**Types Of Constructors In C#**

1. Default Constructor
2. Parameterized Constructor
3. Static Constructor
4. Copy Constructor
5. Private Constructor

**Default Constructors**

* A constructor which has not defined any parameters or we can say without any parameters is called default constructor. It initializes the same value of every instance of class.
* C# creates a default constructor for a class if no constructor is specified within the class.
* The default constructor automatically initializes all the numeric data type instance variables of the class to zero.
* If you define a constructor in the class, the default constructor is no longer used.

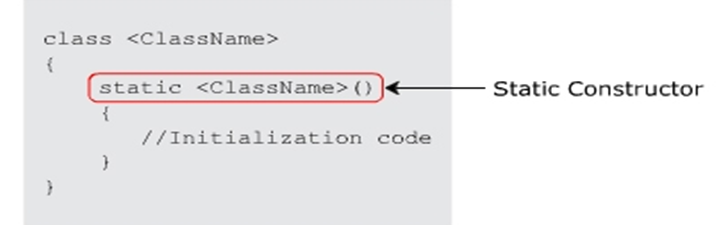
**Parameterized Constructor**

A constructor which has at least one parameter is called Parameterized Constructor. Using this type of constructor we can initialize each instance of the class to different values.

**Static Constructor**

* A static constructor is used to initialize static variables of the class and to perform a particular action only once.
* It is invoked before any static member of the class is accessed.
* A static constructor does not take any parameters and does not use any access modifiers because it is invoked directly by the CLR instead of the object.

**The following figure illustrates the syntax for a static constructor:**



**The following code shows how static constructors are created and invoked.**

using System;

class Multiplication

{

staticint \_valueOne = 10;

staticint \_product;

static Multiplication()

{

Console.WriteLine(“Static Constructor initialized”);

\_product = \_valueOne \* \_valueOne;

}

public static void Method()

{

Console.WriteLine(“Value of product = “ + \_product);

}

static void Main(string[] args)

{

Multiplication.Method();

}

}

**In Above Code,**

* The static constructor Multiplication() is used to initialize the static variable \_product.
* Here, the static constructor is invoked before the static method Method() is called from the Main() method.

**Output**

Static Constructor initialized  
Value of product = 100

**Copy Constructor**

* The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance to the values of an existing instance.
* In c#, Copy Constructor is a parameterized constructor which contains a parameter of same class type. The copy constructor in C# is useful whenever we want to initialize a new instance to the values of an existing instance.
* In simple words, we can say copy constructor is a constructor which copies a data of one object into another object.

**Private Constructor In C#**

When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class. They are usually used in classes that contain static members only. Some key points of a private constructor are:

* One use of a private constructor is when we have only static members.
* Once we provide a constructor that is either private or public or any, the compiler will not add the parameter-less public constructor to the class.
* In the presence of parameterless private constructor you cannot create a default constructor.
* We cannot inherit the class in which we have a private constructor.
* We can have parameters in private constructor. YES

**Source Code Of Static Constructor**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace STATIC\_CONSTRUCTOR

{

class person

{

public static string PersonName;

public static int PersonAge;

static person()

{

PersonName = "Ali";

PersonAge = 23;

Console.WriteLine("Static constructor invoked !!");

}

public person()

{

Console.WriteLine("default constructor invoked !!");

}

public static void GetDetails()

{

Console.WriteLine("Person name is: {0}", PersonName);

Console.WriteLine("Person Age is: {0}", PersonAge);

}

}

class Program

{

static void Main(string[] args)

{

person umar = new person();

person Ali = new person();

Console.ReadLine();

}

}

}

**Source Code Of Parameterized Constructor In C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConstructorsInCsharp

{

class Employees

{

int EmpId;

string EmpName;

int EmpAge;

public Employees(int EmpId, string EmpName, int EmpAge)

{

this.EmpId = EmpId;

this.EmpName = EmpName;

this.EmpAge = EmpAge;

}

public int getId()

{

return this.EmpId;

}

public string getName()

{

return this.EmpName;

}

public int getAge()

{

return this.EmpAge;

}

static void Main(string[] args)

{

Employees Ali = new Employees(11,"Ali Khan",22);

Employees Zain = new Employees(12, "Zain Ali", 21);

Console.WriteLine("Employee Id is {0}",Ali.getId());

Console.WriteLine("Employee Name is {0}", Ali.getName());

Console.WriteLine("Employee Age is {0}", Ali.getAge());

Console.WriteLine("--------------------");

Console.WriteLine("Employee Id is {0}", Zain.getId());

Console.WriteLine("Employee Name is {0}", Zain.getName());

Console.WriteLine("Employee Age is {0}", Zain.getAge());

Console.ReadLine();

}

}

}

**Source Code Of Copy Constructor In C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CopYConstructor

{

class Example

{

string name;

int age;

public Example(string name, int age)

{

this.name = name;

this.age = age;

}

public Example(Example e) // copy constructor

{

this.name = e.name;

this.age = e.age;

}

public void getData()

{

Console.WriteLine("Name is: {0}", name);

Console.WriteLine("Age is: {0}", age);

}

}

class Program

{

static void Main(string[] args)

{

Example obj = new Example("Adil",25);

obj.getData();

Example obj1 = new Example(obj);

obj1.getData();

Console.ReadLine();

}

}

}

**Source Code Of Private Constructor In C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PrivateConstructor

{

class Example

{

public static int a;

private Example()

{

}

public static int getIncrement()

{

return ++a;

}

public static void getTime()

{

Console.WriteLine(DateTime.Now);

}

}

//class Example2 : Example

//{

//}

class Program

{

static void Main(string[] args)

{

//Example e = new Example("Ali","Khan");

//Example e = new Example();

//Example.getTime();

//Example.a = 20;

//Console.WriteLine(Example.getIncrement());

Console.ReadLine();

}

}

}