DUE DATES:

Monday recitations: 9:00 PM on 10/30
Wednesday recitations: 9:00 PM on 11/1
Thursday recitations: 9:00 PM on 11/2
Friday recitations: 9:00 PM on 11/3
Saturday recitations: 9:00 PM on 11/4

Ready!

This lab you we have again provided you with a simple graphical user interface for a calculator. This time the calculator is incomplete and buggy. You will need to fix bugs and add missing functionality. In particular, you will add to it the ability to handle binary operators, like addition, subtraction, multiplication and (integer) division.

The lab is structured into four parts, each in its own package: part0, part1, part2, and part3. Inside each package there is additional code. The exact code differs a between parts, but each part has at least these files:

> **code.Driver.java**: Use this to run your code. This class contains a main method that instantiates the user_interface.CalculatorUI class. Do not modify this class.
>
> **code.Calculator.java**: The basic calculator code. You will need to add code to this class.
>
> **user_interface.CalculatorUI.java**: Creates a graphical user interface for the calculator. Do not modify this class.

Your TA will show you how to run automated tests to get feedback on how closely your solution matches our expectations. Remember that you can submit to Web-CAT as many times as you want prior to the deadline. We encourage you to keep working on the lab until you score 100 on the automated tests.

Set!

1. Log in
2. Start Eclipse
3. Switch to the CVS Repository Exploring perspective
4. Check out the CSE115-Lab7 project from the Labs repository
5. Switch to the Java perspective

Part 0

For this part you need only modify the file part0.code.Calculator.java

Read the comments embedded in the code, and run the tests associated with part 0. Your TA will show you how to do this, and how to interpret the results. Make changes to the part0.code.Calculator class so that all the tests pass.

Part 1

Before you start part 1 make sure you make the corrections changes from part 0 in the corresponding classes in part 1.

In this part you will focus on making just one of the operation buttons work correctly. We'll start with the addition button. For this part you need modify both the file part1.code.Calculator.java and the part1.operations.Add.java files. Read the comments embedded in the code, and run the tests associated with part 1. Make changes so that all the tests pass.

Any time a button in pressed in the graphical user interface a method is called on the Calculator object. For part 1 you have to define two methods to handle clicks on the '+' and '=' buttons respectively. They have been partially defined as follows:

```
public void equalKeyPressed() {
      _ui.updateDisplay();
}

public void addKeyPressed() {
      _ui.updateDisplay();
}
```

There are many ways we could approach doing this problem; we are going to represent each operation by an object. In part1, because we are dealing with just one operation (addition) we can simply define a class an instance of which represents the add operation.

To define the Add operation we have given you a partially defined class in part1.operations.Add.java. In this class you need to complete the definition of the following method:

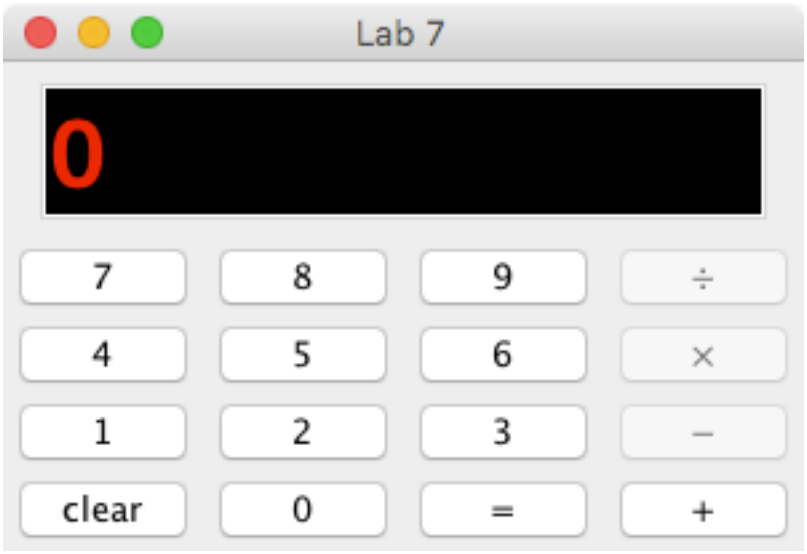
```
public int perform(int left, int right)
```

Read the comment block just above this method to see how it should be defined.

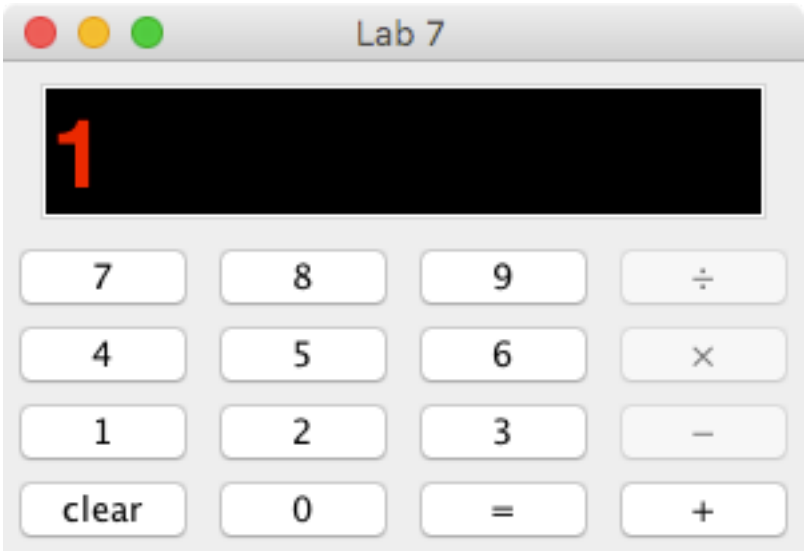
When the "+" key is pressed and the addKeyPressed() method is called, create a new Add object, ***and remember it***. To create it simply instantiate the class. To remember it, store a reference to the Add object in an instance variable of the Calculator class.

When the "=" key is pressed the equalKeyPressed() method is called. Call the perform method on the Add operation object, passing the remembered and current values of the calculator as the left and right arguments, respectively. The perform method returns the result: set the value of the calculator to the returned result. Afterwards make the calculator "forget" its remembered value (set the corresponding variable to zero) and the remembered operation (set the corresponding variable to null).

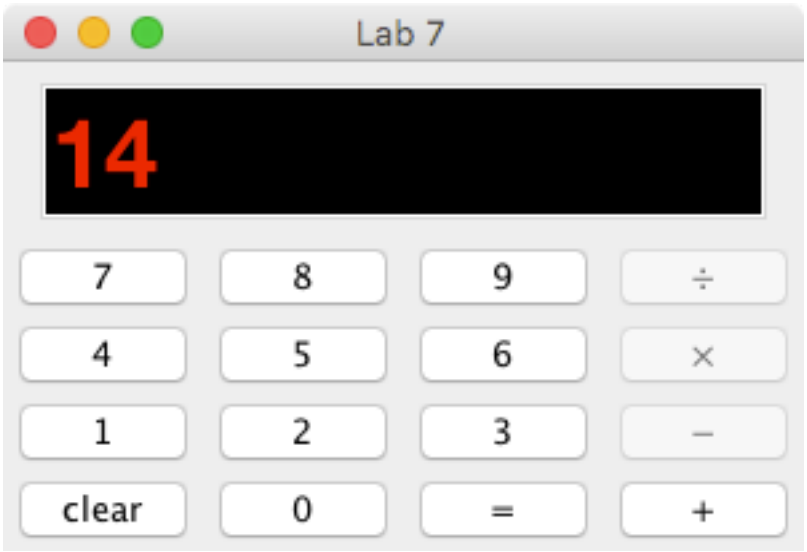
For example, the part1 calculator should behave as follows. When the calculator first is started its value is zero, which is what is displayed:



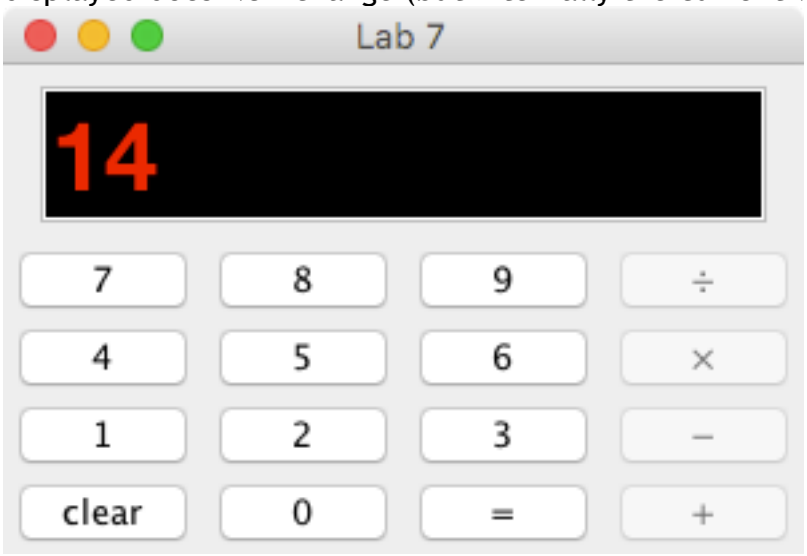
As digit keys are pressed a value is built up in the calculator as expected. For example, if we press '1' the value becomes 1:



If the '4' key is pressed next the value becomes 14:



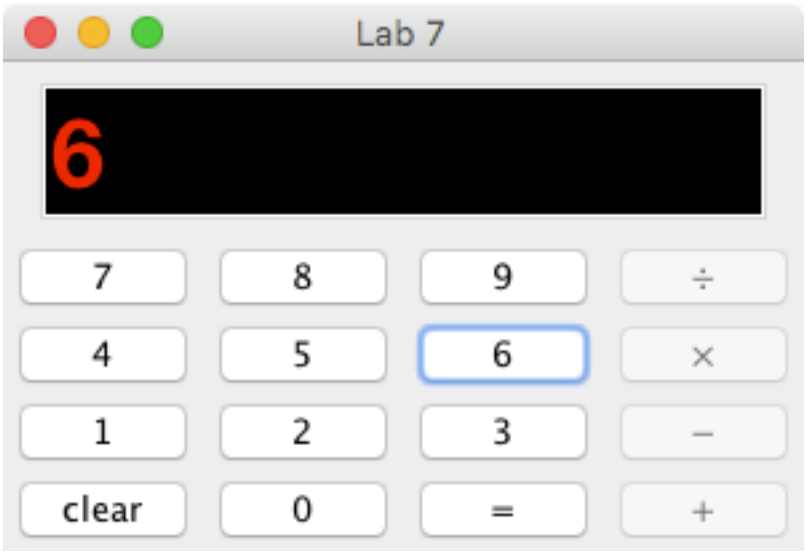
Now suppose that the '+' key is pressed. At this point **a new Add object must be created** and the current value of the calculator (14) becomes the remembered value. Notice that the value displayed does NOT change (but internally the current value must be reset to zero):



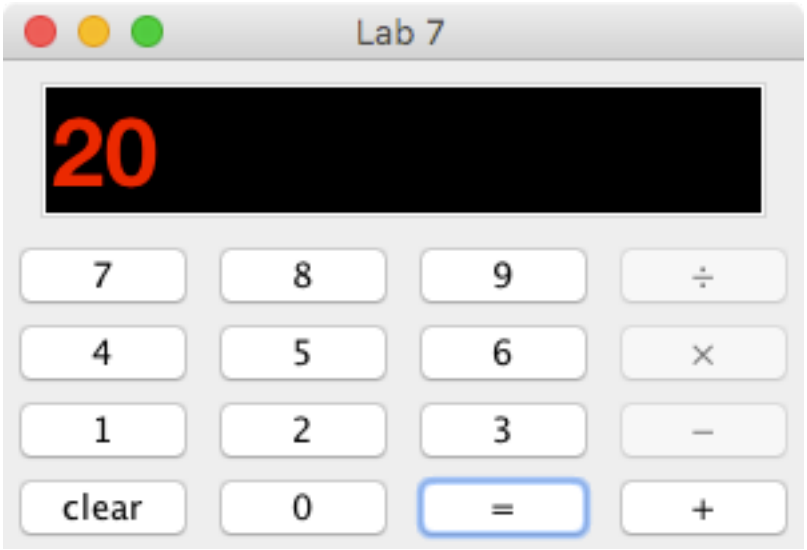
If you look closely you will notice that the '+' key is now disabled. This is intentional, and is handled by the user interface. Doing this helps to simplify the operation of the calculator. The operation key will stay disabled until one of two things happen:

- a second argument is entered and the '=' key is pressed to actually perform the computation
- the 'clear' key is pressed (which resets the value of the calculator to zero and clears any pending operations)

Assuming that '6' is pressed next:



the value of the calculator becomes 6, and the display changes to reflect that. Notice that the display is NOT 146). If the '=' key is now pressed the addition operation is performed and the result, 20 (which is of course the value of the expression 14+6) is displayed:



The '+' button is re-enabled by the user interface.

Part 2

Before you start part 2 make sure you make the corrections changes from parts 0 and 1 in the corresponding classes in part 2.

For this part you will extend the approach you took in part 1 to cover additional arithmetic operations, such as subtraction, multiplication, and division.

We need to define a class for each kind of operation we wish to perform. In part 1 you defined the Add operation as a class with the following method:

```
public int perform(int left, int right)
```

You defined the perform method so that it returned the sum of left and right.

Now you want to do this for the remaining three operations: Subtract, Multiply and Divide. The main conceptual jump here is how we can treat objects instantiated from different classes as the same at some level of abstraction – we don't want to have treat these four operations as completely unrelated in our code.

The solution, as we've seen at in lecture, is to introduce a new type! In this case it'll be an interface (BinaryOperation) which can serve as a common supertype to all four concrete operations, Add, Subtract, Multiply, Divide. In other words, each of the four operation classes must implement the BinaryOperation interface.

Once we've done this we can remember the currently selected operation in a variable of type operations.BinaryOperation.

Ask your TA to help you draw the UML class diagram showing what's going on.

Part 3

Before you start part 3 make sure you make the corrections changes from parts 0, 1 and 2 in the corresponding classes in part 3.

If you experiment with the completed part1 or part2 calculator you will see that the when there is no remembered operation the calculator can product a runtime error. This is because when forgetting an operation we set the corresponding variable to null. We can demonstrate this: assuming you have finished the part2 calculator, run it and click '6', '+', '4', '=' and then '=' again. You should see something like this in your console view:

```
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
        at part3.code.Calculator.equalKeyPressed(Calculator.java:43)
        at part3.user_interface.EqualKeyListener.actionPerformed(EqualKeyListener.java:21)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:2022)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2346)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:402)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:259)
        at javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:252)
        at java.awt.AWTEventMulticaster.mouseReleased(AWTEventMulticaster.java:289)
        at java.awt.Component.processMouseEvent(Component.java:6527)
        at javax.swing.JComponent.processMouseEvent(JComponent.java:3321)
        at java.awt.Component.processEvent(Component.java:6292)
        at java.awt.Container.processEvent(Container.java:2234)
        at java.awt.Component.dispatchEventImpl(Component.java:4883)
        at java.awt.Container.dispatchEventImpl(Container.java:2292)
        at java.awt.Component.dispatchEvent(Component.java:4705)
        at java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:4898)
        at java.awt.LightweightDispatcher.processMouseEvent(Container.java:4533)
        at java.awt.LightweightDispatcher.dispatchEvent(Container.java:4462)
        at java.awt.Container.dispatchEventImpl(Container.java:2278)
        at java.awt.Window.dispatchEventImpl(Window.java:2739)
        at java.awt.Component.dispatchEvent(Component.java:4705)
        at java.awt.EventQueue.dispatchEventImpl(EventQueue.java:746)
        at java.awt.EventQueue.access$400(EventQueue.java:97)
        at java.awt.EventQueue$3.run(EventQueue.java:697)
        at java.awt.EventQueue$3.run(EventQueue.java:691)
```

```
at java.security.AccessController.doPrivileged(Native Method)
at java.security.ProtectionDomain$1.doIntersectionPrivilege(ProtectionDomain.java:75)
at java.security.ProtectionDomain$1.doIntersectionPrivilege(ProtectionDomain.java:86)
at java.awt.EventQueue$4.run(EventQueue.java:719)
at java.awt.EventQueue$4.run(EventQueue.java:717)
at java.security.AccessController.doPrivileged(Native Method)
at java.security.ProtectionDomain$1.doIntersectionPrivilege(ProtectionDomain.java:75)
at java.awt.EventQueue.dispatchEvent(EventQueue.java:716)
at java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:201)
at java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:116)
at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:105)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:93)
at java.awt.EventDispatchThread.run(EventDispatchThread.java:82)
```

Whenever '=' is clicked we try to perform the current operation, but if no operation has been selected the code tries to call the perform method on 'null' – resulting in a java.lang.NullPointerException.

To get the behavior we are looking for we will introduce a new type of operation: a "Null" operation[1]. Define a NullOp class as another variant of BinaryOperation. Our NullOp's perform method simply returns the value of its 'right' parameter (it's a "null operation" in the sense that it does nothing to the current value of the calculator – it returns it unchanged).

Submitting your project to Web-CAT

Make sure you submit your work on time; due dates are listed at the beginning of this lab description. This lab will be automatically graded by Web-CAT. You may submit as many times as you wish. Your last submission is the one that counts (so consider carefully whether you want to make any late submissions, as the late penalty is 20 points per day or portion thereof late).

Pay attention to the output produced by Web-CAT. If your submission scores 100 (without any early submission bonus you might be entitled to) you're all set. If your submission does NOT score 100, fix the problem and resubmit. **Prior to the submissions deadline we expect that you will continue working until your submission scores 100.**

---

[1] If you are interested, you can read more about the Null Object Pattern at many sites on-line, including http://www.oodesign.com/null-object-pattern.html