Cook implements ActionListener //because timer

Data	Messages

List<Order> orders;
String name;
 	
Map<String, Integer> foodDictionary = TreeMap<String, Integer>();
 	
**public class Order
{
Waiter waiter;
String choice;
Int tableNumber;
Enum OrderState { pending, cooking, cooked;}
Timer cookingTimer;
}
	HeresAnOrder(order)
	{
	orders.add(new Order(choice, table));
	}

Schedule	**Actions**

if ∃ an Order o in orders ∍ o.OrderState == pending
then CookOrder(o);
 	
if ∃ an Order o in orders ∍ o.OrderState == cooked
then tellWaiterOrderIsReady(order)
remove order;
	CookOrder(Order o){
	Do() // gui
	o.setTimer(); // starts the timer on the order.
	}

HostAgent

Data	Messages

List<Table> tables;
List<Waiter> waiters;
List<Customers> waitingCustomers;
 	
public class Tables //don't make this an inner class. It's such a pain in the ass.
{
  CustomerAgent occupiedBy;
  Int tableNumber;
  Int guiPosX;
  Int guiPosY;
}
	IWantToEat(Customer c){
	  waitingCustomers.add(c);
	}
	TableIsClear(Table t){
	  t.occupiedBy = null;
	}

Scheduler	**Actions**

if !waitingCustomer.empty() ∃ a Table t in tables ∍ t.occupiedBy == null and ∃ a Waiter w in waiters ∍ w.State == idle
  then notifyWaiter(t, w);
	notifyWaiter(Table t, Waiter w){
	  Do(); // or gui equivalent
	  w.SeatAtTable(waitingCustomer.remove(0), t);
	}

Waiters

Data	Scheduler

List<MyCustomer> myCustomers;
Cook cook;
Host host;
Boolean idle;
Int numberOfCustomers
 	
class MyCustomer {
Customer customer;
Table table;
Order order;
enum CustomerState {waiting, seated, readyToOrder, ordered}
}
	if ∃ an MyCustomer mc in myCustomers ∍ mc.State == waiting
	SeatCustomer(mc.table, mc);}
	if ∃ an MyCustomer mc in myCustomers ∍ mc.State == readyToOrder
	TakeOrder(mc);}
	if ∃ an MyCustomer mc in myCustomers ∍ mc.State == ordered
	GiveOrderToCook(mc);}
	if ∃ an MyCustomer mc in myCustomers ∍ mc.State == orderReady
	GiveFoodToCustomer(mc);}
	if ∃ an MyCustomer mc in myCustomers ∍ mc.State == doneEating
	CustomerLeaving(mc);}

Messages	**Actions**

- SeatAtTable(Customer c, Table t){
MyCustomer mc = new MyCustomer(c, t);
t.occupiedBy = c;
idle = true;
mc.state = waiting;
numberOfCustomers ++;
myCustomers.add(new MyCustomer(c, t));}
- ReadyToOrder(Customer c){
foreach MyCustomer mc in myCustomer{
if (mc == c){
mc.CustomerState = readyToOrder;
} }}
- HeresMyChoice(String c){
foreach MyCustomer mc in myCustomer{
if (mc == c){
mc.order = new Order(c, this, tableNumber);
mc.CustomerState = ordered;
} }}
- OrderIsReady(Order o){ foreach MyCustomer mc in myCustomer{
if (mc == c){
mc.state = orderReady
} }}
- ImDone(Customer c){ foreach MyCustomer mc in myCustomer{
if (mc == c){} mc.state = doneEating;}}
	- SeatCustomer(MyCustomer mc){
	mc.customer.FollowMe(new Menu());
	DoSeatCustomer(); //GUI
	mc.CustomerState = seated;
	WaiterState = idle;}
	
	- TakeOrder(MyCustomer mc){
	DoTakeOrder();
	mc.customer.WhatWouldYouLike();
	WaiterState = idle;}
	
	- GiveOrderToCook(MyCustomer mc){
	DoGiveOrderToCook();
	cook.HeresAnOrder(mc.order);
	WaiterState = idle;}
	
	- GiveFoodToCustomer(MyCustomer mc){
	DoGiveFoodToCustomer();
	mc.customer.HeresYourOrder(mc.order.choice);
	WaiterState = idle;}
	
	- CustomerLeaving(MyCustomer c){
	DoCustomerLeaving();
	host.TableIsClear(c.table);
	WaiterState = idle;}

Customer

Data	Scheduler

Host host;
Waiter waiter;
String choice;
Menu menu;
 	
Timer eatingTimer = new Timer();
 	
enum CustomerState {DoingNothing, WaitingInRestaurant, BeingSeated, Seated, Ordering, Eating, DoneEating, Leaving};
 	
enum CustomerEvent {none, gotHungry, followWaiter, seated, readyToOrder, ordered, doneEating, doneLeaving};
	if (state == CustomerState.*DoingNothing* && event == CustomerEvent.*gotHungry*){
	state = CustomerState .*WaitingInRestaurant*;
	goToRestaurant();}
	if (state == CustomerState .*WaitingInRestaurant* && event == CustomerEvent .*followWaiter*){ followWaiter();
	state = CustomerState .*BeingSeated*;}
	if (state == CustomerState .*BeingSeated* && event == CustomerEvent .*seated*){
	state = CustomerState .*ReadingMenu*;
	ChooseFood();}
	if (state == CustomerState .Seated && event == CustomerEvent .readyToOrder){
	state = CustomerState .Ordering;
	CallWaiter();}
	if (state == CustomerState.Ordering && event == CustomerEvent.ordered){
	state = WaitingForFood;
	TellWaiterMyChoice();
	if (state == CustomerState .*Eating* && event == CustomerEvent .*doneEating*){
	state = CustomerState .*Leaving*;
	leaveTable();}
	if (state == CustomerState .*Leaving* && event == CustomerEvent .*doneLeaving*){
	state = CustomerState .*DoingNothing*;}

Messages	**Actions**

IsHungry(){ DoIsHungry();
CustomerEvent = gotHungry; }
FollowMe(Menu m){
menu = m;
CustomerEvent = followWaiter;
}
//Get a message from customer GUI when we reach the table to handle animation. Once we reach the table set Customer State to seated.
WhatWouldYouLike(){
TellWaiterMyChoice();
}
HeresYourOrder(String order){
if order != choice then output Complain.
CustomerState = Eating;
EatFood();
}
	goToRestaurant(){
	DoGoToRestaurant();
	host.IWantToEat(this);
	}
	CallWaiter(){
	DoCallWaiter(); // GUI
	waiter.ReadyToOrder();
	}
	ChooseFood(){
	readMenuTimer.start();
	Timer readMenuTimer = new Timer(readingMenuTime,
	Public void actionPerformed(ActionEvent e){
	choice = random(menu);
	CustomerEvent = readyToOrder;
	}}
	EatFood(){
	DoEatFood(); // GUI stuff
	eatTimer.start(); //eatTimer sets AgentEvent.doneEating.
	}
	TellWaiterMyChoice(){
	waiter.HeresMyChoice(choice);

	CustomerEvent = ordered;}
	leaveTable(){
	DoLeaveTable(); //GUI
	CustomerState = leaving;
	//After you have left set AgentEvent.doneLeaving = true;
	waiter.ImDone(this);
	}