Project 2: A Calculator

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

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

The goal of this project is to review the basics of object-oriented programming and to give you an initial try at designing an object-oriented system.

Your task is to build a program to emulate a four-function (addition, subtraction, multiplication, and division) calculator such as were common many years ago. The use will be presented a window that looks like a calculator and be able to click on the buttons to get a result. I am providing the GUI, which is analogous to the outside cover and buttons of a real-world calculator. Your task will be to write the code for the internal workings, analogous to the electronics inside a calculator.

This will give you practice using objects to represent information, as well as designing the interaction between objects. This will also give you some practice working with action listeners.

Basic setup

After moving into your folder for this course, unpack the starter code given for this project:

```
$ cd 245
$ tar -xzf ~devinpohly/Public/csci245/project2.tar.gz
```

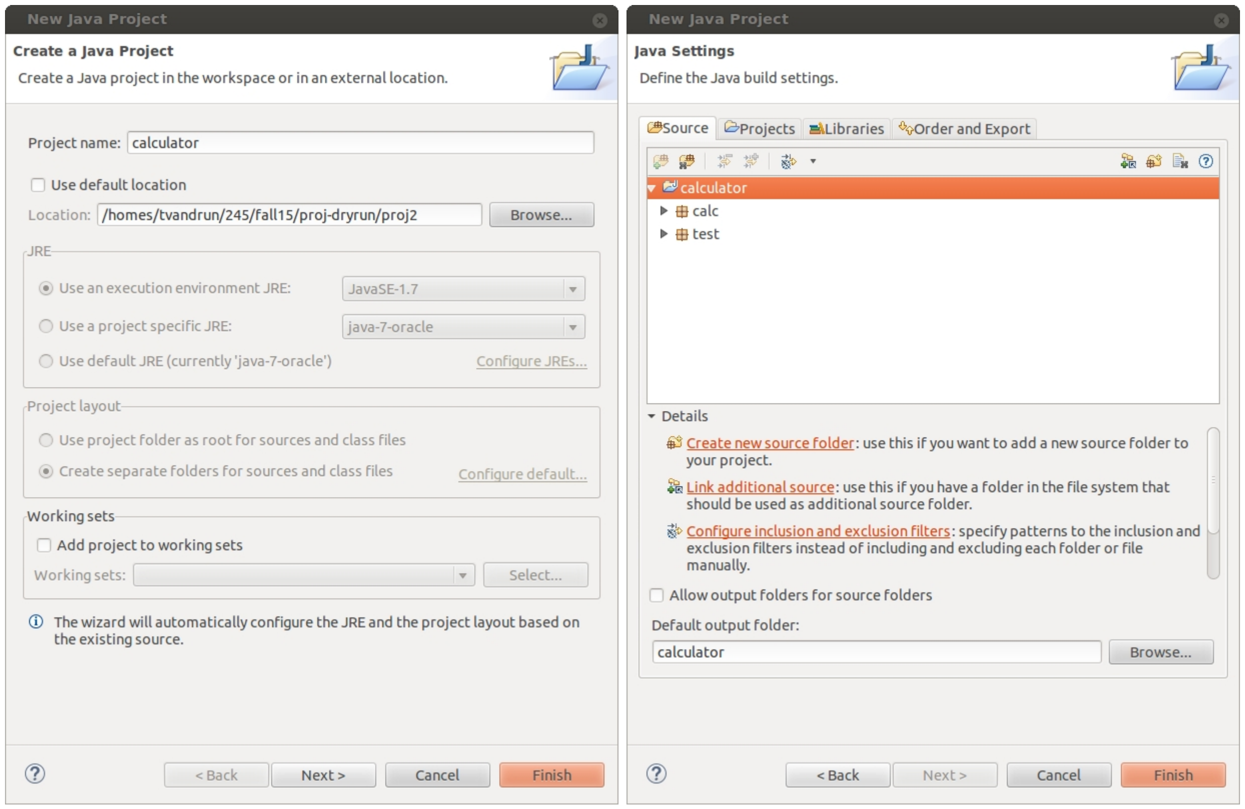
This will give you directory called `project2` with the following files and folders:

- `.classpath`: a hidden file (which shows up only if you do `ls -a`) that Eclipse will use for the project setup; you can ignore it
- `calc`: a package folder containing the following files:
  - `CalculatorFace.java`: an interface defining how other classes will interact with the calculator's face
  - `PlainCalculatorFace.java`: the code for the GUI
  - `NoSuchButtonException.java`
  - `SetUp.java`: a class containing the `main` method
- `test`: a folder containing JUnit tests to help you with debugging

Now open Eclipse and start a new project. When the new project wizard (Figure 1) comes up, give it a reasonable project name, un-check "Use default location," and point the location to the `project2` folder, which contains the package folders `calc` and `test`.

When you hit "Next," make sure you see a screen like the right side of Figure 1. If it looks different (for example, having "src" and "bin" folders), then you've missed a detail somewhere.

You can hit "Finish" now.


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: calculator

☐ Use default location

Location: /homes/tvandrun/245/fall15/proj-dryrun/proj2

JRE

● Use an execution environment JRE: JavaSE-1.7

○ Use a project specific JRE: java-7-oracle

○ Use default JRE (currently 'java-7-oracle')

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

The wizard will automatically configure the JRE and the project layout based on the existing source.

New Java Project

Java Settings

Define the Java build settings.

calculator
- calc
- test

Details

Create new source folder: use this if you want to add a new source folder to your project.

Link additional source: use this if you have a folder in the file system that should be used as additional source folder.

Configure inclusion and exclusion filters: specify patterns to the inclusion and exclusion filters instead of including and excluding each folder or file manually.

☐ Allow output folders for source folders

Default output folder: calculator

Figure 1: Eclipse setup wizard

The calculator face

Open `CalculatorFace.java`. As mentioned above, this is an interface defining how your code will interact with the GUI. It defines how to attach action listeners to the buttons and how to write to the screen. Its documentation tells you everything you need to know about communicating between your code and the window.

`PlainCalculatorFace` is a class implementing `CalculatorFace`, that is, it makes/models the window. **You do not need to modify this class.** In fact, you do not need even to *look* at this class: the interface `CalculatorFace` gives you everything you need to know to interact with this or any other class that implements the interface. When we grade the project submissions we will not use `PlainCalculatorFace`. We will use another class that implements the interface.

If you work on this project on your own computer instead of the lab machines, then there is one change you might make to `PlainCalculatorFace` so that the window will appear correctly on another platform; see the documentation in the file for details.

SetUp, action listeners, and calculator internals

Your task in the project has three parts:

- Write a class (possibly with other helper classes) to represent the internal state of the calculator.
- Write classes that implement the `ActionListener` interface that will be attached to the buttons and will be a bridge between the calculator face and the calculator internals.
- Finish the method `SetUp.setUpCalculator()`, which should initialize the calculator including connecting the action listeners and internals to the calculator face.

If you haven't used the `ActionListener` interface before or don't remember it well, you can read the relevant sections in the textbook (from chapter 17, particularly chapter 17.2) and look at the Java API. Here's a quick review/overview:

> An action listener is an object that is "attached" to a GUI component—for example, a button. Action listener classes implement the interface `ActionListener` in the package `awt.event`, which has the method `actionPerformed(ActionEvent ae)`. For our purposes, you can ignore the parameter (but don't let that stop you from finding out how to use it if you're curious). For example, in this project, you may want to make a `PlusListener` class and attach an instance to the plus button on the calculator face. When the button is pressed by the user, the method `actionPerformed()` is invoked. For our present purposes, the `actionPerformed()` methods will cause some change to the calculator internals which will probably then be reflected on the calculator's screen.

Open `SetUp`. You will notice that the method `setUpCalculator()` takes a `CalculatorFace`—not necessarily a `PlainCalculatorFace`, though! The code you add to this method presumably will:

- instantiate the class or classes representing the calculator internals,
- instantiate the action listener classes, and
- attach the action listeners to appropriate buttons on the given face.

For example, suppose you have written a class `PlusListener` that implements `ActionListener` and handles the pressing of the plus button. You can attach an object to the button with the code:

```
face.addActionListener('+', new PlusListener());
```

The `SetUp` class has a `main` method that instantiates a `PlainCalculatorFace` and calls `setUpCalculator`. You will use this for testing your calculator by hand (as opposed to automated tests, described below). So, to run the program from the command line, use

```
$ java calc.SetUp
```

Or hit the "run" button for `SetUp` in Eclipse.

You may assume that your code will only be used after `SetUp.setUpCalculator()` has been called, but you must not assume that it will be called from `SetUp.main()`. The method `SetUp.setUpCalculator()` is public and may be called from other code.

Specification details

The intent is to make this program work "just like a hand-held four-function calculator." You can find a decent example for reference at `http://www.calculator.com/pantaserv/makecalc`; Kelly may also have one in the CS office that you can borrow briefly. Don't try to implement anything fancy, and don't use the CentOS built-in calculator as a reference.

Here are a few specific scenarios and what should happen in terms of external behavior (what happens *inside* the calculator is up to you):

- As the user enters a first operand, the number entered so far should appear on the screen. If the decimal button is pressed, then subsequent number button presses will add digits to the right of the decimal point.
- If the decimal button is pressed a second time for the same operand, it is ignored.
- When the user presses an operator button, the first operand remains on the screen, but as soon as the user begins to enter the second operand, the first operand *disappears from the screen.* The screen then shows the second operand as it is being entered.
- When the equals button is pressed (after a second operand), then the operation that had been entered between the two operands is executed and the result is displayed.

- If, after the result is displayed, the user begins entering a new number, then the earlier result is forgotten and the new number is treated like a first operand. The screen shows this new operand as it is being entered.
- If, after the result is displayed, the user presses another operation button, then the result is treated like a first operand to that new operation. It remains on the screen until the user begins to enter the second operand, at which point the screen shows the new operand as it is being entered.
- If, after entering a second operand, the user presses an operation button instead of equals (for example, `1 + 2 +` instead of `1 + 2 =`, then the first operation is executed, its result is shown on the screen, and that result is taken as the first operand to the new operation. Thus `1 + 2 +` has the same effect as `1 + 2 = +`
- Pressing the plus-minus button toggles whether the currently displayed number is positive or negative. It has the same effect if pressed before, during, or after the number is being entered. If pressed twice, for example, the number will be positive.
- The `C` (clear) button resets the calculator back to its initial state.

Examples:

Input	Output
5	5
51	51
12.5	12.5
12.5.3	12.53
±15	-15
1±5	-15
15±	-15
±15±	15
1+2	2
1+2=	3
1+2+	3
1+2=8	8
1+2=8-6=	2
1+2=+4	4
1+2=+4=	7
1+2+4	4
1+2+4=	7
1+5C4+9=	13

Whether integer results appear as plain integers (`3`) or with a zero after the decimal point (`3.0`) is up to you.

This doesn't cover all possible scenarios, of course. Part of this assignment is for you to think through what the user might do and how the calculator should behave in those circumstances. Feel free to ask me questions like "What should happen if....?"

Important: The "screen" on the calculator is only 15 characters wide. Java displays `double` numbers using more than 15 characters. Make sure that when you display results that you do not run off the end of the screen. You need to think about how to format your results. There are several ways to do this. One way is to use the `DecimalFormat` class; it's described in *Absolute Java* (check the index, pages differ with edition) or the Java API. You also can devise your own formatting strategy using `String` manipulation.

Also, be sure your code is documented according to the specifications given in the style principles.

Finally, as has been mentioned in class, this is a point where many students realize they never understood what the **`static`** parts of a class were all about—which usually stems from having not yet mastered the concepts of class and instance. So if something is wrong and Eclipse suggests you make some variable static, *don't do it.*

Something else is wrong; ask for help if needed. Specifically, *any static variable must include in its documentation an explanation why that variable is static.*

Tips

This is the time of the semester when I get to know students in this class, because so many of them show up at office hours wanting help on this project. You are warmly invited to stop by for help, but here are things you should think about before that conversation:

There are two basic strategies for using action listeners: either you can write busy `actionPerformed()` methods, which do a lot of the actual work of the calculator themselves; or you can write simple `actionPerformed()` methods which merely call another method in a central object. The choice between these is up to you, though that doesn't mean they're equally good options. As we go on this semester, we'll discuss some of the trade-offs in a decision like this, and at this point you should try to use the principles and examples we've seen so far to make as good a design as possible.

Don't confuse the calculator's screen with the calculator's memory or state. The screen is used to show the user a result or the operand the user is currently entering. It is not supposed to show everything that is going on inside the calculator, and it shouldn't be used to store information. That's the job—or, one of the jobs—of the calculator internals.

Don't confuse the event of an operation button (such as plus) with the actual operation happening. When the user presses the + button, the calculator doesn't even have the second operand yet. The action listener on the plus button isn't supposed to do addition. Instead it should cause an addition to be stored as a pending operation, to be done later.

Don't try to write this entire project at once. Start implementing the number buttons first, making numbers appear on the screen. Then implement the plus button. Once that's working, start doing the other operators.

Testing

Test your calculator "by hand" by running it through `calc.SetUp`. Additionally, I have provided some JUnit tests that will exercise how well your calculator handles some of the scenarios described earlier. Run these tests by running `test.TestCalcBasic` as a JUnit test.

You may inspect these tests and add your own. Here's how they are set up:

```
@Test
public void addEq() {
   testSequence("1+2=", new String[] {"3", "3.0"});
}
```

The first parameter to `testSequence()`, (here, the string `"1+2="`) indicates a sequence of buttons pressed on a newly initialized calculator. The second parameter is an array of strings indicating the set of acceptable results that could be on the screen after that sequence of buttons. In this case, the test will pass if either `3` or `3.0` is on the screen after the given sequence of buttons were pressed.

The test cases given do not represent all the tests your submitted project will be graded against. We will run your submission against a wider range of tests; you should test it more thoroughly too.

Submission and grading

Please turn in all the files you wrote or modified (i.e., the files you didn't modify don't need to be turned in) by attaching them to an email to the instructor (`devin.pohly@wheaton.edu`). Remember to Cc the TA as well (`david.pineda@my.wheaton.edu`).

Getting your calculator to work perfectly right is actually pretty hard to do. It is not expected that students will get full marks on the first try of this project—and that's OK, because you will have a chance to revisit this problem in a later project.

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

- Correctness: 16 (How does your calculator behave when tested against various button-sequence scenarios?)
- Design / efficiency: 3 (How well does your code adhere to principles of good design?)
- Documentation / coding style: 1 (How well does your code adhere to the documentation and style principles?)

Design and style are weighted disproportionately low in the grading of this project, since you are just learning how to design systems. The relative weight of design and style in the grading of projects will increase as the semester goes on.