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 −

| Sr.No. | Property & Description |
|---|---|
| 1 | **Count**<br>Gets the number of key-and-value pairs contained in the Hashtable. |
| 2 | **IsFixedSize**<br>Gets a value indicating whether the Hashtable has a fixed size. |
| 3 | **IsReadOnly**<br>Gets a value indicating whether the Hashtable is read-only. |
| 4 | **Item**<br>Gets or sets the value associated with the specified key. |
| 5 | **Keys**<br>Gets an ICollection containing the keys in the Hashtable. |
| 6 | **Values**<br>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 & Description |
|---|---|
| 1 | **public virtual void Add(object key, object value);**<br>Adds an element with the specified key and value into the Hashtable. |
| 2 | **public virtual void Clear();**<br>Removes all elements from the Hashtable. |
| 3 | **public virtual bool ContainsKey(object key);**<br>Determines whether the Hashtable contains a specific key. |

**public virtual bool ContainsValue(object value);**

4

Determines whether the Hashtable contains a specific value.

**public virtual void Remove(object key);**

5

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", "maitri");
         ht.Add("002", "dikshita");
         ht.Add("003", "dhruvita");
         ht.Add("004", "shruchita");
         ht.Add("005", "niketa");
         ht.Add("006", "bansri");
         ht.Add("007", "dipika");

         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();
      }
   }
}
```

# 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.

| Sr.No. | Class & Description and Useage |
|---|---|
| 1 | ArrayList<br><br>It represents ordered collection of an object that can be **indexed** individually.<br><br>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. |
| 2 | Hashtable<br><br>It uses a **key** to access the elements in the collection.<br><br>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. |
| 3 | SortedList<br><br>It uses a **key** as well as an **index** to access the items in a list.<br><br>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. |
| 4 | Stack<br><br>It represents a **last-in, first out** collection of object.<br><br>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. |
| 5 | Queue<br><br>It represents a **first-in, first out** collection of object.<br><br>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**. |
| 6 | BitArray<br><br>It represents an array of the **binary representation** using the values 1 and 0.<br><br>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. |

**=======➔1 ARRAY LIST**

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 −

| Sr.No. | Property & Description |
|---|---|
| 1 | **Capacity**<br>Gets or sets the number of elements that the ArrayList can contain. |
| 2 | **Count**<br>Gets the number of elements actually contained in the ArrayList. |
| 3 | **IsFixedSize**<br>Gets a value indicating whether the ArrayList has a fixed size. |
| 4 | **IsReadOnly**<br>Gets a value indicating whether the ArrayList is read-only. |
| 5 | **Item**<br>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. | Method & Description |
|---|---|
| 1 | **public virtual int Add(object value);**<br>Adds an object to the end of the ArrayList. |
| 2 | **public virtual void AddRange(ICollection c);**<br>Adds the elements of an ICollection to the end of the ArrayList. |
| 3 | **public virtual void Clear();**<br>Removes all elements from the ArrayList. |
| 4 | **public virtual bool Contains(object item);**<br>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.

**public virtual int IndexOf(object);**

6

Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.

**public virtual void Insert(int index, object value);**

7

Inserts an element into the ArrayList at the specified index.

**public virtual void InsertRange(int index, ICollection c);**

8

Inserts the elements of a collection into the ArrayList at the specified index.

**public virtual void Remove(object obj);**

9

Removes the first occurrence of a specific object from the ArrayList.

**public virtual void RemoveAt(int index);**

10

Removes the element at the specified index of the ArrayList.

**public virtual void RemoveRange(int index, int count);**

11

Removes a range of elements from the ArrayList.

**public virtual void Reverse();**

12

Reverses the order of the elements in the ArrayList.

**public virtual void SetRange(int index, ICollection c);**

13

Copies the elements of a collection over a range of elements in the ArrayList.

**public virtual void Sort();**

14

Sorts the elements in the ArrayList.

**public virtual void TrimToSize();**

15

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
```

The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.

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.

# Methods and Properties of the SortedList Class

The following table lists some of the commonly used **properties** of the **SortedList** class −

| Sr.No. | Property & Description |
|---|---|
| 1 | **Capacity**<br>Gets or sets the capacity of the SortedList. |
| 2 | **Count**<br>Gets the number of elements contained in the SortedList. |
| 3 | **IsFixedSize**<br>Gets a value indicating whether the SortedList has a fixed size. |
| 4 | **IsReadOnly**<br>Gets a value indicating whether the SortedList is read-only. |
| 5 | **Item**<br>Gets and sets the value associated with a specific key in the SortedList. |
| 6 | **Keys**<br>Gets the keys in the SortedList. |
| 7 | **Values**<br>Gets the values in the SortedList. |

The following table lists some of the commonly used **methods** of the **SortedList** class −

| Sr.No. | Method & Description |
|---|---|
| 1 | **public virtual void Add(object key, object value);**<br>Adds an element with the specified key and value into the SortedList. |
| 2 | **public virtual void Clear();**<br>Removes all elements from the SortedList. |
| 3 | **public virtual bool ContainsKey(object key);**<br>Determines whether the SortedList contains a specific key. |
| 4 | **public virtual bool ContainsValue(object value);**<br>Determines whether the SortedList contains a specific value. |
| 5 | **public virtual object GetByIndex(int index);**<br>Gets the value at the specified index of the SortedList. |
| 6 | **public virtual object GetKey(int index);** |

Gets the key at the specified index of the SortedList.

**public virtual IList GetKeyList();**

7

Gets the keys in the SortedList.

**public virtual IList GetValueList();**

8

Gets the values in the SortedList.

**public virtual int IndexOfKey(object key);**

9

Returns the zero-based index of the specified key in the SortedList.

**public virtual int IndexOfValue(object value);**

10

Returns the zero-based index of the first occurrence of the specified value in the SortedList.

**public virtual void Remove(object key);**

11

Removes the element with the specified key from the SortedList.

**public virtual void RemoveAt(int index);**

12

Removes the element at the specified index of SortedList.

**public virtual void TrimToSize();**

13

Sets the capacity to the actual number of elements in the SortedList.

# Example

The following example demonstrates the concept −

```
using System;
using System.Collections;

namespace CollectionsApplication {

   class Program {

      static void Main(string[] args) {
         SortedList sl = new SortedList();

         sl.Add("001", "kohli");
         sl.Add("002", "dhoni");
         sl.Add("003", "bumrah");
         sl.Add("004", "kartik");
         sl.Add("005", "sharma");
         sl.Add("006", "patel");
         sl.Add("007", "balaji");

         if (sl.ContainsValue("kohli")) {
            Console.WriteLine("This student name is already in the list");
```

```
         } else {
            sl.Add("008", "parthiv");
         }

         // get a collection of the keys.
         ICollection key = sl.Keys;

         foreach (string k in key) {
            Console.WriteLine(k + ": " + sl[k]);
         }
      }
   }
}
```

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.

# Methods and Properties of the Stack Class

The following table lists some commonly used **properties** of the **Stack** class −

| Sr.No. | Property & Description |
|---|---|
| 1 | **Count**<br>Gets the number of elements contained in the Stack. |

The following table lists some of the commonly used **methods** of the **Stack** class −

| Sr.No. | Method & Description |
|---|---|
| 1 | **public virtual void Clear();**<br>Removes all elements from the Stack. |
| 2 | **public virtual bool Contains(object obj);**<br>Determines whether an element is in the Stack. |
| 3 | **public virtual object Peek();**<br>Returns the object at the top of the Stack without removing it. |
| 4 | **public virtual object Pop();**<br>Removes and returns the object at the top of the Stack. |
| 5 | **public virtual void Push(object obj);**<br>Inserts an object at the top of the Stack. |

**public virtual object[] ToArray();**

6

Copies the Stack to a new array.

# Example

The following example demonstrates use of Stack −

```
using System;
using System.Collections;

namespace CollectionsApplication {

   class Program {

      static void Main(string[] args) {
         Stack st = new Stack();

         st.Push('A');
         st.Push('M');
         st.Push('G');
         st.Push('W');

         Console.WriteLine("Current stack: ");
         foreach (char c in st) {
            Console.Write(c + " ");
         }
         Console.WriteLine();

         st.Push('V');
         st.Push('H');
         Console.WriteLine("The next poppable value in stack: {0}",
st.Peek());
         Console.WriteLine("Current stack: ");
         foreach (char c in st) {
            Console.Write(c + " ");
         }

         Console.WriteLine();

         Console.WriteLine("Removing values ");
         st.Pop();
         st.Pop();
         st.Pop();

         Console.WriteLine("Current stack: ");
         foreach (char c in st) {
            Console.Write(c + " ");
         }
      }
   }
}
```

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**.

# Methods and Properties of the Queue Class

The following table lists some of the commonly used **properties** of the **Queue** class −

| Sr.No. | Property & Description |
|---|---|
| 1 | **Count**<br>Gets the number of elements contained in the Queue. |

The following table lists some of the commonly used **methods** of the **Queue** class −

| Sr.No. | Method & Description |
|---|---|
| 1 | **public virtual void Clear();**<br>Removes all elements from the Queue. |
| 2 | **public virtual bool Contains(object obj);**<br>Determines whether an element is in the Queue. |
| 3 | **public virtual object Dequeue();**<br>Removes and returns the object at the beginning of the Queue. |
| 4 | **public virtual void Enqueue(object obj);**<br>Adds an object to the end of the Queue. |
| 5 | **public virtual object[] ToArray();**<br>Copies the Queue to a new array. |
| 6 | **public virtual void TrimToSize();**<br>Sets the capacity to the actual number of elements in the Queue. |

# Example

The following example demonstrates use of Stack −

```
using System;
using System.Collections;

namespace CollectionsApplication {
```

```
   class Program {

      static void Main(string[] args) {
         Queue q = new Queue();

         q.Enqueue('A');
         q.Enqueue('M');
         q.Enqueue('G');
         q.Enqueue('W');

         Console.WriteLine("Current queue: ");
         foreach (char c in q) Console.Write(c + " ");

         Console.WriteLine();
         q.Enqueue('V');
         q.Enqueue('H');
         Console.WriteLine("Current queue: ");
         foreach (char c in q) Console.Write(c + " ");

         Console.WriteLine();
         Console.WriteLine("Removing some values ");
         char ch = (char)q.Dequeue();
         Console.WriteLine("The removed value: {0}", ch);
         ch = (char)q.Dequeue();
         Console.WriteLine("The removed value: {0}", ch);

         Console.ReadKey();
      }
   }
}
```