код илюстрира полиморфичните действия чрез използването на класове Rectangle и Circle:

```
Figure fig;
Console.Write("Figure type. Press 1 for Rectangle, press any other key for Circle");
int n = Int32.Parse(Console.ReadLine());
if (n == 1)
{
       fig = new Rectangle();
}
else
{
       fig = new Circle();
}
fig.Input();
fig.Output();
```

```
Console.WriteLine("The area is " + fig.Area());
```

В посочения пример абстрактният клас Figure има само абстрактни методи. Такъв клас се **чист** абстрактен клас.

Възможно е в абстрактен клас да има методи, които не са дефинирани като абстрактни. В абстрактен клас могат да се съдържат и полета. Като цяло, в даден абстрактен клас може да има и членове, който не са абстрактни. Пример:

```
abstract public class Pet
    {
        private string name;
        private int age;

        public Pet(string name, int age)
        {
            this.name = name;
            this.age = age;
        }
        public void Info()
        {
            Console.WriteLine("Name: {0} Age: {1} years", name, age);
        }

        public abstract string GetSound();

    }

class Dog:Pet
    {
        public Dog(string name, int age):base(name, age)
        {
            Console.WriteLine("a new dog is appeared");
        }
        public override string GetSound()
        {
            return "I'm a dog! Bark! Bark!";
        }

    }

    class Cat : Pet
    {
        public Cat(string name, int age) : base(name, age)
        {
            Console.WriteLine("a new cat is appeared");
        }

        public override string GetSound()
        {
            return "I'm a cat! Miaooow!";
        }

}
```

В посочения пример са декларирани абстрактен базов клас Pet и производните класове Dog и Cat. Абстрактният клас има само един абстрактен метод GetSound. В абстрактния клас има полета, конструктор с два параметъра, както и метод, който не е абстрактен (метод Info). В производните класове са дефинирани конструктори с параметри, който правят обръщение към

конструктора на базовия клас. Метод GetSound е предефиниран в двата производни класа.

Следващият примерен код демонстрира работата с тези класове:

```
Pet myPet = new Dog("Шаро",3);
myPet.Info();
Console.WriteLine(myPet.GetSound());

Pet yourPet = new Cat("Маца", 2);
yourPet.Info();
Console.WriteLine(yourPet.GetSound());
```

В случая са дефинирани два обекта от класовете Dog и Cat като се използват референции към абстрактния базов клас Pet. Обектите и от двата класа извикват метод Info. Методът принадлежи на абстрактния клас, без той самият да е абстрактен метод.

2. Интерфейси

2.1. Деклариране на интерфейс

Освен чрез абстрактни класове, някои от проблемите, свързани с наследяването могат да се решат и чрез използване на интерфейси. Интерфейсът е набор от семантически свързани абстрактни членове. Броят на членовете зависи от това какво поведение се моделира с помощта на интерфейса.

Интерфейсите се декларират подобно на класовете, чрез ключова дума interface:

interface ИмеИнтерфейс
{

}

В интерфейса се декларират методи без да се имплементират и без да се посочва модификатор на достъп. Интерфейсите се използват за наследяване и интерфейс не се имплементира т.е. не се дефинира обект от тип интерфейс. От това произтичат следните ограничения:

- Не е допустимо да се поставят полета в интерфейса, дори и те са статични. Полето е имплементация на атрибут на обект , а не е възможно да се дефинира обект от тип интерфейс.
- Не е разрешено да се дефинират конструктори в интерфейс, тъй като конструкторът обикновено изпълнява действия, свързани с инициализацията на полетата.
- Не може да се дефинира деструктор в интерфейс.
- Недопустимо е да се слагат модификатори на достъп пред декларациите на методите в интерфейса, тъй като модификаторът на достъп по подразбира не е public и не може да бъде променян.
- Не е разрешено да се влагат типове в интерфейс (изброявания, структури, класове, други интерфейси)
- Не е разрешено интерфейс да наследява структура или клас. Наследявайки тези типове, интерфейсът би наследил техните полета.

- Даден интерфейс може да наследи един или повече интерфейси.

Интерфейсите се създават с цел да бъдат наследявани или от класове, или от структури. Съответно, в класа или структурата методите на интерфейса имат своята имплементация.

Пример:

```
interface IExamp
{
      string Name();
}

class Examp : IExamp
{
      string IExamp.Name()
      {
      }
}
```

При наследяване на интерфейс е необходимо да се съблюдават следните правила:

- Имената на методите и типовете на връщаните данни трябва да си съответстват точно.
- Всички параметри, техните типове и модификатори in, out, ref трябва да си съответстват точно. Изключение прави модификатор params.
- Името на интерфейса трябва да предшества името на метода. Този подход се нарича явна имплементация на интерфейс.
- При явната имплементация на интерфейс методът не може да има модификатори на достъп. Всички методи, които имплементират интерфейс са общодостъпни.

Въпреки, че в C# липсва множествено наследяване, при което един производен клас наследява няколко базови, в езика има възможност един клас да наследи повече от един интерфейси. Възможно е също един интерфейс да наследи няколко интерфейса. На практика в C# множествено наследяване се реализира с помощта на интерфейси.

2.2. Явна и неявна имплементация на членове на интерфейс

При имплементацията на интерфейса, всички негови членове трябва да имат своята имплементация в производния клас т.е. в класа, който наследява интерфейса. Членовете могат да бъдат имплементирани явно или неявно. При неявната имплементация методите се предефинират така, като дефинира метод на клас. Пример:

```
interface ICar
    {
        void Repair();
        void Drive();
    }

    class Car : ICar
```

```
    {
        public void Repair()      // неявна имплементация
        {
            Console.WriteLine("The car has been reapired");
        }
        public void Drive()       //неясна имплементация
        {
            Console.WriteLine("The car is on the road");
        }

     }
```

Ако бъде изпуснат модификатор на достъп пред името на метода в тялото на класа, тогава методът има модификатора по подразбиране – ***private***.

Извикването на методите на класа е чрез обект от класа. Например, чрез следващите програмни редове се дефинира обект от клас Car и последователно се извикват двата имплементирани метода:

```
Car myCar = new Car();
myCar.Repair();
myCar.Drive();
```

При явната имплементация на членове на интерфейс преди името на метода се записва името на интерфейса:

```
class Car : ICar
{
        void ICar.Repair()        // явна имплементация
        {
            Console.WriteLine("The car has been reapired");
        }
        void ICar.Drive()         // явна имплементация
        {
            Console.WriteLine("The car is on the road");
        }
 }
```

При явната имплементация на методите не се записва модификатор на достъп. По подразбиране той е ***public***. Друга особеност е, че такъв метод не може да бъде извикан чрез обект от тип клас. Например, следващият програмен ред би предизвикал грешка:

```
myCar.Repair();
```

Извикването на явно имплементиран метод е чрез референция към интерфейс. Пример:

```
ICar car=myCar;
car.Drive();
car.Repair();
```

В случая идентификатор car е референция към интерфейс, а myCar е обект от клас Car. Извикването на двата явно имплементирани метода е чрез референцията от тип интерфейс.

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