**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 no body 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.
* class Welcome  
      {  
          // Default constructor  
          public Welcome()  
          {  
              Console.WriteLine("Welcome message from Default Constructor...");  
          }
* // Parametarized constructor   
          public Welcome(string name)  
          {  
              Console.WriteLine("\n\nThis message from parametarized constructor");  
              Console.WriteLine("Welcome to Constructor sample, by {0}", name);  
          }  
      }

Calling of constructor

static void Main(string[] args)  
        {  
            // Creating object for Welcome class  
            // This will called default constructor  
            Welcome obj = new Welcome();

            // Creating object for welcome class by passing parameter  
            // This will called parametarized constructor which matches  
            Welcome pObj = new Welcome("Bibekananda Panigrahi");

            Console.Read();  
        }  
    }

// C# constructor example  
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();  
}  
}

**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.

// C# constructor overloading  
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()  
{  
Complex c1 = new Complex(20,25);// displays 'constructor with 2 integer arguments'  
Complex c2 = new Complex(2.5,5.9); // displays 'constructor with 2 double arguments'  
Complex c3 = new Complex(); displays 'no argument constructor'  
}  
}

**Static Constructor**  
  
A static constructor has the same name as the class name but preceded with the static keyword; it will be called at the time of class load.

* No access specifier for static constructor.
* Static constructor will not return anything.
* Static constructor will accept only static members.
* Static constructor will call at the time of class loading.
* Static constructor will not allow overloading, so there is no parameterized static constructor.

 class Welcome  
    {  
        public static string Name = " Bibekanand Panigrahi ";

        // Static Constructor  
        static Welcome()  
        {  
            Console.WriteLine("Welcome message from static Constructor...");  
            Console.WriteLine("{0} name is coming from static member",Name);  
        }

        // Parametarized constructor   
        public Welcome(string name)  
        {  
            Console.WriteLine("\n\nThis message from parametarized constructor");  
            Console.WriteLine("Welcome to Constructor sample, by {0}", name);  
        }  
    }  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            // Creating object for welcome class by passing parameter  
            // This will called parametarized constructor which matches  
            Welcome pObj = new Welcome("Bibekanand Panigrahi");

            Console.Read();  
        }  
    }

In the above program we have created parameterized object for Welcome class, here the static constructor is called at the time of class load after then it's called corresponding parameterized constructor.

**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.

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();  
}

However the following program do not compile since it contain only private constructors

class Complex  
{  
private Complex()  
{  
Console.WriteLine("no argument constructor");   
}  
}

class MyClient  
{  
public static void Main()  
{  
Complex c3 = new Complex();  
}  
}  
public void Method1()  
{  
Console.WriteLine("Method of a non-abstract class");  
}  
}

**Constructor Chaining**

C# supports constructor overloading. In C#, even one constructor can invoke another constructor in the same class or in the base class of this class. This is what is known as constructor chaining.A special type of syntax is used for constructor chaining as follows.

class Complex  
{  
private Complex()  
{  
Console.Write("1");   
}  
private Complex(int x):this()  
{  
Console.Write("2");   
}  
public Complex(int x, int y):this(10)  
{  
Console.Write("3");   
}  
}  
class MyClient  
{  
public static void Main()  
{  
Complex c = new Complex(10,20); // Displays 123  
}

In the above program the Complex(int x, int y) invokes the Complex(int x) constructor by using a special syntax ':' this(arguments), which in turn invokes the Complex() constructor.

**Destructors**

* A class can have one destructor only.
* Destructors cannot be inherited or overloaded.
* Destructors are invoked automatically.
* Destructor can not have modifiers or parameters.
* When destructor is called, Finalize is called from destructor implicitly.

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.

In C# all classes are implicitly derived from the super base class object. The object class contains one special method, Finalize(), which every class can override. The Garbage Collection mechanism in .NET will call this method prior to the garbage collection of the objects this class. Remember when we provide a destructor in a class, during the compilation time, the compiler automatically generates the Finalize() method. That means that a destructor and overridden Finalize() method can't co-exist in a class. The following code will generate a compilation error because of the above program   
  
class Complex   
{  
~Complex()  
{  
}  
protected override void Finalize()  
{  
}  
}

Example of constructor and destructor

1. class **ShowMessage**
2. {
3. public **ShowMessage**()
4. {
5. **Console**.**WriteLine**("Hi, I am default message");
6. }
8. public **ShowMessage**(string message)
9. {
10. **Console**.**WriteLine**(message);
11. }
13. ~**ShowMessage**()
14. {
15. **Console**.**WriteLine**("I am destructor and I clean the resources and free up memory as soon as program closed");
16. **Console**.**ReadLine**();
17. }
18. }
19. class **Program**
20. {
21. static void **Main**(string[] args)
22. {
23. **ShowMessage** sm = new **ShowMessage**(); //Default constructor
24. **ShowMessage** sm1 = new **ShowMessage**("I am parameterized constructor. Press Enter to run Destructor"); //parameterized constructor
25. **Console**.**ReadLine**();
26. }
27. }