Assignment	1. Write a **`Stopwatch`** class that can be used to compute elapsed time. 2. Rewrite the FileTest to eliminate duplicate code and be easier to reuse. 3. Explain the results of running the tasks
What to Submit	1. Create a repository named **`filereader`** (lowercase) on Github. 2. Commit your source code. Please do **not** commit the "**bin**" directory. Use a `.gitignore` file to prevent this. 3. Write a good README.md that describes project and your answers.
Evaluation	1. Correctness of code. 2. Quality of code, including Javadoc and code format. 3. Quality of your explanation for problem 3. 4. Quality of your solution to problem 4.

1. Write a Stopwatch

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

Stopwatch
+ getElapsed( ) : double + isRunning( ) : boolean + start( ) : void + 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**, as accurately as possible.
(a) If the stopwatch is running, then return the 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 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 time %.6f sec\n", timer.getElapsed() );
```

2. Write Two File-reader Methods and Time the Results

1. Read a text file one character at a time. Append all the characters to a String and return the String.

`public static String readFileToString(String filename)`

* The method should catch I/O exceptions and print a message. Return an empty string (not null).

* Always close the file before returning.

2. Same as above, but read the file to a StringBuilder object.

`public static String readFileToStringBuilder(String filename)`

* The method should catch I/O exceptions and print a message. Return an empty string (not null).

* Always close the file before returning.

* To append characters to a `StringBuilder` object, use the append( ) method.

* To return the result as a String, just call `stringBuilder.toString( ).`

3. Read the file as text one line at a time using a BufferedReader, and append the resunt to a String.

A BufferedReader uses another Reader object (reads characters) as input and outputs line.

* Read the input *one line at a time* using the readLine() method. Append each line to the String containing file data.

* readLine() removes the line-end character. To make the String result the same as other methods, you should append a `\n` character after each line, e.g.: result = result + line + '\n';

* Return the result as a String.

* Always close the file before returning.

4. Write a main( ) method. For each task you should:

a) print a description of the task, including filename

b) start the stopwatch

c) do the task

d) stop the stopwatch

e) print the result and elapsed time.

```
Reading Alice-in-Wonderland.txt using FileReader, append to String.
Read 52,539 chars in 1.859887 sec.
Reading Alice-in-Wonderland.txt using FileReader, append to StringBuilder.
Read 52,539 chars in 0.007097 sec.
Reading Alice-in-Wonderland.txt using BufferedReader, append lines to String.
Read 52,539 chars in 0.042596 sec.
```

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

Create a **`README.md`** file in your repository. In this file, write the **times reported** for running each task and explain the differences. You should explain why the difference in times. What is causing one to be faster or slower?

Example README.md

This file can contain formatting using Markdown syntax.

```
# Input-Output Tasks
by Bill Gates

I ran the tasks on a Microsoft SurfaceBook with 2.4Ghz i5-7200U,
and got these results:

Task                                            | Time
------------------------------------------------|-------:
```

```
Read file 1-char at a time to String          |  1.8598 sec
Read file 1-char at a time to StringBuilder   |  0.0071 sec
Read file line at a time using BufferedReader |  x.xxxx sec

## Explanation of Results

explain why some tasks are slower than others.  what are the factors?
```

Markdown is a simple text formatting syntax that is used on Github, Bitbucket, and many other places. You should know how to use it. Github and Bitbucket both have Markdown tutorials.

4. Rewrite the Tasks to Use Polymorphism

The class contains a lot of duplicate code, and we can't easily use it to time different tasks.

Rewrite the FileTest class to a) eliminate duplicate code, b) use polymorphism to enable code reuse.

Create a **`TaskTimer`** class that performs any *Runnable* task and prints results.

Hints:

- "*Let's Eliminate Duplicate Code*" by Thai: **`http://goo.gl/TGiUqC`** Explains why and how to solve this problem.
- Apply the principle: "*Separate the part that varies from the part that stays the same.*" You want to separate the tasks (the thing that varies) from the code that runs the task and computes elapsed time (the part that stays the same). Then, "*encapsulate the part that varies.*"
- Write one class for each task (reading a file) as a class that implements *Runnable*. The run() method does the work. The constructor prepares the task and saves data, toString() is used by TaskTimer to describe the task.
- Create a **`TaskTimer`** class that will compute and print the elapsed time for *any* task, without any duplicate code.
  - print description of the task (get description from the task's own toString method)
  - run the task and measure the elapsed time
  - print the elapsed time

Write a Main class that creates task objects, creates a TaskTimer, and then uses TaskTimer to run the tasks.

When you finish, your code should have (almost) no duplicate code and use polymorphism.