**What is Control.Invoke?**

Effectively, what Invoke does is ensure that the code you are calling occurs on the thread that the control "lives on" effectively preventing cross threaded exceptions.

**Control.InvokeRequired Property**

Gets a value indicating whether the caller must call an invoke method when making method calls to the control because the caller is on a different thread than the one the control was created on

**What does the tilde do like this?** ~Disposable()

{

Dispose(false);

}

That is called a [finalizer](http://msdn.microsoft.com/en-us/library/0s71x931.aspx) in C#. What it does is override the Finalize method using a syntax similar to a destructor.

**What is a destructor in C#?**

**What is an As Cast?**

**Tip:** For reference types, the as-cast is recommended. It is both fast and safe. If cast fails it returns null.

**Tip 2:** We can test the resulting variable against null and then use it. This eliminates extra casts.

**What Does Mutable Mean?**

Prone to frequent change as opposed to immutable which is rigid or unchanging.

**string concatenation V/S string builder.**

 A string is immutable; it cannot be modified in place.

The C# + concatenation operator builds a new string and causes reduced performance when it concatenates large amounts of text.

**StringBuilder** class is optimized for string concatenation. It provides the same benefits as using a C++ character array.

**1) What is a class ?**  
A class is the generic definition of what an object is . A Class describes all the attributes of object, as well as the methods that implements the behavior of member object.

**2) What is an Object?**  
Object is an instance of a class, it contains real values instead of variables.

**Why would we override ToString() in our classes?**

By default ToString() will return the fully qualified name of our class (Namespace.ClassName) which is not always very usefull.

If we override ToString() we can customize what is returned from the method call (like a comma separated list of properties)

**Why would we override Equals() in our classes?**

If the type is a reference type, then by default **"=="** operator checks for **reference equality** and **.Equals()** method checks for **value equality**. Let's understand what we mean by reference and value equality.

**3) What are the Access Modifiers in C# ?**

Different Access Modifier are - Public, Private, Protected, Internal, Protected Internal

* **Public** – When a method or attribute is defined as Public , It can be accessed from any code in project.
* **Private** - When a method or attribute is defined as Private , It can be accessed by any code within the containing type(class, method) only.
* **Protected** - The **protected** keyword is a member access modifier. A protected member is accessible within its class and by derived class instances.
* **Internal** – If an attribute or method is defined as Internal , Access is restricted to classes within the current project assembly
* **Protected Internal** – If an attribute or method is defined as Protected Internal , Access is restricted to classes within the current project assembly and types derived from the containing class.

**6. Define Constructors?**

A constructor is a member function in a class that has the same name as its class. The constructor is automatically invoked whenever an object class is created. It can construct and fill values of data members while initializing the class.

**7. What is Jagged Arrays?**

The array which has elements of type array is called jagged array. The elements can be of different dimensions and sizes. We can also call jagged array as Array of arrays.

**8. What is the difference between ref & out parameters?**

An argument passed as ref must be initialized before passing to the method whereas out parameter needs not to be initialized before passing to a method. The method using the **out** parameter has to set it to something.

**9. What is the use of using statement in C#?**

The using block is used to obtain a resource and use it and then automatically dispose of when the execution of block completed.

Provides a convenient syntax that ensures the correct use of [IDisposable](https://msdn.microsoft.com/en-us/library/system.idisposable.aspx) objects

.

**10. What is serialization?**

When we want to transport an object through network then we have to convert the object into a stream of bytes. The process of converting an object into a stream of bytes is called Serialization. For an object to be serializable, it should inherit ISerialize Interface.

Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

**11. Can “this” be used within a static method?**

No, ‘This’ in a static method because we can only use static variables/methods in a static method.

**12. What is difference between constants and read-only?**

**12. What is difference between constants, read-only and static?**

## Constant

## Constant fields or local variables must be assigned a value at the time of declaration and after that they cannot be modified. By default constant are static, hence you cannot define a constant type as static.

**Read Only**

A readonly field can be initialized either at the time of declaration or within the constructor of same class. Therefore, readonly fields can be used as run-time constants.

## Static

## The static keyword is used to specify a static member, which means static members are common to all the objects and they are not tied to a specific object.

A constant or type declaration is implicitly a static member.

**While an instance of a class contains a separate copy of all instance fields of the class, there is only one copy of each static field.**

**13. What is an interface class?**

Interface is an abstract class which has only public abstract methods and the methods only have the declaration and not the implementation. These abstract methods must be implemented in the implementing classes.

**What are value types and reference types?**

Value types are stored in the Stack whereas reference types stored on heap.

**Value types:** int, enum , byte, decimal, double, float, long

**Reference Types:**  string , class, interface, object

**16. What are sealed classes in C#?**

We create sealed classes when we want to restrict the class from being inherited. Sealed modifier used to prevent derivation from a class. If we forcefully specify a sealed class as base class then a compile-time error occurs.

**NOTE:** You can also use the **sealed** modifier on a method or property that overrides a virtual method or property in a base class. This enables you to allow classes to derive from your class and prevent them from overriding specific virtual methods or properties.

**17. What is method overloading?**

Method overloading is creating multiple methods with the same name with unique signatures in the same class. When we compile, the compiler uses overload resolution to determine the specific method to be invoke.

**18. What is the difference between Array and Arraylist?**

In an array, we can have items of the same type only. The size of the array is fixed. An arraylist is similar to an array but it doesn’t have a fixed size.

The internal structure of an ArrayList is an array.

ArrayList (System.Collections namespace) is a datatype collection. In order to fill an ArrayList, one can use the .Add method. ArrayLists are very dynamic in the sense that when you add and/or remove items from it, the performace stays the same.

**19. Can a private virtual method be overridden?**

No, because they are not accessible outside the class.

The virtual keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class. For example, this method can be overridden by any class that inherits it

**20. Describe the accessibility modifier “protected internal”.**

Protected Internal variables/methods are accessible within the same assembly and also from the classes that are derived from this parent class.

**21. What are the differences between System.String and System.Text.StringBuilder classes?**

System.String is immutable. When we modify the value of a string variable then a new memory is allocated to the new value and the previous memory allocation released. System.StringBuilder was designed to have concept of a mutable string where a variety of operations can be performed without allocation of separate memory location for the modified string.

**22. What’s the difference between the System.Array.CopyTo() and System.Array.Clone() ?**

Using Clone() method, we **creates a new array** object containing all the elements in the original array and using CopyTo() method, all the elements of existing array copies into another **existing** array. Both the methods perform a shallow copy.

**23. How can we sort the elements of the array in descending order?**

Using Sort() methods followed by Reverse() method.

**25.   What’s the difference between an interface and abstract class?**

**Interfaces**

An interface is a contract: the guy writing the interface says, "*hey, I accept things looking that way*", and the guy using the interface says "*OK, the class I write looks that way*".

All method signatures are by default public and you can not assign an access modifier to them.

**An interface is an empty shell**, there are only the signatures (name / params / return type) of the methods. The methods do not contain anything. The interface can't do anything. It's just a pattern.

**Abstract classes**

look a lot like interfaces, but they have something more : you can define a behavior for them. It's more about a guy saying, "these classes should look like that, and they have that in common, so fill in the blanks!".

**26. What is the difference between Finalize() and Dispose() methods?**

Dispose() is called when we want an object to release any unmanaged resources allocated to it. On the other hand Finalize() is used for the same purpose but it doesn’t assure the garbage collection of an object.

**27. What are circular references?**

Circular reference is situation in which two or more resources are interdependent on each other causes the lock condition and make the resources unusable.

**28. What are generics in C#.NET?**

Generics allow us to **design classes and methods decoupled from the data types**. Generic classes are extensively used by collection classes available in System.Collections.Generic namespace.

**30. List the commonly used types of exceptions in .Net?**

ArgumentException, ArgumentNullException , ArgumentOutOfRangeException, ArithmeticException, DivideByZeroException ,OverflowException , IndexOutOfRangeException ,InvalidCastException ,InvalidOperationException , IOEndOfStreamException , NullReferenceException , OutOfMemoryException , StackOverflowException etc.

**32. What is a C# delegate?**

A **delegate**is a type safe function pointer.That is, they hold reference(Pointer) to a function.

The **signature**of the delegate **must match** the signature of the function, the delegate points to, otherwise you get a compiler error. This is the reason delegates are called as type safe function pointers.

***Tip to remember delegate syntax:****Delegates syntax look very much similar to a method with a delegate keyword.*

**33. How do you mark a class to inherit from another class in C#?**

**COLON : OPERATOR**

**If class A interties from class B then class A is the derivative class and class B is the Base class.**

**34. What is the base class in .net from which all the classes are derived from?**

**System**

**35. What is the difference between method overriding and method overloading?**

In method overriding, we change the method behavior in the derived class from the behavior in the base class. **We override the behavior.**

Method overloading is creating a method with the same name within the same class having different signatures.

**37. Why can’t you specify the accessibility modifier for methods inside the interface?**

In an interface, we have virtual methods that do not have method definition. All the methods are there to be overridden in the derived class. That’s why they all are public.

**38. How can we set class to be inherited, but prevent the method from being over-ridden?**

Declare the class as public and make the method sealed to prevent it from being overridden.

**39. What happens if the inherited interfaces have conflicting method names?**

Implement is up to you as the method is inside your own class. There might be problem when the methods from different interfaces expect different data, but as far as compiler cares you’re okay.

**40. What is the difference between a Struct and a Class?**

Structs are value-type variables and classes are reference types. Structs stored on the stack, causes additional overhead but faster retrieval. Structs cannot be inherited.

**42. How we can create an array with non-default values?**

We can create an array with non-default values using Enumerable.Repeat.

IEnumerable<string> strings =

Enumerable.Repeat("I like programming.", 4);

foreach (String str in strings)

{

Console.WriteLine(str);

}

/\*

This code produces the following output:

I like programming.

I like programming.

I like programming.

I like programming.

**43. What is difference between is and as operators in c#?**

**“is”** operator is used to check the compatibility of an object with a given type **cast** and it returns the result as Boolean.

**“as**” operator is used for casting of object to a type or a class. if it fails it return null. **Very fast cast**.

**44. What’s a multicast delegate?**

A delegate having multiple handlers assigned to it is called multicast delegate. Each handler is assigned to a method.

The delegate is assigned multiple functions.

**45. What are indexers in C# .NET?**

Indexers are known as smart arrays in C#. It allows the instances of a class to be indexed in the same way as array.

**Allows client code to use [] notation on the class instance itself.**

**48. How to implement singleton design pattern in C#?**

In singleton pattern, a class can only have one instance and provides access point to it globally.

You are building an application in C#. You need a class that has only one instance, and you need to provide a global point of access to the instance.

public class Singleton

{

private static Singleton instance;

private Singleton() {}

public static Singleton Instance

{

get

{

if (instance == null)

{

instance = new Singleton();

}

return instance;

}

}

}

This implementation has two main advantages:

 Because the instance is created inside the **Instance** property method, the class can exercise additional functionality (for example, instantiating a subclass), even though it may introduce unwelcome dependencies.

 The instantiation is not performed until an object asks for an instance; this approach is referred to as *lazy instantiation*. Lazy instantiation avoids instantiating unnecessary singletons when the application starts.

**Much like static.**

**Singletons preserve the conventional class approach, and don't require that you use the static keyword everywhere. They may be more demanding to implement at first, but will greatly simplify the architecture of your program.**

**50. Is C# code is managed or unmanaged code?**

C# is managed code because Common language runtime can compile C# code to Intermediate language.

**==================================================================================================**

**What is indexer**

Indexers allow instances of a class or struct to be indexed just like arrays. Indexers resemble [properties](https://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx) except that their accessors take parameters.

The **virtual** keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class. For example, this method can be overridden by any class that inherits it

Shallow copying means that the copied object's fields will reference the same objects as the original object. To allow shallow copying, add the following Clone method to your class:

Deep copying or cloning means that the copied object's fields will reference new copies of the original object's fields. This method of copying is more time-consuming than the shallow copy. To allow deep copying, add the following Clone method to your class:

Cloning is the ability to make an exact copy (a clone) of an instance of a type. Cloning may take one of two forms: a shallow copy or a deep copy. Shallow copying is relatively easy. It involves copying the object that the Clone method was called on.

Software development hiring managers and potential interviewees will find these open-ended C# proficiency interview questions and answers useful.

Good help is hard to find. There is an art to finding a developer who fits well in your organization in terms of personality and work ethic; fortunately, it's more straightforward to determine their technical expertise.

I worked at a couple of places where development managers loved drilling job candidates on syntax while having them write code, and it was clearly a stressful experience for the interviewee. I do not like asking specific syntax questions during interviews since most developers do not know language syntax or class names off the top of their heads.

In terms of the basic interview, I prefer to ask open-ended questions where the candidate can explain concepts and how they would attack problems. Some organizations like to give candidates tests or have them eyeball code snippets to spot problems, and I think those are good evaluation tools.

This is the first installment in our series of programming language-specific development interview questions and answers. Here is a list of questions (and the answers to those questions) that will help you get a feel for a candidate's proficiency with C#. You can ask follow-up questions based on their replies.

Note: This content is also available as a[*downloadable PDF*](http://www.techrepublic.com/resource-library/downloads/c-developer-interview-cheat-sheet/).

## ****What are namespaces, and how they are used?****

Namespaces are used to organize classes within the .NET Framework. They dictate the logical structure of the code. They are analogous to Java packages, with the key difference being Java packages define the physical layout of source files (directory structure) while .NET namespaces do not. However, many developers follow this approach and organize their C# source files in directories that correlate with namespaces. The .NET Framework has namespaces defined for its many classes, such as System.Xml--these are utilized via the using statement. Namespaces are assigned to classes via the namespace keyword.

## ****What is a constructor?****

A constructor is a class member executed when an instance of the class is created. The constructor has the same name as the class, and it can be overloaded via different signatures. Constructors are used for initialization chores.

## ****What is the GAC, and where is it located?****

The GAC is the Global Assembly Cache. Shared assemblies reside in the GAC; this allows applications to share assemblies instead of having the assembly distributed with each application. Versioning allows multiple assembly versions to exist in the GAC--applications can specify version numbers in the config file. The gacutil command line tool is used to manage the GAC.

## ****Why are strings in C# immutable?****

Immutable means string values cannot be changed once they have been created. Any modification to a string value results in a completely new string instance, thus an inefficient use of memory and extraneous garbage collection. The mutable System.Text.StringBuilder class should be used when string values will change.

## ****What is DLL Hell, and how does .NET solve it?****

DLL Hell describes the difficulty in managing DLLs on a system; this includes multiple copies of a DLL, different versions, and so forth. When a DLL (or assembly) is loaded in .NET, it is loaded by name, version, and certificate. The assembly contains all of this information via its metadata. The GAC provides the solution, as you can have multiple versions of a DLL side-by-side.

## ****How are methods overloaded?****

Methods are overloaded via different signatures (number of parameters and types). Thus, you can overload a method by having different data types, different number of parameters, or a different order of parameters.

## ****How do you prevent a class from being inherited?****

The sealed keyword prohibits a class from being inherited.

## ****What is the execution entry point for a C# console application?****

The Main method.

## ****How do you initiate a string without escaping each backslash?****

You put an @ sign in front of the double-quoted string.

String ex = @"This has a carriage return\r\n"

## ****What is the difference between a struct and a class?****

Structs cannot be inherited. Structs are passed by value and not by reference. Structs are stored on the stack not the heap. The result is better performance with Structs.

## ****What is a singleton?****

A singleton is a design pattern used when only one instance of an object is created and shared; that is, it only allows one instance of itself to be created. Any attempt to create another instance simply returns a reference to the first one. Singleton classes are created by defining all class constructors as private. In addition, a private static member is created as the same type of the class, along with a public static member that returns an instance of the class. Here is a basic example:

public class SingletonExample {

private static SingletonExample \_Instance;

private SingletonExample () { }

public static SingletonExample GetInstance() {

if (\_Instance == null) {

\_Instance = new SingletonExample ();

}

return \_Instance;

}

}

## ****What is boxing?****

[Boxing](http://www.techrepublic.com/article/put-up-your-fists-and-start-boxing-with-net/5766513) is the process of explicitly converting a value type into a corresponding reference type. Basically, this involves creating a new object on the heap and placing the value there. Reversing the process is just as easy with unboxing, which converts the value in an object reference on the heap into a corresponding value type on the stack. The unboxing process begins by verifying that the recipient value type is equivalent to the boxed type. If the operation is permitted, the value is copied to the stack.