**Part 79 - Sort a list of complex types using Comparison delegate**

This is continuation to [Part 78](http://csharp-video-tutorials.blogspot.com/2013/09/part-78-sort-list-of-complex-types-in-c.html). Please watch [Part 78](http://csharp-video-tutorials.blogspot.com/2013/09/part-78-sort-list-of-complex-types-in-c.html), before proceeding.  
  
One of the overloads of the **Sort**() method in List class expects Comparison delegate to be passed as an argument. Let us understand using this overloaded version.  
public void Sort(Comparison<T> comparison)

**Approach 1:**  
**Step 1:**Create a function whose signature matches the signature of System.Comparison delegate. This is the method where we need to write the logic to compare 2 customer objects.  
private static int CompareCustomers(Customer c1, Customer c2)  
{  
    return c1.ID.CompareTo(c2.ID);  
}  
  
**Step 2:** Create an instance of **System.Comparison**delegate, and then pass the name of the function created in Step1 as the argument. So, at this point **"Comparison"** delegate is pointing to our function that contains the logic to compare 2 customer objects.  
Comparison<Customer> customerComparer = new Comparison<Customer>(CompareCustomers);  
  
**Step 3:** Pass the delegate instance as an argument, to Sort() function.  
listCutomers.Sort(customerComparer);  
  
At this point, listCutomers should be sorted using the logic defined in CompareCustomers() function.   
  
**Approach 2:**  
In Approcah1 this is what we have done  
**1.** Created a private function that contains the logic to compare customers  
**2.** Created an instance of Comparison delegate, and then passed the name of the private function to the delegate.

**3.** Finally passed the delegate instance to the Sort() method.  
  
**Do we really have to follow all these steps. Isn't there any other way?**  
The above code can be simplified using delegate keyword as shown below.  
listCutomers.Sort(delegate(Customer c1, Customer c2)   
                    {   
                        return (c1.ID.CompareTo(c2.ID));   
                    });  
  
**Approach 3:**The code in Approach 2, can be further simplified using lambda expression as shown below.  
listCutomers.Sort((x, y) => x.ID.CompareTo(y.ID));  
  
**Example:**  
public class Program  
{  
    public static void Main()  
    {  
        Customer customer1 = new Customer()  
        {  
            ID = 101,  
            Name = "Mark",  
            Salary = 4000  
        };  
  
        Customer customer2 = new Customer()  
        {  
            ID = 103,  
            Name = "John",  
            Salary = 7000  
        };  
  
        Customer customer3 = new Customer()  
        {  
            ID = 102,  
            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.ID);  
        }  
  
        // Approach 1  
        // Step 2: Create an instance of Comparison delegate  
        //Comparison<Customer> customerComparer =   
        //    new Comparison<Customer>(CompareCustomers);  
  
        // Step 3: Pass the delegate instance to the Sort method  
        //listCutomers.Sort(customerComparer);  
  
        // Approach 2: Using delegate keyword  
        //listCutomers.Sort(delegate(Customer c1, Customer c2)  
        //{  
        //    return (c1.ID.CompareTo(c2.ID));  
        //});  
              
        // Aaproach 3: Using lambda expression  
        listCutomers.Sort((x, y) => x.ID.CompareTo(y.ID));  
  
        Console.WriteLine("Customers after sorting by ID");  
        foreach (Customer customer in listCutomers)  
        {  
            Console.WriteLine(customer.ID);  
        }  
  
        listCutomers.Reverse();  
        Console.WriteLine("Customers in descending order of ID");  
        foreach (Customer customer in listCutomers)  
        {  
            Console.WriteLine(customer.ID);  
        }  
    }  
  
    // Approach 1 - Step 1  
    // Method that contains the logic to compare customers  
    private static int CompareCustomers(Customer c1, Customer c2)  
    {  
        return c1.ID.CompareTo(c2.ID);  
    }  
}  
  
public class Customer  
{  
    public int ID { get; set; }  
    public string Name { get; set; }  
    public int Salary { get; set; }  
}