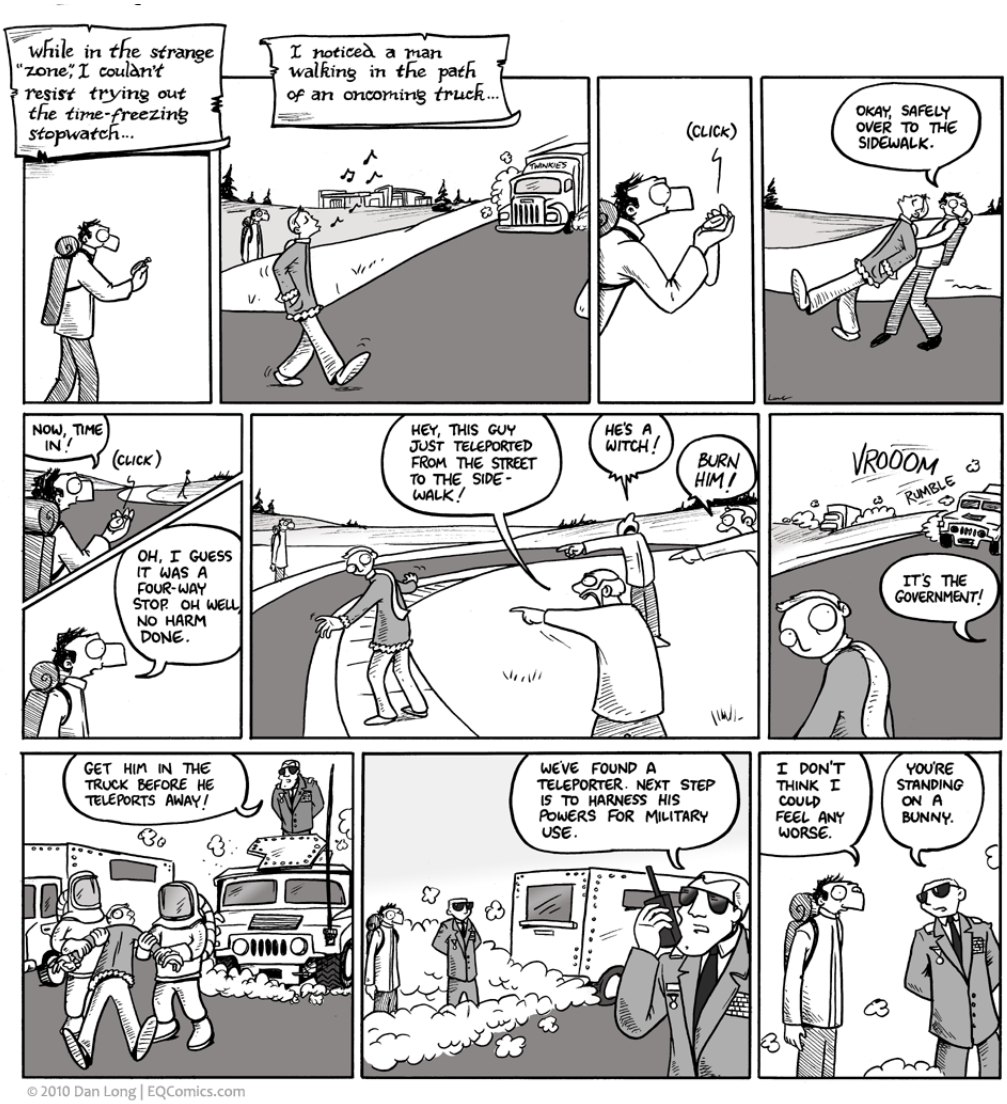
CS201 - Assignment 6, Part 1
Due: Friday, April 4, *at the beginning of class*



http://eqcomics.com/2010/02/09/the-stopwatch/

For this assignment you may (and I'd encourage you to) work with a partner.

Timing `ArrayList`

For part 1, we will use our `StopWatch` class to measure the efficiency of the `ArrayList` class. Specifically, we want to see how execution speed is affected by the how much we increase the underlying array size when we run out of room.

At:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs201/assignments/assign6/part1/`

I have put an `ArrayList` implementation that includes a `setIncrement` method, which allows you to set the amount by which the underlying data array is lengthened when we require more space. If `increment` is set to zero, then the size of the data array is doubled. If it's any non-zero number, the underlying array increases by `increment`.

Create a new Eclipse project named something like `Assign6.1` and add the `ArrayList` class and the `StopWatch` class (look at the code examples in the "Resources" section of the course web page).

Create a new class `ArrayListTimer`, which will be just be a container for the `main` method and a few other static methods. Then, write the following methods:

- `public static long run(int maxSize, int increment)`
  The `run` method creates a new empty ArrayList of type `ArrayList<String>` with the specified increment. It returns the time that it takes to add `maxSize` strings to the ArrayList Use the `add`, and always add the same constant string—your name, for example.
- `public static ArrayList<Long> trial(int size, ArrayList<Integer> incrs)`
  The `trial` method compares the results from a `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.

Present the output in a table like the one below; see the section below about formatted printing. 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.

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

What to turn in

When you're done, submit a .txt file with assignment number 6.2 with a table like that above with your results and a *very* short analysis of the results. Specifically, do they agree with our theoretical analysis?

More fun...

Once you've got all this working, if you want to experiment a bit more you 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.

A note on formatted printing

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.