Parallel Processes

Last updated: January 21st 2016, at 11.04am

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 `Counter`s on or off, and static methods that can be used to affect the speed at which `Counter`s work.

- `CounterException`: This is used for errors in `Counter`s, 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 `Counter`s.

- `ThreadSet`: This interface defines a set of `Thread`s, 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.

- `CountTest.java`: This is a tester class for investigating the behaviour of `ThreadSet`s of `Counter`s. Currently it only contains one test.

2 Programming Exercises

- The `Counter` class does not currently contain a `run()` method. Add a `run` method to the class 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* — the sort of exercise you were doing early on in Software Design & Development last year. 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 does not implement the `runSet()` method demanded by the interface. Implement this method. It should start up all the `Thread`s 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 `Thread`s in the set:

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

3 Tester Code

The `CountTest` class uses the `setUpBeforeClass` method to ensure that the `Counter`s will trace their behaviour, and to set the maximum delay between each `Counter`'s steps to one second. It uses the `setUp` method to ensure that each test starts with a new, empty, count set. Further, it currently contains only one test method, `test_5_10_and_5_0` that will:

- create a `Counter` thread that will (try to) count from 5 to 10 incrementing by one each time
- create a `Counter` thread that will (try to) count from 5 to 0 decrementing by one each time
- run both of these threads

Try running the test unit a few times and observe its behaviour. Try editing `CountTest`'s `setUpBeforeClassMethod` to change the `Counter` delay to low and high delays, and try running the tests again. You should observe a difference in behaviour. What is this difference, and why do you think it occurs?

4 Logbook Exercise

Add a new test to the `CountTest` class that will create two `Counter`s, one that tries to count from 0 to 10, incrementing by one each time, and one that tries to count from 10 to 0, decrementing by one each time. Make sure that both `Counter`s trace their behaviour.

Run this test 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?

End of intro. to concurrent systems tutorial