Programming Assignment 1 - StopWatch

Assignment	1. Write a **`StopWatch`** class that can be used to compute elapsed time. 2. Rewrite the SpeedTest to eliminate duplicate code and be easier to reuse. 3. Explain the results of running the 5 tasks.
What to Submit	1. Create a **private** Bitbucket repository named **`pa1`**, owned by you, and share it with `skeoop:ta` (this is the TA group). 2. Commit your source code. Please don't commit the "bin" directory, eclipse ".settings" directory, or *.class files. (See class wiki for how to use .gitignore)
Evaluation	1. Correctness of code. 2. Quality of code, including Javadoc and code format. 3. Quality of your solution to problem 2 and your explanation for problem 3.
Individual Work	Do this assignment individually. You may discuss *ideas* for solution, but not share code. You can ask TA's questions or for specific problems. All submitted work must be your own.

1. Write a StopWatch

StopWatch
+ getElapsed( ) : double + isRunning( ) : boolean + start( ) : void + stop( ) : void

Write a StopWatch class that computes elapsed time between a start and stop time. StopWatch has 4 methods:

`start( )` reset the timer and start the stopwatch if stopwatch is not running. If the stopwatch is *already* running then **`start`** does nothing.

`stop( )` stop the stopwatch. If the stopwatch is *already* stopped, then **`stop`** does nothing.

`getElapsed( )` return the elapsed time **in seconds** with decimal. There are 2 cases:

(a) If called while stopwatch is running, then return the elapsed time since `start` until the current time.

(b) If stopwatch is stopped, then return the time between the start and stop times.

`isRunning()` returns true if the stopwatch is running, false if stopwatch is stopped.

How to Compute the Time: Java has two methods to get the current time:

`System.nanoTime( )` returns the current time in nanoseconds (long). One second is 1.0E+9 nanoseconds. This is the most accurate method. The time may "wrap" back to 0 (if you are *really* unlucky).

`System.currentTimeMillis()` returns the current time in milliseconds (=1.0E-3 sec).

Your Stopwatch should use **`System.nanoTime()`** since it is most accurate.

Example:

```
StopWatch timer = new StopWatch( );
System.out.println("Starting task");
timer.start( );
doSomething( );        // do some work
System.out.printf("elapsed = %.6f sec\n", timer.getElapsed() );
if ( timer.isRunning() ) System.out.println("timer is running");
```

```
else System.out.println("timer is stopped");
doSomething( );        // do more work
timer.stop( );         // stop timing the work
System.out.printf("elapsed = %.6f sec\n", timer.getElapsed() );
if ( timer.isRunning() ) System.out.println("timer is running");
else System.out.println("timer is stopped");
System.out.printf("elapsed = %.3f sec\n", timer.getElapsed() );
```

Output:

```
Starting task
(doing work)
elapsed = 0.021188 sec
timer is running
(doing work)
elapsed = 0.050274 sec
timer is stopped
elapsed = 0.050274 sec   (no change in value because the stopwatch has already stopped)
```

2. Rewrite the Stopwatch Tasks

The file **`SpeedTest.java`** (in class **`week2`** folder) contains 5 tasks for comparing speed of common java operations:

1. append 100,000 characters to a String. Display length of the result.

2. append 100,000 characters to a StringBuilder, then convert result to String and display its length.

3. sum 100,000,000 double values from an array.

4. sum 100,000,000 Double objects having the same values as in task 3.

5. sum 100,000,000 BigDecimal objects having the same values as in task 3.

For tasks 3-5, `SpeedTest` creates an array of the values *before* it starts the StopWatch, so that the time used to create objects is not included in the time measured when we run the task. The array size is less than the loop count to avoid running out of JVM memory.

The sample code for these tasks contains a lot of **duplicate code.** Rewrite the SpeedTest to eliminate the duplicate code! Here are the guidelines:

- Apply the principle: "*Separate the part that varies from the part that stays the same.*" Then, "*encapsulate the part that varies.*" In each of the 5 tasks in SpeedTest, what part is the same? What part varies (is different in each test)?
- Read this write-up by Thai: **`http://goo.gl/TGiUqC`**
- Create a **`TaskTimer`** class that will compute and print the elapsed time for <u>any</u> task, without any duplicate code. This is described in Thai's write-up.
- Create a separate Java class for each of the 5 tasks, using the design from Thai's write-up. Each class should have a **`toString()`** method that describes the task and a constructor to initialize the task. You can defines the 5 tasks in separate source files or put them in the same source file (but separate classes) as the **`Main`** (application) class. To define multiple classes in one source file, only one class is declared "public", the others omit the "public". The Main (application) class should initialize and run the 5 tasks.
- Use the constructor of each task class to set values you need (the loop counter) and to initialize any code (like the arrays) that isn't part of the task we want to compute elapsed time for.

Now you are using polymorphism and writing DRY code!

3. Create a README.md file to explain the results

Create a **`README.md`** file in your Bitbucket repository for PA1. In this file, write the **times reported** for running each task and explain the differences. You should explain:

- why is there such a big difference in the time used to append chars to a String and to a StringBuilder? Even though we eventually "copy" the StringBuilder into a String so the final result is the same.
- why is there a significant difference in times to sum double, Double, and BigDecimal values?

Template for StopWatch

```
package stopwatch;
/**
 * A StopWatch that measures elapsed time between a starting time
 * and stopping time, or until the present time.
 * @author Bill Gates
 * @version 1.0
 */
public class StopWatch {
   /** constant for converting nanoseconds to seconds. */
   private static final double NANOSECONDS = 1.0E-9;
   /** time that the stopwatch was started, in nanoseconds. */
   private long startTime;

//TODO Implement constructor and methods
}
```

Write good Javadoc

Example of README.md

This file can contain formatting using Markdown syntax.

```
# Stopwatch by Bill Gates (5710200000) (*)

I ran the tasks on a Microsoft Surface 3 (of course), and got these
results:

Task                                  | Time
--------------------------------------|-------:
Append 100,000 chars to String        | 20.1230 sec
Append 100,000 chars to StringBuilder | 5.0880 sec
etc.                                  | ...

## Explanation of Results

I have no idea why the times are different.
```

`(*) That's his net worth in US$, not his student id.`

Markdown is a simple text formatting syntax that is used *a lot* on Bitbucket and Github. You should know how to use it. For an intro to Markdown syntax see:

- **`https://bitbucket.org/tutorials/markdowndemo`**
- **`http://dillinger.io/  (online markdown editor to practice)`**