Calculator
Due Sunday March 1, 2015

Objectives

For this assignment, you will:

- Gain practice using the `Stack` data structure

- Implement an `ActionListener`

- Gain more practice with Java GUIs and events

- Gain more experience with unit testing

Description

Your next program is to implement a postfix calculator application. The calculator should take expressions input in postfix notation and display the results of the computation.

In postfix notation, *the operator comes after the operands*. For example, the expression $3 + 5$ would be written as $3\ 5\ +$. As another example, the expression $52 - (5 + 7) * 4$ would be written as:

$$52\ 5\ 7\ +\ 4\ *\ -$$

It is not necessary for this assignment that you know how to convert from infix to postfix since you can assume that the user of your calculator will input all expressions in postfix notation. You are only responsible for evaluating such expressions and displaying the result.

Let's look at a few examples of evaluating postfix expressions using your calculator. To evaluate the postfix expression $3\ 5\ +$, the user would click on: `3, enter, 5, +`. The `enter` button is used to signal the end of a number being input. Without the `enter`, we would not know whether the user was inputting the number 3 or the number 35. The user can also click on an operator button to signal the end of a number being input. Notice that in our example the user clicked the `enter` button only after the digit 3. After the digit 5, the user clicked the `+` button. When we see the `+` button, we know the user has finished entering a number and we can now add those numbers together.

Here is another example. To calculate 3+5*7 (which is written   3 5 7 * + in postfix notation), click on 3, enter, 5, enter, 7, *, +. Again, note that the only thing clicking enter does is to signal the end of the number being input.

A version of this program running as an applet can be found at
`http://www.cs.pomona.edu/classes/cs062/assignments/CalculatorApplet/CalculatorApplet.html`.

━━━━━ Classes ━━━━━

There are 4 classes for this project: `Calculator`, `DigitButtonListener`, `OpButtonListener`, and `State`. Of these 4 classes, you are expected to complete the `DigitButtonListener` class and the `State` class.

Once you have both of these classes working properly, you must add one or more new buttons to the calculator. Possibilities include a squaring button, factorial, or x^y. You should add the new button so that it looks nice, add an appropriate listener (created from `OpButtonListener` if you like), and make sure that the state knows how to handle that operation.

Calculator class

We have provided you with the code for the main `Calculator` class. This relieves you of the burden of layout of the buttons and display of the calculator. We have also included the code for `MiscButtonListener` as an inner class of `Calculator` (it is nested at the bottom of that class). The class `MiscButtonListener` implements the `ActionListener` interface for the miscellaneous buttons on the calculator: `clear`, `exch`, `pop`, and `enter`. Read through the `Calculator` class until you are confident that you understand how it operates.

OpButtonListener class

The `OpButtonListener` is a listener class for the operation buttons: `+`, `-`, `*`, `/`. When the user clicks on an operation button, the corresponding listener is responsible for informing the state what operation is to be performed. Read through the `OpButtonListener` class until you are confident that you understand how it operates. This class provides a useful example for you when you are implementing the `DigitButtonListener` class.

DigitButtonListener class

This class contains the code to be executed when the user clicks on a digit button. Make sure you understand how the `DigitButtonListener`s are being used in the `Calculator` class. Each digit key has its own specialized `DigitButtonListener` which is responsible for knowing which number key it is listening to. When a button is clicked, the code in the `actionPerformed()` method is executed. This methods should inform the state what digit has been clicked on so that the state can use the digit to build the number being typed in.

StateTest JUnit class

`StateTest` is a JUnit test class for the `State` class. It is partially written for you. You must finish writing unit tests for the remaining methods – i.e. you must fill in the unit tests for the methods that are not yet implemented. As you probably discovered in last week's assignment, it's not uncommon to have multiple unit tests for a single method. In the same way, feel free to add additional unit tests to the `StateTest` class.

State class

A `State` object represents the memory of the calculator. Its purpose is to keep track of the current state of the computation. For instance, it has to keep track of whether the user is entering a number, and what the number is so far. It must also keep track of the display window of the calculator (a `JTextField` that displays the current state of the calculator).

The most important feature of the `State` is its stack, which represents those numbers stored on the calculator. Use `java.util.Stack` rather than writing your own stack.

Let's look at how you would evaluate the example from earlier (`3, enter, 5, +`). First, you must keep track of the number being constructed. When the `enter` button is clicked, you know this number is complete. We would then push the number, 3, onto the stack. The next number is a 5. We would push this number onto the stack as well. Finally, since the user clicks `+` we would pop the two numbers off the stack, add them to get 8, and then push 8 back onto the stack.

Here are some more details:

- If the `enter` button is clicked, the current number being constructed is complete. This new number is stored on top of the stack with any previous numbers sitting below.

- If an operator button is clicked, and there are at least two elements on the stack, then the top two elements are popped off, the operation is performed, and the answer is pushed back on top of the stack. The calculator display always shows the number on top of the stack. Make sure that you perform the operation on the numbers in the correct order!

- If the user makes a mistake, e.g. dividing by 0 or trying to perform an operation when there are fewer than two elements on the stack, then the display should show `Error` and the calculator should be reset. You can detect divide by zero yourself or let the system throw an `ArithmeticException` when a divide by zero occurs. Either way, you must handle the error (i.e., your program should not crash – and no "ugly red stuff") and have the display show `Error`.

- If the `clear` button is clicked, the calculator is cleared and reset.

- If the `pop` button is clicked, and you are currently building a number, then that number should be thrown away. Otherwise, if the `pop` button is clicked, then the top element of the stack should be popped and thrown away. The calculator display should be updated to show the new top element of the stack (or zero if the stack is empty). It is not considered an error if the user pushes the `Pop` button when there is nothing on the stack. Just ignore the key press.

- If the `exch` button is clicked, you should exchange the top two elements of the stack. It is not considered an error if the user clicks the `exch` button and there are fewer then two elements on the stack. Just ignore the key press.

- If the stack is empty, the display should show `0`.

Getting Started

1. All startup code is available in `/common/cs/cs062/assignments/assignment05`. You can copy the starter code using Finder as usual. Or, this week, you can practice using the Terminal to copy the starter code from the command line! Terminal is an application. You can open Terminal from the dock. To copy the starter code, type:

```
cp -r /common/cs/cs062/assignments/assignment05/* ~/Desktop/CS062Workspace/Assignment05/src
```

and press enter. This assumes you have already created your Eclipse project.

The copy command takes two inputs. The first input is where you are copying *from* and the second input is where you are copying *to*. You can specify these locations using either relative or absolute filenames. The * symbol is called a wildcard symbol. It is a placeholder that represents any file. Thus, "assignment05/* " means copy any file inside of the "assignment05" directory.

The "~" symbol is a shortcut for your home directory (e.g. my home directory is "/home/america".) This is convenient so that you don't have to type your home directory every time you specify an absolute pathname. Finally, the "-r" flag stands for **r**ecursive. If you use this flag, the copy command will recursively copy every file in every subdirectory.

2. You might find it easier to write the code first under the assumption that the user must click the `enter` button after punching in each number. However, for full credit, it should also handle the case where the user punches in a number followed immediately by an operation. The result should be equivalent to sticking in an intervening "enter". That is punching in `5, enter, 7, +` should give the same results as `5, enter, 7, enter, +`. (Think about what pressing the enter key actually does. What is the difference between pressing `3 7` and `3 Enter 7`? It makes a difference.)

Grading

You will be graded based on the following criteria:

criterion	points
JUnit tests for each method in `State`	3
Digits and "Enter" handled appropriately	3
Arithmetic operations handled correctly	3
Appropriately handle errors and show message	2
Misc keys handled correctly	3
Digit listener	2
Extra method/calculator functionality	2
Appropriate comments (including JavaDoc)	2
Style and formatting	2

NOTE: Code that does not compile will not be accepted! Make sure that your code compiles before submitting it.

Submitting Your Work

As usual, export your entire folder from Eclipse to your desktop. Make sure the exported folder on your desktop contains both a `bin` folder with your .class files and a `src` folder with your .java files.

Please remember to rename your folder to "Assignment05_LastNameFirstName"! Then drag your renamed folder into the dropbox. Be sure that your code is clear, formatted properly, and commented appropriately using Javadoc. See the "Style Guide" on the handouts webpage for what is expected of you.

Extra Credit

If you are interested in extra credit, you might add a decimal point button so that your calculator can handle floating point numbers. Be sure to check that the user does not input an illegal number (e.g., with two decimal points). Possibilities for new buttons with this include square root, trig functions (e.g., sin, cos, tan, etc.), inverse trig functions, x^y, e^x, factorial, inverses, or logarithms. See the class Math in package java.lang for a list of available mathematical functions in Java.