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

Assignment 1a, 2023

Released: Thursday 2 March, 2023. *Deadline:* 23:59, Thursday 23 March, 2023

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. This first part of the assignment (ie, this part) deals with programming threads in Java, and is worth 10% of your final mark. The second part of the assignment (to be released in a few weeks) deals with modelling in FSP, and is worth 15% of your final mark. Your task is to implement a concurrent simulation of Professor Z and their league of superheroes.

The system to simulate

In an alternate version of our Earth, a genetic mutation causes some people, known as Z-folk, to be born with superhuman abilities which they use to fight crime. These Superheroes are led by Professor Z, whose Mansion they regularly gather in discuss world affairs. Within the Mansion is a Secret Room, where Professor Z and the superheroes meet to allocate crime-fighting Missions.

Within the Mansion, Professor Z and the superheroes interact according to a set of strict rules. Superheroes enter the Mansion, discuss world affairs, and enter the Secret Room (see Figure 1 for a schematic of the system). While Professor Z is in the Mansion, no more superheroes may enter or leave the Mansion. Once all superheroes who are present in the Mansion have also entered the Secret Room, Professor Z starts a meeting in the Secret Room. During a meeting, superheroes report back on and release their completed Missions (if they had one) and acquire new Missions. Once a superhero has acquired a new Mission, they may leave the Secret Room, and spend more time discussing world affairs. Once all superheroes have left the Secret Room, Professor Z ends the meeting and, at some point after this, leaves the Mansion. The superheroes are then free to leave the Mansion and complete their Missions. After completing a Mission, a superhero once more returns to the Mansion, and the process begins again.

Your tasks

Your task is to implement a simulator for the system described above. It should be suitably parameterised such that the timing assumptions can be varied, and the number of superheroes can be varied. You should use your simulator to explore the behaviour of the system to identify any potential problems with its operation.

Note: you **should** *observe problems if you simulate the system as described—you are* **not** *required to fix them as part of this assignment! (though you are encouraged to think about how you might do so).*

You may assume that:

1. Superheroes may only enter the Secret Room after first entering the Mansion.

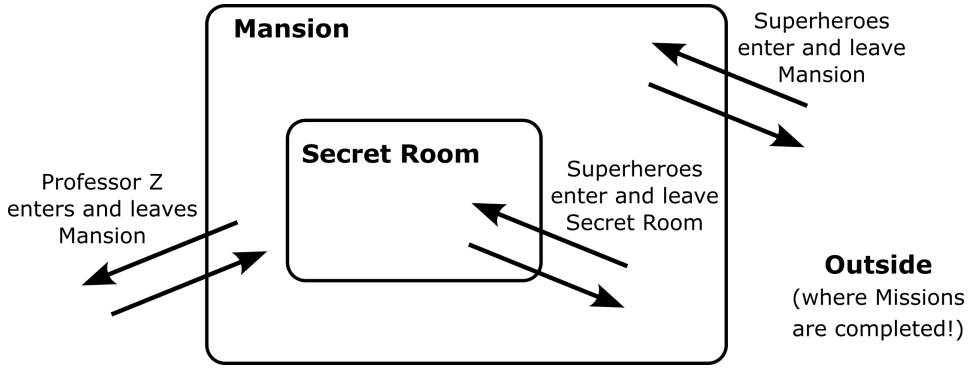

Figure 1: A schematic of the system, showing the relationship between the Mansion, where Superheroes gather to discuss world affairs, the Secret Room, where meetings are held to allow Superheroes to acquire and release Missions, and the outside world, where Missions are completed by Superheroes.

2. Superheroes may only leave the Mansion after first leaving the Secret Room.
3. Superheroes can only acquire or release Missions while they are in the Secret Room and a meeting is in progress.
4. A Superhero may only have one Mission (completed or uncompleted) at a time.
5. Superheros who are currently on a Mission will not return to the Mansion until it is complete.
6. Not all superheroes are required to be in the Mansion before a meeting can start.
7. A pair of Rosters, for new and completed Missions respectively, is used to handle Missions during a meeting.
8. Professor Z does not need to enter or leave the Secret Room; they only need to be in the Mansion in order to start a meeting.

The simulator should produce a trace of events matching that below (here shown in two columns). Before this section of the trace, Superhero 1 had acquired Mission 1 in a previous meeting. Not that, after Professor Z entered the Mansion, the meeting could not begin until Superhero 1 had entered the Secret Room. The meeting ended once there were no Superheroes left in the Secret Room, but none of the Superheroes could leave the Mansion until Professor Z had left.

```
       :
Superhero 2 enters Mansion.
Superhero 3 enters Mansion.
Superhero 2 enters the Secret Room.
Superhero 3 enters the Secret Room.
Superhero 1 exits from Mansion.
Superhero 1 sets of to complete Mission 1!
Superhero 1 completes Mission 1!
Superhero 1 enters Mansion.
Professor Z enters the Mansion.
Superhero 1 enters the Secret Room.
Meeting begins!
Superhero 3 acquires Mission 2.
Superhero 1 releases Mission 1.
Mission 3 added to New Roster.
Superhero 3 leaves the Secret Room.
Superhero 2 acquires Mission 3.
Mission 1 removed from Complete Roster.
Superhero 2 leaves the Secret Room.
Mission 4 added to New Roster.
Superhero 1 acquires Mission 4.
Superhero 1 leaves the Secret Room.
Meeting ends!
Professor Z leaves the Mansion.
Mission 5 added to New Roster.
Superhero 1 leaves from Mansion.
Superhero 1 sets of to complete Mission 4!
Superhero 2 leaves from Mansion.
Superhero 2 sets of to complete Mission 3!
       :
```

A possible design and suggested components

In the context of Java, it makes sense to think of the Mansion and Rosters as **monitors**. The other components of the system could then be implemented as **active processes** (i.e., Threads).

We have made some scaffold code available on LMS, which provides:

`Producer.java` Generates new Missions and adds them to the new mission Roster.

`Consumer.java:` Removes Missions that have been completed from the completed mission Roster.

`Mission.java:` Missions 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 Ed discussion board for discussions about the project. **However, all submitted work must be your own individual work.**
- This project counts for 10 of the 50 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
1 March 2023

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 100 characters (Google Java Style Guide).