# Computer Science 62
# Lab 2

Wednesday February 3, 2016

In this laboratory, we will use our `Stopwatch` class to measure the efficiency of the `Vector` class. Specifically, we want to see how execution speed is affected by the `increment` parameter. 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. Before you start coding make sure you look over the documentation for the `Vector` class.

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

2. Create a new Eclipse project named `Lab2`. Remember to continue to the window in which you can add the `BAILEY` variable. Next, copy the file `/common/cs/cs062/labs/lab02/StopWatch.java` into the `src` directory in your new Eclipse project and select `File/Refresh`.

3. Create a new class `VectorTimer`. This class will contain only a `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, using `new Vector<String>(0, 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.

   -
     ```
     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. *Don't forget that Java uses just-in-time compilation so you'll need to first run several trials and discard the results.*

4. Present the output in a table like the one below; see the tutorial 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 from computer to computer.

```
    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
```

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?

**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.