Jack Diaz
111499298

My design satisfies the validation criteria by using Blocking Queues, CountDownLatches, and Future Tasks. First, when the eaters are lined up outside of the restaurant, they are made to wait until there is an empty table. The way I implemented this was by having a LinkedBlockingQueue holding Eaters called host that has a capacity set to the number of tables in the Simulation. Eaters are put onto the queue which blocks them until there is a spot open on the queue. When they leave the restaurant they remove themselves from the queue opening a table up for another Eater. This ensures that there are no more eaters in the restaurant than there are tables.

I have another LinkedBlockingQueue called waiter that holds orders. When an Eater is in the restaurant they will place their order with the waiter. This has no capacity but basically has an upper limit which is the number of tables as it cannot receive more orders than there are tables at any one time because of the limit of the host. The Cooks will take the orders from the waiter, but only when there are orders to be taken. Otherwise they will block until the waiter receives an order or until the Cook is interrupted. This ensures that Cooks won't start cooking anything until an Eater has placed an order.

Once the Cook has the order he will put each Food item into it's appropriate Machine based on the name of that Food and the name of the Food item that that Machine accepts. The Machines, in the makeFood method, will spawn a new thread which is given a FutureTask that cooks the Food for the amount of time in that Food's cookTimeMS field. The FutureTask is returned to the Cook by the makeFood method. This way the Machine will cook the Food while the Cook is free to put the other Food items into their respective Machines. When the Cook receives the FutureTasks he collects them, and after he has put all the Food items of the order in to cook, he then tries to get the returned items of the FutureTasks he has collected. The FutureTasks will return the Food item that they have just cooked. The Cook will be blocked by get until the Food is cooked and the FutureTask returns. This ensures he does not give the order to the Eater before the Food is done cooking.

The way the Cook tells the Eater that their food is ready is by using a CountDownLatch. The CountDownLatch is set to one and after the Eater places their order they are blocked by await on the latch. When the Machines have finished cooking all the Food items and they are in the Cook's possession he will call countDown on the latch allowing the Eater to continue. This ensures that the Eater does not receive their order and leave before it is done cooking or before it is completed by the Cook.

In order to ensure that the Machines' capacities aren't exceed, they hold a Semaphore set to the capacity of the Machines. When their makeFood methods are called they immediately attempt to acquire from the Semaphore. This ensures that they won't proceed (and neither will the Cook) until there is room in the Machine. Once the Cook gets the Food out of the Machine, the Semaphore's release is called, signalling that there is now room in the Machine so a makeFood method waiting to acquire can do so.

In order to make sure that the Cooks go home after all of the Eaters have received their orders and left the restaurant, I use another CountDownLatch. This one is set to the number of Eaters in the Simulation. After creating all the Threads, the Simulation is blocked on this CountDownLatch. When an Eater leaves the restaurant they call countDown on the CountDownLatch. Once the last Eater has left, the Simulation no longer blocks and moves on to the code to call interrupt on all of the Cooks. This makes sure that no Cooks leave before all of the Eaters have left and that all the Eaters are gone before the Cooks are allowed to leave.

a comprehensive test plan. The test plan should describe how you tested each piece of functionality in your code.

The way I tested my code was by making my validateSimulation method as comprehensive as I thought necessary because without a good validator all the testing of my Simulation would be a waste. To do this I checked the following things:

- No more Eaters than specified by counting each starting Eater and comparing to numEaters
- No more cooks than specified by counting each starting Cook and comparing to numCooks
- The restaurant capacity should not be exceeded by adding for each entering Eater and subtracting for each leaving Eater
- The capacity of each machine should not be exceeded by adding for each added Food item and subtracting for each taken out Food item
- Eater should not receive order until Cook completes it by checking place in the list of each event
- Eater should not leave restaurant until order is received by checking place in the list of each event
- Eater should not place more than one order by checking if they've already ordered
- Cook should not work on order before it is placed by checking place in the list of each event
- Cooks don't leave before all Eaters have left by counting the number of Eaters that have left when a Cook leaves
- Machines get right Food by checking the name of the Food the Machine should be getting against the name of the Food it is getting
- Each Eater's order must be given a unique order number by checking that no orders are done twice and that the number of orders are equal to the number of eaters
- All eater names should be of the form "Eater "+num where num is between 0 and the number of eaters minus one
- Each Eater does everything in order by using Eater states
- All cook names should be of the form "Cook "+num where num is between 0 and the number of cooks minus one.
- Each Cook does everything in order by using Cook states
- Each order is processed in the correct order by using order states
- Cook should not start a new order until last order is finished by using Cook states
- Each Machine does everything in order by using Machine states
- Each Eater who started also left
- Each Cook who started also left

My validate method also has comments that may help.

I also ran my Simulation numerous times with a number of different parameters. I tended to make the number of Eaters really high because that determined how many things there were to do. The more things there are to do, the more likely something undesirable will show up. I also had some runs with a low number of Eaters in order to test the ability of the Cooks to wait for orders. For the other parameters I tried all different variations of high and low numbers compared to each other. Having low tables but high Cooks would test their ability to wait for orders. Having high Cooks and low Machine capacity would test if the Cooks would wait for the Machines properly.

I also tested with Thread.yield() before each logEvent call and in other places where I thought interruption might be inopportune.