Introduction

Broadly speaking, a constructor is a method in the class which gets executed when its object is created. Usually, we put the initialization code in the constructor. Writing a constructor in the class is damn simple, have a look at the following sample:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class mySampleClass

{

public mySampleClass()

{

*// This is the constructor method.*

}

*// rest of the class members goes here.*

}

When the object of this class is instantiated, this constructor will be executed. Something like this:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

mySampleClass obj = new mySampleClass()

*// At this time the code in the constructor will // be executed*

Constructor Overloading

C# supports overloading of constructors, that means, we can have constructors with different sets of parameters. So, our class can be like this:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class mySampleClass

{

public mySampleClass()

{

*// This is the no parameter constructor method.*

*// First Constructor*

}

public mySampleClass(int Age)

{

*// This is the constructor with one parameter.*

*// Second Constructor*

}

public mySampleClass(int Age, string Name)

{

*// This is the constructor with two parameters.*

*// Third Constructor*

}

*// rest of the class members goes here.*

}

Well, note here that call to the constructor now depends on the way you instantiate the object. For example:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

mySampleClass obj = new mySampleClass()

*// At this time the code of no parameter*

*// constructor (First Constructor)will be executed*

mySampleClass obj = new mySampleClass(12)

*// At this time the code of one parameter*

*// constructor(Second Constructor)will be*

*// executed.*

The call to the constructors is completely governed by the rules of overloading here.

Calling Constructor from another Constructor

You can always make a call to one constructor from within another. Say, for example:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class mySampleClass

{

public mySampleClass(): this(10)

{

*// This is the no parameter constructor method.*

*// First Constructor*

}

public mySampleClass(int Age)

{

*// This is the constructor with one parameter.*

*// Second Constructor*

}

}

Very first of all, let us see what is this syntax:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public mySampleClass(): this(10)

Here, this refers to same class, so when we say this(10), we actually mean execute the public mySampleClass(int Age) method. The above way of calling the method is called initializer. We can have at the most one initializer in this way in the method.

Another thing which we must know is the execution sequence i.e., which method will be executed when. Here, if I instantiate the object as:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

mySampleClass obj = new mySampleClass()

Then the code of public mySampleClass(int Age) will be executed before the code of mySampleClass(). So, practically the definition of the method:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public mySampleClass(): this(10)

{

*// This is the no parameter constructor method.*

*// First Constructor*

}

is equivalent to:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public mySampleClass()

{

mySampleClass(10)

*// This is the no parameter constructor method.*

*// First Constructor*

}

**Note**: Above (just above this line) code is mentioned there for pure analogy and will not compile. The intention here is to tell the flow of execution if initializers are used.

We cannot make an explicit call to the constructors in C#, treating them as if any simple method, for example: statement mySampleClass(10) in the above code will not work. The only way you can call one constructor from another is through initializers.

For the VB.NET programmers: you can make the call to another constructor of the same class by the syntax Me.New(param list), but it should be the first line of your calling constructor method. So ultimately, the code of the called constructor runs prior to the code of the calling constructor, which is same as initializers here.

Note that only this and base (we will see it further) keywords are allowed in initializers, other method calls will raise an error.

This is sometimes called Constructor chaining.

Huff… Simple thing made tough, but this is how it is. Anyway, let us proceed further.

Behavior of Constructors in Inheritance

Let us first create the inherited class.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class myBaseClass

{

public myBaseClass()

{

*// Code for First Base class Constructor*

}

public myBaseClass(int Age)

{

*// Code for Second Base class Constructor*

}

*// Other class members goes here*

}

public class myDerivedClass : myBaseClass

*// Note that I am inheriting the class here.*

{

public myDerivedClass()

{

*// Code for the First myDerivedClass Constructor.*

}

public myDerivedClass(int Age):base(Age)

{

*// Code for the Second myDerivedClass Constructor.*

}

*// Other class members goes here*

}

Now, what will be the execution sequence here:

If I create the object of the derived class as:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

myDerivedClass obj = new myDerivedClass()

Then the sequence of execution will be:

1. public myBaseClass() method.
2. and then public myDerivedClass() method.

Note: If we do not provide initializer referring to the base class constructor then it executes the no parameter constructor of the base class.

Note one thing here: we are not making any explicit call to the constructor of base class neither by initializer nor by the base keyword, but it is still executing. This is the normal behavior of the constructor.

If I create an object of the derived class as:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

myDerivedClass obj = new myDerivedClass(15)

Then the sequence of execution will be:

1. public myBaseClass(int Age) method
2. and then public myDerivedClass(int Age) method

Here, the new keyword base has come into picture. This refers to the base class of the current class. So, here it refers to the myBaseClass. And base(10) refers to the call to myBaseClass(int Age) method.

Also note the usage of Age variable in the syntax: public myDerivedClass(int Age):base(Age). [Understanding it is left to the reader].

Private Constructors

Private constructors, the constructors with the "private" access modifier, are a bit special case. It is because we can neither create the object of the class, nor can we inherit the class with only private constructors. But yes, we can have the set of public constructors along with the private constructors in the class and the public constructors can access the private constructors from within the class through constructor chaining.

Say for example, my class is something like this :

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class myClass

{

private MyClass()

{

Console.WriteLine("This is no parameter Constructor");

}

public MyClass(int var):this()

{

Console.WriteLine("This is one parameter Constructor");

}

*// Other class methods goes here*

}

Then we can create the object of this class by the statement:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

MyClass obj = new MyClass(10);

The above statement will work fine, but the statement

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

MyClass obj = new MyClass();

will raise an error : *'Constructors.MyClass.MyClass()' is inaccessible due to its protection level*

It is possible to have the class with only the private constructors. But yes as I said, such class can neither be instantiated nor be inherited. If we try to inherit the class with only private constructors then we will get the same error as above. Also recall, once you provide constructor from your side the compiler will not add the no-parameter public constructor to your class.

Well, one of the usage scenarios of such class could be – when you have only static members in the class and you don't need to instantiate it.

Phew… lost… Anything left in constructors? Yes, Static Constructors. Ha!! Now, what are they? Let us see..

Static Constructors

This is a new concept introduced in C#. By new here, I mean that it was not available for the C++ developers. This is a special constructor and gets called before the first object is created of the class. The time of execution cannot be determined, but it is definitely before the first object creation - could be at the time of loading the assembly.

The syntax of writing the static constructors is also damn simple. Here it is:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class myClass

{

static myClass()

{

*// Initialization code goes here.*

*// Can only access static members here.*

}

*// Other class methods goes here*

}

Notes for Static Constructors:

1. There can be only one static constructor in the class.
2. The static constructor should be without parameters.
3. It can only access the static members of the class.
4. There should be no access modifier in static constructor definition.

Ok fine, all the above points are fine, but why is it like that? Let us go step by step here.

Firstly, the call to the static method is made by the CLR and not by the object, so we do not need to have the access modifier to it.

Secondly, it is going to be called by CLR, who can pass the parameters to it, if required. So we cannot have parameterized static constructor.

Thirdly, non-static members in the class are specific to the object instance. So static constructor, if allowed to work on non-static members, will reflect the changes in all the object instances, which is impractical. So static constructor can access only static members of the class.

Fourthly, overloading needs the two methods to be different in terms of methods definition, which you cannot do with Static Constructors, so you can have at the most one static constructor in the class.

Now, one question raises here, can we have two constructors as:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/7011/An-Intro-to-Constructors-in-C)

public class myClass

{

static myClass()

{

*// Initialization code goes here.*

*// Can only access static members here.*

}

public myClass()

{

*// Code for the First myDerivedClass Constructor.*

}

*// Other class methods goes here*

}

This is perfectly valid, though doesn't seem to be in accordance with overloading concepts. But why? Because the time of execution of the two methods are different. One is at the time of loading the assembly and one is at the time of object creation.

Constructors FAQs

1. Is the constructor mandatory for a class?

Yes, it is mandatory to have the constructor in the class and that too should be accessible for the object i.e., it should have a proper access modifier. Say, for example, we have only private constructor(s) in the class and if we are interested in instantiating the class, i.e., want to create an object of the class, then having only private constructor will not be sufficient and in fact it will raise an error. So, proper access modifies should be provided to the constructors.

1. What if I do not write the constructor?

In such case, the compiler will try to supply the no parameter constructor for your class, behind the scene. Compiler will attempt this only if you do not write the constructor for the class. If you provide any constructor (with or without parameters), then compiler will not make any such attempt.

1. What if I have the constructor public myDerivedClass(), but not the public myBaseClass()?

It will raise an error. If either the no parameter constructor is absent or it is in-accessible (say it is private), it will raise an error. You will have to take the precaution here.

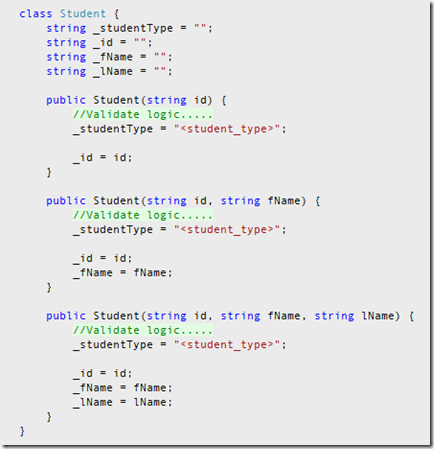
1. Can we access static members from the non-static (normal) constructors?

Yes, we can. There is no such restriction on non-static constructors. But there is one on static constructors that it can access only static members.

**What is Constructor Chaining?**

Constructor Chaining is an approach where a constructor calls another constructor in the same or base class.

This is very handy when we have a class that defines multiple constructors. Assume we are developing a class Student. And this class has three constructors. On each constructer we have to validate the student's ID and categorize him/her. So if we do not use the constructor chaining approach, it would be something similar to what is shown below:

[](http://lh5.ggpht.com/_zprglM1x5x4/TQB0_UpDl7I/AAAAAAAAANU/f-qvk3CVT_Y/s1600-h/screen_01%5B4%5D.png)

Even though the above approach solves our problem, it duplicates code. (We are assigning a value to ‘\_id’ in all our constructors). This is where constructor chaining is very useful. It will eliminate this problem. This time we only assign values in one constructor which consists of the most number of parameters. And we call that constructor when the other two constructers are called.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/271582/Constructor-Chaining-in-Csharp)

class Student {

string \_studentType = "";

string \_id = "";

string \_fName = "";

string \_lName = "";

public Student(string id)

: this(id, "", "") {

}

public Student(string id, string fName)

: this(id, fName, "") {

}

public Student(string id, string fName, string lName) {

*//Validate logic.....*

\_studentType = "<student\_type>";

\_id = id;

\_fName = fName;

\_lName = lName;

}

}

**\*\*Please note**: If you do not specify anything [in this example, we used ‘this’), it will be considered that we are calling the constructor on the base class. And it’s similar to using ‘: base(…)’].

using System;

public class A // This is the base class.

{

public A(int value)

{

// Executes some code in the constructor.

Console.WriteLine("Base constructor A()");

}

}

public class B : A // This class derives from the previous class.

{

public B(int value)

: base(value)

{

// The base constructor is called first.

// ... Then this code is executed.

Console.WriteLine("Derived constructor B()");

}

}

class Program

{

static void Main()

{

// Create a new instance of class A, which is the base class.

// ... Then create an instance of B, which executes the base constructor.

A a = new A(0);

B b = new B(1);

}

}

**Output**

Base constructor A()

Base constructor A()

Derived constructor B()

