

Project 1

The Grocery Store Simulation

Grocery Store Simulation

- Project 1: Grocery Store Simulation
 - Handouts / Description now on Web site
- Due Dates:
 - Minimum Effort Due: January 9, 2004
 - Full Project Due: January 17, 2004

Grocery Store Simulation

- CS2 Newsgroup
 - rit.cs.courses.4003.232

Grocery Store Simulation

- Goal for this project is to simulate the register area of a large grocery store.
- Rules:
 - Store has a number of registers
 - Number of registers read in from standard in
 - Each register will have a queue (line) associated with it.
 - All lines are the same (no express, coupon only, no-candy lines)

Grocery Store Simulation

- Customer Entering a line
 - When a customer is ready to check out:
 - Will go to the register with the smallest line
- Cashing out:
 - When a cashier is not busy, he/she will cash out the customer at the front of the queue
 - After checkout, the customer leaves the store and the cashier checks out the next customer.

Grocery Store Simulation

- Simulation:
 - Input:
 - A random number generator seed
 - Average rate of arrival of customers
 - Average service time for cashiers
 - Number of registers
 - Amount of time to run the simulation

Grocery Store Simulation

- Simulation:
 - Simulation loop (for passage of one unit of time)
 - Customer generator is asked for a list of new customers
 - Each new customer places his/her self onto a register queue
 - Each register checkout determines if current customer is done
 - If yes, update statistics for the register and get next customer from the queue (if there is one)
 - Update your clock

Grocery Store Simulation

- Simulation:
 - Statistics collected:
 - Average customer service time
 - Average customer waiting time
 - Average cashier idle time
 - No work actually done...this is a simulation
 - Questions so far?

Grocery Store Simulation

- Specification:
 - To run:
 - `java GroceryStore`
 - Input:
 - From standard input (`System.in`)
 - Random Number seed (int)
 - Average customer arrival rate (double)
 - Average service time (double)
 - Number of cashiers (int)
 - Total time for simulation (int)
 - Must check data format..if incorrect print error and terminate.

Grocery Store Simulation

- Specification:
 - Output:
 - Must print collected statistics after the simulation is complete.
 - Since no “real” work is done, diagnostics will be printed to indicate what is going on with the simulation.

Grocery Store Simulation

- Classes:
 - `GroceryStore`
 - Main program for running simulation
 - `Clock`
 - Time keeper for the simulation
 - `CustomerQueue`
 - Models the lines for each register
 - When a customer is added – message is printed
 - When a customer is removed – message is printed.
 - `Customer`
 - Models the customers in the system
 - Can determine shortest line and place themselves on the shortest line.

Grocery Store Simulation

- Classes:
 - `Register`
 - Models the cash registers.
 - Maintains a queue
 - Service Customer (`checkIfCustomerDone()`)
 - Must keep track of
 - Total customers serviced
 - Total time servicing customers
 - Total time spent in queue
 - Total idle time

Grocery Store Simulation

- Classes:
 - `CustomerGenerator`
 - Generates random set of customers for each time tick.
 - `InvalidValueException`
 - Thrown during invalid data entry
 - `ExponentialRandom`
 - Determines time until next event
 - Used by `CustomerGenerator`
 - `PoissonRandom`
 - Determine number of events per unit of time
 - Used by `CustomerGenerator`

Your Job...

- Write the classes:
 - Javadocs included for all classes
 - `Clock`
 - `CustomerQueue`
 - `Customer`
 - `Register`
 - Please use RCS
 - java files supplied for `GroceryStore` & `InvalidValueException` (in RCS)
 - .class files supplied for all other classes.

Important Algorithms/Data Structures

- Data Structures
 - Queue
- Algorithms
 - Running the simulation
 - Finding the shortest line
 - Servicing a customer

Circular Queue

- Queue
 - New items get placed at the end of the queue
 - Items get removed from the front of the queue
 - First in / First Out (FIFO)



Circular Queue

- Queue methods
 - `add` – Places a new object at the rear of the queue
 - `getFront` – return object at front of queue without removing it.
 - `getSize` – returns number of elements in the queue
 - `isEmpty` – returns a boolean indicating if the queue is empty
 - `isFull` – returns a boolean indicating if the queue is full
 - `remove` – Removes and returns element at the front of the queue

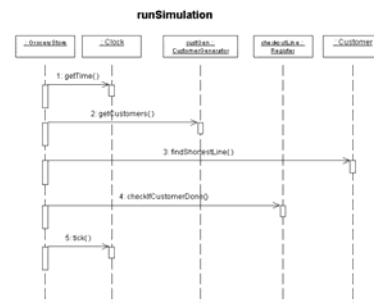
Circular Queue

- Implementation
 - Place data in an array
 - Keep track of the front and rear of the queue
 - Keep track of number of elements
- Let's see this in action.

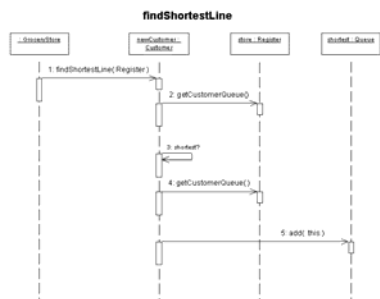
Circular Queue

- For the CustomerQueue:
 - Must be dynamic
 - When full, and a new Customer is added,
 - Must make the queue larger.
 - Initial array size = 5
 - isFull will always return false!

Running the Simulation



Finding the Shortest Line



Servicing a Customer

- Check to see if we need to begin serving a new customer
- Check to see if the current customer is finished

Need new customer?

- If we are finished with current customer:
 - Check to see if there is anyone behind him or her in the queue.
 - Yes, there are customers:
 - Remove the next customer from the queue and make him or her the current customer.
 - Remember the time this customer will require to be serviced.
 - No, there are no customers:
 - Increment the idle time by 1 time unit.
- If we are not finished with this customer, proceed to step 2.

Finished with current customer?

- If there is time left:
 - Decrement the time left on this customer.
 - Is there still time left on this customer?
 - Time left: do nothing.
 - No time left:
 - Record the data for the final statistics reporting.
 - Set the current customer to null
- Otherwise, do nothing.

Testing your work

- You can use try to test out your classes
 - try `cs2-grd project1-test infile`
 - Will run our solution on test data in file `infile`
 - You can redirect the output into a file and then compare with our output
 - try `cs2-grd project1-test infile > correctSolution`
 - `java GroceryStore < infile > mySolution`
 - `diff mySolution correctSolution`
 - Can also submit your own java files to test
 - try `cs2-grd project1-test infile somefile.java`
- Sample input files include in `/data` directory of jar file.
- Be sure that output matches exactly!

Submissions

- 4 submissions
 - Minimum: `Clock.java` / `CustomerQueue.java`
 - due 1/9/04
 - Need stubbed `Customer` class.
 - Submission 2: `Customer.java` / `Register.java`
 - due 1/17/04
- All submissions via `try`
- NO LATE SUBMISSIONS!

Submissions

- About the minimum submission
 - `Clock.java` & `CustomerQueue.java` is the minimum reasonable effort requirement for this project
 - Due January 9, 2004
 - Must submit `Clock.java` & `CustomerQueue.java` successfully by Jan 9.
 - Otherwise, you fail the course
 - Advice: Submit Before Break!!!

Grading

- 100 points for functionality
 - Up to 35 point deduction for bad implementation
 - Up to 30 point deduction for bad style
 - Including non-use of RCS
- Submission percentages
 - `Clock` – 10 %
 - `CustomerQueue` – 20%
 - `Customer` – 30 %
 - `Register` – 30%
 - All Together – 10 %

Questions?

- Tomorrow:
 - Java I/O