Comparator Interface in Java with Examples

Difficulty Level : Medium Last Updated : 21 Dec, 2021

A comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class**.** Following function compare obj1 with obj2.

Syntax:

```
public int compare(Object obj1, Object obj2):
```

Suppose we have an Array/ArrayList of our own class type, containing fields like roll no, name, address, DOB, etc, and we need to sort the array based on Roll no or name?

Method 1: One obvious approach is to write our own sort() function using one of the standard algorithms. This solution requires rewriting the whole sorting code for different criteria like Roll No. and Name.

Method 2: Using comparator interface- Comparator interface is used to order the objects of a user-defined class. This interface is present in java.util package and contains 2 methods compare(Object obj1, Object obj2) and equals(Object element). Using a comparator, we can sort the elements based on data members. For instance, it may be on roll no, name, age, or anything else.

Method of Collections class for sorting List elements is used to sort the elements of List by the given comparator.

```
public void sort(List list, ComparatorClass c)
```

To sort a given List, ComparatorClass must implement a Comparator interface.

How do the sort() method of Collections class work?

Internally the Sort method does call Compare method of the classes it is sorting. To compare two elements, it asks “Which is greater?” Compare method returns -1, 0, or 1 to say if it is less than, equal, or greater to the other. It uses this result to then determine if they should be swapped for their sort.

Example

```java
// Java Program to Demonstrate Working of
// Comparator Interface

// Importing required classes
import java.io.*;
import java.lang.*;
import java.util.*;

// Class 1
// A class to represent a Student
class Student {

    // Attributes of a student
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name, String address)
    {

        // This keyword refers to current instance itself
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    // Method of Student class
    // To print student details in main()
    public String toString()
    {

        // Returning attributes of Student
        return this.rollno + " " + this.name + " "
            + this.address;
    }
}

// Class 2
// Helper class implementing Comparator interface
class Sortbyroll implements Comparator<Student> {

    // Method
    // Sorting in ascending order of roll number
    public int compare(Student a, Student b)
    {

        return a.rollno - b.rollno;
    }
}
```

```java
// Class 3
// Helper class implementing Comparator interface
class Sortbyname implements Comparator<Student> {

    // Method
    // Sorting in ascending order of name
    public int compare(Student a, Student b)
    {

        return a.name.compareTo(b.name);
    }
}

// Class 4
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating an empty ArrayList of Student type
        ArrayList<Student> ar = new ArrayList<Student>();

        // Adding entries in above List
        // using add() method
        ar.add(new Student(111, "Mayank", "london"));
        ar.add(new Student(131, "Anshul", "nyc"));
        ar.add(new Student(121, "Solanki", "jaipur"));
        ar.add(new Student(101, "Aggarwal", "Hongkong"));

        // Display message on console for better readability
        System.out.println("Unsorted");

        // Iterating over entries to print them
        for (int i = 0; i < ar.size(); i++)
            System.out.println(ar.get(i));

        // Sorting student entries by roll number
        Collections.sort(ar, new Sortbyroll());

        // Display message on console for better readability
        System.out.println("\nSorted by rollno");

        // Again iterating over entries to print them
        for (int i = 0; i < ar.size(); i++)
            System.out.println(ar.get(i));

        // Sorting student entries by name
        Collections.sort(ar, new Sortbyname());

        // Display message on console for better readability
        System.out.println("\nSorted by name");

        // // Again iterating over entries to print them
        for (int i = 0; i < ar.size(); i++)
            System.out.println(ar.get(i));
    }
}
```

Output

```
Unsorted
111 Mayank london
131 Anshul nyc
121 Solanki jaipur
101 Aggarwal Hongkong

Sorted by rollno
101 Aggarwal Hongkong
111 Mayank london
121 Solanki jaipur
131 Anshul nyc

Sorted by name
101 Aggarwal Hongkong
131 Anshul nyc
111 Mayank london
121 Solanki jaipur
```

By changing the return value inside the compare method, you can sort in any order that you wish to, for example: For descending order just change the positions of ‘a’ and ‘b’ in the above compare method.

Sort collection by more than one field

In the previous example, we have discussed how to sort the list of objects on the basis of a single field using Comparable and Comparator interface But, what if we have a requirement to sort ArrayList objects in accordance with more than one field like firstly, sort according to the student name and secondly, sort according to student age.

Example

```java
// Java Program to Demonstrate Working of
// Comparator Interface Via More than One Field

// Importing required classes
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;

// Class 1
// Helper class representing a Student
class Student {

    // Attributes of student
    String Name;
    int Age;

    // Parameterized constructor
    public Student(String Name, Integer Age)
    {

        // This keyword refers to current instance itself
        this.Name = Name;
        this.Age = Age;
    }

    // Getter setter methods
    public String getName() { return Name; }

    public void setName(String Name) { this.Name = Name; }

    public Integer getAge() { return Age; }

    public void setAge(Integer Age) { this.Age = Age; }

    // Method
    // Overriding toString() method
    @Override public String toString()
    {
        return "Customer{"
            + "Name=" + Name + ", Age=" + Age + '}';
    }

    // Class 2
    // Helper class implementing Comparator interface
    static class CustomerSortingComparator
        implements Comparator<Student> {
```

```
        // Method 1
        // To compare customers
        @Override
        public int compare(Student customer1,
                           Student customer2)
        {

            // Comparing customers
            int NameCompare = customer1.getName().compareTo(
                customer2.getName());

            int AgeCompare = customer1.getAge().compareTo(
                customer2.getAge());

            // 2nd level comparison
            return (NameCompare == 0) ? AgeCompare
                                      : NameCompare;
        }
    }

    // Method 2
    // Main driver method
    public static void main(String[] args)
    {

        // Create an empty ArrayList
        // to store Student
        List<Student> al = new ArrayList<>();

        // Create customer objects
        // using constructor initialization
        Student obj1 = new Student("Ajay", 27);
        Student obj2 = new Student("Sneha", 23);
        Student obj3 = new Student("Simran", 37);
        Student obj4 = new Student("Ajay", 22);
        Student obj5 = new Student("Ajay", 29);
        Student obj6 = new Student("Sneha", 22);

        // Adding customer objects to ArrayList
        // using add() method
        al.add(obj1);
        al.add(obj2);
        al.add(obj3);
        al.add(obj4);
        al.add(obj5);
        al.add(obj6);

        // Iterating using Iterator
        // before Sorting ArrayList
        Iterator<Student> custIterator = al.iterator();

        // Display message
        System.out.println("Before Sorting:\n");

        // Holds true till there is single element
        // remaining in List
        while (custIterator.hasNext()) {

            // Iterating using next() method
            System.out.println(custIterator.next());
        }

        // Sorting using sort method of Collections class
        Collections.sort(al,
                         new CustomerSortingComparator());

        // Display message only
        System.out.println("\n\nAfter Sorting:\n");

        // Iterating using enhanced for-loop
        // after Sorting ArrayList
        for (Student customer : al) {
            System.out.println(customer);
        }
    }
}
```

Output

```
Before Sorting:


Customer{Name=Ajay, Age=27}
Customer{Name=Sneha, Age=23}
Customer{Name=Simran, Age=37}
Customer{Name=Ajay, Age=22}
Customer{Name=Ajay, Age=29}
Customer{Name=Sneha, Age=22}



After Sorting:


Customer{Name=Ajay, Age=22}
Customer{Name=Ajay, Age=27}
Customer{Name=Ajay, Age=29}
Customer{Name=Simran, Age=37}
Customer{Name=Sneha, Age=22}
Customer{Name=Sneha, Age=23}
```

This article is contributed by **Rishabh Mahrsee**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.