

EE422C - Assignment 6

Multithreaded Programming – Individual assignment

Please find the **starter code** and **grading script** on Canvas.

EE422C - Assignment 6

- 1 Problem
- 2 Requirements
 - Allowed assumptions
 - Pseudocode
- 3 Theater Configuration
- 4 Concurrency
- 5 Class Details
 - Ticket Class (Can be an inner class)
 - Seat Class (Can be an inner class)
 - Theater Class
 - BookingClient
- 6 Example Output
- 7 Submission
- 8 Grading details
- 9 FAQ
- 10 Checklist

1 Problem

A certain theater plays one show each night. The theater has multiple box office outlets to sell tickets, and the box offices sell 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 offices and the theater. 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.

2 Requirements

Allowed assumptions

- the lines are never empty at the start

- 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

Pseudocode

Clients are processed according to the following **pseudocode**:

```
Repeat for each client until show is sold out or there are 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
If (sold out)
  Output to the screen only once "Sorry, we are sold out!", without the quotes.
```

3 Theater Configuration

The theater can have N rows and M seats for each row. Rows are ordered sequentially using letters ($A, B, C, \dots, Z, AA, AB, AC, \dots, AAA$) and so on for a total of N number of rows). Row A is the first row, numbered 1. 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 $M = 3$: $[A1 > A2 > A3 > B1 > B2 > B3]$

4 Concurrency

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.

- 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.

5 Class Details

Ticket Class (Can be an inner class)

The `toString()` method should return the **string** that prints to console like the one below. We have provided you with the method, please double check to use it correctly.

You **need not** worry about overly long show names, ID's, seat numbers, etc. that won't fit into the bounding box as shown 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. We have provided you with the method; double-check to use it correctly.

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 small **delay** for human readability (use the `printDelay` variable inside the Theater class to control the delay time i.e. the time your thread needs to wait for another ticket buying transaction). 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, and then invoking simulate.

```
main()
```

- 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()`.

6 Example 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"}

There are 15 clients. Note that (i) the Seats are in order (they don't have to be printed in order of bestness, but they should be in correct order in the output of `getTransactionLog()`) (ii) the Box Office order is mixed up (as it should be) and (iii) the Client IDs are not required to be in order (they could be in order or not), but are unique.

```
| 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! (You may print this line just once overall. Hint: You could use synchronization to ensure that the line is printed only once.).

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

7 Submission

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

Zip the code together and name the zip file **Project6_EID.zip**

When unzipped, the folder structure should be:

```
project6_EID(folder)
  assignment6(folder)
    Theater.java
    BookingClient.java
    other.java
```

8 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:

1. Correctness of the program - 12 points
2. Correct usage of concurrency constructs - 7 points
3. Console output – 3 points
4. Coding style - 3 point
5. UML diagram
6. Code structure

9 FAQ

Q1: Should the client ID's match the ones in the sample output?

A1: No. The client ID's should be unique, that's all. You should use some sort of reasonable ordering scheme, so that in most cases, no numbers in a sequence will be altogether skipped, although they might be printed out of order. The seat ID's should be in order of sale.

Q2: Where do I implement the pseudo-code listed on page 1?

A2: Wherever it is suitable.

Q3: What should we print out if the number of clients run out before the show is sold out?

A3: Print out no final message. Just exit the program.

Q4: How should we input the initial parameters for the booking offices and the theater?

A4: You may hard-code these values in your code when submitting.

Q5: Are we allowed to add fields to the classes? Any restrictions (like static, private, etc.)?

A5: Don't use static variables in BookingClient or Theater. You may use non-static variables in Theater, for example to hold client ID. Don't change the number or type of arguments passed into the constructor. Don't change method declaration (except as described further down in this document). You are allowed to add helper methods (but make them private), except for getters and setters.

Q6: May we use System.exit()?

A6: No.

Q7: Is it ok to add a variable inside the static Seat class? Is it ok to add an extra line to the constructor of the static Seat class?

A7: You may add variables to the class and lines to the constructor, just not change the number or type of arguments passed into the constructor.

Q8: If there are N seats, should we print "Sold Out" as soon as N tickets are sold?

A8: Yes, you should, but only after the last ticket has been printed. Don't wait for client N+1 to be processed.

Q9: Are we allowed to change the method header to add 'synchronized'?

A9: Yes.

Q10: Is a class diagram sufficient for the UML diagram requirement?

A10: Yes.

Q11: How do I debug multiple threads using Eclipse?

A11: Search the web. For example

<http://stackoverflow.com/questions/5375538/debugging-multiple-threads-in-eclipse>

Q12: In my program, each box office gets processed in turn before other unprocessed ones. Is this OK?

A12: No, this is not, as you are not treating them all fairly.

Q13: No src folder this time under assignment6?

A13: No.

Q14: Do we get a grading script and test files?

A14: Yes, you can find the grading script and sample test on Canvas.

Q15: Are we allowed to make our classes implement interfaces?

A15: Yes.

Q17: Are we allowed to add other inner classes?

A17: Yes.

Q18: (4) What is the purpose of the the BookingClient Constructor?

A18: `List<Thread> simulate()`

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 and then invoking simulate.

Q19: In the lab descriptor, it says the getTransactionLog method "returns a list of all tickets sold for the current show in the order in which they were purchased." Does this mean the most recent purchase is the first in the list, or would it be the last?

A19: Use list/queue, the first one sold is the first one in the list.

Q20: What would be a good way to compare Seats?

A20: You could make a Comparator class, and then pass it to some sort function.

10 Checklist

- ☐ 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?