COOK AGENT

Data

```
private String name;
HostAgent host;

boolean PayingAttention = true;

public enum CookState {pending, cooking, done, out, outoffood};
Timer timer = new Timer();

public List<Order> allOrders
= new ArrayList<Order>();

public List<WaiterAgent> Waiters
= new ArrayList<WaiterAgent>();

public List<MarketAgent> Markets
= new ArrayList<MarketAgent>();
public List<Boolean> OutOfBeef = new ArrayList<Boolean>();
public List<Boolean> OutOfChicken = new ArrayList<Boolean>();
public List<Boolean> OutOfPizza = new ArrayList<Boolean>();
public List<Boolean> OutOfSalad = new ArrayList<Boolean>();
boolean orderedSteak=false;
boolean orderedChicken=false;
boolean orderedPizza=false;
boolean orderedSalad=false;


Inventory inventory = new Inventory();

//  Classes

private class Food {
        String choice;
        int cooktime;

        Food(String c) {
                choice=c;

                if (c == "Steak") {
```

```
                cooktime=10000;
            }
            else if (c == "Chicken") {
                cooktime=8000;
            }
            else if (c == "Salad") {
                cooktime=1000;
            }
            else {
                cooktime=5000;
            }
        }

        String getChoice() {
            return choice;
        }
}

private class Inventory {
        int AmountOfChicken;
        int AmountOfSteak;
        int AmountOfPizza;
        int AmountOfSalad;

        Inventory() {
            AmountOfChicken = 0;
            AmountOfSteak = 0;
            AmountOfPizza = 0;
            AmountOfSalad = 0;
        }

        public int GetAmountOf (String foodtype) {
            if (foodtype == "Steak") {
                return AmountOfSteak;
            }
            else if (foodtype == "Chicken") {
                return AmountOfChicken;
            }
            else if (foodtype == "Pizza") {
                return AmountOfPizza;
            }
            else {
                return AmountOfSalad;
```

```
		}
	}

	public void AddMore (String foodtype, int amount) {
		if (foodtype == "Steak") {
			AmountOfSteak = AmountOfSteak + amount;
		}
		else if (foodtype == "Chicken") {
			AmountOfChicken = AmountOfChicken + amount;
		}
		else if (foodtype == "Pizza") {
			AmountOfPizza = AmountOfPizza + amount;
		}
		else if (foodtype == "Salad") {
			AmountOfSalad = AmountOfSalad + amount;
		}
	}

	public void CookOneOf (String foodtype) {
		if (foodtype == "Steak") {
			AmountOfSteak--;
		}
		else if (foodtype == "Chicken") {
			AmountOfChicken--;
		}
		else if (foodtype == "Pizza") {
			AmountOfPizza--;
		}
		else if (foodtype == "Salad") {
			AmountOfSalad--;
		}
	}
}

private class Order {
	WaiterAgent waiter;
	//String choice;
	int table;
	CustomerAgent customer;
	CookState state;
	Food order;
```

```
Order(WaiterAgent Waiter, int tableNumber, String o) {
        this.table = tableNumber;
        this.waiter = Waiter;
        this.order=new Food(o);

        state=CookState.pending;
}

void setStateOutofFood() {
        state = CookState.outoffood;
}

void setStateCooking () {
        state = CookState.cooking;
}

void setStateDone () {
        state = CookState.done;
}

void setStateOut () {
        state = CookState.out;
}

CookState getState () {
        return state;
}

void setTableNumber (int n) {
        table = n;
}

int getTableNumber () {
        return table;
}

void setOrder (String o) {
        order.choice = o;
}

Food getOrder () {
        return order;
```

```
		}

		void setCustomer(CustomerAgent cust) {
			customer = cust;
		}

		CustomerAgent getCustomer() {
			return customer;
		}

		void setWaiter(WaiterAgent w) {
			waiter = w;
		}

		WaiterAgent getWaiter() {
			return waiter;
		}

	}
```

Messages

```
public void msgNewOrder(WaiterAgent waiter, int table, String order) {
		allOrders.add(new Order(waiter,table, order));
		stateChanged();
}

public void msgOrderFullfilled(String food, int amount) {

		if (food == "Steak") {
			orderedSteak=false;
		}
		else if (food == "Chicken") {
			orderedChicken=false;
		}
		else if (food == "Pizza") {
			orderedPizza=false;
		}
		else if (food == "Steak") {
			orderedSalad=false;
		}

		inventory.AddMore(food, amount);
```

```
                stateChanged();
}

public void msgCanNotFullfillOrder(String food, int amount, int available, MarketAgent m) {
                if (available > 0) inventory.AddMore(food, available);

                if (food == "Steak") {
                        orderedSteak=false;
                }
                else if (food == "Chicken") {
                        orderedChicken=false;
                }
                else if (food == "Pizza") {
                        orderedPizza=false;
                }
                else if (food == "Steak") {
                        orderedSalad=false;
                }

                for (int i = 0; i < Markets.size(); i++) {
                        if (m == Markets.get(i)) {
                                if (food == "Steak") {
                                        OutOfBeef.set(i, true);
                                        break;
                                }
                                else if (food == "Chicken") {
                                        OutOfChicken.set(i, true);
                                        break;
                                }
                                else if (food == "Pizza") {
                                        OutOfPizza.set(i, true);
                                        break;
                                }
                                else if (food == "Steak") {
                                        OutOfSalad.set(i, true);
                                        break;
                                }
                        }
                }
        stateChanged();
}
```

Scheduler

1.)
```
for (Order order : allOrders){
        if(order.state==CookState.done) {
                order.setStateOut();
                callWaiter(order);
                return true;
        }
}
```

2.)
```
if (PayingAttention==true && Markets.isEmpty()==false) {
        if (inventory.GetAmountOf("Steak") <= 1 && orderedSteak==false) {
                Restock("Steak");
                return true;
        }
        if (inventory.GetAmountOf("Chicken") <= 1 && orderedChicken==false) {
                Restock("Chicken");
                return true;
        }
        if (inventory.GetAmountOf("Pizza") <= 1 && orderedPizza==false) {
                Restock("Pizza");
                return true;
        }
        if (inventory.GetAmountOf("Salad") <= 1 && orderedSalad==false) {
                Restock("Salad");
                return true;
        }
}
```

3.)
```
for (Order order : allOrders){
        if(order.state==CookState.pending &&
inventory.GetAmountOf(order.order.getChoice()) == 0) {
                print("Out of " + order.order.getChoice());
                order.setStateOutofFood();
                GetMoreFood(order);

                return true;
        }
}
```

4.)
```
for (Order order : allOrders){
    if(order.state==CookState.pending &&
inventory.GetAmountOf(order.order.getChoice()) > 0) {
        order.setStateCooking();
        Cook(order);


        return true;
    }
}
```

5.)
```
return false;
```

Actions

```
private void GetMoreFood(Order o) {
    o.getWaiter().msgGetNewOrder(o.getCustomer());

    for (int i=0;i<Markets.size();i++){
    if(o.getOrder().getChoice() == "Steak" && OutOfBeef.get(i)==false &&
orderedSteak==false) {
        orderedSteak=true;
        Markets.get(i).msgNewOrders(o.getOrder().getChoice(), 3);
        break;
    }
    else if(o.getOrder().getChoice() == "Chicken" && OutOfChicken.get(i)==false &&
orderedChicken==false) {
        orderedChicken=true;
        Markets.get(i).msgNewOrders(o.getOrder().getChoice(), 3);
        break;
    }
    else if(o.getOrder().getChoice() == "Pizza" && OutOfPizza.get(i)==false &&
orderedPizza==false) {
        orderedPizza=true;
        Markets.get(i).msgNewOrders(o.getOrder().getChoice(), 3);
        break;
    }
    else if(o.getOrder().getChoice() == "Salad" && OutOfSalad.get(i)==false &&
orderedSalad==false) {
        orderedSalad=true;
        Markets.get(i).msgNewOrders(o.getOrder().getChoice(), 3);
```

```
                        break;
                    }

                }

                stateChanged();
        }

private void Restock(String type) {

        for (int i=0;i<Markets.size();i++){
                if(type == "Steak" && OutOfBeef.get(i)==false && orderedSteak==false) {
                        Markets.get(i).msgNewOrders(type, 3);
                        orderedSteak=true;
                        break;
                }
                else if(type == "Chicken" && OutOfChicken.get(i)==false &&
orderedChicken==false) {
                                Markets.get(i).msgNewOrders(type, 3);
                                orderedChicken=true;
                                break;
                            }
                else if(type== "Pizza" && OutOfPizza.get(i)==false && orderedPizza==false) {
                                Markets.get(i).msgNewOrders(type, 3);
                                orderedPizza=true;
                                break;
                        }
                else if(type == "Salad" && OutOfSalad.get(i)==false && orderedSalad==false) {
                                Markets.get(i).msgNewOrders(type, 3);
                                orderedSalad=true;
                                break;
                        }

                }

                stateChanged();
        }

        private void callWaiter(Order o) {
                o.getWaiter().msgOrderReady(o.order.getChoice(), o.getTableNumber());
                stateChanged();
        }
```

```
	private void Cook(final Order o) {
		inventory.CookOneOf(o.order.getChoice());
		print("There are " + inventory.GetAmountOf(o.order.getChoice()) +" " +
o.order.getChoice() + "left now");

		if (inventory.GetAmountOf(o.order.getChoice())==0) {
			Restock(o.getOrder().getChoice());
		}

		print("Started cooking " + o.order.getChoice());
		timer.schedule(new TimerTask() {
			Object cook = 1;
			public void run() {
				//look at menu, call waiter when ready
				o.setStateDone();
				print("Finished cooking " + o.order.getChoice());
				//waiter.msgReadyToOrder(temp);
				stateChanged();
			}
		},
		o.order.cooktime);

	}
```