Operating Systems (coe628) Lab 7

March 12, 2017
Duration 1 week

Description

This is a *tutorial* lab to introduce you to the basics of multi-threading in a programming language that supports the *monitor* concept of concurrency control. (Yes, the lab is **very** simple. Its purpose is to remind you of how Java works and introduce concurrency control in a multi-threaded Java application.)

In particular, we examine how to use Java concurrency control.

A bare-bones Tutorial on Java Threads

- Like everything else in Java, a *Thread* is some kind of Object.
- Java provides a ***Thread*** class which is usually sub-classed for a specific kind of Thread.
- For example,

```
package coe628.lab7;

public class CounterThread extends Thread {

    Counter counter;
    int n = 0;

    public CounterThread(Counter counter, int n) {
        this.counter = counter;
        this.n = n;
    }

    @Override
    public void run() {
        for (int i = 0; i < n; i++) {
            counter.add(i);
        }
    }
}
```

- It looks like an ordinary class. It has two instance variables: *counter* and *n*. The *counter* instance variable is some kind of object of type `Counter` and *n* is a simple integer.

- Since it extends the class `Thread` it is reasonable to assume that it inherits useful stuff. Indeed, it does, including a method called `start` whose use we shall see shortly.
- The `CounterThread` class also implements a method called `run` which is public, returns nothing and has no parameters.
- The `run` method is absolutely essential, however; it specifies what the Thread should do when it runs.
- In this case, *run* invokes the *add* method of its *Counter object n* times.
- As we are about to see, doing this adds the integers 1 + 2 + ... *n-1* ($\sum_{i=1}^{n} = \frac{n(n-1)}{2}$).
- Here is the (initial) code for the Counter class:

```
package coe628.lab7;
public class Counter {
    int count = 0;

    public void add(int value) {
        this.count += value;
        try {
            Thread.sleep(10);
        } catch (InterruptedException ex) {
            System.err.println("Should not get here!" + ex);
        }
    }
}
```

- Basically, the add method adds “value” to the object's “count” instance variable.
- The remaining code (*try...catch..)* is boiler-plate code to deal with something called an “InterruptedException”.
- For the purposes of this tutorial, you do not have to understand this. It is just necessary for a variety of reasons.
- The main method that gets things going is shown below:

```
package coe518.lab7;

public class Main {
    public static void main(String[] args)
                 throws InterruptedException {
        Counter counter = new Counter();
        Thread threadA = new CounterThread(counter, 10);
        Thread threadB = new CounterThread(counter, 11);
        System.out.println("Starting A");
        threadA.start();
        System.out.println("Starting B");
        threadB.start();
        threadB.join();
        threadA.join();
        System.out.println("count: " + counter.count);
    }
}
```

- Two threads are created; each is passed the same `counter` object and different values of n (10 and 11).
- ThreadA will increment the counter object 45 times while ThreadB will increment it 55 times.
- In all the counter will be incremented 100 times.
- However, there is a race condition as both threads are changing the same "count" instance variable in the Counter object.
- When you run the project, you may get 100 as the final answer but you are more likely to get a lower total.
- To fix it, all that needs to be done is make the "add" method in Counter *synchronized*. This is done as follows:

```
public synchronized void add(int value) {
```

- When a method is synchronized, only one Thread at a time is allowed to execute it. This solves the race condition.

What you have to do (tutorial)

- Download the Netbeans project (a zip file) here
- Unzip the file. This creates the lab7 project.
- Run the code.
- Insert the keyword synchronized into the add method and observe that the result is now correct.
- You should now get the correct result.
- Try commenting out one or both of the "join" statements. Expalin what happens.

And Finally: Submit your lab

To submit your lab do:

1. Zip your source code files (*.java) into a file called Lab6.zip

2. Submit the zip file with the command: `submit coe518 lab7 Lab7zip`

That's all folks....