Handout 8
CSCI 136: Spring, 2005
14 March

Lab 6

Due 10am, Wednesday 6 April

P.S. It's Just a Stack

1 Short Answers

Bring answers to the following questions *to class on Wednesday*.

9.3
9.4
9.5

2 Lab Program

This week we will implement a small portion of the stack-based language Postscript, a language designed to describe graphical images. When you create a Postscript file with gnuplot or other tools, you actually create a file that contains a program, written in this language. A printer or viewer interprets that program to render the image described by the file. The lab is described in Section 9.5 of *Bailey*.

Read the assignment and prepare a design for the program *before Wednesday's class* so that you can start working right away.

3 Notes

1. The starter files and javadoc documentation are on the handouts page. Familiarize yourself with these files *before* starting to work on your lab.
2. Make use of the functionality of the classes you are given. Be careful not to spend time developing code that is already there!
3. Name your interpreter class `Interpreter`. You should only need to modify the `Interpreter` class, and nothing else. Your program should read commands from standard input. You can also redirect input from a file by using a command like `java Interpreter < program.ps`.
4. Make your `main` method very short. All it should do is create an `Interpreter` object and tell it to start parsing the postscript program presented at the command line. Create a method `interpret` that takes a single parameter of type `Reader` and processes the PostScript tokens returned by that `Reader`.
5. Develop your `interpret` method incrementally. Get your simple push, pop, and pstack operations working, then move on to the arithmetic operators, and finally the definition and usage of symbols.
6. Your program should throw exceptions when it encounters invalid input, but these should contain meaningful error messages. You can use `Assert.condition()` and `Assert.fail()` for this.
7. Implementing the basic operations– `pstack`, `add`, `sub`, `mul`, `div`, `dup`, `exch`, `eq`, `ne`, `def`, `pop`, `quit`, and `ptable`– will allow you to earn 18 out of 20 points. You can earn the last two points by implementing the extensions outlined in thought questions 3, 4, and 5. In particular, you should implement *procedure* definitions and calls, and the `if` instruction. These extensions may require a little thought, but ought to be straight-forward to implement if you have designed your interpreter engine well.

8. You can use the `gs` interpreter on the computers in the lab if you want try out the commands yourself. Type the command `gs -dNODISPLAY`. This will give you a text-only postscript interpreter. You can type commands at the prompt as they appear in the lab assignment. Type `quit` to exit the interpreter.

4 Deliverables

Turn in your well-documented `Interpreter.java` file. You *do not* need to answer the thought questions this week.