**The following methods belong to Quantifiers category**  
All  
Any  
Contains   
  
   
  
All these methods return true or false depending on whether if some or all of the elements in a sequence satisfy a condition.  
  
**All()**method returns true if all the elements in a sequence satisfy a given condition, otherwise false.  
  
**Example 1 :**Returns true, as all the numbers are less than 10

int[] numbers = { 1, 2, 3, 4, 5 };

var result = numbers.All(x => x < 10);

Console.WriteLine(result);

There are 2 overloaded versions of **Any()**method. The version without any parameters checks if the sequence contains at least one element. The other version with a predicate parameter checks if the sequence contains at least one element that satisfies a given condition.  
  
**Example 2 :**Returns true as the sequence contains at least one element

int[] numbers = { 1, 2, 3, 4, 5 };

var result = numbers.Any();

Console.WriteLine(result);

**Example 3 :** Returns false as the sequence does not contain any element that satisfies the given condition (No element in the sequence is greater than 10)

int[] numbers = { 1, 2, 3, 4, 5 };

var result = numbers.Any(x => x > 10);

Console.WriteLine(result);

There are 2 overloaded versions of the **Contains()**method. One of the overloaded version checks if the sequence contains a specified element using the default equality comparer. The other overloaded version checks if the sequence contains a specified element using an alternate equality comparer.  
  
**Example 4 :**Returns true as the sequence contains number 3. In this case the default equality comparer is used.

int[] numbers = { 1, 2, 3, 4, 5 };

var result = numbers.Contains(3);

Console.WriteLine(result);

**Example 5 :** Returns true. In this case we are using an alternate equality comparer (StringComparer) for the comparison to be case-insensitive.

string[] countries = { "USA", "INDIA", "UK" };

var result = countries.Contains("india", StringComparer.OrdinalIgnoreCase);

Console.WriteLine(result);

When comparing complex types like **Employee, Customer**etc, the default comparer will only check if the object references are equal, and not the individual property values of the objects that are being compared.  
  
**Example 6 :**Returns false, as the default comparer will only check if the object references are equal.

List<Employee> employees = new List<Employee>()

{

    new Employee { ID = 101, Name = "Rosy"},

    new Employee { ID = 102, Name = "Susy"}

};

var result = employees.Contains(new Employee { ID = 101, Name = "Rosy" });

Console.WriteLine(result);

**To solve the problem in Example 6, there are 3 ways**  
**1.** Use the other overloaded version of **Contains()**method to which we can pass a custom class that implements **IEqualityComparer**  
**2.** Override **Equals()**and **GetHashCode()**methods in **Employee**class  
**3.** Project the properties into a new anonymous type, which overrides **Equals()**and **GetHashCode()**methods  
  
We discussed implementing these **3 options for Distinct() method in**[**Part 26**](http://csharp-video-tutorials.blogspot.com/2014/08/part-26-set-operators-in-linq.html)**of**[**LINQ Tutorial**](https://www.youtube.com/playlist?list=PL6n9fhu94yhWi8K02Eqxp3Xyh_OmQ0Rp6). In the same way these options can be implemented for Contains() method.