Lab 5 – A Calculator with Swing

Goal:
The goal of this lab is to understand how to design and implement a graphical application – a calculator – using WindowBuilder.

Preliminary – Install WindowBuilder
Install the WindowBuilder software from this site. Click on the Release Version link for the Update Site for Eclipse version 4.6 (Neon).

You will find yourself at a page that has detailed instructions on what to do next. Follow these instructions. When you add the URL to the “Work with:” field and hit enter, three items will appear. Select all three. Follow the dialog prompts. Eclipse will need to restart.

Part 1 – Getting started using the designer.
The first thing you’re going to do is create a blank panel. Start by creating a project and a package, just like you normally would. However, instead of making an empty Java class to start with, you’re going to make an auto-generated class. Right click on your package and go to New | Other. Then select WindowBuilder | Swing Designer | JFrame and click next. Give it the name "CalculatorFrame" and click finish after you make sure the checkbox that says "Use advanced template for generate JFrame" is checked off.

Once you’ve clicked finish you should see your generated class. It should have an auto-generated main method and an auto-generated constructor. The main method code boils down to constructing a new frame and then showing it on the screen. The constructor is where the properties of the frame and the frame’s panel are set. Keep in mind the values that the properties are set to as you move forward. On the bottom of the editor you’ll see a tab labeled "Design." Click on the tab to open the design view. You can use this view to graphically change your application and the corresponding code will automatically be generated. Try dragging the bottom right corner of the window until the size is about 250 x 350. Then switch back to the source view and check out the constructor again. Note that the width and height arguments of the setBounds method have changed to 250 and 350 respectively. Try changing the width from 250 to 350 from the source view and then switching to the design view. You’ll notice that you can use both the graphical and raw code to alter your application. Change the width back to 250 when you’re done seeing the correspondence.

Part 2 – Adding components.
Switch to the design view if you’re not already in it. Then click on the JTextField component from the palette of components. If you hover your mouse over your frame you’ll see that there are several areas that you can place the text field. This is because of the panel’s layout manager. The default one should be a BorderLayout, but this isn’t what you want. Instead, you want a GridBagLayout. If you switch to the Source view and look at the constructor you’ll see where it sets the BorderLayout. Go back to the Design view. Under “Layouts”, click GridBagLayout, then move the cursor to the frame and click. Try adding a text field again. Note how the suggested areas are now laid out like a grid. Place the text field in the top left grid cell. You will see instead that it is placed in the center area. This is because the layout manager
gets the final say on where everything goes. Also, if you look at the structure window to the left you’ll notice your new component called "textField." This isn’t a very descriptive name for a variable, so you should change it to something better. This will be the first of two input fields for your calculator, so change the name to something that reflects that. You can change the name using either the raw source (tedious and error prone) or by editing the value of the Variable property in the Properties view.

Now you’re going to add the rest of your components, starting with the other text field. Click on the text field component from the palette again. This time, move your mouse to the right of the existing text field until you see a green box. Place the text field there. The layout should force the two text fields to occupy the same line. Don’t forget to give the text field a descriptive name.

Next you’re going to add four buttons – one for add, one for subtract, one for multiply, and one for divide. Put the first button in the cell under the left text field and give it the text "Add". A good name for the button would be "btnAdd". Create the next button and put it to the in the cell to the right of the add button. Give it the text "Subtract" and give it the name "btnSubtract". You might immediately notice that the two buttons are a different size. To fix this you can right click on each of the buttons and set their horizontal alignment to "Fill." This forces the buttons to fill out the entire cell. Create the buttons for multiply and divide underneath the buttons for add and subtract respectively. Set their horizontal alignments to fill as well.

The last component you need to add is a JLabel. Place the label under the multiply button and give it an initial text value of "0". This is the label that displays the calculator’s answer, so give it a name reflecting that task. Then drag the right green box so that the label’s right boundary matches the subtract and divide buttons’ right boundaries. Now the label is centered without inserting a new cell between the buttons.

Your calculator’s interface is finished. Before you move on, try resizing the application while it’s running. Note how the components resize as well. This is because of the layout manager. The way the components move and resize is different depending on the layout manager you used. Had you used an absolute layout, for example, the components wouldn’t move or resize at all.

**Part 3 – Adding functionality**

Your application’s interface is all set up, but it doesn’t do anything. You can edit the text fields, but the buttons don’t do anything. You need to add event handlers to each of the four buttons for when they’re clicked. Double click on the add button. You’ll see some auto-generated code, including the declaration of a method called "actionPerformed." This is the method that is run when the user clicks on the button – i.e., it handles the event. (This is an example of what an anonymous inner class looks like.)

Your event handler needs to get the text from the text fields and convert the strings into numbers. Then, it needs to add the numbers together and set the label’s text to the sum. Start by getting the input by calling the getText methods of each of the input text fields. Then, parse the strings for floats by using the Float.parseFloat method. However, the input string might not always contain a number, so you should put the parse lines in a try-catch block.
As a reminder, here is the syntax for a try-catch block:

```java
try {
    // Do something that might throw an exception.
}
catch (SomeKindOfException e) {
    // Do something to handle the exception.
}
```

The exception that you’re looking to handle is a NumberFormatException. In the event that this exception is thrown, just set the output label’s text to an error message and return from the method. The last thing you need to do is add the numbers together and set the output label to the answer. You can use the Float.toString method to convert the sum to a string, and you can use the label’s setText method to set the text. Once you’ve done this try running your application. Type in some numbers and click add. Then type in some non-numbers and try the same thing to see your error message.

Try implementing the other three operations yourself. Make sure to handle errors for each of the operations. Lucky for you, Java’s floats already handle divide-by-zero and other arithmetic exceptions by using Infinity, -Infinity, and NaN (not a number). But keep in mind that this isn’t true for every language (usually it’s not true) and you should get in the habit of checking for these errors yourself. Once you’ve implemented each of the operations you can submit the lab.

**Submission**

When you’ve finished the lab you can export the project to a zip file and submit the zip file on InstructAssist. The name of the assignment in InstructAssist is **Lab 5**. To export a project, go to File | Export… and choose Archive File from the General folder. Check off the entire project in the top left window and make sure the format is .zip (not .tar). Finally, give the archive file a name and click finish.

**Acknowledgement**

Thank you to Tom Whitner, who developed the original version of this lab for CS 220X.