In Java programming language, an interface is used to specify a behavior that classes must implement. Java world offers us two such interfaces Comparable and Comparator! Comparable in Java is used to sort the objects with natural ordering while Comparator is used to sort attributes of different objects. Let's understand these interfaces through the medium of this article.

I have covered the following pointers which demonstrate comparable and comparator in Java:

Now that we are clear with our agenda, let's begin!

What is Comparable in Java?

As the name itself suggests, **Comparable** is an interface which defines a way to compare an object with other objects of the same type. It helps to sort the objects that have self-tendency to sort themselves, i.e., the objects must know how to order themselves. **Eg:** Roll number, age, salary. This interface is found in *java.lang package* and it contains only one method, i.e., *compareTo().* Comparable is not capable of sorting the objects on its own, but the interface defines a method *int compareTo()* which is responsible for sorting.

Further, you must be thinking what is the compareTo method? Well, let me explain that to you!

What is the compareTo method and how it is used?

This method is used to compare the given object with the current object. The **compareTo()** method returns an int value. The value can be either positive, negative, or zero. So now we are well acquainted with the theoretical knowledge of Comparable interface in Java and **compareTo** method.

Let's hop into understanding the implementation process. First, let's see how to implement Comparable.

Java Comparable example

Below code depicts the usage of comparable in Java.

```
public class Student implements Comparable {
    private String name;
    private int age;
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public int getAge() {
        return this.age;
    }
    public String getName() {
        return this.name;
    }
    @Override
    public String toString() {
        return "";
    }
    @Override
    public int compareTo(Student per) {
        if(this.age == per.age)
            return 0;
        else
            return this.age > per.age ? 1 : -1;
    }

    public static void main(String[] args) {
        Person e1 = new Person("Adam", 45);
        Person e2 = new Person("Steve", 60);
        int retval = e1.compareTo(e2);

        switch(retval) {
            case -1: {
System.out.println("The " + e2.getName() + " is older!");
                break;
```

```
                                    }

                                case 1: {
                    System.out.println("The " + e1.getName() + " is older!");
                                     break;
                                    }

                                default:
                System.out.println("The two persons are of the same age!");

                                    }
                                    }
                                    }
```

In the above example, I have created a class Student with two fields, name and age. Class Student is implementing the Comparable interface and overrides the compareTo method. This method sorts the instances of the Student class, based on their age.

Now that I have covered Comparable in Java, moving on I will talk about another interface i.e. Comparator in Java. Let's move to understanding Comparator in Java!

What is Comparator in Java?

A Comparator interface is used to order the objects of a specific class. This interface is found in java.util package. It contains two methods;

- compare(Object obj1,Object obj2)
- equals(Object element).

The first method, `compare(Object obj1,Object obj2)` compares its two input arguments and showcase the output. It returns a negative integer, zero, or a positive integer to state whether the first argument is less than, equal to, or greater than the second.

The second method, `equals(Object element),` requires an Object as a parameter and shows if the input object is equal to the comparator. The method will return true, only if the mentioned object is also a Comparator. The order remains the same as that of the Comparator.

After attaining brief learning about Comparator in Java, it's time to move a step ahead. Let me show you an example depicting Comparator in Java.

Java, J2EE & SOA Certification Training

Reviews
4(39395)

Python Scripting Certification Training

Reviews
5(8599)

Python Django Training and Certification

Reviews
5(4388)

Comprehensive Java Course Certification Training

Reviews
5(16201)

Mastering Perl Scripting Certification Training

Reviews
5(4287)

Spring Framework Certification Training

Reviews
5(7899)

PHP & MySQL with MVC Frameworks Certification Training

Reviews
5(3001)

Node.js Certification Training

Reviews
5(5857)

How to implement Comparator in Java

Here is an example of using Comparator in Java:

```
import java.util.Comparator;

public class School {
private int num_of_students;
private String name;
public Company(String name, int num_of_students) {
this.name = name;
this.num_of_students = num_of_students;
}
public int getNumOfStudents() {
return this.num_of_students;
}
public String getName() {
return this.name;
}
}
public class SortSchools implements Comparator {
@Override
public int compare(School sch1, School sch2) {
if(sch1.getNumOfStudents()== sch2.getNumOfStudents())
return 0;
else
return sch1.getNumOfStudents() > sch2.getNumOfStudents() ? 1 : -1;
}
public static void main(String[] args) {
School sch1 = new School("sch1", 20);
School sch2 = new School("sch2", 15);
SortSchools sortSch = new SortSchools();
int retval = sortSch.compare(sch1, sch2);
switch(retval) {
case -1: {
System.out.println("The " + sch2.getName() + " is bigger!");
break;
}
case 1: {
System.out.println("The " + sch1.getName() + " is bigger!");
break;
}
default:
System.out.println("The two schools are of the same size!");
}
}
}

```

```
Output:
The sch1 is bigger!
```

Well, no need to panic here. The above-written code is really easy to understand. Let's go!

First, I created a class School that consists of the name and age of the students. After that, I created another class, SortSchools, in order to implement the Comparator interface which accomplishes the goal of imposing an order between instances of the first class named, School, according to the number of students.

After understanding about Comparator as well as Comparable in Java, let's move towards our next topic.

Comparable v/s Comparator in Java

Comparable in Java	Comparator in Java
Comparable interface is used to sort the objects with natural ordering.	Comparator in Java is used to sort attributes of different objects.
Comparable interface compares "this" reference with the object specified.	Comparator in Java compares two different class objects provided.
Comparable is present in java.lang package.	A Comparator is present in the java.util package.
Comparable affects the original class, i.e., the actual class is modified.	Comparator doesn't affect the original class
Comparable provides compareTo() method to sort elements.	Comparator provides compare() method, equals() method to sort elements.

I hope the above-mentioned differences brought some clarity regarding the two concepts.

With this, we have reached towards the end of our article. Hope that the content turned out to be informative and imparted knowledge to your Java world. Stay tuned!

See Batch Details

*Now that you have understood Java Collections, check out the **Java training** by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. Edureka's Java J2EE and SOA training and certification course is designed for students and professionals who want to be a Java Developer. The course is designed to give you a head start into Java programming and train you for both core and advanced Java concepts along with various Java frameworks like Hibernate & Spring.*

Got a question for us? Please mention it in the comments section of this "Comparable in Java" blog and we will get back to you as soon as possible.