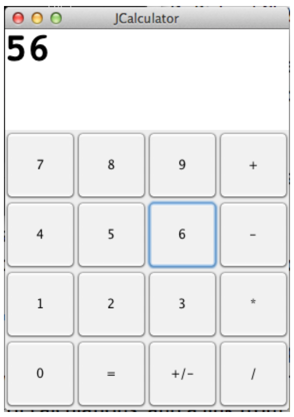
COM6516: Object oriented programming and software design: Practical session 8

The aim of this exercise is to give you more experience with GUIs and event handling. Read through each section before you start programming. You should also look at the example solutions for previous lab sessions and ensure you understand them.

Task 1 – working with GUIs



Your aim is to write a program that simulates a simple calculator. As with last week, this will involve writing a `JFrame`, some `JPanels`, and links from buttons to actions. This week we will use also an inner class to implement the `ActionListener`. To get started for this programming task, we need to set up three components of the interface; a `JFrame` to display on screen, a `JPanel` to contain the buttons, and a `JTextArea` to display the numbers entered by the user and the results of calculations.

The starting point for the `JFrame` is similar to what we have written before, and a template `JCalculator.java` can be found from the lab material. Note that the lab material unpacks into source code and one directory with a compiled solution (but no source code).

The first thing to do is to add a text panel at the top of the frame:

```
JTextArea display = new JTextArea(1, 20);
contentPane.add(display, BorderLayout.NORTH);
```

Use the documentation for `JTextArea` to see how to make it display text in Courier 40 point font; how to set the text area to have a size of around 300x100 pixels; and how to ensure that the user cannot type over the contents. Experiment with `setText` to display string and integer values.
(Hint) `Integer.toString`

Task 2 – buttons and visualisation

The next stage is to set up a class that will display the buttons; you can use the `CalculatorButtons.java` template. This extends `JPanel` and declares some constants that define the button labels. The constructor uses a factory method to generate each button, and add each button to the panel. The constructor takes the `JTextArea` as a parameter so that the action listeners that it adds can update the display. We need to add some code to `JCalculator` to do the following:

1. Create a new CalculatorButtons object:
   ```
   CalculatorButtons buttons = new CalculatorButtons(display);
   ```
2. Set the layout of this object so that the buttons are arranged in a 4x4 grid. You can use the `setLayout` method inherited (indirectly) from `Container` and a `GridLayout(4, 4)`. You may refer to the Java documentation and the Oracle tutorial on using layout managers as needed —
   http://docs.oracle.com/javase/8/docs/api/
   http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html
3. Add this object to the centre of the `JCalculator`.

Task 3 – handling buttons using `ActionListener`

Your code should now display the calculator on the screen. The next stage is to link the buttons to a class that implements the `ActionListener` interface. We will use an inner class for this: take the `ButtonAction.java` template and insert it in `CalculatorButtons.java` at the point indicated. (This template does not compile as supplied, because Java does not allow a private class to be defined in its own file.) Note that `ButtonAction` is private and therefore inaccessible outside `CalculatorButtons`. We now need to add two lines of code to the `makeButton` method: one to create a new `ButtonAction` for each button, and one to add the `ButtonAction` object to the button as a listener using `addActionListener`.

`ButtonAction` implements `ActionListener` so it has an `actionPerformed` method. This has minimal functionality, but if you compile and run your code you should see that when a button is clicked, its label is displayed in the text area and on the console. Every `ButtonAction` object has a label (the identity of the button that has been pressed), and a link to the text area, so we can use the inherited `JTextComponent.setText` to update the screen.

The next stage is to add additional code to the `actionPerformed` method so that the operation of the calculator is simulated correctly. A compiled version of a sample approach is provided (`JCalculatorSolution.class`) to demonstrate one way of solving the problem, but it is up to you to write your own implementation. Run the sample with the `java JCalculatorSolution` command.

The sample approach uses three additional instance fields (which are private, but accessible to `ButtonAction`) in `CalculatorButton`. Field `displayedValue` stores the value currently displayed on the screen. If this value were not recorded, we would have to convert the string on the screen to an integer for calculations. Field `operand1` stores the first operand for each calculation (`displayedValue` being the second operand). Finally, `operation` stores an enumerated type

```
    enum OP { PLUS, MINUS, MULT, DIV }
```

representing the arithmetic operation. If it is `null`, no operation has yet been selected since the previous calculation was completed.

The `actionPerformed` method proceeds as follows:

1. The `opButtonLabels` string is used to determine if a number button has been pressed.
2. If a number button has been pressed, then its label is appended to the `displayedValue` and the text area is updated.
3. Otherwise, if the +/- button has been pressed, then `displayedValue` is multiplied by -1 and redisplayed. (Hint) `setText` and `Integer.toString`

4. Otherwise, if the +, -, *, or / buttons are pressed, then the first operand is retrieved from `displayedValue` and stored in `operand1`, the display is set to the operation name, and the type of the operation is stored in the operation field. If +, -, *, or / is clicked when the `operation` field is not `null`, it is ignored.
5. When = is pressed, two cases have to be considered. If the operation is `null` (no operation selected after the last calculation), the display is unchanged. If the operation is not `null`, the second operand value is taken from `displayedValue` and the correct type of operation is identified; the result of computing the operation on `operand1` and the second operand value, is displayed and stored in `displayedValue`.

Further tasks

If you experiment with the sample solution you will see that it is quite easy to break. Make your solution more robust, so it handles divide by zero correctly and avoids integer overflow in both entry of values and computations. Adapt your solution so that it handles floating point numbers by replacing the +/- key with a ` . ‘ key to act as a decimal point.