1. What is Generic List<T> Collection in C#?

The Generic List<T> in C# is a Collection Class that belongs to System.Collections.Generic namespace. This Generic List<T> Collection Class represents a strongly typed list of objects which can be accessed by using the integer index which is starting from 0. It also provides lots of methods that can be used for searching, sorting, and manipulating the list of items.

We can create a collection of any data type by using the Generic List<T> Collection Class in C#. For example, if you want then you can create a list of strings, a list of integers, and a list of doubles, and it is also possible to create a list of user-defined complex types such as a list of customers, a list of products, a list of students, etc. The most important point that you need to keep in mind is the size of the collection grows automatically when we add items to the collection.

1. How to Create a Generic List Collection in C#?

If you want to add elements to the generic list collection then you need to use the following Add() and AddRange() methods. If you notice the Add and Add method expects the data of type T. And here, T is nothing but the parameter type that you need to specify while creating the instance of the Generic List Class.

**Add(T item):** The Add(T item) method is used to add an element to the end of the Generic List. Here, the parameter item specifies the object to be added to the end of the Generic List. The value can be null for a reference type.

**AddRange(IEnumerable<T> collection):** The AddRange(IEnumerable<T> collection) method is used to add the Elements of the specified collection to the end of the Generic List. The parameter collection specifies the collection whose elements should be added to the end of the Generic List. The collection itself cannot be null, but it can contain elements that are null if type T is a reference type.

**List<string> countries = new List<string>();**

**countries.Add(“India”);**

**countries.Add(“Srilanka”);**

* Adding Elements using the AddRange Method of the List Class. In this case, we have two collections and we need to add one collection into another collection as follows. The following is our first collection.

**List<string> countries = new List<string>();**

**countries.Add(“India”);**

**countries.Add(“Srilanka”);**

The following is our second collection.

**List<string> newCountries = new List<string>();**

**newCountries.Add(“USA”);**

**newCountries.Add(“UK”);**

Now, we want to add our second collection at the end of the first collection. To do so, we need to use the AddRange method as follows:

**countries.AddRange(newCountries);**

Even, it is also possible in C# to create a List<T> collection using Collection Initializer Syntax as follows:

**List<string> countries = new List<string>**

**{**

**“India”,**

**“Srilanka”,**

**“USA”**

**};**

So, these are the different ways to add elements to the end of a generic list collection.

1. How to access a Generic List<T> Collection in C#?

* Using Index to Access Generic List<T> Collection in C#:

The List<T> Class Implements the IList<T> interface. So, we can access the individual elements of the List Collection in C# by using the integral index.

**countries[0]; //First Element**

**countries[1]; //Second Element**

**countries[2]; //Third Element**

* Using For-Each Loop to Access Generic List<T> Collection in C#:

You can also use a for-each loop to access the elements of a Generic List<T> collection in C# as follows. Here, the loop variable must be of the same type as the type of elements you stored in the collection.

**foreach (string item in countries)**

**{**

**Console.WriteLine(item);**

**}**

Even you can also use the var keyword to define the loop variable and based on the values it will automatically decide the data type as follows.

**foreach (var item in countries)**

**{**

**Console.WriteLine(item);**

**}**

* Using For Loop to Access Generic List<T> Collection in C#:

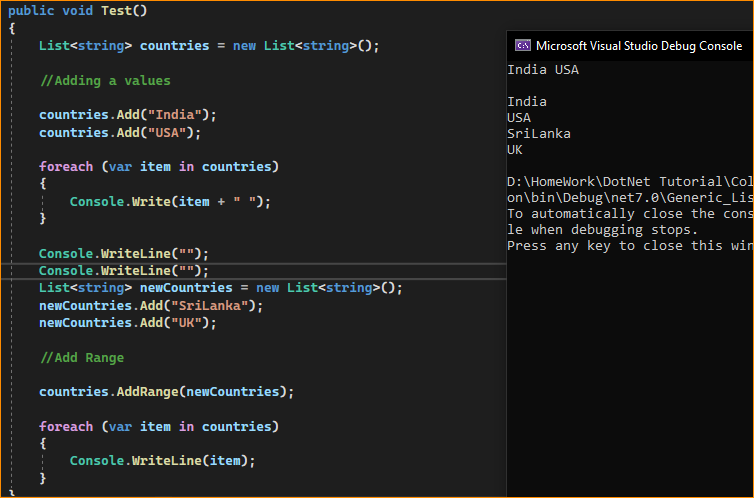
You can also access the Generic List<T> collection in C# using a for loop as follows. Here, we need to get the count of the list collection by using the Count properties of the List class and then start the loop from 0 and fetch the element of the list collection using the Index position. Here, the loop variable is going to be of integer which is going to point to the index position of the collection.

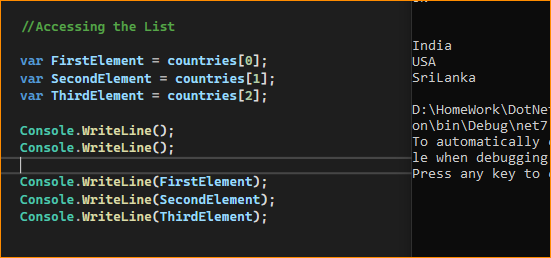
**for (int i = 0; i < countries.Count; i++)**

**{**

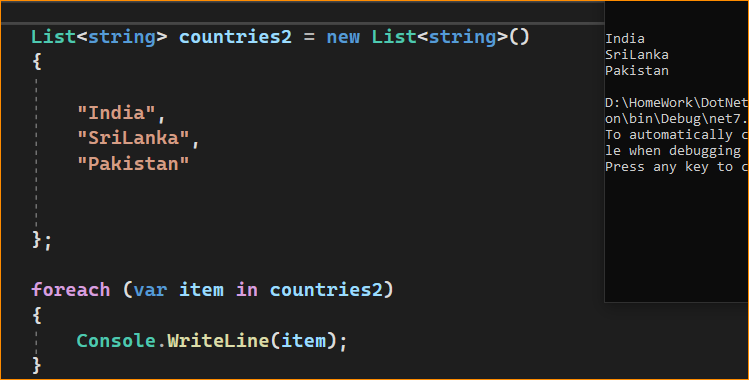
**var element = countries[i];**

**}**





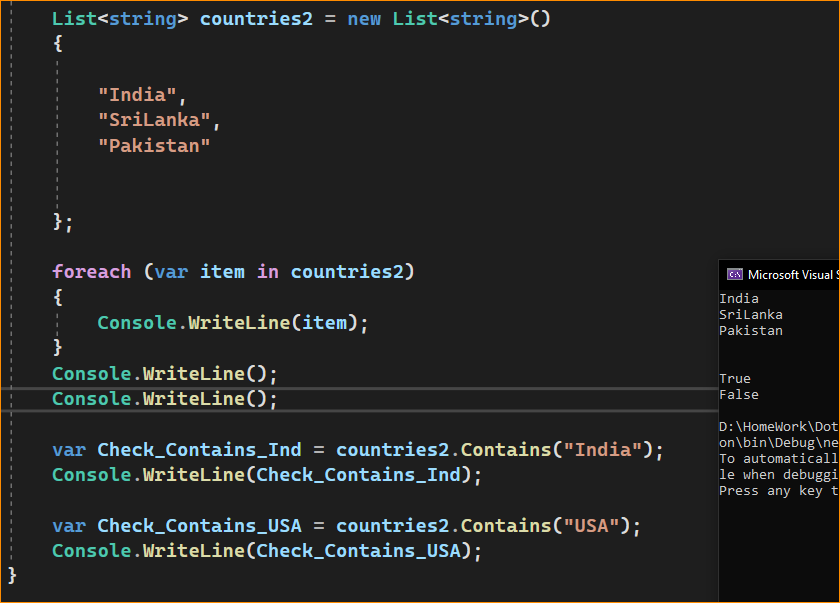
* Creating a Generic List of string type and adding elements using collection initializer



* How to Check the Availability of an Element in a List Collection in C#?

If you want to check whether an element exists or not in the Generic List Collection, then you need to use the following Contains method of the Generic List Collection Class in C#.

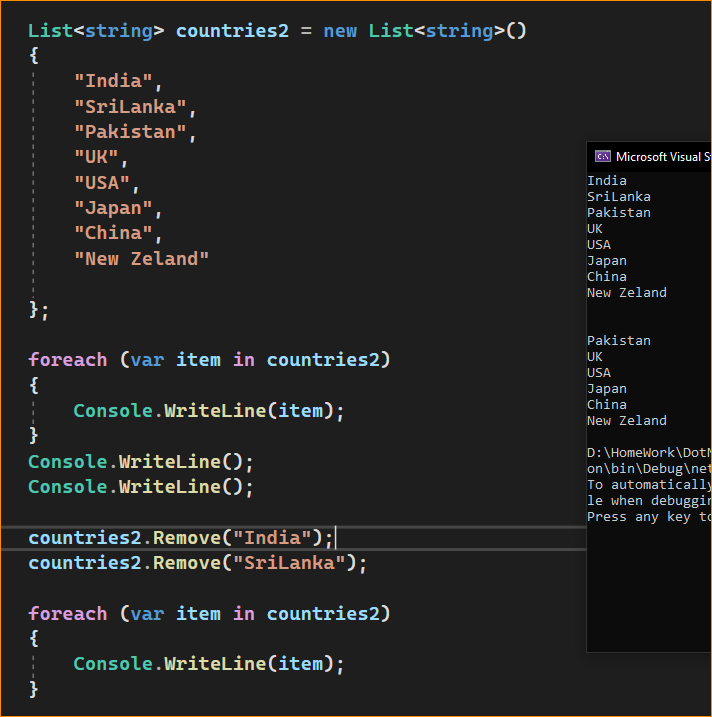
**Contains(T item):** The Contains(T item) method of the Generic List Collection Class is used to check if the given item is present in the List or not. The parameter item specifies the object to locate in the Generic List. The value can be null for reference types. It returns true if the item is found in the Generic List; otherwise, false.



1. How to Remove Elements from a Generic List Collection in C#?

If you want to remove elements from the list, then you can use the following methods of the List collection class.

**Remove(T item):** This method is used to remove the first occurrence of a specific object from the Generic List. Here, the parameter item specifies the object to remove from the Generic List. It returns true if the item is successfully removed; otherwise, false. This method also returns false if the item was not found in the Generic List.

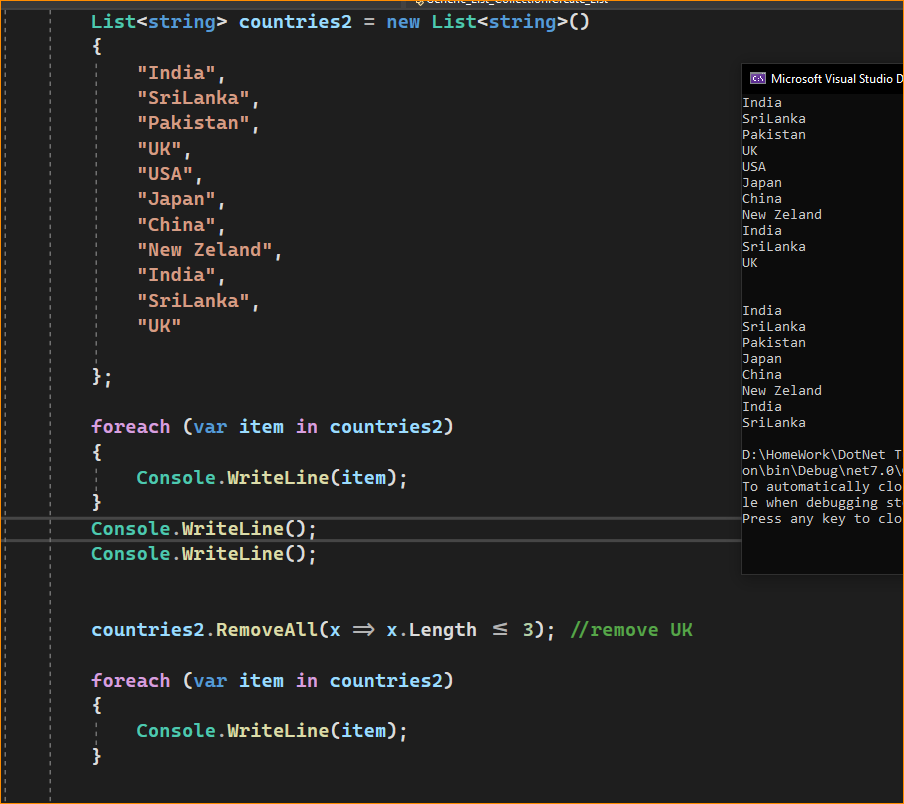


In here we removed India and Sri Lanka

**RemoveAll(Predicate<T> match):** This method is used to remove all the elements that match the conditions defined by the specified predicate. Here, the parameter match specifies the predicate delegate that defines the conditions of the elements to remove. It returns the number of elements removed from the Generic List. If the parameter match is null, then it will throw ArgumentNullException.

Using RemoveAll method to Remove Elements from the List. Here, we are removing element whose length is less than 3 or equal 3 (UK= length 2 < 3 so removed, USE length =3 so removed)

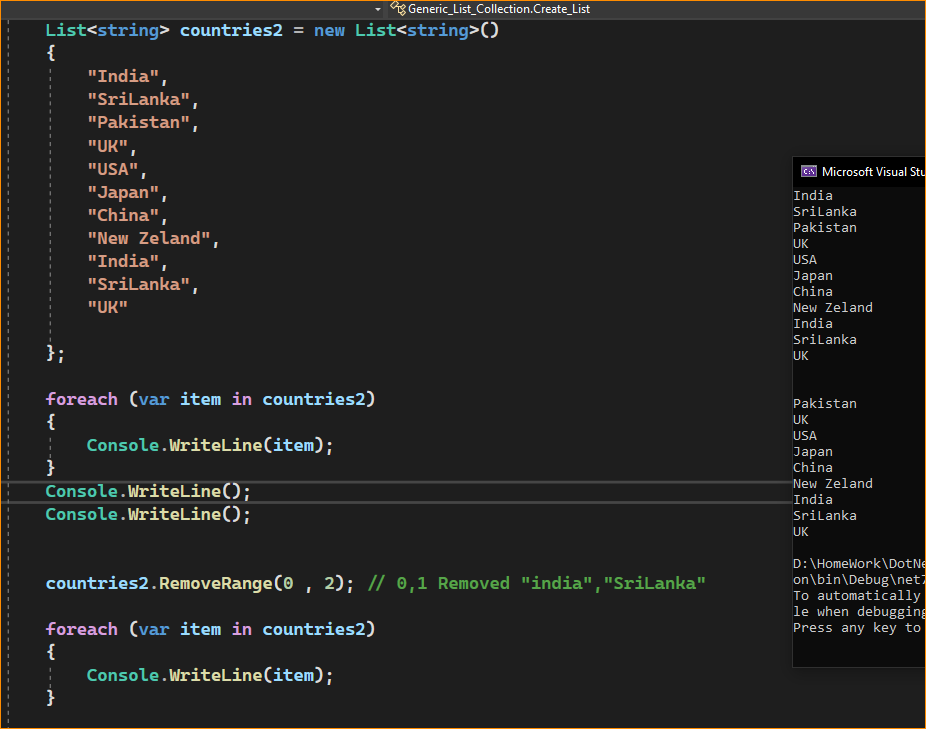
**countries.RemoveAll(x => x.Length <= 3);**



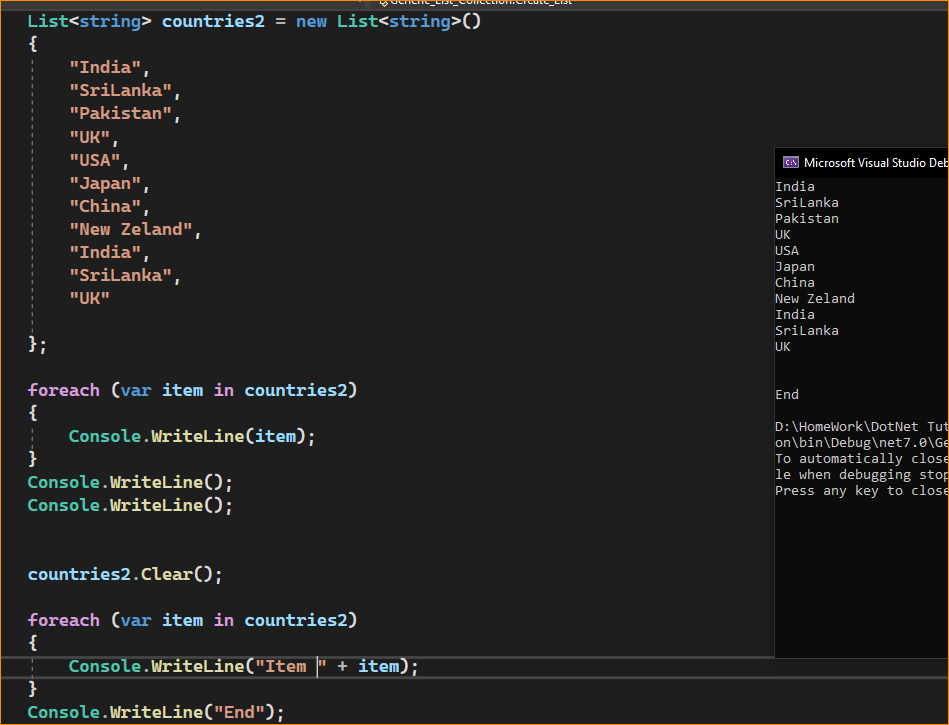
**RemoveAt(int index):** This method is used to remove the element at the specified index of the Generic List. Here, the parameter index is the zero-based index of the element to remove. If the index is less than 0 or the index is equal to or greater than Generic List Count, then it will throw ArgumentOutOfRangeException.

countries2.RemoveAt(0); // Remove 1 st element "India"

**RemoveRange(int index, int count):** This method is used to remove a range of elements from the Generic List. Here, the parameter index is the zero-based starting index of the range of elements to remove and the parameter count is the number of elements to remove. If the index is less than 0 or the count is less than 0, then it will throw ArgumentOutOfRangeException. If the index and count do not denote a valid range of elements in the Generic List, then it will throw ArgumentException.



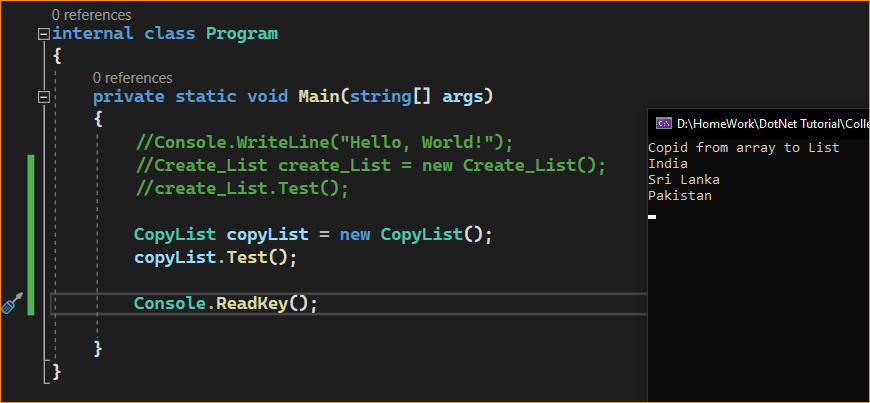
**Clear():** This method is used to remove all elements from the Generic List.



1. How to Copy an Array to a List in C#?

To Copy an Array to a List we need to use the following overloaded constructor of the List class in C#. As you

public List(IEnumerable<T> collection): This constructor is used to initialize a new instance of the Generic List class that contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied. The parameter collection specifies the collection whose elements are copied to the new list.

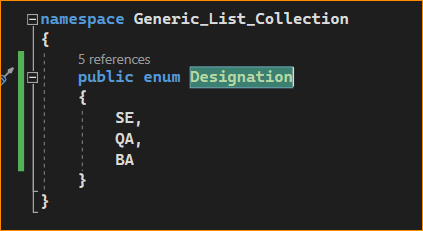


As of now, we are working with Generic List Collection with built-in primitive data types. Now, let us proceed further and try to understand how to work with Complex Data Types like Student, Employee, Product, etc.

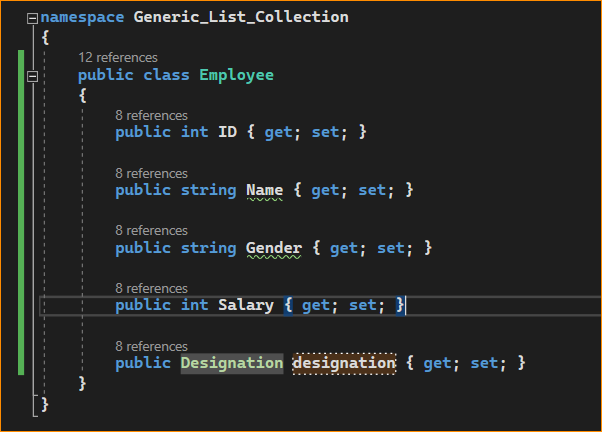
1. Generic List Collection with Complex Type in C#:

Let us see an example of the List Collection class with Complex Types in C#. Please have a look at the below example code. As you can see in the below code, we have created a class called Employee. Then we create a few employee objects and then we create a Generic List Collection of type Employee and store all the employee objects in that collection. Finally, we are performing different types of operations using the methods provided by the List<T> Generic Collection Class.

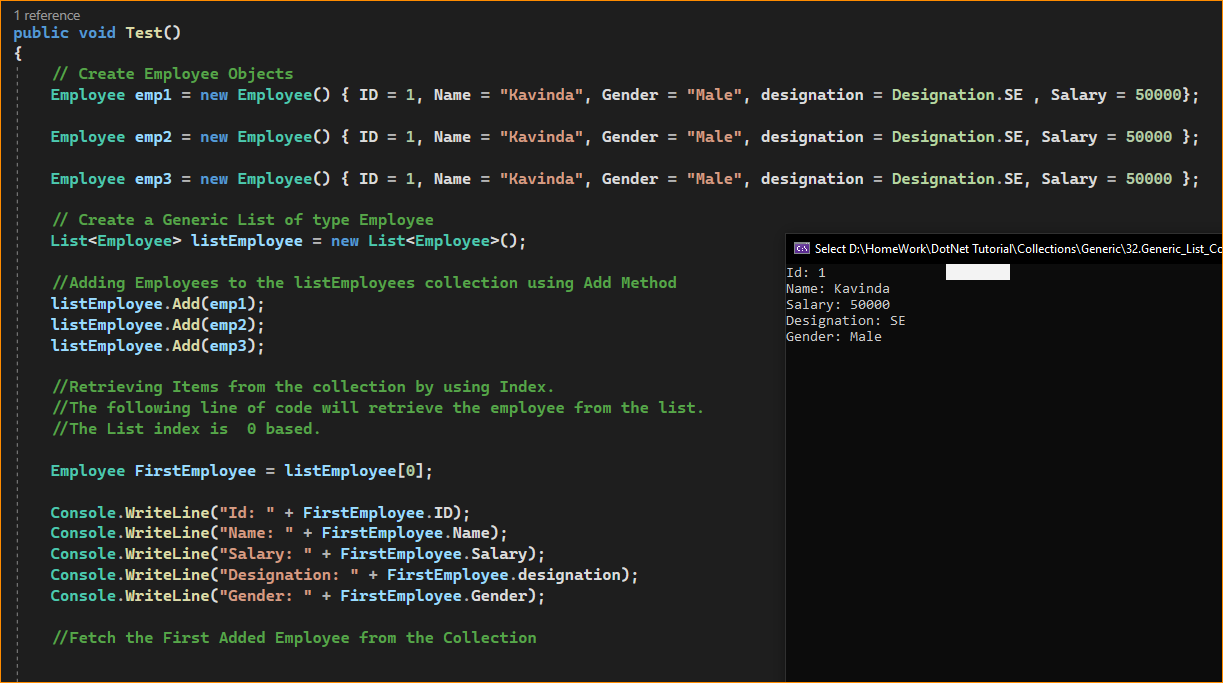
Employee Designation Enum –



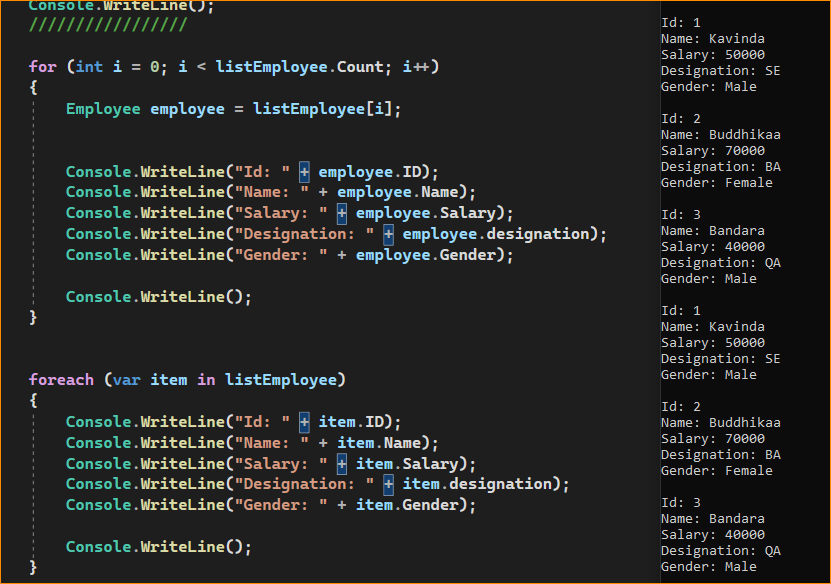
Employee class –



Create list –

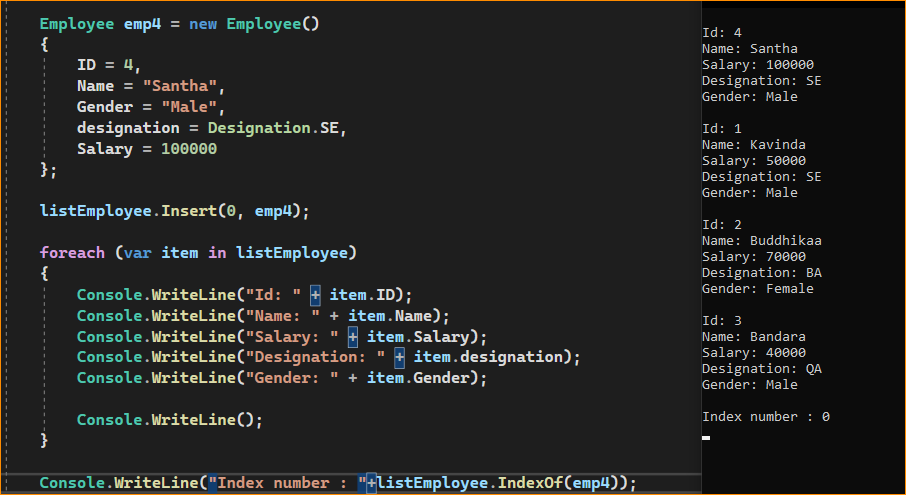


For loop-



Insert New Item –& IndexOf

Insert(elemet, list) and check index IndexOf()

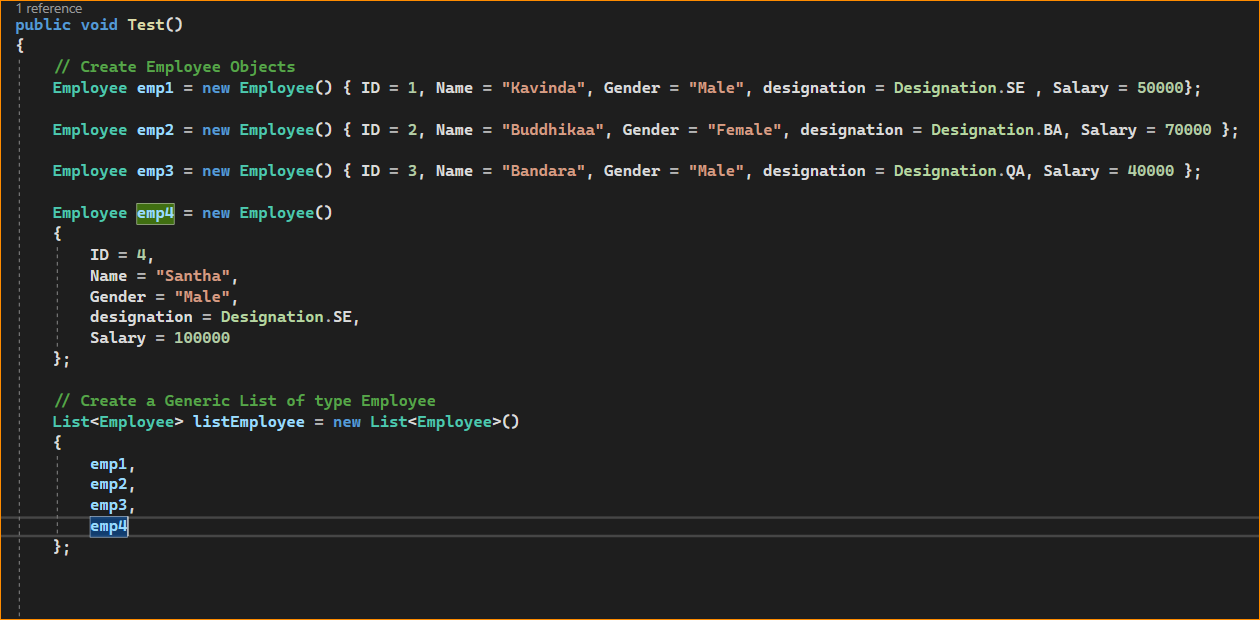


Note: All the Generic Collection Classes in C# are Strongly Typed. That means if we have created a List of type Employee, then we can only add objects of type Employee into the list. If we try to add an object of a different type, then we will get a compile-time error.

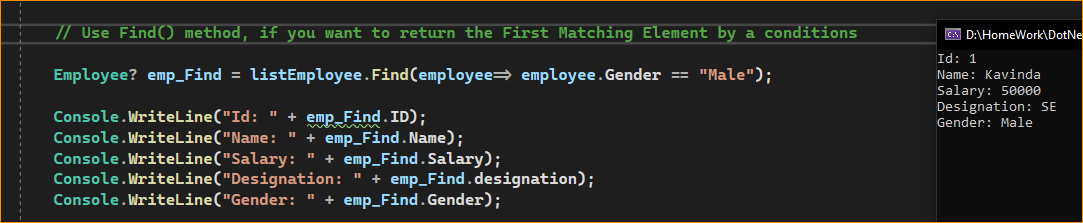
1. How to Find Element in a Generic List Collection in C#?

The Generic List Collection Class in C# provides a lot of useful methods that we can use to find elements on a collection of List Types. The List Collection class provides the following important methods to find elements in a collection.

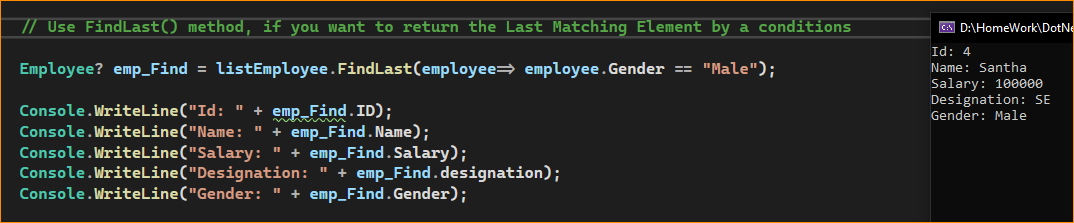
Ex- base list



**Find():** The Find() method is used to find the first element from a list based on a condition that is specified by a lambda expression.

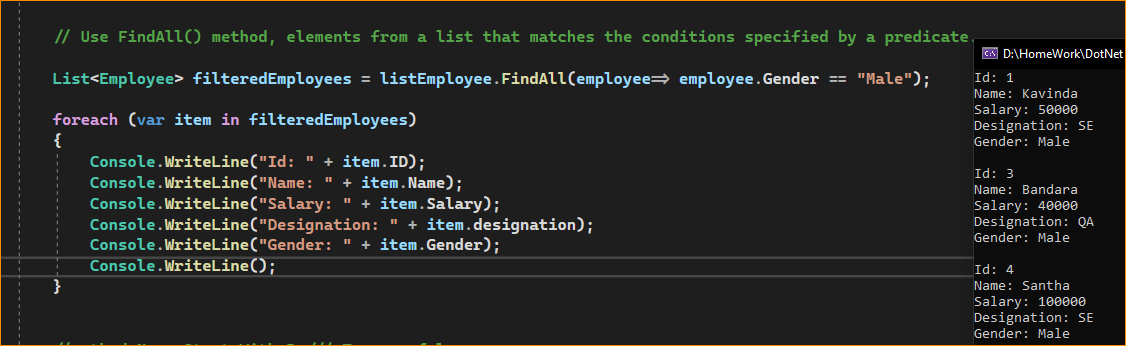


**FindLast():** The FindLast() method is used to search for an element that matches the conditions specified by a predicate. If it found any elements with that specified condition then it returns the Last matching element from the list.

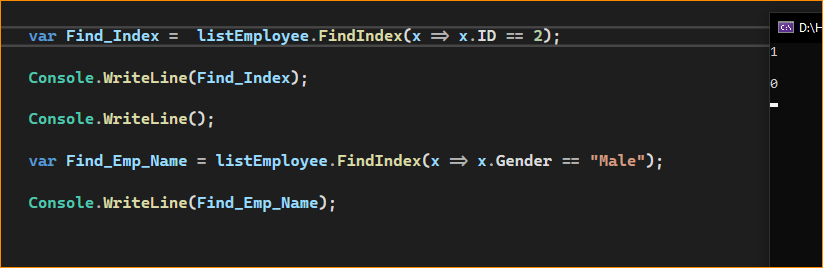


**FindAll():** The FindAll() method is used to retrieve all the elements from a list that matches the conditions specified by a predicate.

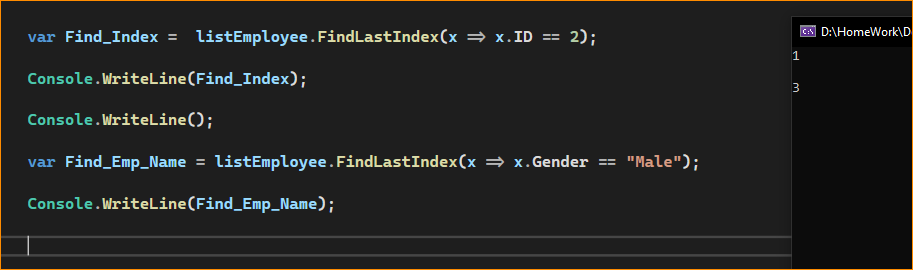
Here we save date to another list type Employee.



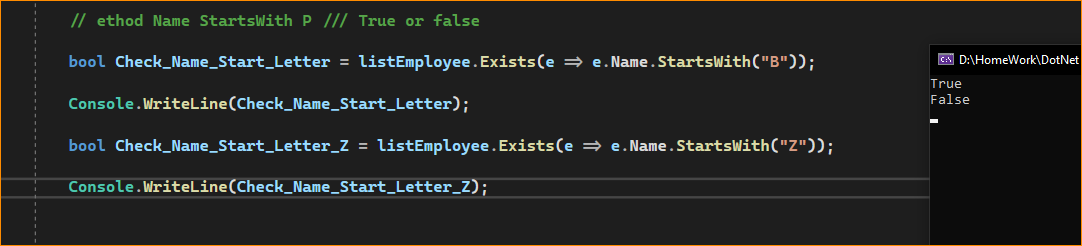
**FindIndex():** The FindIndex() method is used to return the index position of the first element that matches the conditions specified by a predicate. The point that you need to remember is the index here in generic collections is zero-based. This method returns -1 if an element that matches the specified conditions is not found. There are 2 other overloaded versions of this method is available, one of the overload versions allows us to specify the range of elements to search within the list.



**FindLastIndex():** The FindLastIndex() Method searches for an element in the list that matches the condition specified by the lambda expression and then returns the index of the last occurrence of the item within the list. There are 2 other overloaded versions of this method is available, one of the overload versions allows us to specify the range of elements to search within the list.

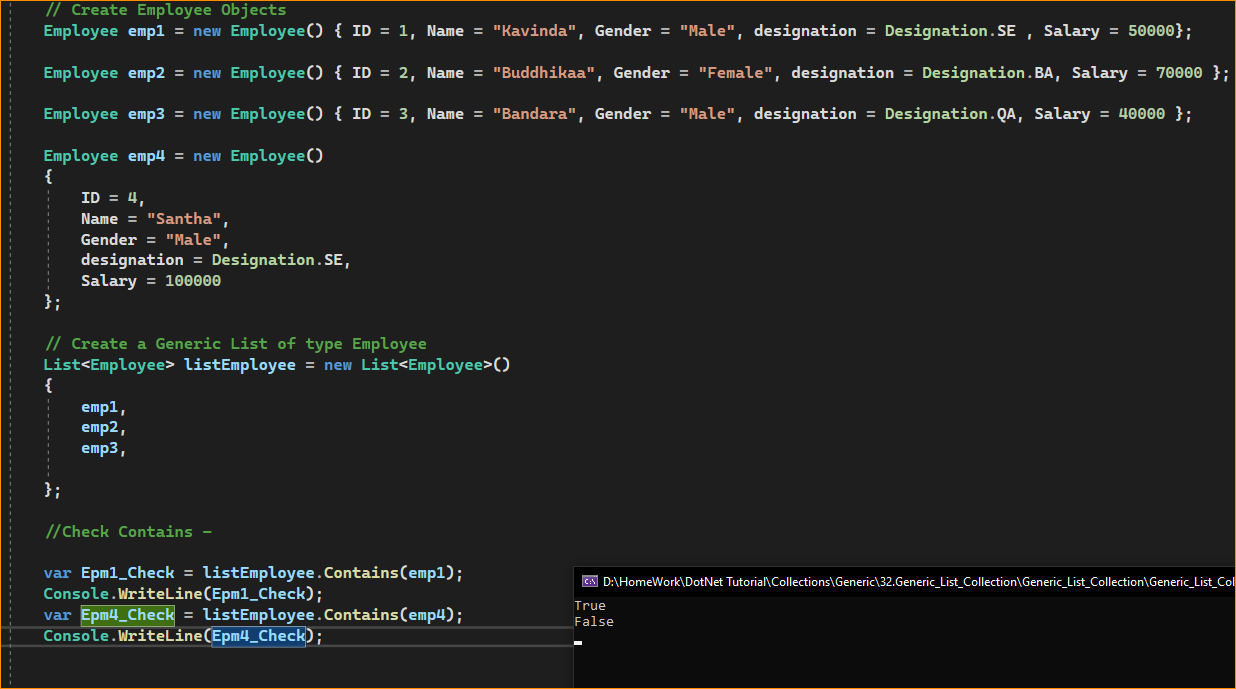


**Exists ():** The Exists() method is used to check or determine whether an item exists or not in a list based on a condition. If the item exists then it will return true else it will return false.



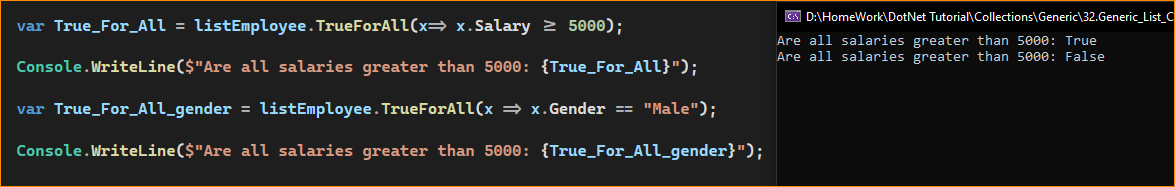
**Contains ():** The Contains() method is used to determine whether the specified item exists or not in the list. If the specified item exists, then it will return true else return false.

Ex- Remove emp04 from the list and test

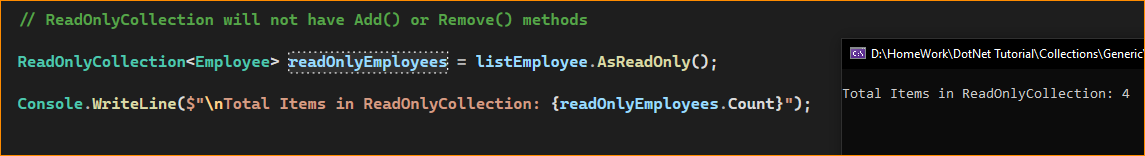


1. Generic List Class important methods in C#:

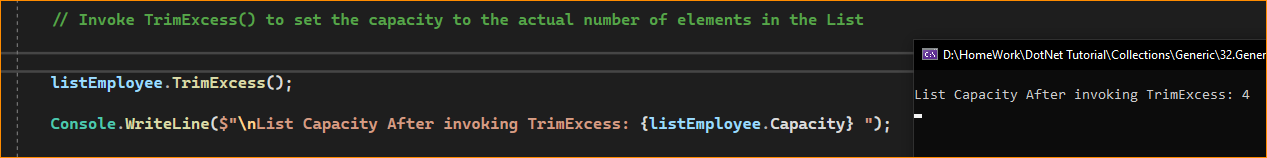
**TrueForAll():** This method returns true or false depending on whether every element in the list matches the conditions defined by the specified predicate.



**AsReadOnly():** This method returns a read-only wrapper for the current collection. Use this method, if you don’t want the client to modify the collection i.e. add or remove any elements from the collection. The ReadOnlyCollection will not have methods to add or remove items from the collection. We can only read items from this collection.



**TrimExcess():** This method sets the capacity to the actual number of elements in the List if that number is less than a threshold value.



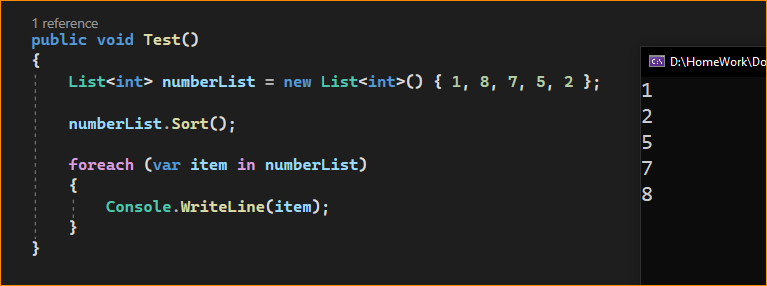
1. How to Sort a List of Simple Types in C#?

In C#, sorting a list of simple types like int, double, char, string, etc. is straightforward. Here, we just need to call the Sort() method which is provided by the Generic List class on the list instance, and then the data will be automatically sorted in ascending order. For example, if we have a list of integers as shown below.

**List<int> numbersList = new List<int>{ 1, 8, 7, 5, 2};**

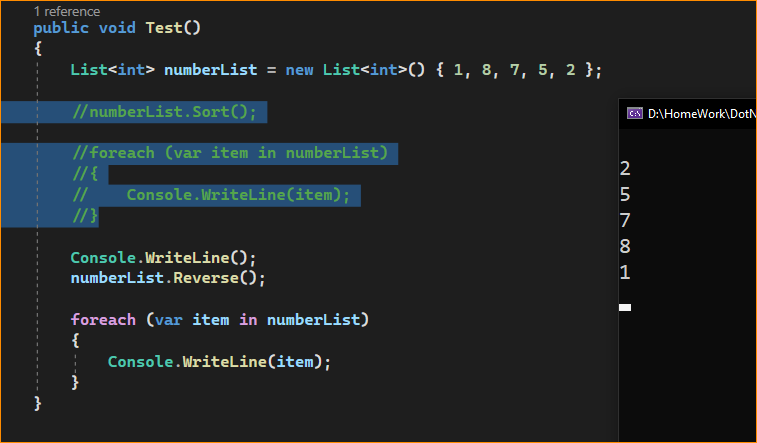
Then we just need to invoke the Sort() method on numbersList collection as shown below

**numbersList.Sort();**

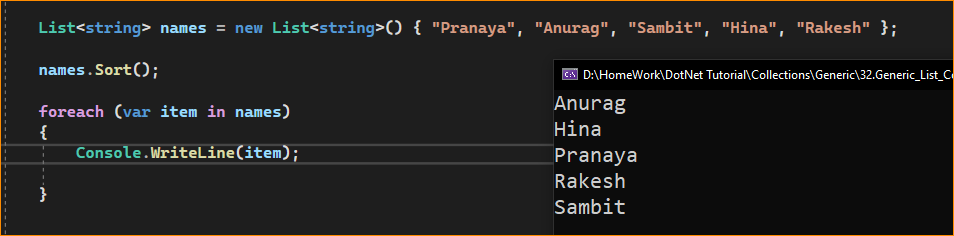


If you want the data to be retrieved in descending order, then use the Reverse() method on the list instance as shown below.

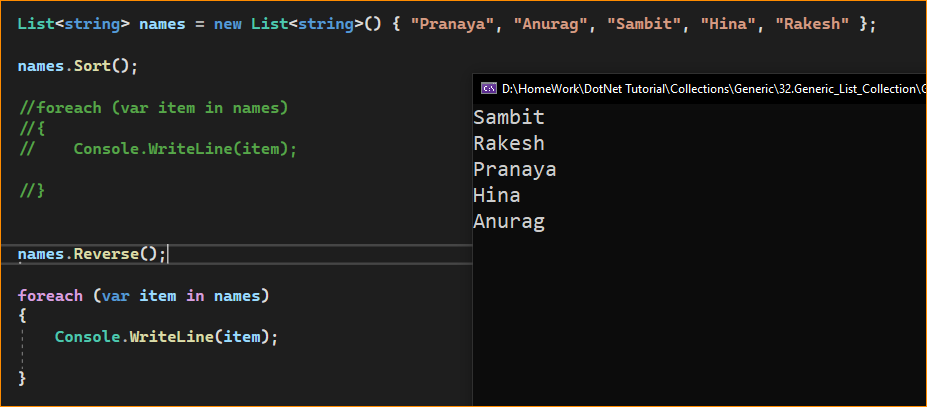
**numbersList.Reverse();**

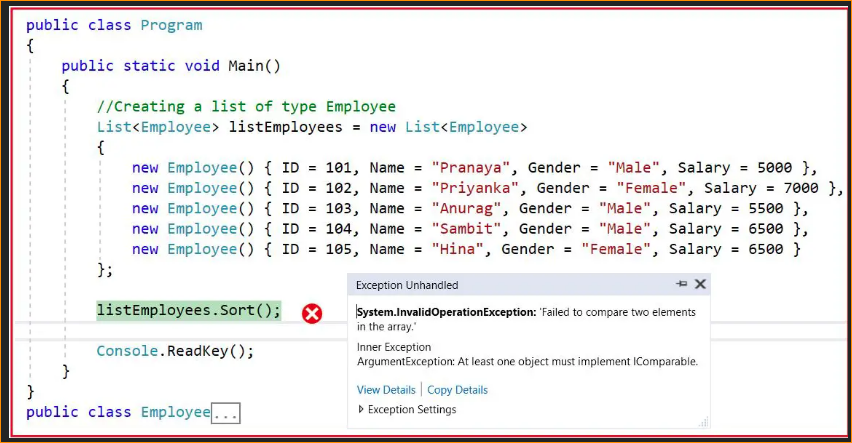


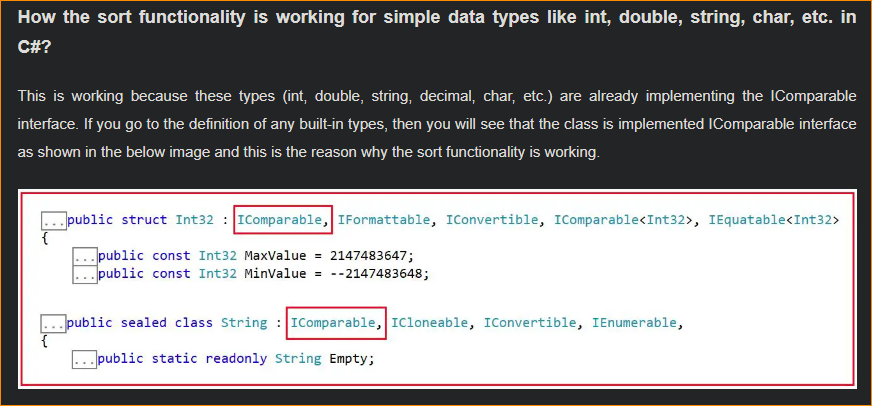
Staring sort- Alphabetically order



Reverse – Alphabetically order reverse







Summary of Generic List<T> Collection Class in C#:

* The List<T> collection is different from the arrays. The List can be resized dynamically but arrays cannot resize dynamically.
* The Generic List<T> Collection Class in C# can accept null values for reference types and it also accepts duplicate values.
* When the element’s Count becomes equal to the Capacity of the list collection, then the capacity of the List increased automatically by reallocating the internal array. The existing elements will be copied to the new array before the addition of the new element.
* The Generic List class is the generic equivalent of the Non-Generic ArrayList class.
* The Generic List<T> class implements the IList<T> generic interface.
* We can use both equality and ordering comparer with the generic List class.
* The List<T> class elements are not sorted by default and elements are accessed by a zero-based index.
* For very large List<T> objects, you can increase the maximum capacity to 2 billion elements on a 64-bit system by setting the enabled attribute of the configuration element to true in the run-time environment.