**Lab 8 – A Priority Customer Service Queue for the Bank Simulation**

**MAKE SURE YOUR SIMULATION FROM LAB5B IS COMPLETE BEFORE WORKING ON THIS LAB.**

Have you ever wondered about that ubiquitous phrase, *“Your call is important to us; your call will be answered in the order it was received*”? They say that all customers are equal but do they actually mean it? Do you suspect that some customers might be a little “*more equal than others*”? Let's explore how data structures might help in achieving this kind of evil!

Suppose a company wants to serve customers that come to them, but they want to rank customers in the order of their importance and serve the more important customers before the less important ones. To do so, the company assigns a priority rank to each customer, with lower integers indicating higher ranks. Then, if there are several customers waiting for service, the company picks the one with the lowest priority value to serve next. (If two or more customers have the same low priority, choose the one that arrived first.)

You are to write a simulation of this situation, based on the bank simulation from **lab05b**. ***Copy*** all the necessary files from your **lab05b** folder into your folder for this lab (so you still have a working copy of the original bank simulation).

### *Problem 1: Modifying the Customer class* Make the following changes to the *Customer* class:

### Have it implement the *Comparable<Customer>* interface.

### Add a *private* field *priority* which is an *int* holding the customer’s priority value.

### Provide an accessor (*getPriority*) method for the field.

### Implement the *compareTo* method so that it compares the *priority* fields and (if the priorities are equal) the *arrivalTime* fields of the two *Customer* objects.

### The constructor should take an *int* parameter to set the *priority* field along with the parameter to set the *arrivalTime* field.

### This class shouldn't need any other new methods unless you choose to override the *toString* method.

***Problem******2: Testing the modified Customer class***

Use the provided *PriorityCustomerTest* JUnit test class to test the methods of your *Customer* class. This class includes five *Customer* objects, an init method where the five objects are created with some objects having different priorities and some with the same priority but different arrival times, a testInitialState method where all the accessor methods are tested and three testCompareToReturns*[…]* methods where each object is compared to all the other objects including itself.

***Problem******3: Modifying the bank simulation***Modify the bank simulation program so that it uses a priority queue of *Customer*s instead of a regular queue. Use Java’s *PriorityQueue* class (online: <http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html>) in place of the regular queue in the simulation program. Please read the remainder of this section before you start working on this part.

The simulation should now take another input from the user representing the largest possible priority value as an integer. When a new customer object is created, its priority value should be a randomly generated integer between 0 and the largest possible priority value specified as input to the simulation. Please note that low priority values indicate higher priority customers which are to be served before customers with high priority values. This works well when using Java’s *PriorityQueue* class because it sorts items by their natural ordering with lower values being serviced before larger values.

When you display simulation results, you need to display customers with their ids, arrival times, and priority values and whether they were served or put in the queue. In addition, display the priority value of customers left in the queue at the end of the simulation.Below are a couple of input/output scenarios (with all details of the simulation printed) for you to mimic closely. Notice how customers with lower priority values were served before ones with higher priorities values. Also notice how customers tied for priority values were served in order of arrival time.

Try to create scenarios in which all customers are served promptly, even when they have a wide range of priorities. Try to create other scenarios in which some low-priority customers wait an unacceptably long time for service or perhaps are never served. Vary the number of priority levels, the arrival rate and the number of tellers to get different scenarios. *For the most part, you are on your own here in terms of how to make the required changes.*

|  |
| --- |
| **SCENARIO I**        **SCENARIO II** |