<http://www.indiabix.com/c-sharp-programming/questions-and-answers/>

<http://www.hanselman.com/blog/NewInterviewQuestionsForSeniorSoftwareEngineers.aspx>

<http://csharpinterviewfaq.blogspot.in/>

### [WPF](http://csharpinterviewfaq.blogspot.in/2010/01/wpf.html)

**What is WPF?**  
**What is Hardware acceleration ? what is the use of it?**  
**What is Xaml?**  
**What is Declative programming?**  
**What is Dependency Property?**  
**When should I use WPF instead of Windows Forms in C#?**  
  1) Use it when you require various media type applications(to incorporate a video,documents or 3d content or animated transition between a sequence of images)  
2)If you need a skinned user interface.  
3)if you need to bind to xml data  
4)to dynamically load portions of use interface from a webservice.  
5) to create web like navigation style desktop applications.  
  
**What are the advantages of WPF over windows applications?**  
1)Databinding in WPF is superior to what Windows Forms offers.  
2)UI and C# business logic can be cleanly separated in WPF .  
3)Data/control templates – a much cleaner way than anything Windows Forms can offer.   
  
4)Styles – cool and simple. Its so easy to style all your buttons in an application to have the same look and feel.   
  
5)UI virtualization   
  
6)3D support.   
7)I can add a User Experience engineer to my team, and with no C# knowledge he can work magic in Expression Blend and give the front-office trading application a makeover that is guaranteed to win over the business users.   
**Why WPF superior to Windowsforms ?**  
  
1) Declarative Programming .  
2)Triggers  
3)Consistent styles  
4) Data Driven UI  
5)Flexible Control Model  
6)Adaptible UI layout  
7)Rich Application text  
8)Drawing object model  
9)Advanced graphics  
10)lookless controls

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=6723613169923447033)

### [C# interview Questions: Framework Questions](http://csharpinterviewfaq.blogspot.com/2010/01/framework-questions.html)

[C# interview Questions: Framework Questions](http://csharpinterviewfaq.blogspot.com/2010/01/framework-questions.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=3291142742754728740)

### [C# interview Questions: GarbageCollector](http://csharpinterviewfaq.blogspot.com/2010/01/what-is-garbage-collectiongc.html)

[C# interview Questions: GarbageCollector](http://csharpinterviewfaq.blogspot.com/2010/01/what-is-garbage-collectiongc.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=280717522285753382)

### [C# interview Questions: Assemblies](http://csharpinterviewfaq.blogspot.com/2010/01/assemblies.html)

[C# interview Questions: Assemblies](http://csharpinterviewfaq.blogspot.com/2010/01/assemblies.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=8810327928301654320)

### [C# interview Questions: Struct](http://csharpinterviewfaq.blogspot.com/2010/01/struct.html)

[C# interview Questions: Struct](http://csharpinterviewfaq.blogspot.com/2010/01/struct.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=6812006188355110139)

### [C# interview Questions: Reflection](http://csharpinterviewfaq.blogspot.com/2010/01/reflection.html)

[C# interview Questions: Reflection](http://csharpinterviewfaq.blogspot.com/2010/01/reflection.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=4120297676506040886)

### [C# : Generics](http://csharpinterviewfaq.blogspot.com/2010/01/generics.html)

[C# interview Questions: Generics](http://csharpinterviewfaq.blogspot.com/2010/01/generics.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=8497312825090367929)

### [C# interview Questions: Delegates](http://csharpinterviewfaq.blogspot.com/2010/01/delegates.html)

[C# interview Questions: Delegates](http://csharpinterviewfaq.blogspot.com/2010/01/delegates.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=7916497216252222547)

### [C# interview Questions: Constructors](http://csharpinterviewfaq.blogspot.com/2010/01/constructors.html)

[C# interview Questions: Constructors](http://csharpinterviewfaq.blogspot.com/2010/01/constructors.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=8930839033881017341)

### [C# : LINQ Questions](http://csharpinterviewfaq.blogspot.com/2010/01/linq.html" \l "links)

[C# interview Questions: LINQ](http://csharpinterviewfaq.blogspot.com/2010/01/linq.html#links)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=4141031389312613157)

### [C# interview Questions: Exception Handling](http://csharpinterviewfaq.blogspot.com/2010/01/exception-handling.html" \l "links)

[C# interview Questions: Exception Handling](http://csharpinterviewfaq.blogspot.com/2010/01/exception-handling.html#links)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=5617275071792792637)

### [C# : Arrays Questions](http://csharpinterviewfaq.blogspot.com/2010/01/arrays.html)

[C# interview Questions: Arrays](http://csharpinterviewfaq.blogspot.com/2010/01/arrays.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=1636041381163275449)

### [Arrays](http://csharpinterviewfaq.blogspot.in/2010/01/arrays.html)

**What is the difference between arrays in C# and arrays in other programming languages?**  
Arrays in C# work similarly to how arrays work in most other popular languages There are, however, a few differences as listed below1. When declaring an array in C#, the square brackets ([]) must come after the type, not the identifier. Placing the brackets after the identifier is not legal syntax in C#.  
int[] IntegerArray; // not int IntegerArray[];2. Another difference is that the size of the array is not part of its type as it is in the C language. This allows you to declare an array and assign any array of int objects to it, regardless of the array's length.int[] IntegerArray; // declare IntegerArray as an int array of any sizeIntegerArray = new int[10]; // IntegerArray is a 10 element arrayIntegerArray = new int[50]; // now IntegerArray is a 50 element arrayWhat are the 3 different types of arrays that we have in C#?1. Single Dimensional Arrays2. Multi Dimensional Arrays also called as rectangular arrays3. Array Of Arrays also called as jagged arraysAre arrays in C# value types or reference types?Reference types.  
**What is the base class for all arrays in C#?**  
System.Array  
**How do you sort an array in C#?**  
The Sort static method of the Array class can be used to sort array items.  
**Give an example to print the numbers in the array in descending order?**  
using System;  
namespace ConsoleApplication  
{class Program{static void Main()  
{int[] Numbers = { 2, 5, 3, 1, 4 };//Print the numbers in the array without sortingConsole.WriteLine("Printing the numbers in the array without sorting");  
foreach (int i in Numbers){Console.WriteLine(i);}//Sort and then print the numbers in the arrayConsole.WriteLine("Printing the numbers in the array after sorting");Array.Sort(Numbers);  
foreach (int i in Numbers){Console.WriteLine(i);}//Print the numbers in the array in desceding orderConsole.WriteLine("Printing the numbers in the array in desceding order");Array.Reverse(Numbers);  
foreach (int i in Numbers){Console.WriteLine(i);}}}}  
**What property of an array object can be used to get the total number of elements in an array?**  
Length property of array object gives you the total number of elements in an array. An example is shown below.  
using System;  
namespace ConsoleApplication  
{  
class Program  
{  
static void Main()  
{int[] Numbers = { 2, 5, 3, 1, 4 };  
Console.WriteLine("Total number of elements = " +Numbers.Length);  
}}}  
**Give an example to show how to copy one array into another array?**  
We can use CopyTo() method to copy one array into another array. An example is shown below.  
using System  
;namespace ConsoleApplication  
{  
class Program  
{  
static void Main(){int[] Numbers = { 2, 5, 3, 1, 4 };  
int[] CopyOfNumbers=new int[5];  
Numbers.CopyTo(CopyOfNumbers,0);  
foreach (int i in CopyOfNumbers){Console.WriteLine(i);  
}  
}  
}  
}  
3) **Which of these statements correctly declares a two-dimensional array in C#?**  
**int[,] myArray;**  
int[][] myArray;  
int[2] myArray;  
System.Array[2] myArray;  
**Can you store multiple data types in System.Array?**No.  
**What’s the difference between the System.Array.CopyTo() and System.Array.Clone()?**  
The Clone() method returns a new array (a shallow copy) object containing all the elements in the original array. The CopyTo() method copies the elements into another existing array. Both perform a shallow copy. A shallow copy means the contents (each array element) contains references to the same object as the elements in the original array. A deep copy (which neither of these methods performs) would create a new instance of each element's object, resulting in a different, yet identacle object.  
**How can you sort the elements of the array in descending order?**  
By calling Sort() and then Reverse() methods.

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=6762726588511742514)

### [C# : Strings](http://csharpinterviewfaq.blogspot.com/2010/01/strings.html)

[C# interview Questions: Strings](http://csharpinterviewfaq.blogspot.com/2010/01/strings.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=3942986994587338282)

### [C# : DataGrid](http://csharpinterviewfaq.blogspot.com/2010/01/datagrid.html" \l "links)

[C# interview Questions: DataGrid](http://csharpinterviewfaq.blogspot.com/2010/01/datagrid.html#links)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=4881677235390189290)

### [C# : Threading](http://csharpinterviewfaq.blogspot.com/2010/01/threading.html)

[C# interview Questions: Threading](http://csharpinterviewfaq.blogspot.com/2010/01/threading.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=7750065110290059263)

### [C# : OOPs](http://csharpinterviewfaq.blogspot.com/2010/01/objectoriented-concepts.html)

[C# interview Questions: C# : OOPs](http://csharpinterviewfaq.blogspot.com/2010/01/objectoriented-concepts.html)

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=2363821889351659957)

### [C# : OOPs](http://csharpinterviewfaq.blogspot.in/2010/01/objectoriented-concepts.html)

**What is Object?**  
Objects are programming constructs that have data, behavior, and identity. Object data is contained in the fields, properties, and events of the object, and object behaviors are defined by the methods and interfaces of the object.  
Objects have identity — two objects with the same set of data are not necessarily the same object.  
Objects in C# are defined through classes and structs — these form the single blueprint from which all objects of that type operate.  
Objects have the following properties:  
· Everything you use in C# is an object, including Windows Forms and controls.  
· Objects are instantiated; that is, they are created from templates defined by classes and structs.  
· Objects use [Properties](ms-help://MS.MSDNQTR.v90.en/dv_csref/html/e295a8a2-b357-4ee7-a12e-385a44146fa8.htm)to obtain and change the information they contain.  
· Objects often have methods and events that allow them to perform actions.  
· All C# objects inherit from the Object.  
**What is Class?**  
Classes are symbolic representations of objects; classes describe the properties, fields, methods, and events that form objects. Classes control user access to object items through encapsulation.  
a class defines the data and behavior of the data type. Programmers can then create objects that are instances of this class.  
Classes have the following properties:  
· a class can inherit implementation from only one base class.  
· A class can implement more than one interface.  
· Class definitions can be split between different source files.  
· Static classes are sealed classes that contain only static methods.  
**What is the difference between class and object?**Classes describe the structure of objects, while objects are usable instances of classes. Each instance is an exact yet distinct copy of its class. Because an object is an instance of a class, the act of creating an object is called instantiation.  
Usually, changing the data in one object does not change the data in any other object.  
**How will you create objects?**  
A class defines a type of object, but it is not an object itself. An object is a concrete entity based on a class, and is sometimes referred to as an instance of a class.  
Objects can be created by using the new keyword followed by the name of the class that the object will be based on, like this:  
Customer object1 = new Customer();  
When an instance of a class is created, a reference to the object is passed back to the programmer.

**What is encapsulation?**  
Encapsulation is the ability to contain and control access to a group of associated items. Classes provide one of the most common ways to encapsulate items. Encapsulation allows you to control how the data and procedures are used. You can use access modifiers, such as Private or Protected, to prevent outside procedures from executing class methods or reading and modifying data in properties and fields. You should declare internal details of a class as Private to prevent them from being used outside of your class; this technique is called data hiding.One basic rule of encapsulation is that class data should be modified or retrieved only via Property procedures or methods. Hiding the implementation details of your classes prevents classes from being used in undesired ways, and lets you to later modify such items without risk of compatibility problems.  
  
**What is Inheritance?**  
You can extend the functionality of an existing class by creating a new class that derives from the existing class. The derived class inherits the properties of the base class, and you can add or override methods and properties as required.  
In C#, both inheritance and interface implementation are defined by the : operator.  
The base class should always be leftmost in the class declaration.  
C# does not support multiple inheritance meaning that classes cannot inherit from more than one class. You can,

however, use interfaces for that purpose  
  
e.g:  
public class CoOrds  
{  
Private int x,y;  
Public coOrds()  
{  
x=0;  
y =0;  
}  
  
Public int X  
{  
get {return x;}  
set { x =value;}  
}  
  
Public int Y  
{ get { return y;}  
Set { y=value ;}  
}  
}  
  
You derive a new class, called ColorCoOrds, from the CoOrds class, as follows:  
  
ColorCoOrds : CoOrds  
{  
  
}  
ColorCoOrds then inherits all the fields and methods of the base class, to which you can add new ones to provide extra features in the derived class according to our needs.  
  
The constructor of the derived class implicitly calls the constructor for the base class. In inheritance, all base class constructors are called before the derived class's constructors in the order that the classes appear in the class hierarchy.  
  
**How will you typecast to a baseclass?**you cannot use a reference to a base class to access the members and methods of a derived class even if the base class reference may contain a valid reference to an object of the derived type.  
You can reference a derived class with a reference to the derived type implicitly:  
ColorCoOrds color1 = new ColorCoOrds();  
CoOrds coords1 = color1;  
In this code, the base class reference, coords1, contains a copy of the color1 reference.  
  
**What is the use of base keyword?**  
You can access base class members in a subclass even when those base members are overridden in the superclass using the base keyword.  
For instance, you can create a derived class which contains a method with the same signature as in the base class. If you prefaced that method with the new keyword, you indicate that this is an all-new method belonging to the derived class. You could still provide a method for accessing the original method in the base class with the base keyword.  
For instance, say your base CoOrds class had a method called Invert(), which swaps the x and y coordinates. You could provide a substitute for this method in your derived ColorCoOrds class with code like this:  
public new void Invert()  
{  
int temp = X;  
X = Y;  
Y = temp;  
screenColor = System.Drawing.Color.Gray;  
}  
As you can see, this method swaps x and y, and then sets the coordinates' color to gray. You could provide access to the base implementation for this method by creating another method in ColorCoOrds, such as this one:  
public void BaseInvert()  
{  
base.Invert();  
}  
You then invoke the base method on a ColorCoOrds object by calling the BaseInvert() method.  
ColorCoOrds color1 = new ColorCoOrds();  
color1.BaseInvert();  
Remember that you would get the same effect if you assigned a reference to the base class to an instance of ColorCoOrds, and then accessed its methods:  
CoOrds coords1 = color1;  
coords1.Invert();  
  
In case of Base class constructor and derived class constructor which one executed first?  
Base class objects are always constructed before any deriving class. Thus the constructor for the base class is executed before the constructor of the derived class. If the base class has more than one constructor, the derived class can decide the constructor to be called.  
  
e.g: public class CoOrds  
{  
private int x, y;  
  
public CoOrds()  
{  
x = 0;  
y = 0;  
}  
  
public CoOrds(int x, int y)  
{  
this.x = x;  
this.y = y;  
}  
}  
  
You could then change the ColorCoOrds class to use a particular one of the available constructors using the base keyword:  
public class ColorCoOrds : CoOrds  
{  
public System.Drawing.Color color;  
  
public ColorCoOrds() : base ()  
{  
color = System.Drawing.Color.Red;  
}  
  
public ColorCoOrds(int x, int y) : base (x, y)  
{  
color = System.Drawing.Color.Red;  
}  
}  
  
**What is Method Overriding?**  
  
A derived class may override the method of a base class by providing a new implementation for the declared method.  
methods must be explicitly marked as virtual using the virtual modifier. Property accessors, as well as methods, can be overridden in much the same way.  
  
**What is virtual method?**  
A method that is to be overridden in a derived class is declared with the virtual modifier. In a derived class, the overridden method is declared using the override modifier.  
The override modifier denotes a method or a property of a derived class that replaces one with the same name and signature in the base class. The base method, which is to be overridden, must be declared as virtual, abstract, or override: it is not possible to override a non-virtual or static method in this way. Both the overridden and the overriding method or property must have the same access-level modifiers.  
  
**What is polymorphism?**  
ability of a derived class to redefine, or override, methods that it inherits from a base class.  
**Explain About Access modifiers?**  
Access modifiers are keywords used to specify the declared accessibility of a member or a type. This section introduces the four access modifiers:  
· public  
· protected  
· internal  
· private  
The following five accessibility levels can be specified using the access modifiers:  
public: Access is not restricted.  
protected: Access is limited to the containing class or types derived from the containing class.  
Internal: Access is limited to the current assembly.  
protected internal: Access is limited to the current assembly or types derived from the containing class.  
private: Access is limited to the containing type.  
Difference between const and readonly?  
The readonly keyword differs from the const keyword. A const field can only be initialized at the declaration of the field. A readonly field can be initialized either at the declaration or in a constructor. Therefore, readonly fields can have different values depending on the constructor used. Also, although a const field is a compile-time constant, the readonly field can be used for run-time constants, as in this line: public static readonly uint l1 = (uint)DateTime.Now.Ticks;  
**What is Property?**  
Properties are members that provide a flexible mechanism to read, write, or compute the values of private fields. Properties can be used as if they are public data members, but they are actually special methods called accessors. This enables data to be accessed easily and still helps promote the safety and flexibility of methods.  
· Properties enable a class to expose a public way of getting and setting values, while hiding implementation or verification code.  
· A get property accessor is used to return the property value, and a set accessor is used to assign a new value. These accessors can have different access levels. For more information  
· The value keyword is used to define the value being assigned by the set indexer.  
· Properties that do not implement a set method are read only.  
· For simple properties that require no custom accessor code, consider the option of using auto-implemented properties.  
**What is Indexer?**  
Indexers are a syntactic convenience that enable you to create a class, struct, or interface that client applications can access just as an array. Indexers are most frequently implemented in types whose primary purpose is to encapsulate an internal collection or array.  
**What is Abstract class?**  
The abstract keyword enables you to create classes and class members solely for the purpose of inheritance—to define features of derived, non-abstract classes.  
An abstract class cannot be instantiated. The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.  
What is sealed class?  
A sealed class cannot be used as a base class. For this reason, it cannot also be an abstract class. Sealed classes are primarily used to prevent derivation. Because they can never be used as a base class, some run-time optimizations can make calling sealed class members slightly faster.  
A class member, method, field, property, or event, on a derived class that is overriding a virtual member of the base class can declare that member as sealed.  
  
This negates the virtual aspect of the member for any further derived class. This is accomplished by putting the sealed keyword before the override keyword in the class member declaration. For example:  
  
public class D : C  
{  
public sealed override void DoWork() { }  
}  
  
**What is interface?**  
Interfaces describe a group of related functionalities that can belong to any class or struct. Interfaces can consist of methods, properties, events, indexers, or any combination of those four member types. An interface cannot contain fields. Interfaces members are automatically public.  
· A class or struct can inherit more than one interface.  
· When a class or struct inherits an interface, it inherits only the method names and signatures, because the interface itself contains no implementations.  
· To implement an interface member, the corresponding member on the class must be public, non-static, and have the same name and signature as the interface member. Properties and indexers on a class can define extra accessors for a property or indexer defined on an interface.  
· Interfaces and interface members are abstract; interfaces do not provide a default implementation.

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=2687334059844510649)

### [Threading](http://csharpinterviewfaq.blogspot.in/2010/01/threading.html)

**What is Thread ?**  
Threads are the basic unit to which an operating system allocates processor time, and more than one thread can be executing code inside that process. Each thread maintains exception handlers, a scheduling priority, and a set of structures the system uses to save the thread context until it is scheduled. The thread context includes all the information the thread needs to seamlessly resume execution, including the thread's set of CPU registers and stack, in the address space of the thread's host process.  
**When to use multiple threads?**  
To increase responsiveness to the user and decrease the data processing time of your application. If you are doing intensive input/output work, you can also use I/O completion ports to increase your application's responsiveness.  
**In what kind of scenarios a single application domain could use multiple threads?**  
Without modification, the same application would dramatically increase user satisfaction when run on a computer with more than one processor. Your single application domain could use multiple threads to accomplish the following tasks:  
Communicate over a network, to a Web server, and to a database.  
· Perform operations that take a large amount of time.  
· Distinguish tasks of varying priority. For example, a high-priority thread manages time-critical tasks, and a low-priority thread performs other tasks.  
· Allow the user interface to remain responsive, while allocating time to background tasks.  
  
**What is the use of volatile keyword?**  
The volatile keyword indicates that a field might be modified by multiple threads that are executing at the same time. Fields that are declared volatile are not subject to compiler optimizations that assume access by a single thread. This ensures that the most up-to-date value is present in the field at all times.  
The volatile modifier is usually used for a field that is accessed by multiple threads without using the lock statement to serialize access.  
  
**What kind of types of fields volatile support?**  
The volatile keyword can be applied to fields of these types:  
· Reference types.  
· Pointer types (in an unsafe context). Note that although the pointer itself can be volatile, the object that it points to cannot. In other words, you cannot declare a "pointer to volatile."  
· Integral types such as sbyte, byte, short, ushort, int, uint, char, float, and bool.  
· An enum type with an integral base type.  
· Generic type parameters known to be reference types.  
· IntPtr and UIntPtr.  
  
**Can I declare a local variable as volatile?**Local variables cannot be declared volatile.

[http://www.blogger.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=7185510103680337834&postID=8721969579427968744)

[Older Posts](http://csharpinterviewfaq.blogspot.in/search?updated-max=2010-01-18T18:04:00%2B06:00&max-results=20)[Home](http://csharpinterviewfaq.blogspot.in/)

Subscribe to: [Posts (Atom)](http://csharpinterviewfaq.blogspot.com/feeds/posts/default)

## SEARCH THIS BLOG

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |

## FOLLOWERS

### [Framework Questions](http://csharpinterviewfaq.blogspot.in/2010/01/framework-questions.html)

[**What is the .NET Framework?**](http://www.dotnetinterviewfaqs.com/what-is-microsoft-dot-net-framework.aspx)  
The .NET Framework is an integral Windows component that supports building and running the applications and Web services. The .NET Framework provides a managed execution environment, simplified development and deployment, and integration with a wide variety of programming languages.The key components of the .NET Framework are the common language runtime (CLR) and the .NET Framework class library  
[**What is CLR (Common Runtime Language)?**](http://www.dotnetinterviewfaqs.com/what-is-clr-common-runtime-language-in-microsoft-dot-net.aspx)  
The common language runtime is the foundation of the .NET Framework. manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness.  
· The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the common language runtime.  
· The runtime enforces code access security.  
· The runtime also enforces code robustness by implementing a strict type-and-code-verification infrastructure called the common type system (CTS).  
**What is NET Framework class library ?**  
A library of classes, interfaces, and value types that are included in the .NET Framework SDK. This library provides access to system functionality and is designed to be the foundation on which .NET Framework applications, components, and controls are built. See also: class, CLS-compliant, Common Language Specification, interface, value type.  
**What is CTS?**  
The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:  
· Establishes a framework that helps enable cross-language integration, type safety, and high performance code execution.  
· Provides an object-oriented model that supports the complete implementation of many programming languages.  
· Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.  
**What IS CLS?**  
To fully interact with other objects regardless of the language they were implemented in, objects must expose to callers only those features that are common to all the languages they must interoperate with. For this reason, the Common Language Specification (CLS), which is a set of basic language features needed by many applications, has been defined. The CLS rules define a subset of the [Common Type System](ms-help://MS.MSDNQTR.v90.en/dv_fxintro/html/53c57c96-83e1-4ee3-9543-9ac832671a89.htm)  
[**What is MSIL (Microsoft Intermediate Language)?**](http://www.dotnetinterviewfaqs.com/what-is-msil-microsoft-intermediate-language-in-dot-net.aspx)  
When compiling to managed code, the compiler translates source code into Microsoft intermediate language (MSIL), which is a CPU-independent set of instructions that can be efficiently converted to native code.  
MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. Before code can be run, MSIL must be converted to CPU-specific code, usually by a just-in-time (JIT) compiler. Because the common language runtime supplies one or more JIT compilers for each computer architecture it supports, the same set of MSIL can be JIT-compiled and run on any supported architecture.  
**What is Just-in time Compilation?**  
Before you can run Microsoft intermediate language (MSIL), it must be compiled against the common language runtime to native code against the common language runtime for the target machine architecture. The .NET Framework provides two ways to perform this conversion:  
· A .NET Framework just-in-time (JIT) compiler.  
· The .NET Framework Native Image Generator (Ngen.exe).  
JIT compilation converts MSIL to native code on demand at application run time, when the contents of an assembly are loaded and executed. Because the common language runtime supplies a JIT compiler for each supported CPU architecture, developers can build a set of MSIL assemblies that can be JIT-compiled and run on different computers with different machine architectures. However, your managed code will run only on a specific operating system if it calls platform-specific native APIs, or a platform-specific class library.  
JIT compilation takes into account the fact that some code might never get called during execution. Rather than using time and memory to convert all the MSIL in a portable executable (PE) file to native code, it converts the MSIL as needed during execution and stores the resulting native code in memory so that it is accessible for subsequent calls in the context of that process.

**What is TypeSafty in .net?**

The ability for languages and classes to exchange information through commonly agreed upon definitions and usage patterns for types.

**What is Managed Code?**

Compilers and tools expose the runtime's functionality and enable you to write code that benefits from this managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code; it benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

To enable the runtime to provide services to managed code, language compilers must emit metadata that describes the types, members, and references in your code.

**Explain about Types of jit?**

prejit,ecno jit,normal jit

pre jit : convert [source code](http://www.newinterviewquestions.com/interview/8687/) to native code in [single](http://www.newinterviewquestions.com/interview/8687/) completion of cycle.Normally this can be done at the time of deployment.

econo jit : coverts the only called methods to native code,however it can be removed when are not required.

normal jit : compliled the called methods to native code.In the the methods can be compiles for the first time.For latter calls it can be displayed using cached items.

### [GarbageCollector](http://csharpinterviewfaq.blogspot.in/2010/01/what-is-garbage-collectiongc.html)

[**What is garbage collection(GC)?**](http://www.dotnetinterviewfaqs.com/what-is-gc-garbage-collection-in-microsoft-dot-net.aspx)  
The .NET Framework's garbage collector manages the allocation and release of memory for your application. Each time you use the newoperator to create an object, the runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.  
**What Object.Finalize method does?**  
allows an object to clean up its unmanaged resources properly when the garbage collector reclaims the memory used by the object.

**What are the limitations of Finalizers?**  
· The exact time of execution during garbage collection is undefined.

* · The finalizers of two objects are not guaranteed to run in any specific order, even if one object refers to the other.   
  · The thread on which the finalizer is run is unspecified.  
  Finalize cannot be called on resurrected objects during garbage collection.

**What is strong reference and what is weak reference?**

There are two types of memory references, strong & weak. When a root references an object, it is said to be a strong reference as the object is being pointed to by application code. The other type of object, that is not being referenced by the application code is called the weak reference, and this may be collected by the GC. However, this may still be accessed by the application code if required. But for the application to access the weakly referenced object, this has to be converted to a strong reference (and note that this has to be done before the GC collects the weakly referenced object).  
**What is Dispose Method?**  
A type's Dispose method should release all the resources that it owns. It should also release all resources owned by its base types by calling its parent type's Dispose method. The parent type's Dispose method should release all resources that it owns and in turn call its parent type's Dispose method, propagating this pattern through the hierarchy of base types. To help ensure that resources are always cleaned up appropriately, a Dispose method should be callable multiple times without throwing an exception.  
A Dispose method should call the GC.SuppressFinalize method for the object it is disposing. If the object is currently on the finalization queue, GC.SuppressFinalize prevents its Finalize method from being called. Remember that executing a Finalize method is costly to performance. If your Dispose method has already done the work to clean up the object, then it is not necessary for the garbage collector to call the object's Finalize method.  
**What will you do to prevent the garbage collector from calling [Object.Finalize](ms-help://MS.MSDNQTR.v90.en/fxref_mscorlib/html/99598e55-a3d9-940d-93de-e09c36440968.htm) on an object that does not require it?**call GC.SuppressFinalize() method .

**What are Destructors?**

Destructors are the C# mechanism for performing cleanup operations. Destructors are used to destruct instances of classes. A class can only have one destructor.Destructors cannot be inherited or overloaded. · Destructors cannot be called. They are invoked automatically.

} The destructor implicitly calls Finalize on the base class of the object. Finalize method is called recursively for all instances in the inheritance chain, from the most-derived to the least-derived. Empty destructors should not be used. When a class contains a destructor, an entry is created in the Finalize queue. When the destructor is called, the garbage collector is invoked to process the queue. If the destructor is empty, this just causes a needless loss of performance. The programmer has no control over when the destructor is called because this is determined by the garbage collector. The garbage collector checks for objects that are no longer being used by the application. If it considers an object eligible for destruction, it calls the destructor (if any) and reclaims the memory used to store the object. Destructors are also called when the program exits. It is possible to force garbage collection by calling Collect, but most of the time, this should be avoided because it may create performance issues

**Explain About latency modes?**

To reclaim objects, the garbage collector must stop all of an application's executing threads. In some situations, such as when an application retrieves data or displays content, a full garbage collection can occur at a critical time and impede performance. You can adjust the intrusiveness of the garbage collector by setting the LatencyMode property to one of the GCLatencyMode values.  
Latency refers to the time that the garbage collector intrudes in your application. During low latency periods the garbage collector is more conservative, and less intrusive, in reclaiming objects. Generation 2 collections occur less frequently, which causes the application working set to grow over time. As a result, it is recommended that you use the LowLatency mode only for the short period of time when it is needed. Otherwise, if the system is under memory pressure, the garbage collector will trigger a collection, which can briefly pause the application and disrupt a time-critical operation.  
You should use the latency mode with applications that include a block of code that runs over a brief period of time and must run with minimal interruptions from the runtime. Although the LowLatency mode is designed to be used in scenarios where there are some time constraints, it is not intended to be a solution for scenarios where there are strict real-time constraints.

**How many Options of Latencymode for garbage collector? what are they?**

There are three latency modes

1)Batch: Disables garbage collection concurrency and reclaims objects in a batch call. This is the most intrusive mode.This mode is designed for maximum throughput at the expense of responsiveness.

2)Interactive:The default latency mode. Enables garbage collection concurrency and reclaims objects while the application is running.

3)LowLatency:Enables garbage collection that is more conservative in reclaiming objects. Collections occur less frequently. This is the least intrusive mode.  
This mode is not available on the server garbage collector.

Other Questions:

**1)If i have 100 objects in my application are out of scope.when first time garbage collected how many objects memory reference are free?**

All the objects of genereation 0 are freed.

**2)destructor, Dispose, Finalize - How they relate to each other?**

These are used to release unmanaged resources

**3)How we free the memory in C#.NET.**

The .NET Framework's garbage collector manages the allocation and release of memory for your application.

**4)How u call destructor and dispose methode in c#.NET**

for destructor :

class Car  
{  
~Car() // destructor  
{  
// cleanup statements...  
}  
}

For Dispose:

DisposableResource TestObj = new DisposableResource(fs);

TestObj.Dispose();

**5)Where Destructors Can not be defined?**

in structs

**6)What are the different generaions of Garbage Collection and how do they work ?**

Generations in the Garbage Collector is a way of enhancing the garbage collection performance**.**There are 3 Generations...0,1,2.

Generation 0 - When an object is initialized, its in generation 0. These are new objects that have never been played around with by the GC. As and when more objects get created, the process of Garbage Collection is invoked by the CLR. Generation 1 - The objects that survive the garbage collection process are considered to be in generation 1. These are the old objects.Generation 2 - As more new objects get created and added to the memory, the new objects are added to generation 0, the generation 1 old objects become older, and so are considered to be in generation 2. Generation 2 is the highest level generation in the garbage collection process. Any further garbage collection process occuring causes the level 1 objects promoted to level 2, and the level 2 objects stay in level 2 itself, as this generation level is the highest level.

**7)Does c# supports destructors?**

yes .The destructor implicitly calls Finalize on the base class of the object.

### [Assemblies](http://csharpinterviewfaq.blogspot.in/2010/01/assemblies.html)

[**What is Namespace?**](http://www.dotnetinterviewfaqs.com/what-is-namespace-in-microsoft-dot-net.aspx)  
The namespace keyword is used to declare a scope. This namespace scope lets you organize code and gives you a way to create globally unique types.  
Within a namespace, you can declare one or more of the following types:  
· another namespace  
· [class](http://msdn.microsoft.com/en-us/library/0b0thckt.aspx)  
· [interface](http://msdn.microsoft.com/en-us/library/87d83y5b.aspx)  
· [struct](http://msdn.microsoft.com/en-us/library/ah19swz4.aspx)  
· [enum](http://msdn.microsoft.com/en-us/library/sbbt4032.aspx)  
· [delegate](http://msdn.microsoft.com/en-us/library/900fyy8e.aspx)  
Whether or not you explicitly declare a namespace in a C# source file, the compiler adds a default namespace. This unnamed namespace, sometimes referred to as the global namespace, is present in every file. Any identifier in the global namespace is available for use in a named namespace.  
Namespaces implicitly have public access and this is not modifiable.  
It is possible to define a namespace in two or more declarations.e.g: namespace MyCompany.Proj1{ class MyClass { }} namespace MyCompany.Proj1{ class MyClass1 { }}  
  
Namespaces have the following properties:  
· They organize large code projects.  
· They are delimited by using the . operator.  
· The using directive obviates the requirement to specify the name of the namespace for every class.  
· The global namespace is the "root" namespace: global::System will always refer to the .NET Framework namespace System.  
**Explain About using Directive?**  
The using directive has two uses:  
· To allow the use of types in a namespace so that you do not have to qualify the use of a type in that namespace  
· To create an alias for a namespace or a type. This is called a using alias directive.  
The scope of a using directive is limited to the file in which it appears. Create a using alias to make it easier to qualify an identifier to a namespace or type.  
Namespaces come in two categories: user-defined and system-defined. User-defined namespaces are namespaces defined in your code.  
[**What is an application domain?**](http://www.dotnetinterviewfaqs.com/what-is-application-domain-in-microsoft-dot-net.aspx)  
A boundary that the common language runtime establishes around objects created within the same application scope. Application domains help isolate objects created in one application from those created in other applications so that run-time behavior is predictable. Multiple application domains can exist in a single process.  
Microsoft.Net Assembly interview Questions & Answers  
[**What is an assembly in Microsoft.Net?**](http://www.dotnetinterviewfaqs.com/what-is-assembly-in-microsoft-dot-net.aspx)  
A collection of one or more files versioned and deployed as a unit. It contains code that the common language runtime executes. It forms a security boundary. An assembly is the unit at which permissions are requested and granted. It forms a type boundary. Every type's identity includes the name of the assembly in which it resides. It forms a reference scope boundary. It forms a version boundary. It forms a deployment unit.  
assemblies can be an efficient way to manage resources in larger projects.Assemblies can contain one or more modules. For example, larger projects may be planned in such a way that several individual developers work on separate modules, all coming together to create a single assembly.  
Assemblies have the following properties:  
· Assemblies are implemented as .exe or .dll files.  
· You can share an assembly between applications by putting it in the Global Assembly Cache.  
· Assemblies must be strong-named before they can be included in the Global Assembly Cache.  
· Assemblies are only loaded into memory if they are required.  
· You can programmatically obtain information about an assembly by using reflection  
· If you want to load an assembly only to inspect it, use a method such as ReflectionOnlyLoadFrom.  
· You can use two versions of the same assembly in a single application.  
Assemblies can be static or dynamic. Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on). Static assemblies are stored on disk in portable executable (PE) files. You can also use the .NET Framework to create dynamic assemblies, which are run directly from memory and are not saved to disk before execution. You can save dynamic assemblies to disk after they have executed.  
Assemblies can be private or shared. In order to share an assembly with other applications, it must be placed in the Global Assembly Cache (GAC).  
private assembly: An assembly that is available only to clients in the same directory structure as the assembly.  
shared assembly : An assembly that can be referenced by more than one application. An assembly must be explicitly built to be shared by giving it a cryptographically strong name.  
  
satellite assembly : A .NET Framework assembly containing resources specific to a given language. Using satellite assemblies, you can place the resources for difference languages in different assemblies, and the correct assembly is loaded into memory only if the user elects to view that application in that language.  
  
**What is Global Assembly Cache?**  
Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer.  
You should share assemblies by installing them into the global assembly cache only when you need to. As a general guideline, keep assembly dependencies private, and locate assemblies in the application directory unless sharing an assembly is explicitly required. In addition, it is not necessary to install assemblies into the global assembly cache to make them accessible to COM interop or unmanaged code.  
There are several ways to deploy an assembly into the global assembly cache:  
· Use an installer designed to work with the global assembly cache. This is the preferred option for installing assemblies into the global assembly cache.  
· Use a developer tool called the Global Assembly Cache tool (Gacutil.exe), provided by the Windows Software Development Kit (SDK).  
· Use Windows Explorer to drag assemblies into the cache.  
· Assemblies deployed in the global assembly cache must have a strong name. When an assembly is added to the global assembly cache, integrity checks are performed on all files that make up the assembly. The cache performs these integrity checks to ensure that an assembly has not been tampered with, for example, when a file has changed but the manifest does not reflect the change.  
**What is Strong Name?**A strong name consists of the assembly's identity—its simple text name, version number, and culture information (if provided)—plus a public key and a digital signature. It is generated from an assembly file using the corresponding private key.  
You can ensure that a name is globally unique by signing an assembly with a strong name. In particular, strong names satisfy the following requirements:  
· Strong names guarantee name uniqueness by relying on unique key pairs.  
· Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly.  
· Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built.  
· When you reference a strong-named assembly, you expect to get certain benefits, such as versioning and naming protection. If the strong-named assembly then references an assembly with a simple name, which does not have these benefits, you lose the benefits you would derive from using a strong-named assembly and revert to DLL conflicts. Therefore, strong-named assemblies can only reference other strong-named assemblies.  
**What is Asmbly Manifest?**  
An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. The assembly manifest can be stored in either a PE file (an .exe or .dll) with Microsoft intermediate language (MSIL) code or in a standalone PE file that contains only assembly manifest information.  
What are the functions An Assembly manifest do?  
· Enumerates the files that make up the assembly.  
· Governs how references to the assembly's types and resources map to the files that contain their declarations and implementations.  
· Enumerates other assemblies on which the assembly depends.  
· Provides a level of indirection between consumers of the assembly and the assembly's implementation details.  
· Renders the assembly self-describing.  
**List out the contents of assembly manifest ?**Assembly name  
Version number  
Culture  
Strong name information  
List of all files in the assembly  
Type reference information  
Information on referenced assemblies  
**List out the contents of Assembly?**In general, a static assembly can consist of four elements:  
· The [assembly manifest](http://msdn.microsoft.com/en-us/library/1w45z383.aspx), which contains assembly metadata.  
· Type metadata.  
· Microsoft intermediate language (MSIL) code that implements the types.  
· A set of resources.  
**Explain the grouping of elements in to an assembly?**  
· There are several ways to group these elements in an assembly. You can group all elements in a single physical file, which is shown in the following illustration.  
· Single-file assembly·  
· Alternatively, the elements of an assembly can be contained in several files. These files can be modules of compiled code (.netmodule), resources (such as .bmp or .jpg files), or other files required by the application.  
**When multifile assemblies required?**  
· Create a multifile assembly when you want to combine modules written in different languages and to optimize downloading an application by putting seldom used types in a module that is downloaded only when needed.  
· The files that make up a multifile assembly are not physically linked by the file system. Rather, they are linked through the assembly manifest and the common language runtime manages them as a unit.  
**What is MetaData?**  
Metadata is binary information describing your program that is stored either in a common language runtime portable executable (PE) file or in memory. When you compile your code into a PE file, metadata is inserted into one portion of the file, while your code is converted to Microsoft intermediate language (MSIL) and inserted into another portion of the file. Every type and member defined and referenced in a module or assembly is described within metadata. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on.  
Metadata describes every type and member defined in your code in a language-neutral manner.  
  
**What kind of information stored in Metadata?**  
Metadata stores the following information:  
· Description of the assembly.  
· Identity (name, version, culture, public key).  
· The types that are exported.  
· Other assemblies that this assembly depends on.  
· Security permissions needed to run.  
· Description of types.  
· Name, visibility, base class, and interfaces implemented.  
· Members (methods, fields, properties, events, nested types).  
· Attributes.  
· Additional descriptive elements that modify types and members.  
**What is Delaysigning?**  
An organization can have a closely guarded key pair that developers do not have access to on a daily basis. The public key is often available, but access to the private key is restricted to only a few individuals. When developing assemblies with strong names, each assembly that references the strong-named target assembly contains the token of the public key used to give the target assembly a strong name. This requires that the public key be available during the development process.  
You can use delayed or partial signing at build time to reserve space in the portable executable (PE) file for the strong name signature, but defer the actual signing until some later stage (typically just before shipping the assembly).  
**Explain the steps to delaysign an assembly?**  
The following steps outline the process to delay sign an assembly:  
1. Obtain the public key portion of the key pair from the organization that will do the eventual signing. Typically this key is in the form of an .snk file, which can be created using the [Strong Name tool (Sn.exe)](http://msdn.microsoft.com/en-us/library/k5b5tt23(VS.80).aspx) provided by the .NET Framework SDK.  
2. Annotate the source code for the assembly with two custom attributes from [System.Reflection](http://msdn.microsoft.com/en-us/library/system.reflection(VS.80).aspx):  
· [AssemblyKeyFileAttribute](http://msdn.microsoft.com/en-us/library/system.reflection.assemblykeyfileattribute(VS.80).aspx), which passes the name of the file containing the public key as a parameter to its constructor.  
· [AssemblyDelaySignAttribute](http://msdn.microsoft.com/en-us/library/system.reflection.assemblydelaysignattribute(VS.80).aspx), which indicates that delay signing is being used by passing true as a parameter to its constructor.  
· [assembly:AssemblyKeyFileAttribute("myKey.snk")]  
[assembly:AssemblyDelaySignAttribute(true)]  
3. The compiler inserts the public key into the assembly manifest and reserves space in the PE file for the full strong name signature. The real public key must be stored while the assembly is built so that other assemblies that reference this assembly can obtain the key to store in their own assembly reference.  
4. Because the assembly does not have a valid strong name signature, the verification of that signature must be turned off. You can do this by using the –Vr option with the Strong Name tool. 5. Later, usually just before shipping, you submit the assembly to your organization's signing authority for the actual strong name signing using the –R option with the Strong Name tool.  
The following example signs an assembly called myAssembly.dll with a strong name using the sgKey.snk key pair.  
sn -R myAssembly.dll sgKey.snk  
**Explain about friend assembly?**  
An internal type or internal member in an assembly can be accessed from another assembly.  
The friend assemblies feature allows you to access internal members; private types and private members will remain inaccessible.

**What is a satellite Assembly?**  
An assembly containing localized resources for another assembly.

**Can 2 different**[**applications**](http://www.newinterviewquestions.com/interview/8690/)**use the same dll in GAC at the same time?**

Yes. two [applications](http://www.newinterviewquestions.com/interview/8690/)can use the same dll. that is why we have shared assemblies and place it in the GAC.

**company sends dll's to the client. someclient is not happy current functionality, so requestsome modification. developer made some changes andsend new dll to all clients. Some client is happy withold version, tell me minimal change to so that neitherclients get affected?**

assign a strong name for that dll .so dat it will mantain different versions and public key token

### [Reflection](http://csharpinterviewfaq.blogspot.in/2010/01/reflection.html)

**What is Reflection?**  
**The process of obtaining information about assemblies and the types defined within them, and creating, invoking, and accessing type instances at run time.**Reflection provides objects that describe assemblies, modules and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, Reflection enables you to access them.  
Reflection provides infrastructure used by language compilers to implement implicit late binding.

Reflection allows known data types to be inspected at runtime. Reflection allows the enumeration of data types in a given assembly, and the members of a given class or value type can be discovered. This is true regardless of whether the type was known or referenced at compile time. This makes reflection a useful feature for development and code management tools.

**When Reflection is useful?**Reflection is useful in the following situations:  
· When you have to access attributes in your program's metadata.  
· For examining and instantiating types in an assembly.  
· For building new types at runtime. Use classes in System.Reflection.Emit.  
· For performing late binding, accessing methods on types created at run time.  
**What is late binding?**  
Binding is the process of locating the declaration that corresponds to a uniquely specified type. When this process occurs at run time rather than at compile time, it is called late binding.  
**What is Custom Binding?**  
The common language runtime supports multiple programming languages, and the binding rules of these languages differ. In the early-bound case, code generators can completely control this binding. However, in late binding through reflection, binding must be controlled by customized binding. The Binder class provides custom control of member selection and invocation.  
Using custom binding, you can load an assembly at run time, obtain information about types in that assembly, specify the type that you want, and then invoke methods or access fields or properties on that type. This technique is useful if you do not know an object's type at compile time, such as when the object type is dependent on user input.

**Which class has significant importance for reflection?**

The System.Type class is central to reflection. The common language runtime creates the Type for a loaded type when reflection requests it. You can use a Type object's methods, fields, properties, and nested classes to find out everything about that type.

**How do you get Type objects from assemblies that have not been loaded ?**

Use Assembly.GetType or Assembly.GetTypes to obtain Type objects from assemblies that have not been loaded, passing in the name of the type or types you want.

**How do you get to Type objects from an assembly that is already loaded?**

Use Type.GetType to get the Type objects from an assembly that is already loadedUse Type..::.GetType to get the Type objects from an assembly that is already loaded.

**How do you obtain module Type objects ?**

Use Module.GetType and Module.GetTypes to obtain module Type objects.

**Which property is used to determine whether the type is generic?**

Use the IsGenericType property to determine whether the type is generic, and use the IsGenericTypeDefinition property to determine whether the type is a generic type definition.

**Which name space contain the classes that are used to enables you to build types at run time?**

System.Reflection.Emit

**What is the use of System.Reflection namespace?**

The System.Reflection namespace contains types that retrieve information about assemblies, modules, members, parameters, and other entities in managed code by examining their metadata. These types also can be used to manipulate instances of loaded types, for example to hook up events or to invoke methods.

### [Generics](http://csharpinterviewfaq.blogspot.in/2010/01/generics.html)

**What are generics?**  
Generics are classes, structures, interfaces, and methods that have placeholders (type parameters) for one or more of the types that they store or use.

### [Delegates](http://csharpinterviewfaq.blogspot.in/2010/01/delegates.html)

**What is delegate?**

Delegates are objects you can use to call the methods of other objects.

A delegate is a type that safely encapsulates a method, similar to a function pointer in C and C++. Unlike C function pointers, delegates are object-oriented, type safe, and secure. The type of a delegate is defined by the name of the delegate.

Once a delegate is assigned a method, it behaves exactly like that method.

The delegate method can be invoked like any other method, with parameters and a return value.

Any method from any accessible class or struct that matches the delegate's signature, which consists of the return type and parameters, can be assigned to the delegate.

The method can be either static or an instance method. This makes it possible to programmatically change method calls, and also plug new code into existing classes. As long as you know the signature of the delegate, you can assign your own delegated method.

Delegates have the following properties:  
Delegates are like C++ function pointers but are type safe.  
Delegates allow methods to be passed as parameters.  
Delegates can be used to define callback methods.  
Delegates can be chained together; for example, multiple methods can be called on a single event.  
Methods do not have to match the delegate signature exactly.

**What is the difference between method overloading and delagate?**

In the context of method overloading, the signature of a method does not include the return value. But in the context of delegates, the signature does include the return value.

**Why callback methods use delegates?**

This ability to refer to a method as a parameter makes delegates ideal for defining callback methods. For example, a sort algorithm could be passed a reference to the method that compares two objects. Separating the comparison code allows for the algorithm to be written in a more general way.

**What is the importance of delegates and events in designing large applications?**

The use of delegates and events in the design of a large application can reduce dependenciesand the coupling of layers.This allows you to develop components that have a higher reusability factor.

**What are the uses of delegates/callback functions?**

1)to sort elements within an array.

2)callback functions are required for window procedures, hook procedures, asynchronous procedure calls.

3) to get assembly load/unload notifications, unhandled exception notifications, database/window state change notifications, file system change notifications, menu item selections, completed asynchronous operation notifications, filtering a set of items, and so on.

4)Delegates are useful in situations where you need an intermediary between a calling procedure and the procedure being called.

5)You can use delegates for other, non-event related tasks, such as free threading or with procedures that need to call different versions of functions at compile time.

e.g:suppose you have a classified-ad application that includes a list box with the names of cars. The ads are sorted by title, which is normally the make of the car. A problem you may face occurs when some cars include the year of the car before the make. The problem is that the built-in sort functionality of the list box sorts only by character codes; it places all the ads starting with dates first, followed by the ads starting with the make.  
To fix this, you can create a sort procedure in a class that uses the standard alphabetic sort on most list boxes, but is able to switch at run time to the custom sort procedure for car ads. To do this, you pass the custom sort procedure to the sort class at run time, using delegates.

6)The use of delegates promotes good separation of functionality between the database and the client code.

**What is Delegate invocation?**

A delegate object is normally constructed by providing the name of the method the delegate will wrap, or with an anonymous Method. Once a delegate is instantiated, a method call made to the delegate will be passed by the delegate to that method. The parameters passed to the delegate by the caller are passed to the method, and the return value, if any, from the method is returned to the caller by the delegate. This is known as invoking the delegate.

**What is asynchronous callback ?**

Delegate types are sealed—they cannot be derived from— and it is not possible to derive custom classes from Delegate. Because the instantiated delegate is an object, it can be passed as a parameter, or assigned to a property. This allows a method to accept a delegate as a parameter, and call the delegate at some later time. This is known as an asynchronous callback, and is a common method of notifying a caller when a long process has completed. When a delegate is used in this fashion, the code using the delegate does not need any knowledge of the implementation of the method being used. The functionality is similar to the encapsulation interfaces provide.

**When to Use Delegates Instead of Interfaces?**

Both delegates and interfaces enable a class designer to separate type declarations and implementation. A given interface can be inherited and implemented by any class or struct. A delegate can be created for a method on any class, as long as the method fits the method signature for the delegate. An interface reference or a delegate can be used by an object that has no knowledge of the class that implements the interface or delegate method. Given these similarities, when should a class designer use a delegate and when should it use an interface?  
Use a delegate in the following circumstances:  
An eventing design pattern is used.  
It is desirable to encapsulate a static method.  
The caller has no need to access other properties, methods, or interfaces on the object implementing the method.  
Easy composition is desired.  
A class may need more than one implementation of the method.  
Use an interface in the following circumstances:  
There is a group of related methods that may be called.  
A class only needs one implementation of the method.  
The class using the interface will want to cast that interface to other interface or class types.  
The method being implemented is linked to the type or identity of the class: for example, comparison methods.

**What are Events?**

Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.

Events have the following properties:  
The publisher determines when an event is raised; the subscribers determine what action is taken in response to the event.  
An event can have multiple subscribers. A subscriber can handle multiple events from multiple publishers.  
Events that have no subscribers are never called.  
Events are typically used to signal user actions such as button clicks or menu selections in graphical user interfaces.  
When an event has multiple subscribers, the event handlers are invoked synchronously when an event is raised.

Events can be used to synchronize threads.

In the .NET Framework class library, events are based on the EventHandler delegate and the EventArgs base class.

**How will you subscibe to events?**

Using event Handler

Define an event handler method whose signature matches the delegate signature for the event. For example, if the event is based on the EventHandler delegate type, the following code represents the method stub:

void HandleCustomEvent(object sender, CustomEventArgs a)

{ // Do something useful here. }

Use the addition assignment operator (+=) to attach your event handler to the event. In the following example, assume that an object named publisher has an event named RaiseCustomEvent. Note that the subscriber class needs a reference to the publisher class in order to subscribe to its events.

publisher.RaiseCustomEvent += HandleCustomEvent;

An event handler can also be added by using a lambda expression:

public Form1()  
{  
InitializeComponent();  
// Use a lambda expression to define an event handler.  
this.Click += (s,e) => { MessageBox.Show(  
((MouseEventArgs)e).Location.ToString());};  
}

To subscribe to events by using an anonymous method:

Use the addition assignment operator (+=) to attach your anonymous method to the event.

In the following example, assume that an object named publisher has an event named RaiseCustomEvent and that a CustomEventArgs class has also been defined to carry some kind of specialized event information. Note that the subscriber class needs a reference to publisher in order to subscribe to its events.

publisher.RaiseCustomEvent += delegate(object o, CustomEventArgs e)

{ string s = o.ToString() + " " + e.ToString();

Console.WriteLine(s);

};

**How to unsubscibe from event?**

To prevent your event handler from being invoked when the event is raised, unsubscribe from the event. In order to prevent resource leaks, you should unsubscribe from events before you dispose of a subscriber object. Until you unsubscribe from an event, the multicast delegate that underlies the event in the publishing object has a reference to the delegate that encapsulates the subscriber's event handler. As long as the publishing object holds that reference, garbage collection will not delete your subscriber object.

To unsubscribe from an event  
Use the subtraction assignment operator (-=) to unsubscribe from an event:

publisher.RaiseCustomEvent -= HandleCustomEvent;

**What is Multicast delegate?**

Multicast delegates can be defined as delegates that have more than one element in their invocation list. In other words, it is a delegate that is subscribed by more than one method.

A MulticastDelegate has a linked list of delegates, called an invocation list, consisting of one or more elements. When a multicast delegate is invoked, the delegates in the invocation list are called synchronously in the order in which they appear. If an error occurs during execution of the list then an exception is thrown.

**How will you handle Exceptions in multicast delegates?**

Suppose you have added multiple delegates to a single multicast delegate. Each of these individual delegates must be invoked, regardless of whether an unhandled exception is thrown within one of the delegates. But, once a delegate in a multicast delegate throws an unhandled exception, no more delegates are fired. You need a way to trap unhandled exceptions within each individual delegate while still allowing the rest of the delegates to fire.  
To avoid breaking the chain, you have to gracefully handle exceptions in all functions. Use the GetInvocationList method. This method returns each individual delegate from a multicast delegate and, by doing so, allows you to invoke each delegate within the try block of an exception handler.

**Expain Anonymous Methods in Delegates?**

Sometimes, you want to use a very small amount of code to call a delegate. Creating functions for such a small code will make the code cumbersome and difficult to read and debug. C# 2.0 comes with a new concept of Anonymous methods. By using anonymous methods, you reduce the overhead in instantiating delegates by eliminating the need of separate method. Here is the small example to illustrate this./\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
AddDelegate add = delegate (int k) {return a + b;};  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Explain Covariance and Contravariance in Delegates ?**

These provide a degree of flexibility while matching delegate methods with delegate signatures.

**Covariance:**Covariance permits a method with a derived return type to be used as a delegate. When a delegate method has a return type that is derived from the return type of delegate, it is called a covariant. Because the return type of method is derived from the return type of delegate, the type conversion is implicit. This enables you to create delegate methods that can be used by base and derived classes.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class BaseClass {

// Some functions}

public class DerivedClass : BaseClass

{ // Some functions}

public class MainClass{ // Define a delegate delegate BaseClass TestDelegate(); private static DerivedClass ReturnDerived()

{ return new DerivedClass(); }

public static void Main(string []args) { // Covariance allows this TestDelegate delg = ReturnDerived; }}/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Contravariance:  
Contravariance permits a method with derived parameters to be used as a delegate. When a delegate method signature has parameters that are derived from the delegate parameters, the method is said to be contravariant./\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
public class BaseClass  
{  
// Some functions  
}  
public class DerivedClass : BaseClass  
{  
// Some functions  
}  
public class MainClass  
{  
// Define a delegate  
delegate BaseClass TestDelegate(BaseClass baseClassArg);  
private static DerivedClass ReturnDerived(DerivedClass dr)  
{  
return dr;  
}  
public static void Main(string []args)  
{  
// Contravariance allows this  
TestDelegate delg = ReturnDerived;  
}  
}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**What is event Handler?**

Event handlers are nothing more than methods that are invoked through.

**How do you declare delegates?**

e.g:public delegate void ProcessBookDelegate(Book book);

**How do you instantiate a delegate?**

e.g:bookDB.ProcessPaperbackBooks(PrintTitle);

**How do you call a delegate?**

processBook(b);

**Explain how delegates are used in asynchronous programming?**

Delegates enable you to call a synchronous method in an asynchronous manner. When you call a delegate synchronously, the Invoke method calls the target method directly on the current thread. If the BeginInvoke method is called, the common language runtime (CLR) queues the request and returns immediately to the caller. The target method is called asynchronously on a thread from the thread pool. The original thread, which submitted the request, is free to continue executing in parallel with the target method. If a callback method has been specified in the call to the BeginInvoke method, the callback method is called when the target method ends. In the callback method, the EndInvoke method obtains the return value and any input/output or output-only parameters. If no callback method is specified when calling BeginInvoke, EndInvoke can be called from the thread that called BeginInvoke.

Note:Compilers should emit delegate classes with Invoke, BeginInvoke, and EndInvoke methods using the delegate signature specified by the user. The BeginInvoke and EndInvoke methods should be decorated as native. Because these methods are marked as native, the CLR automatically provides the implementation at class load time. The loader ensures that they are not overridden.

**Expain about BeginInvoke method of delegates?**

The BeginInvoke method initiates the asynchronous call. It has the same parameters as the method that you want to execute asynchronously, plus two additional optional parameters. The first parameter is an AsyncCallback delegate that references a method to be called when the asynchronous call completes. The second parameter is a user-defined object that passes information into the callback method. BeginInvoke returns immediately and does not wait for the asynchronous call to complete. BeginInvoke returns an IAsyncResult, which can be used to monitor the progress of the asynchronous call.

**Expain about EndInvoke method?**

The EndInvoke method retrieves the results of the asynchronous call. It can be called any time after BeginInvoke. If the asynchronous call has not completed, EndInvoke blocks the calling thread until it completes. The parameters of EndInvoke include the out and ref parameters

of the method that you want to execute asynchronously, plus the IAsyncResult returned by BeginInvoke.

**How to obtain Waithandle ?**

You can obtain a WaitHandle by using the AsyncWaitHandle()property of the IAsyncResult returned by BeginInvoke.

**Explain about WaitHandle?**

You can obtain a WaitHandle by using the AsyncWaitHandle()property of the IAsyncResult returned by BeginInvoke.The WaitHandle is signaled when the asynchronous call completes, and you can wait for it by calling the WaitOne()method.

If you use a WaitHandle, you can perform additional processing before or after the asynchronous call completes, but before calling EndInvoke to retrieve the results.

Note:The wait handle is not closed automatically when you call EndInvoke. If you release all references to the wait handle, system resources are freed when garbage collection reclaims the wait handle. To free the system resources as soon as you are finished using the wait handle, dispose of it by calling the WaitHandleClose()method. Garbage collection works more efficiently when disposable objects are explicitly disposed.

**How will you know when the asynchronous call completes?**

use the IsCompleted() property of the IAsyncResult returned by BeginInvoke .

**When Asynchronous operations are used?**

Asynchronous operations are typically used to perform tasks that might take a long time to complete, such as opening large files, connecting to remote computers, or querying a database. An asynchronous operation executes in a thread separate from the main application thread. When an application calls methods to perform an operation asynchronously, the application can continue executing while the asynchronous method performs its task.

**What are the design patterns .Net provides to accomplish asynchronous operations?**

The .NET Framework provides two design patterns for asynchronous operations:  
Asynchronous operations that use IAsyncResult objects.  
Asynchronous operations that use events.  
The IAsyncResult design pattern allows for a variety of programming models, but is more complex to learn and provides a flexibility that most applications do not require. Where possible, class library designers should implement asynchronous methods using the event-driven model.

**where Asynchronous programming feature supported in .Net?**

Asynchronous programming is a feature supported by many areas of the .NET Framework, including:  
File IO, Stream IO, Socket IO.  
Networking.  
Remoting channels (HTTP, TCP) and proxies.  
XML Web services created using ASP.NET.  
ASP.NET Web Forms.  
Message Queuing using the MessageQueue class.

**Suppose in a scenario if we want to display information only until 4.00 p.m and after that means after 4.00 p.m if any one tries to access the information it should give error mesage. Then how will you write a delegate for this ? Give coding. (Question from allinterviews.com)**

delegate void InformDelegate();  
class Program  
{  
static void Main(string[] args)  
{  
InformDelegate info = null;  
Display disp =new Display();  
if (DateTime.Now.Hour < 16) { info = new InformDelegate(disp.morning); } else { info = null;

}  
}  
class Display  
{  
public void morning()  
{  
Console.WriteLine("display");  
}  
public void Evening()  
{  
Console.WriteLine("You are not allowed to access the information in this time. Try tommarrow morning");  
}  
}

**What is covariance and contravariance? Do Delegate and method overriding support these?**

**how to store the value in textbox using delegates if we have two user control. the value will be called from one user control to another user control. Loading and unloading will be done through delegates.**

### [Constructors](http://csharpinterviewfaq.blogspot.in/2010/01/constructors.html)

**What is Constructor?**  
Constructors are special methods used to initialize types and create instances of types. A class or struct may have multiple constructors that take different arguments. Constructors enable the programmer to set default values, limit instantiation, and write code that is flexible and easy to read. If you do not provide a constructor for your object, C# will create one by default that instantiates the object and sets member variables to the default values Static classes and structs can also have constructors.  
**What is Private Constructor?**  
A private constructor is a special instance constructor. It is generally used in classes that contain static members only. If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances of this class. Private constructors are used to prevent creating instances of a class when there are no instance fields or methods or when a method is called to obtain an instance of a class. If all the methods in the class are static, consider making the complete class static.  
**What is static constuctor?**  
A static constructor is used to initialize any static data, or to perform a particular action that needs performed once only. It is called automatically before the first instance is created or any static members are referenced. Static constructors have the following properties: · A static constructor does not take access modifiers or have parameters. · A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced. · A static constructor cannot be called directly. · The user has no control on when the static constructor is executed in the program. · A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file. · Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method. Explain Guidelines for constructors? · Consider providing simple, ideally default, constructors. A simple constructor has a very small number of parameters, and all parameters are primitive types or enumerations. · Consider using a static factory method instead of a constructor if the semantics of the desired operation do not map directly to the construction of a new instance, or if following the constructor design guidelines feels unnatural. · Do use constructor parameters as shortcuts for setting main properties. · Do use the same name for constructor parameters and a property, if the constructor parameters are used to simply set the property. The only difference between such parameters and the properties should be casing. · Do minimal work in the constructor. Constructors should not do much work other than to capture the constructor parameters. The cost of any other processing should be delayed until required. · Do throw exceptions from instance constructors if appropriate. · Do explicitly declare the public default constructor in classes, if such a constructor is required. · Avoid having default constructors on structures. · Do not call virtual members on an object inside its constructors.  
[**How do I call one constructor from another in C#?**](http://www.dotnetinterviewfaqs.com/calling-one-constructor-from-another-in-c-sharp.aspx)  
You use : base (parameters) or : this (parameters) just before the actual code for the constructor, depending on whether you want to call a constructor in the base class or in this class.

### [LINQ](http://csharpinterviewfaq.blogspot.in/2010/01/linq.html)

**what are Lambda expressions?**   
  
A lambda expression is an anonymous function that can contain expressions and statements, and can be used to create delegates or expression tree types.   
All lambda expressions use the lambda operator =>, which is read as "goes to". The left side of the lambda operator specifies the input parameters (if any) and the right side holds the expression or statement block.   
  
The => operator has the same precedence as assignment (=) and is right-associative.   
  
Lambdas are used in method-based LINQ queries as arguments to standard query operator methods such as Where and Where(IQueryable, String, array[]).

**How do you assign a Lambda expression to a delegate?**

delegate int del(int i);   
del myDelegate = x => x \* x;   
int j = myDelegate(5); //j = 25

**Can we write a lamba expression on the left side of the is or as operator?**

Lambdas are not allowed on the left side of the is or as operator.

**What is Expression Lambda?**

A lambda expression with an expression on the right side is called an expression lambda. Expression lambdas are used extensively in the construction of Expression Trees. An expression lambda returns the result of the expression and takes the following basic form:

(input parameters) => expression

The parentheses are optional only if the lambda has one input parameter; otherwise they are required. Two or more input parameters are separated by commas enclosed in parentheses:

(x, y) => x == y

Sometimes it is difficult or impossible for the compiler to infer the input types. When this occurs, you can specify the types explicitly as shown in the following example:

(int x, string s) => s.Length > x

Specify zero input parameters with empty parentheses:

() => SomeMethod()

Note in the previous example that the body of an expression lambda can consist of a method call. However, if you are creating expression trees that will be consumed in another domain, such as SQL Server, you should not use method calls in lambda expressions. The methods will have no meaning outside the context of the .NET common language runtime.

**What is Statement Lambda?**

A statement lambda resembles an expression lambda except that the statement(s) is enclosed in braces:

(input parameters) => {statement;}

The body of a statement lambda can consist of any number of statements; however, in practice there are typically no more than two or three.

delegate void TestDelegate(string s);   
…   
TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); };   
myDel("Hello");

**What is the difference between Statement Lambdas and Expression Lambdas ?**

Expression lambdas are used extensively in the construction of Expression Trees.

Statement lambdas cannot be used to create expression trees.

**What is Type Inference in Lambdas ?**

When writing lambdas, you often do not have to specify a type for the input parameters because the compiler can infer the type based on the lambda body, the underlying delegate type, and other factors as described in the C# 3.0 Language Specification. For most of the standard query operators, the first input is the type of the elements in the source sequence. So if you are querying an IEnumerable, then the input variable is inferred to be a Customer object, which means you have access to its methods and properties:

customers.Where(c => c.City == "London");

**Explain Rules of lambdas?**   
The lambda must contain the same number of parameters as the delegate type.   
Each input parameter in the lambda must be implicitly convertible to its corresponding delegate parameter.   
The return value of the lambda (if any) must be implicitly convertible to the delegate's return type.

**Does lambda expressions in themselves do have a type?**

lambda expressions in themselves do not have a type because the common type system has no intrinsic concept of "lambda expression." However, it is sometimes convenient to speak informally of the "type" of a lambda expression. In these cases the type refers to the delegate type or Expression type to which the lambda expression is converted.

**Explain about Variable Scope in Lambda Expressions?**

Lambdas can refer to outer variables that are in scope in the enclosing method or type in which the lambda is defined. Variables that are captured in this manner are stored for use in the lambda expression even if variables would otherwise go out of scope and be garbage collected. An outer variable must be definitely assigned before it can be consumed in a lambda expression.

**List out the rules apply to variable scope in lambda expressions?**

A variable that is captured will not be garbage-collected until the delegate that references it goes out of scope.   
Variables introduced within a lambda expression are not visible in the outer method.   
A lambda expression cannot directly capture a ref or out parameter from an enclosing method.   
A return statement in a lambda expression does not cause the enclosing method to return.   
A lambda expression cannot contain a goto statement, break statement, or continue statement whose target is outside the body or in the body of a contained anonymous function.

**What are Expression Trees?**

Expression trees represent language-level code in the form of data. The data is stored in a tree-shaped structure. Each node in the expression tree represents an expression, for example a method call or a binary operation such as x <>

// Create an expression tree.

Expression<func> exprTree = num => num <>

// Decompose the expression tree.ParameterExpression param = (ParameterExpression)exprTree.Parameters[0];BinaryExpression operation = (BinaryExpression)exprTree.Body;ParameterExpression left = (ParameterExpression)operation.Left;ConstantExpression right = (ConstantExpression)operation.Right;

**are Expression Trees Immutable?**

Expression trees are immutable. This means that if you want to modify an expression tree, you must construct a new expression tree by copying the existing one and modifying it. You can use an expression tree visitor to traverse the existing expression tree.

**What is LINQ?**

Language-Integrated Query (LINQ) is a set of features in Visual Studio 2008 that extends powerful query capabilities to the language syntax of C# and Visual Basic. LINQ introduces standard, easily-learned patterns for querying and updating data, and the technology can be extended to support potentially any kind of data store.

**Why LINQ is required?**

Language-Integrated Query (LINQ) that bridges the gap between the world of objects and the world of data.

Traditionally, queries against data are expressed as simple strings without type checking at compile time or IntelliSense support. Furthermore, you have to learn a different query language for each type of data source: SQL databases, XML documents, various Web services, and so on. LINQ makes a query a first-class language construct in C# and Visual Basic. You write queries against strongly typed collections of objects by using language keywords and familiar operators.

</func

### [Arrays](http://csharpinterviewfaq.blogspot.in/2010/01/arrays.html)

**What is the difference between arrays in C# and arrays in other programming languages?**  
Arrays in C# work similarly to how arrays work in most other popular languages There are, however, a few differences as listed below1. When declaring an array in C#, the square brackets ([]) must come after the type, not the identifier. Placing the brackets after the identifier is not legal syntax in C#.  
int[] IntegerArray; // not int IntegerArray[];2. Another difference is that the size of the array is not part of its type as it is in the C language. This allows you to declare an array and assign any array of int objects to it, regardless of the array's length.int[] IntegerArray; // declare IntegerArray as an int array of any sizeIntegerArray = new int[10]; // IntegerArray is a 10 element arrayIntegerArray = new int[50]; // now IntegerArray is a 50 element arrayWhat are the 3 different types of arrays that we have in C#?1. Single Dimensional Arrays2. Multi Dimensional Arrays also called as rectangular arrays3. Array Of Arrays also called as jagged arraysAre arrays in C# value types or reference types?Reference types.  
**What is the base class for all arrays in C#?**  
System.Array  
**How do you sort an array in C#?**  
The Sort static method of the Array class can be used to sort array items.  
**Give an example to print the numbers in the array in descending order?**  
using System;  
namespace ConsoleApplication  
{class Program{static void Main()  
{int[] Numbers = { 2, 5, 3, 1, 4 };//Print the numbers in the array without sortingConsole.WriteLine("Printing the numbers in the array without sorting");  
foreach (int i in Numbers){Console.WriteLine(i);}//Sort and then print the numbers in the arrayConsole.WriteLine("Printing the numbers in the array after sorting");Array.Sort(Numbers);  
foreach (int i in Numbers){Console.WriteLine(i);}//Print the numbers in the array in desceding orderConsole.WriteLine("Printing the numbers in the array in desceding order");Array.Reverse(Numbers);  
foreach (int i in Numbers){Console.WriteLine(i);}}}}  
**What property of an array object can be used to get the total number of elements in an array?**  
Length property of array object gives you the total number of elements in an array. An example is shown below.  
using System;  
namespace ConsoleApplication  
{  
class Program  
{  
static void Main()  
{int[] Numbers = { 2, 5, 3, 1, 4 };  
Console.WriteLine("Total number of elements = " +Numbers.Length);  
}}}  
**Give an example to show how to copy one array into another array?**  
We can use CopyTo() method to copy one array into another array. An example is shown below.  
using System  
;namespace ConsoleApplication  
{  
class Program  
{  
static void Main(){int[] Numbers = { 2, 5, 3, 1, 4 };  
int[] CopyOfNumbers=new int[5];  
Numbers.CopyTo(CopyOfNumbers,0);  
foreach (int i in CopyOfNumbers){Console.WriteLine(i);  
}  
}  
}  
}  
3) **Which of these statements correctly declares a two-dimensional array in C#?**  
**int[,] myArray;**  
int[][] myArray;  
int[2] myArray;  
System.Array[2] myArray;  
**Can you store multiple data types in System.Array?**No.  
**What’s the difference between the System.Array.CopyTo() and System.Array.Clone()?**  
The Clone() method returns a new array (a shallow copy) object containing all the elements in the original array. The CopyTo() method copies the elements into another existing array. Both perform a shallow copy. A shallow copy means the contents (each array element) contains references to the same object as the elements in the original array. A deep copy (which neither of these methods performs) would create a new instance of each element's object, resulting in a different, yet identacle object.  
**How can you sort the elements of the array in descending order?**  
By calling Sort() and then Reverse() methods.

### [Strings](http://csharpinterviewfaq.blogspot.in/2010/01/strings.html)

**Write a method to reverse a string in C#?**  
public string Reverse(String str)  
{  
char[] arr = str.ToCharArray();  
Array.Reverse(arr);  
return new string(arr);  
}  
  
**Write a method to reverse the order of the words in a sentence? For example For example for a given string: "This is a pen", convert it to "pen a is This".**  
  
public static string WordReversal(string sentence)  
{  
string[] words = sentence.Split(' ');  
Array.Reverse(words);  
return string.Join(" ", words);  
}  
**Will the following code compile and run?**  
string str = null;  
Console.WriteLine(str.Length);  
The above code will compile, but at runtime System.NullReferenceException will be thrown.  
**How do you create empty strings in C#?**  
Using string.empty as shown in the example below.string EmptyString = string.empty;   
**How do you determine whether a String represents a numeric value?**  
To determine whether a String represents a numeric value use TryParse method as shown in the example below. If the string contains nonnumeric characters or the numeric value is too large or too small for the particular type you have specified, TryParse returns false and sets the out parameter to zero. Otherwise, it returns true and sets the out parameter to the numeric value of the string.  
string str = "One";  
int i = 0;  
if(int.TryParse(str,out i))  
{Console.WriteLine("Yes string contains Integer and it is " + i);  
}  
else  
{  
Console.WriteLine("string does not contain Integer");  
}

<http://programcall.com/10/interview/write-a-sample-program-that-implements-bubble-sort-in-csnet.aspx>

using System;  
  
class myclass  
{  
    static void Main()  
    {  
        int[] a = { 4, 6, 9, 83, 34, 45 };  
        int temp;  
  
        for (int pass = 1; pass <= a.Length - 2; pass++)  
        {  
            for (int i = 0; i <= a.Length - 2; i++)  
            {  
                if (a[i] > a[i + 1])  
                {  
                    temp = a[i + 1];  
                    a[i + 1] = a[i];  
                    a[i] = temp;  
                }  
  
            }  
  
        }  
  
        Console.WriteLine("The Sorted array");  
        foreach (int aa in a)  
            Console.Write(aa + " ");  
  
        Console.Read();  
    }  
}

**Output**  
The Sorted array  
4 6 9 34 45 83

Write a program to reverse a number using C#

using System;  
  
namespace Learn  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Enter a Number");  
            int numb = int.Parse(Console.ReadLine());  
            int reverse = 0;  
            while (numb > 0)  
            {  
                int rem = numb % 10;  
                reverse = (reverse \* 10) + rem;  
                numb = numb / 10;  
                 
            }  
            Console.WriteLine("Reverse number={0}", reverse);  
            Console.ReadLine();  
        }  
    }  
}

**Output  
----------**  
Enter a Number  
4567  
Reverse number=7654

Write a program to reverse a string in C#.NET

using System;  
  
namespace Learn  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            string Str, Revstr = "";   
            int Length;  
  
            Console.Write("Enter A String : ");  
            Str = Console.ReadLine();  
  
            Length = Str.Length - 1;  
              
            while (Length >= 0)  
            {  
  
                Revstr = Revstr + Str[Length];  
                Length--;  
  
            }  
  
            Console.WriteLine("Reverse  String  Is  {0}", Revstr);  
  
            Console.ReadLine();  
  
        }  
    }  
}

**OUTPUT  
-------------**Enter A String : HELLO  
Reverse  String  Is  OLLEH

Program to print Fibonacci Numbers below 100

using System;  
class myclass  
{  
    static void Main()  
    {  
        int fn = 0;  
        int sn = 1;  
        int tn = 1;  
  
        Console.WriteLine(fn);  
        Console.WriteLine(sn);  
        while (true)  
        {  
  
            tn = fn + sn;  
            if (tn >= 100)  
            {  
                break;  
            }  
            Console.WriteLine(tn);  
            fn = sn;  
            sn = tn;  
  
        }  
        Console.Read();  
  
    }  
}

**Output**0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55  
89

Write program to find LCM of 2 numbers in C#.NET

using System;  
  
public class FindLCM  
{  
    public static int determineLCM(int a, int b)  
    {  
        int num1, num2;  
        if (a > b)  
        {  
            num1 = a; num2 = b;  
        }  
        else  
        {  
            num1 = b; num2 = a;  
        }  
  
        for (int i = 1; i <= num2; i++)  
        {  
            if ((num1 \* i) % num2 == 0)  
            {  
                return i \* num1;  
            }  
        }  
        return num2;  
    }  
  
    public static void Main(String[] args)  
    {  
        int n1, n2;  
  
        Console.WriteLine("Enter 2 numbers to find LCM");  
  
        n1 = int.Parse(Console.ReadLine());  
        n2 = int.Parse(Console.ReadLine());  
  
        int result = determineLCM(n1, n2);  
  
        Console.WriteLine("LCM of {0} and {1} is {2}",n1,n2,result);  
        Console.Read();  
    }  
}

**Output**Enter 2 numbers to find LCM  
8  
12  
LCM of 8 and 12 is 24

Write a program to find GCD of 2 numbers using C#.NET

**Method 1**

**using System;  
  
class myclass  
{  
    static void Main()  
    {  
        int i1, i2;  
  
        Console.WriteLine("Enter 2 numbers to find GCD");  
        i1 = int.Parse(Console.ReadLine());  
        i2 = int.Parse(Console.ReadLine());  
  
        int n1, n2;  
        //Making sure n1 is greater than n2  
        if (i1 > i2)  
        {  
            n1 = i1;  
            n2 = i2;  
        }  
        else  
        {  
            n1 = i2;  
            n2 = i1;  
        }  
        int result = gcd(n1, n2);  
        Console.WriteLine("The GCD of {0} and {1} is {2}",i1,i2,result);  
        Console.Read();  
    }  
  
    private static int gcd(int n1, int n2)  
    {  
        int rem = 5;  
        while (n2 > 0)  
        {  
            rem = n1 % n2;  
            if (rem == 0)  
                return n2;  
            n1 = n2;  
            n2 = rem;  
  
        }  
        //gcd of any number with 0 is number itself.  
  
        return n1;  
  
    }  
}**

**Output**Enter 2 numbers to find GCD  
125  
85  
The GCD of 125 and 85 is 5  
  
  
**Method 2**Another and interesting logic for finding  GCD of 2 numbers

using System;  
  
class myclass  
{  
    public static int euclid(int a, int b)  
    {  
        if (b == 0)  
            return a;  
        else  
            return euclid(b, a % b);  
    }  
  
    public static void Main()  
    {  
        int n1 = 126, n2 = 45;  
        int gcd = euclid(n1, n2);  
        Console.WriteLine(gcd);  
  
        Console.Read();  
  
    }  
}

Write a Program to convert Decimal to Binary in C#.NET

using System;  
class myclass  
{  
    static void Main()  
    {  
        int num;  
        Console.Write("Enter a Number : ");  
        num = int.Parse(Console.ReadLine());  
        int quot;  
  
        string rem = "";  
  
        while (num >= 1)  
        {  
            quot = num / 2;  
            rem += (num % 2).ToString();  
            num = quot;  
        }  
  
        // Reversing the  value  
        string bin = "";  
        for (int i = rem.Length - 1; i >= 0; i--)  
        {  
            bin = bin + rem[i];  
  
        }  
  
        Console.WriteLine("The Binary format for given number is {0}",bin);  
  
  
        Console.Read();  
  
    }  
}

**Output**  
Enter a Number : 100  
The Binary format for given number is 1100100

### [Basic Programs useful for interviews](http://csharpinterviewfaq.blogspot.in/2010/03/basic-programs-useful-for-interviews.html)

**1) Write a program to find Factorial of a number in C#?**  
class Factorial  
{ private static int temp = 1;  
public static int CalFactorial(int n)  
  
{ temp = temp \* n;  
  
   return temp;   
}  
static void Main(string[] args)  
{  
Console.WriteLine("Enter A Number:");  
int fact = Convert.ToInt32(Console.ReadLine());  
int x = 1;   
for (int i = 1; i <= fact; i++)  
{  
x = Factorial.CalFactorial(i);  
}  
System.Console.WriteLine(x);  
Console.ReadLine();  
  }  
}

**Parallel.ForEach**

string[] lines = File.ReadAllLines(txtProxyListPath.Text);

List<string> list\_lines = new List<string>(lines);

Parallel.ForEach(list\_lines, line =>

{

//Your stuff

});

**Async Await**

Visual Studio 2012 introduced a simplified approach, async programming, that leverages asynchronous support in the .NET Framework 4.5 and higher as well as in the Windows Runtime.

*Async Methods*

// Three things to note in the signature:

// - The method has an async modifier.

// - The return type is Task or Task<T>. (See "Return Types" section.)

// Here, it is Task<int> because the return statement returns an integer.

// - The method name ends in "Async."

async Task<int> AccessTheWebAsync()

{

// You need to add a reference to System.Net.Http to declare client.

HttpClient client = new HttpClient();

// GetStringAsync returns a Task<string>. That means that when you await the

// task you'll get a string (urlContents).

Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");

// You can do work here that doesn't rely on the string from GetStringAsync.

DoIndependentWork();

// The await operator suspends AccessTheWebAsync.

// - AccessTheWebAsync can't continue until getStringTask is complete.

// - Meanwhile, control returns to the caller of AccessTheWebAsync.

// - Control resumes here when getStringTask is complete.

// - The await operator then retrieves the string result from getStringTask.

string urlContents = await getStringTask;

// The return statement specifies an integer result.

// Any methods that are awaiting AccessTheWebAsync retrieve the length value.

return urlContents.Length;

}

# Covariance and Contravariance