Assignment 6
Multithreaded Programming
EE422C—The University of Texas at Austin—Fall 2016

Problem – A certain theater plays one show each night. The box office sells tickets to clients on a first come, best seat basis. That is, the next client in line always gets the best available seat. Your program will simulate the box office. You will process a line of clients, determine the next-best available seat for each client, acquire that particular seat and finally print the ticket.

Processing Requirements

Allowed assumptions:

- the line is never empty
- each client can buy only one ticket
- clients waiting in line are numbered sequentially starting at 1
- client numbers are unique even across multiple lines
- when the show sells out you can stop processing clients
- when there are no more clients you can stop selling tickets

Clients are processed according to the following pseudocode:

```
Repeat for each client until show is sold out or no more clients
     Seat <- find the best available seat
     If there is an available seat, then
          mark the seat as taken
          print the ticket
End repeat
Output to the screen only once "Sorry, we are sold out."
```

Theater configuration

The theater can have *N* rows and *M* seats for each row. Rows are ordered sequentially in base 26 using letters (A, B, C, …, Z, AA, AB, AC and so on for a total of *N* number of rows). Seats are numbered from 1 to *M*. The best seat is the least combination of seat row and seat number.

Best seat order example for N = 2 and M = 2: [A1 > A2 > B1 > B2]

Concurrency Requirements

Since waiting in a single line is inefficient, the theater manager has multiple box offices working to sell tickets simultaneously. Each box office is represented as a separate *thread* of execution and has its own queue of clients. Remember that there is still only 1 theater and 1 show that is being booked.

Concurrency constraints

- Only one box office may print out a ticket for any single seat even if multiple clients are processed simultaneously. You must create a design that works to enforce this constraint.

Ticket class (can be an inner class)

The `toString()` method should return the string that prints to console like the one below.

```
-------------------------------
| Show: Ouija                 |
| Box Office ID: BX1          |
| Seat: D104                  |
| Client: 4                   |
-------------------------------
```

Seat class (can be an inner class)

The `toString()` method should return a string with the row letter appended with the seat number like in the ticket example above – `D104`.

Theater class

`Seat bestAvailableSeat()`

- This method calculates the next best available seat according to the theater configuration. For example, consider a theater with 2 rows, 2 seats per row and 2 tickets have been sold. This method would return B1 since A1 and A2 have been reserved.

`Ticket printTicket(String boxOfficeId, Seat seat, int client)`

- This method prints the ticket as if the box office physically processed and printed a ticket for a particular seat. Also print each ticket sold to the console with a reasonable delay for human readability (no more than 1 second). Return `null` if a box office failed to reserve the seat.

`List<Ticket> getTransactionLog()`

- This method returns a list of all tickets sold for the current show in the order in which they were purchased.

BookingClient

`List<Thread> simulate()`

- This method starts the multithreaded simulation. Create your threads in this method and also start them. Return a list of all the threads you create. We will test your program by instantiating BookingClient with different parameters then invoking simulate.

Main method

Your BookingClient class file must have a main method. This main method must initialize the offices and theater with the same data as the example output shown below, and must call simulate().

Example Console Output

Newlines may be slightly different from the data below, although you should try to reproduce the output as shown. You may assume that show names are short.

Simulation Parameters:

- office: {BX1=3, BX3=3, BX2=4, BX5=3, BX4=3}
- theater: {3 rows, 5 seats per row, show: “Ouija"}

```
------------------------------
| Show: Ouija                |
| Box Office ID: BX1         |
| Seat: A1                   |
| Client: 1                  |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX4         |
| Seat: A2                   |
| Client: 5                  |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX2         |
| Seat: A3                   |
| Client: 3                  |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX5         |
| Seat: A4                   |
| Client: 4                  |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX4         |
| Seat: A5                   |
```

```
| Client: 7                    |
--------------------------------

--------------------------------
| Show: Ouija                  |
| Box Office ID: BX1           |
| Seat: B1                     |
| Client: 6                    |
--------------------------------

--------------------------------
| Show: Ouija                  |
| Box Office ID: BX2           |
| Seat: B2                     |
| Client: 8                    |
--------------------------------

--------------------------------
| Show: Ouija                  |
| Box Office ID: BX2           |
| Seat: B3                     |
| Client: 12                   |
--------------------------------

--------------------------------
| Show: Ouija                  |
| Box Office ID: BX5           |
| Seat: B4                     |
| Client: 9                    |
--------------------------------

--------------------------------
| Show: Ouija                  |
| Box Office ID: BX4           |
| Seat: B5                     |
| Client: 10                   |
--------------------------------

--------------------------------
| Show: Ouija                  |
| Box Office ID: BX5           |
| Seat: C1                     |
| Client: 14                   |
--------------------------------
```

```
------------------------------
| Show: Ouija                |
| Box Office ID: BX2         |
| Seat: C2                   |
| Client: 13                 |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX1         |
| Seat: C3                   |
| Client: 11                 |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX3         |
| Seat: C4                   |
| Client: 2                  |
------------------------------

------------------------------
| Show: Ouija                |
| Box Office ID: BX3         |
| Seat: C5                   |
| Client: 15                 |
------------------------------

Sorry, we are sold out!

Process finished with exit code 0 (not required in your output)
```

What to submit:

1. The package directory 'assignment6' and all Java files it contains
2. A UML diagram showing your classes.

***** Zip these two item together and name the zip file 'project6_EID.zip' *****

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:

Correctness of the program - 12 points
Correct usage of concurrency constructs - 7 points
Console output – 3 points
Coding style - 3 point
UML diagram
Code structure

CHECKLIST – Did you remember to:

- ☐ Work on the assignment on your own?
- ☐ Re-read the requirements after you finished your program to ensure that you meet all of them?
- ☐ Make sure that your program passes all your test cases?
- ☐ Ensure that all your .java files have the proper header?
- ☐ Upload your solution to Canvas in a zip file with the proper filename?
- ☐ Include the UML diagram in a PDF file, with your Name and EID on it?
- ☐ Download your uploaded solution into a fresh directory and re-run all test cases?