Biyani's Think Tank

Concept Based Notes

.NET Framework with C#

Mr. Sachin Bagoria

Asst. Professor (Dept. of IT)

Biyani Girls College, Jaipur

BIYANI
Where you can trust

BCA Vth Semester

Published by :

Think Tanks

Biyani Group of Colleges

Concept & Copyright :

Biyani Shikshan Samiti Sector-3, Vidhyadhar Nagar, Jaipur-302 023 (Rajasthan)

Ph : 0141-2338371, 2338591-95 Fax : 0141-2338007

E-mail : acad@biyanicolleges.org

Website :www.gurukpo.com; www.biyanicolleges.org

ISBN :

Edition: 2025

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs ofthe students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self- explanatory and adopts the “Teach Yourself” style. It is based on question- answer pattern. The language of book is quite easy and understandable basedon scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

BCA-76T-311: .NET Framework With C#

Unit-I

Introduction to .Net framework: Managed Code and the CLR Intermediate Language, Metadata and JIT Compilation Automatic Memory Management

The Framework Class Library: .Net objects- ASP .NET, NET web services, Windows Forms.

Elements : Variable and constants data types, declaration. Operators, types precedence, Expressions Program flow, Decision statements, if then if. Then.else.select.case, Loop statements while and while, do.loop. for next for each.next

Unit-II

Types: Value data types Structures, Enumerations, Reference data types, arrays.

Windows Programming: Creating windows forms windows controls, Button, Check box, Combo box, Label, List box Radio Button, Text box, Events, Click, close deactivate, Load, mousemove, mousedown, mouseup.

Menus and Dialog Boxes : Creating menus, menu items, context menu, Using dialog boxes, show dialog() method.

Unit – III

ADO.NET : Architecture of ADO.NET, ADO.NET providers, Connection, Command, Data Adapter, Dataset, Connecting to Data Source, Accessing Data with Data set and Data reader, Create an ADO.NET application, Using Stored Procedures.

ASP.NET Features: Application of States and Structure; Change the Home Directory in IIS- Add a Virtual Directory in IIS- Set a Default Document for IIS – Change Log File Properties for IIS-Stop, Start, or Pause a Web Site.

Unit-IV

Creating Web Controls: Web Controls, HTML Controls, Using Internist Control, Using Input Validation Controls, Selecting Controls for Applications, Data Controls and Adding web controls to a page.

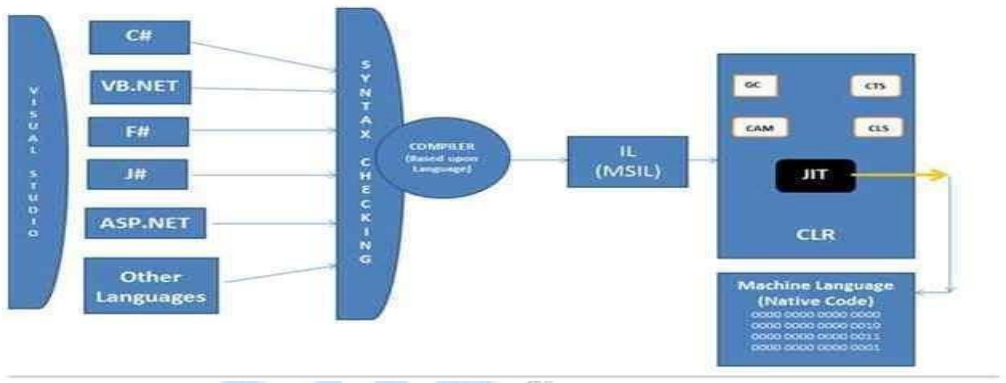
Creating Web Forms: Server Controls, Types of Server Controls, Adding ASP.NET Code to a page.

Web Services and WCF : Web Services protocol and standards – WSDL Documents-Visual Studio.NET Architecture of WCF, WCF Client

Question 1: What is the .NET Framework?

Answer: The .NET is a Framework, which is a collection of classes of reusable libraries given by Microsoft to be used in other .NET applications and to develop, build and deploy many types of applications on the Windows platform including the following:

- Console Applications
- Windows Forms Applications
- Windows Presentation Foundation (WPF) Applications
- Web Applications
- Web Services
- Windows Services
- Services-oriented applications using Windows Communications Foundation (WCF)
- Workflow-enabled applications using Windows Workflow Foundation (WF) that primarily runs on the Microsoft Windows Operating System.



Question 2: What is CLR?

Answer: The CLR stands for Common Language Runtime and it is an Execution Environment. It works as a layer between Operating Systems and the applications written in .NET languages that conforms to the Common Language Specification (CLS). The main function of Common Language Runtime (CLR) is to convert the Managed Code into native code and then execute the program. The Managed Code compiled only when it is needed, that is it converts the appropriate instructions when each function is called. The Common Language Runtime (CLR)'s just in time (JIT) compilation converts Intermediate Language (MSIL) to native code on demand at application run time.

When a .NET application is executed at that time the control will go to Operating System, then Operating System create a process to load **CLR.**

The program used by the operating system for loading CLR is called runtime host, which are different depending upon the type of application that is desktop or web based application i.e.

The runtime host for **desktop applications** is API function called **CorbinToRuntime**.

The runtime host for **web based** applications is ASP.NET worker process **(aspnet-wp.exe)**.

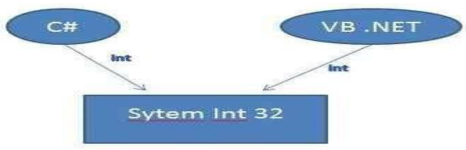
CLR runtime engine comes with set of services, which are classified as follows

CLR services

- Assembly Resolver
- Assembly Loader
- Type Checker
- COM marshalled
- Debug Manager
- Thread Support
- IL to Native compiler
- Exception Manager
- Garbage Collector

Question 3: What is CTS?

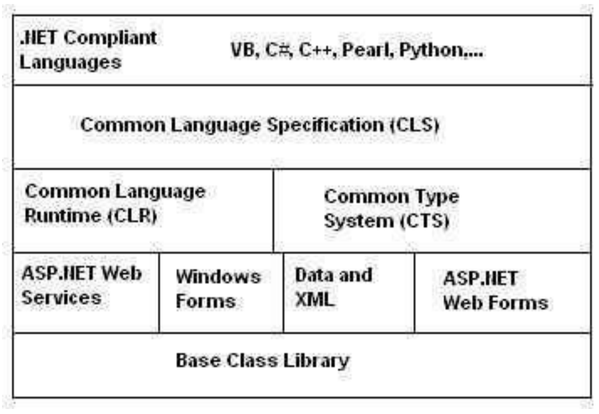
Answer: The Common Type System (CTS) standardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages.



To implement or see how CTS is converting the data type to a common data type, for example, when we declare an int type data type in C# and VB.NET, then they are converted to int32. In other words, now both will have a common data type that provides flexible communication between these two languages.

Question 4: What is CLS?

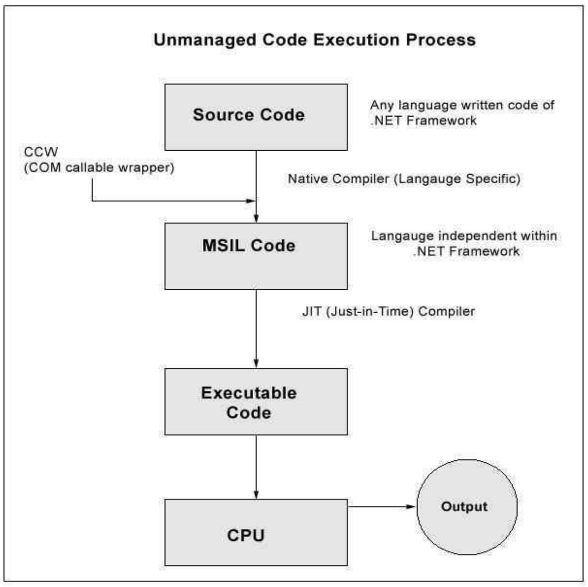
Answer: One of the important goals of .NET Framework is to support Multiple Languages. This is achieved by CLS. For multiple languages to interoperate, it is necessary that they should go on in common in certain features such as Types that are used. For example, every language has its own size and range for different data types. Thus CLS is the agreement among language designers and class library designers concerning these usage conventions.



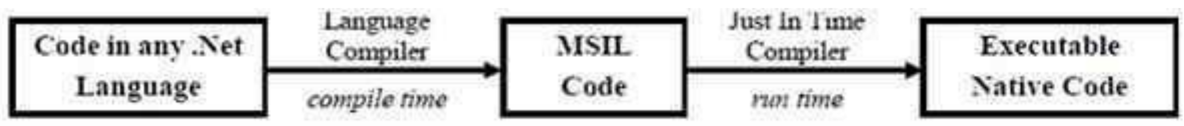
Question 5: What is managed code?

Answer: The resource, which is within your application domain is, managed code. The resources that are within domain are faster. The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in NET Framework is managed code.

Managed code uses CLR which in turn looks after your applications by managing memory, handling security, allowing cross - language debugging, and so on.

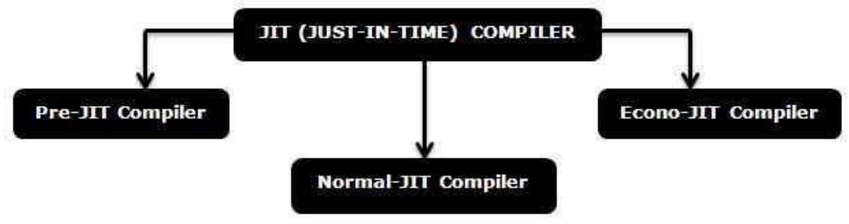

Unmanaged Code Execution Process
Source Code
Any language written code of .NET Framework
CCW (COM callable wrapper)
Native Compiler (Langauge Specific)
MSIL Code
Langauge independent within .NET Framework
JIT (Just-in-Time) Compiler
Executable Code
CPU
Output

Question 6: What is MSIL?
Answer: When we compile our .NET code then it is not directly converted to native/binary code; it is first converted into intermediate code known as MSIL code which is then interpreted by the CLR. MSIL is independent of hardware and the operating system. Cross language relationships are possible since MSIL is the same for all .NET languages. MSIL is further converted into native code.



Question 7: What is JIT?
Answer: A Web Service or Web Forms file must be compiled to run within the CLR. Compilation can be implicit or explicit. Although you could explicitly call the appropriate compiler to compile your Web Service or Web Forms files, it is easier to allow the file to be complied implicitly. Implicit compilation occurs when you request the .asmx via HTTP-SOAP, HTTP-GET, or HTTP-POST. The parser (xsp.exe) determines whether a current version of the assembly resides in memory or in the disk. If it cannot use an existing version, the parser makes the appropriate call to the respective compiler (as you designated in the **Class** property of the .asmx page).
When the Web Service (or Web Forms page) is implicitly compiled, it is actually compiled twice. On the first pass, it is compiled into IL. On the second pass, the Web Service (now an assembly in IL) is compiled into machine language. This process is called Just-In-Time JIT compilation because it does not occur until the assembly is on the target machine.

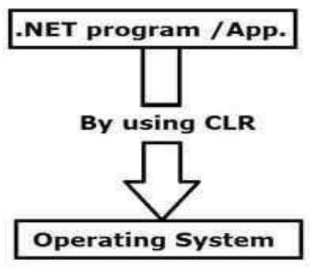
JIT Types:



Question 8: What is portable executable (PE)?

Answer: Every .NET program first compiles with an appropriate compiler like if we write a program in C# language then it gets compiled by C# compiler (i.e. csc.exe).

In .NET framework every program executes (communicate) in an operating system by using CLR



(Common Language Runtime).

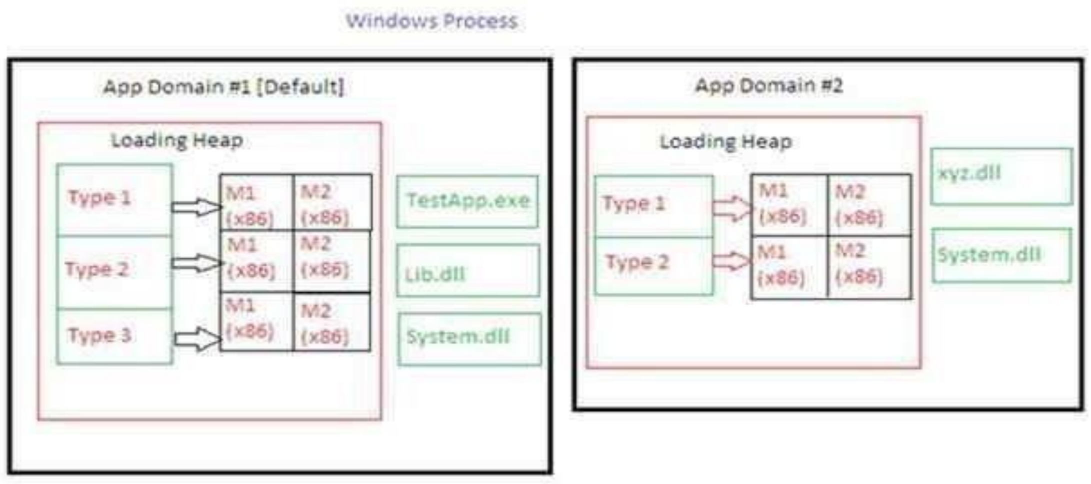
Managed module is standard windows Portable Executable (PE) file which contains the following parts.

- **PE Header- It is similar to common object file format.**
- **CLR Header-** This contains CLR version required to run this managed module, location & metadata. This also contains entry point of function i.e. the address of entry point of function.
- **Metadata-** This contains table information means variable with its data types and default values, functions / methods which are declared & defined in our program.

Question 9: What is an application domain?

Answer: An Application Domain is a logical container for a set of assemblies in which an executable is hosted. As you have seen, a single process may contain multiple Application Domains, each of which is hosting a .NET executable. The first appdomain created when the CLR is initialized is called the default AppDomain and this default one is destroyed when the Windows process is terminated.

- An AppDomain can be independently secured.
- An AppDomain can be unloaded.
- Independently configured.
- No mutual intervention by multiple appdomains.
- Performance



How does an AppDomain get created-

The AppDomain class is used to create and terminate Application Domains, load and unload assemblies and types and enumerates assemblies and threads in a domain. The following table shows some useful methods of the AppDomain class:

Methods	Description
CreateDomain()	It allows us to create a new Application Domain.
CreateInstance()	Creates an instance of type in an external assembly.
ExecuteAssembly()	It executes an *.exe assembly in the Application Domain.
Load()	This method dynamically loads an assembly into the current app domain.
UnLoad()	It allows us to unload a specified AppDomain within a given process.
GetCurrentThread()	Returns the ID of the active thread in the current Application Domain.

In addition, the AppDomain class also defined as a set of properties that can be useful when you wish to monitor the activity of a given Application Domain.

Properties	Description
CurrentDomain	Gets the Application Domain for the currently executing thread.
FriendlyName	Gets the friendly name of the current Application Domain.
SetupInformation	Get the configuration details for a given Application Domain.
BaseDirectory	Gets the directory path that the assembly resolver uses to probe for assemblies.

Question 10: What is an assembly?

Answer: An Assembly is a basic building block of .NET Framework applications. It is basically compiled code that can be executed by the CLR. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An Assembly can be a DLL or exe depending upon the project that we choose.

Assemblies are basically the following two types:

1. Private Assembly
2. Shared Assembly

Question 11: What are the contents of assembly?

Answer: Assembly An Assembly is a basic unit of application deployment and versioning.An Assembly is also called the building block of a .NET application. An Assembly is either an .exe or .dll file. An **ssembly structure consists of the following parts:**

- Assemblies manifest (name, language and version).

- CIL code (logic part).
- Type information (Datatype).
- Resources.

Question 12: What are the different types of assembly?

Answer: An Assembly contains metadata and manifest information. The reason for the emergence of assembly concept was to overcome the common "**DLL Hell**" problem in COM. The assembly contains all the code, resources, metadata and even version information. Metadata contains the details of every "type" inside the assembly. In addition to metadata, assemblies also have a special file called Manifest. It contains information about the current version of the assembly, culture information, public key token if available and other related information.

There are in all 3 different types of assemblies:

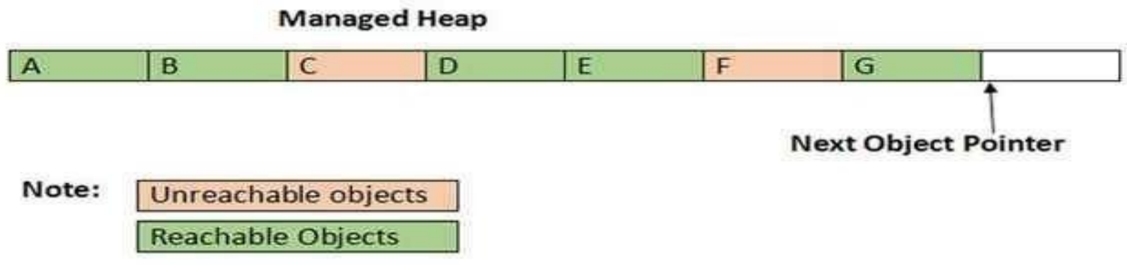
1. Private Assembly
2. Shared or Strong named assembly
3. Satellite assembly

Question 13: What is a dynamic assembly?

Answer: Technically, the act of loading external assemblies on demand is known as Dynamic Loading. Using the Assembly class, we can dynamically load both private and shared assemblies from the local location to a remote location as well as, explore its properties.

To illustrate dynamic loading, we are creating a console based application that loads an external TestLib.dll assembly. During the execution, the application asks the user to specify the dynamic loading assembly name and that reference is passed to the helper method that is responsible for loading the assembly.

Question 14: What is GAC?



Answer: The GAC is a shared location of computer where we can put an assembly so that it will be accessible from many locations; I mean it is accessible from another project or application. It's always a good practice to provide a strong name to a public assembly, I mean the assembly to be registered in the GAC, and otherwise the DLL hell problem may occur.

Problems that occurred-

I have seen DLLs added to the GAC that you can't remove - very frustrating. I have seen registered DLLs into the cache - verified everything is there ok using ILDASM only to find the DLLs are no longer in the GAC

Strongly naming the assembly-

When doing this make sure you get the directory slashes \\ correct within the assembly file (assembly.cs). - If not, you will get errors whilst the code is looking for the .snk file. If you get errors which leave you scratching your head - best bet is to remove the .snk file and start over.

Project References-

Also be careful and watch where you build projects as the referenced DLLs can easily be built to the development instead of the release folder - sometimes even when you specify the release folder. This can be very, very frustrating.

Conclusion-

My conclusion on using the GAC was only use it if you really need to as it isn't the 'end of DLL hell' as first thought. Also only use it if you are using a DLL that is shared by other projects. Don't put it in the GAC if you don't have to.

Question 15: What is a garbage collector?

Answer: The Garbage Collector (GC) is the part of the .NET Framework that allocates and releases memory for your .NET applications. The Common Language Runtime (CLR) manages allocation and deallocation of a managed object in memory. C# programmers never do this directly; there is no delete keyword in the C# language. It relies on the garbage collector.

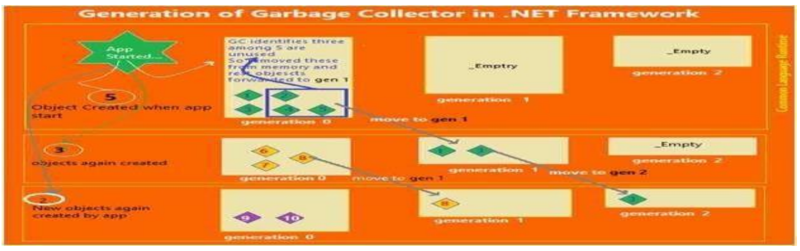
Example:

Assume the managed heap contains a set of objects named A, B, C, D, E, F and G. During garbage collection, these objects are examined for active roots. After the graph has been constructed, unreachable objects (that we will assume are objects C and F) are marked as garbage in reddish color in the following diagram.

Question 16: What are generations and how are they used by the garbage collector?

Answer: Basically the generation of Garbage Collection (GC) shows the life of objects, it means it defines how long an object will stay in the memory. It's categorized into the following three generations:

- Generation 0
- Generation 1
- Generation 2



Question 17: What is C#?

Answer: C# is the best language for writing Microsoft .NET applications. C# provides the rapid application development found in Visual Basic with the power of C++. Its syntax is similar to C++ syntax and meets 100% of the requirements of OOPs like the following:

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

Question 18: What is an Object?

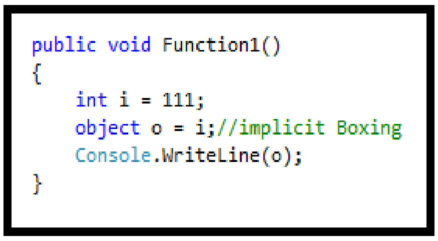
Answer: According to MSDN, "a class or struct definition is like a blueprint that specifies what the type can do. An object is basically a block of memory that has been allocated and configured according to the blueprint. A program may create many objects of the same class. Objects are also called instances, and they can be stored in either a named variable or in an array or collection. Client code is the code that uses these variables to call the methods and access the public properties of the object. In an object-oriented language such as C#, a typical program consists of multiple objects interacting dynamically".

Question 19: What is Managed or Unmanaged Code?

Answer: Managed Code- "The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with the help of managed code execution. Any language that is written in .NET Framework is managed code".
Unmanaged Code- The code, which is developed outside .NET framework, is known as unmanaged code.
"Applications that do not run under the control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write such applications, which, for example, access low - level functions of the operating system. Background compatibility with the code of VB, ASP and COM are examples of unmanaged code".
Unmanaged code can be unmanaged source code and unmanaged compile code. Unmanaged code is executed with the help of wrapper classes.
Wrapper classes are of two types:

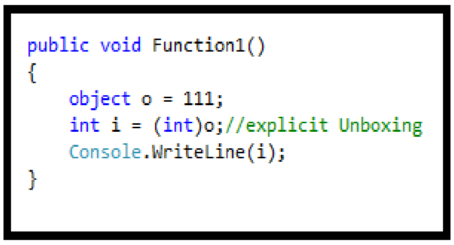
- CCW (COM Callable Wrapper).
- RCW (Runtime Callable Wrapper).

Question 20: What is Boxing and Unboxing?
Answer: Boxing and Unboxing both are used for type conversion but have some difference:
Boxing: Boxing is the process of converting a value type data type to the object or to any interface data type which is implemented by this value type. When the CLR boxes a value means when CLR is converting a value type to Object Type, it wraps the value inside a System.Object and stores it on the heap area in application domain.
Example:



```
public void Function1()
{
    int i = 111;
    object o = i;//implicit Boxing
    Console.WriteLine(o);
}
```

Unboxing: Unboxing is also a process which is used to extract the value type from the object or any



```
public void Function1()
{
    object o = 111;
    int i = (int)o;//explicit Unboxing
    Console.WriteLine(i);
}
```

implemented interface type. Boxing may be done implicitly, but unboxing have to be explicit by code.
Example:
The concept of boxing and unboxing underlines the C# unified view of the type system in which a value of any type can be treated as an object.
Question 21: What is the difference between a struct and a class in C#? Answer: Class and Struct both are the user defined data type but have some major difference: **Struct-**

- The struct is value type in C# and it inherits from System.Value Type.
- Struct is usually used for smaller amounts of data.
- Struct can't be inherited to other type.
  A structure can't be abstract.
- No need to create object by new keyword.
- Do not have permission to create any default constructor.

Class-

- The class is reference type in C# and it inherits from the System.Object Type.
- Classes are usually used for large amounts of data.
- Classes can be inherited to other class.
- A class can be abstract type.
- We can't use an object of a class with using new keyword.
- We can create a default constructor.

Question 22: What is the difference between Interface and Abstract Class?
Answer: Theoretically there are some differences between Abstract Class and Interface which are listed below:

- A class can implement any number of interfaces but a subclass can at most use only one abstract class.
- An abstract class can have non-abstract methods (concrete methods) while in case of interface all the methods has to be abstract.
- An abstract class can declare or use any variables while an interface is not allowed to do so.
- In an abstract class all data member or functions are private by default while in interface all are public, we can't change them manually.
- In an abstract class we need to use abstract keyword to declare abstract methods while in an interface we don't need to use that.
- An abstract class can't be used for multiple inheritance while interface can be used as multiple inheritance.

- An abstract class use constructor while in an interface we don’t have any type of constructor.

Question 23: What is enum in C#?

Answer:

- An enum is a value type with a set of related named constants often referred to as an enumerator list. The enum keyword is used to declare an enumeration. It is a primitive data type, which is user defined.
- An enum type can be an integer (float, int, byte, double etc.). But if you used beside int it has to be cast.
- An enum is used to create numeric constants in .NET framework. All the members of enum are of enum type. There must be a numeric value for each enum type.

The default underlying type of the enumeration element is int. By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.

```
enum Dow {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

Some points about enum-

- Enums are enumerated data type in c#.
- Enums are not for end-user, they are meant for developers.
- Enums are strongly typed constant. They are strongly typed, i.e. an enum of one type may not be implicitly assigned to an enum of another type even though the underlying value of their members is the same.
- Enumerations (enums) make your code much more readable and understandable.
- Enum values are fixed. Enum can be displayed as a string and processed as an integer.
- The default type is int, and the approved types are byte, sbyte, short, ushort, uint, long, and ulong.
- Every enum type automatically derives from System.Enum and thus we can use System.Enum methods on enums.
- Enums are value types and are created on the stack and not on the heap.

Question 24: What is the difference between “continue” and “break” statements in C#?

Answer: Using break statement, you can 'jump out of a loop' whereas by using continue statement, you can 'jump over one iteration' and then resume your loop execution.

Break Statement Example-

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
 namespace break_example {
   {
      Class brk_stmt {
         public static void main(String[] args) {
            for (int i = 0; i <= 5; i++) {
               if (i == 4) {
                  continue;  }
               Console.ReadLine(“The number is” + i); }} }  }
```

Output:

The number is 0; The number is 1; The number is 2; The number is 3;

Continue Statement Example-

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
 namespace continue_example
{
   Class cntnu_stmt
   {
      public static void main(String[]
      {
         for (int i = 0; i <= 5; i++)
         {
            if (i == 4)
            {
               continue; }
            Console.ReadLine("The number is" +i); } } } }
```

Output:

```
The number is 1; The number is 2; The number is 3; The number is 5;
```

Question 25: What is the difference between constant and read only in c#?

Answer: Constant (const) and **Readonly** (readonly) both looks like same as per the uses but they have some differences:
Constant is known as "const" keyword in C# which is also known immutable values which are known at compile time and do not change their values at run time like in any function or constructor for the life of application till the application is running.
Readonly is known as "readonly" keyword in C# which is also known immutable values and are known at compile and run time and do not change their values at run time like in any function for the life of application till the application is running. You can assay their value by constructor when we call constructor with "new" keyword.

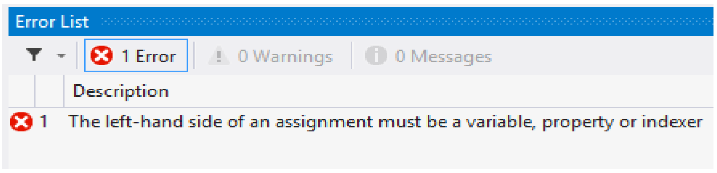
See the example-

We have a Test Class in which we have two variables one is readonly and another is constant.

```
class Test {
   readonly int read = 10;
   const int cons = 10;
   public Test() {
      read = 100;
      cons = 100;
   }
   public void Check() {
      Console.WriteLine("Read only : {0}", read);
```

```
        Console.WriteLine("const : {0}", cons);
    }
}
```

Here I was trying to change the value of both the variables in constructor but when I am trying to change the constant it gives an error to change their value in that block which have to call at run time.



Error List

1 Error | 0 Warnings | 0 Messages

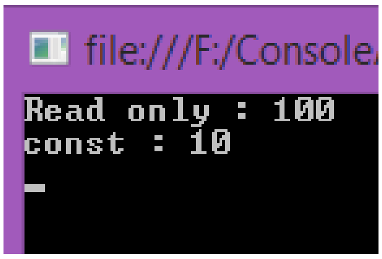
Description

1 The left-hand side of an assignment must be a variable, property or indexer

So finally remove that line of code from class and call this Check() function like the following code snippet:

```
class Program {
    static void Main(string[] args) {
        Test obj = new Test();
        obj.Check();
        Console.ReadLine();
    }
}
class Test {
    readonly int read = 10;
    const int cons = 10;
    public Test() {
        read = 100;
    }
    public void Check() {
        Console.WriteLine("Read only : {0}", read);
        Console.WriteLine("const : {0}", cons);
    }
}
```

Output:

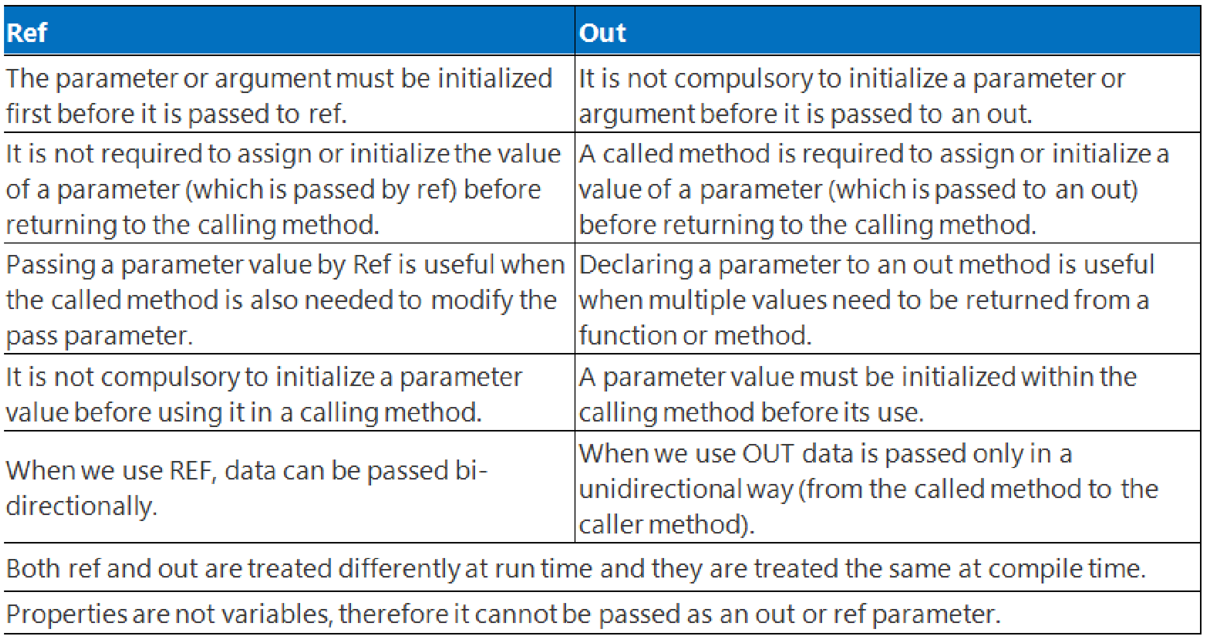


file:///F:/Console

```
Read only : 100
const : 10
_
```

Question 26: What is the difference between ref and out keywords?

Answer: In C Sharp (C#) we can have three types of parameters in a function. The parameters can be in parameter (which is not returned back to the caller of the function), out parameter and ref parameter. We have lots of differences in both of them.



Ref	Out
The parameter or argument must be initialized first before it is passed to ref.	It is not compulsory to initialize a parameter or argument before it is passed to an out.
It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method.	A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method.
Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter.	Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method.
It is not compulsory to initialize a parameter value before using it in a calling method.	A parameter value must be initialized within the calling method before its use.
When we use REF, data can be passed bi-directionally.	When we use OUT data is passed only in a unidirectional way (from the called method to the caller method).
Both ref and out are treated differently at run time and they are treated the same at compile time.	
Properties are not variables, therefore it cannot be passed as an out or ref parameter.	

Question 27: Can "this" be used within a static method?

Answer: We can't use this in static method because keyword 'this' returns a reference to the current instance of the class containing it. Static methods (or any static member) do not belong to a particular instance. They exist without creating an instance of the class and call with the name of a class not by instance so we can't use this keyword in the body of static Methods, but in case of Extension Methods we can use it the functions parameters. Let's have a look on "this" keyword.

The "this" keyword is a special type of reference variable that is implicitly defined within each constructor and non-static method as a first parameter of the type class in which it is defined. For example, consider the following class written in C#.

Question 28: Define Property in C# .net?

Answer: Properties are members that provide a flexible mechanism to read, write or compute the values of private fields, in other words by the property we can access private fields. In other words we can say that a property is a return type function/method with one parameter or without a parameter. These are always public data members. It uses methods to access and assign values to private fields called accessors.

Now question is what are accessors?

The get and set portions or blocks of a property are called accessors. These are useful to restrict the accessibility of a property, the set accessor specifies that we can assign a value to a private field in a property and without the set accessor property it is like a read-only field. By the get accessor we can access the value of the private field, in other words it returns a single value. A Get accessor specifies that we can access the value of a field publically.

We have the three types of properties

- Read/Write.
- ReadOnly.
- WriteOnly

Question 29: What is extension method in c# and how to use them?

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

How to use extension methods?

An extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.

Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

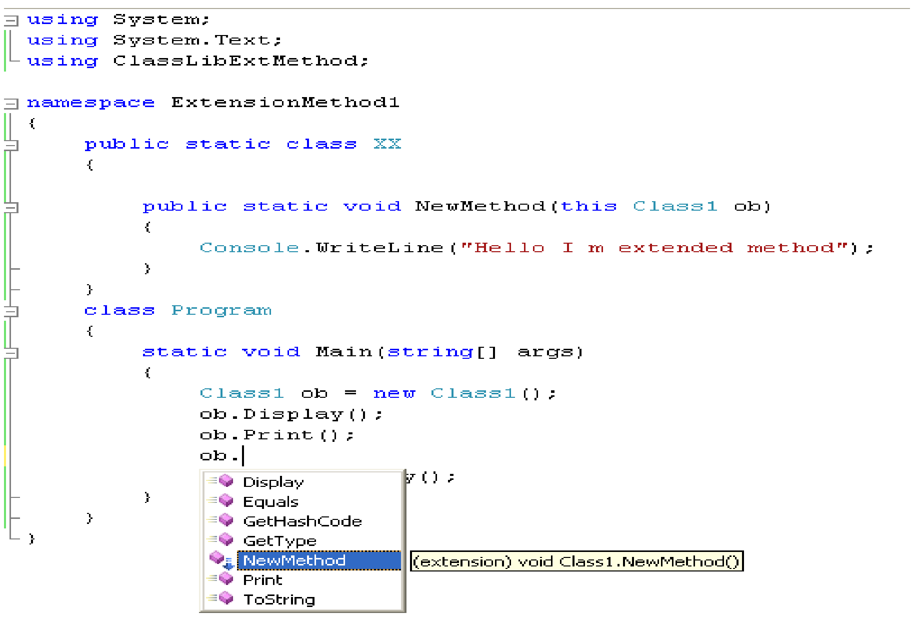
Like: suppose we have a class like bellow:

```
public class Class1 {
    public string Display() {
        return ("I m in Display");
    }
     public string Print() {
        return ("I m in Print");
    }
}
```

Now we need to extend the definition of this class so m going to create a static class to create an extinction method like:

```
public static class XX {
    public static void NewMethod(this Class1 ob) {
        Console.WriteLine("Hello I m extended method");
    }
}
```

Here I just create a method that name is NewMethod with a parameter using this to define which type of data I need to be extend, now let’s see how to use this function.

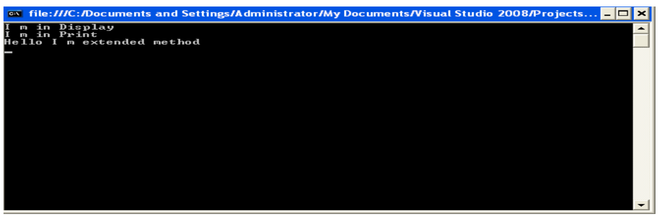


```
using System;
using System.Text;
using ClassLibExtMethod;

namespace ExtensionMethod1
{
    public static class XX
    {
        public static void NewMethod(this Class1 ob)
        {
            Console.WriteLine("Hello I m extended method");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Class1 ob = new Class1();
            ob.Display();
            ob.Print();
            ob.|
                      y();
        }
    }
}
```

Code will look like that:

```
class Program {
    static void Main(string[] args) {
        Class1 ob = new Class1();
        ob.Display();
        ob.Print();
        ob.NewMethod();
        Console.ReadKey();
    }
}
```

Output will be:



file:///C:/Documents and Settings/Administrator/My Documents/Visual Studio 2008/Projects...

```
I m in Display
I m in Print
Hello I m extended method
_
```

Question 30: What is the difference between dispose and finalize methods in c#?

Answer: Finalize and dispose both are used for same task like to free unmanaged resources but have some differences see.

Finalize:

- Finalize used to free unmanaged resources those are not in use like files, database connections in application domain and more, held by an object before that object is destroyed.
- In the Internal process it is called by Garbage Collector and can't called manual by user code or any service.
- Finalize belongs to System.Object class.
- Implement it when you have unmanaged resources in your code, and make sure that these resources are freed when the Garbage collection happens.

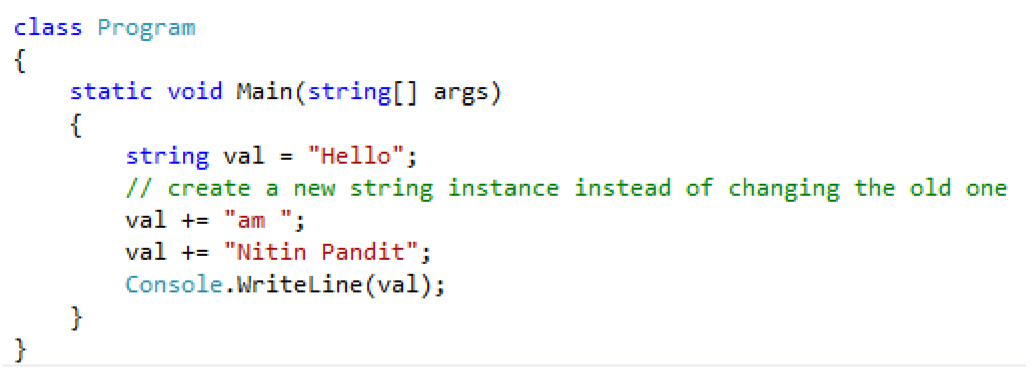
Dispose:

- Dispose is also used to free unmanaged resources those are not in use like files, database connections in Application domain at any time.
- Dispose explicitly it is called by manual user code.
- If we need to dispose method so must implement that class by IDisposable interface.
- It belongs to IDisposable interface.
- Implement this when you are writing a custom class that will be used by other users.

Question 31: What is the difference between string and StringBuilder in c#?

Answer: StringBuilder and string both use to store string value but both have many differences on the bases of instance creation and also for performance:

String: String is an immutable object. Immutable like when we create string object in code so we cannot modify or change that object in any operations like insert new value, replace or append any value with existing value in string object, when we have to do some operations to change string simply it will dispose the old value of string object and it will create new instance in memory for hold the new value in string object like:



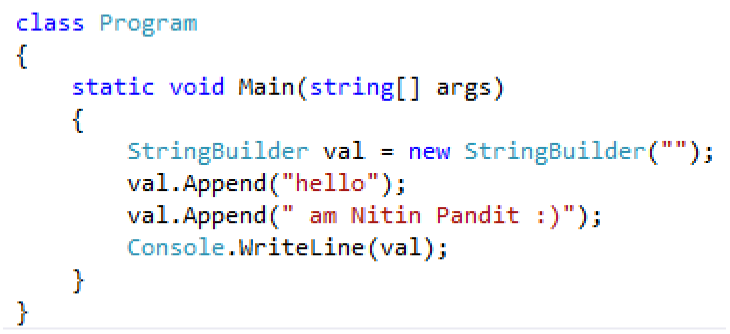
```
class Program
{
    static void Main(string[] args)
    {
        string val = "Hello";
        // create a new string instance instead of changing the old one
        val += "am ";
        val += "Nitin Pandit";
        Console.WriteLine(val);
    }
}
```

Note:

- It’s an immutable object that holds string value.
- Performance wise string is slow because its’ create a new instance to override or change the previous value.
- String belongs to System namespace.

StringBuilder:

System.Text.Stringbuilder is mutable object which also hold the string value, mutable means once we create a System.Text.Stringbuilder object we can use this object for any operation like insert value in existing string with insert functions also replace or append without creating new instance of System.Text.Stringbuilder for every time so it’s use the previous object so it’s work fast as compare than System.String. Let’s have an example to understand System.Text.Stringbuilder like:



```
class Program
{
    static void Main(string[] args)
    {
        StringBuilder val = new StringBuilder("");
        val.Append("hello");
        val.Append(" am Nitin Pandit :)");
        Console.WriteLine(val);
    }
}
```

Note:

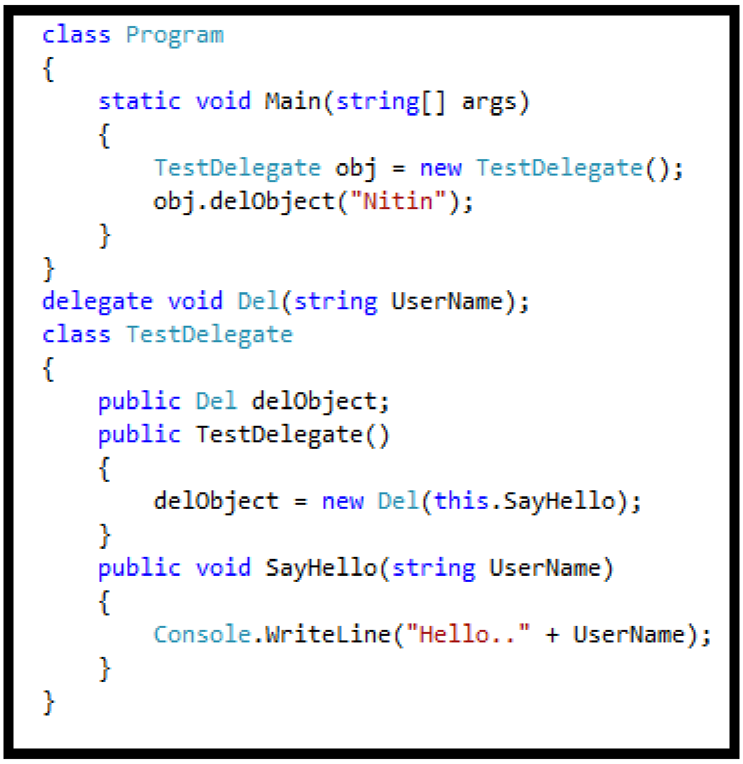
- StringBuilder is a mutable object.
- Performance wise StringBuilder is very fast because it will use same instance of StringBuilder object to perform any operation like insert value in existing string.
- StringBuilder belongs to System.Text.Stringbuilder namespace.

Question 32: What are delegates in C# and uses of delegates?

Answer: C# delegates are same as pointers to functions, in C or C++. A delegate Object is a reference type variable that use to holds the reference to a method. The reference can be changed at runtime which is hold by an object of delegate, a delegate object can hold many functions reference which is also known as Invocation List that refers functions in a sequence FIFO, we can new functions ref in this list at run time by += operator and can remove by -= operator.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

Let’s see how to use Delegate with Example:



```
class Program
{
    static void Main(string[] args)
    {
        TestDelegate obj = new TestDelegate();
        obj.delObject("Nitin");
    }
}
delegate void Del(string UserName);
class TestDelegate
{
    public Del delObject;
    public TestDelegate()
    {
        delObject = new Del(this.SayHello);
    }
    public void SayHello(string UserName)
    {
        Console.WriteLine("Hello.." + UserName);
    }
}
```

Question 33: What is sealed class in c#?

Answer: Sealed classes are used to restrict the inheritance feature of object oriented programming. Once a class is defined as a sealed class, the class cannot be inherited.

In C#, the sealed modifier is used to define a class as sealed. In Visual Basic .NET the Not Inheritable keyword serves the purpose of sealed. If a class is derived from a sealed class then the compiler throws an error.

If you have ever noticed, structs are sealed. You cannot derive a class from a struct. The following class definition defines a sealed class in C#:

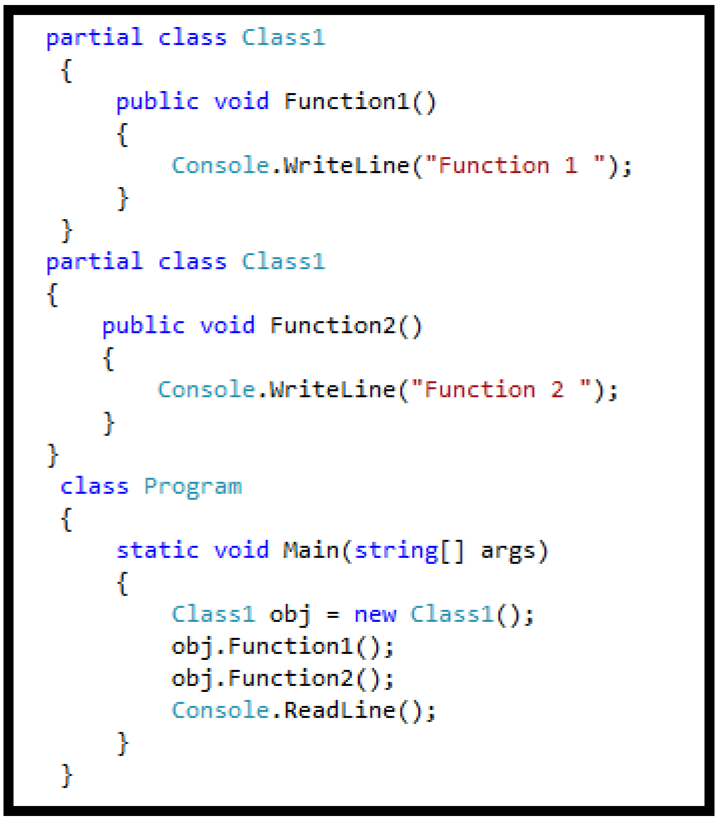
```
// Sealed class
sealed class SealedClass
{

}
```

Question 34: What are partial classes?

Answer: A partial class is only use to splits the definition of a class in two or more classes in ~~a same~~ source code file or more than one source files. You can create a class definition in multiple files but it will be compiled as one class at run time and also when you'll create an instance of this class so you can access all the methods from all source file with a same object.

Partial Classes can be create in the same namespace it's doesn't allowed to create a partial class in different namespace. So use "partial" keyword with the entire class name which you want to bind together with the same name of class in same namespace, let's have an example:

Example:

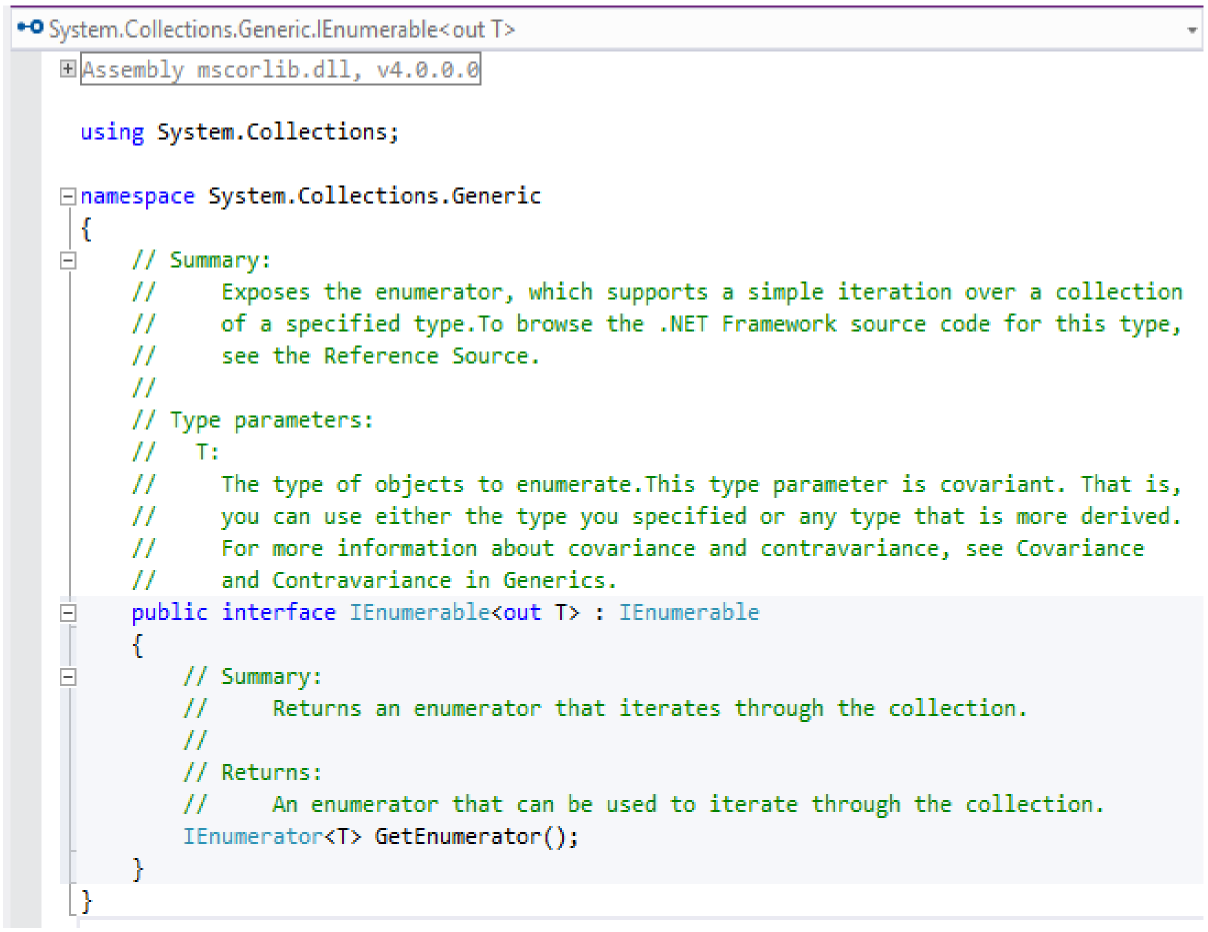


```
partial class Class1
 {
     public void Function1()
     {
         Console.WriteLine("Function 1 ");
     }
 }
partial class Class1
{
    public void Function2()
    {
        Console.WriteLine("Function 2 ");
    }
}
 class Program
 {
     static void Main(string[] args)
     {
         Class1 obj = new Class1();
         obj.Function1();
         obj.Function2();
         Console.ReadLine();
     }
 }
```

Question 35: What is IEnumerable<> in c#?

Answer: IEnumerable is the parent interface for all non-generic collections in System.Collections namespace like ArrayList, HastTable etc. that can be enumerated. For the generic version of this interface as IEnumerable<T> which a parent interface of all generic collections class in System.Collections.Generic namespace like List<> and more.

In System.Collections.Generic.IEnumerable<T> have only a single method which is GetEnumerator() that returns an IEnumerator. IEnumerator provides the power to iterate through the collection by exposing a Current property and Move Next and Reset methods, if we doesn't have this interface as a parent so we can't use iteration by foreach loop or can't use that class object in our LINQ query.



```
System.Collections.Generic.IEnumerable<out T>
Assembly mscorlib.dll, v4.0.0.0

using System.Collections;

namespace System.Collections.Generic
{
    // Summary:
    //     Exposes the enumerator, which supports a simple iteration over a collection
    //     of a specified type.To browse the .NET Framework source code for this type,
    //     see the Reference Source.
    //
    // Type parameters:
    //   T:
    //     The type of objects to enumerate.This type parameter is covariant. That is,
    //     you can use either the type you specified or any type that is more derived.
    //     For more information about covariance and contravariance, see Covariance
    //     and Contravariance in Generics.
    public interface IEnumerable<out T> : IEnumerable
    {
        // Summary:
        //     Returns an enumerator that iterates through the collection.
        //
        // Returns:
        //     An enumerator that can be used to iterate through the collection.
        IEnumerator<T> GetEnumerator();
    }
}
```

Question 36: What is difference between late binding and early binding in c#?

Answer: Early Binding and Late Binding concepts belongs to polymorphism so let’s see first about polymorphism:

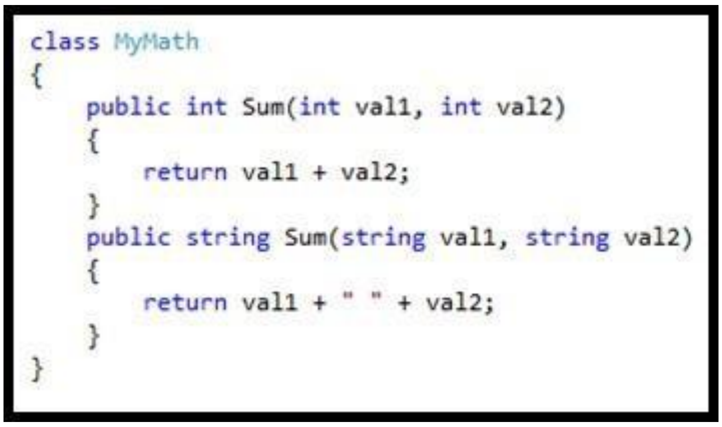
Polymorphism is an ability to take more than one form of a function means with a same name we can write multiple functions code in a same class or any derived class.

Polymorphism we have 2 different types to achieve that:

- Compile Time also known as Early Binding or Overloading.
- Run Time also known as Late Binding or Overriding.

Compile Time Polymorphism or Early Binding: In Compile time polymorphism or Early Binding we will use multiple methods with same name but different type of parameter or may be the number or parameter because of this we can perform different-different tasks with same method name in the same class which is also known as Method overloading.

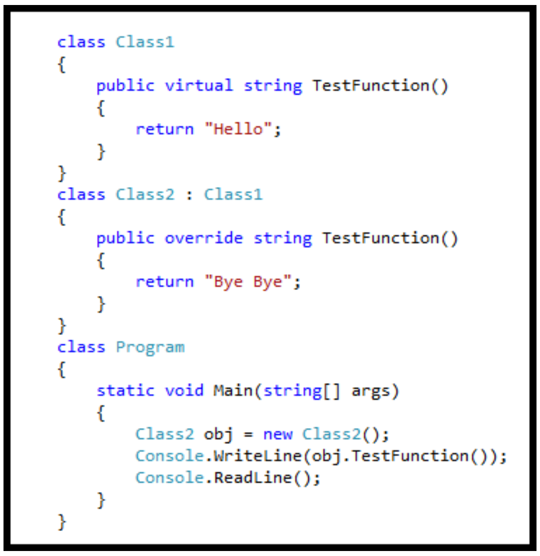
See how we can do that by the following example:



```
class MyMath
{
    public int Sum(int val1, int val2)
    {
        return val1 + val2;
    }
    public string Sum(string val1, string val2)
    {
        return val1 + " " + val2;
    }
}
```

Run Time Polymorphism or Late Binding: Run time polymorphism also known as late binding, in Run Time polymorphism or Late Binding we can do use same method names with same signatures means same type or same number of parameters but not in same class because compiler doesn't allowed that at compile time so we can use in derived class that bind at run time when a child class or derived class object will instantiated that's way we says that Late Binding. For that we have to create my parent class functions as partial and in driver or child class as override functions with override keyword.

Like as following example:



```
class Class1
{
    public virtual string TestFunction()
    {
        return "Hello";
    }
}
class Class2 : Class1
{
    public override string TestFunction()
    {
        return "Bye Bye";
    }
}
class Program
{
    static void Main(string[] args)
    {
        Class2 obj = new Class2();
        Console.WriteLine(obj.TestFunction());
        Console.ReadLine();
    }
}
```

Question 37: What are the differences between IEnumerable and IQueryable?

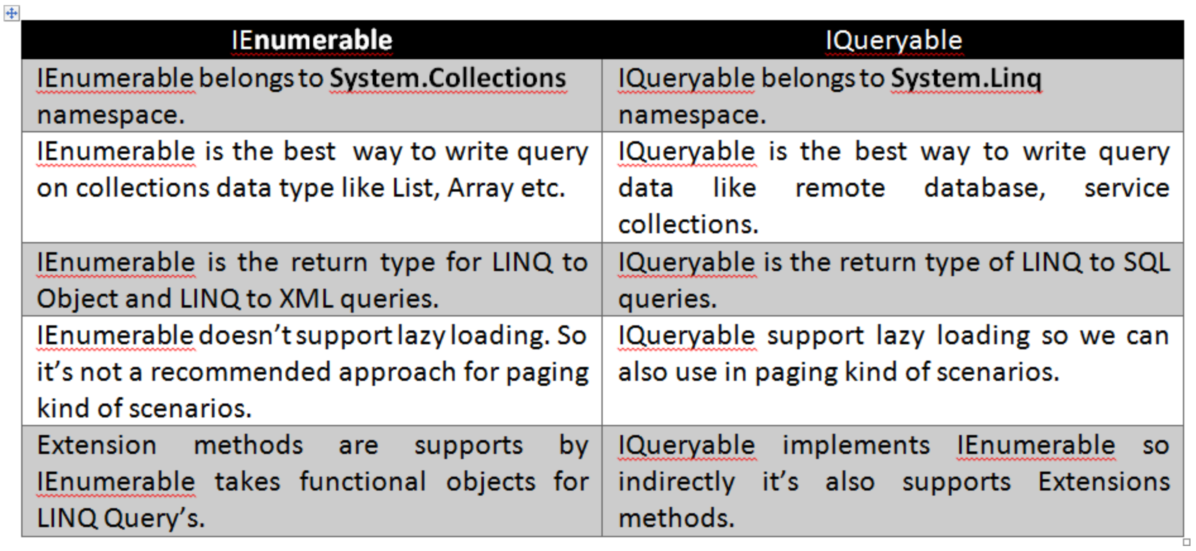
Answer: Before the differences learn what is IEnumerable and IQueryable.

IEnumerable:

Is the parent interface for all non-generic collections in System.Collections namespace like ArrayList, HastTable etc. that can be enumerated. For the generic version of this interface as IEnumerable<T> which a parent interface of all generic collections class in System.Collections.Generic namespace like List<> and more.

IQueryable:
As per MSDN IQueryable interface is intended for implementation by query providers. It is only supposed to be implemented by providers that also implement IQueryable<T>. If the provider does not also implement IQueryable<T>, the standard query operators cannot be used on the provider's data source.

The IQueryable interface inherits the IEnumerable interface so that if it represents a query, the results of that query can be enumerated. Enumeration causes the expression tree associated with an IQueryable object to be executed. The definition of "executing an expression tree" is specific to a query provider. For example, it may involve translating the expression tree to an appropriate query language for the underlying data source. Queries that do not return enumerable results are executed when the Execute method is called.



IEnumerable	IQueryable
IEnumerable belongs to System.Collections namespace.	IQueryable belongs to System.Linq namespace.
IEnumerable is the best way to write query on collections data type like List, Array etc.	IQueryable is the best way to write query data like remote database, service collections.
IEnumerable is the return type for LINQ to Object and LINQ to XML queries.	IQueryable is the return type of LINQ to SQL queries.
IEnumerable doesn't support lazy loading. So it's not a recommended approach for paging kind of scenarios.	IQueryable support lazy loading so we can also use in paging kind of scenarios.
Extension methods are supports by IEnumerable takes functional objects for LINQ Query's.	IQueryable implements IEnumerable so indirectly it's also supports Extensions methods.

Question 37: What happens if the inherited interfaces have conflicting method names?

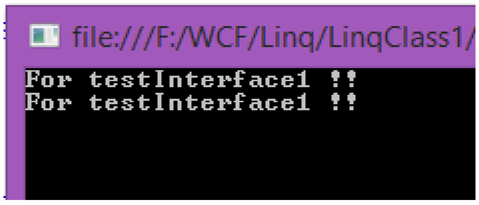
Answer: If we implement multipole interface in the same class with conflict method name so we don't need to define all or in other words we can say if we have conflict methods in same class so we can't implement their body independently in the same class coz of same name and same signature so we have to use interface name before method name to remove this method confiscation let's see an example:

1. interface testInterface1 {

```
    void Show();  }
interface testInterface2 {
    void Show();  }
class Abc: testInterface1,
testInterface2 {
   void testInterface1.Show() {
        Console.WriteLine("For testInterface1 !!");   }
    void testInterface2.Show() {
        Console.WriteLine("For testInterface2 !!");
    }
}
```

Now see how to use those in a class:

```
class Program {
    static void Main(string[] args) {
        testInterface1 obj1 = new Abc();
        testInterface1 obj2 = new Abc();
        obj1.Show();
        obj2.Show();
     Console.ReadLine();   }  }
```



```
For testInterface1 !!
For testInterface1 !!
```

Question 38 : What are the Arrays in C#.Net?

Answer: Arrays are powerful data structures for solving many programming problems. You saw during the creation of variables of many types that they have one thing in common; they hold information about a single item, for instance an integer, float and string type and so on. So what is the solution if you need to manipulate sets of items? One solution would be to create a variable for each item in the set but again this leads to a different problem. How many variables do you need?

So in this situation Arrays provide mechanisms that solves problem posed by these questions. An array is a collection of related items, either value or reference type. In C# arrays are immutable such that the number of dimensions and size of the array are fixed.

Arrays Overview-

An array contains zero or more items called elements. An array is an unordered sequence of elements. All the elements in an array are of the same type (unlike fields in a class that can be of different types). The elements of an array accessed using an integer index that always starts from zero. C# supports single-dimensional (vectors), multidimensional and jagged arrays.

Elements are identified by indexes relative to the beginning of the arrays. Indexes are also commonly called indices or subscripts and are placed inside the indexing operator ([]). Access to array elements is by their index value that ranges from 0 to (length-1).

Array Properties

- The length cannot be changed once created.
- Elements are initialized to default values.
- Arrays are reference types and are instances of System.Array.
- Their number of dimensions or ranks can be determined by the Rank property.
- An array length can be determined by the GetLength() method or Length property.

Question 39: What is the Constructor Chaining in C#?

Answer: Constructor Chaining is a way to connect two or more classes in a relationship as Inheritance, in Constructor Chaining every child class constructor is mapped to parent class Constructor implicitly by base keyword so when you create an instance of child class to it'll call parent's class Constructor without it inheritance is not possible.

Question 40: What's the difference between the System.Array.CopyTo() and System.Array.Clone()?
Answer:

Clone - Method creates a shallow copy of an array. A shallow copy of an Array copies only the elements of the Array, whether they are reference types or value types, but it does not copy the objects that the references refer to. The references in the new Array point to the same objects that the references in the original Array point to.

CopyTo - The Copy static method of the Array class copies a section of an array to another array. The CopyTo method copies all the elements of an array to another one-dimension array. The code listed in

Listing 9 copies contents of an integer array to an array of object types.

Question 41: Can Multiple Catch Blocks executed in c#?
Answer: We can use multiple Catches block with every try but when any Exceptions is throw by debugger so every catches match this exception type with their signature and catch the exception by any single catch block so that means we can use multiple catches blocks but only one can executed at once like:

```
using System;
class MyClient {
    public static void Main() {
        int x = 0;
        int div = 0;
        try {
            div = 100 / x;
            Console.WriteLine("Not executed line");

        } catch (DivideByZeroException de) {
            Console.WriteLine("DivideByZeroException");
        } catch (Exception ee) {
            Console.WriteLine("Exception");
        } finally {
            Console.WriteLine("Finally Block");
        }
        Console.WriteLine("Result is {0}", div);
    }
}
```

Question 42: What is Singleton Design Patterns and How to implement in C#?

Answer: Singleton Design Pattern-

1. Ensures a class has only one instance and provides a global point of access to it.
2. A singleton is a class that only allows a single instance of itself to be created, and usually gives simple access to that instance.
3. Most commonly, singletons don't allow any parameters to be specified when creating the instance, since a second request of an instance with a different parameter could be problematic! (If the same instance should be accessed for all requests with the same parameter then the factory pattern is

more appropriate.)

4. There are various ways to implement the Singleton Pattern in C#. The following are the common characteristics of a Singleton Pattern.

Some key points:-

- A single constructor, that is private and parameterless.
- The class is sealed.
- A static variable that holds a reference to the single created instance, if any.
- A public static means of getting the reference to the single created instance, creating one if necessary.

This is the example how to write the code with Singleton:

```
namespace Singleton {
  class Program {
    static void Main(string[] args) {
      Calculate.Instance.ValueOne = 10.5;
      Calculate.Instance.ValueTwo = 5.5;
      Console.WriteLine("Addition : " + Calculate.Instance.Addition());
      Console.WriteLine("Subtraction : " + Calculate.Instance.Subtraction());
      Console.WriteLine("Multiplication : " + Calculate.Instance.Multiplication());
      Console.WriteLine("Division : " + Calculate.Instance.Division());
      Console.WriteLine("\n ----------------------\n");
      Calculate.Instance.ValueTwo = 10.5;
      Console.WriteLine("Addition : " + Calculate.Instance.Addition());
      Console.WriteLine("Subtraction : " + Calculate.Instance.Subtraction());
      Console.WriteLine("Multiplication : " + Calculate.Instance.Multiplication());
      Console.WriteLine("Division : " + Calculate.Instance.Division());
      Console.ReadLine();
    }
  }
  public sealed class Calculate {
    private Calculate() {}
    private static Calculate instance = null;
    public static Calculate Instance {
      get {
        if (instance == null) {
          instance = new Calculate();
        }
        return instance;
        }
    }
    public double ValueOne {
        get;
        set;
    }
    public double ValueTwo {
        get;
        set;
    }
    public double Addition() {
        return ValueOne + ValueTwo;
    }
    public double Subtraction() {
        return ValueOne - ValueTwo;
    }
    public double Multiplication() {
        return ValueOne * ValueTwo;
    }
    public double Division() {
        return ValueOne / ValueTwo;
    }
  }
}
```

Question 43 : Difference between Throw Exception and Throw Clause.

Answer: The basic difference is that the Throw exception overwrites the stack trace and this makes it hard to find the original code line number that has thrown the exception.

Throw basically retains the stack information and adds to the stack information in the exception that it is thrown.

Let us see what it means rather speaking so many words to better understand the differences. I am using a console application to easily test and see how the usage of the two differs in their functionality.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace TestingThrowExceptions {
    class Program {
        public void ExceptionMethod() {
            throw new Exception("Original Exception occurred in ExceptionMethod");
        }
        static void Main(string[] args) {
            Program p = new Program();
            try {
                p.ExceptionMethod();
            } catch (Exception ex) {
                throw ex;
            }
        }
    }
}
```

Now run the code by pressing the F5 key of the keyboard and see what happens. It returns an exception and look at the stack trace:

Question 44: What is Indexer in C# .Net?

Answer: Indexer allows classes to be used in more intuitive manner. C# introduces a new concept known as Indexers which are used for treating an object as an array. The indexers are usually known as smart arrays in C#. They are not essential part of object-oriented programming.

An indexer, also called an indexed property, is a class property that allows you to access a member variable of a class using the features of an array.

Defining an indexer allows you to create classes that act like virtual arrays. Instances of that class can be accessed using the [] array access operator.

Creating an Indexer:

```
< modifier >
<return type > this[argument list] {
    get {
        // your get block code
    }
    set {
        // your set block code
    }
}
```

In the above code:

<modifier> - can be private, public, protected or internal.

<return type> - can be any valid C# types.

Question 45: What is multicast delegate in c#?

Answer: Delegate can invoke only one method reference has been encapsulated into the delegate.it is possible for certain delegate to hold and invoke multiple methods such delegate called multicast delegates.multicast delegates also know as combinable delegates, must satisfy the following conditions:

- The return type of the delegate must be void. None of the parameters of the delegate type can be delegate type can be declared as output parameters using out keywords.
- Multicast delegate instance that created by combining two delegates, the invocation list is formed by concatenating the invocation list of two operand of the addition operation. Delegates are invoked in the order they are added.

Implement Multicast Delegates Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
delegate void MDelegate();
class DM {
    static public void Display() {
        Console.WriteLine("Meerut")     }
    static public void print() {
        Console.WriteLine("Roorkee"); }
}
class MTest {
    public static void Main() {
        MDelegate m1 = new MDelegate(DM.Display);
        MDelegate m2 = new MDelegate(DM.print);
        MDelegate m3 = m1 + m2;
        MDelegate m4 = m2 + m1;
        MDelegate m5 = m3 - m2;
        m3();
        m4();
        m5();
    }
}
```

Question 46 : Difference between Equality Operator (==) and Equals() Method in C#.

Answer: Both the == Operator and the Equals() method are used to compare two value type data items or reference type data items. The Equality Operator (==) is the comparison operator and the Equals() method compares the contents of a string. The == Operator compares the reference identity while the Equals() method compares only contents. Let’s see with some examples.

In this example we assigned a string variable to another variable. A string is a reference type and in the following example, a string variable is assigned to another string variable so they are referring to the same identity in the heap and both have the same content so you get True output for both the == Operator and the Equals() method.

```
using System;
namespace ComparisionExample {
    class Program {
        static void Main(string[] args) {
            string name = "sandeep";
            string myName = name;
            Console.WriteLine("== operator result is {0}", name == myName);
            Console.WriteLine("Equals method result is {0}", name.Equals(myName));
            Console.ReadKey();
        }
    }
}
```

Question 47: Difference between “is” and “as” operator in C#.

Answer: "is" operator-

In the C# language, we use the "is" operator to check the object type. If the two objects are of the same type, it returns true and false if not.

Let's understand the preceding from a small program. We defined the following two classes:

```
class Speaker {
    public string Name {
        get;
        set;
    }
}
class Author {
    public string Name {
        get;
        set;
    }
}
```

Now, let's try to check the preceding types as:

```
var speaker = new Speaker { Name="Gaurav Kumar Arora"};
```

We declared an object of Speaker as in the following:

```
var isTrue = speaker is Speaker;
```

In the preceding, we are just checking the matching type. Yes, our speaker is an object of Speaker type.

```
Console.WriteLine("speaker is of Speaker type:{0}", isTrue);
```

So, the results as true.

But, here we get false:

```
var author = new Author { Name = "Gaurav Kumar Arora" };
var isTrue = speaker is Author;
Console.WriteLine("speaker is of Author type:{0}", isTrue);
```

Because our speaker is not an object of Author type.

"as" operator-

The "as" operator behaves similar to the "is" operator. The only difference is it returns the object if both are compatible to that type else it returns null.

Let's understand the preceding with a small snippet as in the following:

```
public static string GetAuthorName(dynamic obj)
{
Author authorObj = obj as Author;
return (authorObj != null) ? authorObj.Name : string.Empty;
}
```

We have a method that accepts dynamic objects and returns the object name property if the object is of the Author type.

Here, we declared two objects:

1. var speaker = new Speaker { Name="Gaurav Kumar Arora"};
2. var author = new Author { Name = "Gaurav Kumar Arora" };

The following returns the "Name" property:

1. var authorName = GetAuthorName(author);
2. Console.WriteLine("Author name is:{0}", authorName);

It returns an empty string:

1. authorName = GetAuthorName(speaker);
2. Console.WriteLine("Author name is:{0}", authorName);

Question 48 : How to use Nullable<> Types in .Net?

Answer: A nullable Type is a data type is that contain the defined data type or the value of null. You should note here that here variable datatype has been given and then only it can be used.

This nullable type concept is not comaptible with "var". I will explain this with syntax in next section.

Declaration:

Any DataType can be declared nullable type with the help of operator "?". Example of the syntax is as Follows:-

1. int? i = null;

As discussed in previous section "var" is not compatible with this Nullable Type. So we will have Compile Time error if we are declaring something like: -

1. var? i = null;

Though following syntax is completely fine:-

1. var i = 4;

Question 49: Different Ways of Method can be overloaded.

Answer: Method overloading is a way to achieve compile time Polymorphism where we can use a method with the same name but different signature, Method overloading is done at compile time and we have multiple way to do that but in all way method name should be same.

- Number of parameter can be different.
- Types of parameter can be different.

- Order of parameters can be different.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Hello_Word {
    class overloding {
        public static void Main() {
            Console.WriteLine(volume(10));
            Console.WriteLine(volume(2.5F,  8));
            Console.WriteLine(volume(100L, 75, 15));
            Console.ReadLine();
        }
        static int volume(int x) {
            return (x * x * x);
        }
        static double volume(float r, int h) {
            return (3.14 * r * r * h);
        }
        static long volume(long l, int b, int h) {
            return (l * b * h);
        }
    }
}
```

Question 50: What is an Object Pool in .Net?

Answer: Object Pooling is something that tries to keep a pool of objects in memory to be re-used later and hence it will reduce the load of object creation to a great extent. This article will try to explain this in detail. The example is for an Employee object, but you can make it general by using Object base class.

What does it mean?

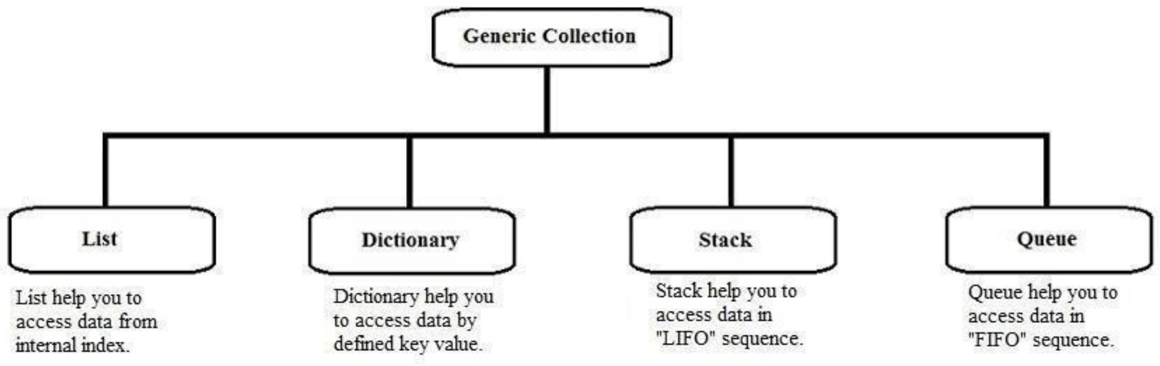
Object Pool is nothing but a container of objects that are ready for use. Whenever there is a request for a new object, the pool manager will take the request and it will be served by allocating an object from the pool.

How it works?
We are going to use Factory pattern for this purpose. We will have a factory method, which will take care about the creation of objects. Whenever there is a request for a new object, the factory method will look into the object pool (we use Queue object). If there is any object available within the allowed limit, it will return the object (value object), otherwise a new object will be created and give you back.

Question 51: What are generics in c#.net?

Answer: Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in the program. In other words, generics allow you to write a class or method that can work with any data type.

You write the specifications for the class or the method, with substitute parameters for data types. When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.


Generic Collection
List
Dictionary
Stack
Queue
List help you to access data from internal index.
Dictionary help you to access data by defined key value.
Stack help you to access data in "LIFO" sequence.
Queue help you to access data in "FIFO" sequence.

Generic classes and methods combine reusability, type safety and efficiency in a way that their non-generic counterparts cannot. Generics are most frequently used with collections and the methods that operate on them. Version 2.0 of the .NET Framework class library provides a new namespace, System.Collections.Generic, which contains several new generic-based collection classes. It is recommended that all applications that target the .NET Framework 2.0 and later use the new generic collection classes instead of the older non-generic counterparts such as ArrayList.

Features of Generics:
Generics is a technique that enriches your programs in the following ways:

- It helps you to maximize code reuse, type safety and performance.
- You can create generic collection classes. The .NET Framework class library contains several new generic collection classes in the System.Collections.Generic namespace. You may use these generic collection classes instead of the collection classes in the System.Collections namespace.
- You can create your own generic interfaces, classes, methods, events and delegates.

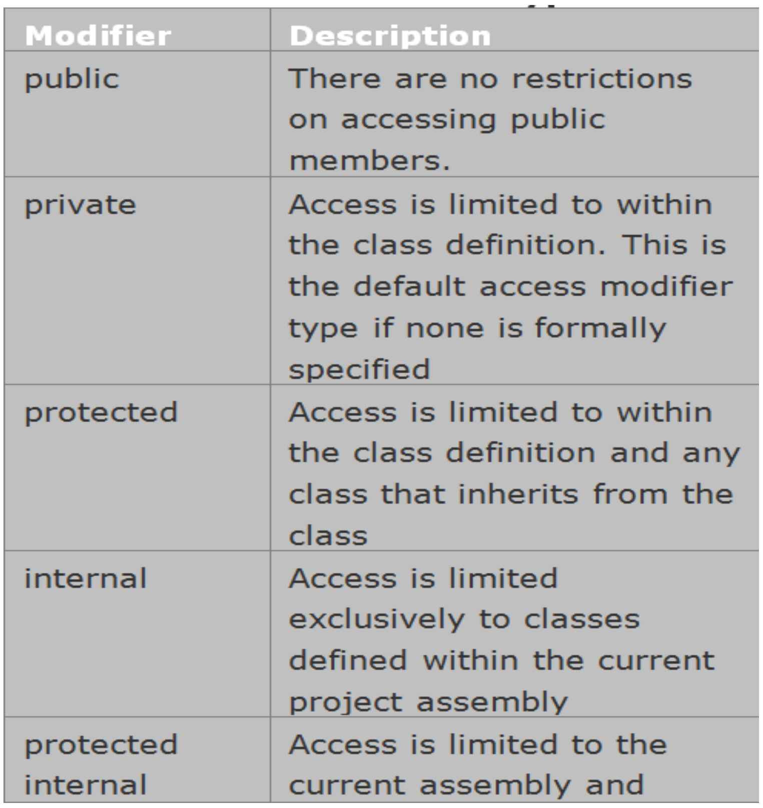
- You may create generic classes constrained to enable access to methods on specific data types.
- You may get information on the types used in a generic data type at run-time using reflection.

Question 52: Describe the accessibility modifiers in c#.Net.

Answer: Access modifiers are keywords used to specify the declared accessibility of a member or a type.

Why to use access modifiers?

Access modifiers are an integral part of object-oriented programming. They support the concept of encapsulation, which promotes the idea of hiding functionality. Access modifiers allow you to define who does or doesn't have access to certain features. In C# there are 5 different types of Access Modifiers:



Modifier	Description
public	There are no restrictions on accessing public members.
private	Access is limited to within the class definition. This is the default access modifier type if none is formally specified
protected	Access is limited to within the class definition and any class that inherits from the class
internal	Access is limited exclusively to classes defined within the current project assembly
protected internal	Access is limited to the current assembly and

Question 53: What is Virtual Method in C#?

Answer: A virtual method is a method that can be redefined in derived classes. A virtual method has an implementation in a base class as well as derived the class. It is used when a method's basic functionality is the same but sometimes more functionality is needed in the derived class. A virtual method is created in the base class that can be overridden in the derived class. We create a virtual method in the base class using the virtual keyword and that method is overridden in the derived class using the override keyword.

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.

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.

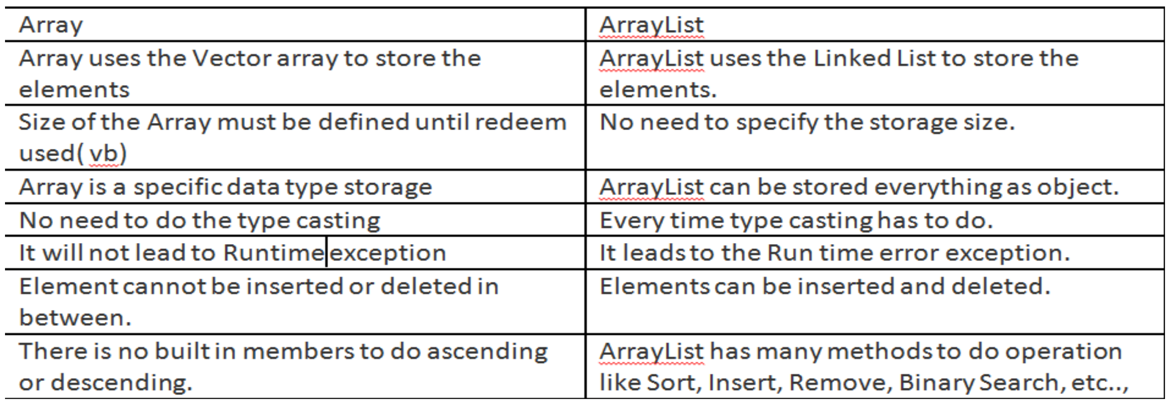
When a virtual method is invoked, the run-time type of the object is checked for an overriding member. The overriding member in the most derived class is called, which might be the original member, if no derived class has overridden the member.

Virtual Method:

1. By default, methods are non-virtual. We can't override a non-virtual method.
2. We can't use the virtual modifier with the static, abstract, private or override modifiers.

Question 54: What is the Difference between Array and ArrayList in C#.Net?

Answer: Difference between Array and ArrayList:



Array	ArrayList
Array uses the Vector array to store the elements	ArrayList uses the Linked List to store the elements.
Size of the Array must be defined until redeem used(vb)	No need to specify the storage size.
Array is a specific data type storage	ArrayList can be stored everything as object.
No need to do the type casting	Every time type casting has to do.
It will not lead to Runtime exception	It leads to the Run time error exception.
Element cannot be inserted or deleted in between.	Elements can be inserted and deleted.
There is no built in members to do ascending or descending.	ArrayList has many methods to do operation like Sort, Insert, Remove, Binary Search, etc..,

Question 55: What you understand by Value types and Reference types in C#.Net?

Answer: In C# data types can be of two types: Value Types and Reference Types. Value type variables contain their object (or data) directly. If we copy one value type variable to another then we are actually making a copy of the object for the second variable. Both of them will independently operate on their values, Value Type member will locate into Stack and reference member will locate in Heap always.

Let consider each case briefly:

1. **Pure Value Type -** Here I used a structure as a value type. It has an integer member. I created two instances of this structure. After wards I assigned second instance to the first one. Then I changed the state of second instance, but it hasn't effect the first one, as whole items are value type and assignments on those types will copy only values not references
   i.e. in a Value Type assignment, all instances have its own local copy of members.

2. **Pure Reference Type -** I created a class and added a "DataTable" as a Reference Type member for this class. Then I performed the assignments just like below. But the difference is that on changing the state of second instance, the state of first instance will automatically alter. So in a Reference Type assignment both Value and Reference will be assigned i.e. all instances will point to the single object.

3. **Value Type with Reference Type -** This case and the last case to come are more interesting. I used a structure in this particular scenario also. But this time it includes a Reference Type (A Custom Class Object) Member besides a Value Type (An Integer) Member. When you performing the assignments, it seems like a swallow copy, as Value Type member of first instance won't effected, but the Reference Type member will alter according to the second instance. So in this particular scenario, assignment of Reference Type member produced a reference to a single object and assignment of Value Type member produced a local copy of that member.

4. **Reference Type With Value Type -** Contrary to the above case, in this scenario, both Reference & Value Types will be affected. I.e. a Value Type member in a Reference Type will be shared among its instances.