Duhok Polytechnic University
Zakho Technical Institute
Department of Information Technology

Object Oriented Programming

2024-2025

Lecture 5: **Constructor**

Lecturer

Sipan M. Hameed

www.sipan.dev

Contents

Constructor

- It is called **constructor** because it constructs the values of data members of the class.
- It is invoked whenever an object is created.

Characteristics of the Constructor:

- ✓ It is the function that have the **same name of the class**
- ✓ Declared as **public**
- ✓ **Invoked automatically** when the object is created
- ✓ Does not have return type, not even void

Creating a constructor

- To create a class **constructor**, use the **public** keyword followed by the name of the class and the constructor parameters, if any.

```
class ClassName
{
    // Default constructor (parameterless)
    public ClassName()
    {
        // Constructor code
    }
}
```

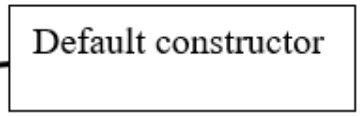
Creating a constructor

```
class ClassName
{
    // Parameterized constructor
    public ClassName(datatype parameter1, datatype parameter2, ...)
    {
        // Constructor code
    }
}
```

Default Constructor

```
class Program
{
    static void Main(string[] args)
    {
        xyz a = new xyz();
        a.printxyz();
        a.setxyz(5, 10, 20);
        a.printxyz();
        Console.ReadLine();
    }
}
```

```
class xyz
    {
        private int x, y, z;
        public xyz()
        {
            x = 0; y = 0; z = 0;
        }
        public void setxyz(int a,int b, int c)
        {
            x = a;        y =b;           z =c;
        }

        public void printxyz()
        {
            Console.WriteLine("X : {0} , Y: {1} , Z: {2}", x, y, z);
        }
    } //end of class
```



Parameterized Constructor

```
class Program
{
    static void Main(string[] args)
    {
        Student S1 = new Student();
        S1.PrintInformation();

        Student S2 = new Student("Saman",20, 'A');
        S2.PrintInformation();

        Console.ReadLine();
    }

  }
```

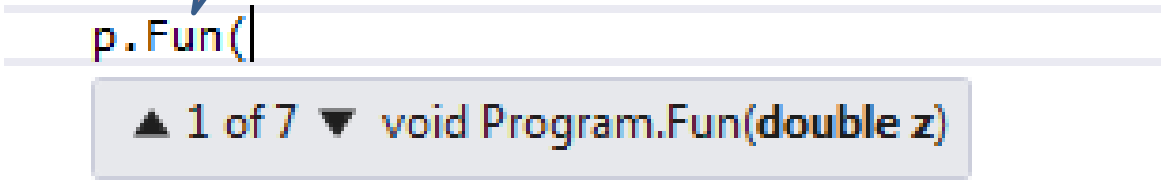
```
class Student
    {
        private string Name;
        private int Age;
        private char className;
        public Student(string name, int age , char cn)
        {
            Name = name;
            Age = age;
            className = cn;
        }
        public  Student()
        {
            Name = "";
            Age = 0;
            className = '#';
        }

        public void PrintInformation()
        {
         Console.WriteLine("name:{0},age:{1}, classname:{2}",Name,Age,className);
        }
    } //end of class
```

Overloading

- A member's signature includes its name and parameter list. Each member's signature must be unique within the type.
- **Members can have the same name** as long as their **parameter lists differ**.
- When two or more members in a type are the same kind of member (method, **constructor**, and so on) and have the same name and different parameter lists, the member is said to be overloaded

```
void Fun(int x)
  {
      ........
  }
   void Fun(string y)
  {
      ........
  }

   void Fun(double z)
  {
      ........
  }

   void Fun(int x, int y)
  {
      ........
  }
   void Fun(string y , int x)
  {
      ........
  }
   void Fun( int x , string y )
  {
      ........
  }
   void Fun(double z ,int x, int y)
  {
      ........
  }
```

When you try to call the method "**Fun**", **7** methods can be invoked with the same name.

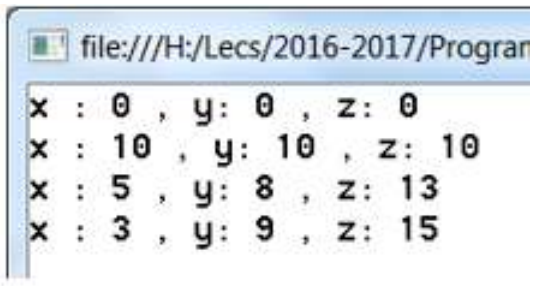


Constructor Overloading

- Similar to method overloading, we can also overload constructors.
- For constructor overloading, there must be two or more constructors with the same name but different:
  - Number of parameters
  - Types of parameters
  - Order of parameters

```
class Multiconst
    {
         int x,y,z;
        public Multiconst()
         {
             x = y = z = 0;
         }
        public Multiconst(int a)
        {
            x = y = z = a;
        }
        public Multiconst(int a, int b)
        {
            x = a;
            y = b;
            z = a+b;
        }
        public Multiconst(int a, int b , int c)
        {
            x=a;
            y=b;
            z=c;
        }

        public void Print()
        {
            Console.WriteLine("x : {0} , y: {1} , z: {2}", x, y, z);
        }
    } //end of class
```

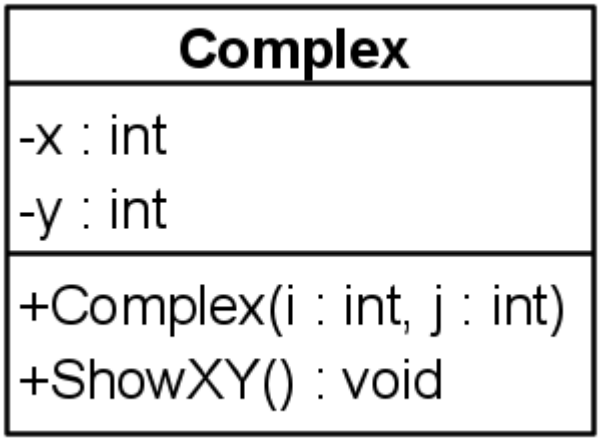
```
class Program
{
    static void Main(string[] args)
    {
        Multiconst M1 = new Multiconst();
        M1.Print();
        Multiconst M2 = new Multiconst(10);
        M2.Print();
        Multiconst M3 = new Multiconst(5,8);
        M3.Print();
        Multiconst M4 = new Multiconst(3,9,15);
        M4.Print();
        Console.ReadLine();
    }
}
```



Practical Examples

UML Class Diagram

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



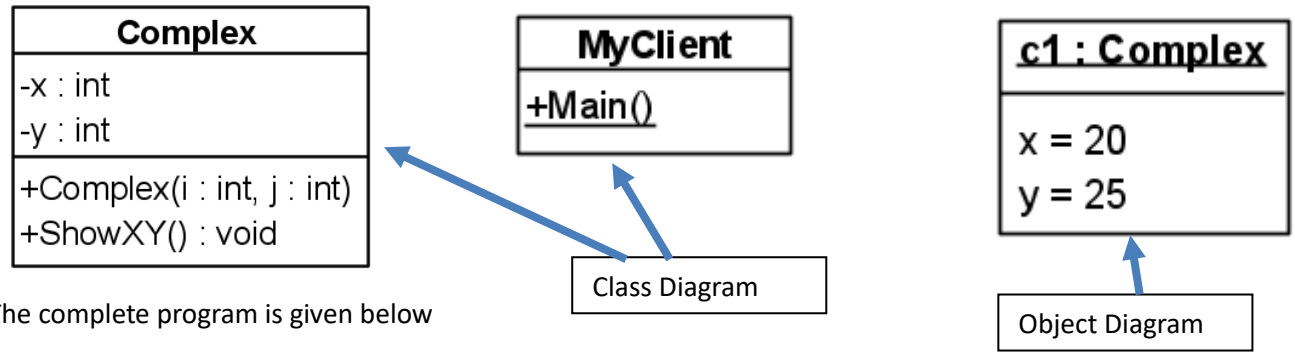

```
class Complex
{
    private int x;
    private int y;
    public Complex(int i, int j)
    {
        x = i;
        y = j;
    }
    public void ShowXY()
    {
        Console.WriteLine(x + "i+" + y);
    }
}
```

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

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

constructor

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.

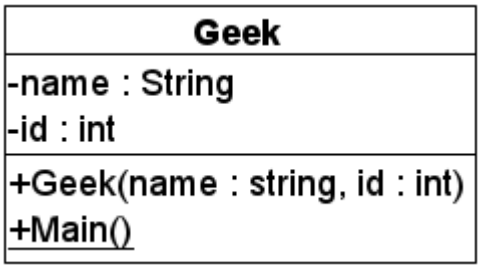


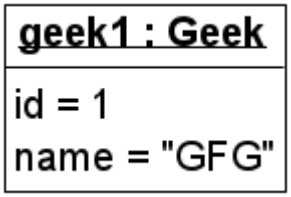
The complete program is given below

```
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();
    }
}
```

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.





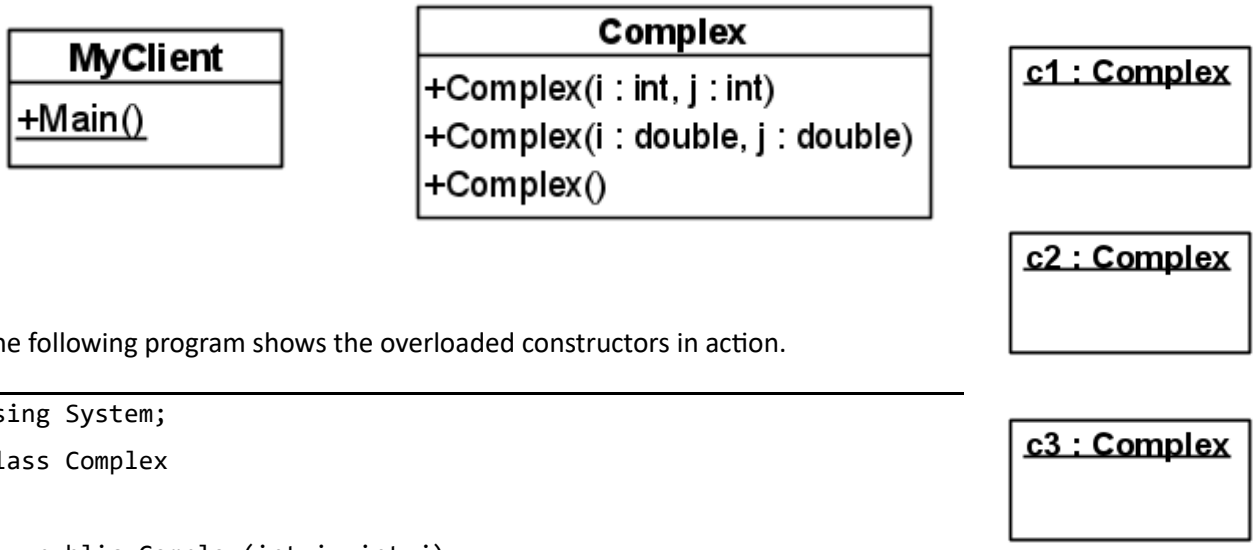

```
using System;
namespace PCExample
{

    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
```

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();
    }
}
```