COLBY COLLEGE
Department of Mathematics
and Computer Science

CS231
Fall 1998

Allen B. Downey

Assignment 7: Process Objects

Due: Monday 9 October

Readability

1. If your translator/evaluator from last week didn't work completely, get it working, using my solutions if you want to.
2. Since it has been a few days since you worked with this code, look it over and identify places that are hard to read, either because the syntax has gotten hairy or the logic is convoluted.
3. Do whatever you can think of to make the program easier to read (and thereby easier to make correct). Some suggestions: create new methods to reduce nested structure. Use indentation and spacing to clarify structure.
4. Add comments at the beginning of every method stating the preconditions for that method (even if there are none). Add any additional comments in places where there is something non-obvious happening.

Robustness

There are two ways to improve the error-handling of your program: (1) survey the code for operations that can cause errors (like `pop`), and make sure that they are protected by a precondition (like `!stack.empty()`), and (2) test a variety of erroneous inputs and try to break it.

1. Perform both of those steps, in that order. What other operations besides `pop` can cause errors?
2. Try out some test cases and see if you both catch the error and produce a useful error message. Your error messages should make sense to the user, not the programmer. For example, "Not enough arguments for the operator +" is better than "Stack empty while translating to postfix."

Keep in mind that the hypothetical user is just using a calculator; he doesn't know that you are translating to postfix and he has no idea what the stack is.

Add a new operator

1. Do Exercise 4.2.2 on page 113 of Standish.
2. Use your solution to the previous problem to add a new operator, $\wedge$, to your calculator. It should raise the first operand to the power specified by the second operand.

Add a new capability

1. Modify your program so that it handles multiple-digit operands.

Restructure your code

Reorganize your program as a system of Process Objects.

1. Divide your code into three classes, Calculator, Translator and Evaluator. Calculator will contain `main` and `inputLoop`. Translator will be a process object for translating infix to postfix. Evaluator will be a process object for evaluating postfix.

   Each time through the loop, Calculator will create a new Translator object to perform the translation and a new Evaluator object to evaluate it.

Here is what the public interface looks like for my implementation:

```
public class Translator {
  private int index;
  private String line;
  private StringBuffer out;
  private Stack stack;

  public Translator (String line_);
  public StringBuffer translate ();
}

public class Evaluator {
  private Stack stack;
  private String line;
  private int index;

  public Evaluator (String line_);
  public Evaluator (StringBuffer line_);
  public int evaluate ();
}
```

Notice that the instance variables for the Translator and Evaluator objects encapsulate the state of the translation/evaluation process. They are private because no one outside the class should be mucking about with them.

Also notice that Evaluator has two constructors so that it can handle arguments that are Strings or StringBuffers. That was handy for debugging.

In normal operation, the translate method and the evaluate method will only be invoked once, but you might want to give some thought to dealing with repeated invokations.

Review the code

1. After reorganizing, review the code for things that can be cleaned up or places where you can take advantage of the new structure. For example, I found that when all the state variables

are visible from all functions (since they are instance variables) I was able to reduce the number of arguments I passed around, which made it easier to create new submethods.

Also I was able to generate better error messages, because even deeply nested methods were aware of global state.