Homework 1 — Warm-up exercise in Java programming: Emacs Calc look-alike — assigned Tuesday 24 August, due Thursday 2 September

Task: Write a Java program to emulate Emacs Calc.

In detail:

Emacs Calc is a sophisticated calculator package for use within the Emacs editor. It is installed on UNM CS machines, and can be invoked from Emacs with the command `M-x calc`. It has an extensive help facility. If you wish, you can download the source code from
`http://www.synaptics.com/people/daveg/`. A printed manual is also available, at
`http://www.cs.unm.edu/~darko/classes/2004f-351/calc.pdf`.

Your calculator only needs to cover a tiny subset of Emacs Calc's functionality.

The following commands need to be recognized (note that our syntax is slightly different from Emacs Calc):

- numbers
  - Only integers are accepted, but they may be given in an arbitrary radix (see below). All numbers are internally stored as Java type `long`. If an input number cannot fit in the Java type `long`, an exception should be raised and handled, and the stack left unchanged. Numbers are displayed in the current presentation radix, which is initially 10.
- binary arithmetic operators
  - `+`
  - `-`
  - `*`
  - `/` (integer division, as in Java)
  - `^`
- unary mathematical functions
  - `Q`, $\sqrt{\cdot}$ (integer part of the square root)
- stack manipulation commands
  - `swap`, to swap the two topmost items on the stack
- presentation setup commands
  - `radix` *n*, where $2 \leq n \leq 36$, to set the presentation radix (number system base) to *n*

Unlike in Emacs Calc, typing a single-character operator such as `+` does not take effect immediately; the user must hit carriage **return**.

As in Emacs Calc, letters and digits are used for radices greater than 10, and the radix is prefixed onto the numeral when the radix is other than 10. Thus, the number fifty is displayed as 2#110010 when the presentation radix is 2, as 3#1212 when the presentation radix is 3, as 50 when the presentation radix is 10, as 16#32 when the presentation radix is 16, or as 36#1E when the presentation radix is 36. The same format must be accepted as input. Additionally, it is legal to have an explicit prefix in the input for radix 10 numerals, and both lowercase and uppercase letters are legal.

The program must provide a textual (no GUI) interface to the user. The program will display the complete stack contents, one element per line, bottom to top, with a pictorial stack bottom drawn as `-----`, and an indicator of stack depth for each element, in other words, exactly as in Emacs Calc. The user will enter commands as described above. (Emacs Calc has finer control over the screen and can overwrite text; your program will not do that.)

Any arithmetic exceptions that arise during computation must be caught and handled, such that the stack is unaffected and contains the same elements as before the offending command was executed.

Your code must be placed in a package named `edu.unm.cs.cs351.fall2004.`*yourusername*`.calc`. The main class of your program must be called `Calc`. The program does not accept any command-line arguments. Thus, it must be invoked simply as:
yourfavoritejvm `edu.unm.cs.cs351.fall2004.`*yourusername*`.calc.Calc`.

How to turn in

Turn in your code by running

˜jbrown/handin your-file

on a regular UNM CS machine.

You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft. Normally, you will turn in a top-level directory containing at least the following:

- a `README` file (a brief description of the structure and purpose of the code, with basic instructions for its use);
- a `COPYING` file (a copyright notice or something functionally equivalent);
- a `TODO` file (a list of known bugs and deficiencies with respect to the assignment, or identified but unimplemented extensions);
- a `src` directory containing the Java source hierarchy for your code;
- a `classes` directory containing the compiled Java class files hierarchy, exactly corresponding to the `src` files;
- a `doc` directory containing various detailed project documentation as needed (including a detailed description of code structure (classes, interfaces, and how they are related) in a file `structure.txt` or, preferably, `structure.ps`, a description of the design process that led to this code structure in a file `design.txt` or, preferably, `design.ps`, and a subdirectory `generated/javadoc` with Javadoc-generated HTML files, exactly corresponding to the `src` files);
- a `tests` directory with input files used for testing the correctness of the program, including both valid and invalid inputs;
- a `test-results` directory with output files generated by the program on test inputs; and
- a `RESULTS` file (a summary of the testing process, documenting the program's compliance with the specification).