The University of Melbourne
School of Computing and Information Systems
SWEN90004 Modelling Complex Software Systems

Assignment 1a, 2022

Released: Friday 4 March, 2022. *Deadline:* 23:59, Thursday 24 March, 2022

Objectives

The objectives of this assignment are: to convert a description of a system into a simulation model of that system; to implement that simulation in a shared memory concurrent programming language; to use the implemented simulation to explore the behaviour of the system; to gain a better understanding of safety and liveness issues in concurrent systems.

Background and context

There are two parts to Assignment 1 (a and b), each worth 10% of your final mark. This first part of the assignment deals with programming threads in Java. Your task is to implement a concurrent simulation of tour groups visiting the Museum of Parallel Art (MoPA).

The system to simulate

A highlight of any tourist's stay in the city of Konkurrence is a visit to the Museum of Parallel Art (MoPA). Recently, due to COVID-19, MoPA has had to institute strict safety protocols to minimise contact between groups of visitors.

Visitor Groups arrive at MoPA at random intervals (we are not concerned with how they get there). Groups are *not* free to wander through the rooms of the museum. Rather, they must be escorted from room to room in a set order; that is, after arriving at the museum's Foyer, they proceed to Room 0[1], from Room 0 to Room 1, and so on, until they return to the Foyer and depart. To prevent overcrowding, only a single Group of visitors may be present in any location (i.e., the Foyer or one of the Rooms) at a time. To enforce this restriction, Groups are escorted from location to location by a MoPA Guide, who will only convey a Group from their current location to the next location *after* the previous Group has left that location. Having escorted a Group in this fashion, the Guide then returns to their original location to await the next Group. That is, each Guide moves backwards and forwards between a single pair of locations (e.g., the Foyer and Room 0, or Room 2 and Room 3, etc.)

Before entering the Rooms of the museum, Groups are required to pass through a security check in the Foyer where their bags are screened and stored safely until their departure. This procedure is managed by a single Security Guard, who meets arriving groups outside the Foyer and escorts them through the security check and into the Foyer, where they await the first Guide. Once their visit is complete, the same guard meets them once they leave the final Room, checks that they have not attempted to steal any parallel artworks, returns their bags to them, and escorts them back out of MoPA (at which point we are no longer concerned with them).

The full system is shown in Figure 1. At this point in time, Group 6 is currently in the Foyer, being screened for entry by the Security Guard (red `s`). Group 5 is just leaving Room 1 with a guide (blue `g`) and Group 4 is leaving Room 5, waiting for the Foyer to become free so they can retrieve their bags and depart. After group 6 leaves the Foyer (for Room 0), group 4 will be able to enter and be escorted out by the Security Guard. If instead, there was no Group 4 waiting to enter the Foyer, but more Groups were arriving at MoPA, it would be necessary for the Security Guard to go outside to escort them in. Thus, the Security Guard will occasionally go out of and into the Foyer (at random intervals).

[1] A computer scientist sitting on the MoPA Board of Management insisted on the Rooms being numbered from zero.

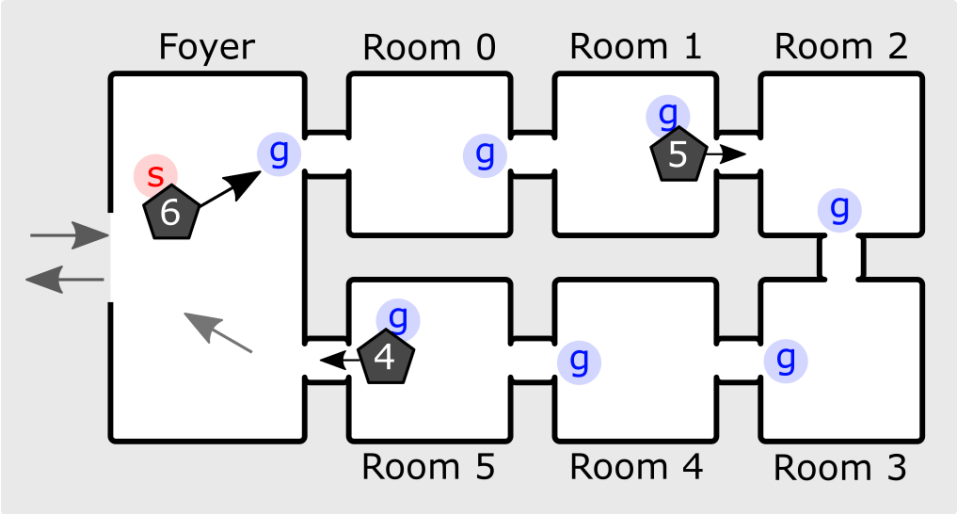

Figure 1: The Museum of Parallel Art (MoPA). Visitor groups are shown as numbered hexagons. The Security Guard, who escorts groups between the outside of the museum and the Foyer, is shown as a red circle labelled "s". The museum Guides, who escort the Groups from one location (Foyer or Room) to the next, are shown as blue circles labelled "g". See text for a description of the system behaviour at this point in time.

Your tasks

Your task is to implement a simulator for MoPA system described above. It should be suitably parameterised such that the timing assumptions can be varied, and the number of Rooms. You should use your simulator to explore the behaviour of the system to identify any potential problems with its operation. The simulator should produce a trace of events matching that below (here shown in two columns). Figure 1 corresponds to the state of the system immediately after "`group [6] arrives at the museum`", near the bottom of the right column. Note that the movement of the museum Guides is not shown in the output trace, but should be implemented.

```
       :
group [3] leaves room 3
security guard goes out
group [4] enters room 1
group [4] leaves room 1
security guard goes in
group [3] enters room 4
group [2] enters the foyer
group [3] leaves room 4
group [2] departs from the museum
security guard goes in
group [4] enters room 2
group [4] leaves room 2
group [3] enters room 5
group [3] leaves room 5
security guard goes out
group [5] arrives at the museum
group [5] leaves the foyer
```

```
group [4] enters room 3
group [4] leaves room 3
security guard goes out
group [5] enters room 0
group [5] leaves room 0
security guard goes in
group [3] enters the foyer
group [3] departs from the museum
group [4] enters room 4
group [4] leaves room 4
security guard goes in
group [5] enters room 1
group [5] leaves room 1
security guard goes out
group [4] enters room 5
group [4] leaves room 5
group [6] arrives at the museum
       :
```

A possible design and suggested components

In the context of Java, it makes sense to think of each location (Foyer and Rooms) as a **monitor**. A possible set of **active processes** would then be:

`Producer`: Generates new Groups arriving at MoPA, subject to the Foyer being unoccupied and the Security Guard being present to escort them in. The times between arrivals should vary.

`Consumer`: Removes Groups that have departed from MoPA, once they have been escorted out of the Foyer by the Security Guard. The times between departures should vary.

`Guide`: Picks up Groups from one location (either the Foyer or a Room) and delivers them to the next location (the next Room or the Foyer). To collect a Group from the Foyer, they must first have been escorted in by the Security Guard. To deliver a Group back to the Foyer, it must be unoccupied, and the Security Guard should be present to return their bags and escort them out.

`Security Guard`: If the Foyer is empty, periodically goes outside for an interval to check for arriving Groups. If there are no arriving groups, periodically goes inside for an interval to check for Groups ready to depart.

We have made some scaffold code available on LMS that follows this outline described above. The components we have provided are:

`Producer.java` **and** `Consumer.java`: as described above.

`Group.java`: Visitor Groups can be generated as instances of this class.

`Params.java`: A class which, for convenience, gathers together various system-wide parameters, including time intervals.

`Main.java`: The overall driver of the simulation. Note that this won't compile until you have defined some additional classes.

System parameters:

The class `Params.java` contains the system parameters, including the number of rooms and a number of timing parameters. Varying these parameters will give you different system behaviours. Once you have a working simulator, you should experiment with the parameter settings.

Procedure and assessment

- The project should be done by students **individually**.
- Late submissions will attract a penalty of 1 mark for every *calendar* day it is late. If you have a reason that you require an extension, email Nic *well before the due date* to discuss this.
- You should submit a single `zip` file via LMS. The `zip` file should include:
  1. A *single* directory containing all Java source files needed to create a file called `Main.class`, such that "`javac *.java`" will generate `Main.class`, and "`java Main`" will start the simulator.
  2. A plain text file `reflection.txt` should contain approximately 500 words that discusses the potential problems that can arise in this system, drawing on observations of your simulator behaviour. You could also use this text to evaluate the success or otherwise of your solution, identify critical design decisions or problems that arose, and summarise any other insights from experimenting with the simulator. Please ensure that this is a *plain text* file; ie, not a `doc`, `docx`, `rtf`, or other file type that requires specific software to read.

  All source files and your text file should contain, near the top of the file, your name and student number.

- We encourage the use of the LMS discussion board for discussions about the project. **However, all submitted work must be your own individual work.**
- This project counts for 10 of the 40 marks allocated to project work in this subject. Marks will be awarded according to the following guidelines:

Criterion	**Description**	**Marks**
Understanding	The submitted code is evidence of a deep understanding of concurrent programming concepts and principles.	3 marks
Correctness	The code runs and generates output that is consistent with the specification. **Note: if we cannot get your code to run, you will receive zero marks for this criterion.**	2 marks
Design	The code is well designed, potentially extensible, and shows understanding of concurrent programming concepts and principles.	2 marks
Structure & style	The code is well structured, readable, adheres to the code format rules (Appendix A), and in particular is well commented and explained.	2 marks
Reflection	The reflection document demonstrates engagement with the project.	1 marks
Total		10 marks

Nic Geard
3 March 2022

A Code format rules

Your implementation must adhere to the following simple code format rules:

- Every Java class must contain a comment indicating its purpose.
- Every method must contain a comment at the beginning explaining its behaviour. In particular, any assumptions should be clearly stated.
- Constants, class, and instance variables must be documented.
- Variable names must be meaningful.
- Significant blocks of code must be commented.

  However, not every statement in a program needs to be commented. Just as you can write too few comments, it is possible to write too many comments.
- Program blocks appearing in if-statements, while-statements, etc., must be indented consistently. They can be indented using tabs or spaces, and can be indented 2, 4, or 8 spaces, as long as it is done consistently.
- Each line must contain no more than 80 characters.