Customer: Customer who orders the food and occupies the place for some time eating.

* Invariants: runningCounter < number of the seats in shop which will be given in the simulation. Should satisfy this all the time when simulation running.
* Need To Do:

1. Three private variables: name for identifying the customer, order for recording of the customer's order, orderNum for identifying the order.
2. One static variable for recording the number of the customer.
3. run should run with the changing of state.
4. CustomerStarting -> means customer thread starting.
5. CustomerEnteredCoffeeShop -> means customer come into the shop.
6. Nothing to do with above two states.
7. CustomerPlaceOrder -> submit the order to cook. If no cook is available, wait until there is at least one cook available and notify the customer.
8. CustomerReceivedOrder -> wait until the cook notify that the order is done.
9. CustomerLeavingCoffeeShop -> wait for some time to finish the food and leave the coffee shop which means the thread terminal.

* How To Do:

1. For step 4, 5, 7, 8, 9, all should use the logEvent in Simulation to log the event and add to list to show in these state.
2. For step 7, look around the event list to find if there is one cook in the state CookEnding. If no cook, lock by event and wait for cook to notify.
3. For step 8, lock order until the cook unlock the order and notify customer.
4. For step 9, use Thread.sleep(time) for simulate eating.

* Places for synchronization:

1. When in the state CustomerPlaceOrder, lock event.
2. When in the state CustomerReceivedOrder, lock order.

Cook: Cook who operate the machine for order.

* Invariants: cook for only one order.
* Need To Do:

1. name to identify the cook.
2. CookStarting -> means cook start.
3. CookEnding -> means cook end.
4. CookReceivedOrder -> means receiving order.
5. CookStartedFood -> means putting the food in the machine thread for cooking. If the machine is full, wait until there is one machine free.
6. CookFinishedFood -> means all machine finished the food.
7. CookCompletedOrder -> means cook complete the order and notify the customer that the order is finished, then the cook is free and move to the end state, which means the cook ready for next order.

* How To Do:

1. For step 2, 3, 4, 5, 6, 7 use logEvent to log the event to the list.
2. Wait in the state CookStarting until be notified by customer that there is order.
3. Obtain the lock of order and assign the order to the right machine when the machine is free. If no machine is free, wait util one free.
4. Wait until all food is finished, notify the customer by the lock order.

5. Reset the state to CookStarting for waiting order.
6. When interrupted, turn to the state CookEnding and end the cook Thread.

* Places for synchronization:

1. When in the state CookStarting, lock event and wait until there is one order.
2. Lock order all time in the cooking process. Notify the customer if finished.

Machine: Cook machine. (should be multi-thread) Each machine can cook just one kind of food. But can cook more than one food.

* Invariants: Use Threads to handle multiple requests for the kind of food. The food in the cook process, which is the max amount of the Threads, should be less than the capability of the machine which is assigned when the machine is completed. If exceed, wait until there is one Thread end. Current Thread should be more than 0.
* Need To Do:

1. machineName identifies the machine, machineFoodType identifies the food the machine can make, capability means the capability of the machine, current means current Thread runs in the machine.
2. Method makeFood used to make food running the thread CookAnItem.
3. Thread CookAnItem runs between four state of the machine.
4. MachineStarting -> wait for cook to call.
5. MachineStartingFood -> receive the order.
6. MachineDoneFood -> wait for some time, finish the food and delivery to the cook.
7. MachineEnding -> machine end.
8. Minus one to current when done.
9. When the thread interrupted, the thread end. The amount of the thread means the capability of the machine.

* How To Do:

1. Method makeFood should create Thread and wait. If there is one food arriving, do it using one Thread. If the number of Thread is equal to the capability, wait util one Thread to end before creating a new Thread.
2. Use Thread.sleep(time) to simulate the cooking process.
3. Use logEvent to log the event. Step 4, 5 is out the Thread.

* Nothing synchronization.

Simulation: Simulate all things for the coffee shop including order, cook, receive and leave.

* Invariants: customers should not exceed the number for the shop. If exceed, the customer should wait until there are someone who leave.
* Need To Do: only method runSimulation. First, start machines. Just instantiate the machine class is ok. Then, add cooks. Just create cook object and run it. The same with the build of customer.
* How To Do: using construction of the classes.
* Nothing synchronization. Synchronizations are all in the Threads.

Validate: Check something like overflowed customers or foods that are cooked.

* Invariants: nothing.
* Need To Do: check if the event is successfully logged in. Check if the number of the customers is over the limit. Check if the number of Threads in machine is over the capability.
* How To Do: check if the number is over and if the state is right. Lock event.