**Approach to Synchronization**

In order to avoid inconsistency and make sure the invariant conditions are always true for this model we will need to implement synchronized threads.

1. Thread (Customer) tries to access Coffee Shop

Validate if numTables> numCustomers, access granted (SimulationEvent.customerEnteredCoffeeShop()) else thread wait()

1. Customer places Order once enters the shop

If randomOrder == true then random number (between 0 to 3) of burger, fries and coffee is set

Else (when randomOrder==false) then order is set to default i.e. two burgers, one fries, one coffee

Order number is generated by Customer constructor (SimulationEvent.customerPlacedOrder()).

Order details are added to the orderList and then thread wait()

1. Thread (Cook) awaits order (accesses the orderList) and passes each item on one order to respective machines (machines are specific to food type)

machine.makefood() is invoked and thread Cook wait() till machine takes time to cook

1. Each machine has fullCapacity (maximum capacity to cook) which will be compared to its currentCapacity

currentCapacity <= fullCapacity

1. machine.makeFood() cooks food in parallel by using thread internally and making sure

currentCapacity <= fullCapacity is always true

This internal thread (cookItem) sleeps for thread.sleep(n), where n is the cook time of the food. Once completed cookItem notifies makeFood() which in turn notifies the thread (Cook) which has been in wait() state.

1. Cook thread once notified in turn notifies (SimulationEvent.customerReceivedOrder()) Customer thread which has been in wait() state too. Cook thread then moves on to the next item on the orderList
2. Customer thread once notified (SimulationEvent.customerReceivedOrder()) is then put to wait() for some time (eats the order) and then terminates (SimulationEvent.customerLeavingCoffeeShop())
3. Once the orderList is completed and no more Customers are waiting to enter the shop the Cook thread can terminate (SimulationEvent.cookEnding())

**Pre/Post Conditions, Invariants & Exceptions**

1. **Simulation.java**

**Invariants**: orderList is empty, occupied tables is zero, customer received the order and left

**runSimulation()**

**PreConditions:**

* numCustomers, numTables, numCooks and machineCapacity >0
* randomOrders = true or false

**PostCondition:**

* returns a list of Events

1. **Customer.java**

**PreConditions:**

* Order, name should not be null

**run()**

**PreConditions**:

* numTables> numCustomers

**PostCondition**:

* customer receives order and leaves the shop

1. **Cook.java**

**PreConditions:**

* name should not be null

**run()**

**PostCondition**:

* Order for specific customer is completed

**Exception:**

* InterruptedException when cook has to go home

1. **Machine.java**

**Invariants**:

* currentCapacity of food being cooked by a machine should always be <= fullCapacity of machine
* Each food item takes at least item.cookTimeMS milliseconds to produce

**makeFood()**

**PostCondition**:

* internal thread CookAnItem is called to manufacture the food item

**CookAnItem**()

**PostCondition**

* food item is cooked and event logged

1. **Validate.java**

**PreConditions**

* Non- empty eventList

**PostCondition**

* Returns Boolean res - whether the simulation was valid or not