**Coffee Shop Simulation:**

**Approach to Synchronization:**

A “Producer- Consumer” pattern based approach is adopted. (Although, we are stopped from using Concurrent collections i.e. Blocking queue, we can create one of our own 😉).

1. **CustomersInQueue:**

* This will be a priority queue (FIFO queue) that defines the customers currently waiting for the restaurant.
* Newly created customers are added to this queue, and will be served by the cook later.
* The size of Priority Queue is unbounded regardless of the number of tables currently available at the coffee shop.
* This queue will be shared by customer and another priority Queue that has customers currently in the shop. It must be synchronized.

*Customer Events:*

1. customerStarting: trigger once the customer is created
2. **CustomersInShop:**

* This will be a priority queue(FIFO queue) that defines customers currently in the restaurant.
* Customers who enter the shop, place their orders and wait here (at the shop) until their order is complete.
* The size of Priority queue is bounded by number of tables in the shop. So, the customers in shop cannot exceed the max capacity of queue and must wait in the previous queue for a place.
* This queue will be used in Customer to add new customer to queue and then to remove a customer leaving the shop.

*Customer events:*

1. customerEnteredShop: trigger once the customer entered coffee shop.
2. **CustomerOrdersStatus:**

* This will be an associative data structure, typically a Hash Map to store the status of each customer in the coffee shop.
* The status can be completed or in-process, represented as a Boolean value.
* Completed = TRUE, in-process =FALSE.
* The purpose is to keep track of customers waiting in the shop for their orders.
* Once a customer receives the order, the status will be true, indicating his/her order is complete.
* The customer must be removed from the CustomersInShop queue, so that those waiting can get into the shop.
* This data structure is accessed to add customer from CustomersInShop queue and by cook to retrieve the customer order.

*Customer events:*

1. customerOrderPlaced: trigger once the customer is added to this hash map.
2. customerReceivedOrder: trigger once the status is marked complete

*Cook events:*

1. cookReceivedOrder: trigger once the order of customer is read by cook, once the customer is added to this map, the customer’s order is taken by cook. Status = in-process.
2. cookCompletedOrder: trigger once the cook has completed the customer order and updated status of CustomerOrderStatus. Status = completed.
3. **OrderItemsList:**

* This is a linked list (specifically a FIFO Queue) used by every machine to process all the order items.
* The cook adds all order items of received order to this list and resume to serve the next customer.
* Machine processes all the received orders in parallel and triggers an event upon completing the order.

*Cook events:*

1. cookStartedFood: trigger once the cook has placed the order in OrderItemsList.
2. cookFinishedFood: trigger once the cook has received trigger from machine that order is done.

*Machine events:*

1. machineCookingFood: trigger once the machine has started processing an order item.

Sleep the machine thread for the time required to cook food item, i.e. cooking food = sleeping the machine thread.

1. machineDoneFood: trigger once the machine has completed processing order item.
2. **OrderCompleteList:**

This will be a linked list, a queue of food items that are completed by machine.

This will be accessed by machine and cook.

*Cook events:*

1. cookCompletedOrder: trigger once the queue is verified to hold all the customer items, (after updating CustomerOrderStatus).

**Invariants, PreConditions, PostConditions:**

**Customer.java**

Invariants:

1. CustomerOrderNumber cannot change.
2. CustomerName cannot change
3. CustomerDiningTime, an attribute that defines customer’s dining time at shop is preset.
4. CustomerOrder may or may not include all the order items but contains at least one order item.

PreConditions:

1. A customersInQueue is created beforehand and customers are in the queue.
2. Every customer object has an orderList created and contains the orderItems.

PostConditions:

1. Customer events such as customerEnteredShop, customerOrderPlaced, customerReceivedOrder, customerLeavingShop are triggered.
2. Customer order is marked complete against CustomerOrderStatus HashMap.
3. Customer is removed from the customerInShop queue.

Exception:

1.Throw InterruptedException if CustomersInShop.size() is greater than threshold.

**Cook.java:**

Invariants:

* + - 1. Cook name cannot change
      2. CookStarting event must be triggered before running cook.

PreConditions:

1. Order received by a cook cannot be empty.

PostConditions:

Update the OrderCompleteList with the completed orders, CustomerOrderStatus

Trigger cookCompletedOrder after finishing the order.

Exceptions:

Throw InterruptedException: when the cook is ending

**Machine.java:**

Invariants:

* + - 1. Trigger MahineStarting should be invoked before any processing to be done on Machine.
      2. CookAnItem thread sleeps only till cookTime mentioned.
      3. MachineName, MacbineFoodType are already set and does not change

PostConditions:

OrderCompleteList updated upon completing an order

Trigger machineDoneFood after completing the order.

Exceptions:

Throw InterruptedException: while the thread is sleeping for cookTime