COMP104 - 2014 - First CA Assignment
Concurrent Process Management in Java

Assessment Information

Assignment Number	1 (of 2)
Weighting	10%
Assignment Circulated	Friday 7th February 2014
Deadline	Friday 21st March 2014; 16.00
Submission Mode	Electronic
Learning outcome assessed	4, viz "Construct programs which demonstrate in a simple form the operation of examples of systems programs, including ... management of concurrent processes."
Purpose of assessment	Provide practical experience of issues in concurrent programming within Java.
Marking criteria	Scheme provided at end of document.
Submission necessary in order to satisfy Module requirements?	No

Details

For the first practical assignment you are asked to write a Java program to implement the following variation of the *Dining Philosophers* problem.

Six *microbiologists spend their working days in a laboratory observing microbe cultures, making notes about their observations and (mentally) analysing their findings, i.e. thinking. Since their department is rather impoverished, it has only been able to afford the purchase of* ***three*** *microscopes and* ***three*** *ballpoint pens. As a result these must be shared between the six. A microbiologist, however, cannot carry out any observations unless she is able to use* ***both simultaneously****. It is assumed that the employment policy of the institution discriminates against microbiologists who are not ambidextrous.*

The scenario is depicted in the diagram below.

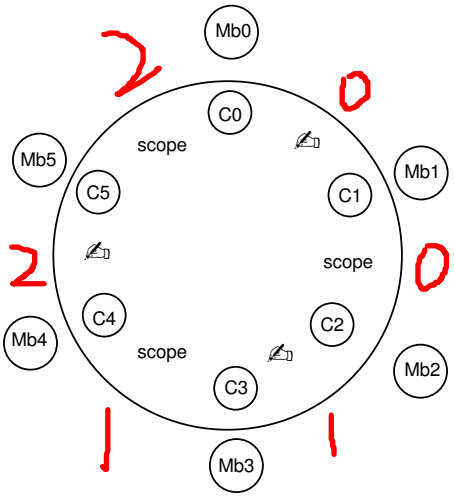

Figure 1: Both pen and microscope are needed in order to observe; neither is required in order to think

Each microbiologist has the following defining characteristics:

1. An *identifying index number* – this being an integer value from $\{0, 1, 2, 3, 4, 5\}$.
2. The *microscope* that she has available (which is shared with a neighbour): for *even* indices this is on the right-hand side, while for *odd* indices this on the left-hand side.
3. The *pen* that she has available (again shared with a neighbour): in a similar manner to the microscope convention, it is assumed that microbiologists who write with their right hand are *odd*.
4. A *minimum* and *maximum* time spent on individual *observations* (measured in seconds)
5. A *minimum* and *maximum* time spent on *thinking* about individual observations (measured in seconds)

The actual time spent on observing and thinking may be chosen as a random (integer) value between the minimum and maximum number of seconds specified. Note that this could vary between successive rounds.

Assignment Requirements

For this assignment you should implement (using the Java thread facilities) a Java program which simulates the concurrent activity defined by the scenario described above. Thus, you need to define and implement Java classes realising:

C1 The actions of a microbiologist in the setting defined. Note that there will be *six* **instances** of the *single* **class**: you should *not* write six different classes.

C2 A class describing how an individual pen is used. Again there will be *three* **instances** of this *single* **class**: one instance for each of the relevant pairs of microbiologists.

C3 In a similar style to (C2) a class describing how individual microscopes are used. Again there will be *three* **instances** of this *single* **class** each corresponding to the microscope shared by a given pair of microbiologists.

The *main()* method must instantiate the instances of the classes described above. The six instances of the class implementing the microbiologist behaviour (i.e. C1) must run *concurrently* and so the class definition must specify the *run()* method that is used.

The classes implementing pen control (C2) and microscope access (C3) must ensure (using appropriate **synchronized** methods) that, at any given time, the shared pen(s) and microscope(s) are used by *at most one* of the two microbiologists with access to these.

Your simulation should run until each microbiologist has completed 5 (*five*) sets of observations.

Output Form

The output from your program should take the form of a running commentary on the activity taking place. This *might* look something like the output below, although, provided the distinct activities (observing, thinking, etc.) are clearly indicated, its exact wording is entirely up to you.

```
Microbiologist 0 is writing with pen
Microbiologist 0 is looking through microscope
Microbiologist 0 is making observation
Microbiologist 1 is waiting for pen to write notes
Microbiologist 2 is writing with pen
Microbiologist 2 is looking through microscope
Microbiologist 2 is making observation
Microbiologist 3 is waiting for pen to write notes
Microbiologist 4 is writing with pen
Microbiologist 4 is looking through microscope
Microbiologist 4 is making observation
Microbiologist 5 is waiting for pen to write notes
...
Microbiologist 5 is writing with pen
Microbiologist 5 is waiting for microscope
...
Microbiologist 4 has finished observation 5 and is thinking
Microbiologist 3 is looking through microscope
Microbiologist 3 is making observation
Microbiologist 2 has finished work and gone to pub
Microbiologist 1 has finished with microscope
Microbiologist 1 has finished writing
Microbiologist 1 has finished observation 4 and is thinking
Microbiologist 3 has finished with microscope
Microbiologist 3 has finished writing
Microbiologist 3 has finished observation 5 and is thinking
...
Microbiologist 5 has finished with microscope
Microbiologist 5 has finished writing
Microbiologist 5 has finished observation 5 and is thinking
Microbiologist 0 has finished work and gone to pub
Microbiologist 3 has finished work and gone to pub
Microbiologist 1 has finished with microscope
Microbiologist 1 has finished writing
Microbiologist 1 has finished observation 5 and is thinking
Microbiologist 5 has finished work and gone to pub
Microbiologist 1 has finished work and gone to pub
```

Requirements

R1. Your solution must be written in **Java** and contained in a **single** file.

R2. Your implementation should make use of the Java *Thread* class as discussed in the lectures, so that the class describing the concurrent actions will be of the form

```
class NameofClass extends Thread
  {
  //
  //***************************
  // Definition of class fields
  //***************************
  ...
  ...
  //*************************************************
  // Constructor (where ..... must be developed by you)
  //*************************************************
  public NameofClass(.........)
    {
    ....
    };
  //
  //***************************************
  // Other methods that you think are needed
  //***************************************
  ...
  ...
  //***************************************
  // Specification of the class actions, ie
  //***************************************
  public void run()
    {
    .....
    }
  }
```

R3 In order to arrange correct synchronization when distinct threads are attempting to access a shared resource, e.g. the *Pen* or *Microscope*, you may **only** use the methods *wait()*, *sleep(int millisec)* and *notify()* (or *notifyAll()*). Recall that the first two of these (*wait()* and *sleep(int millisecs)*) must be embedded as,

```
try
  {
  ....
  wait();
  ...
  } catch (InterruptedException e) { }
```

Or

```
try
  {
  ....
  sleep(delay_time_value); //eg thinking or observing time
  ...
  } catch (InterruptedException e) { }
```

R4 You should use the following values for minimum and maximum observation and thinking times:

ODD biologists $5 \leq$ Observation ≤ 30; $10 \leq$ Thinking ≤ 40.

EVEN biologists $10 \leq$ Observation ≤ 20; $15 \leq$ Thinking ≤ 30.

Remember that the *actual time* spent in each round should be fixed randomly to lie between the permitted minimum and maximum values

Submission Instructions

Firstly, check that you have adhered to the following list:

1. All of your code is within a **single** file. Do **NOT** use more than one file.
2. Both your **name** AND **User ID** are clearly indicated at the start of your code, e.g. by

   ```
   // Name: My Name ; ID u?????
   ```

3. The file's name **MUST** be 'Researchers.java' (capital letter R; lower-case everything else). This means that the main class name must also be 'Researchers'. Submit **only** the Java source: design documentation, compiled .class files, sample outputs, extraneous commentary and similar ephemera are neither required nor desired.
4. Your program is written in Java, not some other language.
5. The file is a **text** file: not compressed or encoded or otherwise mangled.
6. Your program compiles and runs on the Departmental Windows system. If you have developed your code elsewhere (e.g. your home PC), port it to our system and perform a compile/check test before submission. It is your responsibility to check that you can log onto the departmental system well in advance of the submission deadline.
7. Your program does not bear undue resemblance to anybody else's. Electronic checks for code similarity will be performed on all submissions and instances of plagiarism will be dealt with in accordance with the procedures and sanctions prescribed by the relevant University Code of Practice. The rules on plagiarism and collusion are explicit: do not copy anything from anyone else's code, do not let anyone else copy from your code and do not hand in "jointly developed" solutions.

Your solution must be

SUBMITTED ***ELECTRONICALLY***

Electronic submission: Your code must be submitted to the departmental electronic submission system at:

`http://intranet.csc.liv.ac.uk/cgi-bin/submit.pl`

You need to login in to the above system and select **COMP104-1: Concurrent Processes Implementation** from the drop-down menu. You then locate the file containing your program that you wish to submit, check the box stating that you have read and understood the University Code of Practice on Plagiarism and Collusion, then click the Upload File button.

MARKING SCHEME

Below is the breakdown of the mark scheme for this assignment. Each category will be judged on the correctness, efficiency and modularity of the code, as well as whether or not it compiles and produces the desired output.

- Adherence to specification (i.e. information requested, correct naming and allowed synchronization, concurrency primitives etc.) = 10
- Implementation of Microbiologist class = 20
- Implementation of Pen class and its synchronization = 15
- Implementation of Microscope class and its synchronization = 15
- Implementation of overall Researchers class = 20
- Output commentary = 10 marks
- Comments and layout = 10 marks

This assignment contributes 10% to your overall mark for COMP104.

Finally, please remember that it is always better to hand in an incomplete piece of work, which will result in some marks being awarded, as opposed to handing in nothing, which will guarantee a mark of 0 being awarded. Demonstrators will be on hand during the COMP104 practical sessions to provide assistance, should you need it.