Checkout Simulator - A2

Due: Oct 20, 2022 at 11:59PM

1 Specification

1.1 Assignment Link

The skeleton code for the assignment is available at `https://classroom.github.com/a/KGOlj561`.

1.2 Simulators

For most of the history of electronic computers, they have been used to simulate real-world events in a fast and controlled setting. A simulation requires a model, which is a set of entities and events that change those entities. Simulations can run events much faster than the events run in the real world (for example, galaxy formation or plate tectonics).

In this assignment, you will describe the entities and events in a more modest simulation: a group of checkout lines at a grocery store. The simulation has several parameters to allow for experiments in changing the model to see what effect those changes had. There will be two different checkout policies to implement: a single queue and multi-queue approach.

A simulation should run with the following rules. See the code for more specific details on the entities involved.

- All entities should be updated each time step
- When a customer is being served, assume one item from their cart is scanned in that time step
- When a customer arrives, add separate copies of that customer with the same number of items to both simulations
- When a customer has no items left to scan, they are finished checking out and another customer should immediately start checking out (if there is another waiting customer)

Example input/output from two runs:

```
Enter number of lines (2 or greater): 3
Enter max number of items (2 or greater): 40
Enter number of customers to queue (2 or greater): 10
Enter % chance of a customer getting in a line (1-100): 80

Statistics for single line:
Average wait time: 450.000000
Maximum queue length: 7
Customers per hour: 38.709677
Fairness: 209.443708

Statistics for multi line:
Average wait time: 457.000000
Maximum queue length: 4
Customers per hour: 36.363636
Fairness: 235.845995



Enter number of lines (2 or greater): 2
Enter max number of items (2 or greater): 15
```

```
Enter number of customers to queue (2 or greater): 100
Enter % chance of a customer getting in a line (1-100): 50

Statistics for single line:
Average wait time: 917.500000
Maximum queue length: 45
Customers per hour: 93.264249
Fairness: 531.657167

Statistics for multi line:
Average wait time: 932.100000
Maximum queue length: 24
Customers per hour: 93.506494
Fairness: 533.307376
```

Your numbers won't agree exactly because of some random events, but your simulator should produce similar results with each run.

2 Implementation

The first goal of this assignment is to practice writing Java code that uses OOP design ideas, and to use Queue operations in order to simulate a grocery store checkout area.

Rather than labs, where we define the functionality of a data structure, we will use the built-in Java data structures to help solve a larger problems for this assignment.

Java's built-in queue interface is called a `Deque`. The textbook has details on how it can be used as a queue. Luckily the `LinkedList` class implements a `Deque`, so you can create a `LinkedList` and have the queue operations available. Use it in all of the places that require a queue.

For each of the classes, consider how you can write and test them incrementally. It will probably be helpful to write debugging print statements that mark events in code. Delete or comment out these extra print statements before turning in your work.

Write and test small sections separately. As long as you follow the model guidelines in this document, you have flexibility with how to implement those guidelines.

As you read through these class descriptions, look through the corresponding Java file at the same time. There is necessary information in the comments and code that is not duplicated here.

2.1 SingleLine Behavior

A single line policy is quite simple: all customers get into the same queue if no checkout is open. If a checkout opens, a customer moves from the queue to that checkout location.

A customer's wait time should start when they arrive, but their service should only start when they get into a checkout. Early customers won't have to wait in the queue because they can proceed straight to a checkout, but later customers will.

During a time tick, you should update all customers who are in checkouts and all customers in the queue, as well as removing customers from the checkout and moving customers from the queue to checkout at appropriate times. In addition, update all relevant statistics for all entities and consider adding a customer.

Check both the checkouts and queue to determine if there are still customers in the system.

2.2 MultiLine Behavior

A multi-line policy is more common at US grocery stores: each checkout has an independent queue. The class has a list of queues – the list length should be built with the number of lines requested, and the list length should not vary. The length of individual queues certainly should vary.

In this policy, a customer always picks one of the shortest queues when they arrive. Customers start getting served when they reach the front of the queue, but they are not removed from the queue until they are done. This means that you should treat the front of the queues as checkouts, rather than `SingleLine`'s separate queue and checkouts.

During a time tick, you should update all customers in all queues, as well as removing done customers from the front of queues. In addition, update all relevant statistics.

Check all queues to determine if there are customers in the system.

2.3 Simulator Behavior

The simulator should run in an update loop as long as there are customers who have not yet arrived or customers still in one of the systems.

Each time through the loop you should update all the parts of the simulations:

- Decide whether a new customer has arrived, and add a copy of them to both simulations
- If there are customers in a simulation, run another step of that simulation

To decide if a customer has arrived in the current time step, you can use the `Math.random` library method to generate a random number between 0 and 1. If that number is smaller than the % chance and there are still customers who have not arrived, that signifies an arrival. You can use the same method to generate a random amount of items for a new customer before they are added to the simulations. The minimum number of items a customer might have is 1.

2.4 Statistics

We want to measure two important properties of lines: average wait time and fairness. It's easy to calculate average wait time if we keep track of the total wait time and total number of customers. To measure fairness, we can use the standard deviation measure of wait times. This measures how much individual wait times deviate from the average wait time, and a smaller number is better. (In a perfectly fair system, everyone would have to wait the same amount of time.)

Here are definitions of the different statistics that you should collect and calculate:

- Wait time: Time customer has waited since arriving
- Queue length: Maximum length of any queue
- Fairness: Standard deviation of wait times

To calculate the standard deviation (σ):

μ = avgWait

N = number of customers

x_i = the ith customer

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

You can only calculate the standard deviation after the simulation has completed, because you need to know the average wait time. You also need to know all of the individual wait times, which is what the `waitList` variable should be used for.

2.5 Class Implementations

There are `// TODO` markings throughout the different classes that are your responsibility to complete. Methods currently return false if they return anything, but you should replace those return values for proper functionality.

2.6 Results

The second goal of the assignment is to practice analyzing simulation results. Once the simulator is working, it will give information about specific behaviors of the two queue styles. Does one work better than the other? Are they equivalent?

Gather some data from your simulation in order to analyze it in Excel. Test your simulator ten times with the following parameters:

- 4 lines
- 30 items maximum

- 10..100 customers, in steps of 10
- 60% chance of a customer arriving

In Excel, your goal is to show the relationship between total people and average wait time, for both queue types. Create a table in Excel with a column of wait time for the single queue, and a column for the multi-queue. Insert a line chart that compares total number of people to the average wait time, with two lines for the two queue types.

What conclusions can you draw from the chart? Write two or three sentences in Excel (near the chart) with observations about the effect of number of customers, queue type, and any other interesting trends you notice.

Save your .xlsx file as `results.xlsx` in the root directory of your project (in the same directory as `src`).

3 Double Check

- Have you implemented all the `TODO` methods?
- Have you tested the simulator with a range of inputs?
- Have you added your `results.xlsx` file to the git project?
- Have you committed/pushed your code from **all** the files you modified?

3.1 Available Resources

- Lecture slides
- Other sections of the provided code
- me: kreimendahlf@wit.edu or discord. (If you have a general question, feel free to post in #class-chat)
- The textbook
- Java documentation for Deque/LinkedList on the Oracle website

3.2 Assignment Instructions

- This is an *individual* assignment.
- Make sure to read through all of the specifications so your submission is complete.
- Follow all the submission steps in the Setup document by the assignment deadline.

4 Grading

MultiLine working: +30%

SingleLine working: +30%

Simulator working: +25%

`results.xlsx`: +15%