ESS 201 – Programming II (Java)

Term I, 2018-19

Lab 3

14 Aug 2018

A bank provides a Teller counter where customers present requests - deposit or withdraw money or query for their balance.

The Teller does not directly execute these transactions. She has one or more cashiers available and passes on the request to them for processing. She can adjust the number of cashiers deployed at any given based on the load - the number of requests to be processed. Different requests take different amounts of time for processing.

To provide good turnaround time for the customers, the bank uses the following process:

1. Customers line up at the Teller counter and submit their requests. For each request, a customer provides the customer id (assume each customer has only one account at that this bank) and one of:
    a. Deposit: an amount
    b. Withdraw an amount
    c. Query for the balance

    The Teller assigns a new unique id to each request: consecutive integers starting at 1.

2. The Teller collects these requests in the order they are received. When there is a break in the customer queue (i.e. no waiting customers), she "activates" a certain number of cashiers, and distributes the requests to them in *round-robin* fashion: one to each of the cashiers in order, and back to the first cashier and so on until all the available requests are distributed
    a. The number of cashiers activated depends on the number of outstanding requests. For now, we can assume that no cashier will be given more than 3 requests. Obviously, we would like the minimum number of cashiers possible while satisfying this requirement.
    b. Once the outstanding requests are processed, the cashiers go away, and a new set of cashiers are activated for the next set of requests.

3. The Teller then asks each cashier in turn to process the requests given to them. Cashiers process their requests in the reverse order of how they received them - i.e. in *last-in-first-out order*. When they are done with a request, they print out the id of the request and the time at which they have completed that task. Also, the cashier prints out the total value of transactions he has processed (total deposits - total withdrawals) in this activation.

4. Different tasks take different amounts of time:

Deposit: 5 mins
Withdrawal: 10 mins
Query Balance: 2 mins
You can assume that the time for processing all other tasks (collecting requests, printing, passing on requests to cashiers etc) take 0 mins.

5. Once the outstanding requests are processed, the Teller turns back to the customers and processes the next set of waiting customers.

Model this as a Java program with a set of appropriately defined classes. Note that customers (and main) should only interact with the Teller, and are, in fact, aware of cashiers.

Thus, you should have a class that contains the main, and this should do the following:

1. Create a Teller object
2. Read in customer requests (format described below) and pass on the requests to the Teller object. Inform the Teller when there is break in the queue.
3. All other processing should be done by the Teller class or passed on to Cashier class and any other helper classes you need.

Clearly, you will need to modularize these classes into packages, and appropriately set access specifiers for each of the classes and their methods.

To simulate this, we use an input text file with the following:
A series of lines with one of:

- A **request** which contains the following: the character D (for deposit), W (for withdraw) or Q (for query), followed by the customer id (an integer), followed by an amount (integer) if it is D or W
- A **break** in the queue, indicated by the letter B
- **End** of input, indicated by the letter E. (Note E implies a break as well as end of input)

Sample Input

```
D 1 1000
D 2 2000
Q 1
W 2 1000
B
D 3 3000
W 3 1000
E
```

Output:

```
Cashier 1
Request 1 5
Request 3 7
Total 1000
Cashier 2
Request 2 12
Request 4 22
Total 1000
Cashier 1
Request 5 27
Request 6 37
Total 2000
```

You can use the above as test data for the initial test of the program (also available as lab3input1.txt). . The submitted program will be tested against additional input files.