Lab 12 – Queue Client

Goal

In this lab you will work with queues. You will use an event queue along with another queue in a simulation of customers waiting in a line at a bank.

Resources

- Chapter 10: Queues, Deques, and Priority Queues
- Chapter 11: Queue, Deque, and Priority Queue Implementations
- `VectorQueue.java`—A sample implementation of `Queue` (in `QueuePackage`)
- `Bank.jar`—The final animated application
- Lab Manual Appendix —An Animation Framework

In `javadoc` directory

- `QueueInterface.html`—API documentation for the queue ADT
- `PriorityQueueInterface.html`—API documentation for the priority queue ADT
- `SimulationEventQueueInterface.html`—API documentation for the event queue ADT
- `SimulationEventInterface.html`—API documentation for the events on the event queue ADT
- `BankLine.html`—API documentation for a class representing a line of customers in a bank
- `Customer.html`—API documentation for a class representing a customer in a waiting line
- `Report.html`—API documentation for a class representing a class that will display a report for the simulation

Java Files

- `BankActionThread.java`
- `BankApplication.java`
- `BankLine.java`
- `Customer.java`
- `CustomerGenerator.java`
- `Report.java`
- `Teller.java`

There are other files used to animate the application. For a full description, see Appendix: An Animation Framework of this manual.

In `QueuePackage` directory

- `EmptyQueueException.java`
- `PriorityQueueInterface.java`
- `QueueInterface.java`
- `SimulationEvent.java`
- `SimulationEventInterface.java`
- `SimulationEventQueueInterface.java`
- `SimulationEventQueue.java`
- `VectorQueue.java`

The Bank Line Animation

Checkpoint: The bank application should run by compiling ***javac *.java*** *and running* ***java BankApplication****. At the very start of the set up phase, init() will be called. The customer generator in its constructor puts its initial* *event on the event queue. That should show up as the next event. No customers should be in the line. The bank teller Fred should be waiting patiently for customers to show up. The report should indicate that there are no customers waiting or served. The simulation time is 0.0.*

At this point, it would be nice to see the event queue in operation. Code to drive the simulation will be added into the BankActionThread.

The Event Loop

The method `executeApplication()` in `BankActionThread` has code that will repeatedly take events from the event queue and process them. Refer to the pre-lab presentation. Inside the loop after the event has been processed, the post action report is received from the event is used to set `lastEventReport`. If there is a next event, the description is used to set `nextEventAction`. The time is updated for the report and the last code in the loop should pauses the animation by calling animationPause();

Checkpoint: Compile and run the application. Press go. Customers should be generated and placed one at a time into the line. You should see them. Unfortunately, Fred is busy drinking coffee and is not yet helping the customers. The simulation should stop once it hits 1000.

Completing the Teller Event

It is time for Fred to get to work. The process method for CheckForCustomerEvent inside the Teller class needs to be completed. If no customer was served (theLine queue is empty), `serving` is set to `null`. At the end of processing, `postActionReport` is set to a string describing the actions taken by the event. The constructor for `Teller` generates the first `CheckForCustomerEvent`.

Step 1. Refer to the pre-lab presentation and the comments inside the process() method in Teller.java and complete the code for the method `process()`.

Checkpoint: The teller should now take customers from the line. As customers are serviced, the report should change. Step and carefully trace the operation of the simulation. Verify that it is operating correctly.

Change the service interval time and verify that customers are handled quicker.

Post-Lab Follow-Ups

1. Implement a new version of `SimulationEventQueue`, which uses a linked list.
2. Add two private queues to the `CustomerGenerator` that will store name syllables. When generating a name, take one syllable from each queue and concatenate them together. Put the syllables back on the ends of their respective queues. Set the queues so that each has a length that is a different prime number.
3. Modify the simulation so that it has two tellers that take customers from a single line.
4. Modify the simulation so that it has two tellers, each with their own separate line.

5. One of the good things about an event-driven simulation is that it will jump over times where nothing is happening. In the animation, this can cause sudden jumps in time. Add a class that will generate dummy events every second. The only responsibility of the event is to schedule the next dummy event.

6. *For those familiar with statistics and calculus*: Change the `CustomerGenerator` class so that the time between customers is determined according to a Gaussian distribution with a given mean and standard deviation. Make a similar change for the service time for the `Teller` class. There are other distributions you can try as well. (Hint: To do this, consult a book on statistics and find a table that gives the cumulative distribution for a Gaussian distribution. Generate a value between 0 and 1 and interpolate to find a z score. From this, use the mean and standard deviation for the desired distribution to find your value.)

7. Change the `BankLine` to be a priority queue. Each customer generated will have one of two priorities, high or low. Have the `Report` class report the average waiting time for each priority level.