CS61B, Fall 2008 **HW #4** **P. N. Hilfinger**

Due: Wednesday, 1 October 2008

Create a directory to hold your answers. There is a skeleton for your solutions in the repository under `staff/hw4`, and also in the directory `~cs61b/code/hw4`. Use the usual command sequence to copy your final solution to a `hw4-`*N* entry in your tags repository directory.

The purpose of this week's homework is to get you thinking about and working on the first project. Trust me: you do *not* want to wait until the last minute to do this thing!

1. The Project #1 skeleton suggests the use of an abstract class `Command`, whose subtypes would correspond to specific kinds of command: `rect`, `sin`, `draw`, etc. Since all commands have essentially the same syntax, you might expect that there should be common code shared by all commands for use in parsing commands. One possible structure looks like this

```
abstract class Command {
    Command (ArrayList<Command> operands) {
        this.operands = operands;
    }

    abstract Value execute (Interpreter machine);

    protected ArrayList<Command> operands;
}
```

And now, each child of `Command` will have access to its operands, and will have a constructor such as

```
class ColorCommand extends Command {
      ColorCommand (ArrayList<Command> operands) {
          super (operands);
          maybe check to see if operands are valid for me.
      }

      Value execute (Interpreter machine) {
          execute this ColorCommand according to its operands.
      }
}
```

For this to work, you need to be able to take a `Scanner` and read `Command`s from it. In the skeleton file for the class `drawing.Interpreter`, we've suggested having a `readCommand` method that does this. In this homework, you'll provide an implementation for a stripped-down version of the project skeleton. You can freely adapt it to your project if you choose.

Start with the skeleton `hw4` directory in the staff repository or in `~cs61b/code/hw4`. Fill in the indicated places to make 'java HW4' work (it should read a command containing only '`+`' and '`*`' operators and numerals from a file and print the result of evaluating it).

2. The project #1 statement calls for a user manual (called `UserManual`) containing *your own* description of how to use the project, and an internals manual (called `INTERNALS`), containing a guide to your implementation (intended for people who want to maintain or extend it). Provide drafts of both (the more you do here, the less hassle you'll have at the project due date). Naturally, you'll have to have thought through how you're going to implement your project. (Warning: normally, we're not terribly upset at people working together on homework, but projects are different. Make *sure* that your documents are your own work.

3. The project #1 statement indicates that you must provide tests of your implementation. Provide a reasonable draft set of tests (`.drw` files and corresponding `.std` files for the `drawing-tests` directory.) Of course, you might reasonably wonder how you will know the precise values of numeric quantities without the exact implementations, but make a reasonable effort. How can you test the arithmetic operations, given that nothing in the drawing language specifically prints out numbers?