Lab Session 1, Concurrency and Parallelism

Jean-Viven Millo
INRIA Sophia-Antipolis
jean-vivien.millo@inria.fr

Andrea Tettamanzi
UNSA
andrea.tettamanzi@unice.fr

March 7, 2014

Abstract

The goal of this session is to discover the basic mechanisms of multi-threading programming in JAVA. An Eclipse project containing the skeletons of the source files is available at the same address as this document. In order to realize the exercises, you will require the following documents i) this lab session subject ii) the skeleton of the source files iii) the course iv) the JAVADOC (http://docs.oracle.com/javase/1.4.2/docs/api/). We will mostly focus on the packages *java.lang*, *java.io*, and *javax.swing*.

Exercise 1: The *Thread* class

The goal of this very first exercise is to get used to the usage of the *Thread* class in Java (*java.lang*).
1/ Write a program where two threads print concurrently on the standard output (*System.out*) a different message 10000 times each.
2/ After launching the threads, the main will also print 10000 times a message.

Exercise 2: The *Runnable* interface

Write a program that prints a frame (*javax.swing.JFrame*) containing two text fields (*javax.swing.JTextField*) and two buttons (*javax.swing.JButton*). Each button is a runnable entity that is started when one clicks on it (*java.awt.event.ActionListener* and addActionListener()).

When started, it prints the content of the associated text field on the standard output every 500 ms (*Thread.sleep()*). If the text in the field changes, the printed message changes accordingly. Of course, both buttons can be activated simultaneously.

When clicking one more time on the button, it stops printing, then resumes, and so on. When the frame is closed, the two runnable entities are killed!

Exercise 3: PipedStream

Write a program where a reader thread catches the stream of characters typed on the keyboard (*System.in*). When a full line has been typed (the key <enter> has been pressed) (*java.util.Scanner*), the line is put into a pipe (*java.io.PipedOutputStream*).

At the other end of the pipe, another thread reads the content of the pipe (*PipedInputStream*) and prints it in the text area (*javax.swing.JTextArea*) of a frame, line by line.

Could you kill the reader when the frame is closed?

Exercise 4: Parallel bubble sort

Write a program which takes as input a huge array of numbers. This array is split into n sub-arrays and n threads apply a bubble sort on each of the n sub-arrays. Lastly, another thread merges the n sorted sub-arrays into one with the same size as the original array. Of course, the resulting array should be sorted.

Compare the execution of a sequential bubble sort on an array of 40000 elements with the execution of the parallel version (as described above).