Homework 3-1

Design and Specifications

Design and Approach to synchronization:

1.CoffeeShopManager

A Neutral Helper class, which will have a record of all the customers being currently served in coffee shop. Along with the track of a cook serving an order with cook names and order number. This class will be a central and integral part of this application.
There are a few data structures inside this class namely:

a. HashMap<Customer, Boolean> orderStatus: indicating whether a customer's order has been served or not.

b. ArrayList<Machines>: indicating currently serving active machines

c. ArrayList<Customer> currentlyServing: indicating a track of customers being served currently

All the classes such as cook, customer, machine and CookAnItem revolves around the above class. I have thought of two object locks predominantly for Customer side and another for Machines and Cook side.

Synchronization would be obtained on this lock. Most importantly, there would be just one instance of CoffeeShopManager in all. This object will be initiated in Simulation.runSimulation() method with the parameters as passed to runSimulation like numTables, numCustomers, numCooks etc.

2. Customer

A customer thread will acquire lock on CoffeeShopManager's instance variable lockOne and perform the run method. Duties of a customer include waiting for tables to be available, placing an order, receiving an order and wait for customer to eat up his/her order and leave the coffee shop. Another customer will do the same thing.

3. Cook

A cook thread will acquire lock on CoffeeShopManager's instance variable lockTwo and perform run method. Which will include following activities. Track whether cook is available to work or not. Then keep a track of Customer's order. Delegate the order's item individually to specific machine and wait for that machine to finish up. Updating the order status in CoffeeShopManager's array list. Delegating or conveying this information to Customer thread indicating order has been completed.

4. Machine

A machine thread will check if machine is available to work and initiate a thread of CookAnItem as suggested in the pdf to cook. This thread will sleep for certain milliseconds to demonstrate a simulation.

During all of these executions, each of the class like Customer, Cook, machine etc will log the Simulation events for validator to validate.This is the approach I have finalized for this application.

There might be minor changes to the design in case where my implementation does not go as planned.

Invariants, Pre-conditions, Post-conditions:

1. Customer

a. Invariants: Accepts properties of customer like list of order, order number, name of customer and waits for available table inside coffee shop

b. Pre-conditions: Wait till occupied tables count is less than available total number of tables

c. Post-conditions: Entered coffee shop, placed order, received order, eats and leave

d. Exceptions: InterruptedException in case of interrupt on customer thread

2. Cook

a. Invariants: Delegate cooking of an individual item to machines

b. Pre-conditions: Wait till the cook is free to process new order

c. Post-conditions: Pull customers order based on customer's priority. Process each item from order and delegate the work to specific machine based on food. Wait till there is a signal from machine indicating cooked successfully and indicate
Customer about their order.

d. Exceptions: InterruptedException in case of interrupt on Cook thread indicating end of day. Time to leave.

3. Machine

a. Invariants: Cooks food which is to sleep till food cooks and indicate it back to Cook

b. Pre-conditions: Wait till machine's current food preparation capacity is less than max capacity

c. Post-conditions: Initiate threads so that items get cooked in parallel up to the max capacity of each machine and indicate the Cook about food processed successfully

d. Exceptions: InterruptedException in case of interrupt on CookAnItem thread

4. Simulation

a. Invariants: Run simulation with parameters passed from arguments or input.

b. Pre-conditions: Availability of parameters such as numTables, numCustomers, numCooks, randomOrder

c. Post-conditions: Run simulation successfully and validate the execution utilizing Validate class functionality

d. Exceptions: None

5. SimulationEvent

a. Invariants: Log each and every simulation event to validate after execution.

b. Pre-conditions: None

c. Post-conditions: Successfully prints and records each event

d. Exceptions: None

6. CoffeeShopManager

a. Invariants: Keeps track of all current executions like currently serving customers or orders

b. Pre-conditions: None

c. Post-conditions: Provisions solid lock management system for synchronization on several of collections etc.

d. Exceptions: None