Тема 5

Инициализиране на обекти

1. Инициализиране на обекти - особености

Първоначалното присвояване на стойности на програмните единици (инициализация) е важно за всеки език за програмиране. Например, ако на една променлива не бъде зададена начална стойност, това би могло да доведе до трудно откриваеми грешки, тъй като тя участва в програмата със случайна стойност. Проблемът с инициализирането е толкова важен, че в език C# не е позволено да се използват неинициализирани променливи. Компилаторът се грижи да зададе начални стойности на полетата, докато инициализацията на локалните променливи е грижа на програмиста. Ако една променлива не е инициализирана, компилаторът издава съответното предупреждение.

В общия случай, когато се касае за обекти, не само за променливи, инициализацията може да включва и други действия, освен задаването на стойности. Такива действия са: копиране на обект; резервиране на памет; инициализиране на външни устройства; запомняне на текущото състояние на програмата и др. От друга страна, преди даден обект да престане да бъде активен се налага изпълнението на завършващи действия. Например, възстановяване на начални стойности, освобождаване на памет и др.

В обектно ориентиранте програмни езици инициализацията на обектите е поверена на метод, наречен **конструктор**. Този метод се изпълнява автоматично при дефиниране на обект (екземпляр) от класа. Конструкторът на класа (понякога се нарича псевдометод) се изпълнява винаги, когато се дефинира обект и изпълнява всички начални (инициализиращи) действия, свързани с неговото създаване, включително и инициализацията на полетата. Заключителните действия, свързани с премахването на обекта, се изпълняват от друг метод, наречен **деструктор**. Подобно на конструктора, той също се изпълнява автоматично, но при излизане от областта на действие на обекта. Обикновено заключителните действия са свързани с освобождаване на ресурси (премахване на заделена памет), което в платформата .NET е поверено на системата за управление на паметта (GC). Така програмистът е освободен от грижата за освобождаване на ресурсите и затова рядко се налага деструктор на класа да бъде добавян от програмиста.

Особеностите на констукторите и деструкторите, като методи на класовете, могат да бъдат обобщени по следния начин:

- Името на конструктора съвпада с името на класа.
- Името на деструктора е името на класа, предшествано от символ ~ (тилда).
- Един клас може да има няколко конструктора, но само един деструктор.

• Ако даден клас има повече от един конструктори, те трябва да се различават по броя и типа на параметрите. В случая се касае за предефиниране на метод, в случая - на конструктор.

• Деструкторът е метод без параметри.

• Възможно е, конструктор на клас да е метод без параметри. В този случай, той се нарича *конструктор по подразбиране*.

• Типовете на върнатите стойности от конструктор и деструктори са съответно: референция към новосъздаден обект (***this***) и ***void***. Тези типове не могат да бъдат променяни и поради това в дефинициите и декларациите на тези методи не се задава техния тип. Съответно, в телата на конструктор и деструктор липсва оператор return.

2. Дефиниране на конструктор

Общият вид на дефиницията на конструктор е следния:

ИмеКлас(списък формални параметри)
{
//тяло на конструктор
}

На практика, името на класа е и име на конструктора, тип на върнат резултат не се посочва. Подобно на останалите методи, в скобите се посочват параметрите на конструктора.

Както при всеки друг член на класа, се посочва модификатора на достъп. Ако не се посочи такъв, по подразбиране той е **private**. В повечето случаи модификаторът на достъп за конструктор на клас е **public**.

Пример:

```
class Rectangle
    {
        private double height;
        private double width;

        public Rectangle(double h, double w)
        {
            height=h;
            width=w;
        }

        public double Area()
        {
            return height * height;
        }
    }
```

В посочения пример, в клас `Rectangle` е дефиниран конструктор с два параматъра, които дават своите стойности на полетата. При дефиниране на обект от класа, обръщението към метод конструктор е след ключова дума **new**:

```
class Program
{
```

```
        static void Main()
        {
              Rectangle firstRect = new Rectangle(2.5, 3.7);
        }
}
```

Така в тялото на метод при дефинирането на обект **`firstRect`** се извиква конструктор на класа, който е с два параметъра от тип double и полетата на обекта се инициализират със стойности 2.5 и 3.7, съответно. Дефинирането на обекта в метод Main е възможно само защото модификаторът на конструктора на класа е **public**.

Конструктор на класа може да няма параметри. Например, в клас `Rectangle` може да се добави такъв конструктор:

```
public Rectangle()
{
      height=0.0;
      width=0.0;
}
```

Съответно, за да се извика този конструктор при дефиниране на обект, не се посочват параметри при обръщението към конструктора:

```
Rectangle secondRect = new Rectangle();
```

В случая, полетата на обект **`secondRect`** се инициализират със стойност 0, понеже се извиква конструкторът без параметри.

3. Конструктор по подразбиране

Понякога конструкторът без параметри се нарича и конструктор по подразбиране. Причината е, че ако в програмния код, при дефиницията на даден клас програмистът не добави какъвто и да било конструктор, компилаторът се грижи да добави служебен конструктор, който няма параметри. Този служебен конструктор се нарича **конструктор по подразбиране**. Той няма параматри и съответно дефинирането на обект от класа ще бъде чрез програмен ред:

```
ИмеКлас имеОбект = new ИмеКлас();
```

В началото **`ИмеКлас`** е име на тип, докато след ключова дума new **`ИмеКлас()`** е обръщението към конструктора на класа. И ако програмистът не е добавил конструктор, това е обръщението към служебния конструктор по подразбиране. Този конструктор е без параметри и затова в скобите липсват такива.

Ако програмистът добави конструктор в декларацията на класа, независимо дали ще е с параматри или без, компилаторът не добавя служебен конструктор без параметри.

Пример:

```
class Circle
    {
        private double radius;

        public Circle(double r)
        {
            radius=r;
```

```
    }

    public double Area()
    {
        return radius*radius*Math.PI;
    }
}
```

В този случай, за клас `Circle` компилаторът няма да добави конструктор без параметри(`Circle()`). Това означава, че за да бъдат дефинирани обекти от този клас, ще бъде използван само конструкторът с един параматър:

```
Circle cir1 = new Circle(2.4);
```

Програмен ред Circle cir2 = new Circle(); би предизвикал грешка при компилиране, тъй като в класа липсва конструктор без параметри.

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