1. **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 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 is the process of converting a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit. The concept of boxing and unboxing underlies the C# unified view of the type system, in which a value of any type can be treated as an object.

1. T
2. T
3. T
4. T
5. Tt
6. T
7. T
8. T
9. t