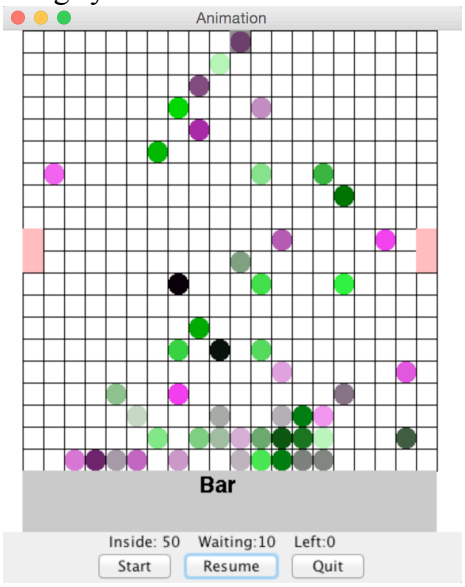
CSC2002S JAVA MULTITHREADING MODULE 2016
ASSIGNMENT 2 (CONCURRENCY)
PARTY SIMULATION

DUE DATE: **24TH AUGUST 2016, 9AM**

In this assignment, you will correct the code for a multithreaded Java simulation, using synchronization mechanisms to ensure that it operates correctly.



1. Problem Description

A good friend has approached you for help with his simulation code. The program runs a simulation of a party in a room of a specified size, where guests arrive at random times through a single entrance door. On entering the room (which is represented as a grid), a guest will immediately head to the refreshments bar to get a soft drink, and then mingle with the other guests until they happen to wander near an exit door, whereupon they leave. The rules are as listed below.

1.1. Basic rules for the simulation

- No guests must arrive before the start of the party.
- After the start of the party, people arrive at random times.
- Guest enter only through the entrance door.
- After arrival, guests must wait until the entrance door is free (unoccupied) before entering.
- After entering the venue, guests must occupy a grid square, but no more than one square.
- People may not share grid squares (at most one person in each grid block).
- A person may not move into a grid square that is currently occupied.
- People can move to any unoccupied grid square in the room, but must move one step at a time – they are not allowed to skip squares.
- Guest may only leave through an exit door.
- Guests do no leave until after they have had a beverage.
- The counters must keep accurate track of the people waiting to enter the venue, those inside, and those who have left.
- The pause button (not currently working) must pause/resume the simulation correctly on successive presses.

1.2. Additional rules

- [*Once you have done the above*] Limit the number people allowed into the room at one time and make people wait to enter if the room is full.
- People should enter in the order in which they arrived.

You are provided with your friend's Java implementation of this simulation (animationSkeletonCode.zip). The simulation runs, but a number of the rules are violated. Your task is to fix the code, identifying and correcting all the concurrency issues so that all rules are followed and the simulation suffers from no safety or liveness problems. You will then submit the code, as well as a report on what you did and why.

2. Requirements

1.3. Input parameters

The program takes as command line parameters (in order):

- the number of people invited to the venue
- the length of the room
- the breadth of the room
- (optional) the number of people allowed into the venue. (If not supplied, no limit is assumed.)

The location of entrance, exit doors and the bar are hard-coded.

1.4. Report

You need to write a concise (short) report**, in pdf format**, that explains the coding you have done. The report must contain:

- A clear list of all the concurrency requirements of the code (e.g. mutual exclusion, deadlock prevention etc.) where they occur in the code and the Java concurrency features you used to solve them (e.g. atomic variables, synchronized classes, synchronized collections, barriers etc.).
- A description of each of the classes you added and any modifications you made to the existing classes provided.
- How you attempted to identify concurrency errors.
- (*If you have time and the energy*) any additional improvements to the concurrent performance of the code you think merit extra credit. There are many things that you can do to improve this simulation (use built-in Java concurrency classes, Readers/Writers locks etc.). However, the simulation must still conform to the basic operations set out above.

2. Assignment submission requirements

- You will need to create, **regularly update**, and submit a GIT archive (see the instructions on Vula on how to do this).
- Your submission archive must consist a technical report (**pdf format**) and your solution code (including a **Makefile** for compilation).
- Upload the file and **then check that it is uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.
- The usual late penalties of 10% a day (or part thereof) apply to this assignment. The

deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.

- You may not ask your tutor for extensions or concessions – **all queries must be directed to the course lecturer.**

3. Marking

Rough/General Rubric for Marking of Assignment 2	
	Marks
Code: correct synchronization mechanisms used: data races and compound actions protected against etc. Basic simulations rules obeyed.	30
Code: additional simulation rules obeyed.	10
Documentation: New classes or class modifications documented and explained, sysnchronisation requirements and implementations clearly explained.	50
Additional features/extensions to/improvements on the simulation clearly explained and justified (and correctly implemented).	10
Total	100

Penalties	
Code does not execute/run.	**-50**
Code executes, but no GIT repository.	**-20**
GIT repository, but no regular updates (at least on 5 separate days).	**-15**
No makefile.	**-10**
Late penalty (10% per day or part thereof)	**10 x d**

Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court.