COOK AGENT

Data

```
private String name;
        HostAgent host;

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

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

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

        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 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;
        }
}
```

Messages

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

Scheduler

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

2.)
```
for (Order order : allOrders){
        if(order.state==CookState.pending) {
                order.setStateCooking();
                Cook(order);
                return true;
        }
}
```

3.)
```
return false;
```

Actions

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

public void Cook(final Order o) {
        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);
}
```