THE UNIVERSITY OF SYDNEY

COMP3221 Lab 1

Multithreading

The goal of the lab is to understand how Java threads work.

Note that you can use Windows, but some labs will require the use of Linux later on. You are also allowed to use eclipse or any integrated development enrivonment (IDE) like eclipse or netbeans but you are expected to know how to use `javac` and `java` on the command line to compile and run your programs.

Exercise 1: Java threads

We will run a thread by creating a class that implements the **`Runnable`** interface. To prepare a Java thread for execution: (1) implements the **`Runnable`** interface, (2) allocate a new thread, and (3) starts it. Below is an example:

```
1  class MyThread implements Runnable {        // (1) implement the Runnable interface
2     public void run() {
3       /* here goes my thread code */
4     }
5
6     public static void main(String args[]) {
7       Thread mt = new Thread(new MyThread());// (2) allocate a new thread
8       mt.start();                            // (3) start it
9     }
10 }
```

Based on the code snippet above create a java class that starts one thread which outputs **`"hello world"`** on the standard output. (Preferably use **`javac`** to compile the **`.java`** files into bytecode and **`java`** to run the obtained class file.)

Duration: 10 min

Exercise 2: Vector-based producer consumer

Now that you know how to spawn threads, you will implement two communicating threads, a producer and a consumer. They communicate by enqueuing and dequeuing messages from a communication channel. The producer and the consumer will be written in files **`Producer.java`** and **`Consumer.java`**, respectively.

The code that spans the thread is located in file **`Factory.java`**, as indicated below:

```
1  import java.util.Date;
2
3  public class Factory {
4      public static void main(String args[]) {
5          // create the message queue
6          Channel<Date> queue = new MessageQueue<Date>();
7
8          // create the producer and consumer threads and pass
9          // each thread a reference to the MessageQueue object.
10         Thread producer = new Thread(new Producer(queue));
11         Thread consumer = new Thread(new Consumer(queue));
12
13         producer.start();
14         consumer.start();
15     }
16 }
```

Finally, the implementation of the channel interface **`Channel.java`** and the class **`MessageQueue.java`** can be downloaded from Ed https://www.dropbox.com/sh/2f5e1mxgvq3x1y4/AADbX3dHwiwD6iFNLGQlwYQYa?dl=0 and placed in the same directory as the other files. Note that this communication channel consists of a **`Vector`** storing the messages.

Both the producer and the consumer execute an infinite loop. In each of its iterations, the producer waits 0.5 second before sending a new **`Date`** message and printing a successful message containing the sent date on the standard output. In each iteration, the consumer waits 0.5 second before receiving the earliest pending message and printing a successful message containing the received date on the standard output.

Duration: 30 min

Exercise 3: Thread-safety

The producer and consumer access the same **`MessageQueue`** concurrently. Find the documentation of **`java.util.Vector`** by browsing the Java API at http://docs.oracle.com/javase/8/docs/api/. Why is the producer consumer implementation thread-safe? What happens if you replace **`Vector`** by an **`ArrayList`** (read the documentation)?

Duration: 10 min