Assignment 3-1

Name: Anubhav Gupta

1. **Customer.java**

    a. **Invariants:** name must be not null and all customer names should be of the form "Customer "+num where num is between 0 and the number of customers minus one.
    b. **Customer():**
        - **Precondition**: Name is not null, order is not null and order contains validFoodType.
        - **PostCondition**: Customer object created and returned.
        - **Exception**: CustomerNotValid Exception.
    c.

2. **Cook.java**

    a. **Invariants** : name must not be null and in the format "Cook"+ num where num is between 0 and the number of cooks minus one.
    b. **Cook()**:
        - **Precondition**: Cookname is received in the form of "Cook "+cookNumber and it cannot be null.
        - **PostCondition**: Cook instance is created
        - **Exception**: CookNameNotValid Exception

3. **Machine.java**

    a. **Invariants** : MachineName and food should not be null. Food should be one of foodType. Capacity should be > 0
    b. **Machine()**:
        - **Precondition**: name, food not null and capacity >0. Food should a valid foodType.
        - **PostCondition**: Machine Object created
        - **Exception**: None
    c. **makeFood()** :
        - **Precondition** : should receive orderNo which must be > 0 and valid order no.
        - **PostCondition** : If there is enough capacity, then cook the food, else wait.
        - **Exception** : None.

Synchronization Strategy:

Define a customer list having a size equal to the number of tables.
When customer enters the coffee shop, it will acquire lock on this list. If the list is full, it will wait else customer gets added to the list.
Then customer acquires the lock on the orders list that will have all the active orders, customer then places the order by adding his order to the list. Customer releases the lock on the orders list by notifyAll().
Customer acquires lock on his order. He then waits for order to be completed.

All the cooks are waiting on orders list lock if no orders have been added yet. Once the orders are added cook will fetch the order by the order they arrive or by priority if they are more than one order in Cook's hand. Cook then acquires lock on the order it is processing. Cook then processes the order by submitting each food item to an appropriate machine. Machine acquires lock on its capacity as it prepares the food item. If the capacity is reached, then the order is put on wait() else the machine proceeds with preparing the item while internally creating cookAnItem thread for each food item. It then sends back the Boolean true once the order is food is cooked.

After the food is prepared by the machine, cook will notify the customer by doing a order.notify() on his order. As the customer is waiting on the same order lock, It will get notified.

It will then acquire lock on customerList again, remove the customer from the list , release the customerList Lock and leave the coffee shop.