Course Object-Oriented Programming

Lecture 5

Constructors. Method Overloading. C# properties. This keyword. Static member.

Content

- C# constructors
- **this** Keyword
- Static and Instance members
- Methods overloading
- C# properties

C# constructors

- A constructor is a public method with the same name as the class with no return type, which is called once upon object creation.
- Constructors may be passed one or more arguments.
- Any construct with no arguments is called the default constructor and if no constructors are written then the compiler creates a default constructor.
- There can be more than one constructor associated with an object if this is useful.
- Each constructor must have unique signature, that is, the types and number of arguments are unique. This is called overloading, something which can be done with all methods (see later).

C# constructors

syntax:

[access-modifier] class-name ([parameters list])

{

constructor-body

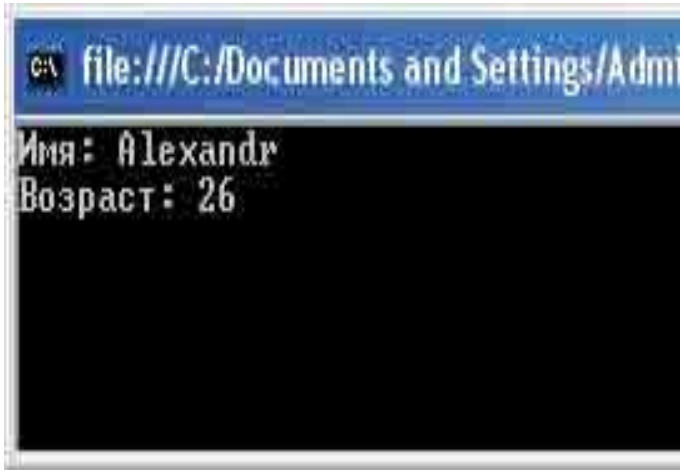
};

[access-modifier] – public;

class-name – the same name as the class, where constructor is created;

[parameters list] – optional;

```
namespace ConsoleApplication1
{
   class MyClass
   {
      public string Name;
      public byte Age;

      // Constructor with parameters
      public MyClass(string s, byte b)
      {
         Name = s;
         Age = b;
      }

      public void reWrite()
      {
         Console.WriteLine("Имя: {0}\nВозраст: {1}", Name, Age);
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         MyClass ex = new MyClass("Alexandr", 26);
         ex.reWrite();
         Console.ReadLine();
      }
   }
}
```



this Keyword

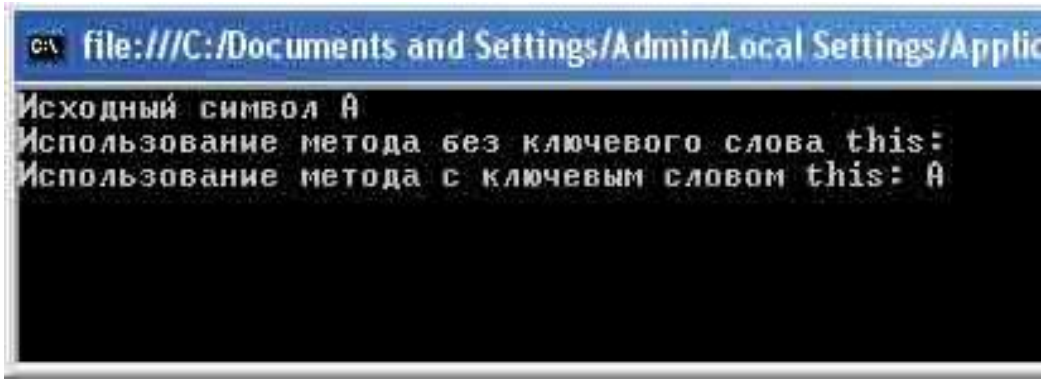
The keyword **this** refers to the current instance of an object. It is used in a variety of settings. Firstly it can be used in the case of ambiguous and unrecommended naming.

For example:

```
public complex(double real)
{
this.real = real;
}
```

Here this distinguishes between the passed in real and the real of the current object instance. This vagueness is preferably avoided by using a sensible naming strategy.

```
namespace ConsoleApplication1
{
   class MyClass
   {
      public char ch;

          public void Method1(char ch)
      {
         ch = ch;
      }

      public void Method2(char ch)
      {
         this.ch = ch;
      }
   }

   class Program
   {
      static void Main()
      {
         char myCH = 'A';
         Console.WriteLine("Исходный символ {0}",myCH);

         MyClass obj = new MyClass();

         obj.Method1(myCH);
         Console.WriteLine("Использование метода без ключевого слова this: {0}",
       obj.ch);
         obj.Method2(myCH);
         Console.WriteLine("Использование метода с ключевым словом this: {0}", obj.ch);
         Console.ReadLine();
      }
   }
}
```



Static and Instance Members

The properties and methods of class can be either instance or static.

By default properties and methods are instance and this means that they are to be used only with a specific instance of a class.

Sometimes it makes sense to have data and methods that operate at a class level, that is, independently of any particular instance. Such methods are prefixed with the keyword **static** in their declaration and can only be accessed using the class name rather than through a particular instance.

Static data and methods can be accessed/invoked inside in instance methods however static methods can only access static data and obviously don't have access to the this object reference.

```
namespace ConsoleApplication1
{
   class myCircle
   {
      // 2 methods return area and long of circle
      public static double SqrCircle(int radius)
      {
         return Math.PI * radius * radius;
      }

      public static double LongCircle(int radius)
      {
         return 2 * Math.PI * radius;
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         int r = 10;
         // Use the methods of another class without creation an instance of this class
      Console.WriteLine("Площадь круга радиусом {0} =
      {1:#.##}",r,myCircle.SqrCircle(r));
      Console.WriteLine("Длина круга равна {0:#.##}",myCircle.LongCircle(r));

         Console.ReadLine();
      }
   }
}
```

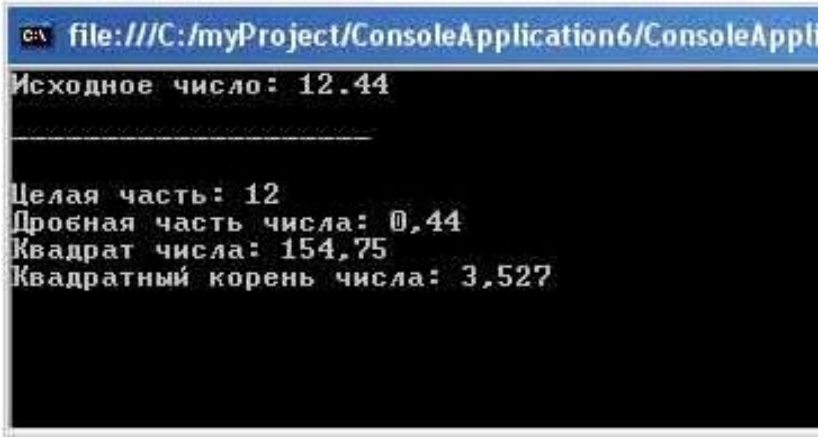
There are some limitations for using static methods:

- In the static method must be absent **this** reference, as such method doesn’t work with any object;
- In the static method allowed immediate call only other static methods, but not an instance method of the same class. The fact that the instance methods operate on the specific objects, and static type method is not called for object;
- Similar restrictions are imposed on the static data. For the static method are directly accessible only to other static data defined in its class.

Static keyword can be used for:

- Methods;
- Constructors;
- Classes;
- Properties (Data);

```
namespace ConsoleApplication1
{
static class MyMath
   {
      static public int round(double d)
      {
         return (int)d;
      }
     static public double doub(double d)
      {
         return d - (int)d;
      }
     static public double sqr(double d)
      {
         return d * d;
      }
      static public double sqrt(double d)
      {
         return Math.Sqrt(d);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Исходное число: 12.44\n\n----------\n");
         Console.WriteLine("Целая часть: {0}",MyMath.round(d: 12.44));
         Console.WriteLine("Дробная часть числа: {0}",MyMath.doub(d: 12.44));
         Console.WriteLine("Квадрат числа: {0:#.##}",MyMath.sqr(d: 12.44));
         Console.WriteLine("Квадратный корень числа: {0:#.###}",MyMath.sqrt(d: 12.44));
         Console.ReadLine();
      }
   }
}
```

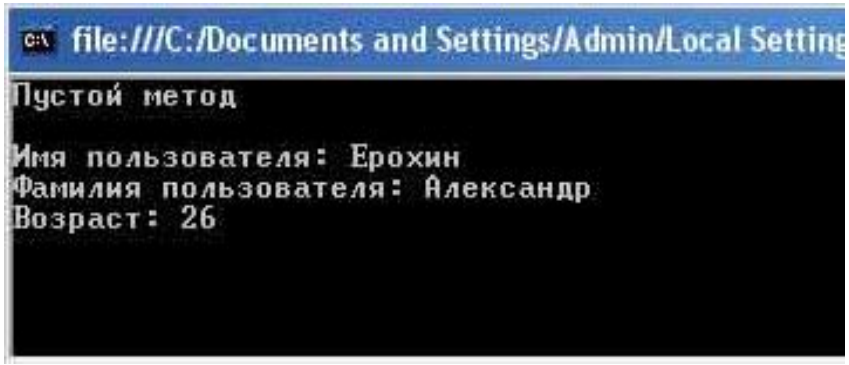


Methods overloading

We already encountered overloading in the case of constructors. We can apply the same logic to all methods, that is we can create many versions of the same method, provided a unique signature is present.

Overloading was developed to reduce the number of different method names to be created by the programmer and it also makes life easier for the end object user. Although there is no restriction on what functionality goes into methods of the same name, it makes good logical and functional sense that the behavior should be in some way related. For example, a method named add shouldn’t implement subtraction.

```
namespace ConsoleApplication1
{
   class UserInfo
   {
      public void ui()
      {
         Console.WriteLine("Пустой метод\n");
      }
      public void ui(string Name)
      {
         Console.WriteLine("Имя пользователя: {0}",Name);
      }
      public void ui(string Name, string Family)
      {
         Console.WriteLine("Имя пользователя: {0}\nФамилия пользователя: {1}",Name,Family);
      }
      public void ui(string Name, string Family, byte Age)
      {
         Console.WriteLine("Имя пользователя: {0}\nФамилия пользователя: {1}\nВозраст: {2}", Name, Family, Age);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         UserInfo user1 = new UserInfo();
         user1.ui();
         user1.ui("Ерохин", "Александр", 26);
         Console.ReadLine();
      }
   }
}
```



C# Properties

```
namespace ConsoleApplication4
{
   class Program
   {
      static void Main(string[] args)
      {
         calculator calc = new calculator();
         Console.WriteLine(calc.plus(10,15));
         Console.WriteLine(calc.minus(10, 15));
         Console.WriteLine(calc.division(10, 15));
         Console.WriteLine(calc.multip(10, 15));
         Console.ReadLine();
      }
   }
   class calculator
   {
      public int plus(int a,int b) { return a + b; }
      public int minus(int a, int b) { return a - b; }
      public int division(int a, int b) { return a / b; }
      public int multip(int a, int b) { return a * b; }
   }
}
```

```
namespace ConsoleApplication4
{
   class Program
   {
      static void Main(string[] args)
      {
         calculator calc = new calculator();
         calc.a = 10;
         calc.b = 15;
         Console.WriteLine(calc.plus());
         Console.WriteLine(calc.minus());
         Console.WriteLine(calc.division());
         Console.WriteLine(calc.multip());
         Console.ReadLine();
      }
   }
   class calculator
   {
      public int a, b;
      public int plus() { return a + b; }
      public int minus() { return a - b; }
      public int division() { return a / b; }
      public int multip() { return a * b; }
   }
}
```

Thank you!