

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 Addobjectvalue; Adds an object to the end of the ArrayList.
2	public virtual void AddRangeICollectionnc; 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 Containsobjectitem; Determines whether an element is in the ArrayList.
5	public virtual ArrayList GetRangeintindex, intcount; Returns an ArrayList which represents a subset of the elements in the source ArrayList.
6	public virtual int IndexOfobject; 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 Insertintindex, objectvalue; Inserts an element into the ArrayList at the specified index.

- 8 **public virtual void InsertRange***intindex, ICollectionc;*
Inserts the elements of a collection into the ArrayList at the specified index.
- 9 **public virtual void Remove***objectobj;*
Removes the first occurrence of a specific object from the ArrayList.
- 10 **public virtual void RemoveAt***intindex;*
Removes the element at the specified index of the ArrayList.
- 11 **public virtual void RemoveRange***intindex, intcount;*
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***intindex, ICollectionc;*
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

```

Loading [MathJax]/jax/output/HTML-CSS/jax.js