HowToDoInJava

# Java Comparable Interface

Last Updated: May 5, 2022 By: Lokesh Gupta Java Collections Java Compare

**Java Comparable** interface is part of Collection Framework and is used to **sort an array or list of objects based on their natural ordering**. The natural ordering is defined by implementing its *compareTo()* method in the objects.

Table Of Contents

## 1. Comparable Interface

*Comparable* interface has a single abstract method ***compareTo()*** that objects need to implement to have a natural ordering.

- The objects must be mutually comparable and must not throw *ClassCastException* for any *key* in the collection.
- The *compareTo()* method must return a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
- Note that *compareTo()* must throw an exception if `y.compareTo(x)` throws an exception.
- Also, the relationship between the comparable objects must be **transitive** i.e. `(x.compareTo(y) > 0 && y.compareTo(z) > 0)` implies `x.compareTo(z)>0`.
- `null` is not an instance of any class so `e.compareTo(null)` should throw a `NullPointerException`.

**Comparable.java**

```
public interface Comparable<T>
{
    public int compareTo(T o);
}
```

For example, for `Employee` class, the natural ordering can be based on the `id` field.

**Employee.java**

```
import java.time.LocalDate;

public class Employee implements Comparable<Employee> {

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

    @Override
    public int compareTo(Employee o)
    {
        return this.getId().compareTo( o.getId() );
    }
}
```

Using *Comparable* interface, we can sort all types of objects including strings, wrapper classes or custom objects.

All *wrapper classes* and *String* already implement `Comparable` interface. Wrapper classes are compared by their values, and strings are compared **lexicographically**.

## 2. Using Comparable

We can use sort the objects, that implement *Comparable* interface, using the following ways:

## 2.1. Collections.sort() and Arrays.sort()

- Use `Collections.sort()` method sort a **List** of objects.
- Use `Arrays.sort()` method sort an **array** of objects.

```
Collections.sort(items);
Arrays.sort(items);
```

## 2.2. Collections.reverseOrder()

This utility method returns a *Comparator* that **imposes the reverse of the *natural ordering*** on a collection of objects.

This enables a simple idiom for sorting (or maintaining) collections (or arrays) of objects that implement the Comparable interface in **reverse-natural-order**.

```
Collections.sort(items, Collections.reverseOrder());
Arrays.sort(items, Collections.reverseOrder());
```

## 2.3. Sorted Collections

Objects that implement this interface can be used as keys in a sorted map or as elements in a sorted set (e.g. *TreeSet*), without the need to specify a comparator.

```
//All all items are automatically sorted
SortedSet<Item> itemsSet = new TreeSet<>();
```

## 2.4. Streams

Stream.sorted() can be used to sort a stream of objects that implement *Comparable* interface. However, note that a *stream.sorted()* does not sort the original collection – *only*

*the items in the stream are sorted.*

```
items.stream()
    .sorted()
    .forEach(i -> System.out.println(i);
```

## 3. Comparable Examples

All given examples sort the lists using **`Collections.sort()`** method. If we need to sort the arrays of objects, simply replace *Collections.sort()* with *Arrays.sort()*.

### 3.1. Sorting Strings

Java program to sort a List of strings using Comparable interface.

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

list.add("E");
list.add("A");
list.add("C");
list.add("B");
list.add("D");

Collections.sort(list);

System.out.println(list);
```

Program Output.

**Output**

```
[A, B, C, D, E]
```

## 3.2. Sort Strings in Reverse Order

Java program to sort a list of strings in reverse order using Comparable interface.

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

list.add("E");
list.add("A");
list.add("C");
list.add("B");
list.add("D");

//Sort in reverse natural order
Collections.sort(list, Collections.reverseOrder());

System.out.println(list);
```

Program Output.

```
[E, D, C, B, A]
```

## 3.3. Sorting Integers

Java program to sort a list of integers, on natural order and reverse order, using Comparable interface.

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

list.add(10);
list.add(300);
list.add(45);
list.add(2);
list.add(5);

//Natural order
```

```
Collections.sort(list);

System.out.println(list);

//Sort in reverse natural order
Collections.sort(list, Collections.reverseOrder());

System.out.println(list);
```

Program Output.

```
[2, 5, 10, 45, 300]
[300, 45, 10, 5, 2]
```

## 3.4. Sort List of Custom Objects

In this example, we are **sorting a list of employees by id**.

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

list.add(new Employee(22l, "Lokesh", LocalDate.now()));
list.add(new Employee(18l, "Alex", LocalDate.now()));
list.add(new Employee(30l, "Bob", LocalDate.now()));
list.add(new Employee(600l, "Charles", LocalDate.now()));
list.add(new Employee(5l, "David", LocalDate.now()));

//Natural order
Collections.sort(list);

System.out.println(list);

//Sort in reverse natural order
Collections.sort(list, Collections.reverseOrder());

System.out.println(list);
```

Program Output.

```
[
    Employee [id=5, name=David, dob=2018-10-29],
    Employee [id=18, name=Alex, dob=2018-10-29],
    Employee [id=22, name=Lokesh, dob=2018-10-29],
    Employee [id=30, name=Bob, dob=2018-10-29],
    Employee [id=600, name=Charles, dob=2018-10-29]
]

//Reverse sorted

[
    Employee [id=600, name=Charles, dob=2018-10-30],
    Employee [id=30, name=Bob, dob=2018-10-30],
    Employee [id=22, name=Lokesh, dob=2018-10-30],
    Employee [id=18, name=Alex, dob=2018-10-30],
    Employee [id=5, name=David, dob=2018-10-30]
]
```

## 4. Conclusion

In this tutorial, we learned about *Comparable* interface. This interface helps in imposing a natural order on objects with simple interface implementation.

We also learned to sort a list of strings, an array of strings, list of integers, an array of integers and we learned how to sort employee objects in java using comparable.

Happy Learning !!

> No

## Recommended Reading:

1. Sorting with Comparable and Comparator
2. Java Comparator Interface
3. Java Iterator interface example
4. Java ListIterator interface
5. Java Spliterator interface
6. Interface vs Abstract Class in Java
7. Java Cloneable interface – Is it broken?
8. Private Methods in Interface – Java 9
9. Spring boot – CommandLineRunner interface example
0. Spring Boot Async Rest Controller with Callable Interface

## 3 thoughts on “Java Comparable Interface”

### Süleyman Emir Akın

November 12, 2021 at 10:09 am

I still do not understand why we have to implement comparable interface when I want to define compareTo method can someone explain this thank you.

Reply

**Lokesh Gupta**

November 12, 2021 at 1:35 pm

You want to implement the `Comparable` interface and this is your goal. To meet that goal, you have to define `compareTo()` method.

Think again. Defining `compareTo()` method is not your goal. Your goal is to implement the `Comparable` interface.

Reply

**ilham sekti**

January 7, 2022 at 9:32 pm

You need to use the method collections.sort() to sort the objects in your list and to do so you need to define the compareTo() method which will be used in the sorting process.
Another point is that collections.sort() method is only available in classes that implement the Comparable interface.

I hope this was useful.

Reply

## Leave a Comment

Name *

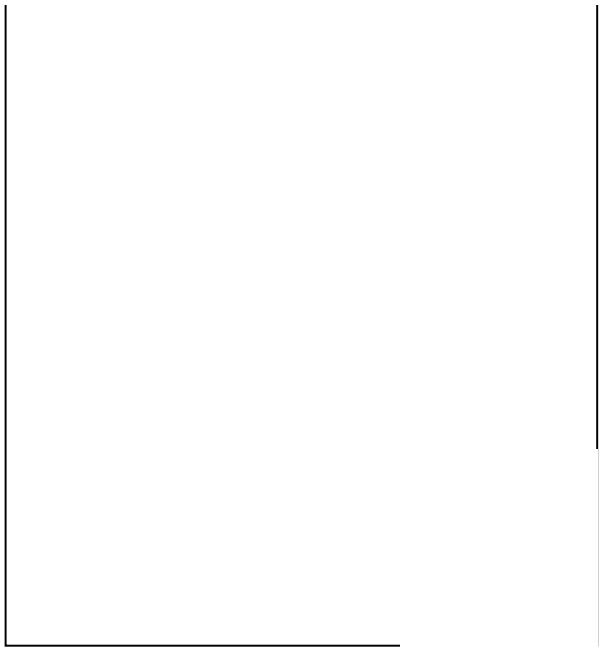
Email *

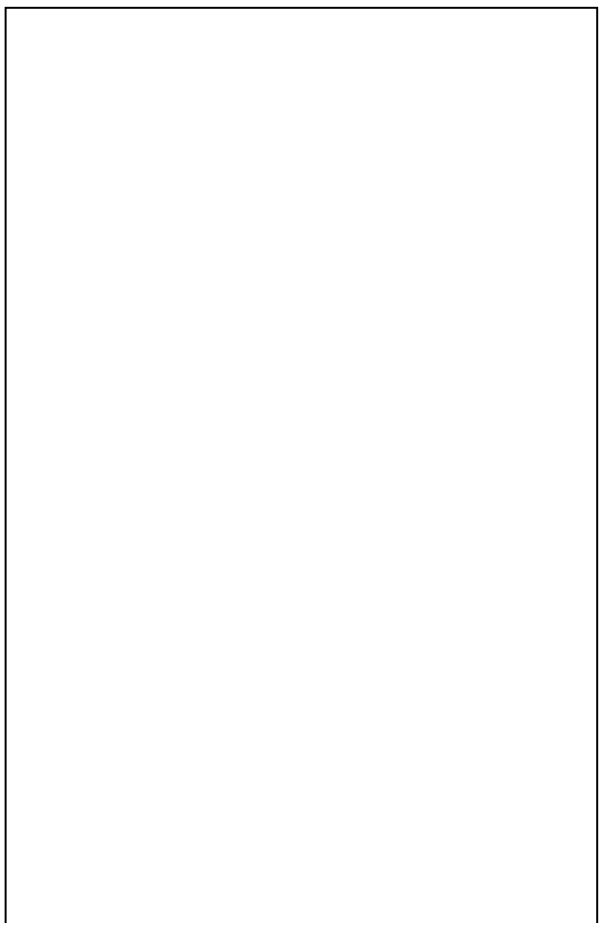
Website

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