100 .NET Interview Questions with Answers

- For Face-to-Face classroom/offline Angular training in Mumbai: www.stepbystepschools.net
- For more step-by-step videos visit: - https://www.questpond.com
- Also subscribe our YouTube Channel: - https://youtube.com/questpondvideos
- Join QuestPond Telegram Channel: - https://tinyurl.com/QuestPondChannel

Whats the difference between a .NET and C#?

> Note: - This probably would not come as a direct question.But its possible that you get in to a discussion about the differences.Just a warning even unknowingly never ever say C# and .NET are one and the same thing. You will create a negative dent in the interview.

.NET is a framework and C# is a programming language, that's what interviewer wants to hear.

.NET framework has collection of ready-made libraries. So for example you want List, Dictionary it has "System.collections.dll" , you want to make HTTP calls it has "System.Net.Http.dll" and so on.

While C# is programming language it has syntax, grammar, it has those "for loops"," IF" conditions and so on.

C# language invokes / orchestrates .NET framework libraries to create an application.

Differentiate between .NET framework vs .NET Core vs .NET 5.0

	.NET Framework 4.X	**.NET core 3.X**	**.NET 5.0**
Cross Platform	Only for windows	Works Cross platform	.NET 5.0 is a unified platform which unifies all .NET runtime like .NET framework, .NET core , Mono and so on. Developers no longer need to choose between .Net Core, .Net Framework and Mono, based on which platform they're developing their applications. This provides a common experience to developers irrespective on which .NET version they are working.
Performance	Slow comparison .NET core	Better	
Types of Application Supported	Winform,WPF,ASP.NET webforms , ASP.NET MVC 5	MVC core 3.X	
WCF WPF WWF	Yes	No	
Desktop based	WPF , Winforms	Nothing for desktop	
Packaging	Packaged as one big framework.	Delivered via modularly using Nuget.	
Microservice support	No	Yes	
CLI tools	More IDE Based.	Full CLI command supported	
Mobile compatibility	No support directly.	With .NET standard compatible with Xamarin	
Cloud	Works but only windows	Yes	

What is an IL code?

IL code is a partially compiled code.

```
Note: - Half compiled means this code is not yet compiled to machine/CPU
specific instructions.
```

Why IL code is not fully compiled?

We do not know in what kind of environment .NET code will run. In other words we do not know what can be the end operating system, CPU configuration, machine configuration, security configuration etc. So the IL code is half compiled and on runtime this code is compiled to machine specific using the environmental properties (CPU,OS, machine configuration etc).

Who compiles the IL code and how does it work?

IL code is compiled by JIT (Just in time compiler).

How does JIT compilationwork?

JIT compiles the code just before execution and then saves this translation in memory. Just before execution JIT can compile per-file, per function or a code fragment.

What are different types of JIT?

In Microsoft .NET there are 3 types of JIT compilers:-

- **Normal-JIT (Default): -** Normal-JIT compiles only those methods that are called at runtime. These methods are compiled the first time they are called, and then they are stored in cache. When the same methods are called again, the compiled code from cache is used for execution.
- **Econo-JIT: -** Econo-JIT compiles only those methods that are called at runtime. However, these compiled methods are not stored in cache so that RAM memory can be utilized in an optimal manner.
- **Pre-JIT: -** Pre-JIT compiles complete source code into native code in a single compilation cycle. This is done at the time of deployment of the application. We can implement Pre-jit by using ngen.exe.

Normal-jit is the default implementation and it produces optimized code. Econo-jit just replaces IL instruction with native counterpart. It does not do any kind of optimization. Econo-jit does not store the compiled code in cache so it requires less memory.

The choice of Normal-JIT and Econo-JIT is decided internally. Econo-JIT is chosen when devices have limitation memory and CPU cycle issues like windows CE powered device. When there is no memory crunch and CPU power is higher than Normal-JIT is used.

Pre-JIT is implemented by using ngen.exe which is explained in the next question.

What is Native Image Generator (Ngen.exe)?

Ngen storesfull compiled.NET native code in to cache. In other words rather than dynamically compiling the code on run time a full image of native compiled code is stored in cache while installing the application. This leads to better performance as the assembly loads and execute faster.

In order to install full compiled native code in cache we can execute the below command line from your visual studio command prompt.

```
ngen.exe install <assemblyname>
```

So does it mean that NGEN.EXE will always improve performance?

No, it’s not always necessary that ngen.exe produces optimized code because it uses the current environments parameters which can change over a period of time. For instance a code compiled in windows XP environment will not be the optimized code to run under windows 2008 server. So we need to once test with ‘ngen’ and without ‘ngen’ to conclude if really the performance increases.

Is it possible to view the IL code ?

Yes by using ILDASM simple tool we can view aIL code of a DLL or EXE. In order to view IL code using ILDASM , go to visual studio command prompt and run “ILDASM.EXE”. Once ILDASM is running you view the IL code.

What is a CLR?

CLR (Common language run time) is the heart of.NET framework and it does 4 primary important things:-

- Garbage collection
- CAS (Code Access security)
- CV (Code verification)
- IL to Native translation.

```
Note: - There are many other uses of CLR but I have kept it short from the
interview point of view. In the further section we will go in depth of these
questions.
```

What is the difference between managed and unmanaged code?

Code that executes under CLR execution environment is called as managed code. Unmanaged code executes outside CLR boundary. Unmanaged code is nothing but code written in C++ , VB6 , VC++ etc. Unmanaged codes have their own environment in which the code runs and it's completely outside the control of CLR.

What is a garbage collector?

Garbage collector is a feature of CLR which cleans unused managed (it does not clean unmanaged objects) objects and reclaims memory. It's a back ground thread which runs continuously checks if there are any unused objects whose memory can be claimed.

```
Note: - GC does not claim memory of unmanaged objects.
```

```
Note: - Garbage collector is one of the very important interview topics due
to complexity of generations, double GC loop because of destructor and the
implementation of finalize and dispose pattern. So please do go through the
video of "What is Garbage collection, Generation, Finalize, Dispose and
Idisposable?" to ensure that you understand the fundamentals well.
```

What are generations in Garbage collector (Gen 0, 1 and 2)?

Generations defines age of the object. There are three generations:-

- **Gen 0**:- When application creates fresh objects they are marked as Gen 0.
- **Gen 1**:- When GC is not able to clear the objects from Gen 0 in first round it moves them to Gen 1 bucket.
- **Gen 2**:- When GC visits Gen 1 objects and he is not able to clear them he moves them gen 2.

Generations are created to improve GC performance. Garbage collector will spend more time on Gen 0 objects rather than Gen 1 and Gen 2 thus improving performance.

```
Note: - More the objects in Gen 0, more your application is stable.
```

Garbage collector cleans managed code, howdo we clean unmanaged code?

Garbage collector only claims managed code memory. For unmanaged code you need to put clean up in destructor / finalize.

But when we create a destructor the performance falls down?

Yes, when we define a destructor, garbage collector does not collect these objects in the first round. It moves them to Gen 1 and then reclaims these objects in the next cycle.

As more objects are created in Gen 1 the performance of the application falls down because more memory is consumed.

So how can we clean unmanaged objects and also maintain performance?

We need to follow the below steps:-

- Implement IDisposable interface and implement the dispose function.
- In Dispose function calls the "GC.SuppressFinalize" method.
- At the client side ensure that the "Dispose" function is called when the object is no more required.

Below goes the code, this is also called as "Finalize and Dispose pattern". This ensures that your objects are created in Generation 0 rather than Generation 1. "GC.SuppressFinalize" tells the garbage collector to not worry about destructor and destroy the objects in the first call itself.

```
class clsMyClass : IDisposable
    {
        ~clsMyClass()
        {
            // In case the client forgets to call
            // Dispose , destructor will be invoked for
            Dispose(false);
        }
        protected virtual void Dispose(bool disposing)
        {
            if (disposing)
            {
                // Free managed objects.
            }
            // Free unmanaged objects
        }

        public void Dispose()
        {
            Dispose(true);
            // Ensure that the destructor is not called
            GC.SuppressFinalize(this);
        }
    }
```

```
Note :- Please do go through the videos of "What is IDisposable interface &
finalize dispose pattern in GC?" in which we have actually showed how
generation performance increases by using Finalize and Dispose pattern.
```

Can we force garbage collector to run?

“System.GC.Collect ()” forces garbage collector to run. This is not a recommended practice but can be used if situations arise.

What is the difference between finalize and dispose?

- Finalize is a destructor and dispose is a function which is implemented via ‘Idisposable’ interface.
- Finalize is nondeterministic, since it’s called by garbage collector. Dispose is a function and needs to be called by the client for clean up. In other finalize is automatically called by garbage collector while dispose needs to be called forcefully.

```
Note: -As a good practice Finalize and dispose is used collectively because
of double garbage collector loop. You can talk about this small note after
you talk about the above 2 differences.
```

What is CTS?

In .NET there are lots of languages like C#, VB.NET, VF.NET etc. There can be situations when we want code in one language to be called in other language. In order to ensure smooth communication between these languages the most important thing is that they should have a common type system. CTS (Common types system) ensures that data types defined in two different languages get compiled to a common data type.

So “Integer” data type in VB6 and “int” data type in C++ will be converted to System.int32, which is data type of CTS.

```
Note: If you know COM programming, you would know how difficult it is to
interface VB6 application with VC++ application. As datatype of both
languages did not have a common ground where they can come and interface, by
having CTS interfacing is smooth.
```

What is a CLS (Common Language Specification)?

CLS is a subset of CTS. CLS is a specification or set of rules or guidelines. When any programming language adheres to these set of rules it can be consumed by any .NET language.

For instance one of the rule which makes your application CLS non-compliant is when you declare your methods members with same name and with only case differences in C#. You can try this create a simple class in C# with same name with only case differences and try to consume the same in VB.NET ,it will not work.

What is an Assembly?

Assembly is unit of deployment like EXE or a DLL.

What are the different types of Assembly?

There are two types of assembly Private and Public assembly. A private assembly is normally used by a single application, and is stored in the application's directory, or a sub-directory beneath. A shared assembly is stored in the global assembly cache, which is a repository of assemblies maintained by the .NET runtime.

Shared assemblies are needed when we want the same assembly to be shared by various applications in the same computer.

What is Namespace?

Namespace does two basic functionalities:-

- It logically groups classes, for instance System.Web.UI namespace logically groups UI related features like textboxes, list control etc.
- In Object Oriented world, many times it is possible that programmers will use the same class name. Qualifying NameSpace with class names avoids this collision.

What is Difference between NameSpace and Assembly?

Following are the differences between namespace and assembly:

- Assembly is physical grouping of logical units, Namespace, logically groupsclasses.
- Namespace can span multiple assemblies while assembly is a physical unit like EXE , DLL etc.

What is ILDASM?

ILDASM is a simple tool which helps you to view IL code of a DLL or EXE. In order to view IL code using ILDASM , go to visual studio command prompt and run "ILDASM.EXE". Once ILDASM is running you view the IL code.

What is Manifest?

Assembly metadata is stored in Manifest. Manifest contains metadata which describes the following things -:

- Version of assembly.
- Security identity.
- Scope of the assembly.
- Resolve references to resources and classes.

The assembly manifest is stored in the DLL itself.

Where is the version information stored of an assembly?

Version information is stored in assembly inside the manifest.

Is versioning applicable to private assemblies?

Yes, versioning is applicable to private assemblies also.

What is the use of strong names?

Note:- This question can also be asked in two different ways , what are weak references and strong reference or how do you strong name a .NET assembly.

When we talk about .NET application it has two parts one is the class library or the DLL and the other the consumer like windows UI etc using this DLL.

If the consumer identifies the DLL library by namespace and class names it's called as weak reference. It's very much possible in deployment environment someone can delete the original class library and fake a similar class library with the same class name and namespace name.

Strong name is a unique name which is produced by the combination of version number, culture information,public key and digital signature. No one can fake this identity or generate the same name.

So your consumer or UI will refer the class library with strong names rather than class and namespace names. In order to create strong name, right click on the class library, click on properties, click on signing tab and click on the new menu to generate strong names as shown in the below figure.

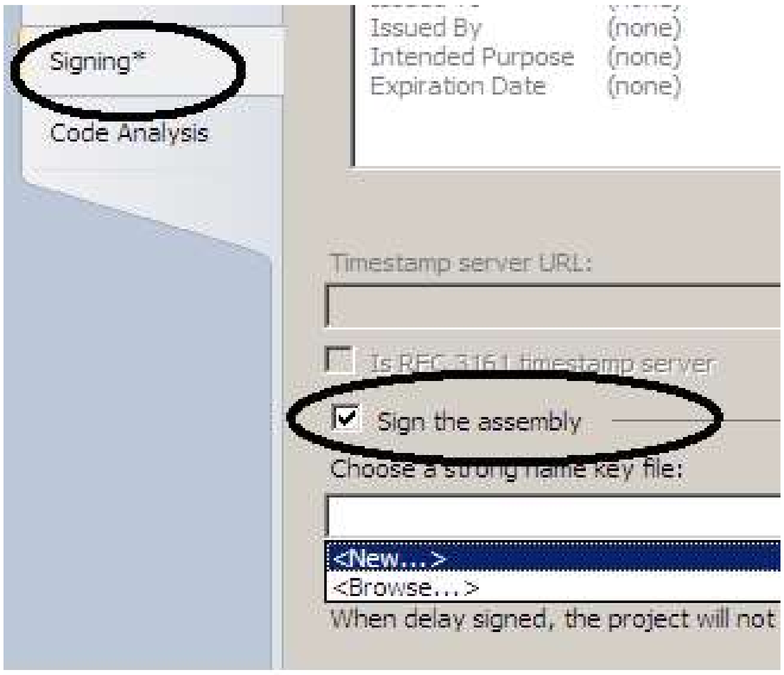


Figure 2.1: - Strong Names

What is Delay signing?

The whole point about strong names is to ensure that the clients (UI , External components etc) who is consuming the DLL knows that the DLL was published from a valid source. This authenticity is verified by using strong names. Strong name protection is good from external hackers but what if your own developers think of doing something mischievous.

That's where delay signing helps. The strong name key has two keys public key and private key. You only share the public key with your developers so that they can work seamlessly. The private key is stored in a secured location and when the DLL is about to be deployed on production the key is injected for further security.

What is GAC?

GAC (Global Assembly Cache) is where all shared .NET assembly resides. GAC is used in the following situations:-

- If the application has to be shared among several applicationwhich is in the same computer.
- If the assembly has some special security, requirements like only administrators can remove the assembly. If the assembly is private then a simple delete of assembly the assembly file will remove the assembly.

How to add and remove an assembly from GAC?

You can use the 'GacUtil' tool which comes with visual studio. So to register an assembly in to GAC go to "Visual Studio Command Prompt" and type "gacutil –i (assembly name)", where (assembly name) is the DLL name of the project.

One you have installed the assembly the DLL can be seen in 'c:\windows\assembly\' folder.

When we have many DLL's to be deployed we need to create setup and deployment package using windows installer. So the common way of deploying GAC DLL in production is by using windows installer.

If we have two versions of the same assembly in GAC how to we make a choice?

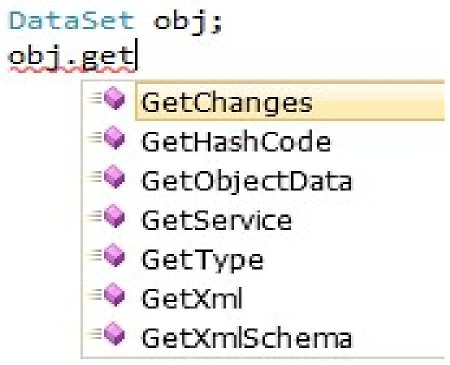
When we have two version of the same assembly in GAC we need to use binding redirect tag and specify the version we want to use in the new version property as shown in the below "app.config" file.

```
<configuration>
<runtime>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<dependentAssembly>
<assemblyIdentity name="ComputerName" publicKeyToken="cfc68d722cd6a164" />
<publisherPolicy apply="yes" />
<bindingRedirect oldVersion="1.1.0.0" newVersion="1.0.0.0" />
</dependentAssembly>
</assemblyBinding>
</runtime>
</configuration>
```

To get LIVE training, new topic release video updates install Telegram app & join us using - https://tinyurl.com/QuestPondChannel

What is Reflection and why we need it?

Reflection is needed when you want to **determine / inspect contents of an assembly.** For example look at your visual studio editor intellisense, when you type "." (dot) before any object , it gives you all members of the object. This is possible because of reflection.



```
DataSet obj;
obj.get
```

Figure 2.2:- Reflection

Reflection also goes one step further; it can also invoke a member which is inspected. For instance if the reflection detects that there is a method called as "GetChanges" in an object. We can get a reference to that method instance and invoke the same on runtime.
In simple words reflection passes through two steps "Inspect" and "Invoke" (optional). "Invoke" process is optional.

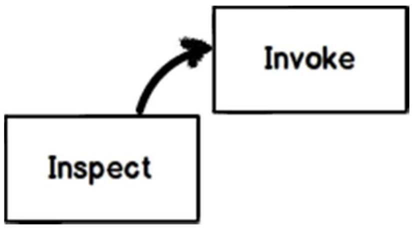


Figure 2.3:- Invoke

How do we implement reflection?

Implementing reflection in c# is a two step process , 1st get the "type" of the object and then use the type to browse members like "methods" , "properties" etc.

Step 1:- The first step is to get the type of the object. So for example you have a DLL called as "ClassLibrary1.dll" which has a class called as "Class1". We can use the "Assembly" (belongs to System.Reflection namespace) class to get a reference to the type of the object. Later we can use "Activator.CreateInstance" to create an instance of the class. The "GetType()" function helps us to get reference to the type of the object.

```
var myAssembly = Assembly.LoadFile(@"C:\ClassLibrary1.dll");
```

To get LIVE training, new topic release video updates install Telegram app & join us using - https://tinyurl.com/QuestPondChannel

```
var myType = myAssembly.GetType("ClassLibrary1.Class1");
dynamic objMyClass = Activator.CreateInstance(myType);
// Get the class type
Type parameterType = objMyClass.GetType();
```

Step 2 :- Once we have reference the type of the object we can then call "GetMembers" or "GetProperties" to browse throw the methods and properties of the class.

```
// Browse through members
foreach (MemberInfo objMemberInfo in parameterType.GetMembers())
{Console.WriteLine(objMemberInfo.Name);}
```

```
// Browse through properties.
foreach (PropertyInfo objPropertyInfo in parameterType.GetProperties())
{Console.WriteLine(objPropertyInfo.Name);}
```

In case you want to invoke the member which you have inspected you can use "InvokeMember" to invoke the method. Below is the code for the same.

```
parameterType.InvokeMember("Display",BindingFlags.Public |
BindingFlags.NonPublic | BindingFlags.InvokeMethod |
BindingFlags.Instance,null, objMyClass, null);
```

What are the practical uses of reflection?

- If you are creating application like visual studio editors where you want show internal of an object by using intellisense.
- In unit testing sometimes we need to invoke private methods. That's a different thing test private members are proper or not.
- Sometimes we would like to dump properties, methods and assembly references to a file or probably show it on a screen.

What is the use of Dynamic keyword?

Programming languages can be divided in to two categories strongly typed and dynamically typed. Strongly typed languages are those where the checks happen during compile time while dynamic languages are those where type checks are bypassed during compile time. In dynamic language object types are known only during runtime and type checks are activated only at run time.

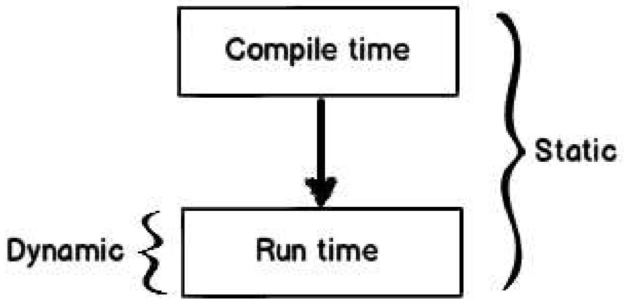
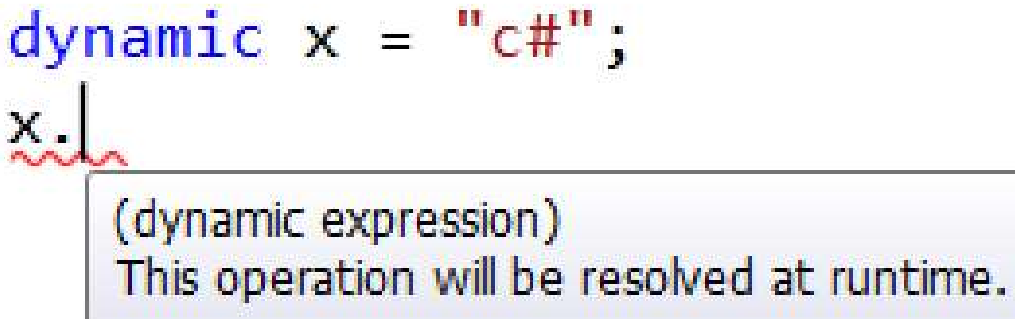


Figure 2.4:- Dynamic Keyword

So we would like to take advantage of both the world. Because many times we do not know object type until the code is executed. In other words we are looking at something like dynamically statically typed kind of environment. That's what dynamic keyword helps us with. If you create a variable using the "Dynamic" keyword and if you try to see members of that object   you will get a message as shown below "will be resolved at runtime".



```
dynamic x = "c#";
x.
```

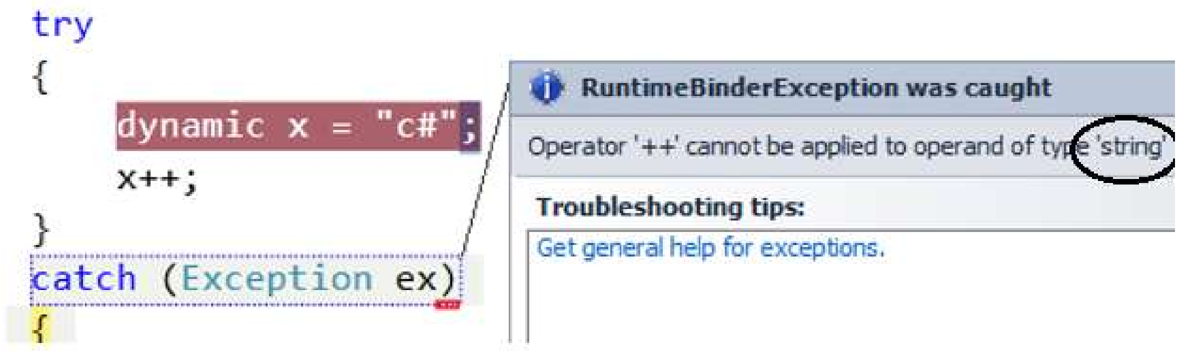
(dynamic expression)
This operation will be resolved at runtime.

Figure 2.5:- Dynamic X

Now try the below code out. In the below code I have created a dynamic variable which is initialized with a string data. And in the second line I am trying to have fun by trying to execute a numeric incremental operation. So what will happen now?.... think.

```
dynamic x = "c#";

x++;
```

Now this code will compile fine without any complains. But during runtime it will throw an exception complaining that the mathematical operations cannot be executed on the variable as it's a string type. In other words during runtime the dynamic object gets transformed from general data type to specific data type (ex: - string for the below code).



```
try
{
    dynamic x = "c#";
    x++;
}
catch (Exception ex)
{
```

RuntimeBinderException was caught

Operator '++' cannot be applied to operand of type 'string'

Troubleshooting tips:

Get general help for exceptions.

Figure 2.6:- String

What are practical uses of Dynamic keyword?

One of the biggest practical uses of dynamic keyword is when we operate on MS office components via interop.
So for example if we are accessing Microsoft excel components without dynamic keyword , you can see how complicated the below code is. Lots of casting happening in the below code, right.

```
// Before the introduction of dynamic.
Application excelApplication = new  Application();
((Excel.Range)excelApp.Cells[1, 1]).Value2 = "Name";
Excel.Range range2008 = (Excel.Range)excelApp.Cells[1, 1];
```

Now look at how simple the code becomes by using the dynamic keyword. No casting needed and during runtime type checking also happens.

```
// After the introduction of dynamic, the access to the Value property and
// the conversion to Excel.Range are handled by the run-time COM binder.
dynamic excelApp = new Application();
excelApp.Cells[1, 1].Value = "Name";
Excel.Range range2010 = excelApp.Cells[1, 1];
```

What is the difference between Reflection and Dynamic?

- Both reflection and dynamic are used when we want to operate on an object during runtime.
- Reflection is used to inspect meta-data of an object. It also has the ability to invoke members of an object on runtime.

- Dynamic is a keyword which was introduced in .NET 4.0. It evaluates object calls during runtime. So until the method calls are made compiler is least bothered if those methods / properties etc exist or not.
- Dynamic uses reflection internally. It caches the method calls made thus improving performance to a certain extent.
- Reflection can invoke both public and private members of an object while dynamic can only invoke public members.

Below is the detail comparison table which shows in which scenario they are suited.

	Reflection	**Dynamic**
Inspect (Meta-data)	Yes	No
Invoke Public members	Yes	Yes
Invoke Private members	Yes	No
Caching	No	Yes

Below is a simple diagram which summarizes visually what reflection can do and what dynamic keyword can do?

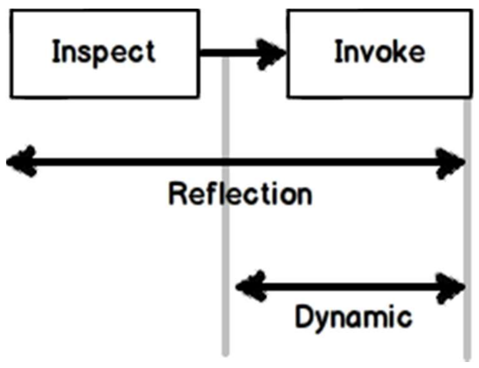


Figure 2.7:- Reflection

Explain the difference between early binding and late binding?

Early binding or early bound means the target methods, properties and functions are detected and checked during compile time. If the method / function or property does not existor has a data type issues then the compiler throws an exception during compile time.

Late binding is opposite of early binding. Methods, properties and functions are detected during runtime rather than compile time. So if the method, function or property does not exist during runtime application will throw an exception.

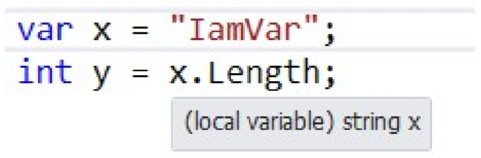
```
Note :- In simple words everything is early binded until you are using
reflection or dynamic keyword.
```

What is the difference between VAR and Dynamic keyword?

```
Note :- Comparing VAR and Dynamic keyword is like comparing Apples and
oranges. Many people confuse VAR with the variant datatype of VB6 and there's
where interviewer tries to confuse you. But there is absolute no connection
of var c# keyword with variant of VB6.
```

VAR is early binded (statically checked) while DYNAMIC is late binded (dynamically evaluated).

VAR keyword looks at your right hand side data and then during compile time it decides the left hand side data type.In other words VAR keyword just saves you typing lot of things.



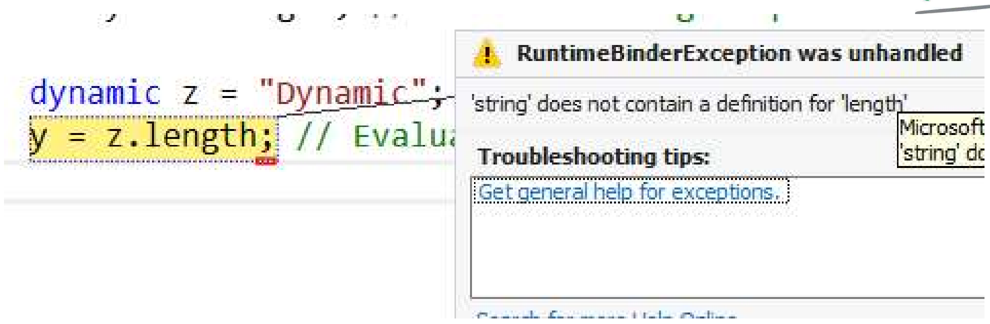
```
var x = "IamVar";
int y = x.Length;
```

Figure 2.8:- VAR

On the other hand dynamic keyword is for completely different purpose. Dynamic objects are evaluated during runtime. For instance in the below code the "Length" property exists or not is evaluated during runtime.

```
dynamic z = "Dynamic";
y = z.Length; // Is Length present or not checked during runtime
```

If for some reason you mistype the "Length" property to "length" (Small "l") you would end up in to a runtime exception as shown in the below figure.



```
dynamic z = "Dynamic";
y = z.length; // Evalu
```

RuntimeBinderException was unhandled

'string' does not contain a definition for 'length'

Troubleshooting tips:

Get general help for exceptions.

Figure 2.9:- Length

So in short sentence VAR does static check while DYNAMIC evaluates and checks during runtime.

Explain term type safety and casting in C#?

There are two types of languages one which is type safe and one which is not. Type safety means preventing type errors. Type error occurs when data type of one type is assigned to other type **UNKNOWINGLY**and we get undesirable results.

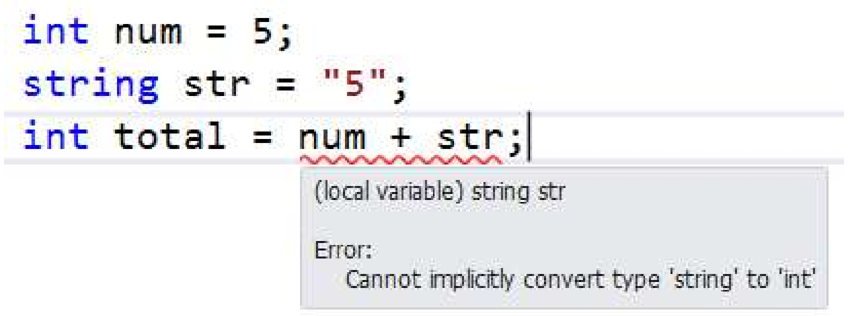
For instance JavaScript is a **NOT** a type safe language. In the below code "num" is a numeric variable and "str" is string. Javascript allows me to do "num + str" , now GUESS will it do arithmetic or concatenation .

Now for the below code the results are "55" but the important point is the confusion created what kind of operation it will do.

This is happening because javascript is not a type safe language.Its allowing to set one type of data to the other type without restrictions.

```
<script>
var num = 5; // numeric
var str = "5"; // string
var z = num + str; // arthimetic or concat ????
alert(z); // displays  "55"
</script>
```

C# is a type safe language. It does not allow one data type to be assigned to other data type. The below code does not allow "+" operator on different data types.



```
int num = 5;
string str = "5";
int total = num + str;
```

(local variable) string str

Error:
    Cannot implicitly convert type 'string' to 'int'

Figure 2.10:- Type Safe

But in certain scenarios we would like to convert the data type to achieve the given results that's where we need to do conversion or casting. The next question talks about the same in detail.

Explain casting, implicit conversion and explicit conversion?

```
Note :- Some of  interviewers term this conversion as casting and some just
prefer to call it as conversion , at the end of the day they mean the same
thing.
```

To understand implicit and explicit conversion of data types first let's try to understand type casting concept. Type casting is a mechanism where we convert one type of data to other type.

For example below is simple code where we want to move integer (value without decimals) value to a double (value with decimals).   When we try to move double data type to integer data type casting occurs. Casting is also termed as conversion.

```
int i = 100;
double d = 0;
d = i; // casting
```

In the above example there is no loss of data. In other words when we moved 100 integer value to double we get the 100 value as it. This is termed as implicit conversion / casting.

Now consider the below code where we are trying to move a double to an integer. In this scenario in integer only 100 will be received decimals will be removed. You can also see I need to explicitly specify that we need to convert in to integer. This is termed as explicit conversion.

```
double d = 100.23;
int i = 0;
i = (int) d; // this will have 100, 0.23 will truncated.
```

To get LIVE training, new topic release video updates install Telegram app & join us using - https://tinyurl.com/QuestPondChannel

What are stack and heap?

Stack and heap are memory types in an application. Stack memory stores data types like int , double , Boolean etc. While heap stores data types like string and objects.

For instance when the below code run the first two variables i.e. “i” and “y” are stored in a stack and the last variable “o” is stored in heap.

```
void MyFunction()
{
int i = 1; // This is stored in stack.
int y = i; // This is stored in stack.
object o = null; // This is stored in heap.
} //  after this end the stack variable memory space is reclaimed while //
the heap memory is reclaimed later by garbage collector.
```

What are Value types and Reference types?

Value types contains actual data while reference types contain pointers and the pointers point to the actual data.

Value types are stored on stack while reference types are stored on heap. Value types are your normal data types like int , bool , double and reference types are all objects.

What is concept of Boxing and Unboxing?

When value type is moved to a reference type it’s called as boxing. The vice-versa is termed as unboxing.

Below is sample code of boxing and unboxing where integer data type is converted in to object and then vice versa.

```
int i = 1;
object obj = i;          // boxing
int j = (int) obj;       // unboxing
```

How performance is affected due to boxing and unboxing?

When boxing and unboxing happens the data needs to jump from stack memory to heap and vice-versa which is a bit of memory intensive process. As a good practice avoid boxing and unboxing where ever possible.

How can we avoid boxing and unboxing?

First thing it's very difficult to avoid boxing and unboxing. For instance most of the time you will moving data from UI objects like text boxes etc to business objects and also vice versa which will demand boxing and unboxing. Below are some key points to remember:-

- First thing is it really necessary to use boxing and unboxing. If it's unavoidable like moving data from UI text boxes to internal c# data types, then go for it.
- Try to see if you can use generics and avoid it.

If we have a class referencing value type, where is value type stored ?

The value type will be stored in a heap.

Are static variables stored on a heap or stack ?

Static variables even if its value type is stored in a heap.

How to prevent my .NET DLL to be decompiled?

By design, .NET embeds rich Meta data inside the executable code using MSIL. Any one can easily decompile your DLL back using tools like ILDASM (owned by Microsoft) or Reflector for .NET which is a third party. Secondly, there are many third party tools, which make this decompiling process a click away. So any one can easily look in to your assemblies and reverse engineer them back in to actual source code and understand some real good logic, which can make it easy to crack your application.

The process by which you can stop this reverse engineering is using "obfuscation". It is a technique, which will foil the decompilers. Many third parties (XenoCode, Demeanor for .NET) provide .NET obfuscation solution. Microsoft includes one that is Dotfuscator Community Edition with Visual Studio.NET.

Note:- We leave this as homework to reader's compile, a DLL obfuscate it using "Dotfuscator Community Edition" which comes with Visual Studio.NET and try viewing the same using ILDASM.

What is the difference between Convert.toString and .toString () method?

Just to give an understanding of what the above question means seethe below code.

```
int i =0;
```

```
MessageBox.Show(i.ToString());
MessageBox.Show(Convert.ToString(i));
```

We can convert the integer "i" using "i.ToString()" or "Convert.ToString" so what is the difference. The basic difference between them is "Convert" function handles NULLS while "i.ToString()" does not.It will throw a NULL reference exception error. So as a good coding practice using "convert" is always safe.

What is the difference between String vs string?

"String" is an alias(the same thing called with different names) of "string". So technically both the below code statements will give the same output.

```
String s = "C# interview questions";
```

or

```
string s = "C# interview questions";
```

In the same way there are aliases for other c# data type as shown below:-

- object: System.Object
- string: System.String
- bool: System.Boolean
- byte: System.Byte
- sbyte: System.SByte
- short: System.Int16
- ushort: System.UInt16
- int: System.Int32
- uint: System.UInt32
- long: System.Int64
- ulong: System.UInt64
- float: System.Single
- double: System.Double
- decimal: System.Decimal
- char: System.Char

So when both mean the same thing why are they different?

When we talk about .NET there are two different things one there is .NET framework and the other there are languages(C# , VB.NET etc) which use that framework.

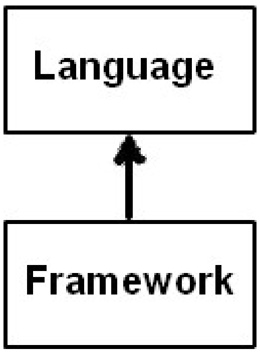


Figure 2.11:- Framework

"System.String" a.k.a "String" (capital "S") is a .NET framework data type while "string" is a C# data type.

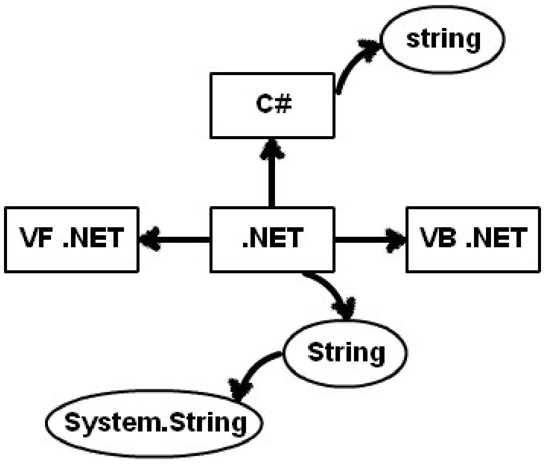


Figure 2.12:- System.String

So when to use "String" and "string"?

First thing to avoid confusion use one of them consistently. But from best practices perspective when you do variable declaration it's good to use "string" (small "s") and when you are using it as a class name then "String" (capital "S") is preferred.

In the below code the left hand side is a variable declaration and it declared using "string". At the right hand side we are calling a method so "String" is more sensible.

```
string s = String.ToUpper() ;
```

How can we handle exceptions in .NET?

Exceptions are handled by “System.Exception” base class. If you want to raise an error from source you need to create the exception object with below code snippet.

```
throw new Exception("Customer code cannot be more than 10");
```

Once the exception is raised if you want to catch the same you need to use the try catch block as shown below.

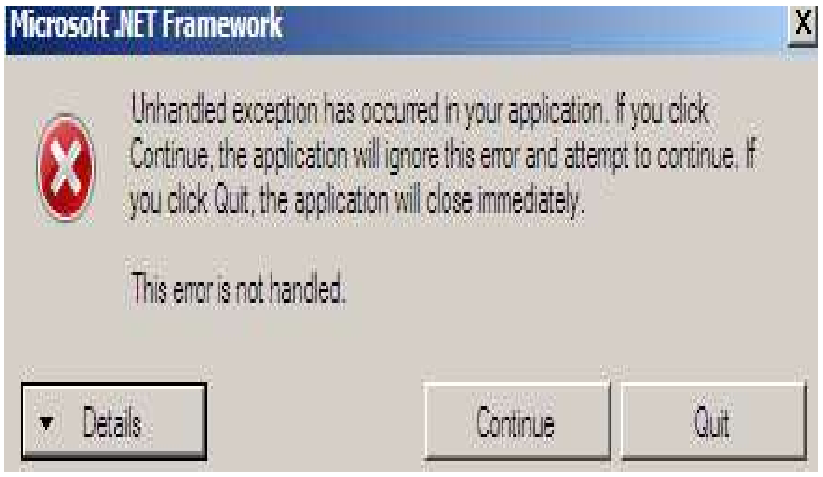
```
try
{
// This section will have the code which
// which can throw exceptions.
}
catch(Exception e)
{
// Handle what you want to
// do with the exception
label.text = e.Message;
}
```

How can I know from which source the exception occurred?

“System.Exception.StackTrace”allows you to identify series of call which lead to this exception.

What if we do not catch the exception?

If you do not catch the error, .NET framework will handle the same and you will get a message box as shown in the below figure.


Microsoft .NET Framework

Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

This error is not handled.

Figure 2.13: - .Net framework handles the errors

What are system level exceptions and application level exceptions?

Exceptions that are thrown by .NET framework are called as system exceptions. These errors are non-recoverable or fatal errors like ArgumentOutOfRangeException, IndexOutOfRangeException, StackOverflowException etc.

Application exceptions are custom exceptions created for the application. Application exceptions are created by deriving from "ApplicationException" class as shown below. You can then create the object of the below exception and throw the same from the code and catch the same on the client side.

```
public class MyOwnException : ApplicationException
{
  private string messageDetails = String.Empty;
  public DateTime ErrorTimeStamp {get; set;}
  public string CauseOfError {get; set;}

  public MyOwnException(){}
  public MyOwnException(string message,
    string cause, DateTime time)
  {
    messageDetails = message;
    CauseOfError = cause;
    ErrorTimeStamp = time;
  }

  // Override the Exception.Message property.
  public override string Message
  {
    get
    {
      return string.Format("This is my own error message");
    }
  }
}
```

Can two catch blocks be executed?

No, once the proper catch section is executed the control goes to the ‘finally’ block. So there is no chance by which multiple catch block will ‘execute’.

What are different types of collections in .NET?

There are five important collections in .NET Arrays, Lists, Hashtable , stacks and queues .

What is the difference between arraylist and list?

- Arrays are fixed in size while Arraylist is resizable.
- Arrays are strongly typed, in other words when you create an array it can store only one data type data. Arraylist can store any datatype.

Are Arraylist faster or Arrays?

Array list takes any data type which leads to boxing and unboxing. As arrays are strongly typed they do not do boxing and unboxing. So arrays are faster as compared to array list.

```
// Array definition
int[] str = new int[10];

// Array list definition
ArrayList MyList = new ArrayList();
```

What are hashtable collections?

In arraylist or array if we have to access any data we need to use the internal index id generated by the array list collection. For instance the below code snippet shows how the internal id is used to fetch data from array list.

In actual scenarios we hardly remember internal id's generated by collection we would like to fetch the data by using some application defined key. There's where hash table comes in to picture.

```
string str = MyList[1].ToString();
```

Hash table helps to locate data using keys as shown below. When we add data to hash table it also has a provision where we can add key with the data. This key will help us to fetch data later using key rather than using internal index id's generated by collections.

```
objHashtable.Add("p001","MyData");
```

This key is converted in to numeric hash value which is mapped with the key for quick lookup.

What are Queues and stack collection?

Queues are collection which helps us to add object and retrieve them in the manner they were added. In other word queues helps us to achieve the first in first out collection behavior.

Stack collection helps us to achieve first in last out behavior.

Can you explain generics in .NET?

Generics help to separate logic and data type to increase reusability. In other words you can create a class whose data type can be defined on run time.

For instance below is a simple class "class1" with a "compareme" function created using generics. You can see how the unknown data type is put in greater than and less than symbol. Even the compare me method has the unknown data type attached.

```
public class Class1<UNNKOWDATATYPE>
    {

        public bool Compareme(UNNKOWDATATYPE v1, UNNKOWDATATYPE v2)
        {

            if (v1.Equals(v2))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

During runtime you can define the datatype as shown in the below code snippet.

```
Class1<int> obj = new Class1<int>();
 bool b = obj.Compareme(1,2); // This compares numeric
Class1<string> obj1 = new Class1<string>();
bool b1 = obj1.Compareme("shiv","shiv"); // This does string comparison
```

Explain generic constraints and when should we use them?

Generic's helps to decouple logic from the data type. But many times some logics are very specific to specific data types.

```
public class CompareNumeric<UNNKOWDATATYPE>
{
        public bool Compareme(UNNKOWDATATYPE v1, UNNKOWDATATYPE v2)
        {
            if (v1>v2)
            {return true;}
            else
{return false;}
        }
}
```

For example above is a simple generic class which does comparison if one number is greater than other number.Now the greater and less than comparison is very specific to numeric data types. These kind of comparison's cannot be done on non-numeric types like string.

So if some uses the classes with "int" type its perfectly valid.

To get LIVE training, new topic release video updates install Telegram app & join us using - https://tinyurl.com/QuestPondChannel

```
CompareNumeric<int> obj = new CompareNumeric<int>();
bool boolgreater = obj.Compare(10,20);
```

If someone uses it with "double" data type again perfectly valid.

```
CompareNumeric<double> obj = new CompareNumeric<double>();
bool boolgreater = obj.Compare(100.23,20.45);
```

But using string data type with this logic will lead undesirable results. So we would like to restrict or put a constraint on what kind of types can be attached to a generic class this is achieved by using "generic constraints".

```
CompareNumeric<string> obj = new CompareNumeric<string>();
bool boolgreater = obj.Compare("interview","interviewer");
```

Generic type can be restricted by specifying data type using the "WHERE" keyword after the generic class as shown in the below code. Now if any client tries to attach "string" data type with the below class it will not allow, thus avoiding undesirable results.

```
public class CompareNumeric<UNNKOWDATATYPE> where UNNKOWDATATYPE : int,
double
{


}
```

Can you explain the concept of generic collection?

Array List, Stack and Queues provide collections which are not type safe. This leads 2 problems first it's not type safe and second it leads to boxing and unboxing.

By using generics we can have type safety and also we can avoid boxing and unboxing. Below is a simple code snippet which shows a strong typed list of type integer and string.

```
List<int> obj;
obj.add(1); // you can only add integers to this list
List<string> obj;
obj.add("shiv"); // you can only add string to this list
```

What is the difference between dictionary and hashtable?

Dictionary collection is a generic collection equivalent for hashtable. Hashtable allows you to add key and value of any type (i.e. objects) . This leads to two problems one is boxing and unboxing issues and second it's not strongly typed.

```
// Creates a strongly typed dictionary with integer key and value
// pair
Dictionary<int,int> obj = new Dictionary<int,int>();
// We can only add integer value and key to the dictionary collection
Obj.Add(123,550);
```

What are the generic equivalent for array list,stack, queues and hashtable?

Below are the listed generic equivalents for each of them:-

- Array list generic equivalent is List<int>.
- Stack generic equivalent is Stack<int>.
- Queue generic equivalent is Queue<int>.
- Hashtable generic equivalent is Dictionary<int,int>.

What is the use of IEnumerable, ICollection, Ilist and IDictionary?

The above 4 entities are nothing but interfaces implemented by collections:-

IEnumerable: - All collection classes use this interface and it helps to iterate through all the collection elements.

ICollection: - This interface has the count property and it's implemented by all collection elements.

IList: -This interface allows random access, insertion and deletion in to the collection.Itsimplemented by array list.

IDictionary: -This interface is implemented by hashtable and dictionary.

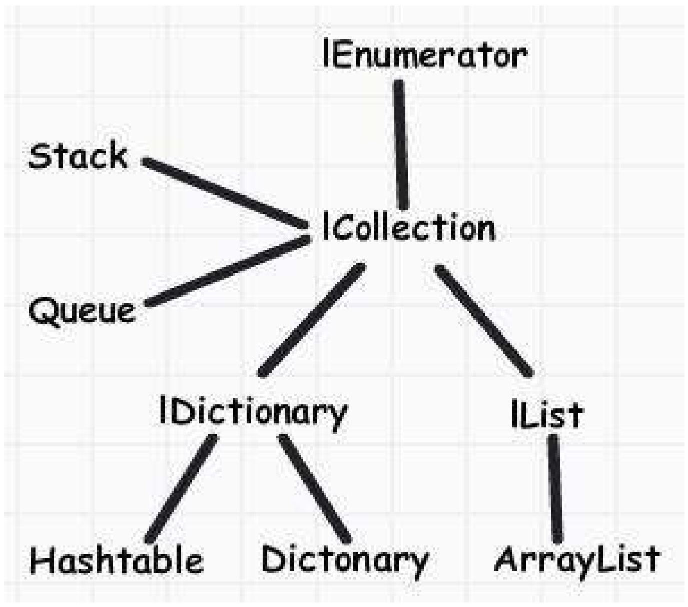


Figure 2.14: - Interface collections

Differentiate IEnumerable vs Iqueryable?

The first important point to remember is "Iqueryable" interface inherits from "Ienumerable", so whatever "Ienumerable" can do, "Iqueryable" can also do.

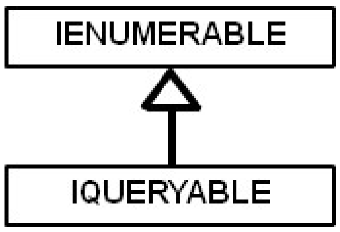


Figure 2.15: - IEnumerable

There are many differences but let us discuss about the one big difference which makes the biggest difference. "IQueryable" interface is useful when your collection is loaded using LINQ or Entity framework and you want to apply filter on the collection.

Consider the below simple code which uses "IEnumerable" with entity framework. It's using a "where" filter to get records whose "EmpId" is "2".

```
EmpEntities ent = new EmpEntities();

IEnumerable<Employee> emp = ent.Employees;

IEnumerable<Employee> temp = emp.Where(x => x.Empid == 2).ToList<Employee>();
```

This where filter is executed on the client side where the "IEnumerable" code is. In other words all the data is fetched from the database and then at the client its scans and gets the record with "EmpId" is "2".

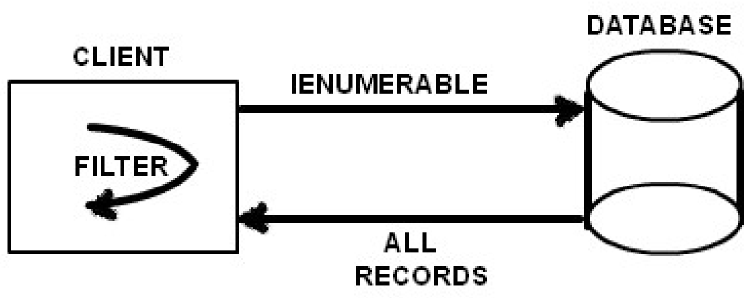


Figure 2.16: - All Recods

But now see the below code we have changed "IEnumerable" to "IQueryable".

```
EmpEntities ent = new EmpEntities();
IQueryable<Employee> emp = ent.Employees;
IEnumerable<Employee> temp = emp.Where(x => x.Empid == 2).ToList<Employee>();
```

In this case the filter is applied on the database using the "SQL" query. So the client sends a request and on the server side a select query is fired on the database and only necessary data is returned.

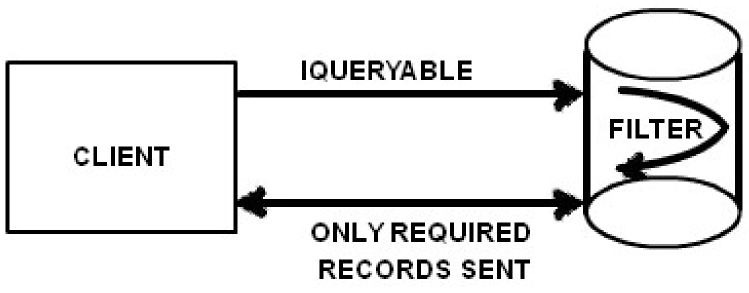


Figure 2.17: - Required Record

So the difference between "IQueryable" and "IEnumerable" is about where the filter logic is executed. One executes on the client side and the other executes on the database.

So if you working with only in-memory data collection "IEnumerable" is a good choice but if you want to query data collection which is connected with database "IQueryable" is a better choice as it reduces network traffic and uses the power of SQL language.

What is code access security (CAS)?

CAS is the part of .NET security model which determines whether or not a particular code is allowed to run and what kind of resources can the code access.

So how does CAS actually work?

It's a four step process:-

- First Evidence is gathered about the assembly. In other words from where did this assembly come? , who is the publisher etc.
- Depending on evidences the assembly is assigned to a code group. In other words what rights does the assembly depending on the evidence gathered.
- Depending on code group security rights are allocated.
- Using the security rights the assembly is run with in those rights

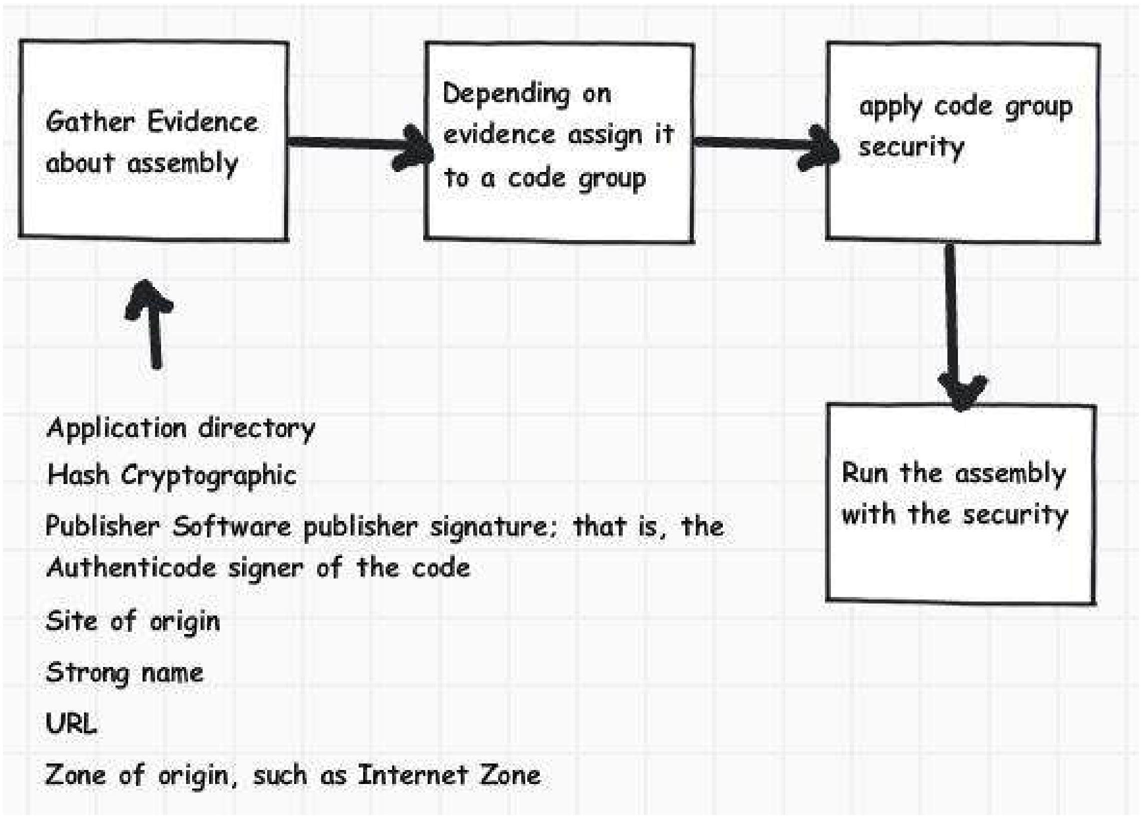

Gather Evidence about assembly
Depending on evidence assign it to a code group
apply code group security
Run the assembly with the security
Application directory
Hash Cryptographic
Publisher Software publisher signature; that is, the Authenticode signer of the code
Site of origin
Strong name
URL
Zone of origin, such as Internet Zone

Figure 2.18: - CAS in action

Is CAS supported in .NET 4.0?

CAS is deprecated in .NET 4.0 and two major changes are brought in:-

- Permission granting is no more the work of CAS; it's now the work of the hosting model. In other words CAS is disabled in .NET 4.0 by default. The host will decide what rights to be given to the .NET assembly.
- A new security model i.e. Security transparent model is introduced. The security transparent model puts code in to separate compartments/ boxes as per the risk associated. If you know a code can do something wrong you can compartmentalizethe code as 'Security transparent' and if you have a code which you trust you can box them in to 'Security critical'.

What is sandboxing?

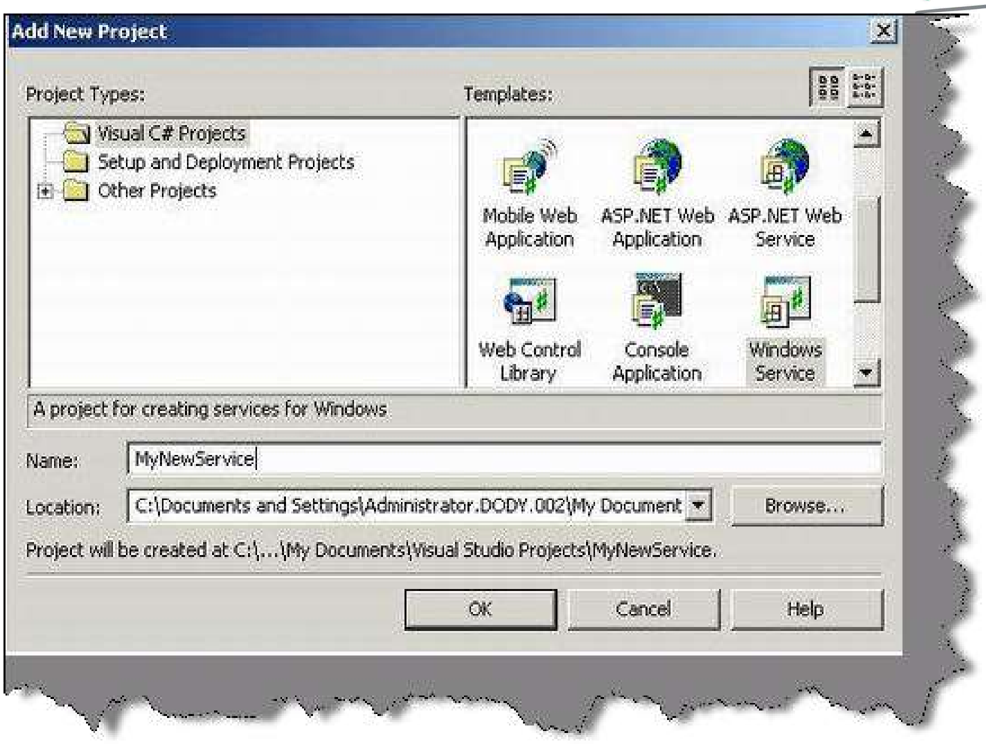
If you have want to execute an untrusted third party DLL, you can create your own 'appdomain' and assign permission sets so that your 3rd party DLL runs under a control environment.

How can we create a windows service using .NET?

Windows Services are long-running processes that runs at the background.It has the ability to start automatically when the computer boots and also can be manually paused, stopped or even restarted.

Following are the steps to create a service:-

Create a project of type "Windows Service".


Add New Project

Project Types:
Visual C# Projects
Setup and Deployment Projects
Other Projects

Templates:
Mobile Web Application
ASP.NET Web Application
ASP.NET Web Service
Web Control Library
Console Application
Windows Service

A project for creating services for Windows

Name: MyNewService

Location: C:\Documents and Settings\Administrator.DODY.002\My Document

Project will be created at C:\...\My Documents\Visual Studio Projects\MyNewService.

Figure 2.19:- Windows service

If you see, the class created it is automatically inheriting from "System.ServiceProcess.ServiceBase".

You can override the following events provided by service and write your custom code. All the three main events can be used that is Start, stop and continue.

```
protected override void OnStart(string[] args)
{
}
protected override void OnStop()
{
}
protected override void OnContinue()
{
}
```

Now to install the service you need to do run the install util exe.

```
InstallUtil <Project Path>\BIN\MyNewService.exe
```

What is serialization and deserialization in .NET?

Serialization is a process where we can convert an object state in to stream of bytes. This stream can then be persisted in a file,database, or sent over a network etc. Deserialization is just vice-versa of serialization where we convert stream of bytes back to the original object.

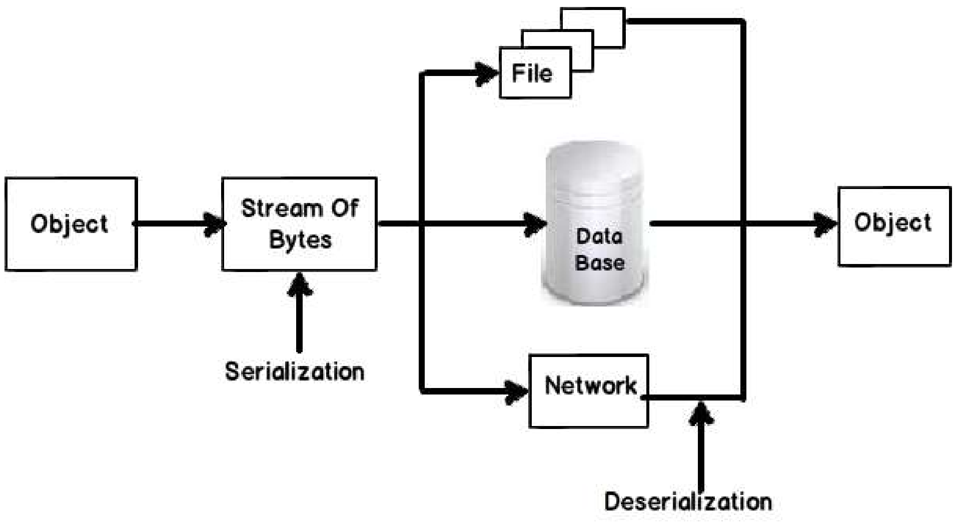


Figure 2.20: - serialization and deserialization in .NET

Below is a simple code of how to serialize and de-serialize an object.

Let's first start with serialization.

Step 1:- Create the object and put some values

```
// Serialization
Customer obj = new Customer(); // Create object
obj.CustomerCode = 123; // Set some values
```

Step 2:- Create the file where to save the object.

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

IFormatter i = new BinaryFormatter(); // Use the binary formatter
Stream stream = new FileStream("MyFile.txt", FileMode.Create,
FileAccess.Write, FileShare.None); // Give file name
```

Step 3:- Use serialize method to save it to hard disk

```
i.Serialize(stream, o); // write it to the file
stream.Close(); // Close the stream
```

Let's also see a simple example of de-serialization.

Step 1:- Read the file

```
// De-Serialization
IFormatter formatter = new BinaryFormatter(); // Use binary formatter
Stream stream = new FileStream("MyFile.txt", FileMode.Open, FileAccess.Read,
FileShare.Read); // read the file
```

Step 2:- Recreate it back to the original object.

```
Customer obj = (Customer)formatter.Deserialize(stream); // take data back to
object
stream.Close(); // close the stream
```

If you want to save to XML or any other content type use the appropriate formatter.

Can you mention some scenarios where we can use serialization?

Below are some scenarios where serialization is needed:-

- Passing .NET objects across network. For example .NET remoting , web services or WCF services use serialization internally.
- Copy paste .NET objects on clip board.
- Saving old state of the object and reverting back when needed. For example when the user hits the cancel button you would like to revert back to the previous state.

When should we use binary serialization as compared to XML serialization?

- Binary is smaller in size, so faster to send across network and also faster to process.
- XML is more verbose, but easy to understand and human readable. But due to the XMLstructure its complex to parse and can impact performance.

```
Note :- Many people answer that XML serialization should be used when you
have different platforms and binary serialization should be used when you
```

```
same platforms. But this answer is actually not right as we have lot of
binary serialization methodology like ASN , protbuf which are cross platform
and widely accepted. So the important difference is that one is XML which
readable by eyes and the binary is not.
```

What is Regular expression?

Regular expression is a pattern matching technique. Most of the data follow patterns, some examples of common data patterns are as shown in the below table :-

Email Address	xyz@pqr.com abc@lmn.com	Email address data starts with some characters followed by “@” and then a “.” and finally the domain extension.
Phone number	901-9090-909023 909-999-1202920	Phone number data starts with an international code followed by region code and then the phone number.

By using regular expression you can represent these data patterns and utilize the same for validations and manipulation of data.

In order to use regular expression we need to create the object of "Regex" class and pass the pattern to Regex. For example the below pattern represent's data pattern of 10 character length which can be any small characters from a to z.

```
Regex obj = new Regex("[a-z]{10}");
```

Once you have specified the pattern as described in the previous step you can then use the "IsMatch" function to check if the data matches with the pattern.

```
if(obj.IsMatch("shivkoirala"))
{
// proper data
}
else
{
// improper data
}
```

```
Note :- In some rare cases we have seen interviewers ask to write down regex
patterns. So we would suggest to see the video "Explain Regex ?"  provided in
the DVD which will help you to write down any regex pattern easily.
```

What is time out support in regex (regular expression)?

This is a new feature of .NET 4.5.Some of the regular expressions are very complex and they can take lot of time to evaluate.
For instance below is a simple regular expression.

```
var regEx = new Regex(@"^(\d+)+$", RegexOptions.Singleline);
```

If someone inputs a huge number as shown in the below code snippet. It will take more than a minute to resolve the expression leading to lot of load on the application. So we would like to give a time out on the expression. So if the validation takes more than a specific interval we would like the application to give up and move ahead.

```
var match = regEx.Match("123453109839109283090492309480329489812093809x");
```

In .NET 4.5 we can now provide the timeout value as parameter on the constructor. For instance in the below code we have provided 2 seconds timeout on the regex (regular expression). So if it exceeds more than 2 second "regexMatchTimeOutException" will occur.

```
try
{
var regEx = new Regex(@"^(\d+)+$", RegexOptions.Singleline,
TimeSpan.FromSeconds(2));
var match = regEx.Match("123453109839109283090492309480329489812093809x");
}


catch (RegexMatchTimeoutException ex)
{
 Console.WriteLine("Regex Timeout");
}
```

Can you explain the concept of “Short Circuiting”?

Short circuiting occurs when you do logical operations like ‘AND’ and ‘OR’.

“When we use short circuit operators only necessary evaluation is done rather than full evaluation.”

Let us try to understand the above sentence with a proper example. Consider a simple “AND” condition code as shown below. Please note we have only one “&” operator in the below code.

```
if(Condition1 & Condition2)
{
}
```

In the above case “Condition2” will be evaluated even if “Condition1” is “false”. Now if you think logically, it does not make sense to evaluate “Condition 2”, if “Condition 1” is false .It’s a AND condition right? , so if the first condition is false it means the complete AND condition is false and it makes no sense to evaluate “Condition2”.

There’s where we can use short circuit operator “&&”. For the below code “Condition 2” will be evaluated only when “Condition 1” is true.

```
if(Condition1 && Condition2)
{
}
```

The same applies for “OR” operation. For the below code (please note its only single pipe (“|”).) “Condition2” will be evaluated even if “Condition1” is “true”. If you think logically we do not need to evaluate “Condition2” if “Condition1” is “true”.

```
if(Condition1 | Condition2)
{
}
```

So if we change the same to double pipe (“||”) i.e. implement short circuit operators as shown in the below code, “Condition2” will be evaluated only if “Condition1” is “false”.

```
if(Condition1 || Condition2)
{
}
```

What is the difference between “Typeof” and “GetType” ?

Both “TypeOf” and “GetType” help you to get the type. The difference is from where the information is extracted. “TypeOf” gets the type from a class while “GetType” gets type from an object.

Will the following c# code compile?

```
double dbl = 109.22;
int i = dbl;
```

No, the above code will give an error. Double size is larger than “int” so an implicit conversion will not take place. For that we need to do explicit conversion. So we need to something as shown in the below code. In the below code we have done an explicit conversion.

```
double dbl = 109.22;
int i =(int) dbl; // this is explicit conversion / casting
```

Also note after explicit conversion we will have data loss. In variable “i” we will get “109” value , the decimal values will be eliminated.

Explain the use of Icomparable in c#?

“IComparable” interface helps to implement custom sorting for collections in c#. Let us try to understand the same with a simple c# example. For instance let’s say you have a list collection of people names as shown in the below code.

```
List<string> Peoples = new List<string>();
Peoples.Add("Shiv");
Peoples.Add("Raju");
Peoples.Add("Sukesh");
Peoples.Add("Ajay");

foreach (string str in Peoples)
{
     Console.WriteLine(str);
}
```

Now if you run the above program you will see that we have not applied sort on the collection. It displays data as they were inserted.

```
Shiv
Raju
Sukesh
Ajay
```

But now if you call “Peoples.Sort()” method on the collection you will get the following sorted output on the “name” value in ascending position.

```
Ajay
Raju
Shiv
Sukesh
```

But now let’s say that we need the person’s age also to be added in the list with the person name value. So logically you will create a “Person” class with “name” and “age” property as shown in the code below.

```
class Person
    {
        private string _Name;

        public string Name
        {
            get { return _Name; }
            set { _Name = value; }
        }

        private int _Age;

        public int Age
        {
            get { return _Age; }
            set { _Age = value; }
        }


    }
```

Once you create the class you would create an object of the person class and add it to the list collection as shown in the below code snippet.

```
class Program
```

```
    {
        static void Main(string[] args)
        {
            List<Person> Peoples = new List<Person>();

            Peoples.Add(Createperson("Shiv",20));
            Peoples.Add(Createperson("Raju", 20));
            Peoples.Add(Createperson("Sukesh", 30));
            Peoples.Add(Createperson("Ajay", 40));
            Peoples.Sort();
            foreach (Person obj in Peoples)
            {
                Console.WriteLine(obj.Name + "  " + obj.Age);
            }
        }
        static Person Createperson(string Name, int Age)
        {
            Person obj = new Person();
            obj.Name = Name;
            obj.Age = Age;
            return obj;
        }
    }
```

But if you now try to call the “Sort” method on the List, he gets confused ?. On which value should he sort on “Name” or “Age” ?.

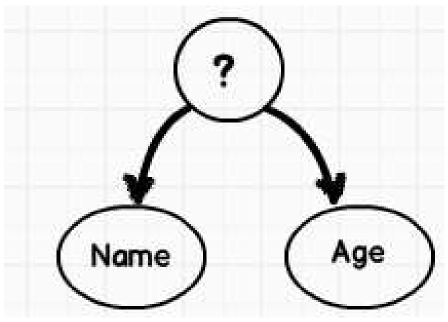


Figure 2.21:- IComparable

If you run the application it will throw up the below exception text. The exception text says that he is confused and he needs specific direction on how the sorting should work. This direction can be provided by using “IComparable” interface.

```
Unhandled Exception: System.InvalidOperationException: Failed to compare two
elements in the array. ---> System.ArgumentException: At least one object
must implement IComparable.
```

So in order to show a proper direction for the sort logic we need to implement “IComparable” interface as shown in the below. The logic of “Sort” needs to be defined in the “CompareTo” method.

```
 class Person : IComparable<Person>
   {
       private string _Name;

       public string Name
       {
           get { return _Name; }
           set { _Name = value; }
       }

       private int _Age;
       public int Age
       {
           get { return _Age; }
           set { _Age = value; }
       }
       public int CompareTo(Person other)
       {

           if (other.Age == this.Age)
           {
               return this.Name.CompareTo(other.Name);
           }
           else
```

```
            {
                return other.Age.CompareTo(this.Age);
            }}}
```

You can see in the "CompareTo" method we have defined the logic saying if the "age" value is same then sort by using the "name" value or else use "age" value for sorting. If you run the application you would get the following output. You can see for "40" and "30" value he has sorted on the basis "Age" but for the remaining people the age is same so he sorted on the "Name" value.

```
Loki  40
Sukesh  30
Ajay  20
Madan  20
Raju  20
Shiv  20
```

So the use of "IComparable" interface is to define custom sorting logic on a collection.

What is difference between Icomparable and IComparer ?

```
Pre-requisite:- Please read "Explain the use of Icomparable in c#?" , before
reading this answer.
```

"IComparable" interface helps you to implement a default sort implementation for the collection. But what if we want to sort using multiple criteria's?.For those instances "IComparable" has a limitation and "IComparer" is the guy.

For example if we want to sort the list by "Name" under some situation or sort by "Age" under some other situations we need to implement "IComparer" interface.

So the first step is to create different classes for each sort criteria. These classes will implement "IComparer" interface and will have sorting logic defined in the "Compare" method. You can see we have two different classes defined "CompareByName" and "CompareByAge". One compares on the basis of "name" and the other on the basis of "age".

```
class CompareByName : IComparer<Person>
    {
        public int Compare(Person x, Person y)
        {
            return string.Compare(x.Name, y.Name);
        }
    }
    class CompareByAge : IComparer<Person>
    {
        public int Compare(Person x, Person y)
        {
            if (x.Age > y.Age) return 1;
            if (x.Age < y.Age) return -1;
             return 0;
        }
    }
```

If you see the logic for “CompareByName” its simple. It uses the string comparison to evaluate the “Name” value sorting. But when we talk about numeric comparison there are 3 possible outputs Greater than , less than or Equal. So if you see “CompareByAge” class it returns 3 values 1 (Greater than) , -1 (Less than) and 0 (for Equal to).

Now that we have created the separate logics we can now invoke the “Peoples’ list by passing the class object. So for example if we want to sort by name, you will use the below code snippet.

```
Peoples.Sort(new CompareByName());
```

The output of the above code is nothing but alphabetically sorted data.

```
Ajay  20
Loki  40
Madan  20
Raju  20
Shiv  20
Sukesh  30
```

If you invoke the sort method of the list by using the age logic it will throw an output sorted on age value.

```
Peoples.Sort(new CompareByAge());
```

```
Ajay  20
Raju  20
Shiv  20
Madan  20
Sukesh  30
Loki  40
```

So the difference is really default internal implementation or customizable external implementation. When we use "IComparable" we can have only one default sorting logic and that logic goes inside the collection itself. For "IComparator" the logic is outside the collection , in other words more extensible and the collection is not disturbed.

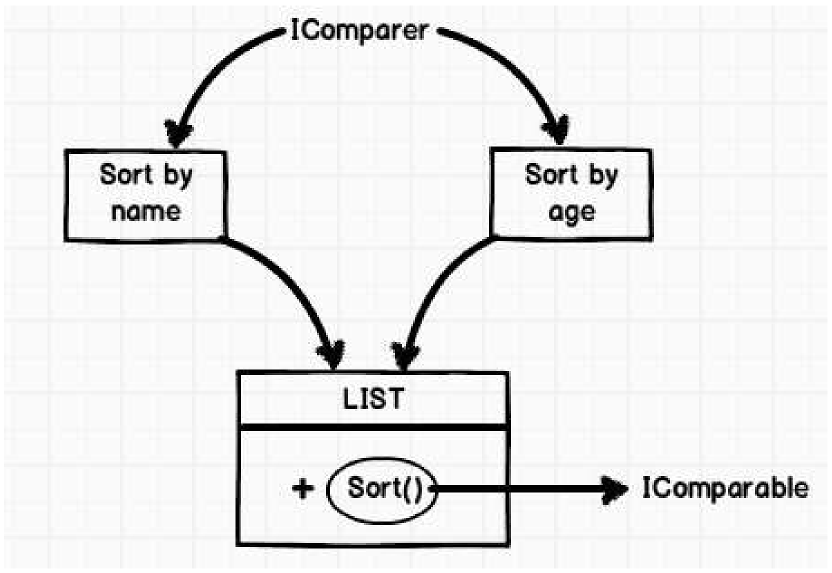


Figure 2.22:- IComparer vs IComparable

Can you explain Lazy Loading?

Lazy loading is a concept where we delay the loading of the object unit the point where we need it.Putting in simple words on demand object loading rather than loading the objects unnecessarily.

For example consider the below example where we have a simple "Customer" class and this "Customer" class has many "Order" objects inside it. Have a close look at the constructor of the "Customer" class. When the "Customer" object is created it also loads the "Order" object at that moment. So even if we need or do not need the address object , it's still loaded.

But how about just loading the “Customer” object initiallyand then on demand basis load the “Order” object.

```
public class Customer
{
private List<Order> _Orders= null;
…
…
public Customer()
{
            _CustomerName = "Shiv";
            _Orders = LoadOrders(); // Loads the address object even though
//not needed

}

private List<Order> LoadOrders()
{
            List<Order> temp = new List<Order>();
            Order o = new Order();
            o.OrderNumber = "ord1001";
            temp.Add(o);
            o = new Order();
            o.OrderNumber = "ord1002";
            temp.Add(o);
            return temp;
}

}
```

So let's consider you have client code which consumes the "Customer" class as shown below. So when the "Customer" object is created no "Order" objects should be loaded at that moment. But as soon as the "foreach" loop runs you would like to load the "Order" object at that point (on demand object loading).

```
Customer o = new Customer(); // Address object not loaded
Console.WriteLine(o.CustomerName);
foreach (Order o1 in o.Orders) // Load address object only at this moment
{
      Console.WriteLine(o1.OrderNumber);
}
```

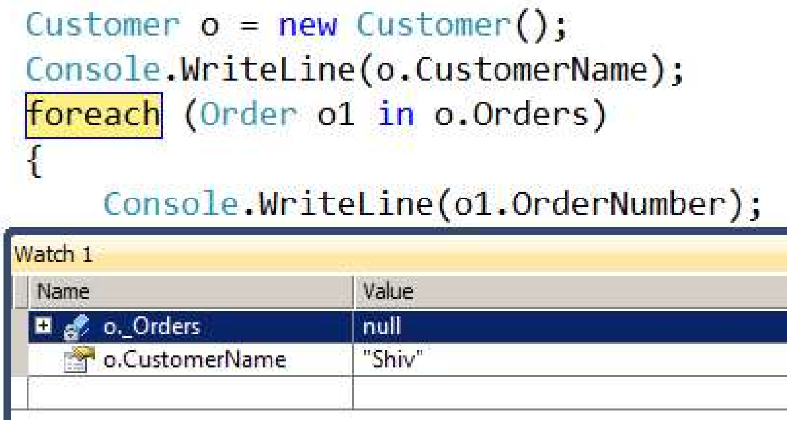
So how do we implement “LazyLoading” ?

So for the above example if we want to implement Lazy loading we will need to make the following changes:-

- Remove the “Order” object loading from the constructor.
- In the “Order” get property, load the “Order” object only if it’s not loaded.

```
public class Customer
{
private List<Order> _Orders= null;
…
…
public Customer()
{
            _CustomerName = "Shiv";
}

public List<Order> Orders
{
      get
        {
                if (_Orders == null)
                {
                    _Orders = LoadOrders();
                }
                return _Orders;
        }

}
```

Now if you run the client code and halt your debugger just before the "ForEach" loop runs over the "Orders"object, you can see the "Orders" object is null (i.e. not loaded). But as soon as the "ForEach" loop runs over the "Order" object it creates the "Order" object collection.



```
Customer o = new Customer();
Console.WriteLine(o.CustomerName);
foreach (Order o1 in o.Orders)
{
    Console.WriteLine(o1.OrderNumber);
```

```
Watch 1
Name              Value
o._Orders         null
o.CustomerName    "Shiv"
```

Figure 2.23: - For Each Loop

Are there any readymade objects in .NET by which we can implement Lazy loading?

In .NET we have "Lazy<T>" class which provides automatic support for lazy loading. So let's say if you want to implement "Lazy<>" in the above code we need to implement two steps for the same:-

Create the object of orders using the "Lazy" generic class.

```
private Lazy<List<Order>> _Orders= null;
```

Attach this Lazy<> object with the method which will help us load the order's data.

```
_Orders = new Lazy<List<Order>>(() => LoadOrders());
```

Now as soon as any client makes a call to the "_Orders" object ,it will call the "LoadOrders" function to load the data.

You will get the "List<Orders>" data in the "Value" property.

```
        public List<Order> Orders
        {
            get
            {
                return _Orders.Value;
            }


        }
```

Below goes the full code for the same.

```
public class Customer
{
private Lazy<List<Order>> _Orders= null;

        public List<Order> Orders
        {
            get
            {
                return _Orders.Value;
            }

        }
 public Customer()
        {
            // Makes a database trip
            _CustomerName = "Shiv";
            _Orders = new Lazy<List<Order>>(() => LoadOrders());

        }
}
```

What are the advantages / disadvantages of lazy loading?

Below are the advantages of lazy loading:-

- Minimizes start up time of the application.
- Application consumes less memory because of on-demand loading.
- Unnecessary database SQL execution is avoided.

The disadvantage is that the code becomes complicated. As we need to do checks if the loading is needed or not. So must be there is a slight decrease in performance.

But the advantages of are far more than the disadvantages.

```
FYI :- The opposite of Lazy loading is Eager loading. So in eager loading we load the all the objects in memory as soon as the object is created.
```

What is the difference between "IS" and "AS" keyword ?

"IS" keyword is useful to check if objects are compatible with a type. For instance in the below code we are checking if "ocust" object is a type of "Customer" class.

```
object ocust = new Customer();


if (ocust is Customer)
{
```

"AS" keyword helps to do conversion from one type to other type. For instance in the below code we are converting object to a string data type. If the "AS" keyword is not able to type cast it returns NULL.

```
object o = "interview";
string str = o as string;
```

What is the use of "Yield" keyword?

"Yield helps you to provide custom stateful iteration over .NET collections."

There are two scenarios where "yield" keyword is useful:-

- Customized iteration through a collection without creating a temporary collection.
- Stateful iteration.

Scenario 1:- Customized iteration through a collection

Let's try to understand what customized iteration means with an example. Consider the below code.
Let say we have a simple list called as "MyList" which has collection of 5 continuous numeric values 1,2,3,4 and 5. This list is browsed/iterated from console application from within static void main method.

For now let's visualize the "main()" method as a caller. So the caller i.e. "main()" method calls the list and displays the items inside it. Simple…till now ;-).

```
static List<int> MyList = new List<int>();


static void FillValues()

{

            MyList.Add(1);

            MyList.Add(2);

            MyList.Add(3);

            MyList.Add(4);

            MyList.Add(5);

}


static void Main(string[] args) // Caller

{

            FillValues(); // Fills the list with 5 values

            foreach (int i in MyList) // Browses through the list

            {

                Console.WriteLine(i);

            }

            Console.ReadLine();

}
```

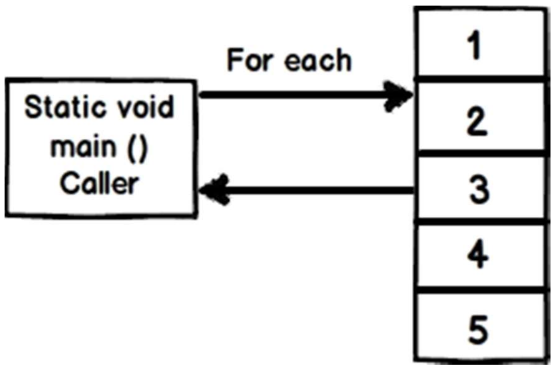


Figure 2.24: - My List

Now let me complicate this situation lets say the caller only wants values greater than "3" from the collection. So the obvious thing as a c# developer we will do is create a function as shown below. This function will have temporary collection. In this temporary collection we will first add values which are greater than "3" and return the same to the caller. The caller can then iterate through this collection.

```
static IEnumerable<int> FilterWithoutYield()
{
            List<int> temp = new List<int>();
            foreach (int i in MyList)
            {
                if (i > 3)
                {
                    temp.Add(i);
                }
            }
            return temp;
}
```

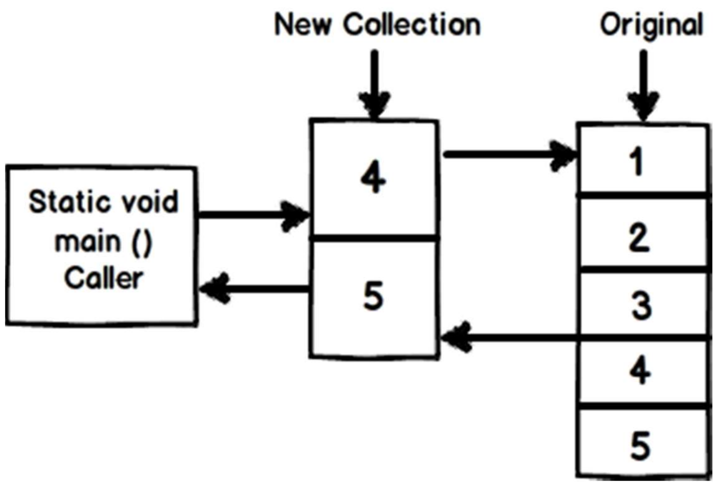


Figure 2.25: - New Collection

Now the above approach is fine but it would be great if we would get rid of the collection, so that our code becomes simple. This where "yield" keyword comes to help. Below is a simple code how we have used yield.

"Yield" keyword will return back the control to the caller, the caller will do his work and re-enter the function from where he had left and continue iteration from that point onwards. In other words "yield" keyword moves control of the program to and fro between caller and the collection.

```
static IEnumerable<int> FilterWithYield()
{
            foreach (int i in MyList)
            {
                if (i > 3) yield return i;
            }
}
```

So for the above code following are details steps how the control will flow between caller and collection. You can also see the pictorial representation in the next diagram shown below.

- Step 1:- Caller calls the function to iterate for number's greater than 3.
- Step 2:- Inside the function the for loop runs from 1 to 2 , from 2 to 3 until it encounters value greater than "3" i.e. "4". As soon as the condition of value greater than 3 is met the "yield" keyword sends this data back to the caller.
- Step 3:- Caller displays the value on the console and re-enters the function for more data. This time when it reenters, it does not start from first. It remembers the state and starts from "5". The iteration continues further as usual.

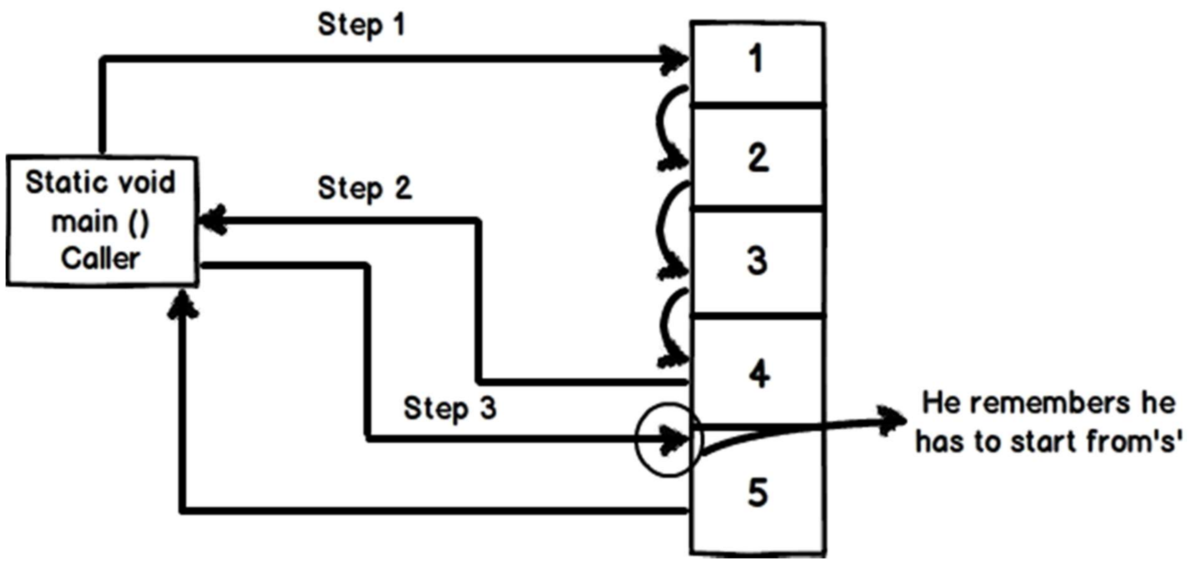

Step 1
Static void main () Caller
Step 2
Step 3
1
2
3
4
5
He remembers he has to start from's'

Figure 2.26: - Static Void Main Caller

Scenario 2:- Stateful iteration

Now let us add more complications to the above scenario. Let's say we want to display running total of the above collection. What do I mean?.

In other words we will browse from 1 to 5 and as we browse we would keep adding the total in variable. So we start with “1” the running total is “1”, we move to value “2” the running total is previous value “1” plus current value “2” i.e. “3” and so on.
Below is the pictorial representation of the running total looks like.

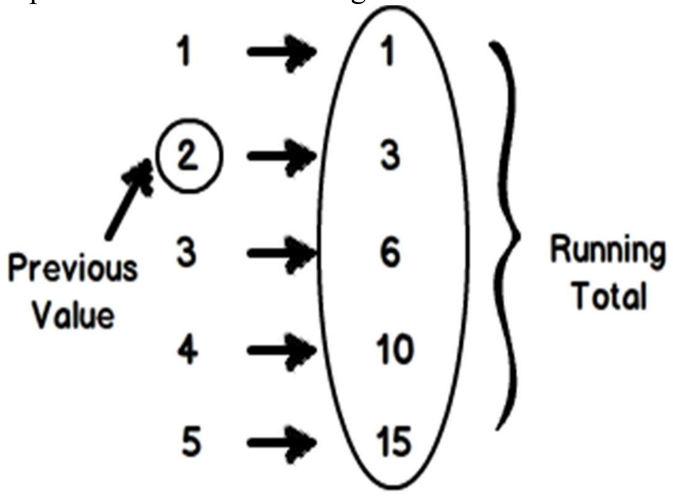


Figure 2.27: - Stateful Iteration

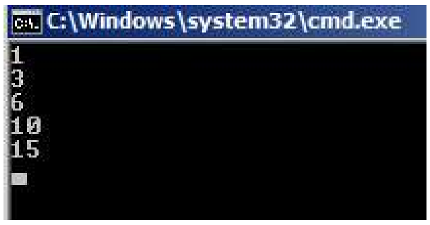
In other words we would like to iterate through the collection and as we iterate would like to maintain running total state and return the value to the caller (i.e. console application). So the function now becomes something as shown below. The “runningtotal” variable will have the old value every time the caller re-enters the function.

```
static IEnumerable<int> RunningTotal()
{
            int runningtotal=0;
             int index=0;
             foreach(int i in MyList)
             {
                 index = index + 1;
                 if(index==1)
                 {
                 runningtotal = i;
                 }
                 else
                 {
                  runningtotal = i + runningtotal;
                 }
```

```
                  yield return (runningtotal);


              }
}
```

Below goes the caller code and output.

```
foreach (int i in RunningTotal())
            {
                Console.WriteLine(i);
            }
            Console.ReadLine();
```



```
C:\Windows\system32\cmd.exe
1
3
6
10
15
```

Figure 2.28: - Output

What is the difference between "==" and .Equals()?

When we create any object there are two parts to the object one is the content and the other is reference to that content.
So for example if you create an object as shown in below code:-

1. ".NET interview questions" is the content.
2. "o" is the reference to that content.

```
object o = ".NET Interview questions";
```

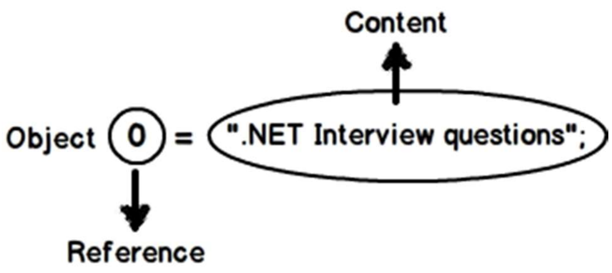


Figure 2.29: - Object

"==" compares if the object references are same while ".Equals()" compares if the contents are same.
So if you run the below code both "==" and ".Equals()" returns true because content as well as references are same.

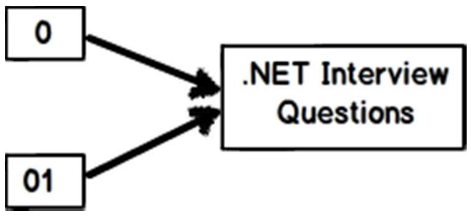


Figure 2.30: - .Equals()

```
object o = ".NET Interview questions";
object o1 = o;

Console.WriteLine(o == o1);
Console.WriteLine(o.Equals(o1));
Console.ReadLine();
```

```
True
True
```

Now consider the below code where we have same content but they point towards different instances. So if you run the below code both "==" will return false and ".Equals()" will return true.

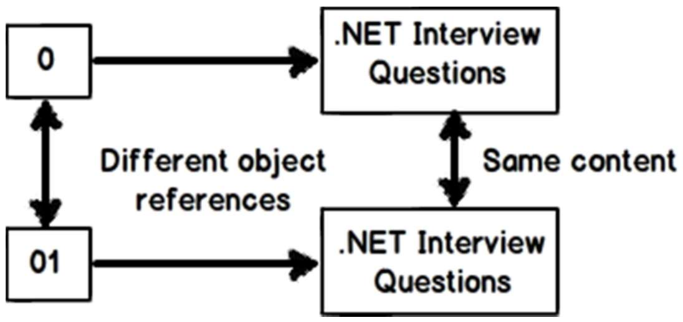


Figure 2.31: - Object References

```
object o = ".NET Interview questions";
object o1 = new string(".NET Interview questions".ToCharArray());



Console.WriteLine(o == o1);
Console.WriteLine(o.Equals(o1));
Console.ReadLine();
```

```
False
True
```

When you are using string data type it always does content comparison. In other words you either use ".Equals()" or "==" it always do content comparison.

What's the difference between catch with parameter and catch without parameter?

First let's try to understand this question: -

```
catch(Exception e)
{
…
}
```

VS

```
Catch
```

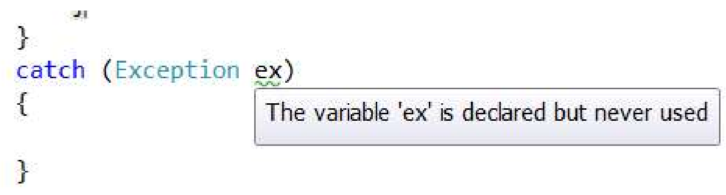
To get LIVE training, new topic release video updates install Telegram app & join us using - https://tinyurl.com/QuestPondChannel

```
{
…
}
```

For this interview question many people answer, “Second one will cause compile error”. But both the codes will work properly. Actually from .NET 2.0 there is no difference. But in the initial versions of .NET i.e. prior 2.0 some of the exceptions thrown by some COM components did not translate to “Exception” compatible object.

From .NET 2.0 both code will execute properly and there is no difference between them internally. Both catches will handle all kind of exceptions.

After 2.0 a catch which does have any code written in it gives a warning as shown below. So many developers mark this as a difference.



```
}
catch (Exception ex)
{
                     The variable 'ex' is declared but never used
}
```

Figure 2.32: - Exception

But you can always overcome this issue by not putting a variable as shown in the below code.

```
catch (Exception)
{

}
```

What are attributes and why do we need it?

“Attribute is nothing but a piece of information”.

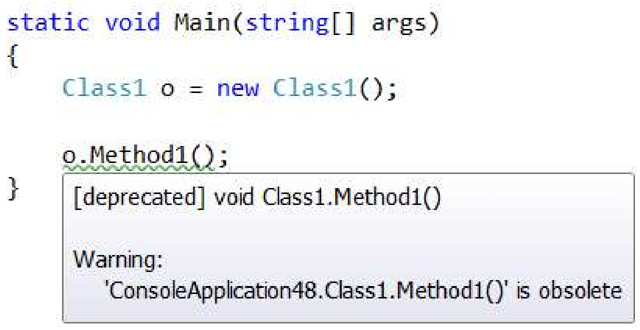
This information can be attached to your method, class, namespace, assembly etc. Attributes are part of your code this makes developers life easier as he can see the information right upfront in the code while he is calling the method or accessing the class and take actions accordingly.

For instance below is a simple class where “Method1” is decorated by the “Obsolete” attribute. Attributes are defined by using the “[]“ symbol. So when developers starting coding in this class they are alerted that “Method1” is obsolete and code should be now written in “NewMethod1”.

```
public class Class1
```

```
{
        [Obsolete]
        public void Method1()
        {
        }
        public void NewMethod1()
        {
        }
}
```

In the same way if somebody is trying to create object of "Class1" he gets an alert in the tool tip as shown in the below code snippet that "Method1" is obsolete and he should use "NewMethod1".



```
static void Main(string[] args)
{
    Class1 o = new Class1();

    o.Method1();
}
[deprecated] void Class1.Method1()

Warning:
   'ConsoleApplication48.Class1.Method1()' is obsolete
```

Figure 2.33: -New Method1

So in short Attributes are nothing small piece of information which is embedded declaratively in the code itself which developers can see upfront.

In case you want to show some message to the developers you can pass the message in the "Obsolete" attribute as shown in the below code snippet.

```
[Obsolete("Please use NewMethod1")]
public void Method1()
{


}
```

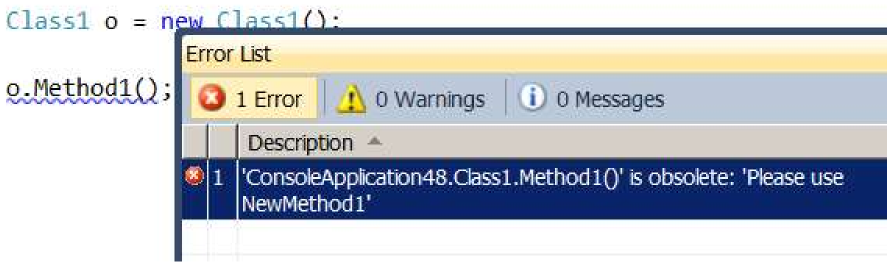
If you want to be bit strict and do not developers to use that method, you can pass 'true" to the "Obsolete" attribute as shown in the below code.

```
[Obsolete("Please use NewMethod1",true)]
public void Method1()
{

}
```

If you want to be bit strict and do not developers to use that method, you can pass 'true" to the "Obsolete" attribute as shown in the below code.

```
[Obsolete("Please use NewMethod1",true)]
public void Method1()
{

}
```

Now in case developers try to make a call to "Method1" they will get error and not just a simple warning.



```
Class1 o = new Class1();

o.Method1();
```

Error List

1 Error | 0 Warnings | 0 Messages

	Description
1	'ConsoleApplication48.Class1.Method1()' is obsolete: 'Please use NewMethod1'

Figure 2.34: - Error List

How can we create custom Attributes?

The "Obsolete" attribute which we discussed at the top is a readymade attribute To create a custom attributes you need to inherit from the attribute class. Below is a simple "HelpAttribute" which has a "HelpText" property.

```
class HelpAttribute : Attribute
{
        public string HelpText { get; set; }

}
```

"HelpAttribute" is applied to the "Customer" as shown in the code below. Now developers who see this class , see the information right in the front of their eyes.

```
[Help(HelpText="This is a class")]
    class Customer
    {
        private string _CustomerCode;

        [Help(HelpText = "This is a property")]
        public string CustomerCode
        {
            get { return _CustomerCode; }
            set { _CustomerCode = value; }
        }

        [Help(HelpText = "This is a method")]
        public void Add()
        {
        }
    }
```

How can we mark a method as deprecated?

Refer the previous answer.

What is the difference between Build Vs Rebuild Vs Clean solution menu ?

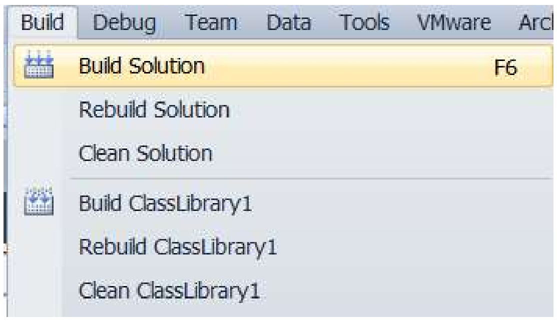


Figure 2.35: - Build VS Rebuild

To get LIVE training, new topic release video updates install Telegram app & join us using - https://tinyurl.com/QuestPondChannel

Build solution menu: - This will perform an incremental build. In other words it will only build code files which have changed. If they have not changed those files will not touched.
Rebuild solution menu: - This will delete all current compiled files (i.e. exe and dll's) and will build everything from scratch, irrespective if there is code change in the file or not.

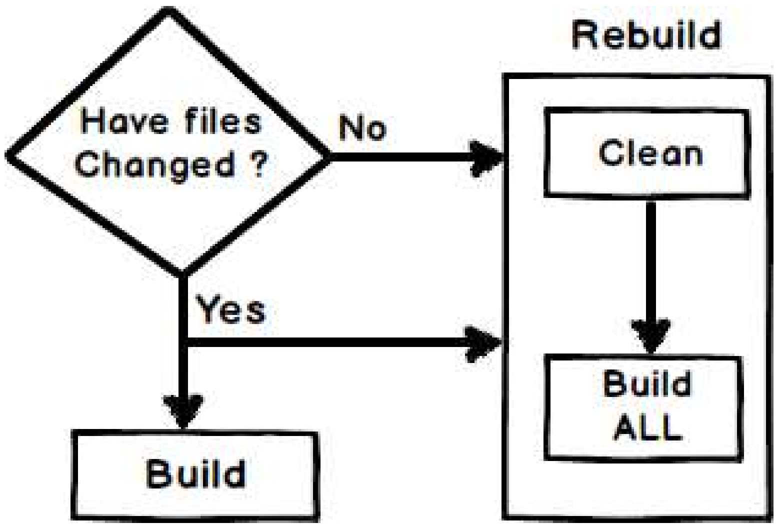


Figure 2.36: - Rebuild Solution

Clean solution menu: - This menu will delete all compiled files (i.e. EXE's and DLL's) from "bin" / "obj" directory.
Now if you read the above 3 points I have discussed you can conclude that:-

Rebuild = Clean + Build

So the next question would be If you do a "Rebuild" and if you do "Clean" + "Build" , what is the difference ?.

The difference is the way the build and clean sequence happens for every project. Let's say if your solution has two projects "proj1" and "proj2". If you do a rebuild it will take "proj1" , clean (delete) the compiled files for "proj1" and build it. After that it will take the second project "proj2" , clean compiled files for "proj2" and compile "proj2".

But if you do a "clean" and build". It will first delete all compiled files for "proj1" and "proj2" and then it will build "proj1" first followed by "proj2".
Below image explains the same in a more visual format.

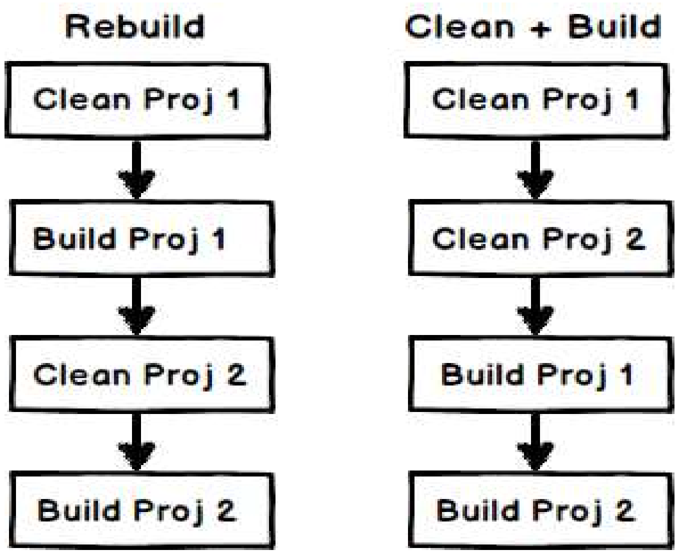
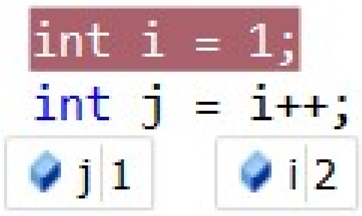


Figure 2.37: - Clean+Rebuild

What is the difference between i++ vs ++i ?

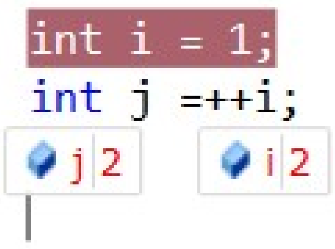
i++ :- In this scenario first the value is assigned and then increment happens. In the below code snippet first the value of "i" is assigned to "j" and then "i" is incremented.



```
int i = 1;
int j = i++;
```

Figure 2.38: - i++ VS ++i

++i :- In this scenario first the increment is done and then value is assigned.In the below code snippet value of "j" is 2 and value of "i" is also 2.



```
int i = 1;
int j =++i;
```

Figure 2.39: - ++i

If you visualize in a pictorial manner below is how it looks like. So i++ is a post fix , it first assigns and then increments. While ++i is a prefix, it first increments and then assigns the value.

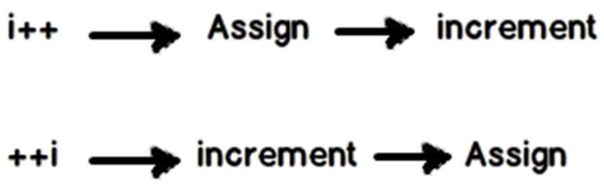


Figure 2.40: - Assign and Increment

Below are some sample code where postfix and prefix fits in.

```
while (i < 5)  //evaluates conditional statement
{
    //some logic
    ++i;       //increments i
}
```

```
while (i++ < 5) //evaluates conditional statement with i value before
increment
{
    //some logic
}
```

```
int i = 0;
int[] MyArray = new int[2];
```

```
MyArray[i++] = 1234; //sets array at index 0 to '1234' and i is incremented
MyArray[i] = 5678;   //sets array at index 1 to '5678'
int temp = MyArray[--i]; //temp is 1234
```

When should we use “??”(NULL Coalescing operator)?

“??” is a NULL coalescing operator. If you see the English meaning of coalescing it says “consolidate together”.Coalescing operator returns the first NON-NULL value from a chain. For example below is a simple coalescing code which chains four strings.

So if “str1” is null it will try “str2” , if “str2” is null it will try “str3” and so on until it finds a string with a non-null value.

```
string final =str1 ??  str2 ?? str3 ?? str4;
```

Explain need of NULLABLE types ?

It is difficult to assign NULLdirectly for value types like int , bool , double etc. By using NULLABLE types you can set value types as NULL. To create a NULLABLE type we need to put “?” before the data type as shown in the below code.

```
int? num1 = null;
```

In what scenario’s we will use NULLABLE types ?

The biggest user of NULLABLE types is when you are reading values from database. Database is one place where there is high possibility of column having NULL’s. So when we want to read those values in to value types NULLABLE types makes it easy as shown in the below code.

```
while (oreader.Read())
{
      int? salary = oreader["Salary"] as int? ;
}
```

What is the benefit of coalescing?

You do not need to write long if condition as shown below.

```
if (str1 != null)
{
                final = str1;
            }
            else
            {
                if (str2 != null)
                {
                    final = str2;
                }
                else
                {
                    if (str3 != null)
                    {
                        final = str3;
                    }
                    else
                    {
                        if (str4 != null)
                        {
                            final = str4;
                        }
                    }
                }
            }
```