CodeQuest 2016 Solution Notes – Robert Engle

# Notes for all problems:

All problems use the BufferedReader class, and each start off with an input of the amount of test cases. These simply require you to loop the amount of times that a test case will appear. I won’t go over how to loop through these since I am more interested in how I will get the actual problem solved.

# **Problem 1:**

This problem is a rather simple one, as you would only need to have a knowledge of how nested for-loops work. This would require a for-loop inside of one, each iterating N times. For each iteration, a “# “would be printed out. I wrote a method that simply prints this out, taking the N \* N integer as an argument.

# **Problem 2:**

This problem is tricky because people may immediately treat these as double values, not taking into consideration round-off error. I originally made this mistake in Prob02, making Prob02\_v2 to correct it. The better option would be to treat this amount as pennies, and sequentially subtracting off 25, 10, 5, and 1(s) pennies at a time, counting the amount of quarters, dimes, nickels, and pennies it has taken.4

# **Problem 3:**

To solve this problem, you must first check that if the three sides of the triangle work with the triangle inequality theorem. To do this, I just added up two sides, and made sure they were greater than the third side. I did this three times to represent all three sides. Once that was done, it is rather straightforward. First I checked for all equal sides(equilateral), then checked for the condition that none of the sides were equal(scalene). Finally, the only other option would be isosceles.

# **Problem 4:**

My first solution to this problem was very complicated in the sense that I would have a complicated way to find out which letters were used. My second version of the problem uses a much simpler method: break up the letters into a character array, and sort the array. Do these for both words. Then, you could sort out both of these character arrays, and if there are any mismatches, then you immediately know that the word is not an anagram.

# **Problem 5:**

My first solution to this problem was also really complicated to follow, having a rounding method which may be incorrect. This problem is simple in terms of how you get your numbers: multiply the bill by the tip percentage(decimal form). The only problem is that wants the answer to decimal places. This can be done using a DecimalFormat object, setting the rounding mode to RoundingMode.HALF\_UP. Then simply take the product of each multiplicated and pass it through DecimalFormat.format().

# **Problem 6:**

The way to solve this problem can be seen different ways. One way would be to grab data for all students, and then find out which one is the best in the school. My first solution involves a bunch of arrays, while my second solution keeps tracks of students much better in the Student object. I took all of the students and put them into an array of Student objects, but there is a better way to solve this problem (which I have not done): using the Student class, have a best student and a student being compared loaded into the memory, and make comparisons like that. This solution offers a lot more flexibility in lower-memory machines and/or high-memory scenarios.

# **Problem 7:**

# **Problem 8:**

For solving this problem, it is rather easy to collect the data the input file gives. I then made a Song class, which contains fields for the name of the song and the name of the artist who made that song. A method called effectiveAuthor is used to remove the first “the” from the artist. After all the data is read, they must now be sorted. This can be done by using a comparator:

|  |
| --- |
| //comparator for sorting the array of song objects  public static class SongComparator implements Comparator<Song>{  @Override  public int compare(Song a, Song b){  //To prevent authors with all capitals in their name from being sorted impropely, we take the effective authors, split them up, take their first word,   //and make sure only the first letter is capitalized. We ignore the rest of the words for purposes of sorting   String aEff = a.effectiveAuthor();  String bEff = b.effectiveAuthor();  String[] aEffSplit = aEff.split(" ");  String[] bEffSplit = bEff.split(" ");  String aFirst = aEffSplit[0].substring(0, 1) + aEffSplit[0].substring(1).toLowerCase();  String bFirst = bEffSplit[0].substring(0, 1) + bEffSplit[0].substring(1).toLowerCase();  //First compare the authors to each other  //Implement CASE\_INSENITIVE\_ORDER later  int rslt = aFirst.compareTo(bFirst);  //if the authors are the same, the value of rslt will be 0. we can then compare songs   if(rslt == 0){  rslt = a.name.compareTo(b.name);  }  //Return result   return rslt;  } } |