Parallel Processes

Last updated: January 15th 2019, at 1.19pm

1 The Code

This week's package contains the following classes and interfaces:

- Counter.java: This is a "shared counter" Thread class. This class is incomplete (see below), but when complete a Counter thread will (attempt to) count from a start value to an end value. All Counter threads share the same internal counter, so it is possible that two or more Counter threads running concurrently might compete to change the value of the internal counter in conflicting directions. The Counter class also contains static methods that can switch tracing of Counters on or off, and static methods that can be used to affect the speed at which Counters work.

- CounterException: This is used for errors in Counters, usually in the initial values used to construct a Counter — e.g. trying to construct a Counter that attempts to count from 0 to 5 in steps of -1. It is also used to report errors in trying to set the delays used to slow down Counters.

- ThreadSet: This interface defines a set of Threads, and requires any implementation to implement a runSet() method that will run all the Threads in the set concurrently.

- ThreadHashSet: This is an incomplete (see below) implementation of the ThreadSet interface.

- Main.java: This class contains a main method that demonstrates how Counters and ThreadSets can be used. It also switches on tracing of Counters so that their behaviour can be observed.

2 Programming Exercises

- **Model question.** The run() method in the Counter class is currently just a stub. Implement the run() method so that when a Counter thread is run it will start a **while** loop to run through all the values of the counter.

Note: this is *easy!* It does not require any knowledge or experience of concurrent programming. Think of it as an exercise suited to an introductory course in Java progamming.

Have a look at the last few methods defined in the Counter class. Using these to implement the loop should then be trivial.

- The ThreadHashSet class claims to implement the ThreadSet interface, but the runSet() method demanded by the interface is also just a stub. Implement this method. It should start up all the Threads in the set, and then wait for them to stop. This is slightly more difficult, as you need to manage starting up the Counter threads, and then waiting for them to stop. See the lecture notes for information on how to do this. If necessary, use "for each" loops to iterate through all the Threads in the set:

```
for (Thread thread:  this) {
    ...
}
```

3 Demo Code

The Main class contains a main method demonstrating the use of Counters and ThreadSets. In this method a ThreadSet is populated with two Counters, one that tries to count from 5 to 10, and another that tries to count from 5 to 0. Tracing is switched on, so that when the code is run the behaviour of the Counters can be observed (if you have correctly implemeted the run() and runSet() metods).

Try running the main method a few times and observe the Counters' behaviour. Try editing the main method to change the Counter delay to low and high delays, and try running the tests again. You may observe a difference in behaviour. If you do, what is this difference, and why do you think it occurs?

4 Logbook Exercise

Edit the main method so that the Counters now try to count from 0 to 10, and from 10 to 0, in steps of ±1. Make sure that the Counters trace their behaviour.

Run this revised method a number of times, and answer the following questions. Make sure that you explain your answers.

1. Will the test always terminate? I.e. is it certain that no matter how often you were to run the test it would always end in a finite length of time?

2. What is the shortest possible output for the test, in terms of the number of lines output?

3. What is the largest possible value that the count can reach when the test is run?

4. What is the lowest possible value that the count can reach when the test is run?