Object Oriented Programming

5. Constructor and Destructors

Assistant Lecturer

Sipan M. Hameed

www.sipan.dev

2024-2025

Contents

5. Constructor and Destructors in C#

5.1. Introduction

A constructor can be used, where every time an object gets created and if we want some code to be executed automatically. The code that we want to execute must be put in the constructor. Constructors are special methods called when a class is instantiated.

- Constructor will not return anything.
- Constructor name is same as class name.
- By default, C# will create default constructor internally.
- Constructor with no arguments and nobody is called default constructor.
- Constructor with arguments is called parameterized constructor.
- Constructor by default public.
- We can create private constructors.
- A method with same name as class name is called constructor there is no separate keyword.

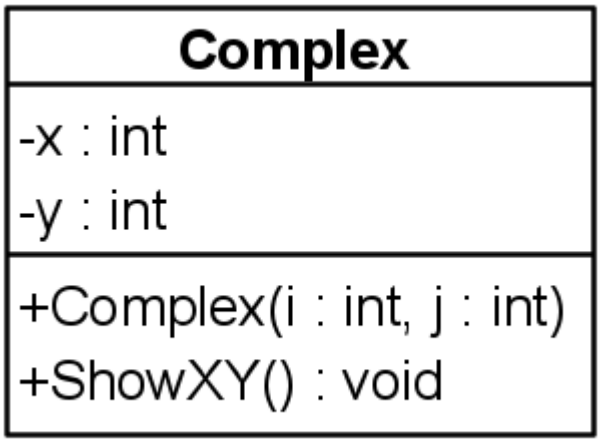
The general form of a C# constructor is as follows

```
modifier constructor_name (parameters)
{
//constructor body
}
```

The modifiers can be private,public, protected or internal.The name of a constructor must be the name of the class, where it is defined. A constructor can take zero or more arguments. A constructor with zero arguments (that is no-argument) is known as default constructor. Remember that there is not return type for a constructor.

5.1.1 UML Class Diagram

The following class contains a constructor, which takes two arguments.



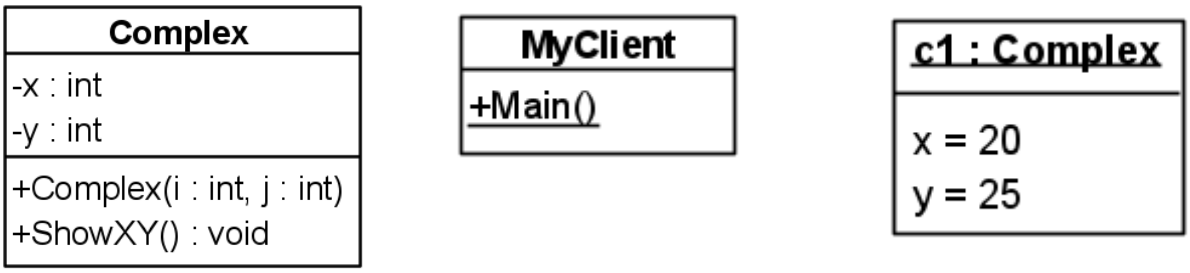

```
class Complex
{
    private int x;
    private int y;
    public Complex(int i, int j)
    {
        x = i;
        y = j;
    }
    public void ShowXY()
    {
        Console.WriteLine(x + "i+" + y);
    }
}
```

The following code segment will display 20+i25 on the command prompt.

```
Complex c1 = new Complex (20,25);
c1.ShowXY (); // Displays 20+i25
```

That is when we create the object of the class Complex, it automatically calls the constructor and initializes its data members x and y. We can say that constructor is mainly used for initializing an object. Even it is possible to do very complicated calculations inside a constructor. The statement inside a constructor can throw exceptions also.

If we don't provide a constructor with a class, the C# provides a default constructor with an empty body to create the objects of the class. Remember that if we provide our own constructor, C# do not provide the default constructor.

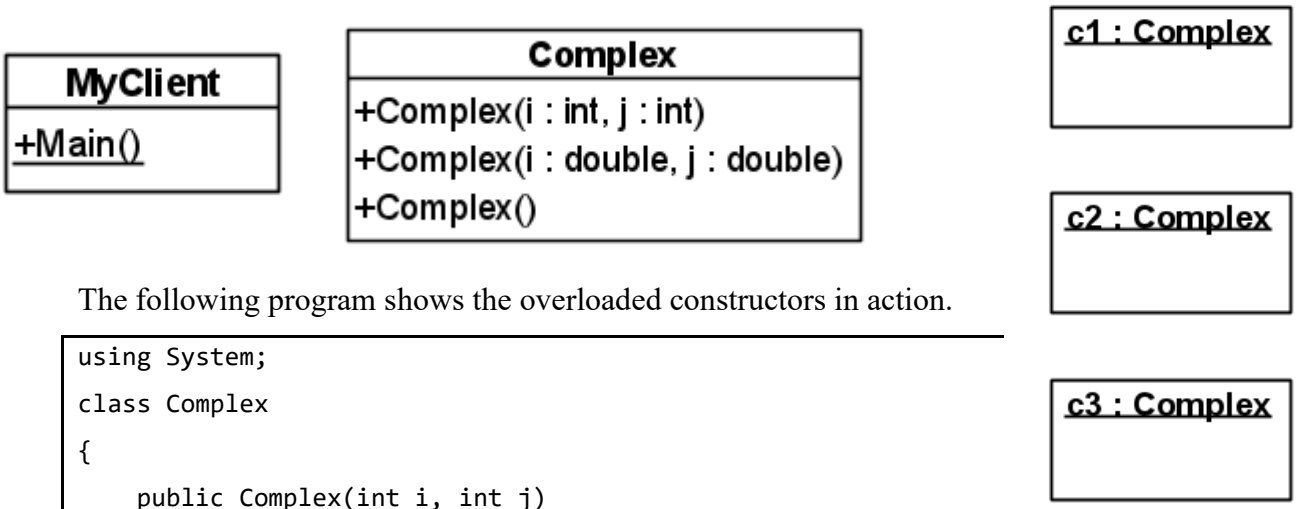


The complete program is given below

```
using System;
class Complex
{
    private int x;
    private int y;
    public Complex(int i, int j) // constructor with 2 arguments
    {
        x = i;
        y = j;
    }
    public void ShowXY()
    {
        Console.WriteLine(x + "i+" + y);
    }
}
class MyClient
{
    public static void Main()
    {
        Complex c1 = new Complex(20, 25);
        c1.ShowXY();
    }
}
```

5.1.2 Constructor Overloading

Just like member functions, constructors can also be overloaded in a class. The overloaded constructor must differ in their number of arguments and/or type of arguments and/or order of arguments.



The following program shows the overloaded constructors in action.

```
using System;
class Complex
{
    public Complex(int i, int j)
    {
        Console.WriteLine("constructor with 2 integer arguemets");
    }
    public Complex(double i, double j)
    {
        Console.WriteLine("constructor with 2 double arguments");
    }
    public Complex()
    {
        Console.WriteLine("no argument constructor");
    }
}
class MyClient
{
    public static void Main()
    {
// displays 'constructor with 2 integer arguments'
        Complex c1 = new Complex(20, 25);
// displays 'constructor with 2 double arguments'
        Complex c2 = new Complex(2.5, 5.9);
//displays 'no argument constructor'
        Complex c3 = new Complex();
    }
}
```

5.1.3 Private Constructors

We already see that, in C#, constructors can be declared as public, private, protected or internal. When a class declares only private constructors, it is not possible other classes to derive from this class or create an instance of this class. Private constructors are commonly used in classes that contain only static members. However, a class can contain both private and public constructor and objects of such classes can also be created, but not by using the private constructor.

The following is a valid program in C#

```
using System;
class Complex
{
    private Complex(int i, int j)
    {
        Console.WriteLine("constructor with 2 integer arguments");
    }
    public Complex()
    {
        Console.WriteLine("no argument constructor");
    }
}
class MyClient
{
    public static void Main()
    {
        Complex c3 = new Complex();
    }
}
```

5.1.4 Static Constructors

The normal constructors, which we explained till now, can be used for the initialization of both static and non-static members. But C# provides a special type of constructor known as static constructor to initialize the static data members when the class is loaded at first. Remember that, just like any other static member functions, static constructors can't access non-static data members directly.

The name of a static constructor must be the name of the class and even they don't have any return type. The keyword static is used to differentiate the static constructor from the normal constructors. The static constructor can't take any arguments. That means there is only one form of static constructor, without any arguments. In other way it is not possible to overload a static constructor.

We can't use any access modifiers along with a static constructor.

For example: -

```
 using System;
class Complex
{
    static Complex()
    {
        Console.WriteLine("static constructor");
    }
}
class MyClient
{
    public static void Main()
    {
        Complex c;
        Console.WriteLine("hiii");
        c = new Complex();
    }
}
```

Note that static constructor is called when the class is loaded at the first time. However, we can't predict the exact time and order of static constructor execution. They are called before an instance of the class is created, before a static member is called and before the static constructor of the derived class is called.

Like non-static constructors, static constructors can't be chained with each other or with other non-static constructors. The static constructor of a base class is not inherited to the derived class.

5.1.5 Destructors

The .NET framework has an in-built mechanism called Garbage Collection to de-allocate memory occupied by the un-used objects. The destructor implements the statements to be executed during the garbage collection process. A destructor is a function with the same name as the name of the class but starting with the character ~.

Example:

```
class Complex
{
    public Complex()
    {
        // constructor
    }
    ~Complex()
    {
        // Destructor
    }
}
```

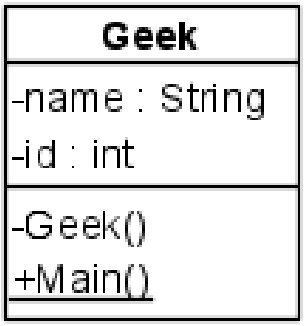
Remember that a destructor can't have any modifiers like private, public etc. If we declare a destructor with a modifier, the compiler will show an error. Also, destructor will come in only one form, without any arguments. There is no parameterized destructor in C#.

Destructors are invoked automatically and can't be invoked explicitly. An object becomes eligible for garbage collection, when it is no longer used by the active part of the program. Execution of destructor may occur at any time after the instance or object becomes eligible for destruction.

5.2. Practical Examples:

5.2.1 Parameterized Constructor

A constructor having at least one parameter is called as parameterized constructor. It can initialize each instance of the class to different values.




```
using System;
namespace PCExample
{

    class Geek
    {

        // data members of the class.
        String name;
        int id;

        Geek(String name, int id)
        {
            this.name = name;
            this.id = id;
        }

        // Main Method
        public static void Main()
        {

            // This will invoke parameterized
            // constructor.
            Geek geek1 = new Geek("GFG", 1);
            Console.WriteLine("GeekName = " + geek1.name +
                              " and GeekId = " + geek1.id);
        }
    }
}
```

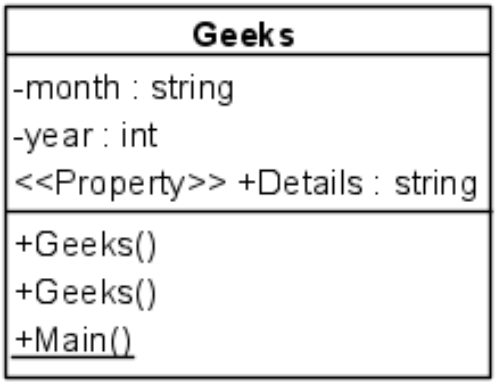
Output:

```
GeekName = GFG and GeekId = 1
```

5.2.2 Copy Constructor

This constructor creates an object by copying variables from another object. Its main use is to initialize a new instance to the values of an existing instance.

```
using System;
namespace copyConstructorExample
{

    class Geeks
    {

        private string month;
        private int year;

        // declaring Copy constructor
        public Geeks(Geeks s)
        {
            month = s.month;
            year = s.year;
        }

        // Instance constructor
        public Geeks(string month, int year)
        {
            this.month = month;
            this.year = year;
        }
        public string Details
        {
            get
            {
                return "Month: " + month.ToString() +
                          "\nYear: " + year.ToString();
            }
        }

        public static void Main()
        {

            // Create a new Geeks object.
            Geeks g1 = new Geeks("June", 2018);

            // here is g1 details is copied to g2.
            Geeks g2 = new Geeks(g1);

            Console.WriteLine(g2.Details);
        }
    }
}
```



Output:

```
Month: June
Year: 2018
```