HowToDoInJava

Java Comparator Interface

Last Updated: January 4, 2022 By: Lokesh Gupta Java Collections Java Compare

Java **Comparator** interface is used to sort an array or List of objects based on **custom sort order**. The custom ordering of items is imposed by implementing Comparator's *compare()* method in the objects.

Table Of Contents

1. When to Use Comparator Interface

Java `Comparator` interface imposes a **total ordering** on the objects which may not have a desired natural ordering.

For example, for a *List* of *Employee* objects, the natural order may be ordered by employee's id. But in real-life applications, we may want to sort the list of employees by their first name, date of birth or simply any other such criteria. In such conditions, we need to use `Comparator` interface.

We can use the *Comparator* interface in the following situations.

- Sorting the array or list of objects, but **NOT in natural order**.
- Sorting the array or list of objects where we **can not modify the source code** to implement Comparable interface.
- Using **group by sort** on list or array of objects on multiple different fields.

2. Overriding compare() Method

To enable total ordering on objects, we need to create a class that implements the *Comparator* interface. Then we need to override its `compare(T o1, T o2)` method.

The *compare()* compares its two arguments for *order*. It returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

The implementor must also ensure that the relation is **transitive**: `((compare(x, y)>0) && (compare(y, z)>0))` implies `compare(x, z)>0`.

For a given *Employee* class, the order by employee name can be imposed by creating a *Comparator* like below.

NameSorter.java

```
import java.util.Comparator;
```

```
public class NameSorter implements Comparator<Employee>
{
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.getName().compareToIgnoreCase( e2.getName() );
    }
}
```

Employee.java

```
import java.time.LocalDate;

public class Employee {

    private Long id;
    private String name;
    private LocalDate dob;
}
```

3. Using Comparator With

3.1. Collections.sort() and Arrays.sort()

1. Use **Collections.sort(list, Comparator)** method sort a **list** of objects in order imposed by provided comparator instance.
2. Use **Arrays.sort(array, Comparator)** method sort an **array** of objects in order imposed by provided comparator instance.

3.2. Collections.comparing()

This utility method accepts a function that extracts a sort key for the class. This is essentially a field on which the class objects will be sorted.

```
//Order by name
Comparator.comparing(Employee::getName);

//Order by name in reverse order
Comparator.comparing(Employee::getName).reversed();

//Order by id field
Comparator.comparing(Employee::getId);

//Order by employee age
Comparator.comparing(Employee::getDate);
```

3.3. Collections.thenComparing()

This utility method is used for **group by sort**. Using this method, we can chain multiple comparators to sort the list or array of objects on multiple fields.

It is very similar to **SQL GROUP BY clause** to order rows on different fields.

```
//Order by name and then by age
Comparator.comparing(Employee::getName)
          .thenComparing(Employee::getDob);

//Order by name -> date of birth -> id
Comparator.comparing(Employee::getName)
          .thenComparing(Employee::getDob)
          .thenComparing(Employee::getId);
```

Using the above syntax, we can create virtually any sorting logic.

3.4. Collections.reverseOrder()

This utility method returns a comparator that imposes the reverse of the *natural ordering* or *total ordering* on a collection of objects that implement the Comparable

interface.

```
//Reverse of natural order as specified in
//Comparable interface's compareTo() method

Comparator.reversed();

//Reverse of order by name

Comparator.comparing(Employee::getName).reversed();
```

3.5. Other Collection Classes

Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps) to provide an ordering that is not natural ordering.

```
SortedSet<Employee> sortedUniqueEmployees = new TreeSet<Employee>(nev
```

4. Java Comparator Examples

4.1. Sorting List of Custom Objects

Java example to **sort a list of employees by name** using Comparator.

```
ArrayList<Employee> list = new ArrayList<>();

//Sort in reverse natural order
Collections.sort(list, new NameSorter());
```

4.2. Sort List in Reverse Order

Java example to **sort a list of employees by name** using Comparator in **reverse order**.

```
ArrayList<Employee> list = new ArrayList<>();

Collections.sort(list, Comparator.comparing( Employee::getName ).reve
```

4.3. Group By Sorting

Java example to sort a list of employees on multiple fields i.e. **field by field**.

```
ArrayList<Employee> list = new ArrayList<>();

Comparator<Employee> groupByComparator = Comparator.comparing(Employe
                                         .thenComparing(Employee::getL
                                         .thenComparing(Employee::getL

Collections.sort(list, groupByComparator);
```

5. Conclusion

In this tutorial, we learned about *Comparator* interface of Java collection framework. It helps in imposing a total order on objects without any change to the source code of that class.

We learned to sort list and array of custom objects. We also learned to *reverse sort* and *implement group by sort in Java*.

Happy Learning !!

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Java 8 Comparator example with lambda
2. Java 8 – Comparator thenComparing() example
3. Sorting with Comparable and Comparator
4. Java Comparable Interface
5. Java Iterator interface example
6. Java ListIterator interface
7. Java Spliterator interface
8. Interface vs Abstract Class in Java
9. Java Cloneable interface – Is it broken?
0. Private Methods in Interface – Java 9

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial