**Constructor In Inheritance**

* A constructor is a method with the same name as the class name and is invoked automatically when a new instance of a class is created.
* Constructors of both classes must be executed when the object of child class is created.
* Derived Class’s constructor invokes constructor of Base class.
* Explicit call to the super class constructor from sub class’s can be made using **base()**.
* If u don’t write **base()** explicitly then C# compiler implicitly write the **base()**.
* If base Class have parameterized constructor then you can add parameters in base().

**Base Keyword**

**The base keyword allows you to do the following:**

* Access the variables and methods of the base class from the derived class.
* Re-declare the methods and variables defined in the base class.
* Invoke the derived class data members.
* Access the base class members using the base keyword.

**The following syntax shows the use of the base keyword:**

class <ClassName>

{

<accessmodifier><returntype><BaseMethod>{}

}

class <ClassName1>:<ClassName>

{

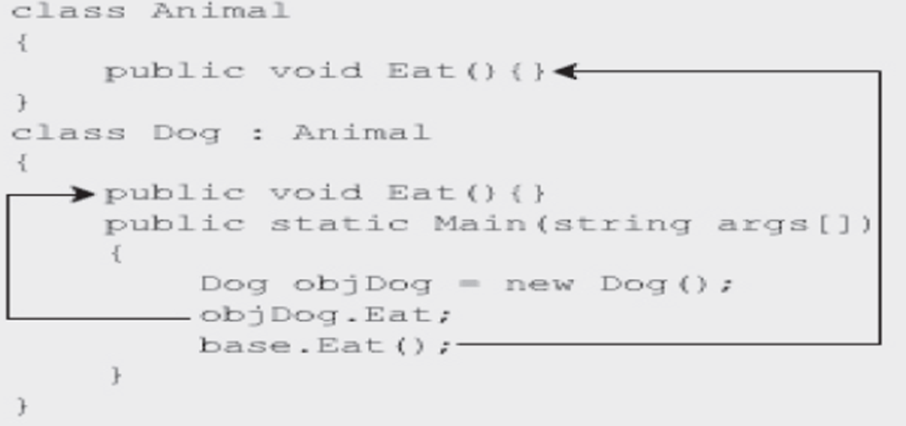
base.<BaseMethod>;

}

**where,**

* <**ClassName**>: Is the name of the base class.
* <**accessmodifier**>: Specifies the scope of the class or method.> : Specifies the scope of the class or method.
* <**returntype**>: Specifies the type of data the method will return.
* <**BaseMethod**>: Is the base class method.
* <**ClassName1**>: Is the name of the derived class.
* **base**: Is a keyword used to access the base class members.

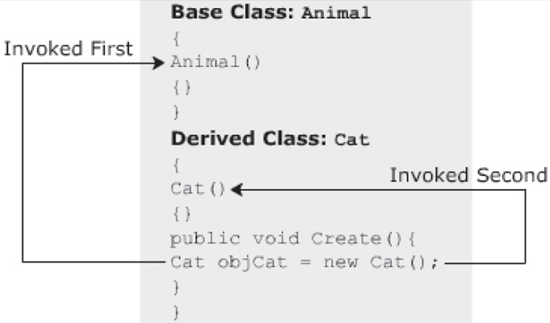
**The following figure displays an example of using the base keyword:**



**Constructor In Inheritance**

* In C# you can invoke the base class constructor by either instantiating the derived class or the base class.
* In C# you can invoke the constructor of the base class followed by the constructor of the derived class.
* In C# you can invoke the base class constructor by using the base keyword in the derived class constructor declaration.
* In C# you can pass parameters to the constructor.
* However, C# cannot inherit constructors similar to how you inherit methods.

**The following figure displays an example of constructor inheritance:**



**The following code explicitly invokes the base class constructor using the base keyword:**

class Animal

{

public Animal()

{

Console.WriteLine("Animal constructor without parameters");

}

public Animal(Stringname)

{

Console.WriteLine("Animal constructor with a string parameter");

}

}

class Canine:Animal

{

//base()takes a string value called“Lion”

public Canine():base("Lion")

{

Console.WriteLine("DerivedCanine");

}

}

class Details

{

static voidMain(String[]args)

{

Canine objCanine=new Canine();

}

}

**In Above code,**

* The class **Animal** consists of two constructors, one without a parameter and the other with a string parameter.
* The class **Canine** is inherited from the class **Animal**.
* The derived class **Canine** consists of a constructor that invokes the constructor of the base class Animal by using the base keyword.
* If the base keyword does not take a string in the parenthesis, the constructor of the class **Animal** that does not contain parameters is invoked.
* In the class **Details**, when the derived class constructor is invoked, it will in turn invoke the parameterized constructor of the base class.

**Output**

Animal constructor with a string parameter  
Derived Canine

**Invoking Parameterized Base Class Constructors**

* The derived class constructor can explicitly invoke the base class constructor by using the base keyword.
* If a base class constructor has a parameter, the base keyword is followed by the value of the type specified in the constructor declaration.
* If there are no parameters, the base keyword is followed by a pair of parentheses.

**The following code demonstrates how parameterized constructors are invoked in a multi-level hierarchy:**

using System;

class Metals

{

string\_metalType;

public Metals(stringtype)

{

\_metalType=type;

Console.WriteLine("Metal:\t\t"+\_metalType);

}

}

class SteelCompany : Metals

{

string\_grade;

public SteelCompany(stringgrade):base("Steel")

{

\_grade=grade;

Console.WriteLine("Grade:\t\t"+\_grade);

}

}

class Automobiles:SteelCompany

{

string\_part;

public Automobiles(stringpart):base("CastIron")

{

\_part=part;

Console.WriteLine("Part:\t\t"+\_part);

}

static voidMain(string[]args)

{

Automobiles objAutomobiles=new Automobiles("Chassies");

}

}

**In Above code,**

* The **Automobiles** class inherits the **SteelCompany** class.
* The **SteelCompany** class inherits the **Metals** class.
* In the Main()method, when an instance of the **Automobiles** class is created, it invokes the constructor of the **Metals** class, followed by the constructor of the **SteelCompany** class.
* Finally, the constructor of the **Automobiles** class is invoked.

**Output**

Metal: Steel  
Grade: CastIron  
Part: Chassies

**Source Code Of Constructor In Inheritance**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Constructor\_In\_Inheritance

{

class BaseClass

{

public BaseClass(string message)

{

Console.WriteLine(message);

}

public BaseClass(int i)

{

Console.WriteLine(i);

}

}

class DerivedClass : BaseClass

{

public DerivedClass() : base(25)

{

Console.WriteLine("this is a constructor of Derived class !!");

}

}

class Program

{

static void Main(string[] args)

{

DerivedClass dc = new DerivedClass();

Console.ReadLine();

}

}

}