Laboratory Assignment 3
Using `synchronized`, `wait` and `notifyAll`

Module: Concurrency (CS-210)
Academic year: 2020-21

Allocated marks: This assignment accounts for 2% of the total module marks.

Objectives

The learning objectives of this assignment are as follows.

- To apply Java language features to guard against interference in a concurrent program.

Resources:

- For this session, you may find the *library* code on the github repository useful. The repository for Java code is available at: `https://github.com/AlmaRahat/CS-210-Concurrency/tree/main/java-code`

Tasks

Consider the following scenario.

> A central computer connected to remote **terminals** via communication links is used to automate **seat** reservations for a **concert hall**. A booking clerk can display the current state of reservations on the terminal screen. To book a seat, a client chooses a free seat and the clerk enters the number of the chosen seat at the terminal and issues a ticket. A system is required which avoids double-booking of the same seat while allowing clients free choice of the available seats.

In the previous assignment, you modelled this scenario. If everything worked, you would have seen the following output from performing *Check* → *Supertrace* using the LTSA tool.

```
Analysing using Supertrace (Depth bound 100000 Hashtable size 8000K )...
-- Depth: 29 States: 42 Transitions: 103 Memory used: 31719K
Trace to property violation in {a,b}::Seats.seat.0:Seat:
        b.choose.1
        b.seat.1.query.0
        b.seat.1.reserve
        b.choose.1
        b.seat.1.query.1
        a.choose.1
        a.seat.1.query.1
        b.return
        b.choose.0
        b.seat.0.query.0
        a.return
        a.choose.0
        a.seat.0.query.0
        b.seat.0.reserve
        a.seat.0.reserve
Analysed using Supertrace in: 1ms
```

This indicates that we ought to protect both the *query* and *reserve* methods. Make sure that you can explain to yourself why that is the case. If you cannot justify it, please seek help from the demonstrators.

Task 1: Identifying characteristics of processes.

We can implement this scenario in Java. Analysing the scenario, we found three classes: *Seats*, *Terminal* and *Application*. **Draw arrows** from the left to the right to connect the classes to relevant concepts below.

Classes	Concepts
Seats	Controller: initiates various objects and controls the overall program flow.
Terminal	Passive process or monitor class: hold the shared resources and protects them against concurrent issues.
ConcertHall	Active process: a class that extends Thread or implements Runnable so that many instances of it can be run in parallel.

Task 2: Using `synchronized`.

Implement the classes above in Java that meets the following specifications:

- Use `synchronized` keyword to protect against unwanted concurrent issues.
- Demonstrate correct operation with *three* instances of terminals trying to book *two* seats concurrently. In this case, make the program such that a terminal queries a random seat at different random times, and reserves if available, otherwise returns to main program.

Here is an example output; of course it may show slightly different behaviour based on your implementation.

```
This is thread: t2
This is thread: t1
This is thread: t0
I will sleep for: 354
I will sleep for: 161
I will sleep for: 607
t1 is trying to book: 0
ID: 0 has been reserved.Thread-1
t2 is trying to book: 1
ID: 1 has been reserved.Thread-2
t0 is trying to book: 1
ID: 1 is already booked.
```

Task 3: Extending the scenario, and using `wait` and `notify`.

In this task, you will add a new method *release*. This will allow a seat to be released from booked state. This means that a released seat may be captured by a waiting terminal. This scenario is perfect for using `wait` and `notify` methods in Java.

Demonstrate the correctness of your program like before: three terminals trying to book two seats. In this case, the successful terminals that can book a seat sleep for a random amount of time, and then release the respective seat. If a terminal is unsuccessful in booking a seat then it should simply wait for a notification for a release to proceed with

Here is an example output.

```
This is thread: t2
This is thread: t1
This is thread: t0
t2 will sleep for: 164
t0 will sleep for: 251
t1 will sleep for: 362
t2 is trying to book: 0
Thread-2 has reserved seat 0
t2 will sleep for: 914
t0 is trying to book: 1
Thread-0 has reserved seat 1
t0 will sleep for: 178
t1 is trying to book: 0
t0 is trying to release: 1
Thread-0 has released seat 1
Thread-1is waiting to reserve seat 0
t2 is trying to release: 0
Thread-2 has released seat 0
Thread-1is waiting to reserve seat 0
Thread-1 has reserved seat 0
t1 will sleep for: 962
t1 is trying to release: 0
Thread-1 has released seat 0
```

Once you have completed all the tasks, please make sure that you have been signed off.