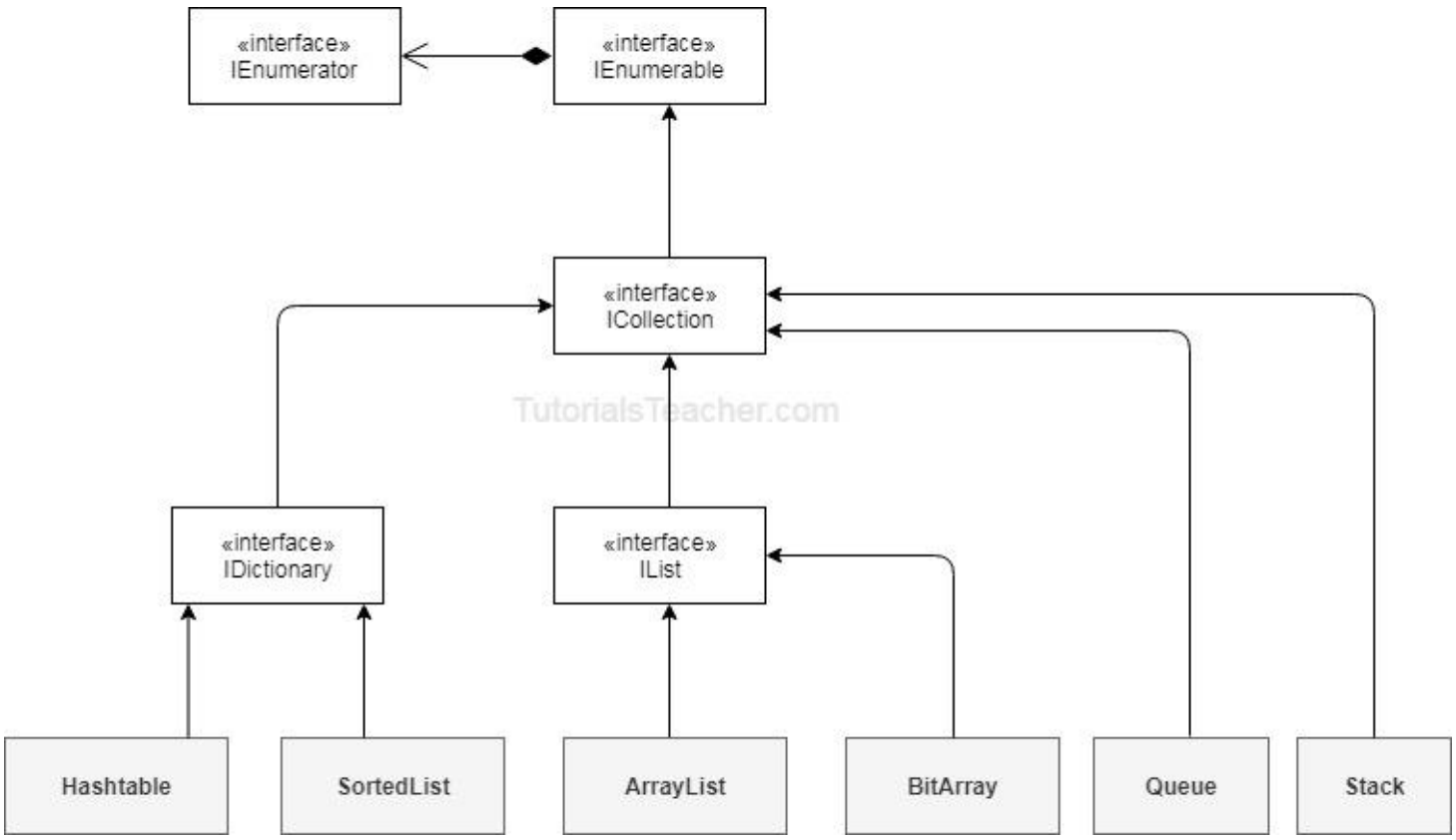
C# Collection:

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. We will learn about non-generic collections in this section.

The *System.Collections* namespace includes the interfaces and classes for the non-generic collections. The following diagram illustrates the hierarchy of the interfaces and classes for the non-generic collections.



C# Collections

As you can see in the above diagram, IEnumerator, IEnumerable, and ICollection are the top level interfaces for all the collections in C#.

IEnumerator: The IEnumerator interface supports a simple iteration over a non-generic collection. It includes methods and property which can be implemented to support easy iteration using foreach loop.

IEnumerable: The IEnumerable interface includes `GetEnumerator()` method which returns an object of IEnumerator.

So, all the built-in collection classes and custom collection classes must implement IEnumerator and IEnumerable interfaces for easy iteration using foreach loop.

ICollection: The ICollection interface is the base interface for all the collections that defines sizes, enumerators, and synchronization methods for all non-generic collections. The Queue and Stack collection implement ICollection inferface.

IList: The IList interface includes properties and methods to add, insert, remove elements in the collection and also individual element can be accessed by index. The ArrayList and BitArray collections implement IList interface.

IDictionary: The IDictionary interface represents a non-generic collection of key/value pairs. The Hashtable and SortedList implement IDictionary interface and so they store key/value pairs.

As you can see from the diagram, ArrayList, BitArray, Hashtable, SortedList, Queue, and Stack collections implement different interfaces and so, they are used for the different purposes.

Non-generic Collections	Usage
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	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	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	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	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).

Let's see each type of collection in the next chapters.

Why a Collection?

- Array size is fixed and cannot be increased dynamically. However, in actual development scenarios, the same type of objects are needed to be processed and added dynamically, and the size of the unit holding the objects should grow or shrink accordingly. This functionality is provided in Collections.
- The insertion and deletion of elements in an array is also costly in terms of performance.
- Also, the way these objects should be stored can be different, there can be a requirement to store the objects sequentially, non-sequentially, sorted order, etc.
- Collections come in handy as a return type in implementation of business methods, generally when data is obtained from a file or database, it can be a single or multiple objects. When one is not sure, it is safe to define return type as collection since it can hold multiple objects. This works even if a single or no object is returned.

Introduction

- A collection is a set of similar type of objects that are grouped together.
- *System.Collections* namespace contains specialized classes for storing and accessing the data.
- Each collection class defined in .NET has a unique feature.

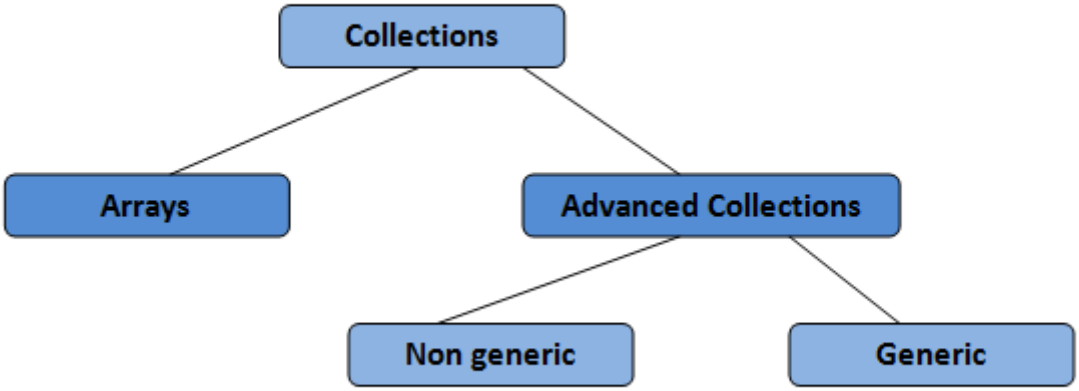
Collection Interfaces

There are some basic operations that are possible on any collection.

1. Search specific object in the collection.
2. Adding or removing objects dynamically in the collection.
3. List the objects in the collection by iterating through it. And so on

These basic operations are defined in the form of interfaces. All the collection classes implement these interfaces to get the required functionality. The implementation can be different for different classes. For example, adding objects in an *Array* is different than adding objects in *ArrayList* collection. *ArrayList* grows and shrinks dynamically. Both the collections need the functionality of iteration using *foreach* loop to display the list of objects contained in them.

Types of collections



- *Arrays*
  Array class is defined in *System* namespace. Arrays can store any type of data but only one type at a time. The size of the array has to be specified at compilation time. Insertion and deletion reduce the performance.

- *Advanced Collections*
  Many times we can't give number elements in the list and we need to perform different operations like inserting, deleting, sorting, searching, comparing, and so on. To perform these operations efficiently, the data needs to be organized in a specific way. This gives rise to advanced collections. Advanced collections are found in *System.Collections* namespace.

Advanced collections are again divided into two types-

Non-generic collections

Every element in non-generic collection is stored as *System.Object* type.

Examples

ArrayList, *Stack*, *Queue*, *HashTable*, and so on.

- *Boxing*
  Conversion of value type to a reference type is known as boxing. When value is boxed, CLR allocates new object on the heap and copies the value of the value type into that instance. CLR returns a reference of newly created object. This is essentially an up cast as all types are derived from *System.Object* class. Developer need not use wrapper classes or structures for value types to perform the conversion.

  Example

```
1. int speed =80
2. Object obj= speed;
```

- *Unboxing*
  It is an opposite operation of boxing, that is copies from reference type to value type on the stack. Explicit casting is required as it is a downcast. It is conversion of a derived type to a base type.

  Example

```
1. int speed =80
2. Object obj= speed // boxing
3. int speed=(int) obj // unboxing
```

Generic collections

These are defined in *System.Collections.Generic* namespace.

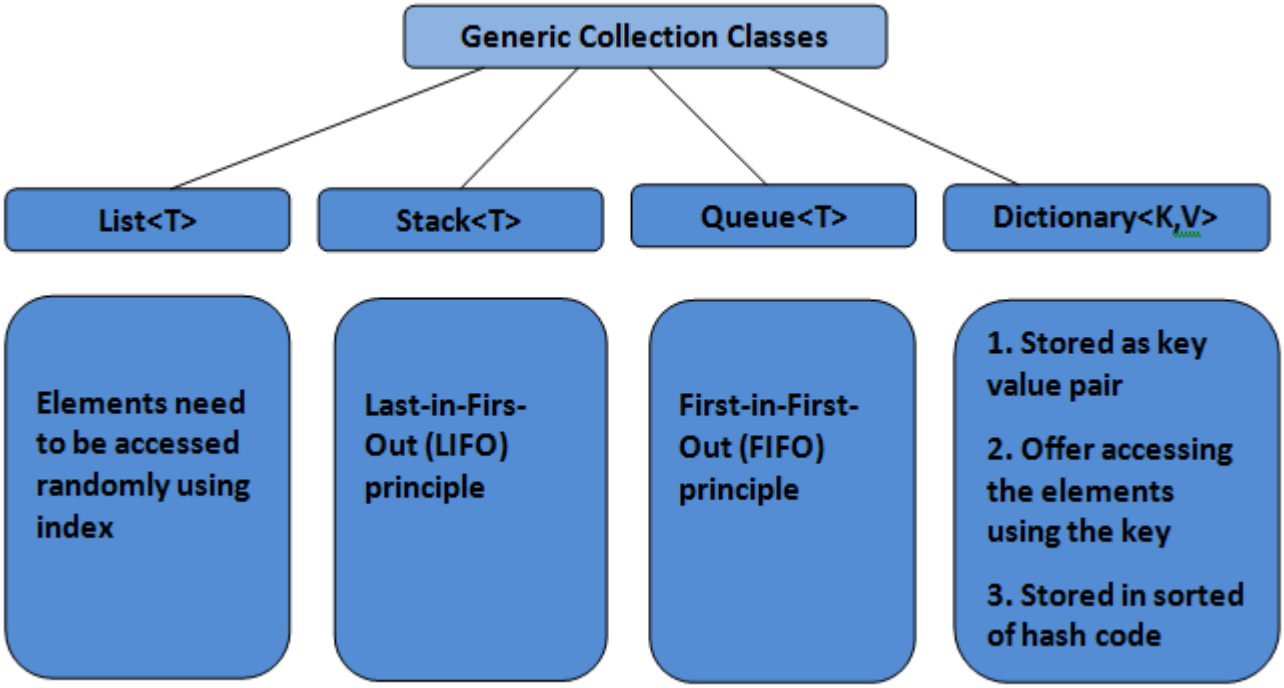
Examples

Generic list, generic queue, and so on. They are template based versions of their counterparts.

Generics help to define generic functions or classes which avoid repetition of code for different data types. Generic collections are very useful when implementing generic constructs like searching, sorting, stacks, Queues, lists, vectors, and so on. These constructs have a generic algorithm that can be implemented for any data type.

Data Type has to be specified at the time of instantiation of generic classes, thus providing type safety. For example, *int* data type is specified to instantiate *ArrayList* class. The methods of *HashTable* class also take parameter of type *K*. *K* is a placeholder. Compiler generates type specific implementation. The compiler does not create a brand new implementation of the generic type. It addresses only those methods and properties of the generic type that are actually invoked. Boxing, unboxing, and casting are not required as the stored elements in the generic collection are of specified type.

Some of the types of classes in generic collection are,



Advantages of generic collections

- No boxing and type casts are required, thereby improves the performance of application.
- Type errors are detected at compile time, thus avoids runtime errors.
- Generic code once written can be reused by instantiating the class with a specific type.

C# and .NET supports various collections. Collections in C# can be broken down into two categories, generic and non-generic. The following table lists and matches these classes.

Non-generic		Generic
ArrayList	---------->	List
HashTable	---------->	Dictionary
SortedList	---------->	SortedList
Stack	---------->	Stack
Queue	---------->	Queue

1. Non-Generic

a. Each element can represent a value of a different type.
b. Array Size is not fixed.
c. Elements can be added / removed at runtime.

ArrayList

1. Arraylist is a class that is similar to an array, but it can be used to store values of various types.
2. An Arraylist doesn't have a specific size.
3. Any number of elements can be stored.

```
1. using System.Collections;
2.
3. protected void Button1_Click(object sender, EventArgs e)
4. {
5.      ArrayList al = new ArrayList();
6.      string str = "kiran teja jallepalli";
7.      int x = 7;
8.      DateTime d = DateTime.Parse("8-oct-1985");
9.      al.Add(str);
10.           al.Add(x);
11.           al.Add(d);
12.
13.           foreach (object o in al)
14.           {
15.              Response.Write(o);
16.              Response.Write("<br>");
17.           }
18.      }
```

Output

kiran teja jallepalli
7
10/8/1985 12:00:00 AM

ForeachLoop

It executes for each and every item that exists in the arraylist object.

Every time the loop rotates it reads one item from the arraylist and assignes it to the variable.

Note:

Arraylist allocates memory for 4 items, whenever an object is created. When a fifth item is added, memory for another 4 items are added.
it reduces the memory allocated for the object.

Capacity: is a property that returns the number of items for which memory is allocated .

HashTable

HashTable is similar to arraylist but represents the items as a combination of a key and value.

```
1. using System.Collections;
2.

3. protected void Button2_Click(object sender, EventArgs e)
4. {
5.     Hashtable ht = new Hashtable();
6.     ht.Add("ora", "oracle");
7.     ht.Add("vb", "vb.net");
8.     ht.Add("cs", "cs.net");
9.     ht.Add("asp", "asp.net");
10.
11.          foreach (DictionaryEntry d in ht)
12.          {
13.             Response.Write (d.Key + " " + d.Value);
14.
15.             Response.Write("<br>");
16.
17.          }
18.      }
```

Output

vb vb.net

asp asp.net
cs cs.net
ora oracle

DictonaryEntry: is a class whose object represents the data in a combination of key & value pairs.

SortedList

1. Is a class that has the combination of arraylist and hashtable.
2. Represents the data as a key and value pair.
3. Arranges all the items in sorted order.

```
1. using System.Collections;
2. 
3. protected void Button3_Click(object sender, EventArgs e)
4. {
5.       SortedList sl = new SortedList();
6.       sl.Add("ora", "oracle");
7.       sl.Add("vb", "vb.net");
8.       sl.Add("cs", "cs.net");
9.       sl.Add("asp", "asp.net");
10.
11.            foreach (DictionaryEntry d in sl)
12.            {
13.                Response.Write(d.Key + " " + d.Value);
14.                Response.Write("<br>");
15.
16.            }
17.       }
```

Output

asp asp.net
cs cs.net
ora oracle
vb vb.net

Stack

```
1. protected void Button4_Click(object sender, EventArgs e)
2. {
3.     Stack stk = new Stack();
4.     stk.Push("cs.net");
5.     stk.Push("vb.net");
6.     stk.Push("asp.net");
7.     stk.Push("sqlserver");
8.
```

```
9.      foreach (object o in stk)
10.           {
11.              Response.Write(o + "<br>");
12.           }
13.      }
```

Output

sqlserver
asp.net
vb.net
cs.net

Queue

```
1. using System.Collections;
2.
3. protected void Button5_Click(object sender, EventArgs e)
4. {
5.      Queue q = new Queue();
6.      q.Enqueue("cs.net");
7.      q.Enqueue("vb.net");
8.      q.Enqueue("asp.net");
9.      q.Enqueue("sqlserver");
10.
11.          foreach (object o in q)
12.          {
13.             Response.Write(o + "<br>");
14.          }
15.      }
```

Output

cs.net
vb.net
asp.net
sqlserver

2. Generic Collections

Generic Collections work on the specific type that is specified in the program whereas non-generic collections work on the object type.

a. Specific type
b. Array Size is not fixed
c. Elements can be added / removed at runtime.

List

```
using System.Collections.Generic;

protected void Button1_Click(object sender, EventArgs e)
{
        List<int> lst = new List<int>();
        lst.Add(100);
        lst.Add(200);
        lst.Add(300);
        lst.Add(400);
             foreach (int i in lst)
             {
                 Response.Write(i+"<br>");
             }
      }
```

Dictonary

```
using System.Collections.Generic;

 protected void Button1_Click(object sender, EventArgs e)
 {
     Dictionary<int, string> dct = new Dictionary<int, string>();
     dct.Add(1, "cs.net");
     dct.Add(2, "vb.net");
     dct.Add(3, "vb.net");
     dct.Add(4, "vb.net");
          foreach (KeyValuePair<int, string> kvp in dct)
          {
              Response.Write(kvp.Key + " " + kvp.Value);
              Response.Write("<br>");
          }
       }
```

SortedList

```
using System.Collections.Generic;


protected void Button3_Click(object sender, EventArgs e)
{
SortedList<string, string> sl = new SortedList<string, string>();
sl.Add("ora", "oracle");
sl.Add("vb", "vb.net");
sl.Add("cs", "cs.net");
sl.Add("asp", "asp.net");

     foreach (KeyValuePair<string, string> kvp in sl)
     {
```

```
13.       Response.Write(kvp.Key + " " + kvp.Value);
14.       Response.Write("<br>");
15.       }
16.       }
```

Stack

```
1. using System.Collections.Generic;

2.

3. protected void Button4_Click(object sender, EventArgs e)
4. {
5. Stack<string> stk = new Stack<string>();
6. stk.Push("cs.net");
7. stk.Push("vb.net");
8. stk.Push("asp.net");
9. stk.Push("sqlserver");
10.
11.       foreach (string s in stk)
12.       {
13.       Response.Write(s + "<br>");
14.       }
15.       }
```

Queue

```
1. using System.Collections.Generic;

2. protected void Button1_Click(object sender, EventArgs e)
3.    {
4.    Queue<string> q = new Queue<string>();
5.
6.    q.Enqueue("cs.net");
7.    q.Enqueue("vb.net");
8.    q.Enqueue("asp.net");
9.    q.Enqueue("sqlserver");
10.
11.         foreach (string s in q)
12.         {
13.         Response.Write(s + "<br>");
14.         }
15.         }
```

Collections in C# is a must-know concept for every developer, whether entry level or an experienced one. There are some of the topics in Collections which are getting advanced with the release of new version of C#. So, let's learn about Collections.

What is “Collections” in C#?

Simply, Collection is a set of objects of similar type, grouped together.

All the collections implement IEnumrable interface which is inherited from ICollection interface. We can say that IEnumrable is the mother of all types of collections present in .NET.

- Standard Collection: This is found under system.Collections namespace.
- Generic Collection: This is found under system.Collections.Generic namespace. These are more flexible to work with data
- Index based: Array , List
- Key value pair: Hashtable , Dictionary
- Prioritized Collection: Stack & Queues
- Specialized Collection: String Collection

Working with Array

```
string[] a = new string[10];
```

1. Arrays are strongly typed entities.
2. It’s a fixed size entity/type which means, if we define an array of size 10, then it will have the space allocated for 10 elements and will need to loop through to find where the data is filled or not.
3. It’s a sequential collection which will be of the same type.
4. Elements can be accessed by using the index of an element.
5. Each index is initialized with a default value depending on the type of array created.

Working with ArrayList

1. It is a sophisticated version of an array.
2. It is one of the generic collection types present in System.Collection.Generic namespace.
3. An ArrayList can be used to create a collection of any type, such as String, Int16 etc., and even complex types.
4. The object stored in the list can be accessed by an Index.
5. Unlike arrays, List can grow its size automatically
6. List has methods to search, sort, and manipulate the list.
7. It uses both,the equality comparer as well as the ordering comparer.
8. List class is a strongly typed class.

Difference between Array and ArrayList

1. Arrays are used when the collection is of fixed length.
2. Array is faster than ArrayList because array doesn't require boxing / un-boxing.
3. Arrays are always strongly typed as compared to ArrayList.

4. ArrayList can grow automatically.

Working With Hashtable

```
Hashtable a = new Hashtable();
```

1. It is a collection of Key-value pairs that are organized on the hash code of the key.
2. It uses key to link the elements in the Collection.
3. Key can be of any type.
4. Key cannot be null.
5. Value against a key can be null.
6. It is not strongly typed.
7. Strong type of Hashtable is called Dictionary.

Difference between ArrayList and Hashtable

1. Hashtable uses the key to access a specific element in a Collection whereas the ArrayList uses index which is zero based.
2. ArrayList is faster than Hashtable because there is no hashcoding involved in it.

Working With Generic Collections

Generic separates the logic from the data type.

There are different variations of Collections, such as hash tables, queues, stacks, dictionaries, and ArrayList. There are some sorted generic Collections, as well. The element of those can be accessed by using a key as well as an index. But, for Generic Collection like Hashtable and Dictionaries, elements can be accessed using a key only. Every Collection has a method to add, remove, and search an element function present in the below namespace.

```
using System.Collections;
using System.Collections.Generic;
```

Working with Collection Interfaces : IEnumrable, IList, ICollection, IDictionary

These interfaces help us achieve two big concepts of OOPS- Encapsulation and Polymorphism.

IEnumerator

1. This helps to iterate over the Collection, without exposing the add, remove methods as in ArrayList, and only helps in browsing the Collection.
2. It has a method called GetEnumrator which returns IEnumerator object that helps to iterate over Collection using foreach loop.
3. Every .NET Collection implements IEnumerable interface.

ICollection

1. This helps to iterate over the Collection, without exposing the add, remove methods as in ArrayList, and also helps in browsing the Collection.
2. It is inherited from IEnumrable Interface.
3. It has extra count property which helps getting the count of elements in the Collection.

IList: This helps to iterate over the Collection along with having the add & remove functionality.

IDictionary: This helps to iterate over the Collection with a key-value pair and also provides with add & remove functionality.

IList

1. This helps to iterate over the Collection along with having the add & remove functionality.
2. It helps to access the element using the index.
3. It has the power of both, IEnumrable & ICollection interface.

IDictionary: This helps to iterate over the Collection with a key-value pair and also provides with add & remove functionality.

Difference between IEnumrable & IQueryable

1. IQueryable inherits from IEnumrable.
2. IQueryable executes the query on server side with all filters whereas IEnumrable executes the query and then filters the records.
3. IQueryable is suitable for out-memory operations whereas IEnumrable is suitable for in-memory operations, such as dataset, ArrayList.
4. IQueryable does not support further filtering whereas IEnumrable supports further filtering of data.
5. IQueryable supports lazing loading and is best suitable for paging whereas IEnumrable doesn’t, as it executes the query with all the filters.
6. IQueryable supports custom query whereas IEnumrable doesn’t support custom query.
7. IQueryable is derived from System.LINQ whereas IEnumrable derives from System.Collection.