UNIVERSITY of STIRLING

Concurrent and Distributed Systems

Practical 0 – Refreshing Java

This lab contains a checkpoint towards the end. Please contact any lecturing staff when you reach this point to receive your credit.

It is important that you use the lab session to ask staff any questions you might have on the material. There are no tutorials!
Please do not sit and be stuck!

This lab gives you an opportunity to refresh your knowledge of Java by developing an implementation of the 'Ornamental Garden Problem' which will be used in the next lab.

The 'Ornamental Garden Problem' can be described as: A large ornamental garden is open to members of the public who can enter through either one of two gates at the lower and upper part of the garden. The management want to determine how many people are in the garden at any one time. They would like a computer system to do this.

The gates to the garden are represented by two Java programs. In fact it is only one program which is run twice in parallel to create two processes – one for each gate.

As we are interested in the total number of people in the garden, we need to have the two processes adding up their counts. A Random Access File is used for this. Each gate will write the current count to the file. The other gate reads this value from the file and increments it (one person has entered the garden) and writes the result back to the file.

You are provided with a start of a Java program. The program contains one class: the `garden_gate_problem` class.

Create a new Eclipse project named GardenGateProblem. (Start Eclipse, and select File/New/Project). Then add the source file to your project folder (in Windows). Back in Eclipse, in the Package Explorer panel to the left, right-click the GardenGateProblem project name and then Refresh (or select then F5). The explorer pane should update to show the copied source files.

Try and run the program. To do this you will need to specify a parameter. The parameter should be either ‘gate_bottom’ or ‘gate_top’ (to tell the program which of the two gates is to be simulated). Watch out for the quotes around the parameter! To simulate both gates, the program will have to be run twice. Try ‘gate_bottom’ for now (the bottom gate does the overall initialisation in the program). Afterwards, you should have a file ‘admin.txt’ in your directory. This is the Random Access File used for communication between the two gates. The program initialises this file, i.e. it creates one if it doesn’t exist and sets the value at the first position to 0 (zero). Make sure you understand the given code.

Your task is to complete the program by adding a class representing a gate.

You will have noticed that there are two lines at the bottom of the given class which have been commented out. These are there to link to your class. Thus you should create a class called ‘gate’ in the current Eclipse project. This class also needs to have access to the file ‘admin.txt’. The file is used as a shared counter between the two gates of the park. Thus the constructor needs to create a handle to access the file. You may want to check the class ‘garden_gate_problem’ for some inspiration. Further info on RandamAccessFiles can be found at https://www.digitalocean.com/community/tutorials/java-randomaccessfile-example

The second method to be added to the new class should be called ‘counting()’ (as is shown in the provided ‘garden_gate_problem’ class). This method is to actually simulate the counter. The program should simulate that 50 people enter the park. When a person enters, the current value of the counter is to be read from the file, incremented and written back. Also include **suitable printouts** so the progress of the counting can be followed. Again, check the ‘garden_gate_problem’ class for some sample code for the file access. When you try and run the program do not forget to remove the comments in the ‘garden_gate_problem’ class.

When you run the program you will see that it executes extremely fast and printouts are hard to follow. Try and slow it down using a randomly generated number (in the range of 500ms) and the sleep call (Thread.sleep() – this will need a try-catch block). Good places to insert the sleep may be after the reading and a second one after the writing instruction. Can you follow the program now?

Try and run two instances of your program in parallel (one as ‘gate_bottom’ and the other as ‘gate_top’). Does the combined counter for both gates add up to 100? Why not?

Checkpoint