

# Timing Vector Additions

Tuesday, January 31, 2012

Lab 2

CSC 062: Spring, 2012

---

## Introduction: Vectors and Stopwatch

---

In this laboratory, we will use a `Stopwatch` class to measure the efficiency of the `Vector` class. Specifically, we want to see how execution speed is affected by the `increment`. Recall that `increment` is the amount by which the underlying data array is lengthened when the vector requires more space. If `increment` is set to zero, then the size of the data array is doubled. We'll be using the `Vector` class since the `ArrayList` class only doubles the array length and does not give you incremental building as an option.

We encourage you to work in pairs on this lab as it is useful to learn from others and two pairs of eyes on a program are more likely to find errors. Having someone to discuss the results of your program will also make it more likely that you'll get a deeper understanding of the results.

---

## Stopwatch Class

---

You will be using a `Stopwatch` class written by CS 62 faculty to collect the running time for programs you write today. The `Stopwatch` class has a parameterless constructor and methods:

- `void reset()`, which sets the time to zero
- `void start()`, which starts timing
- `void stop()`, which pauses the timing, and
- `getTime()`, which returns the elapsed time in nanoseconds between various starts and stops.

Getting the running time in nanoseconds is a bit of overkill, so we recommend either dividing the returned time by 1000 to get milliseconds or even by 10,000 to get centiseconds.

---

## Your Program

---

Begin by closing all of your open Eclipse projects, so that errors in them will not affect your work today. Use `Projects/Close`.

Create a new Eclipse project named `Lab2`. Remember to add the `BAILEY` variable to your project. Next, copy the file `/common/cs/cs062/labs/lab2/StopWatch.java` into the `src` directory in your new Eclipse project and select `File/Refresh`.

Start a new class `VectorTimer`, which will be simply a vessel for the `main` method and a few other static methods:

- `public static long run(int maxSize, int increment)`

The `run` method creates a new empty vector of type `Vector<String>` with the specified increment. It returns the time that it takes to add `maxSize` strings to the `Vector`. Use the `Vector<String>` method `add`, and always add the same constant string—your name, for example. To attempt to minimize the impact from garbage collection add the line: `System.gc()`; in your `run` method right before you start the timer. Also don't forget to put

```
import structure5.Vector
```

at the beginning of your file or you will get error messages at every mention of `Vector`.

- `public static ArrayList<Long> trial(int size, ArrayList<Integer> incrs)`

The `trial` method compares the results from `run` for a fixed size and varying increments. It makes one call to `run` for each entry in the `incrs` vector. The results are returned in an `ArrayList` whose size is the same as that of `incrs`.

- `public static void main(String[] args)`

The `main` method runs several trials and prints the results. Start with increments of 1, 10, and 0; and sizes of 0, 5000, 10000, 15000, . . . . You may want to adjust the sizes when you see the results.

*We use static methods and static variables when there is no need for the class to create more than one object. Simply add “static” to the declaration of all instance variables and methods, and then you can call them directly from main without having to call a constructor first.*

*In programs structured this way, the main method does NOT create a new object from the class constructor (notice we don’t have one!). Instead, it contains the commands that might otherwise be put in the constructor, and methods are called directly, e.g., `trial(size,increments)` rather than having it sent as a message to an object.*

## Some hints on getting good results

The nature of Java and the Java Virtual Machine make getting good, consistent results difficult. When a Java program is first run, the JVM will read each instruction to be executed, and interpret this instruction. However, the JVM can also use a technique known as JIT (Just in Time) compilation. As a program continues to run, the JVM will continually take measurements, and will compile often used code directly to machine code, eliminating the need for interpretation. This compiled code will run much, much faster than the interpreted code.

So..when running timing experiments in Java, you should “warm up” the JVM by running the test several times (in the same execution). We suggest that set up your code to perform the test about 20 times, and only run, or keep, the timing measurements from the last run. This should ensure that the test code has been compiled, and that you are getting the most accurate information.

Also, you should close all of your running programs. In addition, you should run your program multiple times (remember, each run has 20 iterations of the test to “warm up” the JVM), and keep the *best time of all these runs*. This seems like cheating, but it’s not. In truth, the best time we get is closest to the real answer. The reason is that your computer has dozens, or even hundreds, of processes running at all times. Even when you have shut down everything that you can, there are many processes that are part of the operating system that are running in the background. The fastest times from your test will represent the test runs least affected by these other processes.

## Presenting and submitting results

Present the output in a table like the one below; see the note below about formatting. The nanosecond precision of `Stopwatch` is too fine; you will need to adjust the scale of the timing values as they are printed, which can vary computer to computer. As suggested above, dividing by 1000 or 10,000 is probably a good choice.

You will need to submit a text file with your results to the dropbox folder. If you have chosen to work in pairs, then your file should be named something like “`Lastname1-Lastname2-Lab2.txt`” where your lastnames are used in the filename. Remember not to use spaces in the filename.

size	linear (1)	linear (10)	double
0	0	0	0
5000	148	14	1
10000	580	58	0
15000	1321	132	0
20000	2733	267	1
25000	4863	491	1
30000	7781	781	1

## A note on formatting textual output.

The object `System.out` has type `PrintStream`, which in turn has a method `format`. `format` is very general and makes it easy to print the lines in the table. The call

```
System.out.format("First: %8d, second: %-12s%n", num, str);
```

creates a string and prints it. The string is formed by

- replacing `%8d` with the numerical value of `num`, right justified in a field eight characters wide, and
- replacing `%-12s` with the string representation of `str`, left justified in a field twelve characters wide.

If `num` and `str` are 47 and XLVII respectively, then

```
First:      47, second: XLVII
```

is the result of the method call above.

The letters after the percent sign, `d` and `s` in this example, indicate the kind of data being formatted; they are not variables. The sequence `%n` is the OS independent newline character. You may have as many `%` expressions in the format string as you want; they are matched with the arguments that follow. There are *many* more options for format strings; see the Java documentation for the classes `PrintStream` and `Formatter` or the tutorial at <http://java.sun.com/docs/books/tutorial/java/data/numberformat.html> for more information.

---

## Understanding the results

---

Be sure to shut down all other processes on your computer (including web pages) before running your program as other processes running can affect the running time of your program.

When you have the results, please find space on the side wall of the room and post your results there.

Look at the results of your program. Are the times monotonically increasing as the size increases? If not, why do you think that is?

What happens as the size of the input doubles in each of the three columns?

We will discuss the significance of your results, and those of your classmates, as they appear. Some things to think about: what is the running time (i.e. Big-O running time) of `increment` vs. `double`? Does your data accurately reflect this?

---

## More fun...

---

Once you've got all this working, if you have time we can try out a few additional things:

- What happens with other increments (besides 1 and 10)? Can you predict what the results will look like, for example what do you think a column headed `linear (100)` would look like?
- Rather than just running one experiment per setting, you can run multiple experiments (say 5 or 10) and average the results in your run method. This will be a bit slower, but should give you more accurate results.
- It may be interesting to compare the performance difference between `ArrayList` and `Vector`. `ArrayList` does **NOT** allow you to adjust the increment size; it always doubles the size. However, you can compare the performance of `Vector` vs. `ArrayList` for doubling sizes. Which is faster?