IComparable, IComparer And IEquatable Interfaces In C#

During implementation, often question rises on how to sort a collection of objects. To sort a collection requires how objects can first of all be compared to each other. A value type such as int, double, float can be compared if both of the objects have equal values. However, a reference type such as a class with multiple fields, the question is often difficult to answer. Well, in other words it depends. It depends how two objects are said to be compared / equated; when all fields have the same value or one of them is enough to decide if they are equal.  
  
C# provides a variety of interfaces to achieve the required behavior. Let’s have a look on them one by one:

1. **public** **class** Car
2. {
3. **public** **string** Name { **get**; **set**; }
4. **public** **int** MaxSpeed { **get**; **set**; }
5. }

**IComparable Interface:**  
Interface has a CompareTo method that takes a reference type as a parameter and returns an integer based on if current instance precedes, follows or occurs in the same position in the sort order as the other object (MSDN).  
  
The implementation of the CompareTo(Object) method must return an Int32 that has one of the three values, as in the following table.

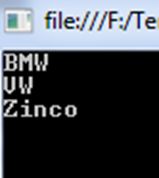
|  |  |
| --- | --- |
| **Value** | **Meaning** |
| Less than zero | The current instance precedes the object specified by the CompareTo method in the sort order. |
| Zero | This current instance occurs in the same position in the sort order as the object specified by the CompareTo method. |
| Greater than zero | This current instance follows the object specified by the CompareTo method in the sort order. |

**Example**

1. **public** **class** Car: IComparable
2. {
3. **public** **string** Name
4. {
5. **get**;
6. **set**;
7. }
8. **public** **int** MaxSpeed
9. {
10. **get**;
11. **set**;
12. }
13. **public** **int** CompareTo(**object** obj)
14. {
15. **if** (!(obj **is** Car))
16. {
17. **throw** **new** ArgumentException("Compared Object is not of car");
18. }
19. Car car = obj **as** Car;
20. **return** Name.CompareTo(car.Name);
21. }
22. }

**At client:**

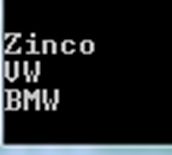
1. **private** **static** **void** Main(**string**[] args)
2. {
3. Car[] cars = **new** Car[]
4. {
5. **new** Car()
6. {
7. Name = "Zinco"
8. }, **new** Car()
9. {
10. Name = "VW"
11. }, **new** Car()
12. {
13. Name = "BMW"
14. }
15. };
16. Array.Sort(cars);
17. Array.ForEach(cars, x => Console.WriteLine(x.Name));
18. }

**Output**  
  
  
 **IComparer interface**  
The CompareTo method from IComparable interface can sort on only one field at a time, so sorting on different properties with it is not possible. IComparer interface provides Compare method that Compares two objects and returns a value indicating whether one is less than, equal to, or greater than the other.  
  
A class that implements the IComparer interface must provide a Compare method that compares two objects.  
  
For example, you could create a CarComparer class that implements IComparer and that has a Compare method that com¬pares Car objects by Name. You could then pass a CarComparer object to the *Array.Sort method*, and it can use that object to sort an array of Car objects.

1. **public** **class** CarComparer: IComparer < Car >
2. {
3. **public** **enum** SortBy
4. {
5. Name,
6. MaxSpeed
7. }
8. **private** SortBy compareField = SortBy.Name;
9. **public** **int** Compare(Car x, Car y)
10. {
11. **switch** (compareField)
12. {
13. **case** SortBy.Name:
14. **return** x.Name.CompareTo(y.Name);
15. **break**;
16. **case** SortBy.MaxSpeed:
17. **return** x.MaxSpeed.CompareTo(y.MaxSpeed);
18. **break**;
19. **default**:
20. **break**;
21. }
22. **return** x.Name.CompareTo(y.Name);
23. }
24. }

**At client side**

1. **private** **static** **void** Main(**string**[] args)
2. {
3. Car[] cars = **new** Car[]
4. {
5. **new** Car()
6. {
7. Name = "Zinco"
8. }, **new** Car()
9. {
10. Name = "VW"
11. }, **new** Car()
12. {
13. Name = "BMW"
14. }
15. };
16. var carComparer = **new** CarComparer();
17. carComparer.compareField = CarComparer.SortBy.MaxSpeed;
18. Array.Sort(cars, carComparer);
19. }

**Output**  
  
  
  
**IEquatable Interface**  
If a class implements the IComparable interface, it provides a CompareTo method that enables you to determine how two objects should be ordered. Sometimes, you may not need to know how two objects should be ordered, but you need to know instead whether the objects are equal. The IEquatable interface provides that capability by requiring a class to provide an Equals method.

1. **class** Employee: IEquatable < Person >
2. {
3. **public** **string** FirstName
4. {
5. **get**;
6. **set**;
7. }
8. **public** **string** LastName
9. {
10. **get**;
11. **set**;
12. }
13. **public** **bool** Equals(Employee other)
14. {
15. **return** ((FirstName == other.FirstName) && (LastName == other.LastName));
16. }
17. }

**At client**

1. // The List of Persons.
2. **private** List < Employee > Employees = **new** List < Employee > ();
3. // Add a Person to the List.
4. // Make the new Person.
5. Employee emp = **new** Employee()
6. {
7. FirstName = “James”
8. LastName = “Moore”
9. };
10. **if** (Employees.Contains(emp))
11. {
12. MessageBox.Show("The list already contains this employee.");