UNBC Laboratory Assignment
Computer Science 222—Winter 2017

Basic Java Threads

Due Date:

This assignment is due Monday, 2017-01-23 *at the beginning of lecture*.

Purpose

The purpose of this lab is to become comfortable with the basics of writing multi-threaded Java programs. A secondary purpose is to see how much variation the class observes in the outcome of the following experiments.

Counters

⇒ Implement a class with a integer counter variable and (unsynchroized) methods to increment and decrement the counter.

⇒ Write a program with three threads: one that increments the counter a million times, one that decrements the counter a million times, and one that prints out the current value of the counter one hundred times. Feel free to play with the numeric values and/or put `Thread.sleep()` calls in the threads.

Comment on the values you observe.

Lock Contention

⇒ Write a program with two threads: one that increments the counter 10 million times and one that decrements the counter the same number of times. Start the two threads at nearly the same time (more below) and time how long it takes for both threads to finish.

⇒ Do the same as the previous paragraph, but this time use a `Counter` class whose methods are `synchronized`, to see how much synchronization affects timing.

Thread synchronization

In order to know when a thread terminates, you need to use the non-static `Thread.join()` method. The method call `thread1.join()` causes the current thread to stop until `thread1` finishes. To have two threads start at nearly the same time, you can have their `run` methods begin with `startingGun.join()`, where `startingGun` is a reference to a `Thread` that does essentially nothing. Then the sequence `thread1.start()`, `thread2.start()`, `startingGun.start()` gives `thread1` and `thread2` chances to start running at almost the same time.

We will look at more general methods of synchronization later.

Observing variable updating errors

⇒ Write a program that launches 100 thousand threads nearly simultaneously, where each thread increments the counter once then quits. Compare the observed final counter value with the expected value.