**Value Type and Reference Type:**

1. All the value types derive from System.ValueType, which in-turn, derives from System.Object.
2. C# 2.0 introduced nullable types for value types so that you can assign null to a value type variable or declare a value type variable without assigning a value to it.
3. Value type stores the value in its memory space, whereas reference type stores the address of the value where it is stored.
4. Primitive data types and struct are of the 'Value' type. Class objects, string, array, delegates are reference types.
5. Value type passes byval by default. Reference type passes byref by default.
6. Value types and reference types stored in Stack and Heap in the memory depends on the scope of the variable.

**C# Interface:**

1. An **Interface** only contains declarations of method, events & properties.
2. An **Interface** can be implement implicitly or explicitly.
3. An **Interface** cannot include private members. All the members are public by default.

**String X StringBuilder**

1. StringBuilder is mutable.
2. StringBuilder performs faster than string when appending multiple string values.
3. Initialize StringBuilder as class e.g. StringBuilder sb = new StringBuilder()
4. Use StringBuilder when you need to append more than three or four strings.
5. Use Append() method to add or append strings with StringBuilder.
6. Use ToString() method to get the string from StringBuilder.

**C# Collection:**

We have learned about an array in the previous section. C# also includes specialized classes that hold many values or objects in a specific series, that are called 'collection'.

There are two types of collections available in C#: non-generic collections and [generic collections](http://www.tutorialsteacher.com/csharp/csharp-generic-collections). We will learn about non-generic collections in this section.

Every collection class implements the [*IEnumerable*](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx) interface so values from the collection can be accessed using a **foreach** loop.

The *System.Collections* namespace includes following non-generic collections.

| **Non-generic Collections** | **Usage** |
| --- | --- |
| [ArrayList](http://www.tutorialsteacher.com/csharp/csharp-arraylist) | ArrayList stores objects of any type like an array. However, there is no need to specify the size of the ArrayList like with an array as it grows automatically. |
| [SortedList](http://www.tutorialsteacher.com/csharp/csharp-sortedlist) | SortedList stores key and value pairs. It automatically arranges elements in ascending order of key by default. C# includes both, generic and non-generic SortedList collection. |
| [Stack](http://www.tutorialsteacher.com/csharp/csharp-stack) | Stack stores the values in LIFO style (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. C# includes both, generic and non-generic Stack. |
| [Queue](http://www.tutorialsteacher.com/csharp/csharp-queue) | Queue stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. C# includes generic and non-generic Queue. |
| [Hashtable](http://www.tutorialsteacher.com/csharp/csharp-hashtable) | Hashtable stores key and value pairs. It retrieves the values by comparing the hash value of the keys. |
| BitArray | BitArray manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0). |

**C# Indexer:**

An Indexer is a special type of property that allows a class or structure to be accessed the same way as array for its internal collection. It is same as property except that it defined with **this** keyword with square bracket and paramters.

Syntax:

Public <return type> this[<parameter type> index]

{

Get{

// return the value from the specified index

}

Set{

// set values at the specified index

}

}

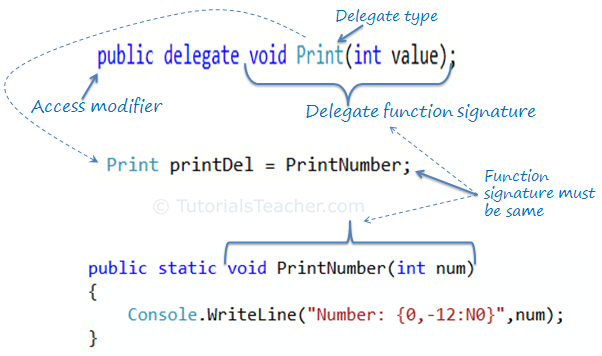
1. An indexer is same as property except that it defined with **this** keyword with square bracket that takes paramter.
2. Indexer can be override by having different types of parameters.
3. Ref and out parameter with the indexer is not supported.
4. Indexer can be included as an interface member.
5. Use code snippet to insert indexer syntax automatically in the visual studio.

**C# Delegate:**

A function can have one or more parameters of different data types, but what if you want to pass a function itself as a parameter? How does C# handle the callback functions or event handler? The answer is - **delegate**.

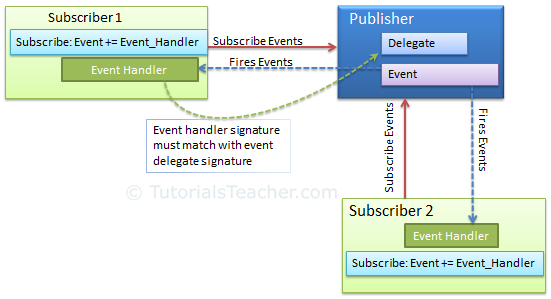
A delegate is like a pointer to a function. It is a reference type data type and it holds the reference of a method. All the delegates are implicitly derived from System.Delegate class.

A delegate can be declared using **delegate** keyword followed by a function signature as shown below.

[](http://www.tutorialsteacher.com/Content/images/csharp/delegate.png)

1. Delegate is a function pointer. It is reference type data type.
2. Syntax: *public delegate void <function name>(<parameters>)*
3. A method that is going to assign to delegate must have same signature as delegate.
4. Delegates can be invoke like a normal function or Invoke() method.
5. Multiple methods can be assigned to the delegate using "+" operator. It is called multicast delegate.

Event:

[](http://www.tutorialsteacher.com/Content/images/csharp/event-model.png)

1. Use event keyword with delegate type to declare an event.
2. Check event is null or not before raising an event.
3. Subscribe to events using "+=" operator. Unsubscribe it using "-=" operator.
4. Function that handles the event is called event handler. Event handler must have same signature as declared by event delegate.
5. Events can have arguments which will be passed to handler function.
6. Events can also be declared static, virtual, sealed and abstract.
7. An Interface can include event as a member.
8. Events will not be raised if there is no subscriber
9. Event handlers are invoked synchronously if there are multiple subscribers
10. The .NET framework uses an [EventHandler](http://msdn.microsoft.com/en-us/library/system.eventhandler.aspx) delegate and an [EventArgs](http://msdn.microsoft.com/en-us/library/system.eventargs.aspx) base class.

Static:

1. Static classes cannot be instantiated using the new keyword
2. Static items can only access other static items. For example, a static class can only contain static members, e.g., variables, methods, etc. A static method can only contain static variables and can only access other static items.
3. Static items share the resources between multiple users.
4. Static cannot be used with indexers, destructors or types other than classes.
5. A static constructor in a non-static class runs only once when the class is instantiated for the first time.
6. A static constructor in a static class runs only once when any of its static members accessed for the first time.
7. Static members are allocated in high frequency heap area of the memory.