Lab 10

Due: 14 May (but see below)

Handout 10
CSCI 136: Spring, 2004
11 May

A day out shopping

1 Introduction

You are to write a program that will simulate the day of several shoppers shopping in a multi-story store. This will be a discrete simulation. Discrete simulations are based on having an event queue, where the events on the queue are ordered by the time that they are supposed to happen.

The kinds of events that will happen in this simulation include:

1. An elevator will arrive at a floor to discharge and load passengers.
2. An elevator will leave a floor to go to the next floor
3. A shopper will finish shopping on a floor and line up to wait for an elevator to go to the next floor they wish to visit.

In a discrete simulation, time advances only when an event is taken from the event queue. For example, it may be time 100 now, but if the next event on the event queue is supposed to take place at time 115 then time should immediately change to that when the event is removed from the queue.

Your job in this program is to print out a description of what is happening during the day. I will provide most of the classes that you will need for this program, leaving you to provide three of those with the most interesting data structures.

Remember that you may work in pairs on this project. If you work with someone else please put both names on your classes and only turn in one copy of the program.

2 The Classes

Simulation. *(Provided).* This class sets up the simulation by creating a building and the shoppers and then telling the building to start the simulation. It only has a main method. It should be executed to run the program.

Building. *(Provided).* This class represents a building with elevators. The building is European, so the bottom floor is floor 0 :-) The constructor takes the following parameters:

- number of floors in the building
- the number of elevators.
- the event queue,
- the capacity of elevators,
- the amount of time elevators should wait on each floor before proceeding to the next
- the amount of time elevators take to go from floor to floor

The methods include:

```
    // add new shopper to store on given floor.
    public void addShopper(Person waiter, int floor)

    // Record that there is now one fewer shopper
    public void stopShopping()

    // Add waiter to appropriate queue for floor
    public void addWaiting(Person waiter, int floor)

    // return list of waiters on floor wanting to go in direction given by up
    public Queue getWaiters(int floor,boolean up)

    // number of floors in building
    public int numberOfFloors()

    // start simulation by posting arrival events for elevators.
    public void startSimulation()
```

Event. *(Provided).* This class represents objects put on the event queue. They represent three different kinds of events: elevator arrival events, elevator departure events, and person done shopping events. Methods are available to return the object the event is happening to (elevator or person), the type of event, and the floor number on which it happens. The actual objects stored on the event queue will be of type `ComparableAssociation`, where the key will be the time of the event (held as an object of type `Integer`). The value will be an event. The important public features are:

```
public class Event {
    // Codes for kinds of events
    public static final int ELEVATOR_ARRIVAL = 0;
    public static final int ELEVATOR_DEPARTURE = 1;
    public static final int STOP_SHOPPING = 2;

    /**
     * Create event
     * @param subject -- who or what it happens to
     * @param eventType -- type of event
     * @param floorNum -- floor number on which it happens
     */
    public Event(EventSubject subject, int eventType, int floorNum)

    // Return the type of event
    public int getType()

    // Return the subject event is happening to
    public EventSubject getSubject()

    // Return floor on which the event will happen
    public int getFloor()
}
```

EventHandler. You will write this class that keeps track of all of the pending events. Here is an outline of the constructor and methods:

```
public class EventHandler {

    // Create event handler data structure
```

```
    public EventHandler()

    // Add arrival event for elevator to arrive at floor floorNum at arrivalTime
    public void addElevatorArrivalEvent(Elevator elevator, int floorNum,
            int arrivalTime)

    // Add departure event for elevator to leave floor floorNum at departTime
    public void addElevatorDepartureEvent(Elevator elevator, int floorNum,
            int departTime)

    // Add stop shopping event for shopper on floor floorNum at doneTime
    public void addDoneShoppingEvent(Person shopper, int floorNum, int doneTime)

    // return next event from event queue
    public ComparableAssociation getNextEvent()

    // Are there no more events left?
    public boolean isEmpty()
}
```

The events need to come off in the order that they are to happen. As you can see from the signature of the `getNextEvent` method, the elements stored in the event queue are `ComparableAssociation`s, where the key is the time (of type `Integer`, and the value is of type `Event`. (This should help explain why events don't have a time field.)

EventSubject. *(Provided).* This is an interface for objects that are subjects of events (e.g., elevators and people). It provides the method:

```
        // Do whatever is necessary to handle this event in simulation
        // Event should happen at timeNow on floor, and type of event is eventType
        void handleEvent(int floor, int timeNow, int eventType);
```

When an event is taken out of the event queue it should be unpacked and then the `handleEvent` message should be sent to the subject of the event, which should handle it properly.

Person. You will write this class (which implements `EventSubject`). It is to represent a person shopping in the store. These are some of the public features that you will need to represent a person.

```
    /** Create person
     * @param name -- name of Person
     * @param itinerary -- list of floors where need to shop
     * @param busyTimeSecs -- how long it takes to shop on each floor
     * @param startingFloor -- where enter building
     * @param building -- where they are shopping
     * @param events -- event queue
     * Constructor also makes sure that itinerary includes only legal floors
     */
    public Person(String name, int[] itinerary,int busyTimeSecs,
                    int startingFloor, Building building, EventHandler events)

    // return name of person
    public String toString()

    // Where should they go next in their itinerary
    public int getNextFloor(){
```

```
/** Do whatever is necessary to handle this event in simulation
 * Event should happen at timeNow on floor, and type of event is eventType.
 * In this case it is an event indicating that the shopper is done
 * shopping for now.  If they have more to do, put them in line for
 * the elevator.
 */
public void handleEvent(int floor, int timeNow, int eventType)
```

Notice that when a person stops shopping they should go stand in line to wait for an elevator to go to the next floor on their itinerary. (Of course it does this by generating an event for that when it starts shopping.)

It will be simplest if you imagine that the building has two lines on each floor for those waiting for elevators. One is for those people who wish to go up, and the other is for those who wish to go down. Remember that each elevator has a capacity, so people should go off the line in the same order they came.

You might want to keep extra information with the person, so that you can tell what the person is actually doing at any time (e.g., shopping, waiting for an elevator, or riding in an elevator).

Elevator. You will be writing the `Elevator` class, which also implements `EventSubject`. The `Elevator` class is the key to the entire simulation. The constructor gives the starting floor. We will assume that it always starts out going up, unless it is on the top floor. We will make it as simple as possible. The elevators will visit every floor, going up to the top floor, then going back down to the bottom floor, continuing as long as the simulation continues. That is, it does not respond to button presses calling an elevator or button presses by passengers requesting floors (but see extra credit!).

At each floor, the elevator stops to unload and pick up passengers. After `serviceTimeSecs` seconds, the elevator leaves for the next floor. It takes `travelTimeSecs` seconds to get there. Elevators are involved in both elevator and arrival and departure events. An arrival event should be generated when it departs, while a departure event should be generated when it arrives.

```
public class Elevator implements EventSubject {

    // Create elevator with given name and startingFloor in office.  Capacity
    // is number of passengers it can hold at a time.  Events is event queue
    // used to post and retrieve events.  ServiceTimeSecs is number of seconds
    // should stay at each floor, while travelTimeSecs is time between floors.
    public Elevator(String name, int startingFloor,
                    int capacity, Building office, EventHandler events,
                    int serviceTimeSecs, int travelTimeSecs)

    // returns name of elevator
    public String toString()

    // Do whatever is necessary to handle this event in simulation
    // Event should happen at timeNow on floor, and type of event is eventType
    // In this case, event is either arrival of elevator at new floor - so
    // passengers must be let off or put on (until elevator at capacity)
    // Alternatively is elevator departure event, so elevator moves to next floor.
    public void handleEvent(int floor, int timeNow, int eventType)
}
```

3 Hints for writing the program

First keep in mind that there is no concurrency in this program. No classes will extend `Thread`. The event queue keeps track of what is supposed to happen when. The `startSimulation` method (or a private method that it calls) in the `Building` class will successively pull events off of the queue and dispatch them to the subjects that are part of the events.

I would suggest that you begin by just getting the elevators moving and ignoring the people. (After all you want to make sure the elevators are well tested before you let in customers.) You will need to print out very complete information that tells you what is happening when.

Only when that is running perfectly should you start adding people to the simulation. The key to getting this to run properly is using the right data structures. See the hints in the description of the `Elevator` class. Notice that there is a two dimensional array of `Queues` in the `Building` class. The first subscript is the floor, while the second indicates whether the queue is for people waiting to go up or down.

4 Output

Your program should print out everything that happens in the simulation. I want to see output when a customer goes shopping, when they finish and get in line, when they get on the elevator, and when they get off the elevator. I want to see output that tells when an elevator arrives at a floor, what people get off, what people get on, and when it leaves a floor (and where it is going next).

The following is some rather crude output that should give you an idea what I have in mind:

```
lift 0 arrived at floor 0 at time 0.
Passengers getting off lift 0 at floor 0 at time 0
Passengers loading on lift 0 on floor 0 at time 0
   bill going to floor 1
   bob going to floor 2
lift 1 arrived at floor 1 at time 0.
Passengers getting off lift 1 at floor 1 at time 0
Passengers loading on lift 1 on floor 1 at time 0
lift 0 departing floor 0 at time 30, going to floor 1
lift 1 departing floor 1 at time 30, going to floor 2
lift 0 arrived at floor 1 at time 90.
Passengers getting off lift 0 at floor 1 at time 90
   bill
Passengers loading on lift 0 on floor 1 at time 90
lift 1 arrived at floor 2 at time 90.
Passengers getting off lift 1 at floor 2 at time 90
Passengers loading on lift 1 on floor 2 at time 90
lift 0 departing floor 1 at time 120, going to floor 2
lift 1 departing floor 2 at time 120, going to floor 1
lift 0 arrived at floor 2 at time 180.
Passengers getting off lift 0 at floor 2 at time 180
   bob
Passengers loading on lift 0 on floor 2 at time 180
lift 1 arrived at floor 1 at time 180.
Passengers getting off lift 1 at floor 1 at time 180
Passengers loading on lift 1 on floor 1 at time 180
lift 0 departing floor 2 at time 210, going to floor 1
lift 1 departing floor 1 at time 210, going to floor 0
lift 0 arrived at floor 1 at time 270.
Passengers getting off lift 0 at floor 1 at time 270
Passengers loading on lift 0 on floor 1 at time 270
lift 1 arrived at floor 0 at time 270.
Passengers getting off lift 1 at floor 0 at time 270
Passengers loading on lift 1 on floor 0 at time 270
   anne going to floor 1
   debbie going to floor 2
lift 0 departing floor 1 at time 300, going to floor 0
lift 1 departing floor 0 at time 300, going to floor 1
```

```
bill is done shopping on floor 1 and is in line for the elevator at time 330
lift 1 arrived at floor 1 at time 360.
Passengers getting off lift 1 at floor 1 at time 360
   anne
```

I'm sure that you can figure out a way to make it even more readable. You can either print this out with `System.out.println` or put it in a window with a scrollable `JTextArea`.

5 Extra Credit:

A wide variety of extra credit is possible. Here are some options:

1. Make the elevator scheduler more interesting. The elevator should only move if there is a reason to. Otherwise it should wait at a fixed floor for riders. When a button is pushed by a waiting shopper, one of the elevators should pick up the customer.

   Deciding on a reasonable logic for this is harder than you might think. I've heard of reports of people riding elevators all day to try to figure out the algorithm that is actually used. Of course this is only an issue when there are multiple elevators.

   Of course, elevators should always place the highest priority on requests by riders, but should always stop if they are going past a floor that has people requesting an elevator going that direction.

2. Print out the time in a more reasonable way. I.e., rather than writing 270 seconds, write 4:30.

3. Gather statistics on the average wait time for shoppers who are waiting for an elevator. (Of course this is much more interesting when you also have a smart way of scheduling elevators.)

4. Let shoppers shop for a random amount of time. Have the length of time that an elevator is on a floor be dependent on the number of people getting on and off the elevator on that floor.

If you do decide to add extra credit features that would change the results of the simulation (e.g., add features 1 or 4), please turn in two versions of your program. One that runs the simulations as in the original assignments, and then a second that adds the extra features. This way we can more easily see that your program generates correct output. (I.e., be nice to Ashok!)

6 Turning it in

Please turn in your program on Cortland as usual. It is due Friday at 4 p.m., but you can have a no penalty extension to Sunday if you need it (though I won't be around TCL over the weekend to help and no TA's will be on duty).