Middle East Technical University   Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming and Systems

Fall 2019-2020

Homework 2 - Seat Reservation System

Due date: December 1st, 2019, Sunday, 23:55

1 Introduction

Purpose of this assigmnet is to familiarize you with multi-threaded application development and synchronization using Java. In this assignment you will implement a seat reservation system where multiple users can reserve seats for an event concurrently. You will be using multi-threading and concurrency classes that are present in Java.

2 Problem Definition

Application we will create will be used to reserve seats for an event (concert, movie etc.). Seats will be arranged in a NxM 2D grid, where N is the number of rows and M is the number of columns in the grid. Users will try to reserve seats from this grid concurrently.

When a user reserves a seat, the seat will be marked as taken and the name of the user who reserved the seat will also be stored. This reservation process lasts 50ms due to some database transactions and because of some lazy developers we currently don't have the code for the database part. So we will simulate this delay using `Thread.sleep()` method. Additionally, because of the same lazy database developers, a user's transaction may fail with a 10% chance. In this case, user will try again later (until he reserves the seats or he sees that he can't reserve the seats) after 100ms delay.

A user may want to reserve a single seat or multiple seats. As you may have guessed, a seat cannot be reserved by multiple users. In case where two users want to reserve the same seat, the conflict will be resolved in a first come first served basis. If a user cannot reserve *all* the seats he wants, he will just give up and won't attend the event.

You need to be careful with multiple users accesing the same seat at the same time. It may cause users to reserve the same seat or it may cause both users to not reserve any seats when one of them could have reserved it. Also be careful about deadlocks in this stage.

Psuedo code of a user's routine is as follows:

Algorithm 1: User thread routine

```
while shouldRetry do
    Check whether the seats are taken or not
    if All wanted seats are empty then
        if Database didn't fail then
            Update seat information for all seats (mark them as taken)
            Sleep 50ms
            Logger.LogSuccessfulReservation
        else
            Logger.LogDatabaseFailiure
            Sleep 100ms
    else
        Logger.LogFailedReservation
```

2.1 Implementation Specifications

- Seats will be represented by a `Seat` class. Each `Seat` object will have its own synchronization mechanism.
- `User` class will implement the `Runnable` interface. After reading the input you will start each user as a seperate thread (preferebly using an executor).
- `User`s will have a name and a list of seats that user wants. Both of these values won't change once set.
- Users(Threads) will use the provided `Logger` class to log any successful or unsuccessful reservations.
- `InitLogger` method of the `Logger` class should be called before the threads are created. Note that you won't need an instance of `Logger`, because all methods in it are static.
- When a database failiure occurs for user A, other users should be able to access (and possibly reserve) user A's wanted seats while user A sleeps for 100ms.
- Seat grid should be represented by an `ArrayList<ArrayList<Seat>>`
- Main thread should wait for all threads to finish, before printing the last state of the grid and exiting.

2.2 Using the Logger Class

`Logger` class has 4 methods inside, one for initialization and other three for logging. While using `Logger` methods make sure you call `Logger.InitLogger` *once* before starting the user threads. Also make sure that you are using the correct log method (successful reservation/database fail/failed reservation) so that your outputs are correct. You don't need to consider synchronization of `stdout` while using the `Logger`, it handles it internally.

Log methods can take 5 paremeters. 4 of these parameters are required and 1 is optional. Parameters are explained on `LogSuccessfulReservation` method below since all log methods require exactly the same parameters.

```
LogSuccessfulReservation(
    String name,        // Name of the User thread calling this method.
    String seats,       // List of seats wanted by the calling user
    long time,          // Time when the function is called, use System.nanoTime()
    [String comment]    // Comment string, optional parameter
)

Example call using the above method from a user thread:

Logger.LogSuccessfulReservation(this.name, wantedSeats, System.nanoTime(), "20 min adventure Morty!")
```

3 Input & Output

3.1 Input

Your program should read the data from `stdin` and output any output to `stdout`.

First line of the input will be two integers denoting the grid size (N and M). Next line will be a single integer K, denoting the number of users. Next K lines will consist of a string S for username followed by L number of seats ($L <= N * M$), denoting the seats that user will reserve. Seats in the input will be denoted by the concatanation of a single capital letter (A-Z) and a single digit(0-9) (e.g. 'A2' or 'F8') Where letter identifies the row and the digit identifies the column. Letter 'A' will correspond to the 1st row and Z will correspond to 26th row. A sample input and grid layout with seat names are given below. Note that seats are not ordered in any way in the input.

```
5 3
5
Kadir A2 A0 A1
Bilal C2 B0 B1
Ulfet C2 C1 C0
Akcay C1 C0 B2
Selim A1 A0 A2
```

A0	A1	A2
B0	B1	B2
C0	C1	C2
D0	D1	D2
E0	E1	E2

3.2 Output

Output of your program will consist of the output of the Logger's output and the state of the grid in the end. Logger output consists of five columns. First column of a log line is the time in milliseconds. Second column is the thread id of the thread printing the log. Third column is the user's name. Fourth column is the set of wanted seats by that user (order is not important). Last column is the comment column. Comment column is not important for the grading, but it might come in handy while debugging.

After the log output is the state of the grid at the end. Each seat is printed as `<seat status>:<user's name>`. Where seat status is a single letter: 'E' for empty, 'T' for taken. User name is self explanatory. Seats on a line are seperated by a single space character (' '). Top left corner of the grid is the seat A0 and the bottom right is the seat with highest row and column number (E2 in the example below). A sample output is given below.

```
[-] 0008.39760 | 13 | Kadir | [A0, A1, A2] | Comment: Failed, trying again.
[*] 0062.56434 | 14 | Bilal | [B0, B1, C2] | Comment: Retry No: 1
[X] 0069.00452 | 15 | Ulfet | [C0, C1, C2] | Comment: Seats are not available
[*] 0108.55673 | 13 | Kadir | [A0, A1, A2] | Comment: Retry No: 2
[X] 0109.26386 | 17 | Selim | [A0, A1, A2] | Comment: Seats are not available
[*] 0119.70952 | 16 | Akcay | [B2, C0, C1] | Comment: Retry No: 1
T:Kadir T:Kadir T:Kadir
T:Bilal T:Bilal T:Akcay
T:Akcay T:Akcay T:Bilal
E:      E:      E:
E:      E:      E:
```

4 Specifications

- Your code must be written in Java
- Submissions will be evaluated with both black box and white box techniques. Correctness of your outputs is important. Make sure that you use the `Logger` correctly. Also don't modify `Logger` class because it will be overwritten during evaluation. White box evaluation will be employed to make sure that your code adheres to OOP principles.
- Non-terminating submissions will suffer a penalty, as well as submission that don't adhere to OOP principles.
- Everything you submit should be your own work. Usage of binary source files and codes found on internet is strictly forbidden.
- Please follow the course page on ODTUClass for updates and clarifications.
- Please ask your questions related to homework through ODTUClass instead of emailing directly to teaching assistant.

5 Submission

Submission will be done via ODTUClass. Create a zip file named `hw2.zip` that contains all your java source code files without any directory. All your code should be under defaultpackage. Your code should be able to compile and run using this command sequence.

```
> javac *.java
> java Main
```