# System.Collections

## Collection Classes Overview

- Collection classes are either in the namespace
  - `System.Collections`
  - `System.Collections.Generic`
- Using generic collection classes provides increased type-safety and in some cases can provide better performance, especially when storing value types.

## System.Collections

- Interfaces
  - **IEnumerator** and **IEnumerable<T>**
  - **ICollection**
  - **IList**

## Recall `IEnumerator` and `IEnumerable` interfaces

- Write a class which uses yield statement to display tables for any given number up to the specified number of times.

## `IComparer`

- Exposes a method that compares two objects.
- Method
  - **`int Compare (Object x, Object y ):`**

  Compares two objects and returns a value indicating whether one is less than, equal to, or greater than the other.

## IEqualityComparer

- Defines methods to support the comparison of objects for equality.
- Methods
  - **`bool Equals (Object x, Object y )`**

    Determines whether the specified objects are equal.
  - **`int GetHashCode( Object obj )`**

    Returns a hash code for the specified object.

## ICollection

- Defines size, enumerators, and synchronization methods for all non-generic collections.
- Properties and methods
  - **`Count:`** Gets the number of elements contained in the ICollection.
  - **`IsSynchronized:`** Gets a value indicating whether access to the ICollection is synchronized (thread safe).
  - **`SyncRoot:`** Gets an object that can be used to synchronize access to the ICollection
  - **`void CopyTo(Array array, int index ) :`** Copies the elements of the ICollection to an Array, starting at a particular Array index

## **IList**

- Represents a non-generic collection of objects that can be individually accessed by index.
- Implements **`ICollection`**, **`IEnumerable`**
- Properties and methods
  - **`IsFixedSize:`** Gets a value indicating whether the IList has a fixed size.
  - **`IsReadOnly`** : Gets a value indicating whether the **`IList`** is read-only.
  - **`Item:`** Gets or sets the element at the specified index.
  - **`int Add ( Object value ) :`** Adds an item to the IList.
  - **`void Clear`** : Removes all items from the **`IList`**.

- **`bool Contains ( Object value )`** : Determines whether the **`IList`** contains a specific value.
- **`int IndexOf ( Object value )`**: Determines the index of a specific item in the **`IList`**.
- **`void Insert ( int index, Object value`** ) Inserts an item to the **`IList`** at the specified index.
- **`void Remove ( Object value )`** Removes the first occurrence of a specific object from the **`IList`**.
- **`void RemoveAt ( int index )`** Removes the **`IList`** item at the specified index.

## IDictionary

- Represents a non-generic collection of key/value pairs.
- Implements **`ICollection`**, **`IEnumerable`**
- Properties and methods:
  - **`IsFixedSize`** Gets a value indicating whether the **`IDictionary`** object has a fixed size.
  - **`IsReadOnly`** Gets a value indicating whether the **`IDictionary`** object is read-only.
  - **`Item`** Gets or sets the element with the specified key.
  - **`Keys`** Gets an **`ICollection`** object containing the keys of the **`IDictionary`** object.
  - **`Values`** Gets an **`ICollection`** object containing the values in the **`IDictionary`** object.

- **`void Add ( Object key, Object value ):`** Adds an element with the provided key and value to the **`IDictionary`** object.
- **`void Clear ():`** Removes all elements from the **`IDictionary`** object.
- **`bool Contains ( Object key ):`** Determines whether the **`IDictionary`** object contains an element with the specified key.
- **`IDictionaryEnumerator GetEnumerator ():`** Returns an **`IDictionaryEnumerator`** object for the IDictionary object.
- **`void Remove ( Object key ):`** Removes the element with the specified key from the **`IDictionary`** object.

Another interface of the collections

## `ArrayList` Class

- Implements the **`IList`** interface using an array whose size is dynamically increased as required.

```
using System;
using System.Collections;
public class NameList  {
   public static void Main()  {
      ArrayList list = new ArrayList();
      list.Add("Sheela");
      list.Add("Soham");
      list.Add("Susan");
Console.WriteLine( "    Count:    {0}", list.Count );
Console.WriteLine( "    Capacity: {0}", list.Capacity );
Console.Write( "    Values:" );
 foreach ( Object obj in list )
    Console.Write( "   {0}", obj );
       Console.WriteLine();
    }
}
```

```
Count:    3
 Capacity: 4
 Values:    Sheela    Soham    Susan
```

## `BitArray` Class

- 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).
- Implements **`ICollection, IEnumerable, ICloneable`**
- Constructors
  - **`BitArray (Int32)`**
  - **`BitArray (BitArray)`**
  - **`BitArray (Boolean[])`**
  - **`BitArray (Byte[])`**
  - **`BitArray (Int32, Boolean)`**

## BitArray methods

- Apart from the methods of the interfaces that BitArray implements, other useful methods are
- **`public BitArray Not()`** : Inverts all the bit values in the current BitArray, so that elements set to true are changed to false, and elements set to false are changed to true.
- **`public BitArray Or(BitArray value)`** :Performs the bitwise OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.
- **`public void Set(int index, bool value )`** :Sets the bit at a specific position in the BitArray to the specified value
- **`public void SetAll(bool value):`** Sets all bits in the BitArray to the specified value.
- **`public BitArray Xor(BitArray value):`** Performs the bitwise exclusive OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.

```
using System;
using System.Collections;
public class BitArrayEx  {
   public static void Main()  {
BitArray b1 = new BitArray( 5 );
BitArray b2 = new BitArray( 5, true);
byte[] bs = new byte[5] { 1, 2, 3, 4, 5 };
BitArray b3 = new BitArray( bs );

Console.WriteLine( "b1" );
Console.WriteLine( "   Count:    {0}",
b1.Count );
Print( b1 );
Console.WriteLine( "b2 values"  );
Print( b2 );
Console.WriteLine( "b3 values" );
Print( b3);
   }
public static void Print( IEnumerable myList)
{
        foreach ( Object obj in myList ) {
           Console.Write( "{0}, ", obj );
        }
        Console.WriteLine();
    }}
```

```
Count:    5
False, False, False, False, False,
b2 values
True, True, True, True, True,
b3 values
True, False, False, False, False, False, False, False, False, True, False,
False, False, False, False, False, True, True, False, False, False, False,
False, False, False, False, True, False, False, False, False, False, True,
False, True, False, False, False, False, False,
```

*Take a look at b3 values- can you guess what it is printing?– move on to the next slide for a clue*

## Big Clue!

*Let us try and print 8 objects in a row and analyse*

```
public static void Print( IEnumerable myList)
{
        foreach ( Object obj in myList ){
            Console.Write( "{0}, ", obj );
        }
        Console.WriteLine();
    }}
```

```
b3 values
   True,  False,  False,  False,  False,  False,  False,  False,
  False,   True,  False,  False,  False,  False,  False,  False,
   True,   True,  False,  False,  False,  False,  False,  False,
  False,  False,   True,  False,  False,  False,  False,  False,
   True,  False,   True,  False,  False,  False,  False,  False,
```

## Hashtable

- Represents a collection of key/value pairs that are organized based on the hash code of the key.
- Implements **`IDictionary`**, **`ICollection`**, **`IEnumerable`**, **`ISerializable`**, **`IDeserializationCallback`**, **`ICloneable`**

```
using System;
using System.Collections;
class IDNameMap{
    public static void Main(){
        Hashtable nlist = new Hashtable();
       nlist.Add(11, "Trisha");
        nlist.Add(12, "Nisha");
        nlist.Add(13, "Nimisha");
        nlist.Add(14, "Varsha");
Console.WriteLine(" all keys");
foreach( int i in nlist.Keys )
Console.Write(i+ ", ");
Console.WriteLine("\nall values");
foreach( string s in nlist.Values )
Console.Write(s+ ",");
Console.WriteLine(nlist[13]);
nlist[13]="Disha";
foreach( DictionaryEntry de in nlist ){
Console.WriteLine("Key = {0}, Value =
{1}", de.Key, de.Value);
 }
//nlist.Add(14,"Varsha");

nlist.Remove(14);
if (!nlist.ContainsKey(14))      {
            Console.WriteLine("Key 14 is
not found.");
 }}}
```

Unhandled Exception: System.ArgumentException: Item has already been added. Keyin dictionary: '14'  Key being added: '14'

## Result of execution

```
 all keys
14, 13, 12, 11,
all values
Varsha,Nimisha,Nisha,Trisha,Nimisha
Key = 14, Value = Varsha
Key = 13, Value = Disha
Key = 12, Value = Nisha
Key = 11, Value = Trisha
Key 14 is not found.
```

## `SortedList` class

- Represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index.
- Implements **`IDictionary, ICollection, IEnumerable, ICloneable`**

## `Stack` Class

- Represents a simple last-in-first-out (LIFO) non-generic collection of objects
- Implements **`ICollection, IEnumerable, ICloneable`**
- Members  (apart from the interface methods)
  - **`public virtual Object Peek ():`** Returns the object at the top of the Stack without removing it.
  - **`virtual Object Pop ():`** Removes and returns the object at the top of the Stack.
  - **`public virtual void Push ( Object obj ):`** Inserts an object at the top of the Stack.

## Queue Class

- Represents a first-in, first-out collection of objects.
- Members (apart from the interface methods)
  - **`public virtual Object Dequeue ()`** Removes and returns the object at the beginning of the Queue.
  - **`public virtual void Enqueue ( Object obj ):`** Adds an object to the end of the Queue
  - **`public virtual Object Peek ()`**: Returns the object at the beginning of the Queue without removing it.

## Collection with generics

- All the collection interfaces have that ‘generic’ counter-part also.
- They are defined in **`System.Collections.Generic Namespace`**
- The generic collection classes are type-safe**`.`**
- Some of the important generic classes
  - **`List`**
  - **`Dictionary`**
  - **`LinkedList`**
  - **`Queue`**
  - **`Stack`**
  - **`SortedList`**

## Understanding type-safe collection

- `List<T>` is like `ArrayList` but it is type-safe.

```
using System;
using System.Collections;
using System.Collections.Generic;
public class TypeSafe{
 public static void Main()     {
 List<string> numbers = new
  List<string>();
   numbers.Add("One");
        numbers.Add("Two");
   numbers.Add(3);
}
}
```

TypeSafe.cs(10,15): error CS1503: Argument '1': cannot convert from 'int' to 'string'

Possible with `ArrayList`

## **`LinkedList`** class

- Represents a doubly linked list.
- Implements **`ICollection<T>, IEnumerable<T>, ICollection, IEnumerable, ISerializable, IDeserializationCallback`**
- Methods
  - **`AddAfter`**
  - **`AddBefore`**
  - **`AddFirst`**
  - **`AddLast`**
  - **`Find`**
  - **`FindLast`**
  - **`Remove`**
  - **`RemoveFirst`**
  - **`RemoveLast`**

```
using System;
using System.Text;
using System.Collections.Generic;
public class LLEx{
    public static void Main()    {
        string[] words =   { "sing",
"dance","play"};
        LinkedList<string> sentence = new
LinkedList<string>(words);
        Display(sentence);
        sentence.AddFirst("read");
        // Move the first node to be the last
node.
        LinkedListNode<string> mark1 =
sentence.First;
        sentence.RemoveFirst();
        sentence.AddLast(mark1);
        Display(sentence);
sentence.RemoveLast();
 sentence.AddLast("swim");
 Display(sentence);
    LinkedListNode<string> current =
sentence.Find("dance");
sentence.AddAfter(current, "roll");
sentence.AddBefore(current, "ring");
 Display(sentence);
    }
private static void
Display(LinkedList<string> words)  {
  foreach (string word in words)        {
  Console.Write(word + " ");          }
  Console.WriteLine();
    }
}
```

```
sing dance play
sing dance play read
sing dance play swim
sing ring dance roll play swim
```