- Program Creek
  - R
  - Research
  - Machine Learning
  - Contact
- Si nple Java
- va 8
- ding Interview
- va
  - Java Basics
  - Java Object Oriented Concepts
  - Java Collections & Generics
  - Java File I/O
  - Java Database
  - Java Multi-Threading
  - Java XML Parsing
  - Advanced Topics
- DP Stories
- API Examples
- Web Frameworks
  - Struts 2
  - GWT
  - Spring
- Eclipse
  - Eclipse Architecture
  - RCP
  - JDT

20 Like 10 Tweet 13 g+1

Comparable vs. Comparator in Java

By X Wang

`Comparable` and `Comparator` are two interfaces provided by Java Core API. From their names, we can tell they may be used for comparing stuff in some way. But what exactly are they and what is the difference between them? The following are two examples for answering this question. The simple examples compare two HDTV's size. How to use Comparable vs. Comparator is obvious after reading the code.

1. Comparable

`Comparable` is implemented by a class in order to be able to comparing object of itself with some other objects. The class itself must implement the interface in order to be able to compare its instance(s). The method required for implementation is `compareTo()`. Here is an example:

```
class HDTV implements Comparable<HDTV> {
        private int size;
        private String brand;

        public HDTV(int size, String brand) {
                this.size = size;
                this.brand = brand;
        }

        public int getSize() {
                return size;
        }

        public void setSize(int size) {
                this.size = size;
        }

        public String getBrand() {
                return brand;
        }

        public void setBrand(String brand) {
                this.brand = brand;
        }

        @Override
        public int compareTo(HDTV tv) {

                if (this.getSize() > tv.getSize())
                        return 1;
                else if (this.getSize() < tv.getSize())
                        return -1;
                else
                        return 0;
        }
}

public class Main {
        public static void main(String[] args) {
                HDTV tv1 = new HDTV(55, "Samsung");
                HDTV tv2 = new HDTV(60, "Sony");

                if (tv1.compareTo(tv2) > 0) {
                        System.out.println(tv1.getBrand() + " is better.");
                } else {
                        System.out.println(tv2.getBrand() + " is better.");
                }
        }
}
```

```
Sony is better.
```

2. Comparator

In some situations, you may not want to change a class and make it comparable. In such cases, `Comparator` can be used if you want to compare objects based on certain attributes/fields. For example, 2 persons can be compared based on `height` or `age` etc. (this can not be done using comparable.)

thod required to implement is *compare()*. Now let's use another way to e those TV by size. One common use of `Comparator` is sorting. Both ns and `Arrays` classes provide a sort method which use a `Comparator`.

```
 java.util.ArrayList;
 java.util.Collections;
 java.util.Comparator;

HDTV {
  private int size;
  private String brand;

  public HDTV(int size, String brand) {
          this.size = size;
          this.brand = brand;
  }

  public int getSize() {
          return size;
  }

  public void setSize(int size) {
          this.size = size;
  }

  public String getBrand() {
          return brand;
  }

  public void setBrand(String brand) {
          this.brand = brand;
  }
}

class SizeComparator implements Comparator<HDTV> {
        @Override
        public int compare(HDTV tv1, HDTV tv2) {
                int tv1Size = tv1.getSize();
                int tv2Size = tv2.getSize();

                if (tv1Size > tv2Size) {
                        return 1;
                } else if (tv1Size < tv2Size) {
                        return -1;
                } else {
```

```
                                return 0;
                        }
                }
}

public class Main {
        public static void main(String[] args) {
                HDTV tv1 = new HDTV(55, "Samsung");
                HDTV tv2 = new HDTV(60, "Sony");
                HDTV tv3 = new HDTV(42, "Panasonic");

                ArrayList<HDTV> al = new ArrayList<HDTV>();
                al.add(tv1);
                al.add(tv2);
                al.add(tv3);

                Collections.sort(al, new SizeComparator());
                for (HDTV a : al) {
                        System.out.println(a.getBrand());
                }
        }
}
```

Output:

```
Panasonic
Samsung
Sony
```

Often we may use `Collections.reverseOrder()` method to get a descending order Comparator. Like the following:

```
ArrayList<Integer> al = new ArrayList<Integer>();
al.add(3);
al.add(1);
al.add(2);
System.out.println(al);
Collections.sort(al);
System.out.println(al);

Comparator<Integer> comparator = Collections.reverseOrder();
Collections.sort(al,comparator);
System.out.println(al);
```

Output:

```
[3,1,2]
[1,2,3]
[3,2,1]
```

3. When to use Which?

In brief, a class that implements `Comparable` will be comparable, which means it instances can be compared with each other.

A class that implements `Comparator` will be used in mainly two situations: 1) It can be passed to a sort method, such as `Collections.sort()` or `Arrays.sort()`, to allow
control over the sort order and 2) It can also be used to control the
certain data structures, such as sorted sets (e.g. `TreeSet`) or sorted maps
`eeMap`).

mple, to create a `TreeSet`. We can either pass the constructor a
ator or make the object class comparable.

ch 1 - TreeSet(Comparator comparator)

```
      Dog {
        int size;

        Dog(int s) {
                size = s;
        }
}

class SizeComparator implements Comparator<Dog> {
        @Override
        public int compare(Dog d1, Dog d2) {
                return d1.size - d2.size;
        }
}

public class ImpComparable {
        public static void main(String[] args) {
                TreeSet<Dog> d = new TreeSet<Dog>(new SizeComparator()); // pass c
                d.add(new Dog(1));
                d.add(new Dog(2));
                d.add(new Dog(1));
        }
}
```

Approach 2 - Implement Comparable

```
class Dog implements Comparable<Dog>{
        int size;

        Dog(int s) {
                size = s;
        }
        @Override
        public int compareTo(Dog o) {
                return o.size - this.size;
        }
}

public class ImpComparable {
        public static void main(String[] args) {
                TreeSet<Dog> d = new TreeSet<Dog>();
                d.add(new Dog(1));
                d.add(new Dog(2));
                d.add(new Dog(1));
        }
```

ces:
parable
parator

You May Also Like ...

1. **Sort LinkedList of User-Defined Objects in Java**
2. **Implement Comparable for a TreeSet**
3. **How Developers Sort in Java?**
4. **Java – Sort Map By Value**

Category >> Common Methods >> Versus

If you want to post code, please put the code inside <pre> and </pre> tags.

9 Comments programcreek Login

Sort by Best Share Favorite

Join the discussion…

Sachin P • a year ago
Two ways for sorting - using comparator and comparable. For detailed explanation is here
11 • Reply • Share ›

kk • a year ago
Under Heading 2.Comparator
"Comparator is capable of comparing two DIFFERENT types of objects."
EDIT: Comparator is capable if comparing objects based on different attributes. e.g. 2 men can be compared based on `name` or `age` etc. (this can not be done using comparable.) Even in the example shown above, author provides example where object of SAME type (HDTV) is being used.
7 • Reply • Share ›

ryanlr Mod → kk • a year ago
True. Changed.
1 • Reply • Share ›

Anil Nivargi • 7 months ago
Thanks good explaination....Here another blog also explained nice please go through this blog http://adnjavainterview.blogsp...
1 • Reply • Share ›

Anurag Chaturvedi • 5 months ago
What I feel, that when you are creating new class and which needs to be sorted or compared, it should be implemented by Comparable and if we have some pre-developed class and need to add functinality for compare or sorting, then we need to create a class implemented by Comprator and use this class as a parameter in sort method.
• Reply • Share ›

Micheal • 8 months ago
hi and one more doubt is Collections.sort(al, new SizeComparator());
what is this actually doing
my guess sizecomparatorobject.compare(obj1 of al,obj2 of al)
by the use of result returned by the above compare method (i.e)1,-1,0 how sorting will be

Search

Copyright © 2008 - 2014 programcreek.com