# Comparator Interface - Java Collections

In Java, Comparator interface is used to order(sort) the objects in the collection in your own way. It gives you the ability to decide how elements will be sorted and stored within collection and map.

Comparator Interface defines compare() method. This method has two parameters. This method compares the two objects passed in the parameter. It returns 0 if two objects are equal. It returns a positive value if object1 is greater than object2. Otherwise, a negative value is returned. The method can throw a ClassCastException if the type of objects is not compatible for comparison.

- **Rules for using Comparator interface**

If you want to sort the elements of a collection, you need to implement Comparator interface.

If you do not specify the type of the object in your Comparator interface, then, by default, it assumes that you are going to sort the objects of type Object. Thus, when you override the compare() method ,you will need to specify the type of the parameter as Object only.

If you want to sort the user-defined type elements, then while implementing the Comparator interface, you need to specify the user-defined type generically. If you do not specify the user-defined type while implementing the interface,then by default, it assumes Object type and you will not be able to compare the user-defined type elements in the collection

For Example:

If you want to sort the elements according to roll number, defined inside the class Student, then while implementing the Comparator interface, you need to mention it generically as follows:

class MyComparator implements Comparator<Student>{}

If you write only,

class MyComparator implements Comparator {}

Then it assumes, by default, data type of the compare() method's parameter to be Object, and hence you will not be able to compare the Student type(user-defined type) objects.

Example!

**Creating student class:**

```
class Student

{

 int roll;

 String name;

 Student(int r,String n)

 {

    roll = r;

    name = n;

 }

 public String toString()

 {

    return roll+" "+name;

 }

}
```

**MyComparator class:**

This class defines the comparison logic for Student class based on their roll. Student object will be sorted in ascending order of their roll.

```
class MyComparator implements Comparator<Student>

{

 public int compare(Student s1,Student s2)

  {

     if(s1.roll == s2.roll) return 0;
```

```
        else if(s1.roll > s2.roll) return 1;
        else return -1;
    }
}
```

Now let's create a Test class with main() function,

```
public class Test {
   public static void main(String[] args) {
       ArrayList< Student> ts = new ArrayList< Student>();
       ts.add(new Student(45, "Rahul"));
       ts.add(new Student(11, "Adam"));
       ts.add(new Student(19, "Alex"));
       Collections.sort(ts,new MyComparator()); /*passing the name of the
ArrayList and the object of the class that implements Comparator in a
predefined sort() method in Collections class*/
       System.out.println(ts);
   }
}
```

Output

```
[ 11 Adam, 19 Alex, 45 Rahul ]
```

As you can see in the output Student object are stored in ascending order of their roll.

- **Notes:**

1. When we are sorting elements in a collection using Comparator interface, we need to pass the class object that implements Comparator interface.

2. when ArrayList is used, then we need to call sort method of Collections class and pass the name of the collection and it’s object as a parameter.