Agenda

- What is Collection?
- Collections Framework
- Collections Hierarchy
- Collections Implementations

What is a Collection?

- A collection is a Ordered set of items that can be referred to as unit.
- The .NET framework provides specialized classes for data storage and retrieval.
- Using collections classes any type of data can be stored, retrieved, and manipulated as elements of collections.

Collections Framework

- Collections Framework is a unified architecture for managing collections.
- The main parts of a Collections Framework are:
  - Interfaces:
    - Core interfaces defining common functionality exhibited by collections.
  - Implementations:
    - Concrete classes of the core interfaces providing data structures.
  - Operations:
    - Methods that perform various operations on collections.

System.Collections: Interfaces

Core Interface	Description
`ICollection`	Defines size, enumerators and synchronization methods for all collections.
`IComparer`	Exposes a method that compares two objects.
`IDictionary`	Represents a collection of key-and-value pairs.
`IDictionaryEnumerator`	Enumerates the elements of a dictionary.
`IEnumerable`	Exposes the enumerator, which supports a simple iteration over a collection.
`IEnumerator`	Supports a simple iteration over a collection.
`IHashCodeProvider`	Supplies a hash code for an object, using a custom hash function.
`IList`	Represents a collection of objects that can be individually accessed by index.

System.Collections: Implementations

ICollection	IList	IDictionary
`BitArray`	`ArrayList`	`Hashtable`
`Stack`	`StringCollection`	`SortedList`
`Queue`		

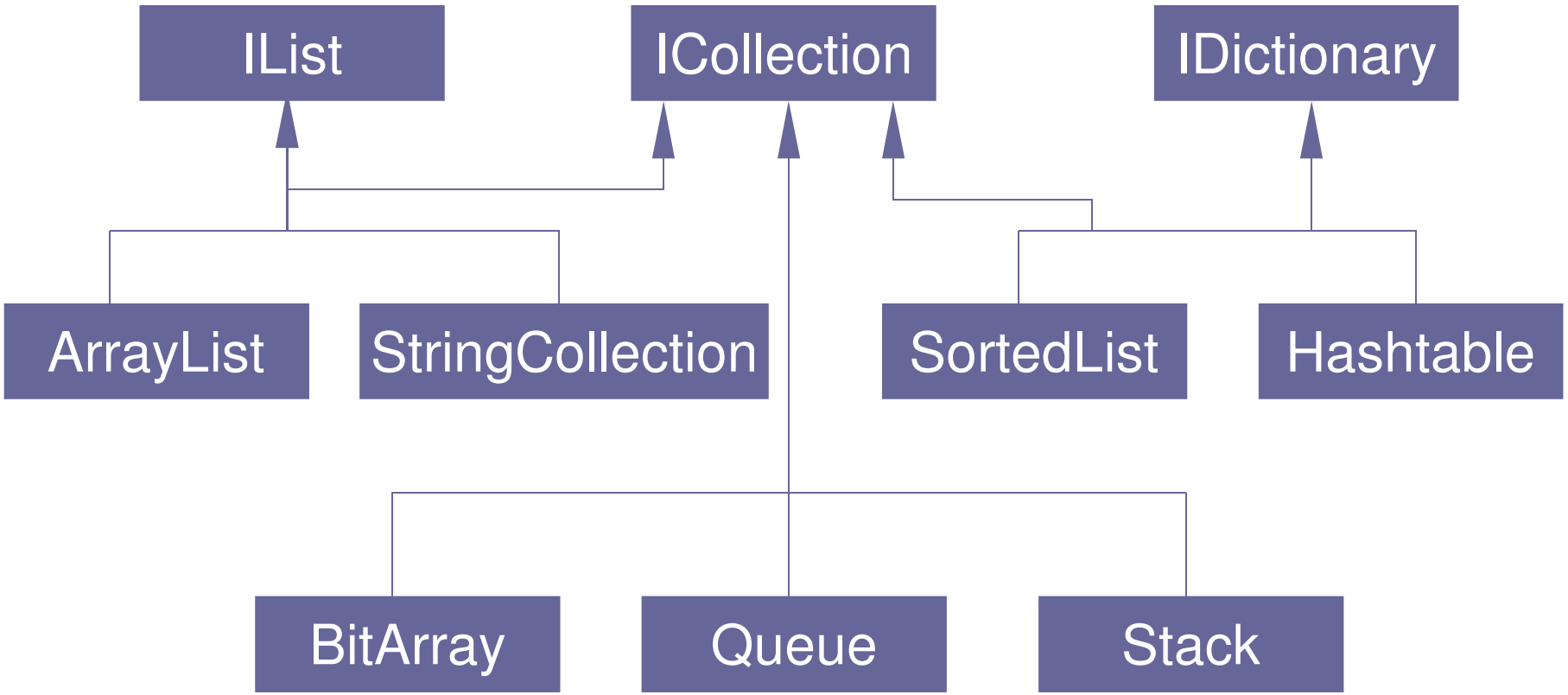
System.Collections: Operations

- Basic collection operations:
  - Check if an object exists in collection.
  - Retrieve an object from collection.
  - Add an object to collection.
  - Remove an object from collection.
  - Iterate collection and inspect each object.
- Each operation has a corresponding method implementation for each collection type.

Collections Characteristics

- Ordered:
  - Elements are stored and accessed in a specific order.
- Sorted:
  - Elements are stored and accessed in a sorted order.
- Indexed:
  - Elements can be accessed using an index.
- Unique:
  - Collection does not allow duplicates.

Collections Hierarchy



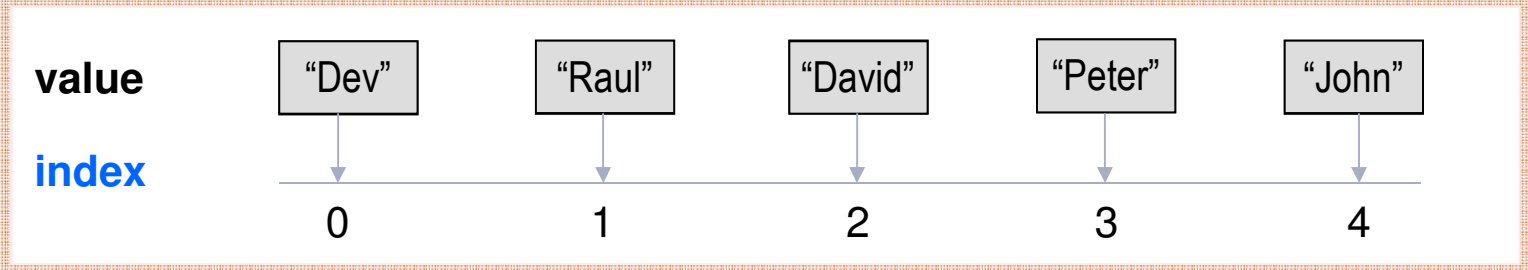
Collection Implementations

List: Lists of things (classes that implement IList)

Map: Things with a unique id (classes that implement IDictionary)

Custom Collections

List



A List cares about the index.

ArrayList (Implementation 1)	ArrayList (Implementation 2)	StringCollection

List Implementations: ArrayList (Implementation 1)

```
using System;
using System.Collections;

namespace TestConsoleApps
{
  public class SamplesArrayList
  {
    public static void Main()
    {
      // Creates and initializes a new ArrayList.
      ArrayList myAL = new ArrayList();
      myAL.Add("One");
      myAL.Add("Two");
      myAL.Add("!");

      //Display the values
      Console.Write("\t Value at 0 : {0}",
myAL[0].ToString());
      Console.WriteLine();
      Console.Write("\t Value at 1 : {0}", myAL[1]);
      Console.WriteLine();
      Console.Write("\t Value at 2 : {0}", myAL[2]);

      Console.Read();
    }
  }
}
```

Declaring the ArrayList

Adding elements to the ArrayList

Output

```
Value at 0 : One
Value at 1 : Two
Value at 2 : !
```

List Implementations: ArrayList (Implementation 2)

```
using System;
using System.Collections;

namespace TestConsoleApps
{
  public class SamplesArrayList
  {
    public static void Main()
    {
      // Creates and initializes a new ArrayList.
      ArrayList myAL = new ArrayList();
      myAL.Add("One");
      myAL.Add("Two");
      myAL.Add("!");

      //Displays the properties and values of the ArrayList.
      Console.WriteLine( "\tCount:    {0}", myAL.Count );
      Console.WriteLine( "\tCapacity: {0}", myAL.Capacity );
      Console.Write( "\tValues:" );
      PrintValues( myAL );
      Console.Read();
    }

    public static void PrintValues( IEnumerable myList )
    {
      System.Collections.IEnumerator myEnumerator = myList.GetEnumerator();
      while ( myEnumerator.MoveNext())
      {
        Console.Write( "\t{0}", myEnumerator.Current );
      }
      Console.WriteLine();
    }
  }
}
```

Declaring an ArrayList

Output

```
Count: 3
Capacity: 4
Values: One Two !
```

Using Enumerator to move through the ArrayList

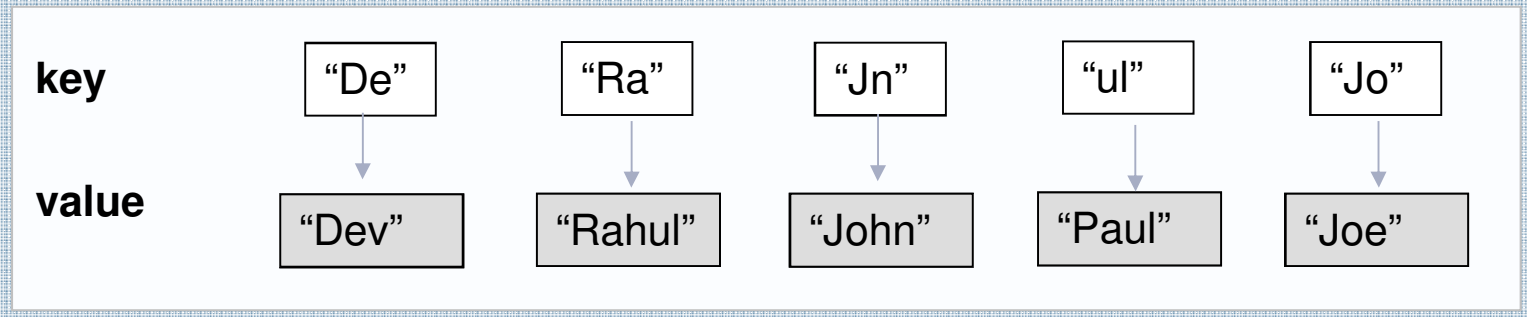
List Implementations: StringCollection

```
using System;
using System.Collections;
using System.Collections.Specialized;

namespace TestConsoleApps
{
  /// <summary>
  /// Summary description for SamplesStringCollection.
  /// </summary>
  public class SamplesStringCollection
  {
    public static void Main()
    {
                    // Creates and initializes a new StringCollection.
                    StringCollection myCol = new StringCollection();

                    // Adds a range of elements from an array to the end of the StringCollection.
                    String[] myArr = new String[] {"One", "Two", "Three", "Four"};
                    myCol.AddRange(myArr);

                    Console.WriteLine("After adding elements:");
                    PrintValues(myCol);

                    // Adds one element to the end of the StringCollection and inserts another at index 3.
                    myCol.Add("* Five");
                    myCol.Insert(3, "* Three");

                    Console.WriteLine( "After adding and inserting:" );
                    PrintValues(myCol);
    }

    public static void PrintValues(IEnumerable myCol)
    {
                    System.Collections.IEnumerator myEnumerator = myCol.GetEnumerator();
                    while (myEnumerator.MoveNext())
                    {
                      Console.WriteLine("   {0}", myEnumerator.Current);
                    }
                    Console.WriteLine();
    }
  }
}
```

Output

```
After adding elements:
   One
   Two
   Three
   Four

After adding and inserting:
   One
   Two
   Three
   * Three
   Four
   * Five
```

Map



A Map cares about unique identifiers.

SortedList **Hashtable**

Map Implementations: SortedList

```
using System;
using System.Collections;

namespace TestConsoleApps
{
  /// <summary>
  /// Summary description for SampleSortedList.
  /// </summary>
  public class SamplesSortedList
  {
    public static void Main()
    {
                // Creates and initializes a new SortedList.
                SortedList mySL = new SortedList();
                mySL.Add("First", "One");
                mySL.Add("Second", "Two");
                mySL.Add("Third", "! ");

                 // Displays the properties and values of the SortedList.
                 Console.WriteLine("  Keys and Values:");

                 for (int i = 0; i < mySL.Count; i++)
                 {
                   Console.WriteLine("\t{0}:\t{1}", mySL.GetKey(i), mySL.GetByIndex(i));
                 }

                //Get a value from a SortedList using key info
                Console.WriteLine("Value for key 'Third' is " + mySL["Third"].ToString());
                Console.Read();
    }
  }
}
```

Output

```
Keys and Values:
    First:   One
    Second: Two
    Third:   !
Value for key ‘Third’ is !
```

Map Implementations: Hashtable

```
using System;
using System.Collections;

namespace TestConsoleApps
{
  /// <summary>
  /// Summary description for SampleHashtable.
  /// </summary>
  public class SamplesHashtable
  {
    public static void Main()
    {
                        // Creates and initializes a new Hashtable.
          Hashtable myHT = new Hashtable();
          myHT.Add("First", "C#");
          myHT.Add("Second", "VB");
          myHT.Add("Third", "C++");

          // Displays the values of the Hashtable.
          Console.WriteLine(" Keys and Values:");
          PrintKeysAndValues(myHT);

          //getting a value from a hashtable using key info
          Console.WriteLine("Value for key 'Second' is " + myHT["Second"].ToString());

          Console.Read();
    }

    public static void PrintKeysAndValues(Hashtable myList)
    {
      IDictionaryEnumerator myEnumerator = myList.GetEnumerator();
      while (myEnumerator.MoveNext())
       {
          Console.WriteLine("\t{0}:\t{1}", myEnumerator.Key, myEnumerator.Value);
       }
       Console.WriteLine();
    }
  }
}
```

Output

```
Keys and Values:
   First:  C#
   Second: VB
   Third:  C++

Value for key 'Second' is VB
```

Accessing a value using key from a hashtable.

Custom Collections

Custom Collections can be implemented by either of the following methods:

- Extending from CollectionBase – Use this to implement a custom collection which is modifiable. It provides methods required to implement a strongly typed custom collection.
- Extending from ReadOnlyCollectionBase – Members of this base class are protected and are intended to be used through a derived class only. The derived class must ensure that its own users cannot modify the underlying collection.

Custom Collection Implementation

```
Class CustCollections:CollectionBase
    {
            public void Add(object val)
            {
                    List.Add(val);
            }
            public int Insert(int inx,object val)
            {
                    return List.Insert(val,inx);
            }
            public void Remove(object val)
            {
                    List.Remove(val);
            }

    }
```