Computer Science 501
Data Structures & Algorithms
The College of Saint Rose
Fall 2013

# Lab 7: P.S.: It's Just a Stack

**Due: 6:00 PM, Wednesday, November 13, 2013**

This week's lab assignment is described in Section 10.5 of Bailey. You will learn about the PostScript language, and gain experience using stacks and making use of an existing partial implementation.

You may work alone or in a group of 2 or 3 for this assignment.

---

## Getting Started

First, carefully read the lab description in the text and this handout. You are certain to have questions, so be sure to leave plenty of time to discuss this assignment with me.

Before you begin, copy the starter files from the the shared area under `labs/postscript` (they will also be emailed to you). Familiarize yourself with these three classes and make sure you understand how they work.

Next, develop a sketch of the `Interpreter` class. This is where you will process the tokens being delivered to your interpreter by the provided `Iterator` class. You are strongly encourages to either see me to go over your sketch of this class or email it to me for comment and discussion before you continue.

---

## Notes and Guidelines

- Please name the file with your interpreter and `main` method `Interpreter.java`.
- Make use of the functionality of the classes you are given to start. Be careful not to spend time developing code that replicates functionality that is already there!
- 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 `java.util.Iterator<Token>` (which will be an instance of the provided `Reader` class) and processes the PostScript tokens returned by that `Iterator<Token>`.
- A "debugging mode" for your program may prove useful to facilitate debugging and testing. In my version, if I specify the word "debug" as a command-line parameter, the program enters a debugging mode and prints out informative messages about what it is doing as it processes the PostScript tokens.

- Develop your `interpret` method incrementally. Get the simple opertations like pushing a token encountered on the input, `pop`, and `pstack` working, then move on to the arithmetic operators, and finally the definition and usage of symbols.
- 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.

---

## Bonus Opportunity

For a bonus point, you can implement procedures as described in thought question 3. If you design everything else properly, this should be almost a trivial extension.

For another bonus point, you can implement the `if` operator as described in thought question 4.

---

## Related Questions

Include your answers to thought questions 1, 2, and 5 in a comment at the top of `Interpreter-.java`. (Note: while you don't have to do questions 3 and 4, you need to read and understand them to be able to do 5.) This means you need only submit that one file, as you should not need to modify the provided classes.

Before 6:00 PM, Wednesday, November 13, 2013, submit your Java program for grading. There are two things you need to do to complete the submission: (*i*) upload a copy of your Java program (the `.java` file only) using Submission Box at `http://sb.teresco.org` under assignment "Postscript", and (*ii*) print and turn in a hard copy of your program.

Don't forget to check your programs for compliance with the Style Guide for CSC 501 Programs

---

## Grading

This lab assignment is graded out of 30 points.

| Grading Breakdown | |
|---|---|
| Program design | 3 points |
| Program style | 3 points |
| Program documentation | 6 points |
| Program correctness | 12 points |
| Procedure support | 1 bonus point |
| `if` operator support | 1 bonus point |
| Thought questions | 6 points |

The program design grade will be based on the design choices you make in the implementation of the `Interpreter` class. The program style grade will be based on code formatting and approriate use of Java naming conventions. The program documentation grade is, of course, based on the comments you provide. The program correctness grade is based on how well your program meets the functionality requirements.