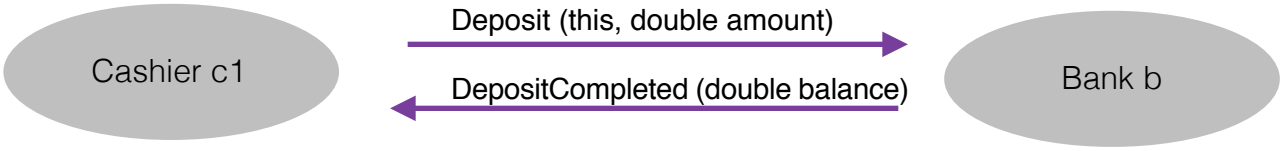
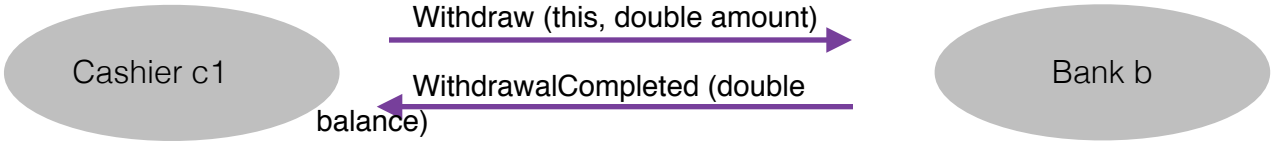
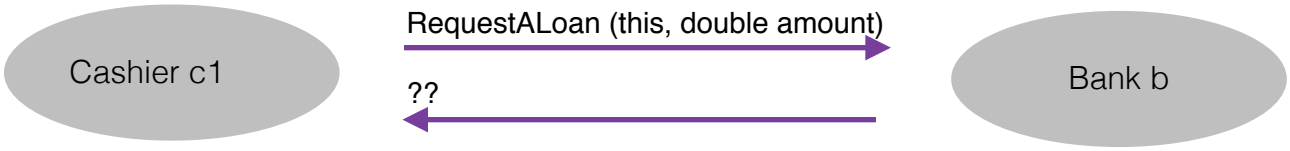
**Requirements:**

- Excess money must be deposited in a Bank
- The restaurant must have working capital for the day
- Workers in the restaurant are paid

**Interaction diagram: (do we still have CashierAgent?)**



**How the design works:**

- When closing the restaurant, Cashier would first pay the staff, then deposit money/ withdraw money/ make a loan, depending on the current bankBalance and workingCapital.

**Pseudo-Code**

**Data**:

```
Double bankBalance; // updated at the end of a day when Bank respond to the request
Double workingCapital; // updated whenever money is involved in daily restaurant activities
// for the above two instance variables, read from RestaurantPanel whenever a CashierRole is
// created OR reactivated;
Boolean onClose; // default is false, turn to true when timer expires (closing restaurant), and
// switch back to false when another timer expires (opening restaurant)
vector<WaiterRole> waiters;
HostRole host;
CookRole cook;
```

**Messaging:**

```
public void DepositCompleted (double balance) {
        bankBalance = balance;
}
public void WithdrawCompleted (double balance) {
        bankBalance = balance;
}
public void ?? // how bank would reply to loan request
```

**Scheduler:**

```
PickAndExecuteAnAction(){
```

```
        if (onClose) {
                closeRestaurant()
        }
        // Other scheduler rules
}
```

**Actions:**

```
public void closeRestaurant() {
        // calculate the total amount that restaurant staff gets paid
        double total = waiterSalary*waiters.size() + hostSalary + cookSalary + cashierSalary;
        for every WaiterRole w in waiters,
                w.getPerson().getPaid (waiterSalary);
        host.getPerson().getPaid (hostSalary);
        cook.getPerson().getPaid(cookSalary);
        this.getPerson().getPaid(cashierSalary);
        workingCapital = workingCapital - total;

        if (workingCapital > 200) // 200 is the minimum workingCapital to open the restaurant the next day
                bank.deposit (workingCapital - 200);
                workingCapital = 200;
        if (workingCapital = 200)
                // do nothing
        if ( workingCapital < 200 )
                if (bankBalance >= (200 - workingCapital))
                        bank.Withdraw (200 - workingCapital)
```

```
                    workingCapital = 200;
                else
                bank.RequestALoan ( 200 - (workingCapital + bankBalance) );
        // update data in RestaurantPanel
        restaurantPanel.setBankBalance(bankBalance);
        restaurantPanel.setWorkingCapital(workingCapital);
}
```

how should one day at simcity look like?

for shared-data between cook and waiter:

in RestaurantPanel.java

```
class Order {
        int table;
        String food;
        SharedDataWaiterAgent waiter;
        Order (int t, String f, SharedDataWaiterAgent w) {
                table  = t;
                food = f;
                waiter = w;
        }
}

class CookWaiterMonitor extends Object {
        private Vector<Order> orders;
        private Vector<Order> completed_orders;
        synchronized public void addOrder (int table, String food, SharedDataWaiterAgent
waiter) {
                orders.add (new Order (table, food, waiter))
                cook.notify();
        }
        synchronized public Order removeOrder () {
                cook.wait();
```

```
                Order temp;
                temp = orders.firstElement();
                orders.removeElementAt(0);
                return temp;
        }
        synchronized public void addCompletedOrder (int table, String food,
SharedDataWaiterAgent waiter) {
                completed_orders.add (new Order(table, food, waiter));
                waiter.notify();
        }
        synchronized public Order removeCompletedOrder (int table, SharedDataWaiterAgent
waiter) {
                waiter.wait();
                Order temp;
                if there exists Order o in completed_orders such that o.table = table
                        temp = o;
                        completed_orders.remove(o);
                return temp;
        }
}

private CookWaiterMonitor theMonitor;
CookAgent c = new CookAgent (“cook”, theMonitor);
SharedDataWaiterAgent w = new SharedDataWaiterAgent(name, theMonitor);
// constructors need to be changed respectively
```

```
Abstract Class WaiterAgent extends Agent {
        // same data
        // same message except for orderIsReady(int table) from cook
        // same actions except for placeOrder(MyCustomer customer)
}
public class NormalWaiterAgent extends WaiterAgent implements Waiter {
        // this is the normal type of waiter that we've been working on so far
        // everything remains the same.
}

public class SharedDataWaiterAgent extends WaiterAgent implements Waiter {
        // same data and scheduler rules as the NormalWaiterAgent
        // Action placeOrder needs to be changed:
        public void placeOrder(MyCustomer customer) {
                print ("Here's an order for table " + customer.tableNumber);
                cook.msgHereIsAnOrder (customer.choice, this, customer.tableNumber);
                theMonitor.addOrder(customer.table, customer.food, this);
                customer.state = customerState.waitingForFood;
                theMonitor.removeCompletedOrder(this, customer.table);
                customer.state = customerState.FoodIsReady;
        }
}
```

in CookAgent class:

```
public Boolean pickAndExecuteAnAction() {
        Order order = new Order(theMonitor.removeOrder());
        // other scheduler rules
        if (order) {
                orders.add(new myOrder(order.table, order.food, order.waiter))
                return true; // or just cook it?
        }
}
```