*Class and Structures*

Classes and structs are two of the basic constructs of the common type system in the .NET Framework. Structs and classes can have methods, fields, properties, constructors, and other functionalities. Structs share some similarities with classes, but they also have certain restrictions.

Difference between them

***Classes:***

* The class is reference type in C# and it inherits from the System.Object Type.
* The class object is stored on the heap memory. The object will be under garbage collection and automatically removed when there is no reference to the created objects.
* Copying the contents of a reference type variable into another variable, copies the reference, which means you now have two references to the same somewhere else storage of the actual data. In other words, after the copy, changing the data in one reference will appear to affect the other as well, but only because you're really just looking at the same data both places.
* The class will have the default constructor and it have destructor.
* The class can inherit the interfaces, abstract classes, classes.
* The new keyword should be used to create the object for the class.
* The class can be declared as abstract, sealed class.
* The class can have the initializes fields.

***Structures:***

* The struct is value type in C# and it inherits from System.ValueType.
* The struct value will be stored on the stack memory.
* Copying the contents of a value type variable into another variable, copies the entire contents into the new variable, making the two distinct. In other words, after the copy, changes to one won't affect the other.
* The struct can't have the default constructor and destructor, and all fields must be initialized in parameterized constructor or after the struct is declared..
* The struct can only inherit the interfaces.
* The struct can instantiated without using the new keyword.
* The struct is by default sealed class hence it will not allow to inherit. It can't use the abstract, sealed, base keyword.
* The struct can't initialize at the time of declaration.

**Note: Boxing and unboxing operations are used to convert between a struct type and object.**

*Abstract class and Interface*

An abstract class is a special kind of class that cannot be instantiated. So the question is why we need a class that cannot be instantiated? An abstract class is only to be sub-classed (inherited from). In other words, it only allows other classes to inherit from it but cannot be instantiated. The advantage is that it enforces certain hierarchies for all the subclasses. In simple words, it is a kind of contract that forces all the subclasses to carry on the same hierarchies or standards.

An interface is not a class. It is an entity that is defined by the word Interface. An interface has no implementation it only has the signature or in other words, just the definition of the methods without the body. As one of the similarities to Abstract class, it is a contract that is used to define hierarchies for all subclasses or it defines specific set of methods and their arguments.

Difference between them

***Abstract class:***

* A class may inherit only one abstract class.
* An abstract class contains both abstract members (minimum one) and complete members.
* An abstract class can contain access modifiers for the subs, functions, properties.
* Complete Member of abstract class can be Static.
* Abstract classes can have consts, members, method stubs and defined methods.
* A child class can define abstract methods with the same or less restrictive visibility.
* Abstract class contains Constructors.
* We can’t use the new operator on it to create a new instance.

***Interface:***

* A class may inherit several interfaces.
* An interface contains only signatures of members.
* An interface cannot have access modifiers by default everything is assumed as public.
* Member of interface can’t be Static
* Interfaces can only have consts and methods.
* A class implementing an interface must define the methods with the exact same visibility.
* Interface doesn’t contains Constructors.
* The derived type is completely free in how to implement the interface.

**Note: You cannot include an abstract method declaration in a nonabstract class.**

**Note:** **Abstract classes are meant to be base classes only.**

**Note: By convention, interface names begin with a capital letter I as in IComparable, ICloneable and etc…**