Doreen Hakimi
CS201 Restaurant V2.1
Fall 2013

DATA

```
class MyCustomer {
        Customer c;
        int tablenum;
        enum state {none, waiting, seated, readyToOrder, ordered, needsToReorder,
couldNotAffordAndLeaving, gone, billed, waitingForFood, serveMe,
                served, gone }
        String choice;
}

List<myCustomer> customers;
Cook cook;
Host host;

Check myBill = null;
MyCustomer billRecipient = null;


//NOTE: FoodOrder is created as an outer class
class FoodOrder {
        String food;
        int val;
}

class Order {
        int tablenum;
        String choice;
        Waiter w;
        enum state= {pending, cooking, cooked, needsReOrder}
}

List<Order> orders;

boolean onBreak = false; //for now until we implement breaks

enum state = { working, goingToSeatCustomer, readyToSeatCustomer, seatingCustomer,
goingToTakeOrder, goingToCook, atKitchen, servingFood, foodAtHandAndAtTable,  atCashier,
onBreak, waitingForBill, billAtHand, goingToGiveBill, billAtHandAndAtTable, goingToGetBill};

boolean onBreak
boolean requestedBreak
boolean canTakeBreak
```

Doreen Hakimi
CS201 Restaurant V2.1
Fall 2013

boolean checkBoxReset = false
wantBreakChecked =false
returnFromWorkChecked = false
disableBoxTillBreak = false

//used states to deal with animations and transitions… will make more optimal for v3 by using more sempahores but am leaving out extra actions and scheduler calls that i added on to deal with gui.

MESSAGES

```
SitAtTable( table, customer) {
        customers.add(new myCustomer(customer, table, waiting)
}

ImReadyToOrder (Customer cust) {
        if there exists myCustomer c1 in customers such that c1.c=cust
                then c1.state=readyToOrder;
}
```

//note: created outer order class because having a waiter have the ability to take multiple orders to the cook instead of one is more realistic

```
HereIsMyChoice(Customer cust, String choice) {
        if there exists myCustomer c1 in customers such that c1.c=cust
        then c1.choice = choice;
              c1.state=ordered;
              orders.add(new Order(choice, c1.tablenum, this));
}

OutOfFood(Order o) {

        if there exists a MyCustomer customer in customers such that customer.tablenum ==
o.tablenum
                then customer.state=needsToReorder;
        if there exists an Order ord in orders such that ord==o
                then ord.state= needsToReorder;
}

OrderIsReady(Order o) {
        if there exists order in orders such that order=o
                then order.state=cooked;
}
```

Doreen Hakimi
CS201 Restaurant V2.1
Fall 2013

```
DoneEatingAndLeaving( Customer cust) {
        if there exists myCustomer c1 in customers such that c1.c = cust
                then c1.state=gone;
}

CantAffordNotStaying(CustomerAgent cust) {
        state=working;
        if there exists a MyCustomer customer in customerst such that customer==cust
                then customer.state=couldNotAffordAndLeaving
IWantABreak() { // comes from GUI
        wantBreakChecked=true;
}

BreakReply(boolean yn) {
if (yn)
        then canTakeBreak=true;
             diableBoxTillBreak=true;
else
        checkBoxReset=true;

OutOfBreak() { // also from GUI
        returnFromWorkChecked=true;
        onBreak=false;
}

HereIsACheck(int tnum, double amnt) {
        if there exists MyCustomer customer in customers sucht that customer.tablenum ==
tnum
                myBill = new Check(customer.choice, tnum, amnt, this);
                billRecipient = customer;
                state=billAtHand

OutOfFood(Order o) {
        if there exists myCustomer c1 in customers such that c1.tablenum=o.tablenum
                then c1.state=needsToReorder;
        if there exists Order order in orders such that order = o
                then order.state=needsReorder;
}

BreakReply(Boolean ans) {
        if (ans) then TakeBreakWhenFinished=true;
}
```

Doreen Hakimi
CS201 Restaurant V2.1
Fall 2013

SCHEDULER

```
if(!onBreak) {
if(wantBreakChecked)
        then AskHostForBreak();
if(returnFromWorkChecked)
        then ReturnToWork();
if(diableBoxTillBreak)
        then DisableCheckBox();
if(state=atCashier)
        if there exists an Order o in orders such that o.state=billPending
                then AskCashierForBill(o);
if(state=billAtHand)
        then TakeBillToCustomer();
if(state-billAtHandAndAtTable)
        then GiveBillToCustomer();
if state= readyToSeatCustomer
        then if there exists MyCustomer c in customers such that c.state=waiting
                then SeatCustomer(c);
if state= atKitchen
        then if there exists an Order o in orders such that o.state=pending
                then GiveCookOrders();
if there exists MyCustomer c in customers such that c.state=readyToOrder
        then GoToCustomer(c);
if there exists MyCustomer c in customers such that c.state=ordered
        then GoToCook();
if there exists MyCustomer c in customers such that c.state=couldNotAffordAndLeaving
        then UpdateHostOnClearTableAndLeave(c);
if there exists MyCustomer c in customers such that c.state=needsToReOrder
        then GoToCustomer(customer);
if there exist Order o in orders such that o.state=cooked;
        then ServeCustomer(o);
if there exists MyCustomer c in customers such that c.state=gone
        then UpdateHostOnClearTable(c);
if(checkBoxReset)
        then CouldNotTakeBreak();
if(canTakeBreak and customer.empty())
        then TakeBreak();
```

ACTIONS

```
TakeBreak() {
        onBreak=true;
```

```
        canTakeBreak=false;
        DoSetBoxToReturn();
}

ReturnToWork() {
        onBreak=false;
        canTakeBreak=false;
        host.BackToWork();
        DoResetCheckBox();
        returnFromWrokChecked=false;
}

private void AskCashierForBill(Order o) {
        if there exists MyCustomer c in customers such that c.tablenum==o.tablenum
        then casheir.ComputeBill(o.choice, o.tablenum, c.customer.name, this);
        o.state=billProcessed;
        state=waitingForBill;
}

SeatCustomer(MyCustomer cust) {
        cust.c.state=seated;
        state=seatingCustomer;
        cust.c.FollowMe(new Menu(), this);
        DoSeatCustomer(cust.c, cust.tablenum);
}

GoToCustomer(MyCustomer cust) {
        DoGoToTable(cust, cust.tablenum);
        cust.c.WhatWouldYouLike();
}

GoToCook() {
        DoGoToCook();
        if there exists an Order o in orders such that o.state=pending
                then cook.HereIsAnOrder(o);
        state=working;
}

ServeCustomer(Order o) {
        DoDisplayCookedLabel(o.choice, o.tablenum);
        o.state=serving;
        if there exists myCustomer cust such that cust.tablenum=o.tablenum
                then cust.state=serveMe
```

```
                    DoGoToTable(cust, o.tablenum);
                    cust.c.FoodIsServed();
                    cust.state=served;
                    state=working;
                    DoLeaveCustomer();
                    orders.remove(o);

}

ServeFood(MyCustomer cust, Order o) {
        cust.customer.FoodIsServed();
        cust.state=served;
        DoGoToCashier();
        o.state=billPending;
}

UpdateHostOnClearTable (MyCustomer cust) {
        host.TableIsClear(c.tablenum);
        DoClearTable();
        customers.remove(cust);
}


DisableCheckbox() {
        DoDisableCheckbox();
        disableBoxTillBreak=false;
}

TakeBillToCustomer() {
        billRecipient.customer.HereIsYourBill(myBill.amount);
        myBill=null;
        billRecipient.state=billed;
        billRecipient=null;
        state=working;
        if there exists an Order o in orders such that o.state=billProcessed
                then orders.remove(o);
        DoGoHangAtTheFront();

TakeBreak() {
        timer.start() {
                host.OffBreak();
        }
}
```