CS-206 - Programming Assignment 2

Mutual Exclusion and Java Locks

1 Peterson's algorithm

In the lectures we studied the Peterson's algorithm, which implements a lock that only allows one out of two processes to proceed into a critical session. Here we consider a variation of this algorithm that allows two out of three processes to simultaneously proceed into a critical session. Your task is to provide a pseudocode implementation for each of the `enterSession(threadId)`,`exitSession(threadId)` and `init()` methods for this case.

2 Snake Game simulation

In this exercise you will simulate a parallel version of the classic Snake game. In this version, a snake occupies a number of adjacent cells on a square cell grid. It moves through the grid by choosing a direction (up, down, left or right) and moving its head to that position. After the head cell moves, each cell in the body moves, with the first cell of the body moving to where the head was, the second cell moving to where the first was and so on. If the head tries to move to a cell that is currently occupied by another snake or itself, the snake dies and does not move any more. Please note that the snake does not move backwards, thus, at every turn it will only have three directions to chose from. The movement of a snake must appear atomic to other snakes.

You will simulate the execution of a `gSize x gSize` grid with `n` snakes with `snakeSize` cells in their bodies. Model each snake as a thread that sleeps for 100ms and tries to move to a random adjacent position: if the position is already occupied by other snake, the snake dies (its thread ends), but it remains in the grid (i.e., other snakes can collide with it); if a snake hits the end of the grid, it wraps around. After moving `nMovements` times, the snake dies, but it remains in the grid.

The snakes are initialized in a random column, with the head on the second row and the body following it downwards. Two snakes cannot be initialized in the same column. Thus, it is not possible for two snakes to overlap with each other at initialization.

You will also have a thread that prints the grid every 300ms. This thread will sleep for 300ms, and print the grid upon waking up. This thread only dies after all snakes are dead. Furthermore, snake movement must appear atomic to the printing thread.

The printed grid will have one character for each grid cell: a $\star$ if there are no snakes in that cell, or the id of the snake that occupies it. Print one row per line using a space character to separate columns.

Thus, the printed 5x5 grid with the initial position 2 snakes of size 2 will be:

```
* * * * *
* 0 * 1 *
* 0 * 1 *
* * * * *
* * * * *
```

Your program will receive the following command line arguments, in that order:

- gSize: the size of a side of the grid (the grid will have $gSize^2$ cells).

- nSnakes: the number of snakes to be in the grid
- snakeSize: the size of the snakes
- nSteps: minimum number of steps by each snake before they stop.

You have to use the `java.util.concurrent.locks.ReentrantLock` for synchronization. This lock implements the same lock semantics you saw in class.

Furthermore, AFTER the end of the simulation, your program will print a log with all movements made by all snakes, using the following format:

```
Snake n died at : (posx,posy) at timestamp1
Snake m moved from: (oldposx,oldposy) to (newposx,newposy) at timestamp2
```

where the first message indicates a snake with id `n` that died at the position with cordinates `(posx,posy)` at the time `timestamp1` and the second message indicates a snake with id `m` that moved from the position `(oldposx,oldposy)` to the position `(newposx,newposy)` at the time `timestamp2`. The `(0,0)` position is the top left cell of the grid. You can generate the timestamp by calling the `System.currentTimeMillis()` method.

Make sure to follow this template correctly, for example, you could have the following log messages:

```
Snake 0 died at : (0,2) at 1427211289653
Snake 1 moved from: (4,2) to (4,1) at 1427211289653
```

3 Submission

Deadline: 21.04.2015

Submit a pdf for exercise 1 and all source files for exercise 2 in a zipped folder.