Project 7: Another Calculator

CSCI 245 ⋆ Programming II: Object-Oriented Design ⋆ Fall 2019

Devin J. Pohly <devin.pohly@wheaton.edu>

The goal of this project is to give you an opportunity to use the Strategy and State patterns in a familiar context.

You will be writing one last calculator program. This time we'll be back to a "normal" (infix) calculator. But you will now be incorporating what we have learned about the Strategy and State design patterns in your design of the calculator.

Setup

In this project you will use the same GUI and other starter code as the other calculator projects. Find the GitHub invitation link on Schoology to create your fork, then clone it to any machine you want to work from (don't download a tarball or ZIP!).

You are encouraged to be committing and pushing small changes as you go. If you mess something up, you can always revert to a committed version, and if you push those commits then GitHub has them in case something happens to your copy of the code.

Calculator details

Now you'll be operating under the following rules:

- No using `if`, `switch`, `try`, or degenerate loops. All decisions must be done using polymorphism, **except** for dealing with division by zero or screen formatting—for those you may use either an if statement or exception handling.

  (What's a "degenerate loop?" Something like this:

  ```
  // I want to check if x < 3 but I can't use an if.
  // I'll use a while loop instead. I'm so clever...

  while (x < 3) {
      System.out.println("Now I know that x is less than three!!!");
      break;   // ok, break out of this "loop"
  }
  ```

  Generally, the point of all this is to use polymorphism... *not* to find sneaky ways to build the equivalent of ifs!)

- Follow the same specification for the operation of the calculator as from project 2 (see below).
- Use the state pattern to govern the behavior of the calculator.

Specification details

See the equivalent section from Project 2. All those things count here, too.

Reminder about `setup()`

(Largely repeated from Project 5)

To clarify one part of the framework for this and other calculator projects, please note that when your submission is graded, your `setUpCalculator()` method will be called, and it will be given an object implementing the `CalculatorFace` interface you are given, but it will not be an instance of the `ConcreteCalculatorFace` class. `ConcreteCalculatorFace` is given for your own testing purposes. That is why your code should not be dependent on that class, *only on the interface*. Also, the main method of `SetUp` will not be called, only the `setUpCalculator()` method. In short, the contract you need to fulfill is

> When `setUpCalculator()` is given an object implementing the interface `CalculatorFace`, it will instantiate and attach action listeners to the given object (and instantiate any other appropriate classes, ie the "brain" or whatever you call it) so that the given object will behave like a calculator.

To turn in

Please make sure you have committed and pushed all of your work to GitHub by the project deadline. Late submissions may be penalized or not accepted.

Grading

Grading for this project will follow this approximate point breakdown (the right to make minor adjustments in point distribution is reserved):

- Correctness: 15 (How does your calculator behave when tested against various button-sequence scenarios?)
- Use of state pattern and other design aspects: 5 (Does your code conform to the specific rules of this project, and how well does your code adhere to other principles of good design? How much improvement have you made since Project 2?)
- Documentation/coding style: 2 (How well does your code adhere to the documentation and style principles?)

Note both that this project is worth a little more in total than Project 2 (22 points vs 20 points) and that design and style are worth a larger portion of the total.