Collections

Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types in C#.

## Various Collection Classes and Their Usage

The following are the various commonly used classes of the **System.Collection** namespace. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Class** | **Description and Useage** |
| [**ArrayList**](http://www.tutorialspoint.com/csharp/csharp_arraylist.htm) | It represents ordered collection of an object that can be **indexed**individually.  It is basically an alternative to an array. However, unlike array you can add and remove items from a list at a specified position using an**index** and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list. |
| [**Hashtable**](http://www.tutorialspoint.com/csharp/csharp_hashtable.htm) | It uses a **key** to access the elements in the collection.  A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a **key/value** pair. The key is used to access the items in the collection. |
| [**SortedList**](http://www.tutorialspoint.com/csharp/csharp_sortedlist.htm) | It uses a **key** as well as an **index** to access the items in a list.  A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key , it is a Hashtable. The collection of items is always sorted by the key value. |
| [**Stack**](http://www.tutorialspoint.com/csharp/csharp_stack.htm) | It represents a **last-in, first out** collection of object.  It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called **pushing** the item and when you remove it, it is called **popping** the item. |
| [**Queue**](http://www.tutorialspoint.com/csharp/csharp_queue.htm) | It represents a **first-in, first out** collection of object.  It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called **enqueue** and when you remove an item, it is called **deque**. |
| [**BitArray**](http://www.tutorialspoint.com/csharp/csharp_bitarray.htm) | It represents an array of the **binary representation** using the values 1 and 0.  It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an **integer index**, which starts from zero. |

# ArrayList Class

It represents an ordered collection of an object that can be indexed individually. It is basically an alternative to an array. However, unlike array you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list.

## Methods and Properties of ArrayList Class

The following table lists some of the commonly used **properties** of the **ArrayList** class:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Capacity | Gets or sets the number of elements that the ArrayList can contain. |
| Count | Gets the number of elements actually contained in the ArrayList. |
| IsFixedSize | Gets a value indicating whether the ArrayList has a fixed size. |
| IsReadOnly | Gets a value indicating whether the ArrayList is read-only. |
| Item | Gets or sets the element at the specified index. |

The following table lists some of the commonly used **methods** of the **ArrayList** class:

|  |  |
| --- | --- |
| **Sr.No.** | **Methods** |
| 1 | **public virtual int Add(object value);**  Adds an object to the end of the ArrayList. |
| 2 | **public virtual void AddRange(ICollection c);**  Adds the elements of an ICollection to the end of the ArrayList. |
| 3 | **public virtual void Clear();**  Removes all elements from the ArrayList. |
| 4 | **public virtual bool Contains(object item);**  Determines whether an element is in the ArrayList. |
| 5 | **public virtual ArrayList GetRange(int index, int count);**  Returns an ArrayList which represents a subset of the elements in the source ArrayList. |
| 6 | **public virtual int IndexOf(object);**  Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it. |
| 7 | **public virtual void Insert(int index, object value);**  Inserts an element into the ArrayList at the specified index. |
| 8 | **public virtual void InsertRange(int index, ICollection c);**  Inserts the elements of a collection into the ArrayList at the specified index. |
| 9 | **public virtual void Remove(object obj);**  Removes the first occurrence of a specific object from the ArrayList. |
| 10 | **public virtual void RemoveAt(int index);**  Removes the element at the specified index of the ArrayList. |
| 11 | **public virtual void RemoveRange(int index, int count);**  Removes a range of elements from the ArrayList. |
| 12 | **public virtual void Reverse();**  Reverses the order of the elements in the ArrayList. |
| 13 | **public virtual void SetRange(int index, ICollection c);**  Copies the elements of a collection over a range of elements in the ArrayList. |
| 14 | **public virtual void Sort();**  Sorts the elements in the ArrayList. |
| 15 | **public virtual void TrimToSize();**  Sets the capacity to the actual number of elements in the ArrayList. |

## Example

The following example demonstrates the concept:

using System;

using System.Collections;

namespace CollectionApplication

{

class Program

{

static void Main(string[] args)

{

ArrayList al = new ArrayList();

Console.WriteLine("Adding some numbers:");

al.Add(45);

al.Add(78);

al.Add(33);

al.Add(56);

al.Add(12);

al.Add(23);

al.Add(9);

Console.WriteLine("Capacity: {0} ", al.Capacity);

Console.WriteLine("Count: {0}", al.Count);

Console.Write("Content: ");

foreach (int i in al)

{

Console.Write(i + " ");

}

Console.WriteLine();

Console.Write("Sorted Content: ");

al.Sort();

foreach (int i in al)

{

Console.Write(i + " ");

}

Console.WriteLine();

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

Adding some numbers:

Capacity: 8

Count: 7

Content: 45 78 33 56 12 23 9

Content: 9 12 23 33 45 56 78

# Hashtable Class

The Hashtable class represents a collection of **key-and-value pairs** that are organized based on the hash code of the key. It uses the key to access the elements in the collection.

A hash table is used when you need to access elements by using **key**, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

## Methods and Properties of the Hashtable Class

The following table lists some of the commonly used **properties** of the **Hashtable** class:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Count | Gets the number of key-and-value pairs contained in the Hashtable. |
| IsFixedSize | Gets a value indicating whether the Hashtable has a fixed size. |
| IsReadOnly | Gets a value indicating whether the Hashtable is read-only. |
| Item | Gets or sets the value associated with the specified key. |
| Keys | Gets an ICollection containing the keys in the Hashtable. |
| Values | Gets an ICollection containing the values in the Hashtable. |

The following table lists some of the commonly used **methods** of the **Hashtable** class:

|  |  |
| --- | --- |
| **Sr.No.** | **Method** |
| 1 | **public virtual void Add(object key, object value);**  Adds an element with the specified key and value into the Hashtable. |
| 2 | **public virtual void Clear();**  Removes all elements from the Hashtable. |
| 3 | **public virtual bool ContainsKey(object key);**  Determines whether the Hashtable contains a specific key. |
| 4 | **public virtual bool ContainsValue(object value);**  Determines whether the Hashtable contains a specific value. |
| 5 | **public virtual void Remove(object key);**  Removes the element with the specified key from the Hashtable. |

## Example

The following example demonstrates the concept:

using System;

using System.Collections;

namespace CollectionsApplication

{

class Program

{

static void Main(string[] args)

{

Hashtable ht = new Hashtable();

ht.Add("001", "Zara Ali");

ht.Add("002", "Abida Rehman");

ht.Add("003", "Joe Holzner");

ht.Add("004", "Mausam Benazir Nur");

ht.Add("005", "M. Amlan");

ht.Add("006", "M. Arif");

ht.Add("007", "Ritesh Saikia");

if (ht.ContainsValue("Nuha Ali"))

{

Console.WriteLine("This student name is already in the list");

}

else

{

ht.Add("008", "Nuha Ali");

}

// Get a collection of the keys.

ICollection key = ht.Keys;

foreach (string k in key)

{

Console.WriteLine(k + ": " + ht[k]);

}

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

001: Zara Ali

002: Abida Rehman

003: Joe Holzner

004: Mausam Benazir Nur

005: M. Amlan

006: M. Arif

007: Ritesh Saikia

008: Nuha Ali