Uttara InfoSolutions

www.uttarainfo.com

Collections Practicals 2

First go through TestSet.java. Run it. Go through the code and understand the output. After that start doing the below problems. You should have also done the first practicals on Collections before you do this. When in doubt, go through javadoc of Collection (given as html). Then go through TestPersonSet.java.

1) Create two strings and invoke == and .equals() on them to verify working of object identity and object equality. Create the strings as literals as well as using the new operator. Invoke hashCode() on both the strings and print the output. Verify if 2 strings having same state.SOP s.hashCode() and s2.hashCode(). Do they have the save value? Create a HashSet object. Add to the set the two strings and print the add() return value. The second add should return false. SOP the set. Verify if you understand the steps HashSet takes when you add an element. You should know exactly why the second string that you add is not getting added. Instead of HashSet, create a TreeSet and a number of strings into it. SOP the set. Verify your understanding of how the TS works. Also add “ramanna”,”ramanuja”,”rameshwara”,”eshwara”,”someshwaraa”,”marirama” and then using foreach print all the strings that have “rama” in it.

2) There are vehicles. Vehicle has a name and an engine capacity [like 100bhp, 200bhp etc => take as int] and provide parameterised constructor. Create a Tester class, create 2 Vehicle objects with same state and invoke v1.equals(v2) and verify what you get as result? Why are you getting that result?
A vehicle is equal to another if both the name and capacity are the same. Override equals() method in Vehicle to implement this functionality. Put SOP in equals() with something like SOP(“Vehicle->equals()->testing…”) and concat the states of the two objects into sop. In a tester class, create two vehicle objects, invoke equals on it and find out whether object equality check is working correctly. Create an ArrayList object, add first vehicle ref and check if the contains(second ref) returns true or not => it should return true as equality check would be used to search.

3) Create a HashSet. Add vehicle objects to it. Create a new vehicle object with an earlier created name and engine capacity. Try to add this to the set. Remember this should not be allowed as it is a duplicate. If it is allowing, why so? invoke hashCode() method on both the objects and SOP the returned values. Are they the same? Now do you understand why set is allowing duplicates?

Override hashCode() method in Vehicle class and see if the same behaviour still persists in the tester class. Put SOPs in equals() and hashCode() so you understand the control flow.

4) Add 2 vehicle objects into a TreeSet in tester class. Run it. Do you get an exception? Why? Make the vehicle class implement Comparable and override compareTo() method to check engine capacity to decide which vehicle is greater/lesser.
Test this first in a tester class and see which vehicle is greater/lesser. Then add vehicle objects to a TreeSet. Iterate over the elements and print out the vehicle names. Verify if the sorting is happening correctly.

5) Design a Student class. A student has a name, id and age. Create the student class with overridden equals(), hashCode() and toString() and implement natural ordering by implementing Comparable to compare by age.
In tester class, create 5 student objects with different states and one more with same state as the first object. Add the 5 to an ArrayList, HashSet, LinkedHashSet, TreeSet (one at a time) and then iterate over the collection and print the results (foreach). Are you now able to understand how the collection implementations work differently even though same methods are exposed?

6) Create a StringLengthComparator (implements Comparator interface) class. Create a TestTree tester class and create 10 strings with varying state and length. Now add them to the TreeSet and SOP the set. Can you see the output is sorted? What is it sorted based on? Now create StringLengthComparator class object and pass this reference to the TreeSet constructor. Now how are the strings getting sorted? Sort the same strings in a List using Collections.sort(list ref) as well.

7) Now to the 5th problem, create a StudentNameComparator & StudentAgeDescending-Comparator and sort the students. Sort the students both in a List as well as in a Set.

8) Take a sentence input (hardcode in main()) and sort the words in the sentence and display the output

9) Take a sentence input (hardcode in main()) and display all the unique words in the sentence without duplicates in the same order.

10) Take a sentence input (hardcode in main()) and display non-repeating words only. For ex: this this is is a what how what is => a how should be output only.