5.Arrays and Collection in C#.Net

- Arrays allow you to store many values into one variable of the same type.
- It is not efficient to declare a separate variable for each value.
- Arrays in C# are reference types and are very much objects in their own right.

The following code shows how to declare references to an int array.

e.g
Int[] odd=new int[]{3,5,7,9};
Int[] odd ={3,5,7,9}; //which can be shortened to
***Declare an array:**
string[] x = {"I", "am", "learning", "C#");
Index 0: I
Index 1: am
Index 2: learning
Index 3: C#

5.1 Getting Array Size

Knowing the size of your array is important in looping through each element in an array. The loop must know when to stop. The demonstration below gives an example of how to get the size or obtain the amount of elements in an array.

5.1 Demo: The demonstration gives an example of how to get the size or obtain the amount of elements in an array

5.2 Multidimensional Arrays

❑ Multidimensional arrays are arrays within arrays also known as rectangular arrays. It may become necessary to nest an array in an array.

Syntax:
<Type>[,] ArrayName;

Arrays can be as many dimensions as you wish. The syntax for this is:
<Type>[,,,,,] <ArrayName>

Declare an array: Example 1:
string [,] Families = new string [2,3];

5.3 Array has Two Types: Rectangular Array , Jagged Array

5.3.1 Rectangular Array: In rectangular array every row is of same length.

```
*Declare Rectangular Arrays
        int[,] odd2;
        odd2 = new int[,] { { 3, 5, 7, 9 }, { 11, 13, 15, 17 } };
```

5.3.1 Demo: The demonstration gives an example of how to work with Rectangular array and what is drawback of it.

5.3.2 C# Jagged Arrays: They are more flexible than the rectangular arrays. With Jagged arrays the rows don't have to be the same size.

```
*Declare Jagged Array
string[][] Families = new string [2][];
Initialize Jagged Array
 string[][] Families = new string[2][] {{"Jamal", "Jane", "Mitch", "Anthony",
"Corel"}, {"Don", "Joan"}};
```

5.3.2 Demo: The demonstration gives an example of how to work with Jagged array and recover draw back of Rectangular array.

5.4 System.Array Class

The System.Array type is the abstract base type of all array types.
Following are the methods of System.Array:

1)CopyTo()
2)GetValue()
3)Setvalue()
4)GetUpperBound()
5)GetLowerBound()
6) GetLength()
7) GetType()

We have also used public **static** (shared) methods.

8) Copy()
9) Reverse()
10) Sort()
11) Clear()
12) IndexOf()

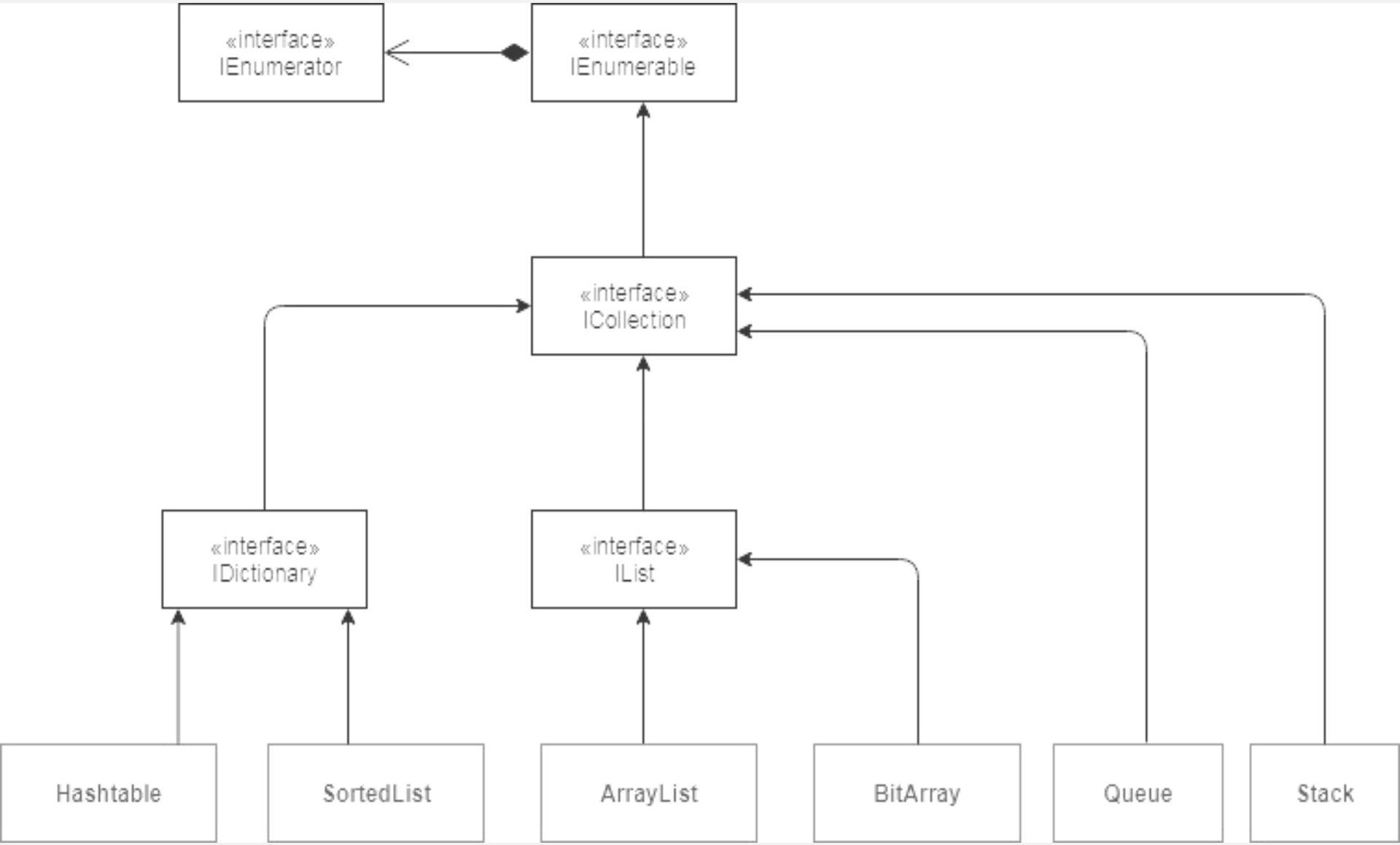
5.4 Demo: The demonstration gives an example of how different methods of System.Array class while working with arrays.

5.5 Collections (System.Collections) [Non-Generic]

System.Collections namespace that has been part of the .NET base class libraries since the initial release. The most primitive container construct would have to be our good friend System.Array. As you have already seen previously, this class provides a number of services (e.g., reversing, sorting, clearing, and enumerating). However, the simple Array class has a number of limitations; most notably, it does not automatically resize itself as you add or clear items. When you need to contain types in a more flexible container, one option is to leverage the types defined within the System.Collections namespace.

Classes from System.Collections:

1) ArrayList
2) Queue
3) Stack
4) HashTable


«interface»
IEnumerator
«interface»
IEnumerable
«interface»
ICollection
«interface»
IDictionary
«interface»
IList
Hashtable
SortedList
ArrayList
BitArray
Queue
Stack

5.5.1 ArrayList Class (System.Collections)

An ArrayList is a class. It uses an array whose size can be dynamically increased As required. ArrayList is more easier to work with multiple elements without bothering about index of an element. Array can only contain single type at time but ArrayList can contain variant type of data. Array size can not be re-declare but ArrayList can increase dynamically.

Properties
Capacity
Count

Methods
Add
Remove
RemoveRange
Contains

5.5.1 Demo: The demonstration gives an example of how manage student data using ArrayList class

5.5.2 Queue(System.Collections)

The Queue Class represents a FIFO(First In First Out) collection of Objects. The elements in a queue are added from one end and are removed from other. Queues are useful for storing messages in the order of which they arrive for sequential Processing. A queue can be imagined as a pipe where we stuff things from one end and Take them out from the other. The first to be pushed is the first to be taken out.

Properties
Count

Methods
Enqueue
Dequeue
Peek
Contains

5.5.2 Demo: The demonstration gives an example of how stock of Books can be Manage using Queue class.

5.5.3 Stack (System.Collections)

A stack is a Data Structure where elements are put in from an end and taken from the same. It represents a LIFO (Last in First Out).The first element is the Last to be taken out or processed where as the last to go in is t he first to come out In C# we have class from in the Collection namespace called the Stack class to Implement this data structure.

Properties
Count

Methods
Push
Pop

5.5.2 Demo: The demonstration gives an example of how stock of Books can be Manage using Stack class.

5.5.3 HashTable (System.Collections)

The Hashtable class represents pairs of associates keys and values .They are organized Based on the value of the hash code. When an entry is added to the Hashtable , the entry Is placed into bucket based on the hash code of the key. Subsequent lookups of the key to Search only one particular bucket.

Properties
Keys
Values

Methods
Add
Remove

5.5.3 Demo: The demonstration gives an example of how to Access name of fields using field caption and vice versa.

Collections (System.Collections.Generics)

The limitation of these Non-Generic collections is that while retrieving items, you need to cast into the appropriate data type, otherwise the program will throw a runtime exception. It also affects on performance, because of boxing and unboxing.

To overcome this problem, C# includes generic collection classes in the ***System.Collections.Generic*** namespace.

The following are widely used generic collections:

Generic Collections	Description
List<T>	Generic List<T> contains elements of specified type. It grows automatically as you add elements in it.
Dictionary<TKey,TValue>	Dictionary<TKey,TValue> contains key-value pairs.
SortedList<TKey,TValue>	SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default.
Hashset<T>	Hashset<T> contains non-duplicate elements. It eliminates duplicate elements.
Queue<T>	Queue<T> 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.
Stack<T>	Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values.

List<T> (System.Collections.Generics)

You have already learned about ArrayList in the previous section. An ArrayList resizes automatically as it grows. The List<T> collection is the same as an ArrayList except that List<T> is a generic collection whereas ArrayList is a non-generic collection.

Property	Usage
Items	Gets or sets the element at the specified index
Count	Returns the total number of elements exists in the List<T>

Method	Usage
Add	Adds an element at the end of a List<T>.
AddRange	Adds elements of the specified collection at the end of a List<T>.
Clear	Removes all the elements from a List<T>.
Contains	Checks whether the speciied element exists or not in a List<T>.
Insert	Inserts an element at the specified index in a List<T>.
Remove	Removes the first occurence of the specified element.
RemoveAt	Removes the element at the specified index.
RemoveRange	Removes all the elements that match with the supplied predicate function.
Sort	Sorts all the elements.

Use the `IList.Add()` method to add an element into a List collection.

```
IList<int> intList = new List<int>();
intList.Add(10);

intList.Add(20);

intList.Add(30);

intList.Add(40);
```

```
IList<Student>      studentList      =      new
List<Student>();
 studentList.Add(new Student());

 studentList.Add(new Student());

 studentList.Add(new Student());
```

Dictionary<TKey, TValue> (System.Collections.Generics)

The Dictionary<TKey, TValue> collection in C# is same as English dictionary. English dictionary is a collection of words and their definitions, often listed alphabetically in one or more specific languages.

In the same way, the Dictionary in C# is a collection of Keys and Values, where key is like word and value is like definition.

Property	Description
Count	Gets the total number of elements exists in the Dictionary<TKey,TValue>.
IsReadOnly	Returns a boolean indicating whether the Dictionary<TKey,TValue> is read-only.
Item	Gets or sets the element with the specified key in the Dictionary<TKey,TValue>.
Keys	Returns collection of keys of Dictionary<TKey,TValue>.
Values	Returns collection of values in Dictionary<TKey,TValue>.

Method	Description
Add	Add key-value pairs in Dictionary<TKey, TValue> collection.
Remove	Removes the first occurrence of specified item from the Dictionary<TKey, TValue>.
ContainsKey	Checks whether the specified key exists in Dictionary<TKey, TValue>.
ContainsValue	Checks whether the specified key exists in Dictionary<TKey, TValue>.
Clear	Removes all the elements from Dictionary<TKey, TValue>.

Example: Dictionary Initialization

```
IDictionary<int, string> dict = new Dictionary<int, string>();

 //or
Dictionary<int, string> dict = new Dictionary<int, string>();
```

SortedList<TKey, TValue> (System.Collections.Generics)

The generic SortedList SortedList<TKey, TValue> represents a collection of key-value pairs that are sorted by key based on associated IComparer<T>. A SortedList collection stores key and value pairs in ascending order of key by default.

Property	Description
Capacity	Gets or sets the number of elements that the SortedList<TKey,TValue> can store.
Count	Gets the total number of elements exists in the SortedList<TKey,TValue>.
IsReadOnly	Returns a boolean indicating whether the SortedList<TKey,TValue> is read-only.
Item	Gets or sets the element with the specified key in the SortedList<TKey,TValue>.
Keys	Get list of keys of SortedList<TKey,TValue>.
Values	Get list of values in SortedList<TKey,TValue>.

Method	Description
Add	Add key-value pairs into SortedList<TKey, TValue>.
Remove	Removes element with the specified key.
ContainsKey	Checks whether the specified key exists in SortedList<TKey, TValue>.
ContainsValue	Checks whether the specified key exists in SortedList<TKey, TValue>.
Clear	Removes all the elements from SortedList<TKey, TValue>.
IndexOfKey	Returns an index of specified key stored in internal array of SortedList<TKey, TValue>.
IndexOfValue	Returns an index of specified value stored in internal array of SortedList<TKey, TValue>

```
SortedList<string,int> sortedList2 = new SortedList<string,int>();
 sortedList2.Add("one", 1);
sortedList2.Add("two", 2);
sortedList2.Add("three", 3);
sortedList2.Add("four", 4);
// Compile time error: cannot convert from <null> to <int>
// sortedList2.Add("Five", null);
SortedList<double,int?> sortedList3 = new SortedList<double,int?>();
sortedList3.Add(1.5, 100); sortedList3.Add(3.5, 200);
sortedList3.Add(2.4, 300); sortedList3.Add(2.3, null);
 sortedList3.Add(1.1, null);
```

Minutes of Chapter

- Array
  - Intro. To Array –Single-Dimension Array
  - Multi-Dimension Array
  - Rectangular Array and Jagged Array
- Collections
  - Intro. To Collections
  - ArrayList
  - Queue
  - Stack
  - HashTable

Assignments for Chapter

- Develop program to print numbers in above format using Array.

11	12	13
14	15	16
17	18	19

- Develop program to print numbers in above format using Array.

A		
B	C	
D	E	F