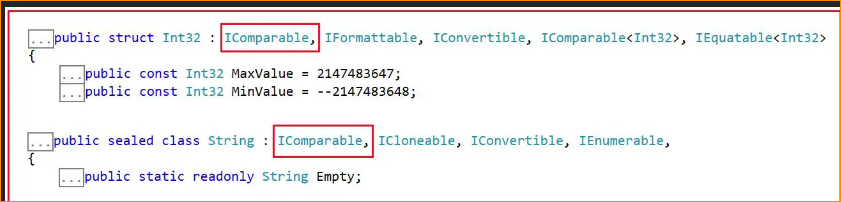
1. How the sort functionality is working for simple data types like int, double, string, char, etc. in C#?

This is working because these types (int, double, string, decimal, char, etc.) are already implementing the IComparable interface. If you go to the definition of any built-in types, then you will see that the class is implemented IComparable interface as shown in the below image and this is the reason why the sort functionality is working.



Generic List<T> Collection Class Sort Methods in C#:

The Generic List Collection Class in C# provides the following four Sort Methods.

Sort(): This method is used to sort the elements in the entire Generic List using the default comparer.

Sort(IComparer<T> comparer): This method is used to sort the elements in the entire Generic List using the specified comparer.

Sort(Comparison<T> comparison): This method is used to sort the elements in the entire Generic List using the specified System.Comparison.

Sort(int index, int count, IComparer<T> comparer): This method is used to sort the elements in a range of elements in a Generic List using the specified comparer.

How to Sort a List of Complex Types in C#?

To sort a list of complex types without using LINQ, the complex type has to implement the IComparable interface and needs to provide the implementation for the CompareTo() method as follows. The CompareTo() method returns an integer value and the meaning of the return value as 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.

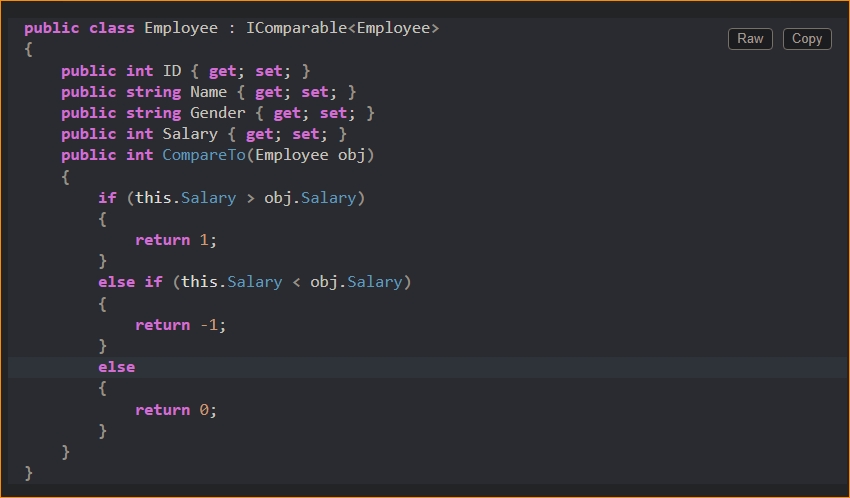
The Return value is ZERO – The current instance is equal to the object being compared with.

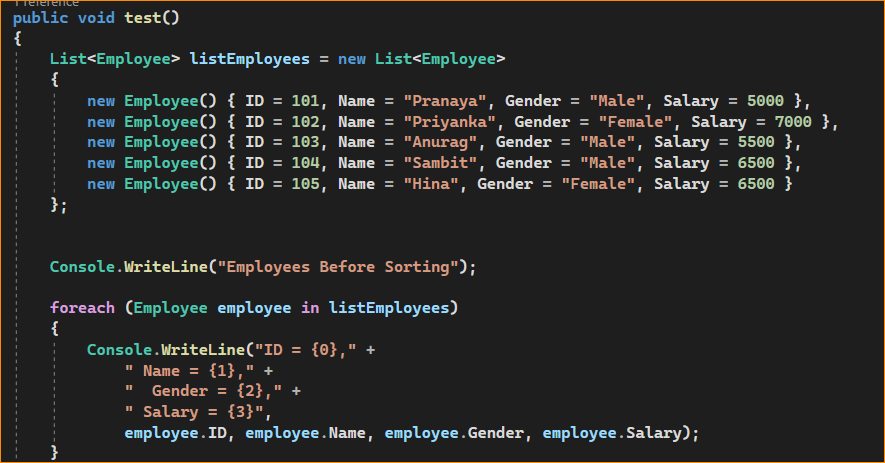
Alternatively, we can also invoke the CompareTo() method directly. The Salary property of the Employee object is int and the CompareTo() method is already implemented on the integer type that we already discussed, so we can invoke this method and return its value as shown below.

**return this.Salary.CompareTo(obj.Salary);**

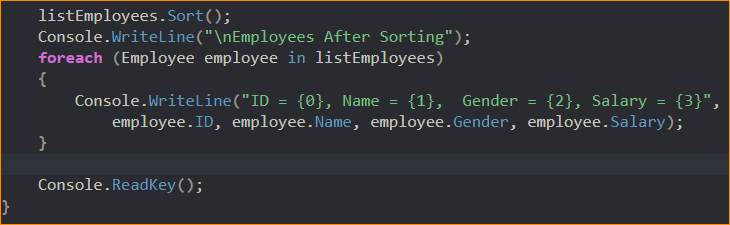
1. Implementing the IComparable interface in C# in Employee Class (Diligation Comparison)

Let us understand this with an example. What we want is, we need to sort the employees based on their Salary. To do so, our Employee class should implement the IComparable interface and should provide an implementation for the CompareTo() method. This method will compare the current object (specified with this) and the object to be compared which it will get as a parameter. So, modify the Employee class as follows and the following code exactly does the same.

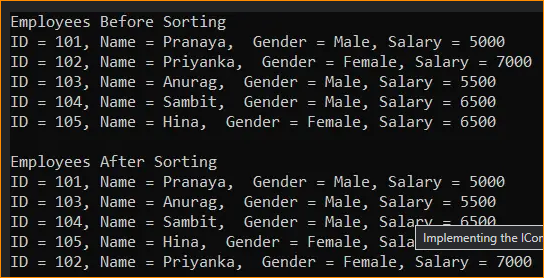




Using this technique-



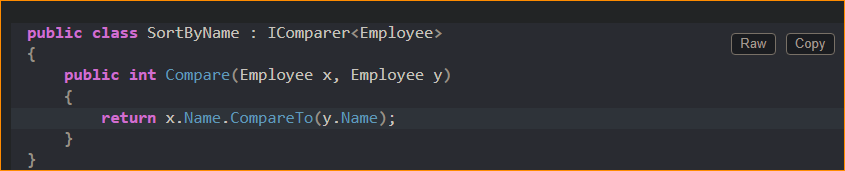
Result –



1. If you do not prefer to use the Sort functionality provided by the Employee class, then you can provide your own implementation by implementing the IComparer interface. For example, if you want the employees to be sorted by Name instead of Salary then you need to follow the below two steps.

Step1: Implement the IComparer interface

Create a class implementing the IComparer<T> interface and provide implementation to the Compare method as follows.

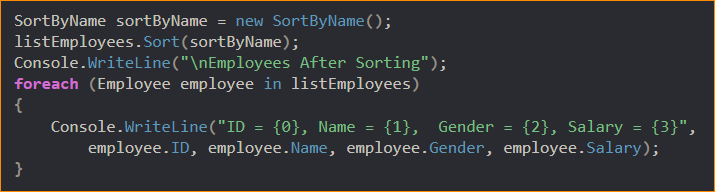


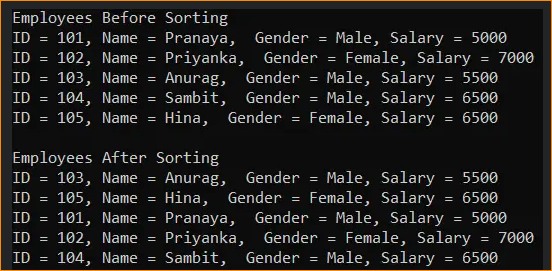
Step2: Use the overloaded version of the Sort method which takes the IComparer interface as a parameter:

Pass an instance of the class that implements the IComparer interface as an argument to the Sort() method as shown below.

**SortByName sortByName = new SortByName();**

**listEmployees.Sort(sortByName);**





1. Sorting Collection of Complex Types using LINQ in C#:

Using LINQ OrderBy and OrderByDescending methods we can easily sort the elements of a collection of complex types in C#. What we need to do is, we need to call the OrderBy method if we want to Sort the elements in ascending and the OrderByDescending method if we want to sort the elements in descending order. For a better understanding please have a look at the below example.

