Calculator - No Control Flow

Due: Wednesday, March 13 @ 11:59pm (Scala Only)

Updates and Pitfalls

-

Objective

Gain experience with advanced OOP techniques (MVC and State Pattern)

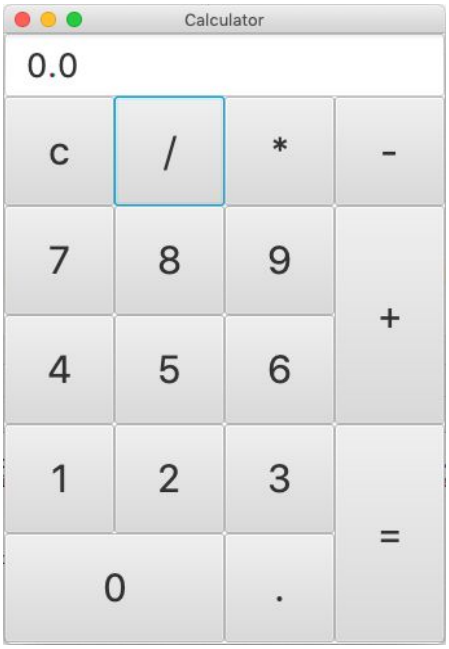
Description

You will write a 4 function calculator without using any control flow, meaning you cannot write any code that will be executed conditionally. Instead you will apply object-orientation programming approaches to make decisions based on types instead of values. Specifically, you are expected to use the state pattern to complete this assignment.

The following are banned in your submissions:

- Conditionals
  - if/else if/else
  - match/case
  - throw/try/catch/finally (Can be used to simulate conditionals)
- Loops
  - for
  - while
  - do/while
- Any way of directly simulating Conditionals or Loops that is against the spirit of this assignment. (Ex. Taking advantage of short circuit evaluations)

If your submission, including testing and comments, contains any of the keywords listed above it will not be graded.



The calculator itself will be a 4-function calculator with the buttons and functionality:

- Digits 0-9
    - Adds the pressed digit to the end of the currently displayed number, or replaces the currently displayed number if an operation has just been pressed (Start RHS) or computed with = (Begin new LHS)
- Decimal
    - Begins the decimal portion of the number being typed. If already in the decimal portion this button has no effect (and must not break other functionality)
- Clear
    - Returns the calculator to its initial state with 0.0 being displayed
- Operations (+, -, *, /)
    - Begins the chosen operation with the currently entered number as the LHS of the operation
    - If previously pressed button was an operation the previous operation is replaced with newly pressed one
        - Ex. If the user presses [3, *, +, 5, =] the displayed text should be 8.0
    - If consecutive operations are used with numbers for each then all operations are computed in the order they are entered (No order of operations)
        - Ex. If the user presses [3, *, 5, +, 1, 0, ., 5, =] the displayed number should be 25.5
    - Nothing happens if = is pressed immediately after an operation without inputting any RHS number
    - [Testing/Grading Note] The displayed number after an operation is pressed is left undefined. This displayed number will not be checked during grading and you should not add it to your tests. To test the operations, simulate an equals press then check the displayed number
- Equals

- Computes the current operation with the currently displayed number as the RHS of the operation and displays the result
- Has no effect if an operation button was not yet pressed
- If pressed again, repeats the previous operation with the same RHS and the currently displayed number as the new LHS of the operation
    - Ex. If a user presses [4, +, 5, =, =] the displayed number should be 14.0
    - Ex. If a user presses [1, 0, -, 6, =, =] the displayed number should be -2.0

Project Structure

1. Create a new project in IntelliJ
2. Pull the Scala Examples repo and copy the calculator package into the src folder in your new project
3. You may add any variables/methods/classes/etc that you'd like as long as the provided Action classes and the calculator's displayNumber method are present. These classes/method define the agreed upon interface between model, view, and controller

Testing Objectives (30 points)

Testing Objective 1

In the calculator.tests package write a test suite named TestEnterNumbers that tests the 10 number buttons on the calculator. This suite should simulate button presses and check that the displayed number is as expected.

Ex. If the user presses [4, 1, 7] then displayNumber() of the calculator should return 417.0

Testing Objective 2

In the calculator.tests package write a test suite named TestSingleOperation that tests the 4 operations of the calculator with only single digit numbers. For this test you will need functionality for the 10 number buttons, the 4 operation buttons, and the equals button. This suite should simulate the button presses and check that the displayed number is as expected.

For this objective you do **not** have to test multiple consecutive presses of operation buttons or multiple operations. Each test case should contain exactly 4 button presses.

Ex. If the user presses [5, *, 4, =] then displayNumber() of the calculator should return 20.0

~~Testing Objective 3~~

There are only 2 testing objective for this assignment. You are expected to be able to write your own tests for the remaining functionality to complete the primary objective. You are encouraged to separate your tests into multiple additional test suites to test each piece of functionality one at a time instead of attempting to challenge the primary objective immediately.

Primary Objective (35/20 points)

The entire calculator functions properly according to all the specifications defined in this document.

Note: Any functionality that is undefined by this document will not be tested. For example the behaviour on the inputs [3, +, 4, *, =, =] is undefined since this document never specifies what = should do after an operation is evaluated without using the = key. In this example the addition operation is evaluated when the * key is pressed, or is stored for later evaluation, making it unclear if the = press should do nothing since an operation was just pressed or reevaluate the addition since an expression was just evaluated. To limit the complexity of such cases these are left undefined and untested.