# Comparable and Comparator – a detailed discussion

dharam

## Introduction

This article is in response to a question asked on one of the groups I am a member of. What is the difference between Comparable and Comparator interfaces in Java. Why there are two interfaces if what they do is provide a method to compare.

This might not seem to be a topic worth discussion for people who are experts in java, but yeah for many of my friends out there, it might help.

## Purpose of the article

The article will describe in detail the difference between the interfaces Comparable and Comparator and also define the usages of both of them. One thing we need to understand is that, they can't be used interchangeably, they are two distinct interfaces and they have their own purpose to exist.

## Comparable

You can actually read the comments in the java source code and understand, but I find it tough for a newbie to understand the java doc comments. So here it is in the simplest English possible. When we say Compar**able** (please note the **able** in the word), it actually means that the interface enables or gives the classes (which implements it) the ability to compare their instances with each other.

Also, the interface provides a method compareTo which takes a parameter T and returns an integer. This means that if a class implements this interface, it has to provide implementation for the compareTo method as well. Now the logic inside can be anything, anything at all. You are free to write your own comparison logic and say how the object makes a comparison with other objects of the same kind (I say same kind because, its pointless to compare two things of different kind, something like comparing a train and a dog might not make sense).

### How to use Comparable?

We define a class, make it implement the Comparable interface and provide implementation code for the compareTo method. Below is a class Car and we want the instances of the Car class to be capable of comparing themselves to each other. There can be more than one property which might be used for a healthy comparison, for simplicity we use just one property for now, the registration number.

```
1  public class Car implements Comparable<Car> {
2
3  int registrationNumber;
4  String brand;
5  int color;
6
7  public int compareTo(Car o) {
8  if (o.registrationNumber < this.registrationNumber)
9  return 1;
```

```
10  else if (o.registrationNumber > this.registrationNumber)
11  return -1;
12  return 0;
13  }
14
15  public int getRegistrationNumber() {
16  return registrationNumber;
17  }
18
19  public void setRegistrationNumber(int registrationNumber) {
20  this.registrationNumber = registrationNumber;
21  }
22
23  public String getBrand() {
24  return brand;
25  }
26
27  public void setBrand(String brand) {
28  this.brand = brand;
29  }
30
31  public int getColor() {
32  return color;
33  }
34
35  public void setColor(int color) {
36  this.color = color;
37  }
38  }
```

The important part in the above code is the compareTo method, it returns 1 if the registrationNumber of this car is greater than the car object passed in the method as parameter. It returns -1 if the situation is reversed and it returns a zero otherwise.

## Consequences of using Comparable

As we saw above we, can use Comparable and make the instances of our classes comparable, this satisfies the basic need of comparing two instances of our Car class. Now, here comes the first shock, what if I wrote this car class, generated a jar out of it, and sent it across to all the programmers to use this in their Code?

Everyone includes the jar and instantiate my comparable Car classes and the are happy using it. One of the programmer has a requirement to compare the car objects on the basis of the color code and not the registrationNumber. What is the solution?

## Solving the problem – multiple flavors of comparison

He is left with two basic options as mentioned below:
1) Write his/her own Car class, make it implement the Comparable interface and provide implementation code to the compareTo method to compare the color codes instead of registrationNumber.

2) Extend our Car class to something like **MySpecialCar extends Car** and override the compareTo

method to compare the color codes.

If you think you have solved the problem, then sorry to disappoint you, this brings in another case, suppose the same programmer wants to compare the cars based on registration number in one part of his/her code and do the comparison based on color codes in other part of the code.
Is it OK? Now either there has to be one instance of Car and one instance of MySpecialCar to represent just one car so that both the parts of the code can use their comparable instances.

You can image what happens if there are four of five criteria to compare. So, lets try to fix this, and you guessed it right, the answer is **Comparator.**

## How to use Comparator

Now that we know, the problem which arises with usage of Comparable, we can effectively understand the reason Comparator is used. The Comparator is not related to our Car object at all, if we want to Compare our Car objects, we really don't need the Car object to implement Comparator.

It is something like this, you tell me that you want to compare two Car objects on the basis of registrationNumber and I create a RegistrationNumberComparator class which will extend the Comparator interface and give you an instance of my RegistrationNumberComparator class, which is a comparator ( an object which is specialized in comparing two cars based on their registrationNumber).

Now the second need might arise to compare based on the color codes and I create a ColorCodeComparator class which will extend the Comparator interface and the comparing logic would be to compare the color codes.

The method compare(T a, T b) is the method of the Comparator interface which accepts two parameters, a and b ,where a has to be compared to b. The comparison logic is whatever you write in the implementation.

### Code for Comparator

```
1   import java.util.Comparator;
2
3   public class RegistrationNumberComparator implements Comparator<Car>{
4
5   public int compare(Car o1, Car o2) {
6   if(o1.getRegistrationNumber() > o2.getRegistrationNumber())
7   return 1;
8   else if(o1.getRegistrationNumber() < o2.getRegistrationNumber())
9   return -1;
10  return 0;
11  }
12  }
13
14  class ColorCodeComparator implements Comparator<Car>{
15
16  public int compare(Car o1, Car o2) {
17  if(o1.getColor() > o2.getColor())
18  return 1;
19  else if(o1.getColor() < o2.getColor())
20  return -1;
21  return 0;
```

```
22  }
23  }
24
25
```

Your Car class need not implement the Comparable interface in the above scenario, hence it will not have any compareTo method as well.

### Usage

Mostly we need comparison when there is a need of storing the objects in some ordered fashion in a Collection. Comparisons are also required while sorting and while implementing certain data structures. So you can always pass the desired Comparator to the Collection you are trying to use. It's up to the user's choice, how to use the Comparator classes he/she created as a part of the above exercise.

## The Debate

The big debate is, if the Comparator interface is so easy and flexible to use, then why use Comparable which has almost no flexibility and the classes cannot be re-used as is, if they implement Comparable.

We can see it very clearly, the restrictions imposed by the Comparable objects help us in restricting the way how objects are compared by default. So, I am sure it is clear to all the readers that Comparable is necessary. To elaborate it, what if I don't want anyone else to compare the objects I created in any fashion other than I wish. There are several such examples, for e.g.: you would never want an integer to be compared to another by there phonetics(how they sound), you will always want it to be compared by their size.

Yes you guessed it right, all the final classes must implement the Comparable interface, to ensure that they are not mis-compared in any code using them. I always advocate using Comparator until you are trying to make a class final.

Few of them are, String, Integer, Byte etc.

## Conclusion

We learnt the difference between Comparable and Comparator, tried to understand the problem when using Comparable and also solved the problem with the Comparator interface.

Hope this helps, happy learning. Please feel free to comment if you like/dislike the article and if you have any question regarding the same.
Download this article as PowerPoint presentation.
Stay connected and stay Subscribed

Copyright © 2015· All Rights Reserved· Techie Me