Java Lab 18: Queues, Enum, RegEx

This lab uses queues and exception handling. Create a project named Lab18. Download ***Customer.java***, ***RatingType.java***, ***Lab18.java***, and the data files ***customers.csv*** and ***testdata.txt*** (used only in #9). Each line of the csv file represents one Customer object - but note that the member data field ratingType must be assigned – see #1.

1. RatingType.java defines an ***`enum`*** for ratings. In the Customer constructor, set ratingType by evaluating the rating field. Assign the ratingType field as follows: if 0 <= rating <=3 is LOW; 4 <= rating <= 7 is MEDIUM; 8 <= rating <= 10 is HIGH. Create a getter for ratingType, but return its String equivalent, not the enum. Then add the ratingType to the existing toString( ) method.

Customer implements Comparable<Customer>; code compareTo() using the rating field.

2. Lab18 contains a main method (described below) and a method with this signature:

`public static ArrayList<Customer> readCustomers(String filename) throws IOException`

It should first create a File object using the filename parameter, then should see if file.exists( ) is false – if so, throw a new FileNotFoundException with the message filename + " not found". Otherwise, it should create a Scanner object using file – but don't put that in try-catch block, because it shouldn't fail (you already checked for that), and even if it did, the method throws its exception. Read the CSV data for Customers (one customer per line). Convert the int and double data by parsing it, as usual, but within separate try-catch blocks. If an exception is thrown, display an error message about which one was bad, but zero out the data and continue. Create a Customer object and add it to the array list. At the end, return the ArrayList of Customer objects.

The next few problems use main( ).

3. Call readCustomers with customers.csv as the parameter; the return value should be an ArrayList of Customer objects. If the exception is thrown, print the exception object's message and end the program. Then print out the list with the label "Original data".

4. Create a PriorityQueue queue1 and fill it by iterating over the ArrayList and inserting them using add( ). Don't forget the try-catch block.

5. Create another PriorityQueue queue2 using a Comparator that compares based on Customer's balance field. Insert the customers from the ArrayList using add( ). Don't forget the try-catch block.

6. Print the data from queue1 with the label “Queue 1 processing” - write a while(true) loop over queue1 using element( ) and remove( ); as you remove the customer object, display it with the label "Queue1 processing". Be sure to enclose the loop inside a try-catch block; if an error is caught (it will be), print "Done". Check the output to make sure the Customer objects were returned from smallest to largest rating.

7. Do the same thing for queue2 as #6 with the label "Queue2 processing". Check the output to make sure the Customer objects were returned from smallest to largest balance.

8. Add a new line of data to customers.csv that causes the Integer parsing to fail and verify that. Add a new line of data to customers.csv that causes the Double parsing to fail and verify that. Change the file name in main( ) to “bob.csv” to verify the FileNotFound exception.

9. This problem has nothing to do with the Customer queue material. Write ***`public static void problem9()`*** that does the following. Read the file testdata.txt into ArrayList<String> lines, one line per entry. Display it.

Write patterns for each description below and insert them into ArrayList<String> patterns. Loop over the patterns – print the pattern – then inside that loop, loop over the lines in test data; display the lines of test data that match each pattern. For example, if the first pattern is “[do]” and the first line is “dog”, the first part of the display will be

```
Pattern: [do]
```

```
Matched: dog
```

and then any other lines that pattern matches; repeat with the next pattern over all the lines.

- lines containing any digit
- lines containing any letter, either case
- lines containing an integer
- any line that begins with the letter "a"
- any line that ends with the letter "s"
- any line that contains a left parenthesis
- lines that contain an "a" and an "e" in either order
- lines containing the lower case vowels next to each other, in order (a, e, i, o, u)
- lines containing the lower case vowels in order, but not necessarily next to each other