DATA

```
boolean restaurantOpening = true; //For initial food ordering

List<Order> orders;

class Order {
        Waiter w;
        String choice;
        int table;
        OrderState s;
}

enum OrderState { pending, cooking, cooked, finished };

Timer timer; // For cooking times

Class Food {
        string type;
        int cookingTime;
        int amount;
        int low;
        int capacity;
        foodOrderingState state;
}

enum foodOrderingState { notYetOrdered, ordered };

map(string, Food) foods; // For cooking times of each food
```

MESSAGES

```
HereIsOrder(Waiter w, string choice, int table) {
        orders.add(new Order(w, choice, table, pending));
}

foodDone(Order o) {
        o.s = cooked;
}

WeWillDeliver(MarketAgent m, List<FoodOrder> orders) {
        for(FoodOrder fo : orders) {
                Food food = foods.get(fo.foodType);
```

```
                if(fo.amount < (food.capacity - food.amount)) {
                        food.state = notYetOrdered;
                }
                else {
                        food.state = ordered;
                }
        }
}

CannotFulfillOrder(List<FoodOrder> orders) {
        //switch to the next market
        for(FoodOrder fo : orders) {
                Food food = foods.get(fo.foodType);
                food.state = notYetOrdered;
        }
}

FoodDelivery(MarketAgent m, List<FoodOrder> orders) {
        for(FoodOrder fo : orders) {
                Food food = foods.get(fo.foodType);
                food.state = notYetOrdered;
                food.inventory += fo.amount;
        }
}
```

SCHEDULER

```
if restaurantOpening
        initialInventoryCheck();
if there is an o in orders such that o.s = cooked
        then plateIt(o);
if there is an o in orders such that o.s = pending
        then cookIt(o);
```

ACTIONS

```
CookIt(Order o) {
        Food thisFood = foods.get(o.choice);
        if thisFood.amount == 0
                then o.w.msgOutOf(o.choice, o.table);
                o.s = finished
                if thisFood.state != ordered
```

```
                then orderFood();
            return;

        if thisFood.amount < thisFood.low
            then orderFood();

        DoCooking(o); // Animation and print statements
        o.s = cooking;
        thisFood.amount--;

        timer.start( run(foodDone(o)), foods.get(o.choice).cookingTime);
}

PlateIt(Order o) {
        DoPlating(o); // Animation and print statements
        o.w.OrderDone(o.choice, o.table);
        o.s = finished;
}

InitialInventoryCheck() {
        if steak.amount < steak.low || fish.amount < fish.low || chicken.amount < chicken.low
            orderMoreFood;

        restaurantOpening = false;
}

orderMoreFood() {
        List<FoodOrder> orderList;

        if(steak.amount < steak.low && steak.state = notYetOrdered)
            then orderList.add(new FoodOrder("steak", steak.capacity - steak.amount);
            steak.state = ordered;

        if(fish.amount < fish.low && fish.state = notYetOrdered)
            then orderList.add(new FoodOrder("fish", fish.capacity - fish.amount);
            fish.state = ordered;

        if(chicken.amount < chicken.low && chicken.state = notYetOrdered)
            then orderList.add(new FoodOrder("chicken", chicken.capacity - chicken.amount);
            chicken.state = ordered;
        currentMarket.msgOrderFood(this, orderList);
}
```