Assignment 6
Show Tickets Problem
EE322C—The University of Texas at Austin—Spring 2016

Assigned:
Due: **Program:** Monday, Apr. 18th by 11:59pm
Points: 20 points. See the Grading Details section below.

You will be working in teams of 2 for this assignment. Each team member will get the same initial score, which will then be adjusted by individual participation level as needed. You get to choose your own team-mate – but be sure to let you respective Lab TAs to know who you team with and in which Lab you team-mate is in.

Purpose – to learn how to do multithreaded and network programming in Java

Problem - A ticket office sells reserved seat tickets for a certain show. The next client in line always gets the best-available seat. Your program will simulate the processing of a waiting line full of clients, and for each of them decide which is the next-best-available seat, acquire that seat and print the ticket.

Code Skeleton - Use both multithreaded and network programming. Finish implementing TicketServer.java and TicketClient.java. Edit and create more test cases in TestTickcetOffice.java. TA will run your TestTickcetOffice.java to test your program, your test cases should cover as many thread schedules as possible.

Processing Requirements

Allowed assumptions:

- the show is performed only once;
- only the ticket offices you create are selling tickets for the show;
- each client can buy only one ticket;
- many people are in line waiting to buy tickets – you may assume that there are a random # of people in line (between 100 and 1000 people initially), and that the line is never empty;
- clients in a waiting line are numbered sequentially starting at 1;
- when the show sells out you can stop processing clients;
- you may ignore money issues

The queue of clients are handled through the following process:

```
Repeat the following for each client in line until show is sold out
    Seat <-  find the bestAvailableSeat()
    If there is an available seat then
        markAvailableSeatTaken(Seat)
        printTicketSeat(Seat)
    Else
        Output to the screen “sorry, we are sold out”
End repeat
```

The program will need to contain several modules:

Procedure - bestAvailableSeat( )

Inputs: theater configuration, next client in line
Return value: best-available seat in the theater; or
–1 if no seat is available.

Procedure - markAvailableSeatTaken(seat)

Input: seat is the place of an available seat in the theater.
output: The place of the seat is marked as taken.

Procedure - printTicketSeat(seat)

Input: seat is the location of an available seat in the theater.
output: A ticket for that seat is printed to the screen – leave it on the screen long enough to be read easily by the client. The output format is up to you, but should contain the essential information found on a theater ticket.

Theater configuration

Theater seats are designated as shown in the Bates Recital Hall configuration at:
http://www.texasperformingarts.org/visit/seating_maps#bates

The most desirable seats are toward the front and in the middle. The next most desirable seats are toward the front but on the sides. The next most desirable seats are toward the back but in the middle. These are followed by the rest of the seats toward the back and on the sides. You should design your seat selection scheme with that in mind. Seat locations are designated by section [house right, house left, middle], row letter(s) [A-AA], and seat # [101-128].

Concurrency Requirements

Since waiting in line takes too long, the theater manager wants to add another ticket office. Both offices will be open at the same time and sell tickets for the same show. Each ticket office will have its own copy of your program and a computer with a printer for printing the tickets it sells. Each ticket office will be represented in your program by a separate *thread* of execution with a separate queue of clients. But there is only 1 theatre and only 1 show that is being booked.

Additional concurrency constraint:

- The two offices may not both print a ticket for the same best available seat for 2 clients being processed simultaneously. You must create a design that works to enforce this constraint.

Additional output requirement:

- print which thread the reservation request came from and its disposition. For example, say Box Office A: Reserved HR, 109A. Or "Box Office B: Failed to reserve HR,109A. Already allocated. Generate your random traffic with no delays between requests.

Grading details:
Your assignment will be graded first by compiling and testing it for correctness. After that, we will read your code to determine whether all requirements were satisfied, as well as judge the overall style

of your code. Points will be deducted for poor design decisions and un-commented, or unreadable code as determined by the TA. Here is the point breakdown:

Overall Correctness of the program - 10 points
Reasonable output format – 3 points
Correct usage of concurrency constructs - 5 points
Proper use of random # constructs – 1 point
Coding style - 1 point