COMP 360, Spring 2013, Assignment 6

In this assignment, you will add multi-threading capabilities to the Ticketmaster project. *You may work alone or with one partner for this project.*

Read this entire document before you start working so you can get a feel for how the classes fit together.

Getting Started

Make a new NetBeans project. Go to my public folder and copy the Java files from the "cs360hw6" directory into your project's "src" directory. The files should then appear in NetBeans.

The Main Idea

I've given you the solution to the project 5 (Ticketmaster). If you want to, you may use your own solution, but at your own risk.

For this project, you will change the seating reservation idea so that every time someone wants to reserve or cancel a seat, a new thread is generated that will run the reservation or cancellation code. The reason for this is so that when someone wants to reserve a block of seats that can't be accommodated, that thread will automatically block (pause) until enough seats become available.

You should implement this functionality with synchronized blocks and/or methods and by using wait() and notifyAll() for threads signaling to each other.

Furthermore, since you will have multiple reservation/cancellation threads all running simultaneously, I have constructed a class called TheaterDisplay that shows a live seating chart of the theater. This class, however, does not automatically monitor the theater, so whenever the theater adds or loses a reservation, you should call refresh() on the TheaterDisplay object.

Classes to Edit

- The `ReservationThread` class handles the task of making a new reservation. It accepts all the information necessary to make a reservation, and when start()'ed, should keep trying to make the reservation until it succeeds. Hint: use wait() and have another thread send the notifyAll() signal when the reservation should try again.

  You will need to write the constructor and the start() method.

- The `CancellationThread` class handles the task of canceling an existing reservation. (There's no particular reason for this to be a thread, since canceling happens immediately, but it's more fun that way.) This class accepts all the information necessary to cancel a reservation, and when start()'ed, should cancel the reservation.

  You will need to write the constructor and the start() method.

- The remaining classes don't need any modification.

  In particular, you do not need to make any changes to the `Ticketmaster` class, but you should look at it so you understand how it works. It's not that much different than project 5,

except it starts threads for reserve and cancel instead of how project 5 did it, and the lookup command is commented out (though you can put it back if you really want).

The code should compile and run "out of the box," even without changes on your part (though it won't make or cancel any reservations).

Sample Run

- (Theater size = 5)

```
reserve kirlin 3
reserve sanders 4
cancel kirlin
```

The result should be kirlin gets seats 0-2, then after kirlin is canceled, sanders automatically steps in and grabs seats 0-3.

- (Theater size = 10)

```
reserve sheard 9
reserve gottlieb 5
reserve seaton 4
reserve mouron 3
reserve dunwell 2
reserve bodine 1
```

Here, the result is harder to predict. After all these command have been entered, sheard and bodine should have all ten seats (sheard 9 and bodine 1). If you then cancel sheard, a bunch of other reservations should take the empty seats automatically, but someone will still be left out. Start canceling people and make sure the last waiting reservation fills the gap as soon as the gap is big enough.

Assessment

Solutions should be:

- Correct
- In good style, including indentation and line breaks
- Written using features discussed in class.

Turn-in Instructions

- Upload all the Java files to Moodle before the project deadline.