Doreen Hakimi
CS201 Restaurant V2.1
Fall 2013

DATA

List<Order> orders

hashMap<String, Food> myFood;

List<List<FoodOrder>> delivery
List<MarketAgent> markets
List<InventoryOrder> myOrders
List<FoodOrder> orderToMarket
List<List<Order>> inventoryRequests

boolean RestaurantIsOpen= false
boolean CheckedAtFirst= false //makes sure checks if kitchen is ready at the start only once
boolean waitingForInventory=false // prevents kitchen from reordering when expecting an order to arrive

int ORDER_ID=1

int initialFoodAmnt; // fill target for kitchen, threshold to reorder is less than half this value .. TO BE DETERMINED BY RUNNER TO TEST CASES

```
class InventoryOrder {
        int orderID;
        List<FoodOrder> myOrder;
        int mktOrderingFrom = 1;
        boolean reorder=false;

}

//note, in my implementation, i made FoodOrder an outer class due to accessibility of class for both waiter and cook
class FoodOrder {
        String food;
        int val
}
//note, in my implementation, i made order an outer class due to accessibility of class for both waiter and cook
class Order {
        int tablenum;
        String choice;
        Waiter w;
        state= {pending, cooking, cooked};
```

```
        Order() { state=pending; }
}
```

MESSAGES

```
HereIsAnOrder(Order o) {
        orders.add(o);
 }

DoneCooking(o) {
        o.state=cooked;
}

HereIsYourFoodOrder(List<FoodOrder> dlv) {
        delivery.add(dlv);
        waitingForInventory=false
}

CouldNotFulfillThese(List<FoodOrder> reorderlist, int ORDERID) {
        if there exists an InventoryOrder order in myOrders such that order.id=ORDERID
                then order.incMarketOrderingFrom();
                order.reorder=true;
                order.myorder=reorderlist;
}
```

SCHEDULER

```
if there exists an Order o in orders such that o.state=pending
        then CookOrder(o);
if there exists an Order o in orders such that state=cooked
        then CallWaiter(o);
if(!delivery.empty())
        then ProcessDelivery(delivery.get(0);
if there exists an InventoryOrder i in myOrders in which i.reorder=true
        then ReorderFood(i);
if(!RestaurantIsOpen and !CheckedAtFirst)
        then CheckIfFullyStocked();
if(!waitingForInventory)
        if there exists any Food f in which f.amount < initialFoodAmnt/2
                then OrderFoodFromMarket();
```

ACTIONS

Doreen Hakimi
CS201 Restaurant V2.1
Fall 2013

```
CheckIfFullyStocked() {
        boolean fullyStocked=true;
        if there exists any Food f in which f.amount < initialFoodAmnt/2
                fullyStocked=false;
        if(fullyStocked) {
                //if there aren't any, ready for open
                host.KitchenIsReady();
                RestaurantIsOpen=true;
        }
        CheckedAtFirst=true;
}

CookOrder(Order o) {
        Food food=myFood.get(o.choice);
        if food.amount=0
                o.waiter.OutOfFood(o);
                orders.remove(o);
                return;
        food.amount -= 1;
        myFood.put(o.choice, food);

        o. state=cooking;
        timer.start() {
                msgDoneCooking(o);
        }
}

CallWaiter(Order o) {
        o.waiter.OrderIsReady(o);
        orders.remove(o);
}

OrderFoodFromMarket() {
        orderToMarket.clear();
        for all food choices s
                if(myFood.get(s).amount<=initialFoodAmnt/2)
                        then orderToMarket.add(new FoodOrder(s, intiialFoodAmnt -
                        (initialFoodAmnt/2)))

        markets.get(0).HereIsAnInventoryOrder(orderToMarket, ORDER_ID)
        myOrders.add(new InventoryOrder(orderToMarket, OrDER_ID));
        ORDER_ID++;
        waitingForInventory=true;
```

```
}

ReorderFood(InventoryOrder reord) {
        if(reord.mktOrderingFrom>markets.size() {
                if(!RestaurantIsOpen and reord.mktOrderingFrom>markets.size()) // markets
        don't have it..
                //should open restaurant anyway
                        then RestaurnatIsOpen=true;
                                host.KitchenIsReady();
                myOrder.remove(reord);
                return;
        }
        //r.mktOrderingFrom is still valid
        markets.get(reord.mktOrderingFrom).HereIsAnInventoryOrder(reor.myorder,
        reord.orderID);
        reord.reorder=false;
}

ProcessDelivery(List<FoodOrder> groceries) {
        for all FoodOrder foo in groceries
                Food temp = myFood.get(foo.food);
                temp.val+=foo.val;
                myFood.put(foo.food, temp);
        waitingForInventory=false;
        delivery.remove(groceries);
        if(!RestaurantIsOpen)
                boolean ready = true;
                for all FoodOptions s
                        if(myFood.get(s)!= initialFoodAmnt)
                                then ready=false
                if(ready)
                        host.KitchenIsReady();
                        RestaurantIsOpen=true;
```