Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 3: Multi-threading
Spring Semester 2017

Assigned on: **07.03.2017** Due by: **13.03.2017**

Overview

This week's assignment is about multi-threaded programs in Java.

- Download the ZIP file named `assignment3.zip` on the course website.
- Import the project in Eclipse: Click on *File* in the top-menu, then select *Import*. In the dialog, select *Existing Projects into Workspace* under the *General* directory, then click on Next. In the new dialog, select the radiobox in front of *Select archive file* to import a ZIP file. Then, click Browse on the right side of the text-box to select the ZIP file you just downloaded from the website (`assignment3.zip`). After that, you should see assignment3 as a project under *Projects*. Click Finish.
- If you have done everything correctly, you should now have a project named assignment3 in your *Package Explorer*.

Task 1 – Parallel Counting

Description: In this exercise we will implement a `Counter` that allows counting number of times a given event occurred. We will start with the following interface of the `Counter`:

```
public interface Counter {
    public void increment();
    public int value();
}
```

As can be seen, the counter has only two methods – `increment` that increases the `Counter` value by one and `value` that returns the current value of the `Counter`. We will use the `Counter` to monitor a progress of executing multiple threads that perform following loop:

```
for (int i = 0; i < numIterations; i++) {
  ... // perform some work

  counter.increment();
}
```

For simplicity, in this assignment no actual work will be performed and each thread will only increment the counter `numIterations` times. The implementation of above loop is already provided to you in `NativeThreadCounter` class. In what follows we will study several different approaches that implement the `Counter` such that can it can be safely used by multiple threads.

JUnit Tests: We provide JUnit tests to check the basic functionality of your solution. Please make sure that those tests are passing before submitting.

Notice: Java libraries not included in the assignment project are not allowed. Do not rename any of the provided methods in the assignment (you can create additional classes and methods).

Tasks

A) To start with, implement a sequential version of the `Counter` in `SequentialCounter` class that does not use any synchronization. That is, the counter simply increments an integer value by one. We already provide code in `taskASequential` method that runs a single thread that increments the counter. Inspect the code and understand how it works. Verify that the `SequentialCounter` works properly when used with a single thread (the test `testSequentialCounter` should pass). Now run the code in `taskAParallel` which creates several threads that all try to increment the counter at the same time. Notice how the expected value of counter at the end of execution is not what we would expect. Discuss why this is the case.

B) To fix this issue, implement a thread safe version of the `Counter` in `AtomicCounter`. In this version we will use and implementation of the **`int`** primitive value, called `AtomicInteger`*, that can be safely used from multiple threads. Run the code in `taskB` that should now produce correct results even with multiple threads.

C) Implement a different thread safe version of the `Counter` in `SynchronizedCounter`. In this version use the standard primitive type **`int`** but synchronize the access to the variable by inserting **`synchronized`** blocks. Run the code in `taskC`.

D) Experimentally to compare the `AtomicCounter` and `SynchronizedCounter` implementations by measuring which one is faster. Observe the differences in the CPU load between the two versions. Can you explain what is the cause of different performance characteristics?

E) Whenever the `Counter` is incremented, keep track which thread performed the increment (you can print out the thread-id to the console). Can you see a pattern in how the threads are scheduled? Discuss what might be the reason for this behaviour.

F) Implement a `FairThreadCounter` that ensures that different threads increment the `Counter` in an round-robin fashion. In round-robin scheduling the threads perform the increments in circular order. That is, two threads with ids 1 and 2 would increment the value in the following order $1, 2, 1, 2, 1, 2$, etc. You should implement the scheduling using the `wait` and `notify` methods. Can you think of implementation that does not use `wait` and `notify` methods?

G) (Optional) Implement a thread that measures execution progress. That is, create a thread that observes the values of the `Counter` during the execution and prints them to the console. Make sure that the thread is properly terminated once all the work is done.

Submission

In order for us to grade your exercises and give you feedback, you need to submit your code to the Subversion repository. You will find detailed instructions on how to install and set-up Eclipse for use with Subversion in Exercise 1.

Once you have completed the skeleton, commit it to SVN in a directory named `assignment3` by following the steps described below.

*https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/atomic/AtomicInteger.html

- **Check-in your project for the first time**
  - Right click your created project called **assignment3**.
  - In the menu go to **Team**, then click **Share Project**.
  - In the dialog that now appears, select **SVN** as a repository type, then click **Next**.
  - In case you have submitted Exercise 1, choose **Use existing repository location** and select the pre-defined URL in the dialog that should look like this **https://svn.inf.ethz.ch/svn/vechev/pprog17/students/NETHZ_USERNAME** Click Finish. Otherwise follow the steps in Exercise 1 to set-up a repository location.
- **Commit changes in your project**
  - Now that your project is connected with the SVN server, you need to make sure that every time you change your code or your report, at the end you submit it to the SVN server as well.
  - Right click your project called **assignment3**.
  - In the menu go to **Team**, then click **Commit**.
  - In the Comment field, enter a comment that summarizes your changes.
  - Then, click on **Ok**.