Assignment 3

Tue Feb 16 (125 points)

For this assignment, we will create an **interpreter** for the PicoScript language. It works by reading tokens one at a time, and either pushing them onto the operand stack or invoking some operator. The language specification has been updated in the 'Library' section to describe approximately 30 built-in operators.

Start by watching these two videos, which demonstrate the way the lexer and interpreter work together.

1. Assignment 2 solution [51:10]
2. Assignment 3 overview [1:15:24]

As you can see, I provided the basic structure of the interpreter. You should get the rest of it working:

- Implement the operators in the library specification. They will all be methods in the `Interpreter` class, similar to the ones I did for you: `add`, `mul`, `def`, and `dup`. However, some of them may require additional facilities in the `Value` class and elsewhere.
- For example, the section on relational operators describes Boolean values, which I did not support yet. You will need to add a new `Value` type `BOOL` and a matching field and constructor. Then update the `toString` method to print Booleans.
- Each operator defined in the library specification comes with a series of examples of its usage. Those examples are also unit tests in the classes that end with `Operators`, such as `ArithmeticOperators` and `StackOperators`.
- Finally, try defining **one or more** of these procedures **in PicoScript** (not in Java):
  - `pow` should calculate integer powers. For example:

    ```
    2 8 pow     % [256]
    5 6 pow     % [15625]
    7 15 pow     % [4747561509943]
    ```

  - `fact` should calculate the factorial function – the product of all integers between 1 and N.

    ```
    5 fact     % [120]
    10 fact     % [3628800]
    42 fact     % [1405006117752879898543142606244511569936384000000000]
    ```

  - `fibs` should generate a list of the first N Fibonacci numbers on the stack. For example:

    ```
    5 fibs     % [1,1,2,3,5]
    10 fibs    % [1,1,2,3,5,8,13,21,34,55]
    ```

```
30 fibs   % [1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,
          %  2584,4181,6765,10946,17711,28657,46368,75025,121393,
          %  196418,317811,514229,832040]
```

– `fillstr` should take the ASCII/Unicode value of a character, and generate a string in which it is repeated N times. For example:

```
46 10 fillstr    % [..........]
36 4 fillstr     % [$$$$]
```

The 46 is the ASCII value of the dot '.' and 36 is the dollar sign '$'.

To submit any of these PicoScript definitions, save them into your `assn3` directory using the operator name and the extension `.ps` – for example, `fillstr.ps` could contain:

```
/fillstr {  % My procedure definition
  BLAH BLAH % Your code goes here
} def
% And then some test cases
46 10 fillstr
36 4 fillstr
```

Then if you paste the above into the interactive interpreter, it should show the resulting stack as:

```
[.........., $$$$]
```