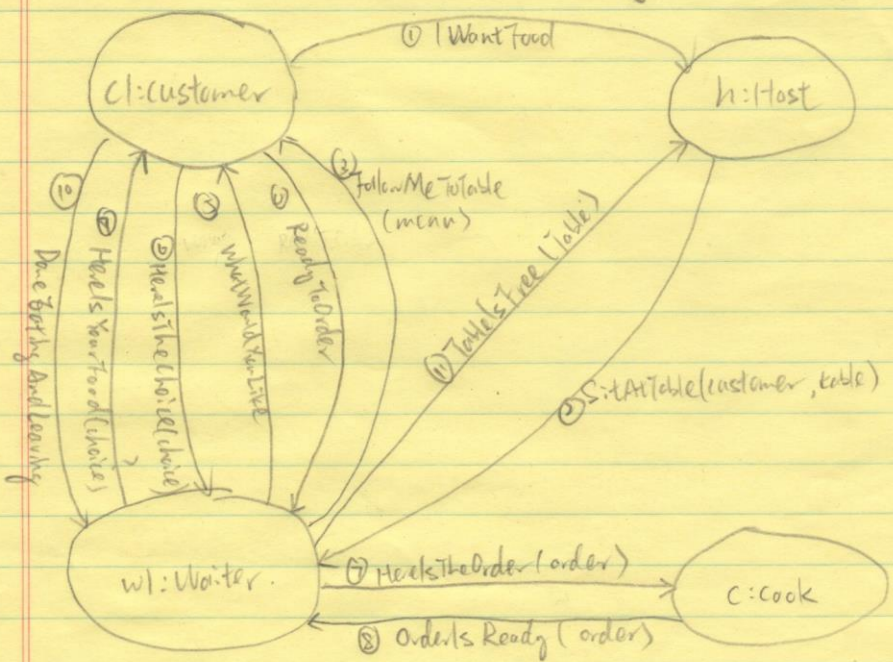
Interaction Diagram.



Waiter:

Data

```
List<MyCustomer> customers;
class MyCustomer {
    Customer c;
    int table;
    String food;
    CustomerState state;
}
enum CustomerState
none, waiting, seated, readyToOrder,
aboutToGiveOrder, orderGiven, orderProcessed,
orderReady, eating, doneEating
Host h;
Cook c;
```

Scheduler

```
if ∃ c in customers -> c.state=finished
    clearCustomer(c);
if ∃ c in customers -> c.state=waiting
    seatCustomer(c);
if ∃ c in customers -> c.state=readyToOrder
    askforchoice(c);
if ∃ c in customers -> c.state=orderGiven
    processOrder(c);
if ∃ c in customers -> c.state=orderReady
    giveOrderToCustomer(c);
```

Message

```
msgSitAtTable((Customer cust, int table){
    customers.add(new MyCustomer(cust,
table, waiting);
}
msgReadyToOrder((Customer cust){
    if(∃ customer in customers ->
    customer.c=cust && customer.state
    = seated)
customer.state = readyToOrder }
msgHereIsTheChoice((Customer cust, String choice){
    if(∃ customer in customers -> customer.c =
    cust && customer.state=aboutToGiveOrder)
customer.food = choice;
customer.state = orderGiven; }
msgOrderIsReady(Order o){
    if(∃ customer in customers -> customer.table=
    o.table && customer.state = orderProcessed)
customer.state = orderReady }
msgDoneEating((Customer cust){
    if(∃ customer in customers -> customer.c=
    cust && customer.state = eating
        customer.state = finished }
```

Action.

```
seatCustomer(MyCustomer c) {
    c.state = Seated;
    c.c.msgFollowMe(menu);
}
askForChoice(MyCustomer c) {
    c.state = aboutToGiveOrder
    c.c.msgWhatWouldYouLike();
}
processOrder(MyCustomer c) {
    c.state = order.Processed
    cook.msgHereIsTheOrder(c.choice, c.table)
}
clearCustomer(MyCustomer c) {
    host.msgTableIsFree(c.c, c.table);
    customers.remove(c);
}
giveOrderToCustomer(MyCustomer c) {
    c.state = eating;
    c.c.msgHereIsYourFood(c.choice);
}
```

Cook

Data

```
List<Order> orders;
class Order
    Waiter w,
    String choice,
    int table,
    OrderState state;
enum OrderState:
  none, pending, cooking, cooked
Timer timer;
```

Message

```
msgHereIsTheOrder(Waiter w, String
choice, int table) {
    orders.add(new Order(w, choice,
    table, OrderState.pending));
}
msgDone(Order o){
    o.state = cooked;
}
```

Scheduler

```
if ∃ order in orders -> order.state = pending
    cookOrder(order);
if ∃ order in orders -> order.state = cooked
    giveOrder(order);
```

Actions:

```
cookOrder(Order o){
    o.state = cooking;
    DoCooking();
    timer.start {
      msgDone(o) };
}
giveOrder(Order o) {
    o.w.msgOrderIsReady(o);
    orders.remove(o);
}
```

Host

Data

```
List<Customer> waitingCustomers
List<Table> tables
List<Waiter> waiters
class Table
  Customer c;
  int tableNumber;
```

Message

```
msgIWantFood(Customer cust){
    waitingCustomers.add(cust);
}
msgTableIsFree(Customer cust, int t_num){
    if ∃ table in tables ->
    table.tableNumber = t_num
    table.setUnOccupied();
}
```

Scheduler.

```
if ∃ table in tables ->
    table.isUnOccupied
  if waitingCustomers.isEmpty()
      seatCustomer(waitingCustomers.get(0),
          table);
```

Action

```
seatCustomer(Customer customer,
                Table t) {
    waiters.get(0).msgSitAtTable
    (customer, table.tableNumber);
    table.setOccupant(customer);
    waitingCustomers.remove(customer);
}
```

Customer

Data

```
String name;
Host h;
Waiter w;
Menu m;
State state;
Event event;
enum State
DoingNothing, Waiting, Seated,
ReadyToOrder, GivenOrder, Eating,
DoneEating, Leaving
enum Event
none, gotHungry, followHost, doneThinking,
orderFood, gotFood, doneEating, doneLeaving
```

Message

```
msgGotHungry() {
    event = gotHungry;
}
msgFollowMe(Waiter w, Menu m) {
    this.w = w;
    this.m = m;
    event = followHost;
}
msgWhatWouldYouLike() {
    event = orderFood;
}
msgHereIsYourFood(String choice) {
    event = gotFood;
}
```

Scheduler

```
if (state = DoingNothing && event = gotHungry)
    state = WaitingInRestaurant;
    goToRestaurant();
if (state = WaitingInRestaurant && event = followHost)
    state = Seated
    thinkAboutMenu();
if (state = seated && event = doneThinking)
    state = ReadyToOrder
    AskWaiterToPickUpOrder();
```

Actions:

```
goToRestaurant() {
    host.msg(WantFood();
}
thinkAboutMenu() {
    timer() {
        choice = menu.random();
        event = doneThinking
    }
}
AskWaiterToPickUpOrder() {
    w.msgReadyToOrder(); }
```

```
if (state = ReadyToOrder && event = orderFood)
    state = GiveOrder;
    GiveOrder();
if (state = GiveOrder && event = getFood)
    state = Eating;
    EatFood();
if (state = Eating && event = doneEating)
    state = Leaving;
    leaveTable();
```

```
GiveOrder(){
  w.msgHereIsTheChoice(choice);
}
EatFood(){
   timer(){
      event = doneEating;
   }
}
leaveTable(){
    w.msgDoneEating();
}
```