♦ Sorting objects in Java interview Q&As

Posted on October 11, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. If I mention the interface names **Comparable** or **Comparator**, what does come to your mind? Why do we need these interfaces?

A1. **Sorting**. SortedSet and SortedMap interfaces maintain sorted order. The elements are sorted as you add or remove them. The other interfaces like List or Set don't sort elements as you add or remove. So you need to sort them on a as needed basis.

If you store objects of type String or Integer in a *List* or *Set*, and would like to occasionally sort them, say for reporting purpose, you can do so as shown below as String or Integer

by default implements the *Comparable* interface and provides a compareTo(..) method to be called while sorting.

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.List;
4
5  public class Sorting1 {
6
7      public static void main(String[ ] args) {
8          List<String> myShoppingList = new ArrayL
9          myShoppingList.add("Cereal");
10         myShoppingList.add("Apples");
11         myShoppingList.add("Soap");
12         myShoppingList.add("Brush");
13         System.out.println("Before sorting: " + 
14        // invokes compareTo method implemented i
15         Collections.sort(myShoppingList);
16         System.out.println("After sorting: " + m
17     }
18 }
```

Output:

Before sorting: [Cereal, Apples, Soap, Brush]
After sorting: [Apples, Brush, Cereal, Soap]

As you can see that the items are sorted lexicographically. This is the default implementation provided by the **compareTo(…)** method in the java.lang.String class. What if you have a special reporting requirement to sort by length of the item name. This is where the Comparator interface comes in handy by giving you more control over ordering. You can define your own ordering logic through the **compare(…)** method as shown below using the **Comparator** interface.

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.Comparator;
4  import java.util.List;
5
6  public class Sorting2 {
7
8      public static void main(String[ ] args) {
9          List<String> myShoppingList = new ArrayL
10         myShoppingList.add("Cereal");
11         myShoppingList.add("Apples");
12         myShoppingList.add("Soap");
13         myShoppingList.add("Brush");
14         myShoppingList.add(null);
15         System.out.println("Before sorting: " + 
16
```


16 Technical Key Areas

```
17          //Anonymous inner class.
18          Collections.sort(myShoppingList, new Com
19              @Override
20              public int compare(String o1, String
21                  if(o1 == null) {
22                      o1 = "";
23                  }
24                  if(o2 == null) {
25                      o2 = "";
26                  }
27                  return new Integer(o1.length( ))
28              }
29          });
30          System.out.println("After sorting: " + m
31      }
32  }
```

Output:

Before sorting: [Cereal, Apples, Soap, Brush]
After sorting: [Soap, Brush, Cereal, Apples]

Note: The above class is using an anonymous class to sort, but if you require to reuse the sorting in a number of places, you must move the compare(…) method to its own class as shown below.

```
1  import java.util.Comparator;
2
3  public class NameLengthComparator  implements Com
4
5          public int compare(String o1, String o2)
6                //implementation goes here. same as
7          }
8  }
```

You can use it as follows

```
1  Collections.sort(myShoppingList, new NameLengthCo
```

Comparable interface for natural ordering

Q2. What if your collection contains custom objects like a ***Pet*** class?
A2. You can provide the default sorting behavior by having the Pet class implement the ***Comparable*** interface and implementing the ***compareTo(…)*** method as shown below:

```
1  public class Pet implements Comparable<Pet> {
2
3      int id;
4      String name;
5
6      public Pet(int id, String name) {
7          this.id = id;
8          this.name = name;
9      }
10
11     // getters and setters go here
12
13    //invoked during sorting
14    public int compareTo(Pet o) {
15         Pet petAnother =  o;
16         // natural alphabetical ordering by name
17         return this.name.compareTo(petAnother.na
18     }
19
20     //invoked when the list is printed
21     public String toString( ) {
22         return "[id=" + id + ", name=" + name +
23     }
24 }
```

Take note of generics being used above. The above Pet class can be used as shown below.

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.List;
4
5  public class Sorting3 {
6
7      public static void main(String[ ] args) {
8          List<Pet> myPetList = new ArrayList<Pet>
9          myPetList.add(new Pet(1, "Dog"));
10         myPetList.add(new Pet(2,"Rabit"));
11         myPetList.add(new Pet(3,"Cat"));
12         myPetList.add(new Pet(2, "Hamster"));
13         System.out.println("Before sorting: " + 
14         //compareTo method  gets invoked on Pet.
15         Collections.sort(myPetList);
16         System.out.println("After sorting: " + m
17     }
18 }
```

Output:

Before sorting: [[1,Dog], [2,Rabit], [3,Cat], [2,Hamster]]
After sorting: [[3,Cat], [1,Dog], [2,Hamster], [2,Rabit]]

Q3. What if you have an additional special sorting requirement to sort first by id and then by name?

A3. You can use the ***Comparator*** interface to sort based on multiple attributes as shown below.

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.Comparator;
4  import java.util.List;
5
6  public class Sorting4 {
7
8      public static void main(String[ ] args) {
9          List<Pet> myPetList = new ArrayList<Pet>
10         myPetList.add(new Pet(1, "Dog"));
11         myPetList.add(new Pet(2, "Rabit"));
12         myPetList.add(new Pet(3, "Cat"));
13         myPetList.add(new Pet(2, "Hamster"));
14         System.out.println("Before sorting: " + 
15         Collections.sort(myPetList, new Comparat
16             @Override
17             public int compare(Pet o1, Pet o2) {
18                 int byIds = o1.getId( ).compareT
19                 // if ids are same, compare by n
20                 if (byIds == 0) {
21                     return o1.getName()
22                             .compareToIgnoreCase
23                 }
24                 return byIds;
25             }
26         });
27         System.out.println("After sorting: " + m
28     }
29 }
```

Output:

Before sorting: [[1,Dog], [2,Rabit], [3,Cat], [2,Hamster]]
After sorting: [[1,Dog], [2,Hamster], [2,Rabit], [3,Cat]]

Note: The above class is using an anonymous class to sort, but if you require to reuse the sorting in a number of places, you must move the *compare(…)* method to its own class.

Q4. What contract do you need to watch out for when writing your own comparator?
A4. As per the Java API for java.util.Comparator, caution should be exercised when using a comparator capable of imposing an ordering inconsistent with the equals(…) method.

```
1 if compare(o1,o2)  == 0    then   o1.equals(o2)
```

```
2 if compare(o1,o2)  != 0    then   o1.equals(o2)
```

Popular Posts

♦ 11 Spring boot interview questions & answers

823 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

765 views

18 Java scenarios based interview Questions and Answers

399 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It

pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ If Java did not have a stack or map, how would you ….

07: ♥ 20+ Pre interview refresher on productivity & debugging tools for Java developers ›

Posted in Collection and Data structures, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer