Queen Mary University of London
School of Electronic Engineering and Computer Science

Lab 1 – Java Threads

1. Simple Threads

a) Write a class called **`Counter`** that **`extends Thread`**. The **`Counter`** should count from **10** to **500**.

b) Write a class called **`SleepyCounter`** that implements the **`Runnable`** interface. The class should behave like **`Counter`**, except that after each output it should sleep for **0.05** seconds. [**Hint**: Remember that **`Thread.sleep()`** can throw an **`InterruptedException`** that must be handled.]

c) Write a class called **`CounterApp`** with a **`main()`** method, create one **`Counter`** thread and one **`SleepyCounter`** thread, and start both of them. Run the program several times and observe its output.

d) **Questions**:

- What does this (i.e. the outputs observed) tell you about *thread scheduling*?
- What advantage does implementing **`Runnable`** have over extending **`Thread`**?
- What are *runnable* and *running*? Are they the same? Explain.

2. Interrupts and Thread Coordination

a) Write the necessary code in the **`main()`** method of **`CounterApp`** to *interrupt* your **`SleepyCounter`** thread while it is running. [**Hint:** There is a specific method that can be used to interrupt a thread.]

b) Remove the **`Thread.sleep()`** method invocation from **`SleepyCounter`**. Observe the output. Write the necessary code for **`SleepyCounter`** to detect the interrupt and immediately **`return`** from the **`run`**() method (i.e. terminate). [**Hint:** Remember that you can use the **`interrupted()`** or **`isInterrupted()`** methods to check for the interrupt flag.]

c) Add a **`System.out.println("Finished")`** statement at the end of the **`main()`** method of **`CounterApp`**. Run the code, observe the output and check where **`"Finished"`** is printed, compared to the counter output. Use the **`join()`** method to ensure that **`"Finished"`** is the final thing printed to the screen, after the **`SleepyCounter`** thread has completed.

d) Add a method called **`pleaseFinish()`** to **`CounterApp`**. When invoked, this method should immediately terminate the thread. To test your code, invoke the method from your **`main()`** method and observe the output.

e) **Questions**:

- Why does removing the **`Thread.sleep()`** method invocation result in the interrupt being ignored?
- What is the difference between using **`join()`** and **`wait()`** to coordinate threads?
- What other ways can be used to terminate your thread?

3. Synchronization

a) Read the examples **`Buffer.java`**, **`Producer.java`**, and **`Consumer.java`** from the course lecture notes. Make sure you fully understand these programs[1].

b) Modify the **`Buffer`** class, using an integer array instead of the character array to represent **`Buffer`**.

c) Modify the **`Producer`** class and the **`Consumer`** class to reflect the change above. The **`Producer`** class should send integers **0** to **50** to the **`Buffer`**. The **`Consumer`** class should receive the **50** integers from the **`Buffer`** and output them to the console. [**Note**: You can use **`System.out.println()`**.]

d) Write a class called **`BufferApp`** with a **`main()`** method, create a **`Buffer`** with size **50**, create one **`Producer`** thread and one **`Consumer`** thread and start both of them. Run the program.

e) **Questions**:

- What is the purpose of the **`wait()`** and **`notify()`** methods?
- When should you *synchronize* all methods in a class? What might be a disadvantage of doing this?
- There is also another problem that can occur with multi-threaded applications. What is it called? How can you fix that problem?

END of Lab 1: "Java Threads"

[1] They were covered in the "Threads" course topic.