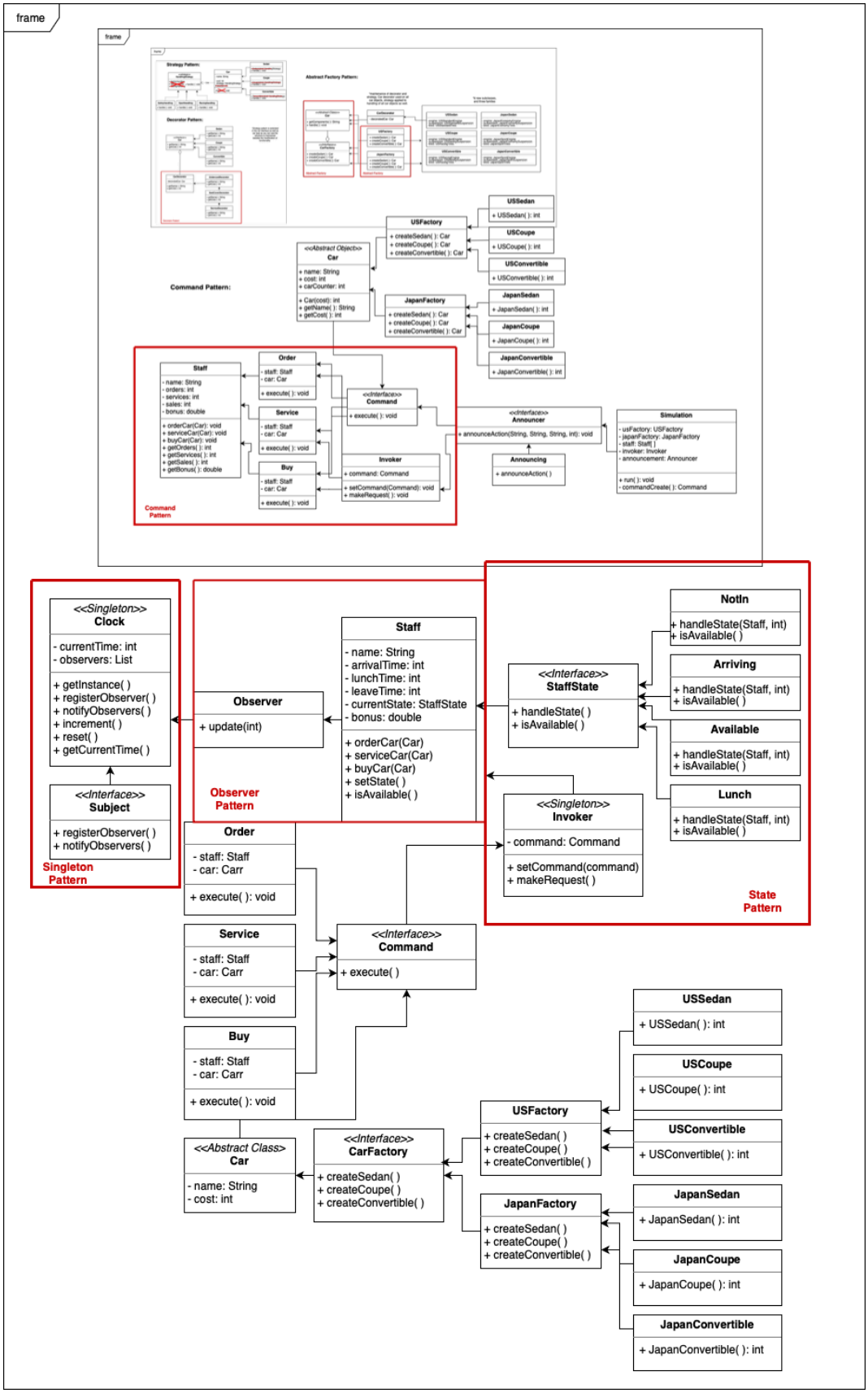
UML Class Diagram:



Source Code:

```
package org.coursera.lab.capstone;

import java.util.ArrayList;



// add your classes to extend your prior command code
// to be able to use the state, singleton, and observer patterns
//
// see below for the main client code from the example
// use something similar for your own testing and for
// structuring your classes and subclasses
//
// see the test code for the autograder expectations

/*
* Your solution for OOAD Course 2 Capstone
* Kate Pendavinji
* April 25,2025
*/

public class Main {
    public static void main(String[] args) {
        // use the main to invoke the simulation
        Simulation s = new Simulation();
        s.run();
    }
}

/*********************************************/
//////        State Pattern Interface       /////
/*********************************************/
interface StaffState {
    void handleState(Staff staff, int currentTime);
    boolean isAvailable();
}


/*********************************************/
//////      Command Pattern Interface      //////
/*********************************************/
/*
 * Command implementation should create an interface with a
 * standard execute method
 */
interface Command {
    void execute();
}

class Order implements Command {
    private Staff staff;
    private Car car;

    public Order(Staff staff, Car car) {
        this.staff = staff;
        this.car = car;
    }

    public void execute() {
        staff.orderCar(car);
    }
}

/*********************************************/
//////           Command Pattern           //////
/*********************************************/
class Service implements Command {
    private Staff staff;
    private Car car;
```

```
    public Service(Staff staff, Car car) {
        this.staff = staff;
        this.car = car;
    }

    public void execute() {
        staff.serviceCar(car);
    }
}

class Buy implements Command {
    private Staff staff;
    private Car car;

    public Buy(Staff staff, Car car) {
        this.staff = staff;
        this.car = car;
    }

    public void execute() {
        staff.buyCar(car);
    }
}

/*********************************************/
//////          Singleton Pattern          /////
/*********************************************/
class Invoker {
    private Command command;

    private static Invoker instance;

    private Invoker() {
    }

    public static Invoker getInstance() {
        if (instance == null) {
            instance = new Invoker();
        }
        return instance;
    }
    public void setCommand(Command command) {
        this.command = command;
    }

    public void makeRequest() {
        command.execute();
    }
}

/*********************************************/
//////          Observer Pattern          //////
/*********************************************/
class Staff implements Observer {
    private String name;
    private int orders;
    private int services;
    private int sales;
    private double bonus;

    private int arrivalTime;
    private int lunchTime;
    private int leaveTime;
    private StaffState currentState;

    /*
     * The Staff class should be extended to include attributes for
     * an integer arrival time, lunch time, and leave time.
     */
    public Staff(String name, int arrival, int lunch, int leave) {
        this.name = name;
        this.arrivalTime = arrival;
        this.lunchTime = lunch;
        this.leaveTime = leave;
        this.currentState = new NotIn();
    }

    /*********************************************/
    //////             State Pattern           /////
```

```java
/*******************************************/
/*
 * Staff will also be extended to use a State pattern and will keep a
 * CurrentState attribute.
 */
public void update(int currentTime) {
    currentState.handleState(this, currentTime);
}

public void setState(StaffState state, int currentTime) {
    this.currentState = state;
    // account for time on standard time
    if (currentTime <= 12) {
        System.out.println(name + " moved to " + state.getClass().getSimpleName() + " at " + currentTime);
    } else {
        System.out.println(name + " moved to " + state.getClass().getSimpleName() + " at " + (currentTime - 12));
    }
}

/*
 * Staff will also be extended to use a State pattern and will keep a
 * CurrentState attribute.
 */
public String getCurrentState(){
    return currentState.getClass().getSimpleName();
}

public boolean isAvailable() {
    return currentState.isAvailable();
}

public int getArrivalTime() {
    return arrivalTime;
}
public int getLunchTime() {
    return lunchTime;
}

public int getLeaveTime() {
    return leaveTime;
}
/*
 * staff objects will have three methods thay can perform:
 * orderCar (bonus is 3% of selected car cost), serviceCar
 * (bonus is 1% car cost), buyCar (bonus is 2%)
 */
 /*********************************************/
 //////      Maintain Command Pattern       //////
 /*********************************************/

public void orderCar(Car car) {
    double addBonus = car.getCost() * 0.03;
    orders++;
    bonus += addBonus;
}

public void serviceCar(Car car) {
    double addBonus = car.getCost() * 0.01;
    services++;
    bonus += addBonus;
}

public void buyCar(Car car) {
    double addBonus = car.getCost() * 0.02;
    sales++;
    bonus += addBonus;
}

/*
 * staff objects should track the number of times these methods are called
 * and the total bonus earned from performing the method
 */
public String getName() {
    return name;
}

public int getOrders() {
    return orders;
}
```

```
    public int getServices() {
        return services;
    }

    public int getSales() {
        return sales;
    }

    public double getBonus() {
        return bonus;
    }
}

/*********************************************/
//////             State Pattern             /////
/*********************************************/
/*
 * Possible staff states are NotIn, Arriving, Available, and Lunch. Staff start
 * each day in the NotIn state
 */
class NotIn implements StaffState {
    public void handleState(Staff staff, int currentTime) {
        if (currentTime == staff.getArrivalTime()) {
            staff.setState(new Arriving(), currentTime);
        }
    }
    public boolean isAvailable() {
        return false;
    }
}

class Arriving implements StaffState {
    public void handleState(Staff staff, int currentTime) {
        if (currentTime > staff.getArrivalTime()) {
            staff.setState(new Available(), currentTime);
        }
    }

    public boolean isAvailable() {
        return false;
    }
}

class Available implements StaffState {
    public void handleState(Staff staff, int currentTime) {
        if (currentTime == staff.getLunchTime()) {
            staff.setState(new Lunch(), currentTime);
        }
        else if (currentTime >= staff.getLeaveTime()) {
            staff.setState(new NotIn(), currentTime);
        }
    }
    public boolean isAvailable() {
        return true;
    }
}

class Lunch implements StaffState {
    public void handleState(Staff staff, int currentTime) {
        if (currentTime > staff.getLunchTime() && currentTime < staff.getLeaveTime()) {
            staff.setState(new Available(), currentTime);
        } else if (currentTime >= staff.getLeaveTime()) {
            staff.setState(new NotIn(), currentTime);
        }

    }

    public boolean isAvailable() {
        return false;
    }
}

/*
 * The Clock class will also implement the publishing or Subject side of
 * Observer, maintaining a list of observers or subscribers, and
 * it will provide registerObserver and notifyObservers methods.
 */
interface Subject {
```

```
    void registerObserver(Observer observer);
    void notifyObservers();
}

/*
 * provide updated times to staff instances
 */
interface Observer {
    void update(int currentTime);
}

/*********************************************/
//////          Observer Pattern            /////
/*********************************************/
class Clock implements Subject {
    /*********************************************/
    ////// Singleton Pattern /////
    /*********************************************/
    private static final Clock instance = new Clock();

    private int currentTime;
    private ArrayList<Observer> observers = new ArrayList<>();

    private Clock(){
        currentTime = 8;
    }

    public void increment() {
        currentTime++;
    }

    /*
     * Use for simulation should receive
     * times for 8-12 and 1-7
     */
    public boolean isWorkHour(){
        return ((currentTime >= 8 && currentTime <= 12) || (currentTime >=1 && currentTime <7));
    }
    public void registerObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(currentTime);
        }
    }

    /*********************************************/
    //////           Singleton Pattern          /////
    /*********************************************/
    public static Clock getInstance(){
        return instance;
    }

    public int getCurrentTime() {
        return currentTime;
    }
    public void reset(){
        currentTime = 8;
    }
}

/*********************************************/
//////           Abstract Factory           /////
/*********************************************/
interface CarFactory {
    Car createSedan();
    Car createCoupe();
    Car createConvertible();
}

/*********************************************/
//////           Abstract Object            /////
/*********************************************/
abstract class Car {
    private String name;
    private int cost;
    private static int carCounter = 0;
```

```
    public Car(int cost) {
        this.cost = cost;
        carCounter++;
        this.name = getClass().getSimpleName() + " " + carCounter;
    }

    public String getName() {
        return name;
    }

    public int getCost() {
        return cost;
    }
}

class USFactory implements CarFactory {
    /*************************************************/
    //////        Concrete Car Instances       //////
    //////   USSedan, USCoupe, USConvertible  //////
    /*************************************************/
    public Car createSedan() {
        return new USSedan();
    }

    public Car createCoupe() {
        return new USCoupe();
    }

    public Car createConvertible() {
        return new USConvertible();
    }
}

/*****************************************************/
//////     Three Car Subclasses for USFactory      //////
/*****************************************************/

class USSedan extends Car {
    public USSedan() {
        super(13000);
    }
}

class USCoupe extends Car {
    public USCoupe() {
        super(18000);
    }
}

class USConvertible extends Car {
    public USConvertible() {
        super(23000);
    }
}

class JapanFactory implements CarFactory {
    /*************************************************/
    //////        Concrete Car Instances       //////
    /// JapanSedan, JapanCoupe, JapanConvertible ///
    /*************************************************/
    public Car createSedan() {
        return new JapanSedan();
    }

    public Car createCoupe() {
        return new JapanCoupe();
    }

    public Car createConvertible() {
        return new JapanConvertible();
    }
}

/*****************************************************/
//////     Three Car Subclasses for JapanFactory   //////
/*****************************************************/
class JapanSedan extends Car {
    public JapanSedan() {
```

```
        super(10000);
    }
}

class JapanCoupe extends Car {
    public JapanCoupe() {
        super(15000);
    }
}

class JapanConvertible extends Car {
    public JapanConvertible() {
        super(20000);
    }
}

class Simulation {
    // create factories for car builds
    private CarFactory usFactory = new USFactory();
    private CarFactory japanFactory = new JapanFactory();

    private CarFactory[] factories = { usFactory, japanFactory };

    private class SedanCreate implements CarCreate {
        public Car create(CarFactory factory) {
            return factory.createSedan();
        }
    }

    private class CoupeCreate implements CarCreate {
        public Car create(CarFactory factory) {
            return factory.createCoupe();
        }
    }

    private class ConvertibleCreate implements CarCreate {
        public Car create(CarFactory factory) {
            return factory.createConvertible();
        }
    }

    private interface CarCreate {
        Car create(CarFactory factory);
    }

    private CarCreate[] carCreate = {
            new SedanCreate() {
                public Car create(CarFactory f) {
                    return f.createSedan();
                }
            },
            new CoupeCreate() {
                public Car create(CarFactory f) {
                    return f.createCoupe();
                }
            },
            new ConvertibleCreate() {
                public Car create(CarFactory f) {
                    return f.createConvertible();
                }
            }
    };
    /*
     * When performing one of these methods, the action should be
     * announced in the console
     */

    interface Announcer {
        void announceAction(String staffName, String action, String carName, int cost);
    }

    private Announcer announcement = new Announcer() {
        public void announceAction(String staffName, String action, String carName, int cost) {
            System.out.println(staffName + " is " + action + " " + carName + " for $" + cost);
        }
    };
    private Command commandCreate(Command command, Staff staff, Car car) {
        if (command instanceof Order) {
            announcement.announceAction(staff.getName(), "ordering", car.getName(), car.getCost());
```

```
            return new Order(staff, car);
        } else if (command instanceof Service) {
            announcement.announceAction(staff.getName(), "servicing", car.getName(), car.getCost());
            return new Service(staff, car);
        } else {
            announcement.announceAction(staff.getName(), "buying", car.getName(), car.getCost());
            return new Buy(staff, car);
        }
    }

    private Command[] commands = {
            new Order(null, null),
            new Service(null, null),
            new Buy(null, null)
    };

    /*
     * automated random command and car creation enhancement of p4
     */
    private Command getRandomCommand() {
        return commands[(int) (Math.random() * commands.length)];
    }

    private Car createRandomCar() {
        CarFactory factory = factories[(int) (Math.random() * factories.length)];
        CarCreate randomCar = carCreate[(int) (Math.random() * carCreate.length)];
        return randomCar.create(factory);
    }

    public void run() {

        // get singleton references to Clock and Invoker
        Clock clock = Clock.getInstance();
        Invoker invoker = Invoker.getInstance();
        // initialize Staff and have them subscribe to clock published events
        Staff[] staff = {
            new Staff("Ann", 8, 12, 4),
            new Staff("Bob", 9, 1, 5),
            new Staff("Cal", 10, 2, 6),
            new Staff("Deb", 11, 3, 7)
        };

        for (Staff s : staff) {
            clock.registerObserver(s);
        }
        // Simulate 10 days of commands
        for (int day = 1; day <= 10; day++) {
            System.out.println("Day " + day);
            clock.reset();
            // increment the clock and let it notify the staff of new time
            while (clock.getCurrentTime() < 19) {
                /*
                 * Each clock time change should be announced to the
                 * console
                 */
                boolean activity = false;
                for (Staff s : staff) {
                    String state = s.getCurrentState();
                    if (!state.equals("NotIn")) {
                        activity = true;
                        break;
                    }
                }
                // only notify if staff is active...
                if(activity){
                    int timeNow = clock.getCurrentTime();
                    if (timeNow < 12) {
                        System.out.println("Time now: " + timeNow + " AM");
                    }
                    else if(timeNow == 12){
                        System.out.println("Time now: 12 PM");
                    }
                    else {
                        System.out.println("Time now: " + (timeNow-12) + " PM");
                    }
                }
                // (this uses Observer to drive State updates)
                clock.notifyObservers();
                // no one works at 7 PM, otherwise handle this hour's customer
```

```
                if (clock.isWorkHour()) {
                    // pick a staff member and check their state
                    for (Staff s : staff) {
                        // if available, keep going
                        if (s.isAvailable()) {
                            // pick a random command
                            Command randomCommand = getRandomCommand();
                            // create a random car instance
                            Car randomCar = createRandomCar();
                            // if not available, pick next staff member
                            Command command = commandCreate(randomCommand, s, randomCar);
                            // have invoker set and make the request (command)
                            invoker.setCommand(command);
                            invoker.makeRequest();
                        }
                        else {
                            System.out.println("Sorry, " + s.getName() + " is Not In");
                        }
                    }
                }
                // continue clock increments until 7 PM
                clock.increment();
            }
        }
        // Display results by staff member
        System.out.println("Results:");
        for (Staff s : staff) {
            System.out.println(s.getName() + ":");
            System.out.println("Orders: " + s.getOrders());
            System.out.println("Services: " + s.getServices());
            System.out.println("Sales: " + s.getSales());
            System.out.println("Bonus: $" + s.getBonus());
            System.out.println();
        }
    }
}
```

Testing:

```
package org.coursera.lab.capstone;

import static org.junit.jupiter.api.Assertions.assertTrue;

import org.coursera.lab.capstone.Main;
import org.junit.jupiter.api.Test;

public class MainTest {

    // use this for your own testing or to try what the autograder will perform
    /*
    @Test
    public void commandTest() {
        assertTrue(true);
    }*/

    /*
     * // this is the autograding code looking for a valid
     * // simulation object after a run
     *
     * @Test
     * public void capstoneTest() {
     * Simulation s = new Simulation();
     * s.run();
     * assertTrue(s instanceof Simulation);
     * }
     */


    @Test
        public void capstoneTest() {
        Simulation s = new Simulation();
        s.run();
        assertTrue(s instanceof Simulation);
```

```
        }
}
```

Results:

```
coder@226c6fbbd0dd:~/project/learn/src/main/java$ java org.coursera.lab.capstone.Main
Day 1
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is ordering USSedan 1 for $13000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is buying JapanCoupe 2 for $15000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering USConvertible 3 for $23000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is buying JapanSedan 4 for $10000
Time now: 1 PM
Deb moved to NotIn at 1
Day 2
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is servicing USConvertible 5 for $23000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is servicing USCoupe 6 for $18000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
```

```
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering JapanSedan 7 for $10000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is buying USCoupe 8 for $18000
Time now: 1 PM
Deb moved to NotIn at 1
Day 3
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is buying JapanCoupe 9 for $15000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is ordering JapanConvertible 10 for $20000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is servicing USSedan 11 for $13000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is buying USSedan 12 for $13000
Time now: 1 PM
Deb moved to NotIn at 1
Day 4
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is servicing USSedan 13 for $13000
```

```
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is ordering JapanConvertible 14 for $20000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is servicing USConvertible 15 for $23000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is buying USSedan 16 for $13000
Time now: 1 PM
Deb moved to NotIn at 1
Day 5
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is ordering JapanConvertible 17 for $20000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is buying USCoupe 18 for $18000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering JapanSedan 19 for $10000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
```

```
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering USCoupe 27 for $18000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is servicing JapanCoupe 28 for $15000
Time now: 1 PM
Deb moved to NotIn at 1
Day 8
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is buying JapanConvertible 29 for $20000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is servicing JapanSedan 30 for $10000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering JapanSedan 31 for $10000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is ordering JapanSedan 32 for $10000
Time now: 1 PM
Deb moved to NotIn at 1
Day 9
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
```

```
Ann is servicing USSedan 33 for $13000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is buying JapanConvertible 34 for $20000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering JapanSedan 35 for $10000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Deb is ordering USSedan 36 for $13000
Time now: 1 PM
Deb moved to NotIn at 1
Day 10
Ann moved to Arriving at 8
Sorry, Ann is Not In
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 9 AM
Ann moved to Available at 9
Bob moved to Arriving at 9
Ann is ordering USSedan 37 for $13000
Sorry, Bob is Not In
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 10 AM
Ann moved to NotIn at 10
Bob moved to Available at 10
Cal moved to Arriving at 10
Sorry, Ann is Not In
Bob is ordering USCoupe 38 for $18000
Sorry, Cal is Not In
Sorry, Deb is Not In
Time now: 11 AM
Bob moved to NotIn at 11
Cal moved to Available at 11
Deb moved to Arriving at 11
Sorry, Ann is Not In
Sorry, Bob is Not In
Cal is ordering USCoupe 39 for $18000
Sorry, Deb is Not In
Time now: 12 PM
Cal moved to NotIn at 12
Deb moved to Available at 12
Sorry, Ann is Not In
Sorry, Bob is Not In
```

```
Sorry, Cal is Not In
Deb is buying USSedan 40 for $13000
Time now: 1 PM
Deb moved to NotIn at 1
Results:
Ann:
Orders: 3
Services: 4
Sales: 3
Bonus: $3080.0

Bob:
Orders: 4
Services: 2
Sales: 4
Bonus: $3680.0

Cal:
Orders: 8
Services: 2
Sales: 0
Bonus: $4020.0

Deb:
Orders: 3
Services: 1
Sales: 6
Bonus: $2990.0
```