Paramesh Korrakuti

Home | Java | Spring | PL SQL | Hibernate | UNIX | JavaScript | Design Patterns

Wednesday, 8 October 2014

Examples of Comparable, Comparator, equals() & hashCode()

We can discuss the use of Comparable, Comparator interfaces & equals(), hashCode() methods by using some problem description.

In Java, there is an utility called Collections to perform most of the required operations on collection framework classes.
One of the most frequently used operation is Sorting. We can do the sorting by using Collections.sort() method.
Collections.sort() method has the following two forms.

sort(List list) - Sorts the specified list into ascending order, according to the natural ordering of its elements.
sort(List list, Comparator c) - Sorts the specified list according to the order induced by the specified comparator.

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?

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().
Objects which implement Comparable in java can be used as keys in a SortedMap like TreeMap or SortedSet like TreeSet without implementing any other interface.

Comparator:
Class whose objects to be sorted do not need to implement this interface. Some third class can implement this interface to sort.
Using Comparator interface, we can write different sorting based on different attributes of objects to be sorted.You can use anonymous comparator to compare at particular line of code.

Parameter	**Comparable**	**Comparator**
Sorting logic	Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects	Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id,name etc.
Implementation	Class whose objects to be sorted must implement this interface.e.g Country class needs to implement comparable to collection of country object by id	Class whose objects to be sorted do not need to implement this interface.Some other class can implement this interface. E.g.-CountrySortByIdComparator class can implement Comparator interface to sort collection of country object by id
Sorting method	int compareTo(Object o1) This method compares this object with o1 object and returns a integer.Its value has following meaning 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1	int compare(Object o1,Object o2) This method compares o1 and o2 objects. and returns a integer.Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1
Calling method	Collections.sort(List) Here objects will be sorted on the basis of CompareTo method	Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator
Package	Java.lang.Comparable	Java.util.Comparator

equals() & hashCode() methods: In Java every object has access to equals() & hashCode() methods, since these are available in java.lang.Object class & Object class is super class by default for all the classes. The main intention of these methods are to compare the given two or more objects and identify their equality.

The default implementation of `equals()` method in `Object` class simply checks if two object references x and y refer to the same object. i.e. It checks if `x == y`. This particular comparison is also known as "shallow comparison".

These two methods have significant relationship with each other. If we want to override any one of the methods, need to override both the methods. We will discuss why do we need to override both, otherwise what is the impact.

The `equals()` method must exhibit the following properties:

1. Symmetry: For two references, `a` and `b`, `a.equals(b)` if and only if `b.equals(a)`
2. Reflexivity: For all non-null references, `a.equals(a)`
3. Transitivity: If `a.equals(b)` and `b.equals(c)`, then `a.equals(c)`
4. Consistency with `hashCode()`: Two equal objects must have the same `hashCode()` value

Why do we need to override equals() method: An object might have many number of variables, but we may need to consider only few

Search Th

Total Pag

10,

About Me

P
View profil

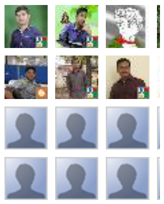
Blog Arch

▼ 2014 (5)
 ▼ October 2014
  Examples of Comparable Comparato equals() & h
 ► August 2014
 ► March 2014 (
 ► January 2014
► 2013 (11)

Followers

Join this site
with Google Friend Cor

Members (24) Mor



Already a member? Sig

Share It

Share this on Fac
Tweet this
View stats
(NEW) Appointment g

variables into consideration while comparing the objects.
Suppose, customer landed on some eCommerce application and would like buy some article. Article may have many number of properties such as name, capacity, cost, availability, seller_name, seller_location, delivery_date..etc. But while coming to comparison, customer may consider name, price & delivery_date.
Hence, it require to apply equals() methods to lookup only these properties while determining equality.

Why to override hashCode() method: hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable, HashMap like data structure. By default, Object's hashCode() method returns and integer representation of memory address where object is stored.
The hashCode() method of objects is used when we insert them into a HashTable, HashMap or HashSet. More about hastables on Wikipedia.org for reference.
To insert any entry in map data structure, we need both key and value. If both key and values are user define data types, the hashCode() of the key will be determine where to store the object internally. When require to lookup the object from the map also, the hash code of the key will be determine where to search for the object.
The hash code only points to a certain "area" (or list, bucket etc) internally. Since different key objects could potentially have the same hash code, the hash code itself is no guarantee that the right key is found. The HashTable then iterates this area (all keys with the same hash code) and uses the key's equals() method to find the right key. Once the right key is found, the object stored for that key is returned.
So, as we can see, a combination of the hashCode() and equals() methods are used when storing and when looking up objects in a HashTable.

NOTES:
1. Always use same attributes of an object to generate hashCode() and equals() both. As in our case, we have used employee id.
2. equals() must be *consistent* (if the objects are not modified, then it must keep returning the same value).
3. Whenever **a.equals(b)**, then *a.hashCode()* must be same as *b.hashCode()*.
4. If you override one, then you should override the other.

Problem Statement:
There are 2 travel companies who have buses travelling from dest A to dest B. they want to merge their buses timetable and produce it to the customer so that they can find out which bus is effective for them and choose bus accordingly(o/p).

They will provide a text file having the I/P as shown above and we need to give O/P as shown above.

Conditions:
1) If the bus travel time is greater than 1hr then it should not be displayed.
2) If the both the travel agency having same time of dept and arrival then we need to show only volvo travel time and ignore BMW.
3) we need to format O/P as
a) if busA starts at 10 and arrives at 11, if busB starts at 10:05 and arrives at 11, then busB has to come before busA.
b) if busA starts late and arrives before busB,then busA has to come before busB.
c) if busA starts at same time as busB but reaches before,then busA has to come before busB.

Input file:

```
volvo 10:10  11:05
volvo 10:15  11:05
volvo 11:00  12:15
volvo 11:50  12:50
volvo 12:00  12:45
BMW 10:05  11:05
BMW 10:10  11:05
BMW 10:50  11:45
volvo 10:50  11:45
BMW 11:05  12:15
```

Output:

```
volvo 10:15  11:05
volvo 10:10  11:05
BMW 10:05  11:05
volvo 10:50  11:45
volvo 12:00  12:45
volvo 11:50  12:50
```

BusInfoVO: Its a Data transfer object which holds the information of a bus.

```
package com.exer.files.vo;

import java.io.Serializable;
import java.util.Date;

public class BusInfoVO implements Comparable<BusInfoVO>, Serializable {

  /**
   *
   */
  private static final long serialVersionUID = 1L;

  private String busType;

  private Date startTime;

  private Date endTime;

  private transient Long duration;

  public String getBusType() {
    return busType;
```

Cheeky Qu

I don't believe in dining on an empty stomach.

W.C. Fields

w.c. fields quotes

```
    }

    public void setBusType(String busType) {
        this.busType = busType;
    }

    public Date getStartTime() {
        return startTime;
    }

    public void setStartTime(Date startTime) {
        this.startTime = startTime;
    }

    public Date getEndTime() {
        return endTime;
    }

    public void setEndTime(Date endTime) {
        this.endTime = endTime;
    }

    public Long getDuration() {
        return duration;
    }

    public void setDuration(Long duration) {
        this.duration = duration;
    }

    @Override
    public int compareTo(BusInfoVO busInfo) {

        if((this.getStartTime().equals(busInfo.getStartTime())) && (this.getEndTime().equals(busInfo.getEndTime()))) {
            return 0;
        } if((this.getStartTime().equals(busInfo.getStartTime())) || (this.getEndTime().equals(busInfo.getEndTime()))) {
            return this.getDuration() < busInfo.getDuration() ? -1 : 1;
        } else if((this.getStartTime().after(busInfo.getStartTime())) && (this.getEndTime().before(busInfo.getEndTime()))) {
            return -1;
        } else if(this.getStartTime().before(busInfo.getStartTime())) {
            return -1;
        }
        return 1;
    }

    @Override
    public boolean equals(Object busInfo) {
        boolean result = false;

        if (busInfo == null || (this.getClass() != busInfo.getClass())) {
            return result;
        }

        BusInfoVO other = (BusInfoVO) busInfo;
        if((this.getStartTime().equals(other.getStartTime())) && (this.getEndTime().equals(other.getEndTime()))) {
            return true;
        }

        return result;
    }

    @Override
    public int hashCode() {
        int hash = 10;
        hash = hash + this.getStartTime().getHours() + this.getStartTime().getMinutes();
        hash = hash + this.getEndTime().getHours() + this.getEndTime().getMinutes();
        return hash;
    }

}
```

Here BusInfoVO class implementing Comparable interface and overrides compareTo() method.

FilesReader.java: Where data can be read and processed as expected output.

```
package com.exe.files.reader;

import java.io.File;
import java.io.FileNotFoundException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Scanner;
import java.util.TreeSet;

import com.exer.files.vo.BusInfoVO;

public class FilesReader {

    private static DateFormat dateFormat = new SimpleDateFormat("HH:mm");

22.   private  List<BusInfoVO> busInfoVOList;
23.
24.   public  List<BusInfoVO> getBusInfoVOList(String path) {
25.     if(path != null) {
26.       File folder = new  File(path);
27.       File[] listOfFiles = folder.listFiles();
28.       busInfoVOList = new  ArrayList<BusInfoVO>();
29.       for  (int  i = 0; i < listOfFiles.length; i++) {
30.         try  (Scanner scanner = new  Scanner(listOfFiles[i])) {
31.           while  (scanner.hasNextLine()) {
32.             String[] info = scanner.nextLine().toString().split(" ");
33.             if  (info != null  && info.length > 0) {
34.               BusInfoVO vo = new  BusInfoVO();
35.               vo.setBusType(info[0]);
36.               Date startTime = dateFormat.parse(info[1]);
37.               Date endTime = dateFormat.parse(info[2]);
38.               vo.setStartTime(startTime);
39.               vo.setEndTime(endTime);
40.               vo.setDuration((endTime.getTime() - startTime.getTime()) / (60  * 1000)); // In minutes.
41.               if(vo.getDuration() <= 60) {
42.                 busInfoVOList.add(vo);
43.               }
44.             }
45.           }
46.         } catch(ParseException | FileNotFoundException exp) {
47.           exp.printStackTrace();
48.         }
49.       }
50.       Collections.sort(busInfoVOList, new  Comparator<BusInfoVO>() {
51.
52.         @Override
53.         public  int  compare(BusInfoVO o1, BusInfoVO o2) {
54.           return  o2.getBusType().compareTo(o1.getBusType());
55.         }
56.       });
57.       return  busInfoVOList;
58.     }
59.
60.     return  null;
61.   }
62.
63.   public  static  void  main(String[] args) throws  ParseException, FileNotFoundException {
64.     String path = "D:\\PARAMESH\\FILES";
65.     FilesReader reader = new  FilesReader();
66.     List<BusInfoVO> list = reader.getBusInfoVOList(path);
67.     TreeSet<BusInfoVO> set = new  TreeSet<BusInfoVO>();
68.     for(BusInfoVO vo : list) {
69.       set.add(vo);
70.     }
71.
72.     for(BusInfoVO vo : set) {
73.       System.out.println(vo.getBusType() + " "  + dateFormat.format(vo.getStartTime()) + " "  + dateFormat.format(vo.getEndTime()));
74.     }
75.   }
76. }
```

Here, we used(@ L 50) "sort(List<T> list, Comparator c)" method to sort the list based on the name field of BusInfoVO.

While adding to the set.add(vo) (@L 69), its looks for the equals() method definition available in BusInfoVO class and compares the given object with all existing objects in the set.
If equals returns : true - Will add the given object to the set.
                          false - Will not add the given object to the set.
Also, when we add the elements in TreeSet, it internally calls compareTo() method available in the given for sorting.

Posted by Paramesh K at 08:57 g+1 +1 Recommend this on Google

Labels: Java