**Part 78 - Sort a list of complex types in c#**

This is continuation to [Part 77](http://csharp-video-tutorials.blogspot.com/2013/08/part-77-sort-list-of-simple-types-in-c.html). Please watch [Part 77](http://csharp-video-tutorials.blogspot.com/2013/08/part-77-sort-list-of-simple-types-in-c.html), before proceeding.  
  
To sort a list of complex types without using **LINQ**, the complex type has to implement **IComparable**interface and provide implementation for **CompareTo**() method. CompareTo() method returns an integer, and the meaning of the return value is shown below.  
**Return value greater than ZERO -**The current instance is greater than the object being compared with.  
**Return value less than ZERO -**The current instance is less than the object being compared with.

**Return value is ZERO -**The current instance is equal to the object being compared with.

Alternatively you can also invoke **CompareTo**() method. Salary property of the Customer object is int. CompareTo() method is already implemented on integer type, so we can invoke this method and return it's value as shown below.  
return this.Salary.CompareTo(obj.Salary);  
  
public class Customer : IComparable<Customer>  
{  
    public int ID { get; set; }  
    public string Name { get; set; }  
    public int Salary { get; set; }  
  
    public int CompareTo(Customer obj)  
    {  
        //if (this.Salary > obj.Salary)  
        //    return 1;  
        //else if (this.Salary < obj.Salary)  
        //    return -1;  
        //else  
        //    return 0;  
  
        // OR, Alternatively you can also invoke CompareTo() method.   
        return this.Salary.CompareTo(obj.Salary);  
    }  
}  
  
If you prefer not to use the **Sort functionality provided by the Customer class**, then you can provide your own by implementing **IComparer**interface. For example, if I want the customers to sorted by name instead of salary.  
**Step 1:**Implement IComparer interface  
public class SortByName : IComparer<Customer>  
{  
    public int Compare(Customer x, Customer y)  
    {  
        return x.Name.CompareTo(y.Name);  
    }  
}  
  
**Step 2:** Pass an instance of the class that implements IComparer interface, as an argument to the Sort() method.  
SortByName sortByName = new SortByName();  
listCutomers.Sort(sortByName);  
  
**Here is the Main() method code:**  
public class Program  
{  
    public static void Main()  
    {  
        Customer customer1 = new Customer()  
        {  
            ID = 101,  
            Name = "Mark",  
            Salary = 4000  
        };  
  
        Customer customer2 = new Customer()  
        {  
            ID = 102,  
            Name = "John",  
            Salary = 7000  
        };  
  
        Customer customer3 = new Customer()  
        {  
            ID = 103,  
            Name = "Ken",  
            Salary = 5500  
        };  
  
        List<Customer> listCutomers = new List<Customer>();  
        listCutomers.Add(customer1);  
        listCutomers.Add(customer2);  
        listCutomers.Add(customer3);  
  
        Console.WriteLine("Customers before sorting");  
        foreach (Customer customer in listCutomers)  
        {  
            Console.WriteLine(customer.Name + " - " + customer.Salary);  
        }  
  
        // Sort() method should sort customers by salary  
        listCutomers.Sort();  
  
        Console.WriteLine("Customers after sorting");  
        foreach (Customer customer in listCutomers)  
        {  
            Console.WriteLine(customer.Name + " - " + customer.Salary);  
        }  
  
        // To sort customers by name instead of salary  
        SortByName sortByName = new SortByName();  
        listCutomers.Sort(sortByName);  
  
        Console.WriteLine("Customers after sorting by Name");  
        foreach (Customer customer in listCutomers)  
        {  
            Console.WriteLine(customer.Name + " - " + customer.Salary);  
        }  
    }  
}