C Sharp-C# is a computer programming language created by Microsoft Corporation for their .NET Framework. C# is a typesafe rich implementation of modern object-oriented language which includes abstraction, encapsulation, inheritance and polymorphism that enable developers to quickly create a wide range of applications for the .NET platform.

Features of c#

1. C# is a modern language.
2. C# is type safe language.
3. C# is an inter operability language.- C# can directly access components from vb.net and from other managed code languages runs on CLR.For example if we write a program in VB.NET that program we can access it in C# too by using DLL of vb.net program.
4. C# versionable-.NET framework handles the versioning. C# code which runs on .NET framework so it is a versioning language. We can have different versions of the same component at the same time and our applications will know automatically which version to use and what version not to use. Versioning is done only on assemblies with strong names.

Typesafe- Dot net don't allow the methods to access other methods private fields is called type safe.

Why Typesafe Importatnt-

Type-safety is important for assembly isolation and security enforcement. When code is type- safe, the common language runtime can completely isolate assemblies from each other. This isolation helps ensure that assemblies cannot affect each other and it increases application reliability.

**How Type Safety Ensured**

When a code runs on CLR (Common Language Runtime) it performs the type safe check called type safe verification and this is done during Just in Time Compilation (JIT) by a tool called **peverify.exe**.

New feature in c# 2.0-

1. Generic
2. Iterators
3. Partial class
4. Nullable types
5. Anonymous method
6. Covariance and contravariance in delegate
7. Friend assembly
8. Volatile
9. Static class
10. Property accessor accessibility

New feature in c# 3.5-

1. Implicitly type variable(var)
2. Auto properties
3. Object initialization
4. Partial methods
5. EXTENSION METHOD
6. Linq
7. Lambda expression
8. Anonymous type
9. Query Expression
10. Expression Tree
11. Collection Initialization.

New feature in c# 4.0-

* Dynamic programming
* Named and optional parameter
* Covariance and contravariance generic type parameter.

Variables-Variables represent storage locations. Every variable has a type that determines what values can be stored in the variable.

Syntax-<data\_type> <variable\_list>;

String-The String type (System.String) stores text values as a sequence of char (System.Char) elements that represent Unicode characters. Strings in .NET are **immutable so i**ts indexer is read-only,! This simply means that once created, strings cannot be modified (without reflection or unsafe code), and the methods that apparently modify a string, really return a new object with the desired value. It has following characterstics-

1. It is a reference type
2. It's immutable Real example of Immutability of strings has positive significance in **multithreaded applications** – any text amendment causes creation of a new variable so there is no need to set up the lock to avoid conflicts while multiple threads simultaneously access text.
3. It can contain nulls
4. It overloads the == operator

Stringbuilder-The [System.Text.StringBuilder](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder(v=vs.110).aspx) class can be used when you want to modify a string without creating a new object. Using the [StringBuilder](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder(v=vs.110).aspx) class can boost performance when concatenating many strings together in a loop. StringBuilder indexer is readable/writeable.

|  |  |
| --- | --- |
| **String** | **StringBuilder** |
| System.String is immutable | System.StringBuilder is mutable |
| Concatenation is used to combine two strings | Append method is used. |
| The first string is combined to the other string by creating a new copy in the memory as a string object, and then the old string is deleted | Insertion is done on the existing string. |
| String is efficient for small string manipulation | StringBuilder is more efficient in case large amounts of string manipulations have to be performed |
|  |  |
|  |  |

**1. Performance of StringBuilder is high**

Main difference-I used String and StringBuilder, the end result is same, but performance issues are there with String class. String always create new object of string type in memory when the concatenation happens while when you use string builder class, it updates string at one place in memory and does not create new space in memory. So in this way StringBuilder consumes less memory and delivers high speed. So we can say that StringBuilder is more efficient for operations that manipulate strings.

**2. StringBuilder is a Mutable Class while String is Immutable**

**StringBuilder is mutable** means that you can insert, append, replace etc. without creating a new StringBuilder each time.

**String is immutable** means that each time you perform an operation like insert, append, replace etc. that "changes" it, you are creating a new string to replace the old one. You create a string you can never change it, rather it will create a new string to store the new value, this can be inefficient if you need to change the value of a string variable a lot.

Char-

Array-In C#, an array index starts at zero. That means the first item of an array starts at the 0thposition. The position of the last item on an array will total number of items - 1. So if an array has 10 items, the last 10th item is at 9th position.   
  
In C#, arrays can be declared as fixed length or dynamic.

A *fixed length* array can store a predefined number of items.

A *dynamic array* does not have a predefined size. The size of a *dynamic array* increases as you add new items to the array. You can declare an array of fixed length or dynamic. You can even change a dynamic array to static after it is defined.

Let's take a look at simple declarations of arrays in C#. The following code snippet defines the simplest dynamic array of integer types that does not have a fixed size.

int[] intArray;

*As you can see from the above code snippet, the declaration of an array starts with a type of array followed by a square bracket ([]) and name of the array.*

The following code snippet declares an array that can store 5 items onlystarting from index 0 to 4.

int[] intArray;

intArray = new int[5];

The following code snippet declares an array that can store 100 items starting from index 0 to 99.

int[] intArray;

intArray = new int[100];

**Array Types**

Arrays can be divided into the following four categories.

         Single-dimensional arrays

         Multidimensional arrays or rectangular arrays

         Jagged arrays

         Mixed arrays.

**Single Dimension Arrays**  
Single-dimensional arrays are the simplest form of arrays. These types of arrays are used to store number of items of a predefined type. All items in a single dimension array are stored contiguously starting from 0 to the size of the array -1.

The following code declares an integer array that can store 3 items. As you can see from the code, first I declare the array using [] bracket and after that I instantiate the array by calling the new operator.

int[] intArray;

intArray = new int[3];

Array declarations in C# are pretty simple. You put array items in curly braces ({}). If an array is not initialized, its items are automatically initialized to the default initial value for the array type if the array is not initialized at the time it is declared.

The following code declares and initializes an array of three items of integer type.

int[] staticIntArray = new int[3] {1, 3, 5};

The following code declares and initializes an array of 5 string items.

string[] strArray = new string[5] { "Mahesh", "Mike", "Raj", "Praveen", "Dinesh" };

You can even directly assign these values without using the new operator.

string[] strArray = { "Mahesh", "Mike", "Raj", "Praveen", "Dinesh" };

You can initialize a dynamic length array as follows:

string[] strArray = new string[] { "Mahesh", "Mike", "Raj", "Praveen", "Dinesh" };

**Multi-Dimensional Arrays**

A multi-dimensional array, also known as a rectangular array is an array with more than one dimension. The form of a multi-dimensional array is a matrix.

**Declaring a multi-dimensional array**

A multi dimension array is declared as following:

string[,] mutliDimStringArray;

A multi-dimensional array can be fixed-sized or dynamic sized.

**Initializing multi-dimensional arrays**

The following code snippet is an example of fixed-sized multi-dimensional arrays that defines two multi dimension arrays with a matrix of 3x2 and 2x2. The first array can store 6 items and second array can store 4 items. Both of these arrays are initialized during the declaration.

int[,] numbers = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

string[,] names = new string[2, 2] { { "Rosy", "Amy" }, { "Peter", "Albert" } };

Now let's see examples of multi-dimensional dynamic arrays where you are not sure of the number of items of the array. The following code snippet creates two multi-dimensional arrays with no limit.

int[,] numbers = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

string[,] names = new string[,] { { "Rosy", "Amy" }, { "Peter", "Albert" } };

You can also omit the new operator as we did in single dimension arrays. You can assign these values directly without using the new operator. For example:  
  
int[,] numbers = { { 1, 2 }, { 3, 4 }, { 5, 6 } };

string[,] names = { { "Rosy", "Amy" }, { "Peter", "Albert" } };

We can also initialize the array items one item at a time. The following code snippet is an example of initializing array items one at a time.

int[,] numbers = new int[3, 2];

numbers[0, 0] = 1;

numbers[1, 0] = 2;

numbers[2, 0] = 3;

numbers[0, 1] = 4;

numbers[1, 1] = 5;

numbers[2, 1] = 6;

**Accessing multi-dimensional arrays**

A multi-dimensional array items are represented in a matrix format and to access it's items, we need to specify the matrix dimension. For example, item(1,2) represents an array item in the matrix at second row and third column.

The following code snippet shows how to access numbers array defined in the above code.

Console.WriteLine(numbers[0,0]);

Console.WriteLine(numbers[0, 1]);

Console.WriteLine(numbers[1, 0]);

Console.WriteLine(numbers[1, 1]);

Console.WriteLine(numbers[2, 0]);

Console.WriteLine(numbers[2, 2]);

**Jagged Arrays**

Jagged arrays are arrays of arrays. The elements of a jagged array are other arrays.

**Declaring Jagged Arrays**

Declaration of a jagged array involves two brackets. For example, the following code snippet declares a jagged array that has three items of an array.

int[][] intJaggedArray = new int[3][];

The following code snippet declares a jagged array that has two items of an array.

string[][] stringJaggedArray = new string[2][];

**Initializing Jagged Arrays**

Before a jagged array can be used, its items must be initialized. The following code snippet initializes a jagged array; the first item with an array of integers that has two integers, second item with an array of integers that has 4 integers, and a third item with an array of integers that has 6 integers.

// Initializing jagged arrays

intJaggedArray[0] = new int[2];

intJaggedArray[1] = new int[4];

intJaggedArray[2] = new int[6];

We can also initialize a jagged array's items by providing the values of the array's items. The following code snippet initializes item an array's items directly during the declaration.

// Initializing jagged arrays

intJaggedArray[0] = new int[2]{2, 12};

intJaggedArray[1] = new int[4]{4, 14, 24, 34};

intJaggedArray[2] = new int[6] {6, 16, 26, 36, 46, 56 };

**Accessing Jagged Arrays**

We can access a jagged array's items individually in the following way:

Console.Write(intJaggedArray3[0][0]);

Console.WriteLine(intJaggedArray3[2][5]);

We can also loop through all of the items of a jagged array. The Length property of an array helps a lot; it gives us the number of items in an array. The following code snippet loops through all of the items of a jagged array and displays them on the screen.

// Loop through all itesm of a jagged array

for (int i = 0; i < intJaggedArray3.Length; i++)

{

    System.Console.Write("Element({0}): ", i);

    for (int j = 0; j < intJaggedArray3[i].Length; j++)

    {

        System.Console.Write("{0}{1}", intJaggedArray3[i][j], j == (intJaggedArray3[i].Length - 1) ? "" : " ");

    }

    System.Console.WriteLine();

}

**Mixed Arrays**

Mixed arrays are a combination of multi-dimension arrays and jagged arrays. The mixed arrays type is removed from .NET 4.0. I have not really seen any use of mixed arrays. You can do anything you want with the help of multi-dimensional and jagged arrays.

**Simple example**

Console.WriteLine("Single Dimension Array Sample");

// Single dim array

string[] strArray = new string[] { "Mahesh Chand", "Mike Gold", "Raj Beniwal", "Praveen Kumar", "Dinesh Beniwal" };

// Read array items using foreach loop

foreach (string str in strArray)

{

    Console.WriteLine(str);

}

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

Console.WriteLine("Multi-Dimension Array Sample");

string[,] string2DArray = new string[2, 2] { { "Rosy", "Amy" }, { "Peter", "Albert" } };

foreach (string str in string2DArray)

{

    Console.WriteLine(str);

}

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

Console.WriteLine("Jagged Array Sample");

int[][] intJaggedArray3 =

{

    new int[] {2,12},

    new int[] {14, 14, 24, 34},

    new int[] {6, 16, 26, 36, 46, 56}

};

// Loop through all itesm of a jagged array

for (int i = 0; i < intJaggedArray3.Length; i++)

{

    Console.Write("Element({0}): ", i);

    for (int j = 0; j < intJaggedArray3[i].Length; j++)

    {

        Console.Write("{0}{1}", intJaggedArray3[i][j], j == (intJaggedArray3[i].Length - 1) ?"" : " ");

    }

    Console.WriteLine();

}

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

Array class-Array class is the mother of all arrays and provides functionality for creating, manipulating, searching, and sorting arrays in .NET Framework. **Array** class, defined in the System namespace, is the base class for arrays in C#. Array class is an abstract base class that means we cannot create an instance of the Array class.

Creating an array-

Array stringArray = Array.CreateInstance(typeof(String), 3);

stringArray.SetValue("Mahesh Chand", 0);

stringArray.SetValue("Raj Kumar", 1);

stringArray.SetValue("Neel Beniwal", 2);

The code snippet in Listing 2 creates a multi-dimensional array.

Array intArray3D = Array.CreateInstance(typeof(Int32), 2, 3, 4);

for (int i = intArray3D.GetLowerBound(0); i <= intArray3D.GetUpperBound(0); i++)

    for (int j = intArray3D.GetLowerBound(1); j <= intArray3D.GetUpperBound(1); j++)

        for (int k = intArray3D.GetLowerBound(2); k <= intArray3D.GetUpperBound(2); k++)

        {

            intArray3D.SetValue((i \* 100) + (j \* 10) + k, i, j, k);

        }

foreach (int ival in intArray3D)

{

    Console.WriteLine(ival);

}

Good article

<http://www.c-sharpcorner.com/UploadFile/mahesh/WorkingWithArrays11232005064036AM/WorkingWithArrays.aspx>

<http://www.codeproject.com/Articles/161465/Arrays-Basics-in-CSharpDotNetTech>

checked-The **checked** keyword is used to explicitly enable overflow checking for integral-type arithmetic operations and conversions.

Unchecked-The **unchecked** keyword is used to suppress overflow-checking for integral-type arithmetic operations and conversions.

The stack and the heap are the places where variables and constants reside.

Stack-The stack is a block of memory for storing local variables and parameters. The stack logically grows and shrinks as a function is entered and exited.

Heap-The heap is a block of memory in which objects (i.e., reference-type instances) reside. Whenever a new object is created, it is allocated on the heap, and a reference to that object is returned. During a program’s execution, the heap starts filling up as new objects are created. The runtime has a garbage collector that periodically deallocates objects from the heap, so your computer does not run out of memory

Value-type instances (and object references) live wherever the variable was declared. If the instance was declared as a field within an object, or as an array element, that instance lives on the heap. You can’t explicitly delete objects in C#.The heap also stores static fields and constants. Unlike objects allocated on the heap (which can get garbage-collected), these live until the application domain is torn down.

Field-A fieldis a variable that is a member of a class or struct

.Net Framework-The .net framework is a software framework developed by Microsoft that run on Microsoft windows. It include large library and provide language interoperability across several programming language. It consist two component-

1. Common language runtime

2. Framework class library.

1. CLR-.Net framework provide runtime environment called common language runtime. Itprovide an environment to run all the .net programs. Thecode which runs under the clr is called as managed code. The common language runtime is a virtual machine component of Microsoft .net framework and is responsible for managing the execution of .net programs. In a process known as just in time compilation the compiler code is converted into machine instruction that is converted into machine instruction that in turn are execution by computer cpu. The clr provides addition services including memory, thread execution, code execution, code safety verification, compilation, and other system services.

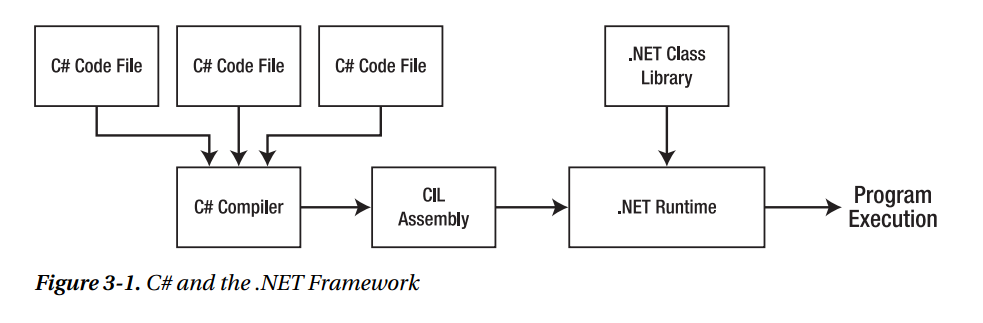
2. FCL-The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. Framework class library is a standard library of Microsoft .net framework. The FCL is a collection of reusable classes, interfaces and value type. The base class library is a part of FCL which provide system.codedome, system. Collection, system.diagnostices etc.

CLS-CLS stands for Common Language Specifications. It is a subset of CTS. CLS is a set of rules or guidelines which if followed ensures that code written in one .NET language can be used by another .NET language. For example one rule is that we cannot have member functions with same name with case difference only i.e we should not have add() and Add(). This may work in C# because it is case-sensitive but if try to use that C# code in VB.NET, it is not possible because VB.NET is not case-sensitive.

JIT-**:**JIT stands for Just In Timer Compiler. It is the internal compiler of .NET which takes IL code from CLR and executes it to machine specific instructions.

**IL Code:**(Microsoft Intermediate language) or IL(Intermediate Language) is machine independent code generated by .NET framework after the compilation of program written in any language by you. The word IL Code stands for Intermediate Language Code. It is a CPU independent partially compiled code. When we develop our .NET application, we don’t know in what kind of environment our code will run i.e. on which operating system it will be finally hosted, what will be the CPU configuration, etc. So for this purpose we have IL code which is compiled according to machine configuration. IL code is given by Language compiler which is different for different languages for example csc.exe compiler for C#, vbc.exe compiler for VB.NET, etc.

Working of c# and .net



CIL-This article helps you in understanding and getting started with the Common Intermediate Lanaguage (CIL).  SinceCIL is also a programming language we can very well program directly in IL and is not very difficult as most of us initially think.Understanding CIL is very important in writing dynamic assembly, allows developer to do changes in assembly by changing the IL code directly even if source code is not available and above all it gives clear understanding of internals of .net languages.

Background

When we compile our code written in any .net language the associated compiler (like C#,VB Compiler) generates binaries called assembly which contains IL code .These Instructions are low level human readable language which can be converted into machine language by the run time compiler during its first execution.It is done just during execution so that the compiler will have before hand knowledge of which environment it’s going to run so that it can emit the optimized machine language code targeting that platform. The .net Framework’s such compiler is called Just-in-Time (JIT) compiler or *Jitter*.    
  
A .net assembly consist of following elements:  
  
1. A Win32 file Header - The header data identifies the kind of application (console,Windows,code lib ) to be hosted by Windows operating system.  
  
2.  A CLR file Header- The CLR header is a block of data that provides information which allow it to be hosted by the CLR. CLR header contain information about the run time version used for building the assembly, public key etc.  
  
3. CIL Code - CIL code are actual implementation of code in terms of instructions. CIL code is described in length below.  
  
4. Type Metadata - Describes the types contained within the assembly and the format of types referenced by that assembly.  
  
5. An Assembly Manifest-Describes the modules within the assembly, the version of the assembly,all the external assemblies reference by that assembly.

CTS-CTS is known as common type system. It describe set of data type that can be used in different .net language in common.CTS ensure that object written in different .net language can interact with each other. For communicating between programs written in any.net complaint language the type have to be compatible on the basic level. The common type system support two general category of type-

1. Value type

2. Reference type.

1. Value type-

1. Value type directly contains their data and instances of value type are either allocated on the stack or allocated inline in a structure.
2. Value type can be built in means implemented by the runtime, user defined or enumerations.
3. Value type can be created at compile time.
4. A value type cannot contain the null value. However, the [nullable types](http://msdn.microsoft.com/en-us/library/1t3y8s4s.aspx) feature does allow for value types to be assigned to null.
5. Garbage collection can’t access the stack.
6. Each value type has an implicit default constructor that initializes the default value of that type
7. If a value type is declared outside of a method but inside a reference type it will be placed within the reference type on the heap. If a value type is boxed it will be placed in the heap. If the value type is within an iterator block it will be placed in the heap else goes in the stack.

2. Reference Type-Reference type store a reference to the value’s memory address and are allocated on the heap. Reference type can be self-describing type, pointer type or interface types

1. Reference type can be created at run time.
2. Stored in heap memory
3. Garbage collector can access heap.
4. Reference type holds the data indirectly.
5. Reference type holds default value.eg-class, object, array, indexers, interface.

Modifiers-Modifiers are used to modify declarations of types and type members. This section introduces the C# modifiers.

1. Access modifiers
2. Abstract
3. Async
4. Const
5. Event
6. Extern
7. New
8. Override
9. Partial
10. Readonly
11. Sealed
12. Static
13. Unsafe
14. Virtual
15. Volatile

Access Modifiers: Access modifiers are keywords used to specify the declared accessibility of a member or a type.  
Access modifiers are an integral part of object-oriented programming. They support the concept of encapsulation, which promotes the idea of hiding functionality.

* Public Access Modifiers: The public keyword is a member access modifier. Public access is the most permissive access level. There are no restrictions on accessing public members,Can be accessed by objects of the class
* Can be accessed by derived classes

Private Access Modifiers: The private keyword is a member access modifier. Private access is the least permissive access level. Private members are accessible only within the body of the class or the struct in which they are declared, as in this example:

* Cannot be accessed by object
* Cannot be accessed by derived classes

Protected Access Modifiers: The protected keyword is a member access modifier. A protected member is accessible within its class and by derived class instances. A protected member of a base class is accessible in a derived class only if the access takes place through the derived class type.

* Cannot be accessed by object
* By derived classes

Internal Access Modifier The internal keyword is an access modifier for types and type members. We can declare a class as internal or its member as internal. Internal members are accessible only within files in the same assembly (.dll).In other words, access is limited exclusively to classes defined within the current project assembly.

Accessibility:

In same assembly (public)

* Can be accessed by objects of the class
* Can be accessed by derived classes

In other assembly (internal)

* Cannot be accessed by object
* Cannot be accessed by derived classes

protected internal

The protected internal accessibility means protected OR internal, not protected AND internal.

In other words, a protected internal member is accessible from any class in the same assembly, including derived classes.

The protected internal access modifier seems to be a confusing but is a union of protected and internal in terms of providing access but not restricting. It allows:

* Inherited types, even though they belong to a different assembly, have access to the protected internal members.
* Types that reside in the same assembly, even if they are not derived from the type, also have access to the protected internal members.

Interface-

1. We create interface using interface keyword.
2. An interface is a reference type containing only abstract members.
3. Just like class interface also contain property, method, indexers, delegate or event but only declaration and no implementation.
4. It is a compile time error to provide implementation for any interface member.
5. Interface member are public by default and they don’t allow explicit access modifiers.
6. Interface cannot contain constants, constructors, data fields, destructors, static members or other interfaces.
7. A class or struct that inherit with interface must provide implementation for all interface member otherwise it generate compile time error.
8. A class or struct can inherit from more than one interface at the same time but a class cannot inherit more than one class at the same time.
9. Interface inherit other interface a class that inherit this interface must provide implementation for all interface member in the entire interface inheritance chain.
10. We cannot create an instance of an interface but an interface reference variable can point to a derived class object.
11. If a class implements more that one interface and the two interfaces have methods named the same, then each method must be implemented unless they have the same definition in which case only one implementation is needed to implement the methods.
12. Using interface based design concept provides loose coupling, component-based programming, easier maintainability, dynamic polymorphism and makes your code base more scalable and makes code reuse much more accessible because implementation is separated from the interface. Interfaces add a *plug and play* like architecture into your applications. Interfaces help define a contract/blueprint between your application and other objects. This indicates what sort of methods, properties and events are exposed by an object.

Difference between abstract class and Interface-

1. An Abstract class doesn't provide full abstraction but an interface does provide full abstraction.
2. An Interface can support multiple inheritance. An abstract class cannot support multiple inheritance.
3. Using Abstract we cannot achieve multiple inheritances but using an Interface we can achieve multiple inheritances.
4. An Interface member cannot be defined using the keyword static, virtual, abstract or sealed.
5. No fields can be defined in **interfac**es. An abstract **class** can have fields and constraints defined
6. An interface can inherit from another interface only and cannot inherit from an abstract class; whereas an abstract class can inherit from another abstract class or another interface.
7. An Interface cannot contain constructors or destructors. An abstract class can contain constructors or destructors.
8. An Interface can be inherited from by structures. An abstract class cannot be inherited from by structures.

When to use interface and abstract class-If you have an implementation that will be the same for all the derived classes, then it is better to go for an abstract class instead of an interface. So, when you have an interface, you can move your implementation to any class that implements the interface. Whereas, when you have an abstract class, you can share implementation for all derived classes in one central place, and avoid code duplication in derived classes.

OOPS-The Object oriented programming (OOP) is a programming model of objects, data fields and methods interacting together for the purpose of application development. It allow decomposition of a problem into a number of entities called Object and then builds data and function around these objects.

1. The data of the objects can be accessed only by the functions associated with that object.
2. The functions of one object can access the functions of other object.

Class-Class is a blueprint of an object that contains variables for storing data and functions to performing operations on these data. Class will not occupy any memory space and hence it is only logical representation of data.

Object-Objects are the basic run-time entities in an object oriented system. They may represent a person, a place or any item that the program has to handle.

Class will not occupy any memory space. Hence to work with thedata represented by the class you must create a variable for the class, which is called as an object.

When an object is created by using the keyword **new**, then memory will be allocated for the class in heap memory area, which is called as an instance and its starting address will be stored in the object in stack memory area.

When an object is created without the keyword new, then memory will not be allocated in heap I.e. instance will not be created and object in the stack contains the value **null.**

When an object contains null, then it is not possible to access the members of the class using that object.

All the programming languages supporting object oriented Programming will be supporting these four main concepts:

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

Abstraction-

1. Abstraction is "To represent the essential feature without representing the back ground details."
2. Abstraction lets you focus on what the object does instead of how it does it.
3. Abstraction provides you a generalized view of your classes or object by providing relevant information.
4. Abstraction is the process of hiding the working style of an object, and showing the information of an object in understandable manner.
5. Abstraction is implemented using interface and abstract class

**Example of Abstraction:**

Suppose you have an object Mobile Phone.

Suppose you have 3 mobile phones as following:-

Nokia 1400 (Features:- Calling, SMS)

Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)

Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS."

so that, for mobile phone object you will have abstract class like following:-

**Simple Example**

    abstract class MobilePhone

    {

        public void Calling();

        public void SendSMS();

    }

    public class Nokia1400 : MobilePhone

    {

    }

    public class Nokia2700 : MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

    }

    public class BlackBerry : MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

        public void Recording();

        public void ReadAndSendEmails();

    }

Abstraction means putting all the variables and methods in a class which are necessary.

For example: Abstract class and abstract method.

**Example from implementation point of view:**

           In an university form for college, you need to fill your details like name, address, date of birth, percentage etc. and for the other application form for the doctor you need to fill the details like name, address, date of birth, blood group, height and weight. Here the common thing netween these two forms are Age, name, address so you can create the class which consist of common thing that is called abstract class.

That class is not complete and it can inherit by other class.

**Real World Example of Abstraction**:-

**Example-1:**

                     While you are driving a car you know how to stop and turn the car using break and staring even you don't know how the internal parts working properly i.e. functionality of engine.

But how the Mobile Phone internally working?, how keypad buttons are connected with internal circuit? is called Encapsulation.

**Example-2:**

                   Same in mobile you can dial a number using keypad buttons. Even you don't know how these are working internally. This is called Abstraction. You have the only information that is needed to dial a number but not its internal working of mobile.

Encapsulation- "Wrapping up data member and method together into a single unit (i.e. Class) is called Encapsulation"

* Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.
* Encapsulation is like your bag in which you can keep your pen, book etc. It means this is the property of encapsulating members and functions.
* Encapsulation means hiding the internal details of an object, i.e. how an object does something.
* Encapsulation prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented.
* Encapsulation is a technique used to protect the information in an object from the other object.
* Hide the data for security such as making the variables as private, and expose the property to access the private data which would be public.

    class Bag

    {

        book;

        pen;

        ReadBook();

    }

**Example of Encapsulation:**

class Demo

{

   private int \_mark;

   public int Mark

   {

     get { return \_mark; }

     set { if (\_mark > 0) \_mark = value; else \_mark = 0; }

   }

 }

**Real world Example of Encapsulation:**

**Example-1:**  
                    Let's take example of Mobile Phone and Mobile Phone Manufacturer

Suppose you are a Mobile Phone Manufacturer and you designed and developed a Mobile Phone design(class), now by using machinery you are manufacturing a Mobile Phone(object) for selling, when you sell your Mobile Phone the user only learn how to use the Mobile Phone but not that how this Mobile Phone works.

This means that you are creating the class with function and by making object (capsule) of it you are making availability of the functionality of you class by that object and without the interference in the original class.

**Example-2:**

TV operation

It is encapsulated with cover and we can operate with remote and no need to open TV and change the channel.

Here everything is in private except remote so that anyone can access not to operate and change the things in TV.

**"Inheritance"**

When a class acquire the property of another class is known as inheritance i.e. a child acquire property of his parents. It is process of code re-usability.

**Example:**  
public class ParentClass

    {

        public ParentClass()

        {

            Console.WriteLine("Parent Constructor.");

        }

        public void print()

        {

            Console.WriteLine("I'm a Parent Class.");

        }

    }

    public class ChildClass : ParentClass

    {

        public ChildClass()

        {

            Console.WriteLine("Child Constructor.");

        }

        public static void Main()

        {

            ChildClass child = new ChildClass();

            child.print();

        }

    }

**Output:**

    Parent Constructor.  
    Child Constructor.  
    I'm a Parent Class.

**"Polymorphism"**

Polymorphism means **one name many forms**.

One function behaves different forms.

In other words, "Many forms of a single object is called Polymorphism."

**Real World Example of Polymorphism:**

**Example-1:**

A Teacher behaves to student.

A Teacher behaves to his/her seniors.

Here teacher is an object but attitude is different in different situation.

**Example-2:**

Person behaves SON in house at the same time that person behaves EMPLOYEE in office.

**Example-3:**

Your mobile phone, one name but many forms

* As phone
* As camera
* As mp3 player
* As radio

**Brief single line description:**

* Encapsulation is accomplished by using Class. - Keeping data and methods that accesses that data into a single unit.
* Abstraction is accomplished by using Interface. - Just giving the abstract information about what it can do without specifying the back ground details.
* Information/Data hiding is accomplished by using Modifiers - By keeping the instance variables private or protected.

Inline Function-Inline function is a technique used by the compilers and instructs to insert complete body of the function wherever that function is used in the program source code.

Friend Function-Friend function is a friend of a class that is allowed to access to Public, private or protected data in that same class. If the function is defined outside the class cannot access such information.

Friend can be declared anywhere in the class declaration, and it cannot be affected by access control keywords like private, public or protected.

Operator Overloading-Operator overloading is a function where different operators are applied and depends on the arguments. Operator,-,\* can be used to pass through the function , and it has their own precedence to execute.

Ternary Operator-Ternary operator is said to be an operator which takes three arguments. Arguments and results are of different data types , and it is depends on the function. Ternary operator is also called as conditional operator.

Different type of argument-A parameter is a variable used during the declaration of the function or subroutine and arguments are passed to the function , and it should match with the parameter defined. There are two types of Arguments.

* Call by Value – Value passed will get modified only inside the function , and it returns the same value whatever it is passed it into the function.
* Call by Reference – Value passed will get modified in both inside and outside the functions and it returns the same or different value.

Super key-Super keyword is used to invoke overridden method which overrides one of its superclass methods. This keyword allows to access overridden methods and also to access hidden members of the superclass.

It also forwards a call from a constructor to a constructor in the superclass.

Token-Tokens are generally any unit that is not whitespace or a comment. They are part of the text of a program.

Pure virtual function-A pure virtual function makes the class an abstract class, which itself cannot be instantiated. This means that to use the class, you have to inherit from it, and you MUST override all pure virtual functions defined in the abstract class.

**What are all the operators that cannot be overloaded?**

Following are the operators that cannot be overloaded -.

1. Scope Resolution (:: )
2. Member Selection (.)
3. Member selection through a pointer to function (.\*)

Base class,sub class and super class-

Base class is the most generalized class , and it is said to be a root class.

Sub class is a class that inherits from one or more base classes.

Super class is the parent class from which another class inherits.

Fields-Field is a data member of class. A *field*is a variable of any type that is declared directly in a [class](http://msdn.microsoft.com/en-us/library/0b0thckt.aspx) or [struct](http://msdn.microsoft.com/en-us/library/ah19swz4.aspx). Fields are *members* of their containing type.

**C# does not support multiple class inheritance**because of the diamond problem that is associated, with multiple class inheritance. Let us understand the diamond problem of multiple class inheritance with an example.

|  |
| --- |
| http://3.bp.blogspot.com/-pDOmliP7miA/Td6r5G94IsI/AAAAAAAAAGc/S69vr0wPUSc/s1600/Multiple+Class+Inheritance.png |

**As shown in the image above:**  
**1.** I have 2 classes - **ClassB** and **ClassC**  
**2.** Both of these classes inherit from **ClassA**  
**3.** Now, we have another class, **ClassD** which inherits from both **ClassB** and **ClassC**  
  
So, if a method in **ClassD** calls a method defined in **ClassA** and **ClassD**has not overriden the invoked method. But both **ClassB**and **ClassC**have overridden the same method differently. Now, the ambiguity is, from which class does, **ClassD**inherit the invoked method: **ClassB**, or **ClassC**?

Abstract class-

1. The abstract keyword enables you to create classes and class members only for the purpose of inheritance.
2. The [abstract](http://msdn.microsoft.com/en-us/library/sf985hc5.aspx) keyword enables you to create classes and [class](http://msdn.microsoft.com/en-us/library/0b0thckt.aspx) members that are incomplete and must be implemented in a derived class.
3. The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.
4. The abstract class is incomplete and hence cannot be instantiated.
5. An abstract class cannot be sealed.
6. An abstract class may contain abstract members (method, property, indexers and events) but not mandatory.
7. Like an interface, you cannot implement an instance of an abstract class; however you can implement methods, fields, and properties in the abstract class that can be used by the child class.
8. A non-abstract class derived from an abstract class must provide implementation for all inherited abstract members.
9. If a class inherit an abstract class there are two option available for that class-

* Option 1 provides implementation for all the abstract members inherited from the base abstract class.
* Option 2 define if the class does not wish to provide implementation for all the abstract members inherited from the abstract class then the class has to be marked as abstract.

Features:

1. An abstract class can inherit from a class and one or more interfaces.
2. An Abstract class can have modifiers for class members.
3. An Abstract class can have constants and fields.
4. An abstract class can implement a property.
5. An abstract class can have constructors or destructors.
6. An abstract class cannot be inherited from by structures.
7. An abstract class cannot support multiple inheritances.
8. Members of an abstract class may have any access modifier.
9. It is not possible to modify an abstract class with the sealed modifier, which means that the class cannot be inherited.
10. A non-abstract class derived from an abstract class must include actual implementations of all inherited abstract methods and accessors.I
11. t is mandatory to override abstract method in the derived class

Sealed Class-

1. The [sealed](http://msdn.microsoft.com/en-us/library/88c54tsw.aspx) keyword enables you to prevent the inheritance of a class or certain class members that were previously marked [virtual](http://msdn.microsoft.com/en-us/library/9fkccyh4.aspx).
2. Sealed class can create instances, but cannot inherit and contain static as well as nonstatic members.
3. Sealed class can be a derived class but can't be a base class
4. A sealed class cannot be used as a base class. For this reason, it cannot also be an abstract class.
5. Sealed classes are primarily used to prevent derivation. Because they can never be used as a base class, some run-time optimizations can make calling sealed class members slightly faster.
6. Advantages of sealed class are it restricts the third party vendor for developing new software by inheriting from our logic.

Partial Class-

1. Partial class is a feature which allows us to write class across multiple files.
2. The partial indicates that the parts of the class, struct, or interface can be defined in the namespace.
3. All the parts must be used with the partial keyword. All the parts must be available at compile time to form the final type or final class.
4. All the parts must have the same accessibility level, such as public, private, protected, and so on.
5. If any part of the class is declared abstract, then the whole type is considered to be as abstract.
6. If any part is declared sealed, then the whole type is considered to be as sealed.
7. If any part declares a base type, then the whole type inherits that class.

Static class-

1. A class can be declared static, indicating that it contains only static members.
2. It is not possible to create instances of a static class using the new keyword.
3. Static classes are loaded automatically by the .NET Framework common language runtime (CLR) when the program or namespace containing the class is loaded.
4. The advantage of using a static class is that the compiler can check to make sure that no instance members are accidentally added.
5. Static classes are sealed and therefore cannot be inherited.
6. Static classes cannot contain a constructor, although it is still possible to declare a static constructor to assign initial values or set up some static state

Method- A method is a member that implements a computation or action that can be performed by an object or class.

Method parameter-

Params- This parameter keyword can be used to specify method parameter that takes variable number of arguments. When the number of values required are not fixed and the internal mechanism can handler variable number of arguments processing, in that situation we can use params keyword for a method parameter. This method parameter can either accept comma-separated list of arguments of specified type, array of arguments of specified type or no arguments.

14 public void DemonstrateParams(paramsint[] inputParamList)

{

if (inputParamList.Length&gt;= 1)

{

for (int counter = 0; counter &gt; inputParamList.Length; counter++)

{

Console.WriteLine("Parameter No:" + counter + " = " + inputParamList[counter]);

}

}

else

{

Console.WriteLine("No parameter found !!");

}

}

}

if we pass null to the method, then null will not be the first element of the object[] array, but will benullinputParamList object[] array. Thus it is a good coding practice to check for null value before using the method parameter having params keyword specified.

To Remember

• After the method parameter with params keyword, no other parameters can be specified.

• There can be only one method parameter with params keyword.

Ref- A normal method parameter is passed by value. (Don’t get this confused by passing reference type and value type). But using “ref” keyword we can pass an argument by reference. This means that when value of that parameter is changed in the method, it gets reflected in the calling environment (caller method). This keyword can be used by specify ref at method definition and at the calling method explicitly.

public void DemonstrateRef(ref intinputParam)

{

Console.WriteLine("Value of ref Parameter is : " + inputParam);

}

Call to the method:

C#

1

2

3 intsomevalue = 10;

refKeywordObj = new RefKeyword();

refKeywordObj.DemonstrateRef(ref somevalue);

To Remember

• An argument that is passed using a ref keyword must be initialized before it is passed.

• Properties cannot be passed to ref parameters. Internally they are functions and not members / variables.

Out- Keyword “out” can be used in two different contexts.

1). Parameter modifier

2). In generic type parameter declarations in interface and delegates.

Keyword “out” also causes argument to be passed by reference like “ref” keyword, but argument with outkeyword can be passed with initialization of the variable or without assigning any value to it. This keyword can be used by specifying out at method definition and at the calling method explicitly.

To Remember

• An argument that is passed using an out keyword must be initialized in the method before it return and from all code paths.

• Properties cannot be passed to out parameters. Internally they are functions and not members / variables.

Similarity between “ref” and “out” keyword in C#

• Both can be used to return multiple values from any method. But out keyword can be used to return multiple values optionally.

• The method parameter with ref and out keyword behaves same at compile time and is considered same. Thus, methods cannot be overloaded if the only difference is that one method takes ref argument and other takes out argument.

Difference between “ref” and “out” keyword in C#

• “ref” parameter must be initialized before calling the method. “out” parameter must be initialized in the method and from all code paths before it returns.

• Reading an out argument in the method before it is initialized or assigned is not allowed. Whereas ref argument can be read anytime once it value is assigned or initialized before call to the method.

• Information cannot be passed into the method when out argument is specified as it has to be initialized in the method before using. On the other hand ref argument can be used to pass information or value to the method and get some value in the same variable back at the calling environment.

Optional Parameter-

When a function has an optional parameter, the caller can decide whether to provide a value or not. (Providing a value for the parameter is optional.) If the caller doesn't provide a value, the function uses a default value.

Named Parameter-?

Abstract method- When an instance method declaration includes an abstract modifier, that method is said to be an abstract method. An abstract method is implicitly a virtual method, it cannot have the modifier virtual. An Abstract method is a method without a body. The implementation of an abstract method is done by a derived class. This requirement is enforced at compile time and is also called dynamic polymorphism.

Features-

• An abstract method is implicitly a virtual method.

• Abstract method declarations are only permitted in abstract classes.

• It is an error to use the static or virtual modifiers in an abstract method declaration.

• It is an error to use the abstract modifier on a static property.

Sealed Method- When an instance method declaration includes a sealed modifier, that method is said to be a sealed method. If an instance method declaration includes thesealed modifier, it must also include the override modifier. Use of the sealed modifier prevents a derived class from further overriding the method.

Partial Method-A partial method is a special method that existswithin a partial class or struct.partial method contains two parts, method definition and implementation. The definition is mostly written by code generation tool and implementation is written manually by developer. If no implementation provided then compiler removes method signature at compile time. Partial methods enable class designers to provide method hooks, similar to event handlers, that developers may decide to implement or not. If the developer does not supply an implementation, the compiler removes the signature at compile time.

Features-

• Signatures in both parts of the partial type must match.

• The method must return void.

• No access modifiers are allowed. Partial methods are implicitly private.

• Partial methods can be declared or defined with in the partial class or struct.

• Partial methods are implicitly private and declarations must have partial keyword.

• Partial methods implementation is optional.

• Partial methods can be static and unsafe and generic.

• Partial methods can have ref parameters but not out parameters since these can't return value.

• You cannot make a delegate to a partial method.

• The signatures of partial method will be same in both parts of the partial class or struct.

• Cannot be marked as extern.

• Can be generic.

• Cannot be declared as virtual.

Static method-

1. Static methods have no instances.
2. They are called with the type name, not an instance identifier.
3. We have no need to create the instance of class to call the static methods.So they are slightly faster than instance methods.
4. The static methods use the[**static**](http://msdn.microsoft.com/en-us/library/98f28cdx%28v=vs.80%29.aspx)keyword.
5. **Static methods** cannot access non-static class level members and do not have a 'this' pointer.
6. Instance methods can access those members, but must be called through an object instantiation, which causes another step and level of indirection.

Features-

• It can access only the static fields of the class.

• It can directly invoke the methods that are defined as static.

Virtual Method-

1. A virtual method is a method that can be redefined in derived classes.
2. A virtual method has an implementation in a base class as well as derived the class.
3. It is used when a method's basic functionality is the same but sometimes more functionality is needed in the derived class.
4. A virtual method is created in the base class that can be overriden in the derived class.
5. We create a virtual method in the base class using the virtual keyword and that method is overriden in the derived class using the override keyword.
6. When a method is declared as a virtual method in a base class then that method can be defined in a base class and it is optional for the derived class to override that method. The overriding method also provides more than one form for a method. Hence it is also an example for polymorphism.
7. When a method is declared as a virtual method in a base class and that method has the same definition in a derived class then there is no need to override it in the derived class. But when a virtual method has a different definition in the base class and the derived class then there is a need to override it in the derived class.
8. When a virtual method is invoked, the run-time type of the object is checked for an overriding member. The overriding member in the most derived class is called, which might be the original member, if no derived class has overridden the member.
9. By default, methods are non-virtual. We can't override a non-virtual method.
10. We can't use the virtual modifier with the static, abstract, private or override modifiers.

Override method-

1. Method overriding means having two methods with same name and same signature, one method in base class and other method in derived class.
2. A subclass inherits methods from a base class. Sometimes, it is necessary for the subclass to modify the methods defined in the base class.
3. This is referred to as method overriding. This can be achieved by using the virtual and override keywords.
4. We have to use the virtual keyword for the method which in base class and override keyword for the method in subclass.
5. By default functions are not virtual in C# and so you need to write “virtual” explicitly.
6. Overriding is the concept of runtime polymorphism. It needs inheritance. Method should have same data type. Method should be public.
7. You cannot override a non-virtual or static method.
8. The overridden base method must be virtual, abstract, or override.
9. An override declaration cannot change the accessibility of the virtual method.
10. Both the override method and the virtual method must have the same access level modifier.
11. You cannot use the following modifiers to modify an override method.
12. An overriding property declaration must specify the exact same access modifier, type, and name as the inherited property, and the overridden property must be virtual, abstract, or override.

Method overloading-

1. Method Overloading means having two or more methods with the same name but with different signature (different parameters list and different type of parameters) in same class or in different classes.
2. Method Overloading forms compile-time polymorphism.
3. It doesn't need inheritance. Method can have different data types .Method can be different access specifies

Method hiding-

1. We can predefined the method of parent class in child class by implementing the concept of method hiding.
2. In method hiding the base(parent) class method is predefined in child class by specify new keyword.

Anonymous method-

1. An anonymous method is inline unnamed method in the code.
2. Anonymous methods provide a technique to pass a code block as a delegate parameter.
3. An anonymous method has only body without name, optional parameters and return type.
4. An anonymous method behaves like a regular method and allows us to write inline code in place of explicitly named methods.

Features of anonymous method

1. A variable, declared outside the anonymous method can be accessed inside the anonymous method.
2. A variable, declared inside the anonymous method can’t be accessed outside the anonymous method.
3. We use anonymous method in event handling.
4. An anonymous method, declared without parenthesis can be assigned to a delegate with any signature.
5. Unsafe code can’t be accessed within an anonymous method.
6. An anonymous method can’t access the ref or out parameters of an outer scope.

Extension method-

1. Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.
2. Extension methods are a special kind of static method, but they are called as if they were instance methods on the extended type.

Key points about extension methods

1. An extension method is defined as static method but it is called like as instance method.
2. An extension method first parameter specifies the type of the extended object, and it is preceded by the "this" keyword.
3. An extension method having the same name and signature like as an instance method will never be called since it has low priority than instance method.
4. An extension method couldn't override the existing instance methods.
5. An extension method cannot be used with fields, properties or events.
6. The compiler doesn't cause an error if two extension methods with same name and signature are defined in two different namespaces and these namespaces are included in same class file using directives. Compiler will cause an error if you will try to call one of them extension method.
7. An extension method will never be called if it has the same signature as a method defined in the type.
8. If the class is sealed than there in no concept of extending its functionality. For this a new concept is introduced i.e. extension methods.

Property-

1. Properties are class members that allow you to expose a characteristic of an object.
2. A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field.
3. Properties can be used as if they are public data members, but they are actually special methods called accessors.
4. This enables data to be accessed easily and still helps promote the safety and flexibility of method
5. Properties enable a class to expose a public way of getting and setting values, while hiding implementation or verification code.
6. A get property accessor is used to return the property value, and a set accessor is used to assign a new value. These accessors can have different access level
7. The value keyword is used to define the value being assigned by the set accessor.
8. Properties that do not implement a set accessor are read only.

Read-only properties-It is possible to create read-only properties. To create a read-only property, we omit the set accessor and provide only the get accessor in the implementation.

Write-only property- It is possible to create Write -only properties. To create a Write -only property, we omit the get accessor and provide only the set accessor in the implementation.

Auto implemented property- There are no bodies for the accessors in an automatically implemented property. You just use the Get and set keywords, followed by a semicolon. Most importantly, there is no field either. Automatically implemented properties allow you to reduce the clutter in your code if you are just using a property to get and set the value of a field. When you compile your class, the C# system creates a variable that is of the same type as the property and generates accessors. We don’t have any access to the variable or even know its name. We can only use the property to get and set its value. Modifer are used in property-public, protected, internal, private, protected internal, virtual, override, abstract, sealed, static.

Indexers-

1. Indexers allow instances of a class or struct to be indexed just like arrays.
2. Indexers resemble properties except that their accessors take parameters.
3. Indexers enable objects to be indexed in a similar manner to arrays.
4. A get accessor returns a value. A set accessor assigns a value.
5. The this keyword is used to define the indexers.
6. The value keyword is used to define the value being assigned by the set indexer.
7. Indexers do not have to be indexed by an integer value; it is up to you how to define the specific look-up mechanism.
8. Indexers can be overloaded.
9. Indexers can have more than one formal parameter, for example, when accessing a two-dimensional

Constructor- Whenever a class or struct is created, its constructor is called. A class or struct may have multiple constructors that take different arguments. Constructors enable the programmer to set default values, limit instantiation, and write code that is flexible and easy to read. Constructors have the same name as the class or struct, and they usually initialize the data members of the new object. A constructor that takes no parameters is called a default constructor. Default constructors are invoked whenever an object is instantiated by using the new operator and no arguments are provided to new.

* A Constructor does not have a return type not even void.
* Multiple overloads of constructors can be defined.
* Constructors can have an access modifier.
* Constructors are mainly used to initialize private fields of the class.
* The constructor is invoked by the new operator immediately after memory is allocated for the new object.
* The class can have only one default constructor.
* Structs cannot contain an explicit default constructor because one is provided automatically by the compiler. This constructor initializes each field in the struct to the default values.

There are five types of instance constructors in C# language; they are:

* Defaut constructor
* Parameterize Constructor
* Copy Constructor
* Static Constructor
* Private Constructor

\*Note- Depends on what kind of constructor it is - if it is the default constructor  
(i.e. constructor with no parameters) then you can't override the parent's  
constructor, it will always execute. If it is a non-default constructor then  
it is automatically overriden unless you call it using the "base" keyword.  
Note that the default constructor of the parent is always called regardless  
of what kind of constructor is being invoked of the child class. Also,  
whenever a base-class constructor executes, it does so before the inherited  
class constructor is run.  
Hope that helps,

Default Constructor-A constructor that has no parameters is called the default constructor. Each class has a default constructor. When we don't want to initialize any values for class members then there is no need to define it for the class but when we want to initialize values of class members then we need to provide it for the class. When we don't create a constructor in the class, then the compiler will automatically create a default constructor in the class that initializes all fields by their default values.It has a drawback that when we create instances of the class then every instance member will be assigned the same values. The default constructor initializes:

1. All numeric fields in the class to zero.
2. All string and object fields to null.

2. Parameterized Constructor

A constructor with at least one parameter is called a parametrized constructor. The advantage of a parametrizedconstructor is that you can initialize each instance of the class to different values.

3. Copy Constructor

he 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.

4. Private Constructor

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 membersonly. Some key points of a private constructor are:

1. One use of a private constructor is when we have only static members.
2. It provides an implementation of a singleton class pattern
3. 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.

• A Private constructor does not have any parameters.

• The declaration of the private constructor is empty.

• It has a private modifier, by default a constructor has a private modifier.

• In a class we can declare only one private constructor.

Static Constructor- When a constructor is created as static, it will be invoked only once for all of instances of the class and it is invoked during the creation of the first instance of the class or the first reference to a static member in the class. A static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.

Static constructors have the following properties:

• A static constructor does not take access modifiers or have parameters.

• A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.

• A static constructor cannot be called directly.

• The user has no control on when the static constructor is executed in the program.

• A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.

• Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method.

• If a static constructor throws an exception, the runtime will not invoke it a second time, and the type will remain uninitialized for the lifetime of the application domain in which your program is running.

Destructor-

Destructors are used to destruct instances of classes.

• Destructors cannot be defined in structs. They are only used with classes.

• A class can only have one destructor.

• Destructors cannot be inherited or overloaded.

• Destructors cannot be called. They are invoked automatically.

• A destructor does not take modifiers or have parameters.

**Destructor**

They are special methods that contains clean up code for the object. You can’t call them explicitly in your code as they are called implicitly by GC (Garbage Collector). In C# they have same name as the class name preceded by the "~" sign. Like-  
  
class MyClass

{

    public MyClass()

    {

    }

    ~MyClass()

    {

    }

}

**Dispose**

These are just like any other methods in the class and can be called explicitly but they have a special purpose of cleaning up the object. In the dispose method we write clean up code for the object. It is important that we freed up all the unmanaged resources in the dispose method like database connection, files etc.

The class implementing dispose method should implement **IDisposable** which is inherited by interface and it contains ***GC.SuppressFinalize*** method for the object it is disposing if the class has destructor because it has already done the work to clean up the object, then it is not necessary for the garbage collector to call the object's Finalize method.

* **Dispose**() is called by the user
* Same purpose as finalize, to free unmanaged resources. However, implement this when you are writing a custom class that will be used by other users.
* Overriding **Dispose** () provides a way for the user code to free the unmanaged objects in your custom class.
* Dispose method can be invoked only by the classes that IDisposable interface.

**Finalize**

**Finalize ()** is called by Garbage Collector implicitly to free unmanaged resources. The garbage collector calls this method at some point after there are no longer valid references to the object. There are some resources like windows handles, database connections which cannot be collected by the garbage collector. Therefore the programmer needs to call **Dispose()** method of **IDisposable** interface.

* Implement it when you have unmanaged resources in your code, and want to make sure that these resources are freed when the Garbage collection happens.
* Finalizers should release unmanaged resources only.
* Finalizers should always be protected, not public or private so that the method cannot be called from the application's code directly and at the same time, it can make a call to the base.Finalize method

class MyClass : IDisposable

{

    private bool IsDisposed = false;

    public void Dispose()

    {

        Dispose(true);

        GC.SupressFinalize(this);

    }

    protected void Dispose(bool Diposing)

    {

        if (!IsDisposed)

        {

            if (Disposing)

            {

                //Clean Up managed resources

            }

            //Clean up unmanaged resources

        }

        IsDisposed = true;

    }

    ~MyClass()

    {

        Dispose(false);

    }

}

**Response.Redirect**

1. Response.Redirect() will send you to a new page, update the address bar and add it to the Browser History. On your browser you can click back.
2. It redirects the request to some plain HTML pages on our server or to some other web server.
3. It causes additional roundtrips to the server on each request.
4. It doesn’t preserve Query String and Form Variables from the original request.
5. It enables to see the new redirected URL where it is redirected in the browser (and be able to bookmark it if it’s necessary).
6. Response. Redirect simply sends a message down to the (HTTP 302) browser.

**Server.Transfer**

1. Server.Transfer() does not change the address bar, we cannot hit back.One should use Server.Transfer() when he/she doesn’t want the user to see where he is going. Sometime on a "loading" type page.
2. It transfers current page request to another .aspx page on the same server.
3. It preserves server resources and avoids the unnecessary roundtrips to the server.
4. It preserves Query String and Form Variables (optionally).
5. It doesn’t show the real URL where it redirects the request in the users Web Browser.
6. Server.Transfer happens without the browser knowing anything, the browser request a page, but the server returns the content of another.

Object Initializer-Object initializers let you assign values to any accessible fields or properties of an object at creation time without having to explicitly invoke a constructor.It is a compile-time error to use an object initializer with a nullable struct.

Example-class Cat

{

// Auto-implemented properties.

public int Age { get; set; }

public string Name { get; set; }

}

Cat cat = new Cat { Age = 10, Name = "Fluffy" };

* Object initialization with anonymous type-Anonymous types are reference types derived form *system.objects*. Anonymous types provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first. The type name is generated by the compiler and is not available at the source code level. The type of each property is inferred by the compiler.

You create anonymous types by using the [new](http://msdn.microsoft.com/en-us/library/vstudio/51y09td4.aspx) operator together with an object initializer.

Collection Initializer-Collection initializers let you specify one or more element initializers when you initialize a collection class that implements [IEnumerable](http://msdn.microsoft.com/en-us/library/vstudio/system.collections.ienumerable.aspx)

var pet = new { Age = 10, Name = "Fluffy" };

Structur- A struct type is a value type that is typically used to encapsulate small groups of related variables. Structs can also contain constructors, constants, fields, methods, properties, indexers, operators, events, andnested types. Structs can implement an interface but they cannot inherit from another struct. For that reason, struct members cannot be declared as protected.

• Within a struct declaration, fields cannot be initialized unless they are declared as const or static.

• A struct cannot declare a default constructor (a constructor without parameters) or a destructor.

• Structs are copied on assignment. When a struct is assigned to a new variable, all the data is copied, and any modification to the new copy does not change the data for the original copy. This is important to remember when working with collections of value types such as Dictionary<string, myStruct>.

• Structs are value types and classes are reference types.

• Unlike classes, structs can be instantiated without using a new operator.

• Structs can declare constructors that have parameters.

• A struct cannot inherit from another struct or class, and it cannot be the base of a class. All structs inherit directly from System.ValueType, which inherits from System.Object.

• A struct can implement interfaces.

• A struct can be used as a nullable type and can be assigned a null value.

Structor Constructor-

• Structs cannot contain explicit parameterless constructors. Struct members are automatically initialized to their default values.

• A struct cannot have an initializer in the form: base (argument-list).

Enum-An enumeration type also known as enum provide an efficient way to define a set of named integral constant that may be assigned to a variable. Enum are value type and stored on stack.Enum are strongly typed constant.By default first enum has the value 0 and the value of each successive enum is increased by 1.

Advantage of using this

1. Enums are not allocated in memory. They exist only on compilation stage. When code runs - there is no enums there anymore.
2. If you Writing the Architectural Code, then You have to go for Enum not Const variable String like 'const sting Word="Word"'
3. In my example I have used the MS product like Word, Excel etc, it clearly tells you, that you have a definite choice of product in the application, not more than that. One more thing, if new person wants to add a new product, he or she clearly knows where to add this product and how it can be used.
4. Enum.Parse (), this is very useful, if you consider the web application if the query string "product=word" in the URL, then we use to Get Enum and use throughout the Application, it is helpful.
5. Main thing is Readability, you should always remember this.
6. Enum.Parse throws error if value is incorrect, so we use Enum.IsDefined to check. So every helper method in Enum helps a lot.
7. If you have choice like list of action or options or possibilities, then you can go for Enum. eg. FileAccess (Read, Write etc).
8. You will get error in compile time itself not run time, it is better and best practice.
9. As you know it is a value type and also we can customize the Enum value as mentioned in this article.
10. It reduces the chance of Bug in code something like case sensitive problem.
11. IntelliSense makes you happy to work as you know how much it is helpful while programming. This is also the best use of Enum. Once place  we use Word, in other place we use word, it is inconsistent too. If you think of it, it helps a lot. You don't have to remember the string fully , just type Enum and Make a dot it shows the list.

Var-Microsoft introduced “var” keyword in C# 3.0. The purpose of using this keyword is when you don’t know type of variable. The technical term for var is "implicitly typed local variable declaration". The variable itself is still statically typed (the type is determined when the code is compiled), and it is still strongly typed (the type cannot change once it has been declared). the compiler infers the type of the variable based upon the initialize for the variable. It is mainly use in anonymous methods, LINQ etc. Some of the programmers use this variable frequently in declaration.var cannot initialize to null value.It must not return the null type.

Dynamic-Visual C# 2010 introduces a new type, dynamic. The type is a static type, but an object of type dynamic bypasses static type checking.

| Var | | dynamic | |
| --- | --- | --- | --- |
| Introduced in C# 3.0 | | Introduced in C# 4.0 | |
| Statically typed – This means the type of variable declared is decided by the compiler at compile time. | | Dynamically typed - This means the type of variable declared is decided by the compiler at runtime time. | |
| Need to initialize at the time of declaration.  e.g., varstr=”I am a string”;  Looking at the value assigned to the variable str, the compiler will treat the variable str as string. | | No need to initialize at the time of declaration.  e.g., dynamic str;  str=”I am a string”; //Works fine and compiles  str=2; //Works fine and compiles | |
| Errors are caught at compile time.  Since the compiler knows about the type and the methods and properties of the type at the compile time itself | | Errors are caught at runtime  Since the compiler comes to about the type and the methods and properties of the type at the run time. | |
| Visual Studio shows intellisense since the type of variable assigned is known to compiler. | | Intellisense is not available since the type and its related methods and properties can be known at run time only | |
| e.g., var obj1;  will  throw a compile error since the variable is not initialized. The compiler needs that this variable should be initialized so that it can infer a type from the value. | | e.g., dynamic obj1;  will compile; | |
| e.g. var obj1=1;  will compile  var obj1=” I am a string”;  will throw error since the compiler has already decided that the type of obj1 is System.Int32 when the value 1 was assigned to it. Now assigning a string value to it violates the type safety. | | e.g. dynamic obj1=1;  will compile and run  dynamic obj1=” I am a string”;  will compile and run since the compiler creates the type for obj1 as System.Int32 and then recreates the type as string when the value “I am a string” was assigned to it.  This code will work fine. | |
| Object | Dynamic | | Var |
| Can able to store any kind of value, because object is the base class of all type in .net framework. | Can able to store any type of the variable, similar to old VB language variable. | | Can able to store any type of value but it require to initialize at the time of declaration.  [http://2.bp.blogspot.com/-f4R_nVJyN7Q/TjKtHKB0NNI/AAAAAAAACcE/vZmriKOlC5Y/s200/var+inialize+error.JPG](http://2.bp.blogspot.com/-f4R_nVJyN7Q/TjKtHKB0NNI/AAAAAAAACcE/vZmriKOlC5Y/s1600/var+inialize+error.JPG) |
| Compiler has little information about the type  [http://1.bp.blogspot.com/-O3j4Ud1rn9s/TjKtOoGsrbI/AAAAAAAACcI/Q1a9HF2Hbok/s200/object.JPG](http://1.bp.blogspot.com/-O3j4Ud1rn9s/TjKtOoGsrbI/AAAAAAAACcI/Q1a9HF2Hbok/s1600/object.JPG) | Compiler doesn't have any information about the this type of variable.  [http://4.bp.blogspot.com/-9B9snB1pPFg/TjKtRr435OI/AAAAAAAACcM/FnUjqaGFYI0/s200/dynamic.JPG](http://4.bp.blogspot.com/-9B9snB1pPFg/TjKtRr435OI/AAAAAAAACcM/FnUjqaGFYI0/s1600/dynamic.JPG) | | It's compiler safe i.e compiler has all information about the stored value, so that it doesn't cause any issue at run-time.  [http://4.bp.blogspot.com/-BXXEgzeJ8XE/TjKtWeI_JGI/AAAAAAAACcQ/iW6xh5sYPg4/s200/var+compiler.JPG](http://4.bp.blogspot.com/-BXXEgzeJ8XE/TjKtWeI_JGI/AAAAAAAACcQ/iW6xh5sYPg4/s1600/var+compiler.JPG) |
| Object type can be passed as function argument and function also can return object type | Dynamic type can be passed as function argument and function also can return object type | | Var type can not be passed as function argument and function can not return object type. This type of variable can work in the scope where it defined.  [http://4.bp.blogspot.com/-dkV7VvttM34/TjKtbFDf0yI/AAAAAAAACcU/PnUFg5KIDcE/s200/var+passing+argument.JPG](http://4.bp.blogspot.com/-dkV7VvttM34/TjKtbFDf0yI/AAAAAAAACcU/PnUFg5KIDcE/s1600/var+passing+argument.JPG) |
| Require to cast object variable to original type before using it. So this assigning to object type and converting to original type called as Boxing and Un-Boxing for value type and for the reference type its casting of types. It's actually increasing the overhead when we do this both operation. Allows to perform operation of given type once it get cast any user defined or primitive data type. | Casting is not require but you need to know the property and methods related to stored type | | No need to cast because compiler has all information to perform operation. |
| Cause the problem at run time if the stored value is not get converted to underlying data type.  [http://2.bp.blogspot.com/-AcTarSXqc08/TjKtgMQ-jiI/AAAAAAAACcY/ldRUTGHpTO4/s200/Object+error.JPG](http://2.bp.blogspot.com/-AcTarSXqc08/TjKtgMQ-jiI/AAAAAAAACcY/ldRUTGHpTO4/s1600/Object+error.JPG) | Cause problem if the wrong method or property accessed because all the information about stored value is get resolve only at run time  [http://2.bp.blogspot.com/-Gk8fm_tIc98/TjKtjhCgnPI/AAAAAAAACcc/BZshBUrwC90/s200/dynamic+error.JPG](http://2.bp.blogspot.com/-Gk8fm_tIc98/TjKtjhCgnPI/AAAAAAAACcc/BZshBUrwC90/s1600/dynamic+error.JPG) | | Doesn't cause problem because compiler has all info about stored value. |
| Useful when doesn't have more information about the data type. | Useful when coding using reflection or dynamic language support or with the COM objects, because we require to write less amount of code. | | Useful when getting result out of the linq queries. In 3.5 framework it introduce to support linq feature. |