| | |
|---|---|
| Assignment | 1. Write a **Stopwatch** class that can be used to compute elapsed time. Use the package **stopwatch** for the source code.<br>2. Write 5 tasks as described here and use Stopwatch to time each one.<br>3. Explain the results of running the 5 tasks. |
| What to Submit | Push your lab2 project to Github Classroom. |
| Evaluation | 1. Correctness of code.<br>2. Quality of code, including Javadoc and code format.<br>3. Quality of your explanation for problem 3.<br>4. Quality of your solution to problem 4. |
| Individual Work | Do this assignment individually. You may discuss *ideas* for solution, but not share code. |

## 1. Write a Stopwatch

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

| Stopwatch |
|---|
| running: boolean<br>startTime: long<br>stopTime: long |
| + getElapsed( ) : double<br>+ isRunning( ) : boolean<br>+ start( ) : void<br>+ stop( ) : void |

`start( )` reset the stopwatch and start if 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 the 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**: The **`System`** class has 2 methods to compute the current time:

`System.nanoTime()` returns the current time in nanoseconds (long). One nanosecond is 1.0E-9 second. 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 more accurate.

**Example:**

```
Stopwatch timer = new Stopwatch( );
System.out.println("Starting task");
timer.start( );
doSomething( );         // do some work
timer.stop( );          // stop timing the work
System.out.printf("elapsed = %.6f sec\n", timer.getElapsed() );
if ( timer.isRunning() ) System.out.println("timer is still running!");
else System.out.println("timer is stopped");
```

**Output:**

```
Starting task
(doing work)
elapsed = 0.021188 sec
timer is stopped
```

**Template for Stopwatch**

```
package stopwatch;
/**
 * A Stopwatch that measures elapsed time between a starting time

 * and stopping time, or until the present time.
 * @author
 * @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 you need two more attributes

//TODO Implement constructor and methods

   /** Start the stopwatch if it is not already running. */
   public void start( ) {
       //TODO
   }
}
```

Write good Javadoc

## 2. Write Tasks and explain the results

Write some common tasks to get a sense of how long they take. The tasks are:

Task 1. Sum 10,000,000 **`double`** values from an array. Initializing the array is not part of the task.

Task 2. Sum 10,000,000 **`Double`** objects from an array. Initializing the array is not part of the task.

Task 3. Sum 10,000,000 **`BigDecimal`** objects having the same values as in task 1. Use BigDecimal to accumulate the sum. To add BigDecimal objects use logic like this:

```
    BigDecimal sum = new BigDecimal(0.0);
    for(int k=0; k<array.length; k++) sum = sum.add(array[k]);
    System.out.println("The sum is "+sum.toString() );
```

Task 4. Append the Unicode characters from (char)65 to (char)65000 to a String, one character at a time using a loop: for k = 65 to limit: result = result + (char)k; then print the length of the String.

You don't need an array. In the "for" loop for this task you can use a cast: result = result + (char)k;

Task 5. Same as Task 4 but use a StringBuilder object to append the characters:

```
StringBuilder sb = new StringBuilder();
for k = 65 to limit:  sb.append( (char)k );
// convert stringbuilder to a String for comparability to Task 4
String result = sb.toString();
```

Task 6: Invent your own task of something you are interested to compute the runtime of.

## Template for Task classes

Write each task in a separate class using a common structure:

Constructor - has a parameter for the upper limit of the task (array size or maximum char value).

void run() - runs the task and prints a one line result. This should be repeatable -- if we call run() again we get the same result.

String toString() - returns a String describing the task.

Example, for Task2: sum Double objects

```
/**
* Add Double objects using an array.
*/
public class Task2 {
    private Double[] array;

    public Task2(int limit) {
        // initialize the array with values 1 ... limit+1
        this.array = new Double[limit];
        for(int k=0; k<array.length; k++)
            array[k] = Double.valueOf(k+1);
    }

    /**
     * Sum values of the array.
     */
    public void run() {
        Double sum = 0.0;
        for(int k=0; k<array.length; k++) sum += array[k];
        // print something so we know it worked
        System.out.println("the sum is "+sum);
    }

    /**
     * Describe this task.
     */
    public String toString() {
        return String.format(
        "Sum an array of %,d Double objects", array.length);
    }
}
```

### Main Class

The main method runs each task. Use a stopwatch to compute how long each task takes and print the elapsed time in seconds or milliseconds. Here is an example:

```
Stopwatch timer = new Stopwatch();
// Add elements of an array
Task1 task1 = new Task1(10_000_000);
System.out.println( task1 ); // calls task1.toString
timer.start();
task1.run();
timer.stop();
// show time in seconds (or convert to millisec for readability)
System.out.printf("Elapsed time %.6f sec\n", timer.getElapsed());
```

Note:

`Task1` to `Task3` create the array of values in the constructor, so that the time used to create the array is not part of the run time measured by Stopwatch. The max array size on your computer may be limited by the memory size of the Java Virtual Machine (JVM). You can change the memory size.

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

Create a **`README.md`** file in the top directory of your repository. 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?

### Example README.md

This file can contain formatting using Markdown syntax.

```
# Stopwatch Tasks

I ran the tasks on a Microsoft Surface Pro, and got these results:

Task                                       | Time   |
-------------------------------------------|-------:|
Task 1: add 10,000,000 double primitives   | 0.888 sec
Task 2: add 10,000,000 Double objects      | 6.123 sec
Task 4: append 65,000 chars to a String    | 2.0880 sec
etc.                                       | ...

## Explanation of Results

Task 1 is faster than Task 2 because ...

Task 1 and 2 are faster than Task 3 because ...
```

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

- https://guides.github.com/features/mastering-markdown/
- https://www.markdownguide.org/cheat-sheet/
- **http://dillinger.io/ (online markdown editor to practice)**