

CptS 122 Assignment #6

Earlier in the semester, we helped Cougar Mart develop a simple checkout system. That system was so successful that they've decided to expand their operations by opening additional checkout lanes. Unfortunately, they are unsure of just how many lanes they need to add. Ideally, Cougar Mart would add enough lanes to satisfy expected demand. However, opening too many lanes would mean that they're paying checkers to staff empty lanes. Your goal is to develop a checkout lane simulation that can be used to determine the optimal number of lanes that Cougar Mart should have open.

Your program should simulate an entire day at Cougar Mart. For this simulation, we will assume that Cougar Mart is open 12 hours (720 minutes) a day. As this is a simulation, you don't actually need to run your program for 12 hours. Instead, consider the time span to be an integer, with each minute being a number. As such, 0 would indicate the start of the simulation, 1 would represent the first minute of the simulation, 2 the second minute, and so on.

Program Flow

Your simulation should operate as follows:

1. Prompt the user for the number of checkstands to simulate
2. Create a data structure that contains the appropriate number of checkstands
3. Set the probability of a new customer arriving equal to 0.
4. For each minute in the simulation
 - a. Determine if a new customer has arrived.
 - i. Generate a random number between 0 and 5.
 - ii. If the random number is less than or equal to the probability of a new customer arriving (Step #3), reset the probability back to 0 and go to Step 4.B.
 - iii. If the number is greater than the probability of a new customer arriving (Step #3), increment the probability of the new customer arriving by 1. Go to Step 4.C.
 - b. If a new customer has arrived, create a new customer whose arrival time is equal to the current simulation time and whose service time is a randomly generated number between 1 and 4. Next, try to find an empty checkstand. If one exists, place the customer in the line at that checkstand. If no empty checkstand exists, randomly select a checkstand for the customer to enter.
 - c. For each checkstand in the simulation:
 - i. If there is at least one customer in the checkstand, go to 4.C.II. Otherwise, go back to 4.C (look at the next checkstand in the simulation).
 - ii. If the first customer in line has not received a departure time, calculate one. $\text{Departure time} = \text{current tick} + \text{service time}$.
 - iii. If the total time elapsed is greater than the first customer's departure time, then the customer has completed the checkout process. Remove them from the queue.
 - d. Increment the elapsed time in the simulation

5. After the simulation completes, write the results to a CSV file in the following format: <Customer ID>, <Arrival Time>, <Time to Check Out>, <Departure Time>, <Total Wait>, <Line Number>.

Also include the total number of customers generated during the simulation at the top of the CSV file. Note that an example CSV file will be included with this assignment description. I have provided a CsvWriter class to aid in the construction of CSV files.

Required Bits

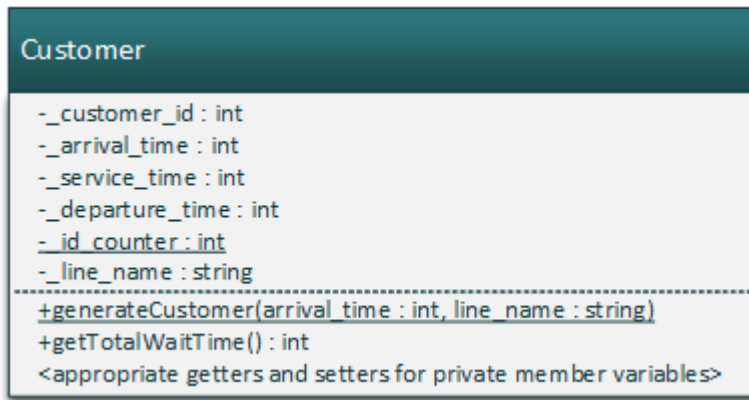
I'm going to leave more of the design decisions up to you, however, at minimum, your program must use a vector or linked list to store the collection of checkstands. Additionally, each individual checkstand must represent its line of customers using a Queue. You may either use the data structures developed in class or those provided to you by the STL.

Optional Bits

For those wanting a bit more direction, my simulation contains two additional classes:

Customer

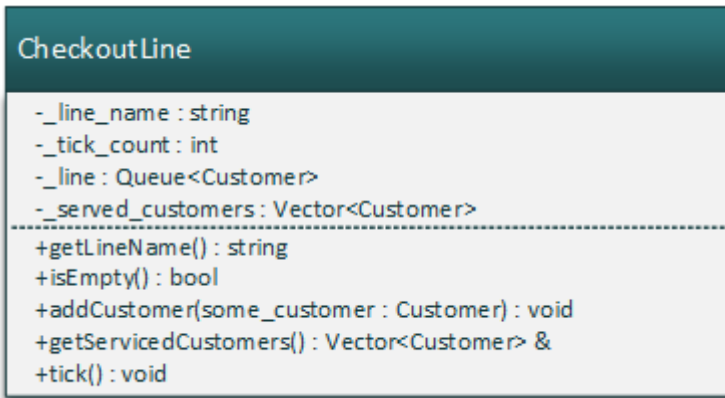
I use the customer class to contain details about individual customers. Here's the UML diagram:



As you can see, I have private member variables to track all of the customer's information. Note that I have a static integer called "`_id_counter`" that ensure that each customer receives a unique customer ID number. Each time that I create a new customer, I increment that number. Related to this is the static method `generateCustomer()`. I use this method to create all my customers, sort of like you would when using a Factory. I also have a method that computes the total wait time, which is simply the arrival time subtracted from the departure time. Finally, I have all the typical getters and setters for my member variables that you would expect.

CheckoutLine

I use CheckoutLine to represent a single checkstand in my simulation. Here's the UML diagram:



Note that each CheckoutLine has a line name. This is so that we can determine what lines serviced which customers when examining the final CSV output file. Also note that each CheckoutLine keeps track of the total minutes, which I call "ticks" (a common term in computer science). Each time the tick() method is called, I increment the _ticks variable. Also, the tick() method is responsible for performing checkstand-specific tasks listed in section 4.C of the simulation algorithm. As part of that method, I move the customers that have finished the checkout process from "_line" into "_served_customers", which I use when creating my CSV file. Here's a breakdown of the other methods in this class:

isEmpty

Returns true if our "_line" queue is empty

addCustomer

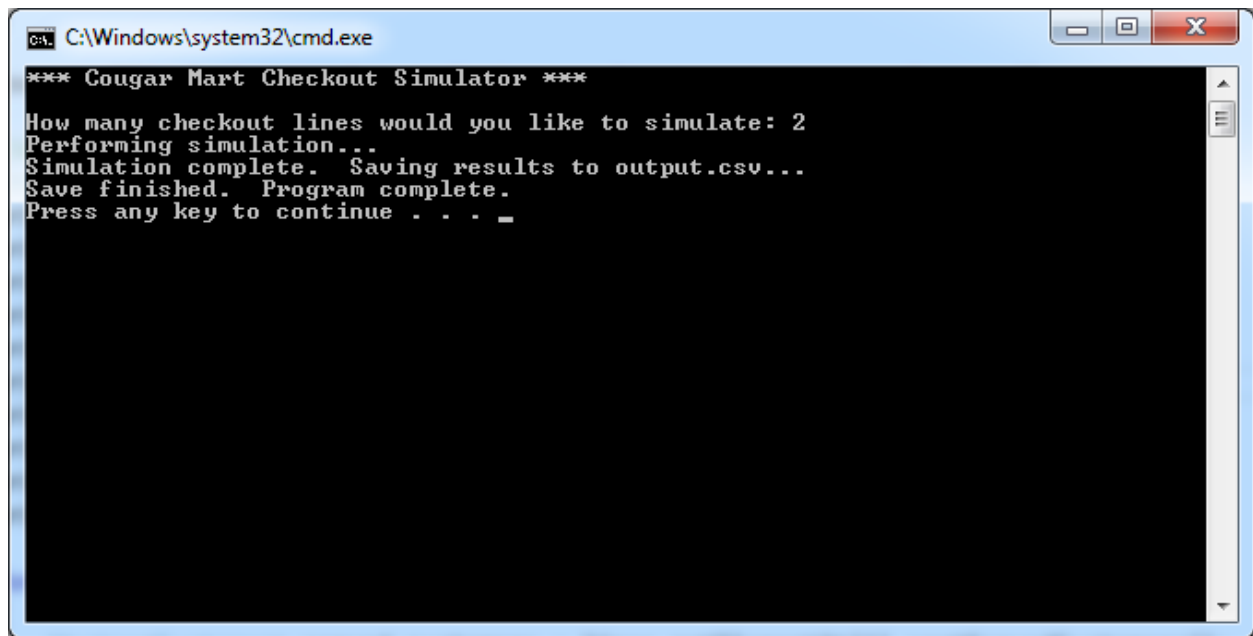
Adds the supplied customer to our "_line" queue of customers

getServicedCustomers

Returns a reference to our vector of serviced customers. To be used when creating the final CSV file.

Sample Output

Below is a screenshot from my program. Note that while you are free to create your own internal implementation, your program must conform to the output below.



```
C:\Windows\system32\cmd.exe

*** Cougar Mart Checkout Simulator ***

How many checkout lines would you like to simulate: 2
Performing simulation...
Simulation complete. Saving results to output.csv...
Save finished. Program complete.
Press any key to continue . . . _
```

Header Comment, and Formatting

1. Be sure to modify the file header comment at the top of your script to indicate your name, student ID, completion time, and the names of any individuals that you collaborated with on the assignment.
2. Remember to follow the basic coding style guide. A basic list of rules can be found on OSBLE.

Deliverables

You must upload your program store through OSBIDE no later than midnight on Wednesday, April 16, 2014. For more information on how to submit an assignment, [read this page](#).

Grading Criteria

Your assignment will be judged by the following criteria:

Error Free Compile (weight: 5%)

- [0] Your program contains compiler errors.
- [1] Your program compiles without issue.

Error Free Runtime (weight: 5%)

- [0] Your program throws a runtime exception.
- [1] Your program does not encounter any runtime exceptions.

Pointer Cleanup (weight: 5%)

- [0] Your program does not delete any dynamically-created pointers
- [1] Your program remembers to delete some, but not all dynamically-created pointers
- [2] Your program deletes all dynamically-created pointers

Style (weight: 10%)

- [0] Your program does not contain good style. Variables are poorly named, your program doesn't contain a header, there is little or no documentation, your document isn't properly formatted (indentation, whitespace, etc.), and is generally hard to read.
- [1] Your program contains suboptimal style. Some of the items listed in the [0] section are accounted for
- [2] Your program contains good style. Most of the items listed in the [0] section are accounted for.
- [3] Your program contains excellent style. Almost everything in listed in the [0] section is accounted for.

Implementation (weight: 20%)

- [0-3] Your program's implementation is inefficient. Class usage is non-existent, your code has a lot of redundancy, and is generally not up to the standards of what is expected of a CptS 122 student.
- [4-6] Your program's implementation is acceptable, but can be improved upon. You may use classes, but they are used poorly. Likewise, some redundancy must exist.
- [7-10] Your program's implementation is exemplar. Classes are used in thoughtful and meaningful ways. There is little, if any, redundancy. If this were kindergarten, you would receive a gold star.

User Interface (weight: 5%)

- [0] Your program does not even attempt to follow the UI guidelines
- [1] Your program's UI has major inconsistencies when compared to the sample output
- [2] Your program's UI has minor inconsistencies when compared to the sample output
- [3] Your program completely matches the specified user interface guidelines

CSV Correctness (weight: 10%)

- [0] Your program does not even attempt to create a CSV file
- [1] Your program creates a CSV file but it is corrupted or unreadable
- [2] Your program creates a readable CSV file, but certain data points are missing
- [3] Your program creates a CSV file in the expected format.

Data Structure Usage (weight : 10%)

- [0-1] Your program uses little to no data structures.
- [2-3] Your program uses the STL data structures or the custom data structures developed in class.

Simulation Algorithm Correctness (weight: 30%)

- [0-2] You do not implement the simulation algorithm as described in this document
- [3-4] Your algorithm implementation differs significantly from the implementation described in this document
- [5-6] Your algorithm mostly follows the implementation listed in this document, but is lacking a few key components
- [7-8] Your algorithm faithfully implements the supplied simulation algorithm.

OSBIDE Reflection

Remember that all assignments require you to post a brief (min: 150 words) reflection to OSBIDE. Some topics to consider:

- How many lanes are required in order to serve all customers? In other words, how many checkstands are necessary so that at the end of the day, there remains little or no customers in the checkout line.
- How many lanes are required in order to ensure that all customers are served in under {15, 10, 5} ticks?
- Assume that it costs \$10/hr to staff a checkstand, and that each checkstand will be open for the entire simulation (12 hours). If you were the owner of Cougar Mart, how many checkstands would you staff? Why?