Тема 10

Виртуални методи и полиморфизъм

1. Полиморфизъм и динамично свързване

Използването на методи с еднакви имена е често срещана практика в програмирането. Причината е, че името на метода се свързва с неговото действие т.е. какво поведение на обекта описва този метод. Ако методите описват еднотипни действия, нормално е те да са с еднакви имена. Когато става дума за методи с еднакви имена от един клас, такива методи се наричат **предекларирани (overloaded)**. Механизмът на предекларирane определя, че по време на компилация, в обръщението към методите се сравняват фактическите и формалните параметри и се избира един от методите с еднакви имена. Ето защо, тези методи трябва да се различават по броя и/или типа на параметрите, което определя тяхната еднозначност. В крайна сметка, обръщението към методи с еднакви имена се свежда до класическото обръщение към функция. Тъй като процесът на реализация на обръщението към метод (или функция) приключва по време на компилация, този механизъм се нарича ***статично свързване (early binding)***.

Ако в различни класове има методи с еднакви имена, компилаторът „разбира“ кой метод да извика според това какъв е обекта или класа, който стои в лявата част на оператор . (точка) в програмния ред. Например, нека и в двата класа Rectangle и Circle има метод Area(), който намира площта на съответната фигура. В програмния код, при обръщението към метод Area() в единия случай се извиква метода на клас Rectangle, а в другия случай – метода на клас Circle. Кой от двата метода Area ще бъде извикан, се определя по типа на обекта:

```
Rectangle firstRectangle = new Rectangle(100, 200);
Console.WriteLine(“The area is “ +firstRectangle.Area());

Circle secondCircle = new Circle(4);
Console.WriteLine(“The area is “ +secondCircle.Area());
```

В този случай отново има статично свързане, тъй като процесът на реализация на обръщението към метод приключва по време на компилация.

Друг случай на използване на методи с еднакви имена възниква при наследяване на класовете, когато собствените методи на наследения клас имат имена, съвпадащи с имената на методи от базовия клас. В този случай отново има значение чрез какъв обект ще бъде извикан метода: ако обекът е от базовия клас, извиква се методът от базовия; ако е от производния, извиква се методът на производния клас. При наследяването на класове обаче е възможно референция към обект от базов клас да сочи както обект от базов клас, така и обект от производен клас. Тъй като стойностите на референцията са известни едва по време на изпълнение на програмата, едва тогава може да се определи кой от двата метода ще бъде извикан. В случая се казва, че има ***динамично свързване (late binding).*** На практика, чрез динамичното свързване се реализира ***полиморфизъм***.

Терминът **полиморфизъм** в обектно-ориентираното програмиране определя, че едни и същи (еднотипни) действия се реализират по различен

начин в зависимост от обектите, върху които се прилагат. Такива действия се наричат ***полиморфични***. За да се реализира полиморфно действие, класовете на обектите, върху които се прилага това действие, трябва да имат общ корен т.е. да бъдат производни на един и същи клас. Реализацията на поломорфизма е чрез методи, които се наричат **виртуални**. Виртуалните методи позволяват да бъдат извикани различни версии (реализации) на един и същ метод, според това какъв е типът на обекта. По този начин се позволява да се работи с група взаимносвързани обекти по идентичен начин.

2. Присвояване на обекти

При присвояване на обекти от един и същи клас трябва да се има предвид, че класовете са референтни типове, което означава, че присвояването на един обект на друг ще доведе до това, че две референции сочат един и същ обект в динамичната памет. Например, след изпълнението на следващия програмен код референциите firstPoint и secondPoint сочат един и същи обект:

```
Point firstPoint = new Point();
Point secondPoint = new Point();
secondPoint = firstPoint;
```

Същият ефект има и инициализацията чрез вече съществуващ обект от същия клас:

```
Point firstPoint = new Point();
Point secondPoint = firstPoint;
```

При дефиниране на обекти от базови и производни класове е възможно на обект от базов клас да се присвои обект от производен клас т.е. обект от производен клас може да служи за инициализация на обект (референция) от базов клас. Възможно е също да се осъществи обръщение към обект чрез променлива от друг тип, ако използвания тип е клас, намиращ се на по-високо ниво в йерархията на наследяване. Обратното не е възможно. Например, нека клас Circle да е производен на клас Point. Възможно е да се декларира референция на клас Point, която да сочи обект от клас Circle:

```
Point a = new Circle();
```

Обратното не е възможно. Например, следващият програмен ред ще предизвика грешка при компилиране:

```
Circle cir = new Point();
```

Възможността референция към базов клас да сочи обект както от базовия, така и обект от някой от производните класове е в основата на реализацията на полиморфизма в език C#.

3. Дефиниране и извикване на виртуални методи

Реализацията на полиморфизъм в език C# е чрез виртуалните методи. За да се дефинират методи като виртуални, в базовия и в производния клас тези методи трябва да са с еднакви имена, еднакъв тип на върнат резултат, еднакъв брой и тип параметри т.е. сигнатурата на метода трябва да е една и съща.

Виртуалните методи се дефинират в базовия клас чрез ключова дума ***virtual***, а в наследените се предефинират с ключова дума ***override***. Ключовите

думи ***virtual*** и ***override*** стоят след модификатора за достъп (***public***). Такива методи се наричат предефинрани (***overridden***) и това се определя с ключовата дума ***override***.

Пример:

```
class Tree
    {
        public virtual void Info()
        {
            Console.WriteLine("It is just a tree");
        }
    }

class AppleTree : Tree
    {
        public override void Info()
        {
            Console.WriteLine("This is an apple-tree");
        }
    }

class CherryTree : Tree
    {
        public override void Info()
        {
            Console.WriteLine("This is a cherry-tree");
        }
}
```

Създаден е клас Tree и производните класове AppleTree и CherryTree. Метод Info е виртуален. В базовия клас методът е дефиниран с ключова дума virtual, а реализацията му в производните класове е с ключова дума override.

Извикването на виртуален метод по механизма на динамичното свързване е чрез референция на обект от базов клас. Причината е, че референция към обект от базов клас може да сочи както обект от базов клас, така и обект от производен. В следващия програмен код се демонстрира извикването на метод Info като се използва рефенция към базовия клас:

```
Tree tr = new Tree();
AppleTree apple = new AppleTree();
CherryTree cherry = new CherryTree();
Console.WriteLine("For apple-tree press 1, for cherry-tree press 2");
short n = Int16.Parse(Console.ReadLine());
switch (n)
{
      case 1: tr = apple;
                    break;
      case 2: tr = cherry;
                    break;
}
tr.Info();
```

Тъй като стойността на референцията tr ще бъде известна едва по време на изпълнение, тогава ще бъде определено коя от версиите на виртуалният метод ще бъде извикана. Това зависи от въведената стойност за променливата n. При стойност 1 се извиква методът от клас AppleTree, а при стойност 2 – метод

Info от клас CherryTree. Ако въведената стойност е различна от 1 и 2, тогава референцията tr продължава да сочи обект от базовия клас и ще бъде извикат метод Info от клас Tree.

Ако в програмния код липсва инициализация на референцията към базовия клас, тогава компилаторът ще генерира грешка поради липса на инициализация. Например, следният програмен код ще генерира точно такава грешка:

```
Tree tr; //създадена е референция, която не сочи към обект
AppleTree apple = new AppleTree();
CherryTree cherry = new CherryTree();
Console.WriteLine("For apple-tree press 1, for cherry-tree press 2");
short n = Int16.Parse(Console.ReadLine());
switch (n)
{
       case 1: tr = apple;
                     break;
       case 2: tr = cherry;
                     break;
}
tr.Info(); //грешка! “Use unassigned local variable tr”
```

Задължително да се осигури инициализация на променливата tr. Това може да се реализира и чрез следващия вариант на програмния код:

```
Tree tr;
AppleTree apple = new AppleTree();
CherryTree cherry = new CherryTree();
Console.WriteLine("For apple-tree press 1, for cherry-tree press 2");
short n = Int16.Parse(Console.ReadLine());
switch (n)
{
                case 1: tr = apple;
                    break;
                case 2: tr = cherry;
                    break;
                default: tr = new Tree();
                   break;
}

tr.Info();
```

В случая, част default в конструкцията switch осигурява инициализацията на обекта tr.

Съществуват някои важни правила, които трябва да се съблюдават при работата с виртуалните методи:

- Статичен метод не може да бъде деклариран като виртуален. Причината е, че статичните методи се стартират чрез клас, а полиморфичното действие се прилага към конкретен обект от клас.
- Не може да се декларира виртуален метод, който е със статут private. Такъв метод няма как да бъде предефиниран в производния клас.
- Методите, които са виртуални и в базовия, и в производния клас трябва да са идентични т.е. да имат едни и същи тип (на върнат резултат), брой и тип на параметрите.

- Двата метода трябва да имат един и същ модификатор на достъп, за разлика от C++, където предефинираните виртуални функции могат да са с различни модификатори.
- Може да се предефинира само виртуален метод. Ако методът в базовия клас не е виртуален и програмистът се опита да го предефинира, компилаторът ще изведе съобщение за грешка. В това има определен замисъл – би трябвало още в базовия клас да се определи кои методи могат да се предефинират и кои – не.
- Ако производният клас не декларира метода чрез ключова дума ***override***, тогава той не предефинира виртуалния метод от базовия клас. С други думи, това е вече друг метод, макар и със същото име. Това ще предизвика предупреждение за скриване при компилиране. Предупреждението може да бъде подтиснато чрез ключова дума **new**.

- Даден метод, който е **override** т.е. предефиниран виртуален метод не може да бъде обявен отново като **virtual** т.е. съвместното използване на двете ключови думи virtual и override не е допустимо.

4. Методи от тип new

Ключовата дума new се използва при предефиниране на метод за предотвратяване на конфликти между идентификаторите, в случая между имената на методите в базовия и производните класове. Когато се изгражда йерархия от класове програмистът се стреми да пише смислени идентификатори и може да попадне в ситуация да използва име, което вече е заето в йерархията.

Пример:

```
class Person
{
      public string Name()
      {
      }
}

class Student : Person
{
      public string Name()
      {
      }
}
```

В този случай компилаторът генерира код, но издава предупреждение за съвпадение на имената. Ако има наследяване на клас Student т.е. добавяне на следващ клас в йерархията, съвпадението на имената на методите ще причини объркване. Ако въпреки това програмистът държи на съвпадението на имената, за да се подтисне предупреждението може да се използва ключова дума ***new***. Програмният код ще изглежда по следния начин:

```
class Person
{
      public string Name()
      {
      }
}

class Student : Person
{
```

```
  new public string Name()
  {
  }
}
```

Така предупреждението за съвпадение на имената на методите ще бъде подтиснато.

Предупреждението за наличието на методи с еднакви имена в програмния код на базовия и на производния клас има смисъл, тъй като се очаква ако има еднакви методи в йерархията от класове, то те да са виртуални. Но понеже е възможно да има съвпадение на имената на методите в различните класове от една наследствена йерархия, без те да са виртуални, тогава е препоръчителна употребата на ключова дума ***new***.

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