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).

1) Create two strings with same state using new operator 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 and print the result. Are the hash codes same? Why?

2) There are vehicles. Vehicle has a name and an engine capacity [like 100bhp, 200bhp etc => take as int]. Generate setter/getter methods and a parameterised constructor. A vehicle is equal to another if both the name and capacity are the same. Without overriding equals(), verify in TestVehicle->main() by creating 2 Vehicle objects with same state whether object equality check works.
Then override equals() method in Vehicle to implement this functionality (do not use auto generate but code the method yourself). 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. Verify if the SOP in equals() is being called or not.

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.
In tester class, create 5 student objects with different state 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. Are you now able to understand how the collection implementations work differently even though same methods are exposed?

6) There are Songs. Every Song has a name, length, singer. Create Song class with all best practices. Implement natural ordering to compare songs by length. Test by adding Song object references into TreeSet and then in a List with duplicates and sort them using Collections.sort(li). Go through TestSongSorting.java for this.

<u>Other Problems:</u>

1) Test how to use a HashSet and TreeSet with strings
2) Test how to use a HashSet and TreeSet with Person (has a name and age)
3) Test how to sort Strings based on length
4) Test how to sort Person based on age
5) Test how to sort List of strings
6) Test how to sort List of Persons
7) Test how to sort List of strings, based on length
8) Test how to sort List of Persons, based on age
9) Test how to shuffle a List, how to check frequency
of a repeating element
10) Test Maps

11) Take a sentence and a word as input from the user and
a) print how many occurrences you find of an input word in the sentence
b) sort the sentence i) with duplicates ii) without duplicates and print
c) sort the sentence using string length comparison
d) remove all the occurrences of the word from the sentence

Go through TestIntvPgms.java for this.

12) Given a sentence as input, find the number of occurrences of every word in it and print it out (using Maps).