Lab 4 - The Calculator

Reminder: You must first have your completed lab checked off by your TA (either in-person or by e-mail). Afterwards, you must submit your lab via Brightspace. Failure to do either will result in a grade of 0.

Goals

In this Lab you will:

1. Learn how to use `NetBeans` to design a simple `GUI` without having to write the code for it yourself.
2. Implement a simple 4-function calculator Desktop app.

Resources

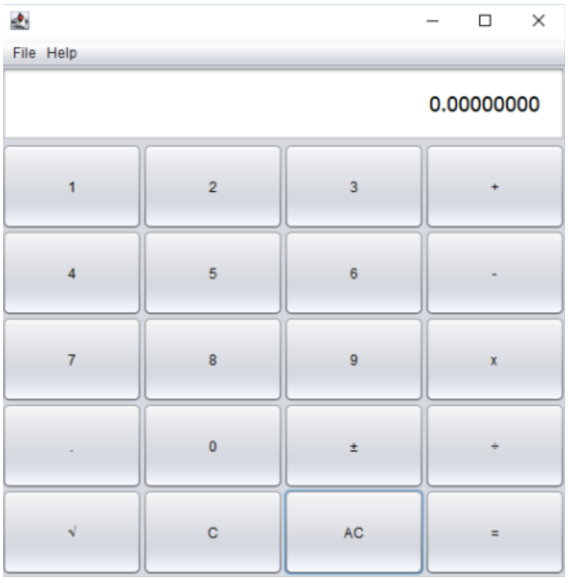
You will want to refer to each and all of the following to complete this Lab:

1. **Lab Overview** (this document).
2. **Calculator.java**, the Starter Code for this Lab (attached).

Background

So far, all of the programs we've made in lab have used archaic, text-only interfaces for interacting with the user. That changes in this Lab; when you're done, you'll have a program you can run that will pop up a window with buttons on it and interact with mouse clicks (or, if run on a tablet, finger taps).

Specifically, we will be implementing a Single-Step Calculator, similar to the built-in utility on Windows and MacOS. It will look like this when you're done:



This section will explore how such a calculator works in the abstract; then in **What To Do, And How** we'll get into actually building it.

Single-Step Calculators

There are a variety of types of calculators; `https://en.wikipedia.org/wiki/Calculator` has a decent exploration of them. A **Single-Step Calculator**, or "Immediate Calculator," is one of the simplest. Unlike a "Scientific Calculator," it does not obey standard mathematical "Order of Operations," but immediately applies the mathematical operations input as they come in:

$$2 + 3 * 5 = 25 \tag{1}$$

Specifically, in this example, the following would occur:

1. The program has just started, or the user has pressed the `AC` button. **0** is displayed.
2. **2** is pressed and becomes displayed.
3. + is pressed; 2 remains displayed.
4. **3** is pressed; 3 is displayed.
5. **x** is pressed; 5 is displayed ($2 + 3 = 5$).
6. **5** is pressed; 5 is displayed (now because it's the latest thing input).
7. = is pressed; 25 is displayed ($5 * 5 = 25$).

A **4-Function** calculator is one that focuses on the four basic arithmetic operations: `+`, `-`, `x`, and ÷. Typically, they also support ± and $\sqrt{}$. They need not be single-step - but ours will be.

Even among single-step 4-function calculators, despite being relatively simple devices, there are subtle differences from one model to another, and coding one can actually be quite complicated. In this lab, we provide you all of the logical code necessary for building the calculator. Your job will be to construct the GUI in NetBeans.

Our Calculator's State Variables:

Note that reading this section is only necessary if you are interested in how the calculator works. Otherwise, you can skip ahead to ***What To Do, And How****.*

Any combination of button presses on our calculator will affect just these 5 variables:

1. `result`, which starts as `0`.
2. `operation`, which starts as '`=`'.
3. `error`, which starts as `false`.
4. `displayLen`, which starts as `0`.
5. `disp`, which starts as `""`.

Our Calculator's Rules:

Note that reading this section is only necessary if you are interested in how the calculator works. Otherwise, you can skip ahead to ***What To Do, And How****.*

Whenever `displayLen` is `0`, the calculator should display the value of `result`, to the most significant 10 digits (or 9, if "`.`" is one of the characters displayed), truncated (so you can use `String.substring()` instead of having to worry about rounding properly).

Whenever `displayLen` is not `0`, the display should show the value of `disp`.

Note that, because there may be a leading negative sign, up to 11 actual characters may be required to display either `result` or `disp`, and that `displayLen` is not always exactly the same thing as `disp.length()`.

There are 11 buttons for entering a number: "`0`" through "`9`" and "`.`". Generally, clicking any of these should increment `displayLen` and append the appropriate `char` to `disp`, unless `displayLen >= 10`, in which case the button click should just be ignored. However:

- "`.`" can not be used if it is already present in `disp` or `displayLen >= 9`; subsequent clicks should just be ignored.
- if and only if `disp` is "`0`" and the `char` for the button press is not "`.`", the "`0`" should be replaced by the `char`.

"`C`" (for "Clear") sets `displayLen` to `0` and `disp` to `""`. Note that it has no effect if `displayLen` was already `0` and `disp` was already `""`.

"`AC`" (for "All Clear") sets all 5 variables to their default state (see above).

For the rest, let `Q` be whatever number is displayed - `disp` if `displayLen > 0`, `result` otherwise - treated as a number (not as a string). Generally you should wait to calculate `Q` until you need it for one of the following button presses, as the user may not be done entering digits.

The two buttons "$\sqrt{}$" and "$\pm$" are immediately applied to `Q`, as square root or negation. Call the result `T`. We then update `disp` to be the `String` form of `T`, to maximum precision (i.e. 9 or 10 digits, so the user can't type anything more), and also update `displayLen` accordingly.

The buttons "`=`", "`+`", "`-`", "`x`", "$\div$", all behave the same way. They trigger an **evaluation** under the operation implied by `operation` operating on the variables `result` and `Q`: note that `operation` is probably NOT the operation represented by the button itself (e.g. if `operation` is "`-`" and the "`+`" button is pressed, the evaluation would complete the lingering subtraction, not do the addition). To be explicit, the various evaluation possibilities given `operation` are:

1. `operation == '='` causes **`result = Q`**.
2. `operation == '+'` causes **`result = result + Q`**.
3. `operation == '-'` causes **`result = result - Q`**.
4. `operation == 'x'` causes **`result = result * Q`**.

5. `operation == '÷'` causes **`result = result / Q`**.

During evaluation, after `result` is updated, `displayLen` is reset to `0`, and `disp` is reset to `""`.

Finally, after evaluation, the button press updates the value of `operation` to be the new operation for that button.

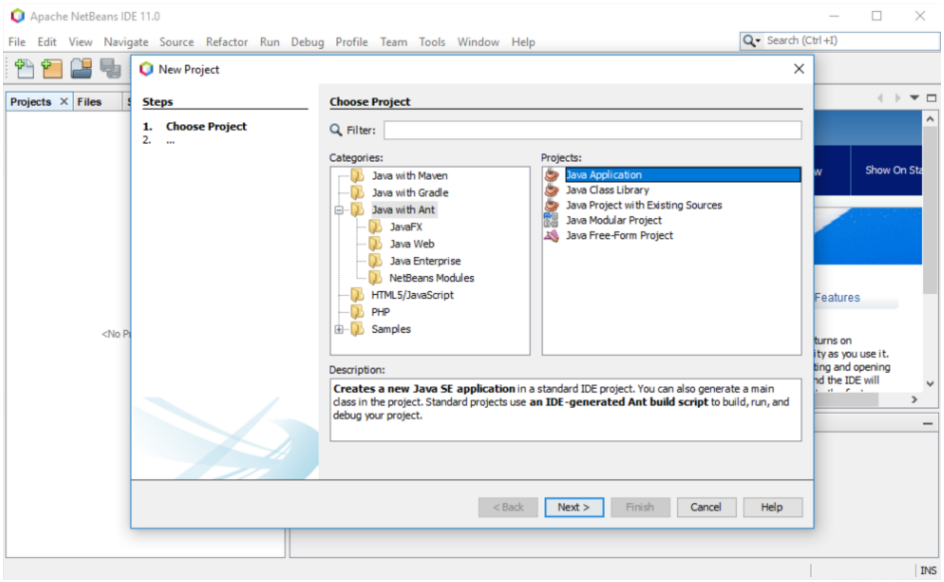
For "`+`", "`-`", "`x`", "`÷`", this has the effect of clearing any lingering number or operation, so that `result` is up to date, and setting up for their own operation in the next step. However "`=`" is a curious case. The "`=`" operation is just to overwrite `result` to be whatever comes next. But most people think of the "`=`" button as finishing the current calculation immediately. In this way, we actually get both; the answer will be displayed, immediately; but if someone then starts typing a new number, it will trigger the "`=`" operation and begin a new line of arithmetic, whenever the next operation button is clicked.

If at any point the user tries to take $\sqrt{}$ of a negative number, or divide by 0, or $|result|$ would be larger than `9999999999`, `error` should be set to true. Whenever `error` is true, all buttons but `AC` are ignored, and the string `"Err"` is displayed instead of `result` or `disp`. This final rule supersedes all previous rules.

What To Do, And How

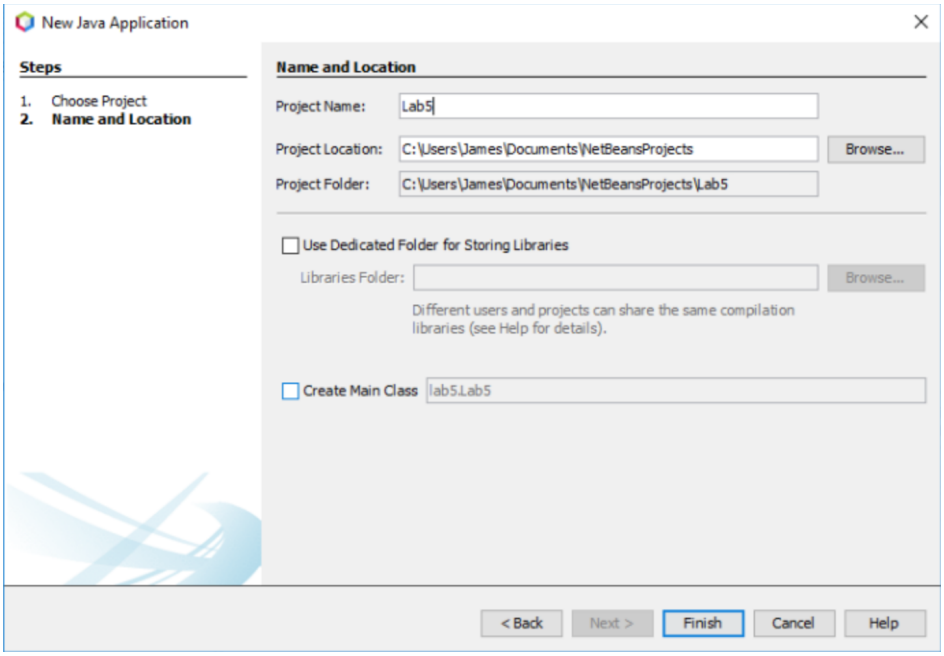
Making the Calculator `GUI` using `NetBeans`

We're going to be using `NetBeans` a little bit differently than we have been, so be sure to pay close attention to the next bunch of screenshots. To start with, we're going to start a new Project...



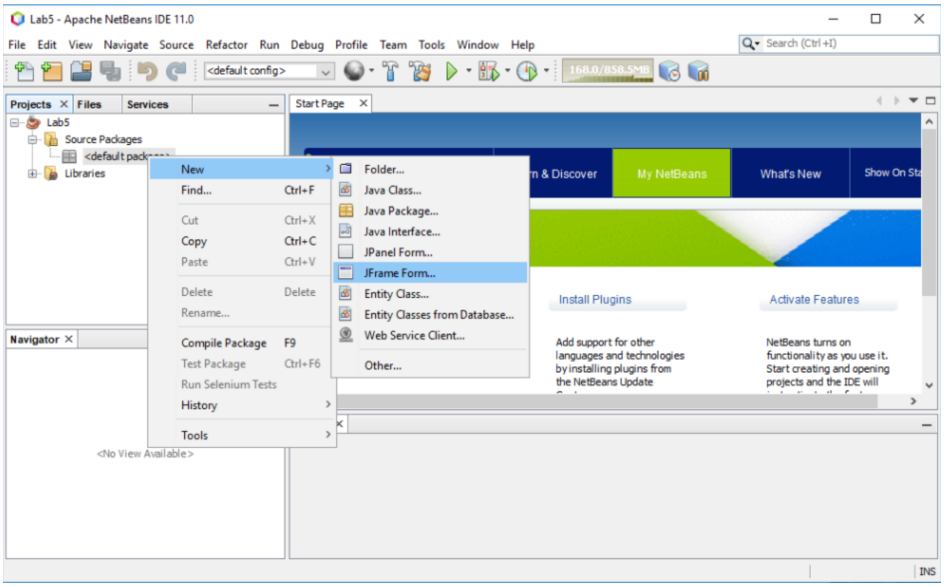
...but then immediately deviate from our usual pattern. In particular, do NOT create the `Main-Class`

like we usually do:

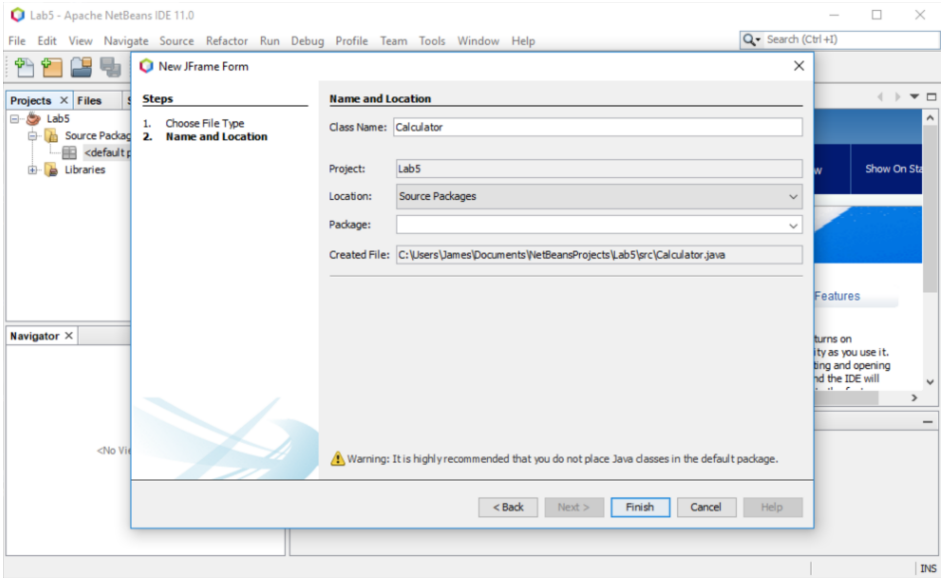


"Create Main Class" is unchecked.

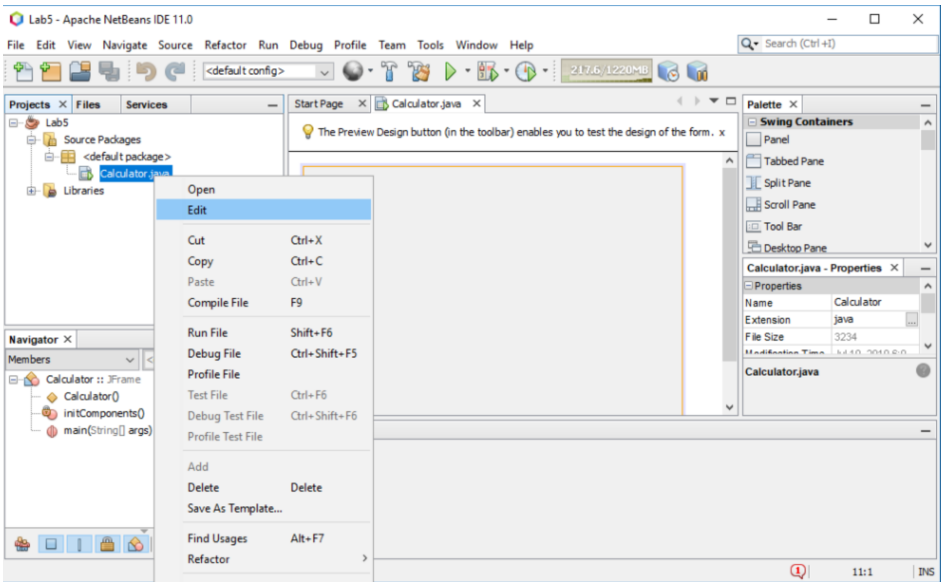
Instead, after creating the project, we're going to right click the "default" package and create a new JFrame Form:

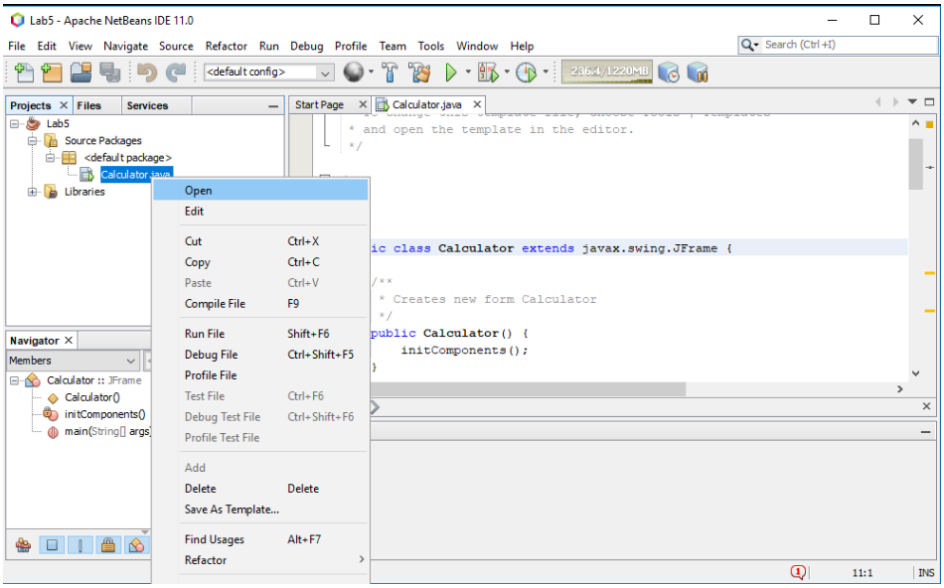


Name the form "Calculator", and leave the `Package` blank:

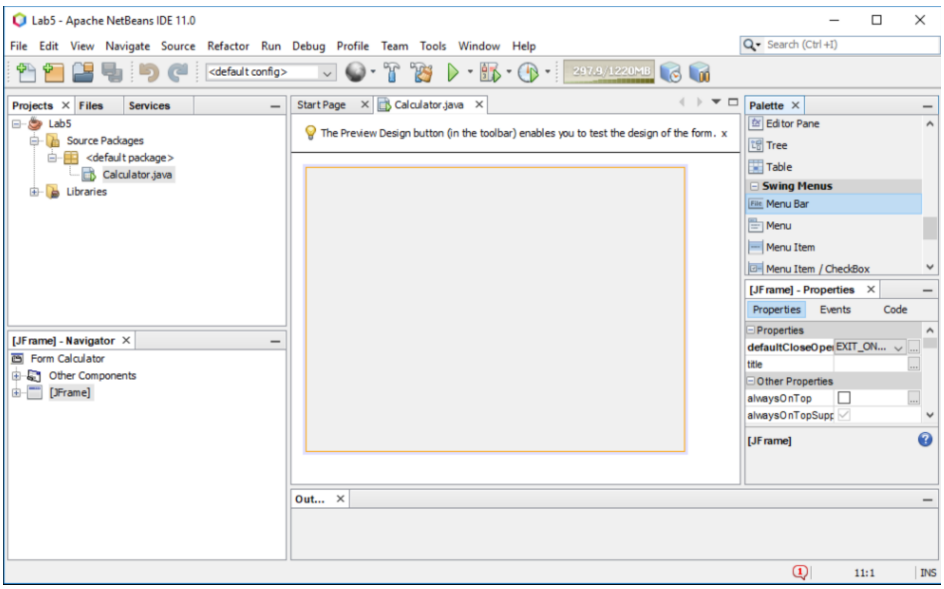


`NetBeans` has created a special entity, and you and `NetBeans` are going to take turns adding Java code to `Calculator.java`. When you right click the file and say "Edit", it will put you in the familiar code editor. But when you right click and say "Open", you enter a WYSIWIG Gui Form editor for the class instead:

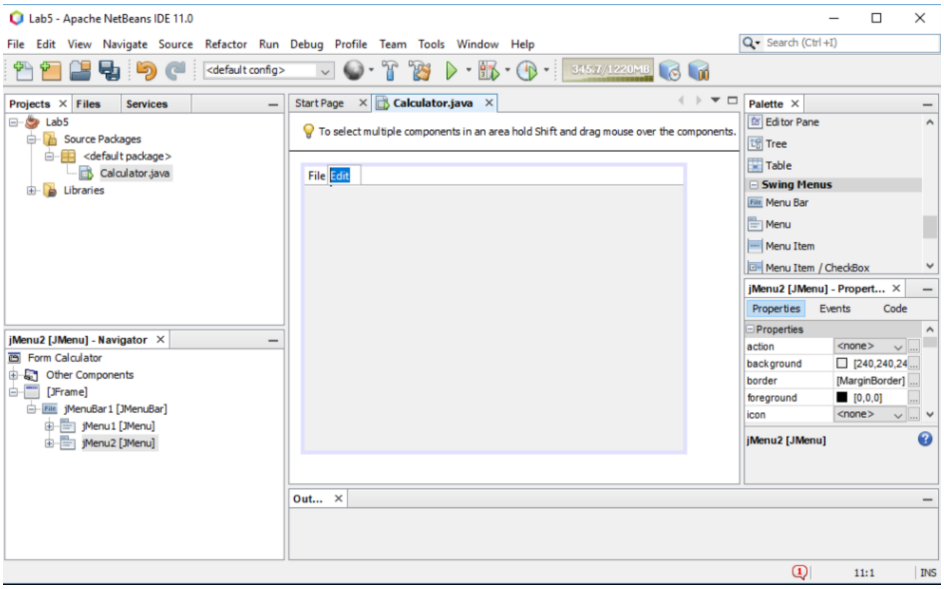




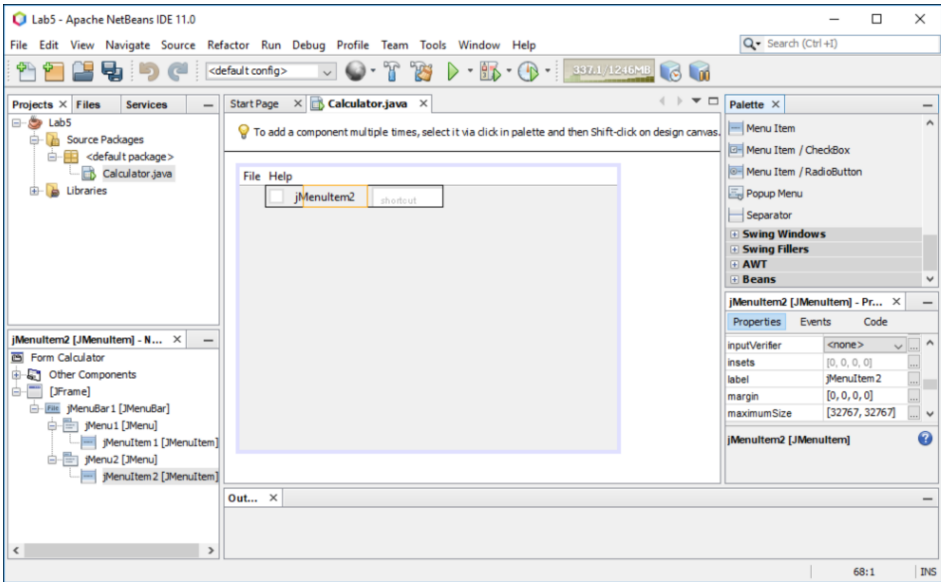
The very first thing we want to add is a `MenuBar`, which you drag in from the section called "Palette":

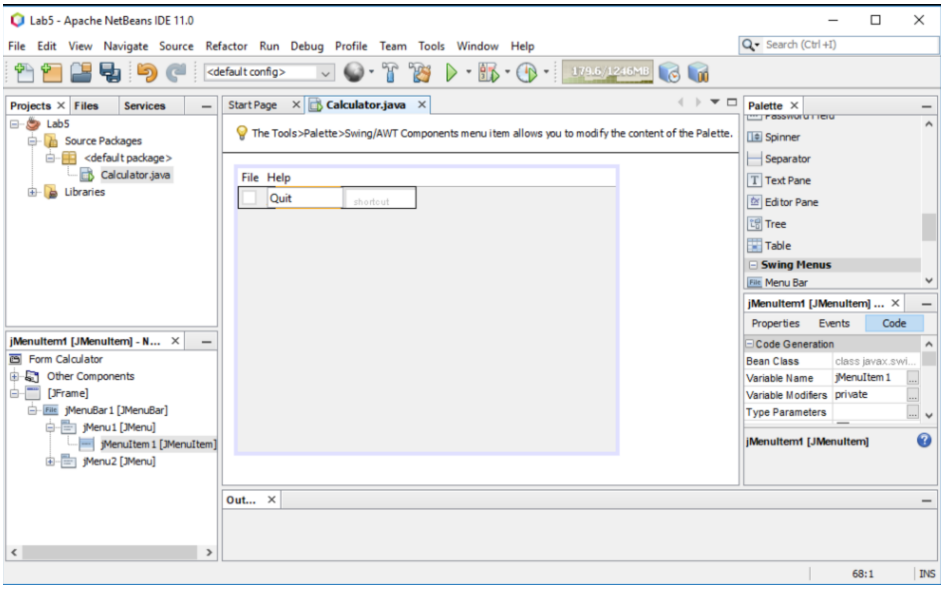


The default `MenuBar` has `File` and `Edit` - click around on "Edit" until it lets you select and change the text, then rename it to "Help":

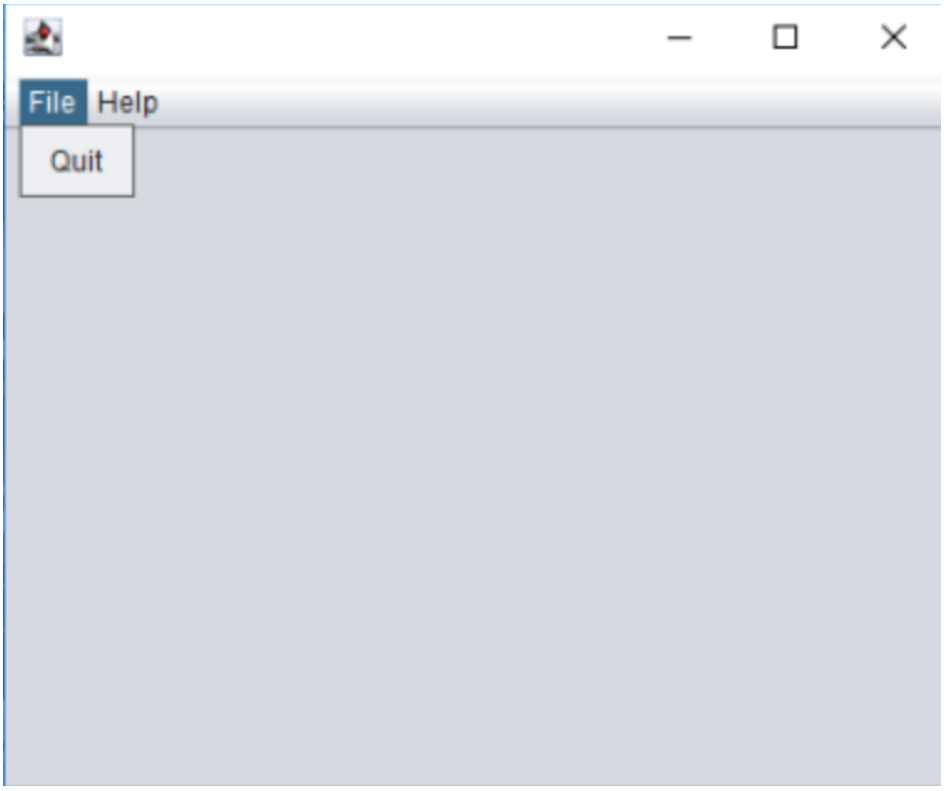


These menus currently have nothing in them. To change that, from the `Palette`, drag one `MenuItem` under each `Menu`. The one under `File` should say `Quit`, and the one under `Help` should say `About`. You can change the display text either by clicking and editing like you did the Menu headings, or you can find the `Text` entry under `Properties` off to the right and edit it there.

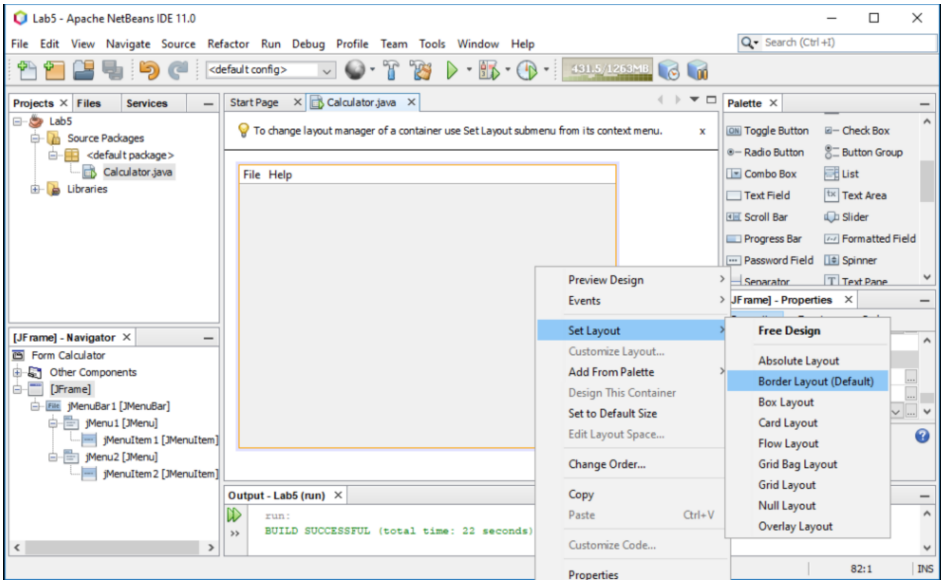




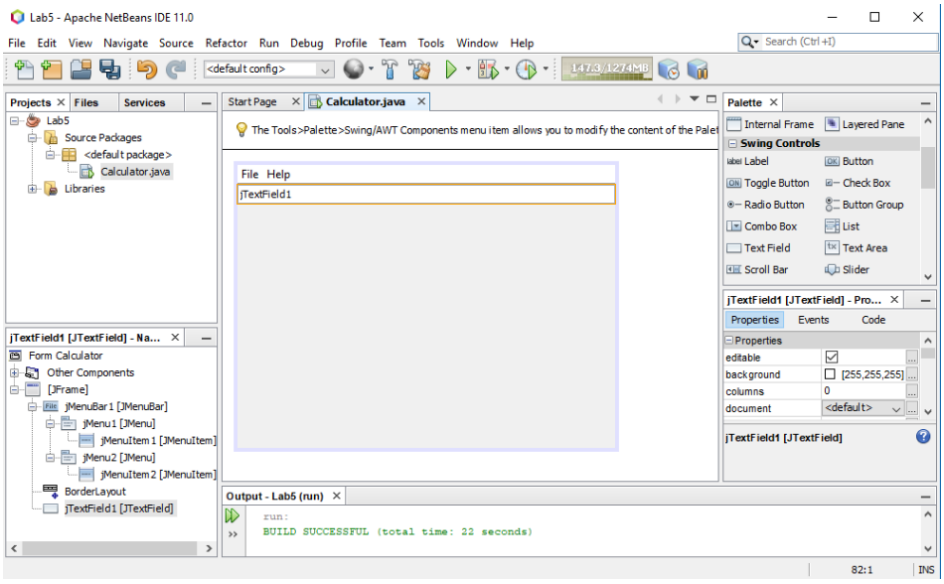
If you were to run the program now, you'd see this window popup. The `MenuItems` currently do nothing when selected.



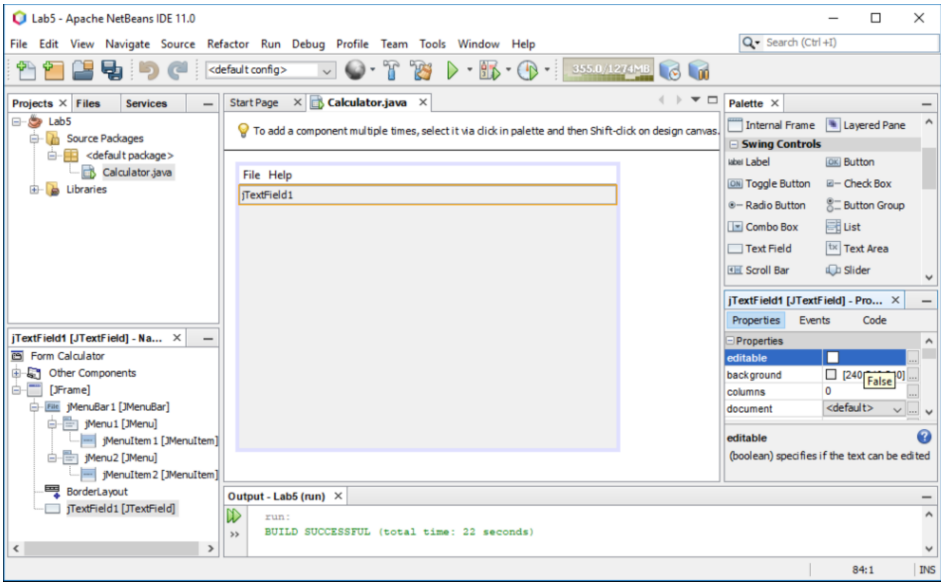
Right click anywhere under the menus, and select "Layout" - confirm that `BorderLayout` is selected:

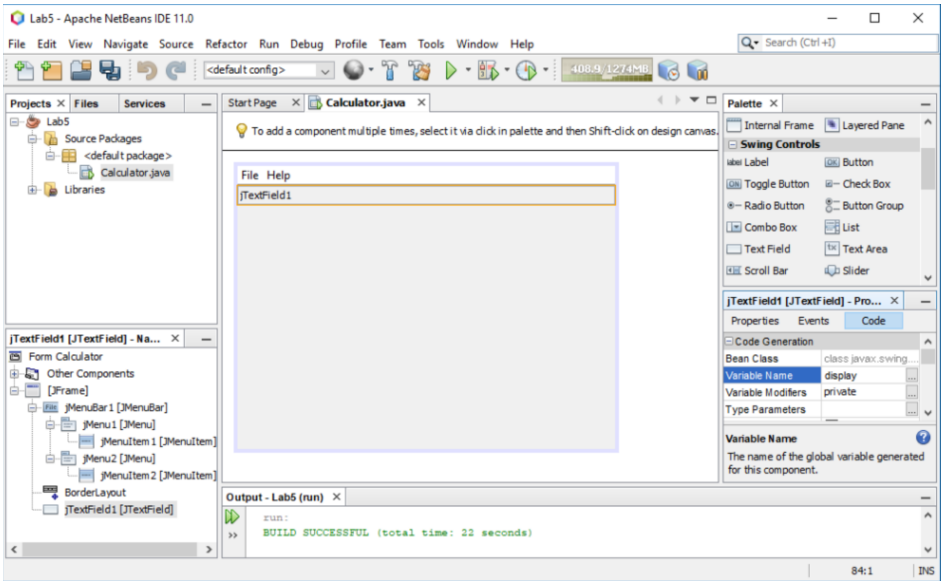


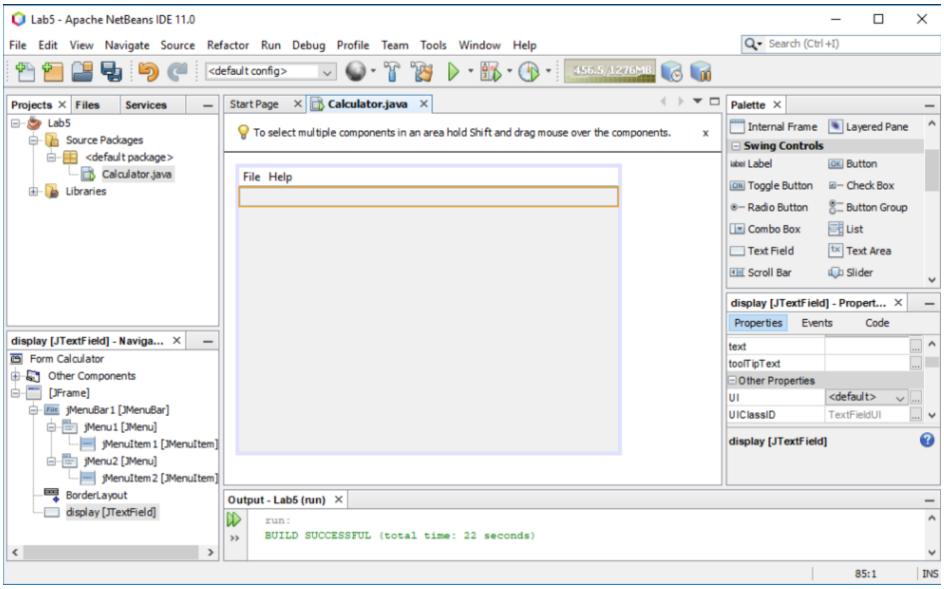
Next, we drag in a `TextField`, where we will show numbers in the calculator. When dragging, make sure to drop just below the Menus, so that it registers with a `NORTH` layout attribute. If it doesn't go where you want it, keep trying to drag it to where it should go until it gets there:



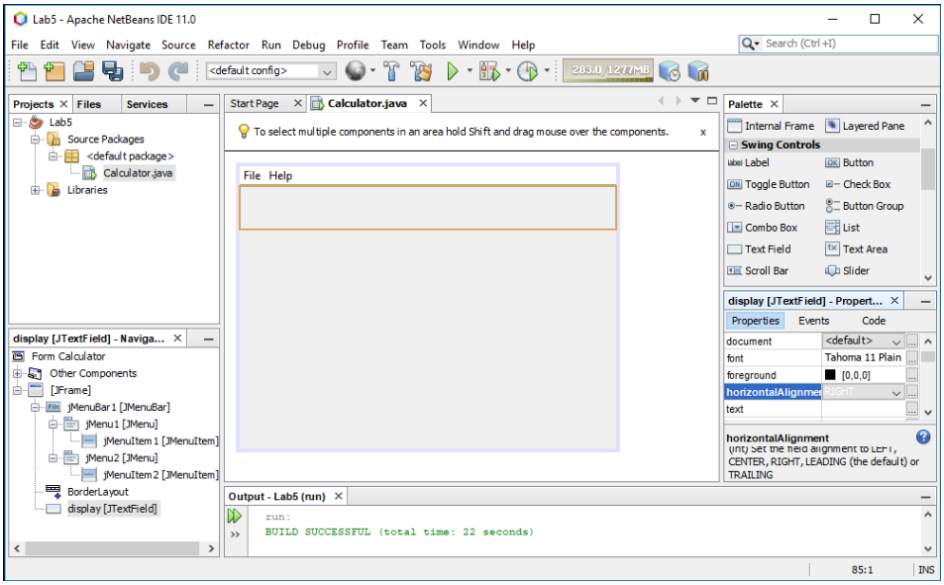
With the `TextField` selected, we want to change some of the fields using the Properties tab. We want `Editable` off; to change the name of `TextField` in code to `display`; and we want the default text for the `TextField` to be `""`:

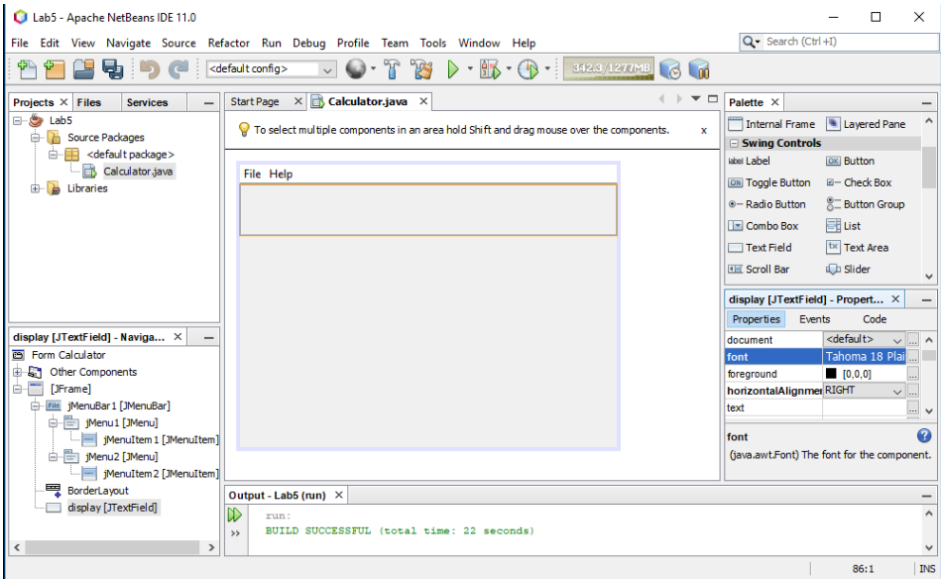

Lab5 - Apache NetBeans IDE 11.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
Lab5
Source Packages
<default package>
Calculator.java
Libraries
Start Page
Calculator.java
To add a component multiple times, select it via click in palette and then Shift-click on design canvas.
File Help
jTextField1
Palette
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
jTextField1 [JTextField] - Pro...
Properties
Events
Code
editable
background
columns
document
<default>
False
editable
(boolean) specifies if the text can be edited
jTextField1 [JTextField] - Na...
Form Calculator
Other Components
[JFrame]
jMenuBar1 [JMenuBar]
jMenu1 [JMenu]
jMenuItem1 [JMenuItem]
jMenu2 [JMenu]
jMenuItem2 [JMenuItem]
BorderLayout
jTextField1 [JTextField]
Output - Lab5 (run)
run:
BUILD SUCCESSFUL (total time: 22 seconds)
84:1
INS

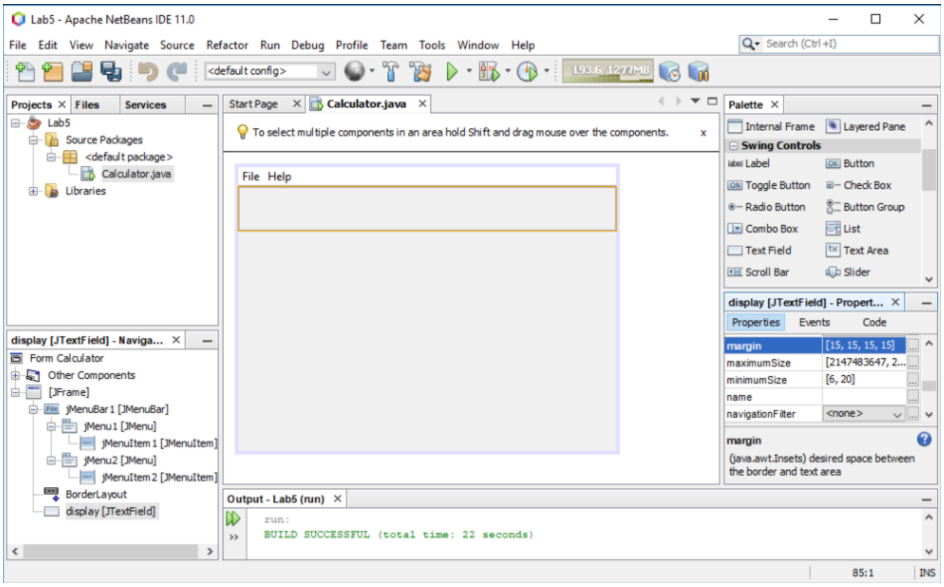

Lab5 - Apache NetBeans IDE 11.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
Lab5
Source Packages
<default package>
Calculator.java
Libraries
Start Page
Calculator.java
To add a component multiple times, select it via click in palette and then Shift-click on design canvas.
File Help
jTextField1
Palette
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
jTextField1 [JTextField] - Pro...
Properties
Events
Code
Code Generation
Bean Class
class javax.swing...
Variable Name
display
Variable Modifiers
private
Type Parameters
Variable Name
The name of the global variable generated for this component.
jTextField1 [JTextField] - Na...
Form Calculator
Other Components
[JFrame]
jMenuBar1 [JMenuBar]
jMenu1 [JMenu]
jMenuItem1 [JMenuItem]
jMenu2 [JMenu]
jMenuItem2 [JMenuItem]
BorderLayout
jTextField1 [JTextField]
Output - Lab5 (run)
run:
BUILD SUCCESSFUL (total time: 22 seconds)
84:1
INS



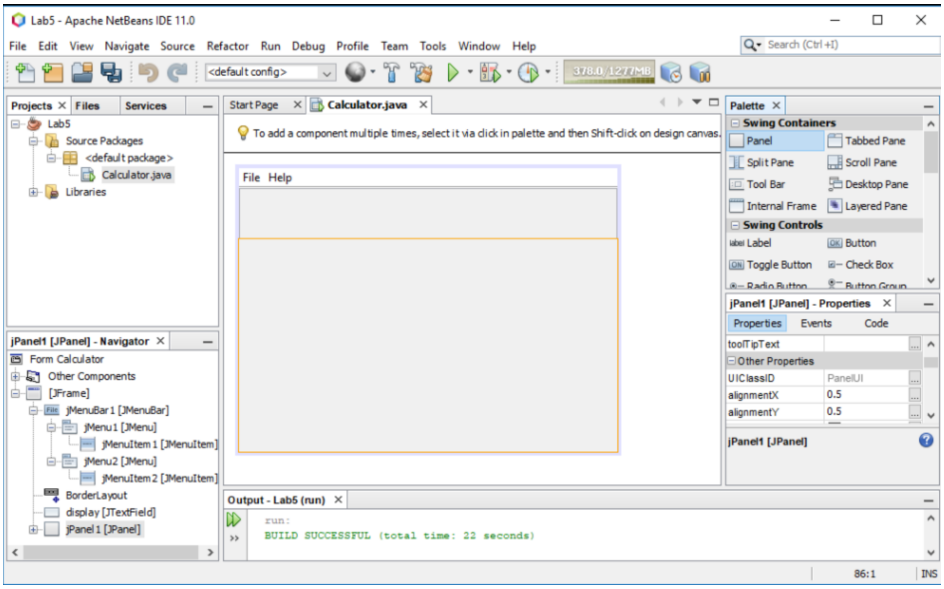
We're not done with `display` yet. Still selected, we want to change text alignment to "Right" using `horizontalAlignment`; to change the font size to be bigger; and to add some margins around the four sides of the `TextField`, essentially making it bigger when we start adding buttons later:



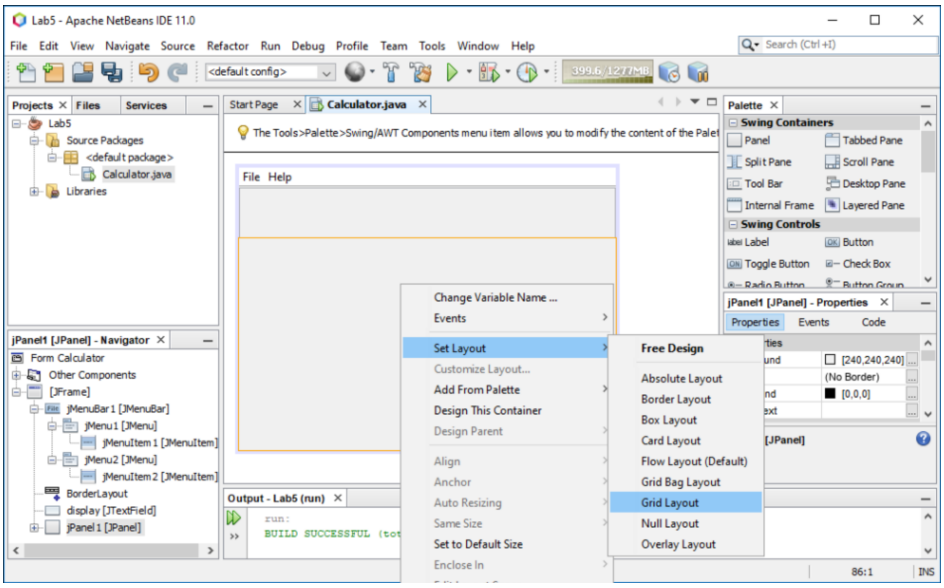


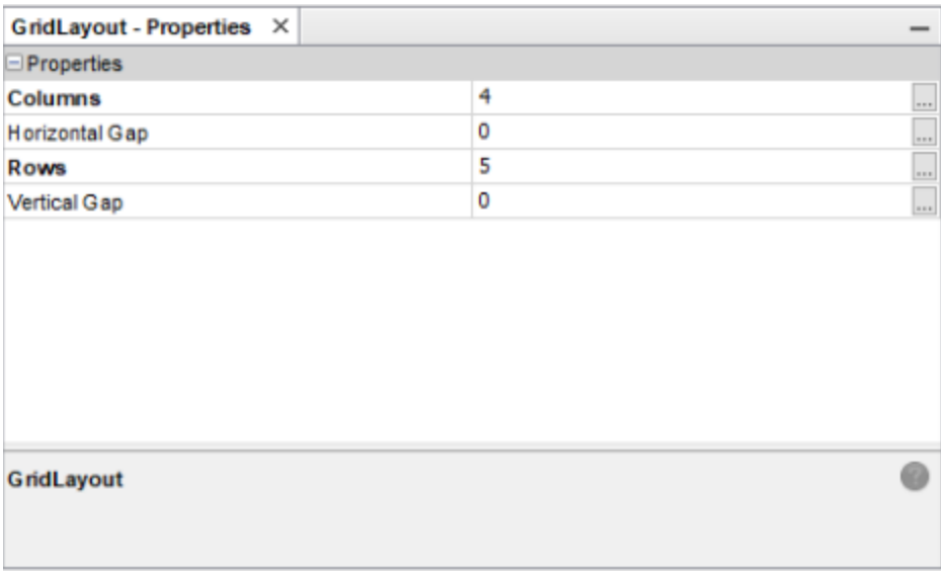


Now we're ready to add our buttons. First we're going to add a `JPanel`, which will let us apply a `GridLayout` to our buttons. When dragging and dropping the panel, make sure it lands as `BorderLayout.CENTER` on the overall form, or drag it around until it's where you want it. Sometimes `NetBeans` gets confused and makes `GUI` elements children of the wrong thing - you can change that by dragging over in the bottom left where it says `Navigator`:

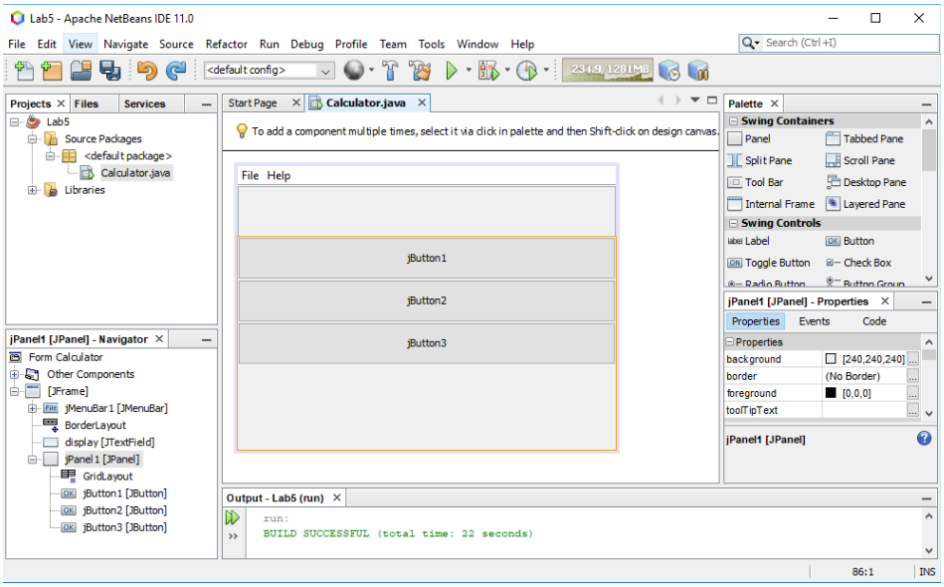


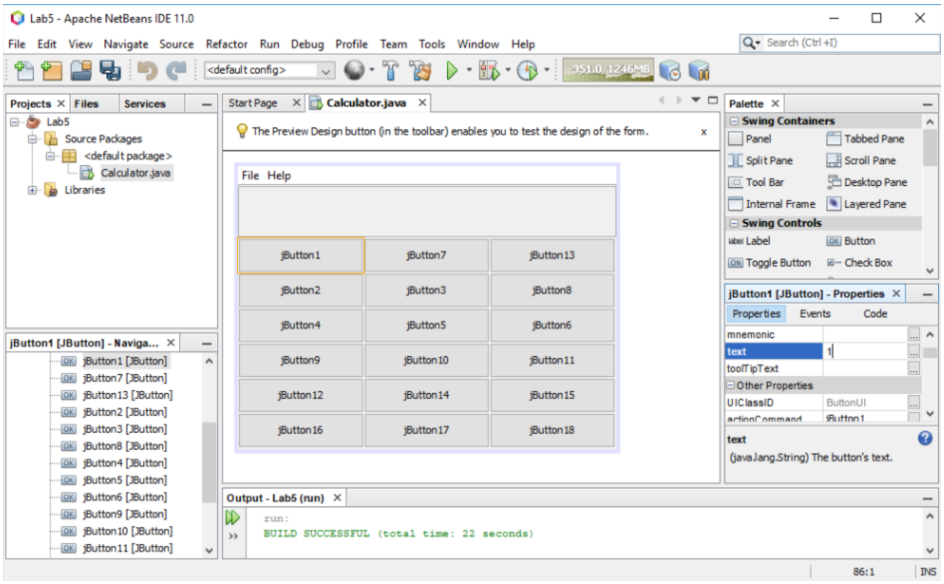
Change the Layout to `GridLayout`, and via `Properties` change it to 4 columns and 5 rows:

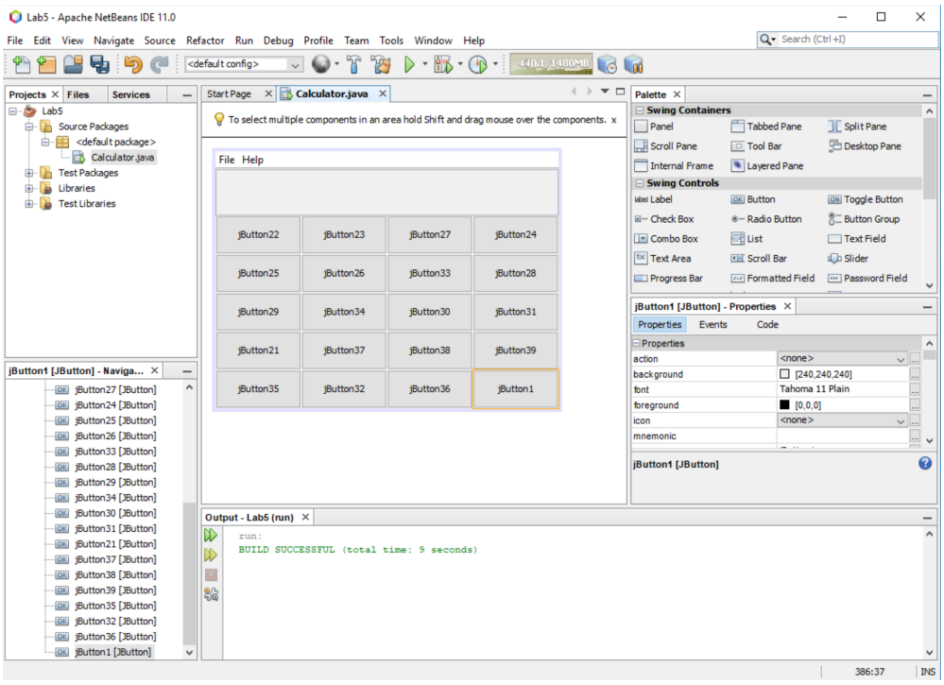




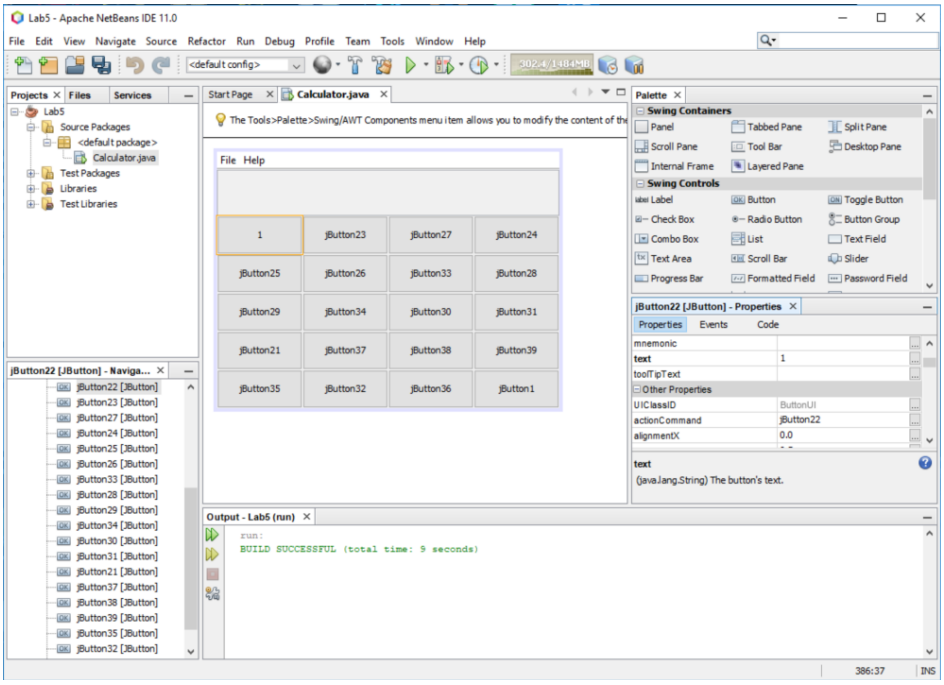
Now we're going to add a bunch of `JButtons`. 20 to be exact! If you're successfully dragging them into the `Panel`, and your `GridLayout` selection is right, they should auto-arrange into a nice 4x5 grid. Otherwise, make sure you aren't accidentally setting them as children of the underlying `Form`, and that your earlier settings are correct. When you're just starting, the buttons will arrange themselves in fewer rows and columns than the final result. Depending on `NetBeans`, your buttons names may also be different from the screenshots - that's ok.

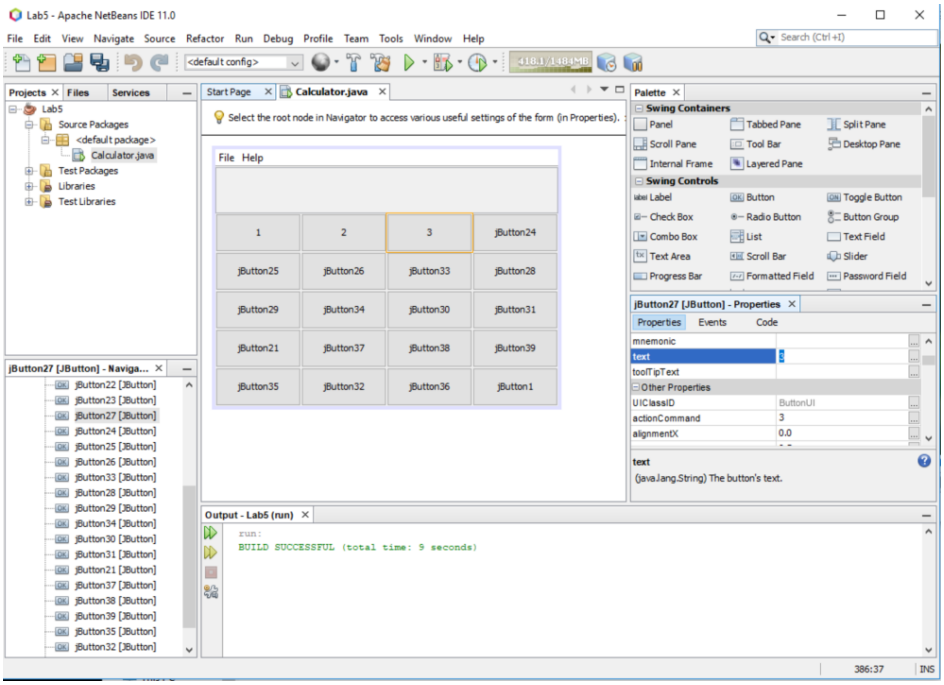


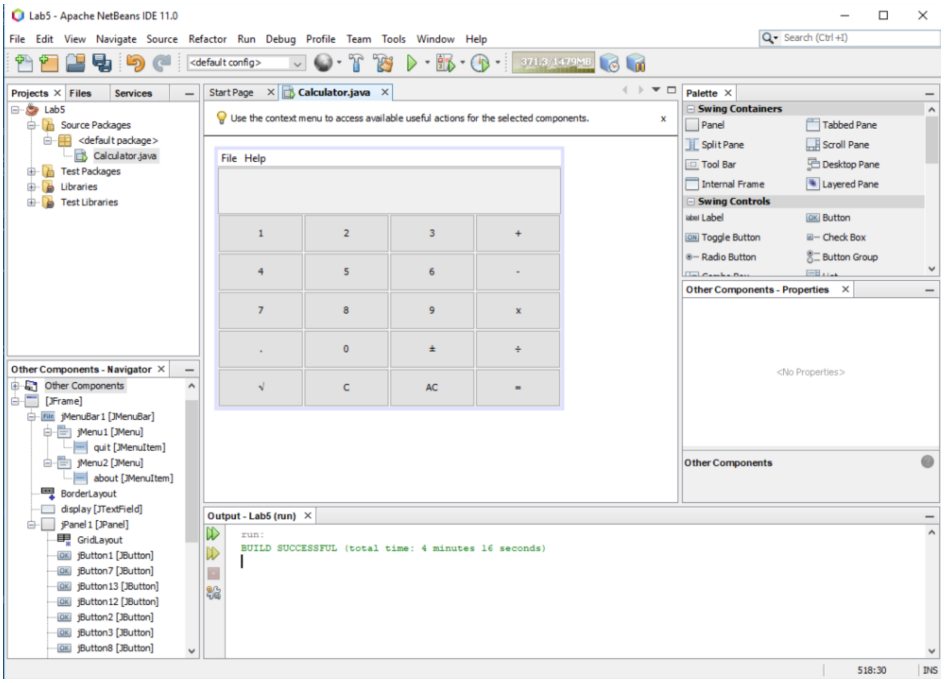




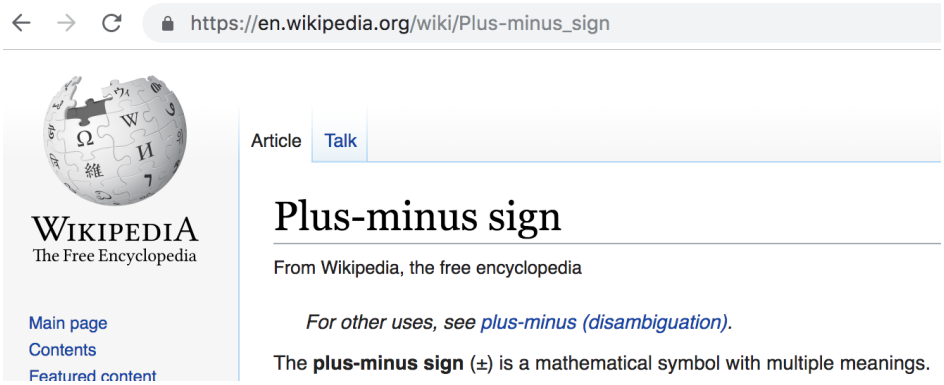
It's starting to look like a calculator, isn't it? Now we need to go in and make the buttons say what we want them to say at runtime. Clicking on any button to select it, you can change its text under the `Properties` section:


Lab5 - Apache NetBeans IDE 11.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
Lab5
Source Packages
<default package>
Calculator.java
Test Packages
Libraries
Test Libraries
Start Page
Calculator.java
The Tools>Palette>Swing/AWT Components menu item allows you to modify the content of th
File Help
1
jButton23
jButton27
jButton24
jButton25
jButton26
jButton33
jButton28
jButton29
jButton34
jButton30
jButton31
jButton21
jButton37
jButton38
jButton39
jButton35
jButton32
jButton36
jButton1
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
jButton22 [JButton] - Properties
Properties
Events
Code
mnemonic
text
1
toolTipText
Other Properties
UIClassID
ButtonUI
actionCommand
jButton22
alignmentX
0.0
text
(java.lang.String) The button's text.
jButton22 [JButton] - Naviga...
Output - Lab5 (run)
run:
BUILD SUCCESSFUL (total time: 9 seconds)
386:37
INS

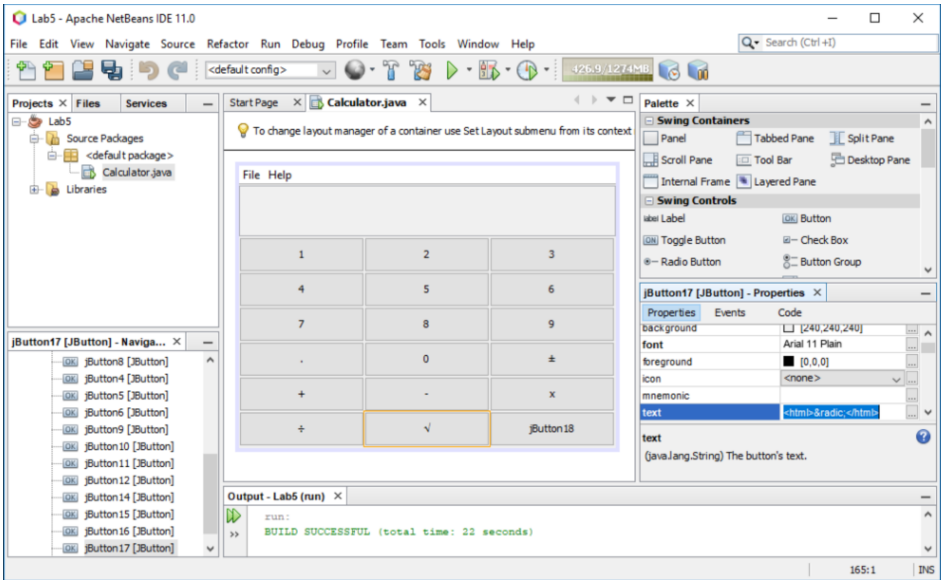

Lab5 - Apache NetBeans IDE 11.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects Files Services
Lab5
Source Packages
<default package>
Calculator.java
Test Packages
Libraries
Test Libraries
Start Page
Calculator.java
Select the root node in Navigator to access various useful settings of the form (in Properties).
File Help
1
2
3
jButton24
jButton25
jButton26
jButton33
jButton28
jButton29
jButton34
jButton30
jButton31
jButton21
jButton37
jButton38
jButton39
jButton35
jButton32
jButton36
jButton1
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
jButton27 [JButton] - Properties
Properties
Events
Code
mnemonic
text
3
toolTipText
Other Properties
UIClassID
ButtonUI
actionCommand
3
alignmentX
0.0
text
(java.lang.String) The button's text.
jButton27 [JButton] - Naviga...
Output - Lab5 (run)
run:
BUILD SUCCESSFUL (total time: 9 seconds)
386:37
INS



A few of the buttons have weird symbols that aren't easily typed on a keyboard. If you don't know how to type arbitrary unicode sequences, feel free to visit Wikipedia and just copy the character from the web page. You can paste it in to `NetBeans` directly[1].

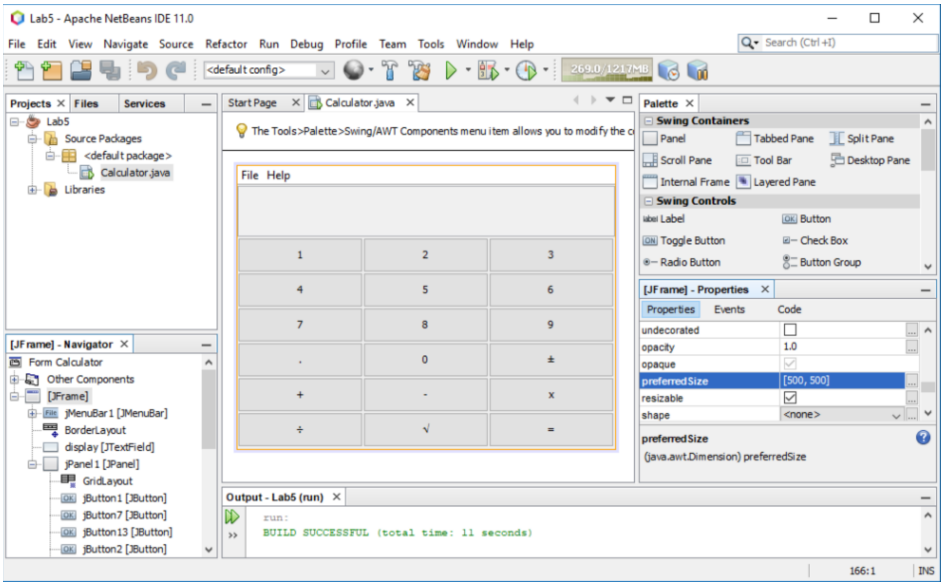


[1]The square root sign may or may not copy over correctly from a Google search (search for "square root symbol" and try to copy/paste the symbol into NetBeans). If it does not work, you may want to try this alternative. *Sometimes* we have found that pasting HTML in as text works in NetBeans. The magic formula is `<html>radic;</html>`.

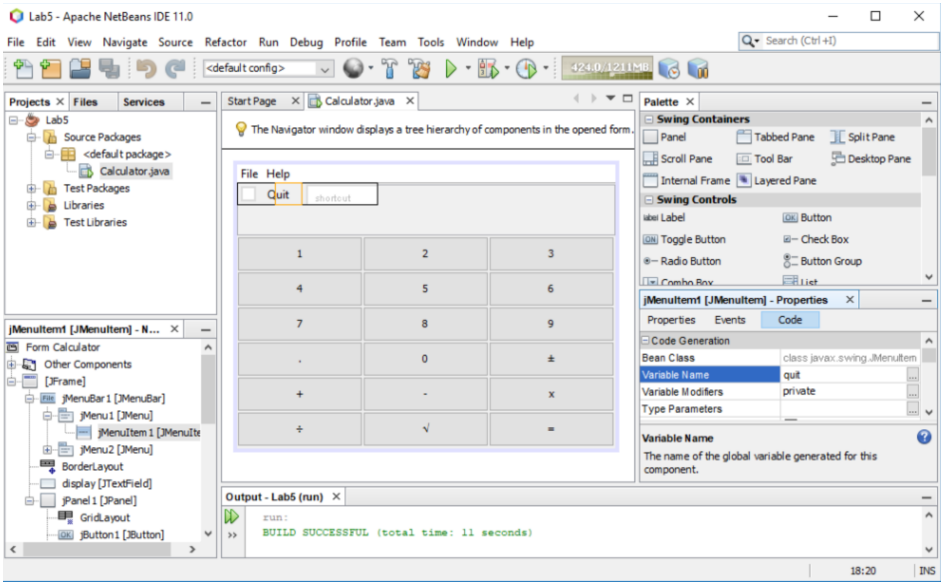


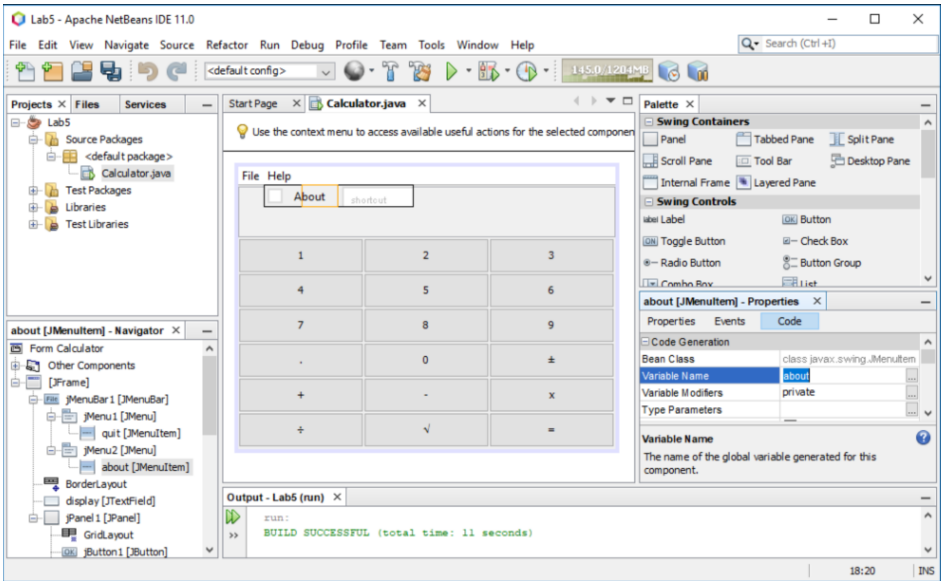
There's a few odds and ends left from earlier. We need to specify the "Preferred Window Size" for the overall form when the program runs[2]:

[2]Note that the next few screenshots don't match the same grid dimensions from earlier. That's OK, and you can have different grid dimensions from the handout, as long as you have the same number of buttons.

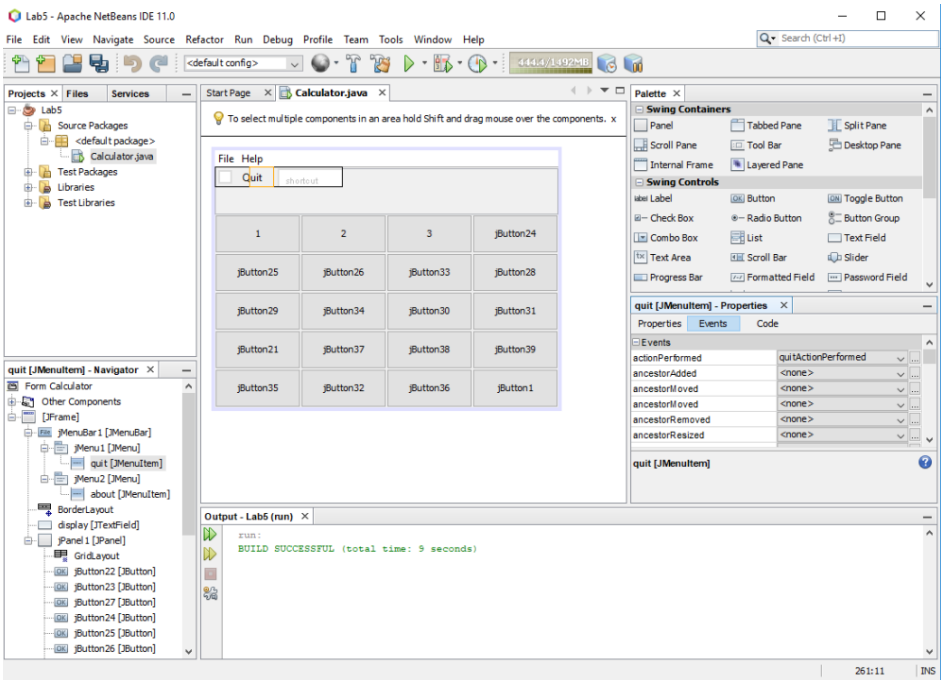


We also need to change the variable names for the `quit` and `about` buttons, using the "Code" tab:



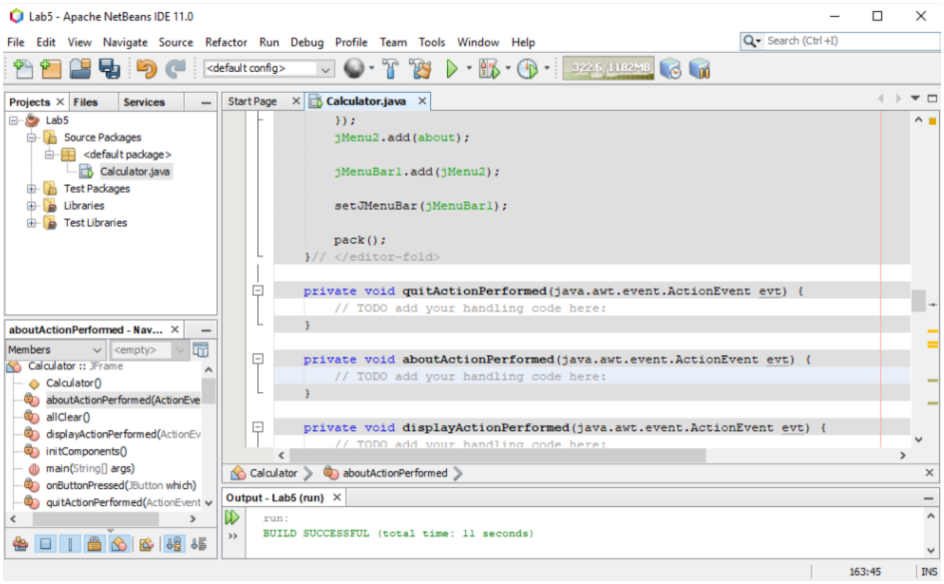


We want those menu items to do something when we click them. To do that, go back and select them, and then look under the `Events` section. You want to click where it says `<none>` under `actionPerformed`, and select the default text it offers you. Doing so should pop you back into the editor:

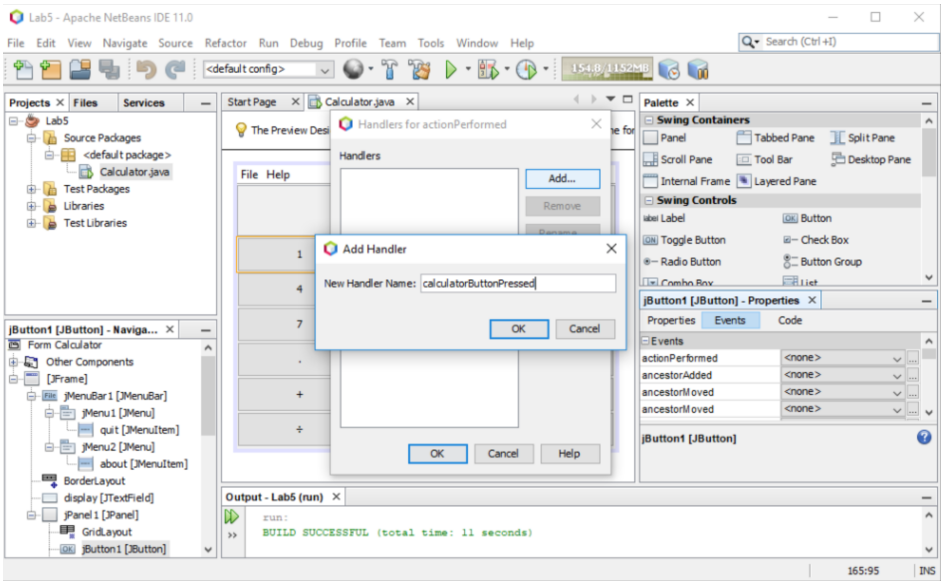


When you've done it for both menu items, you should see something that looks like the following.

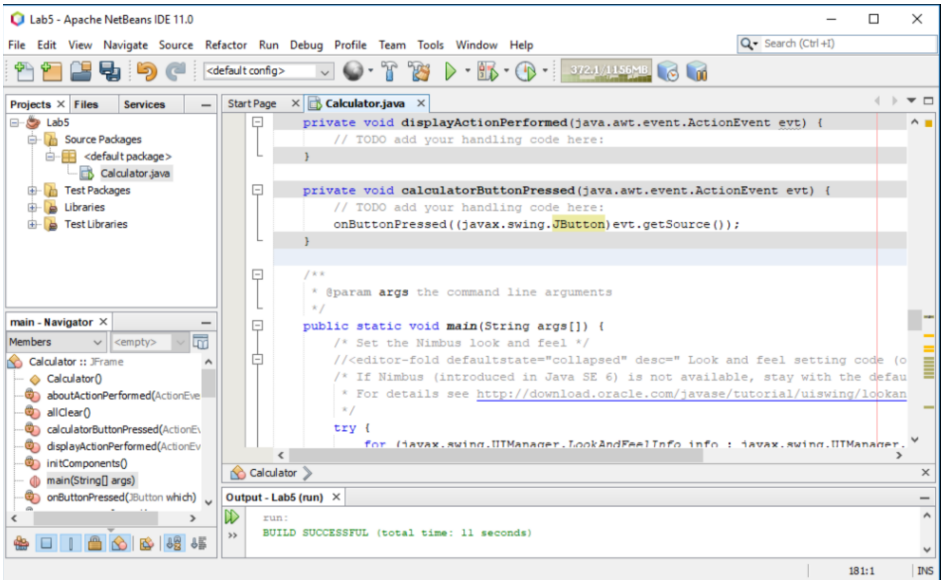
It's up to you to fill `quitActionPerformed()` and `aboutActionPerformed()` in with code (see the next section for hints):



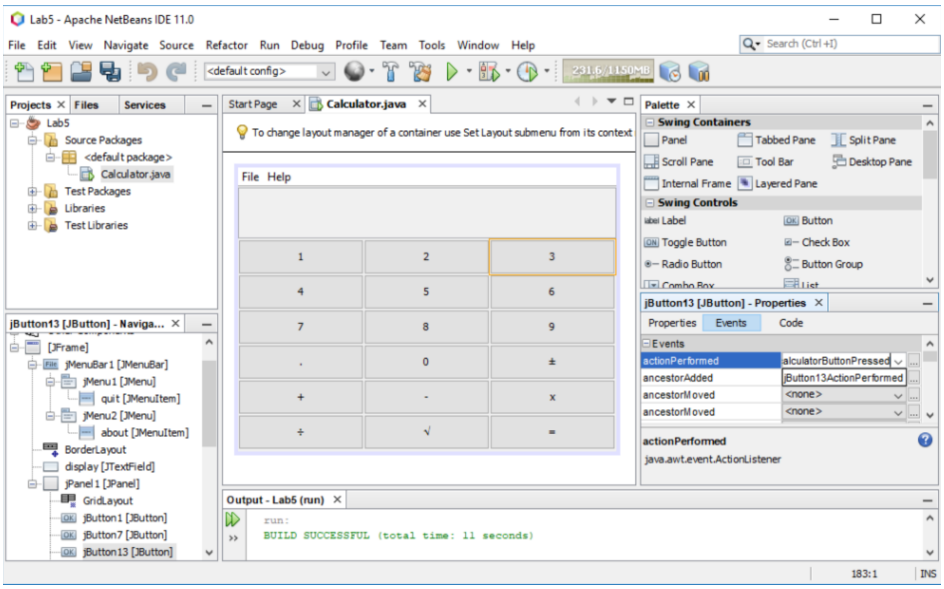
Unfortunately, the next section is a little tedious; `NetBeans` tries to be helpful and instead gets in the way. Select the first button, and go to `Events`. In the triple dot menu to the right of `actionPerformed`, we want to `Add` a new Handler. Call it `calculatorButtonPressed`. Make sure this new handler ends up selected for `actionPerformed` under the drop down menu.



Back in code edit mode, this is the code that should go in that method:



This is where it gets tedious. For all 19 of the remaining buttons, you need to select the `actionPerformed` entry, but then either type in or paste the name `calculatorButtonPressed`. Each time you do, `NetBeans` will switch you to edit mode to go look at it. Calmly switch back (via right click, "Open") to `GUI editing` mode each time.

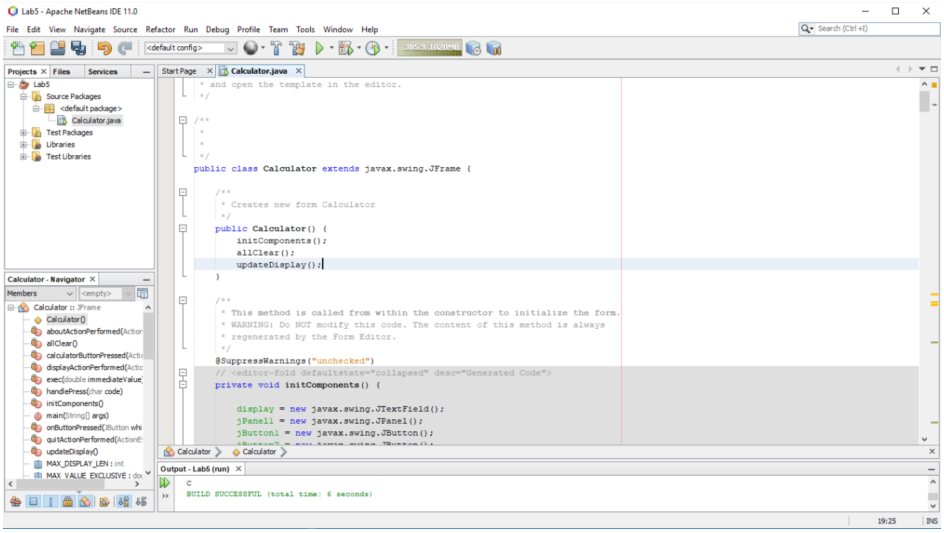


Hurray! You're done editing in `GUI editing` mode, if you want to be. Feel free to poke around and change other settings if you want.

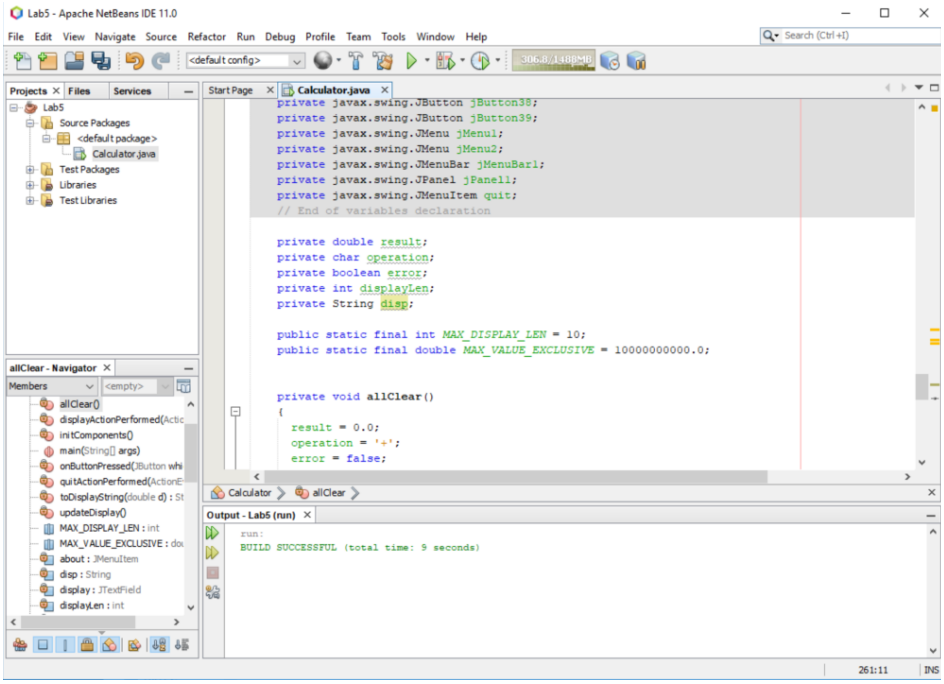
Adding code to `Calculator.java`

We have just a little bit of finishing up to do to the code `NetBeans` added for us.

First, in regular editor mode, go to the constructor and add these two lines:



Second, from the attachment later on in this handout, copy in the **Starter Code**, at the very bottom of your `Calculator.java`. A note of warning - `NetBeans` will not let you edit most of the code it wrote for you, requiring you instead to go back into `GUI editing` mode to change it. But once you've made some space at the end of the file and pasted in the **Starter Code**, you should be able to edit all of that in `NetBeans` as normal:



You now have a few tasks to complete on your own:

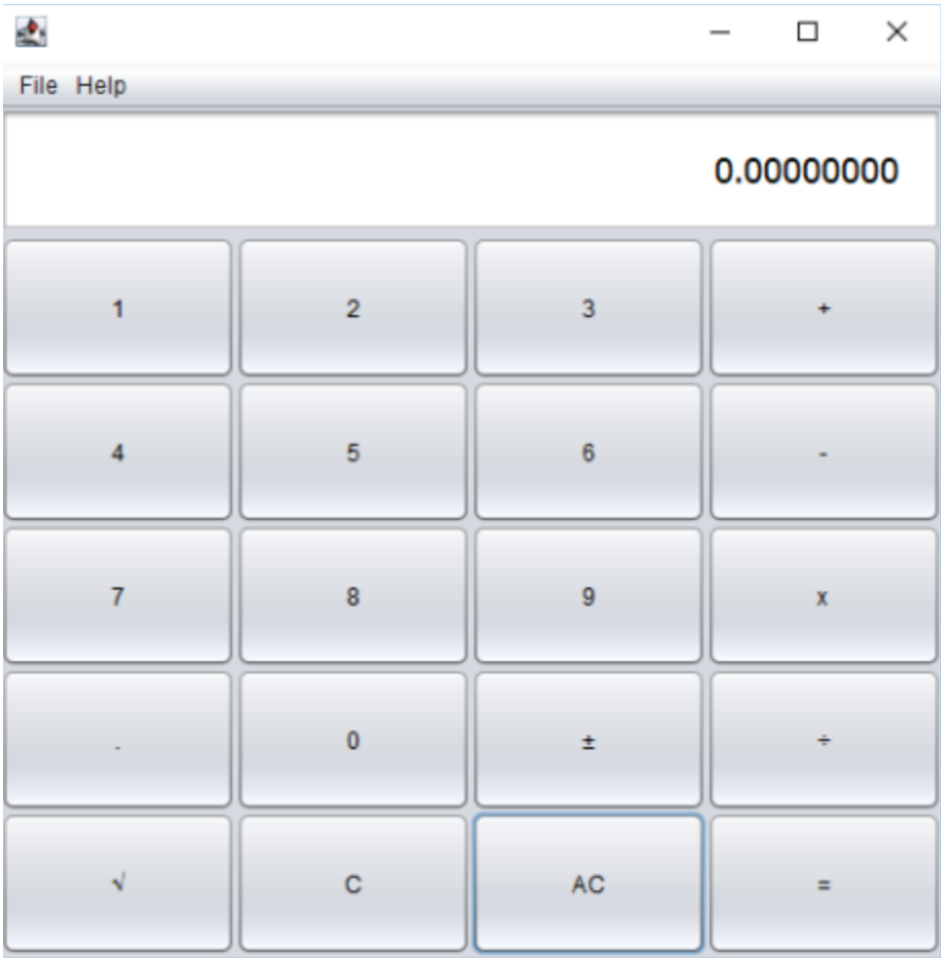
1. Most likely, you will have to fix the line within the onButtonPressed method that checks for the square root symbol. The following line of code may not copy/paste correctly from the PDF, so you will need to correct it:

   ```
   if("<html>&radic;</html>".equals(buttonText) || "√".equals(buttonText))
   ```

2. You will need to add the ÷ symbol to one of the cases of the switch statement, contained in the function *exec*. The ÷ symbol most likely will not copy correctly from the PDF.
3. You will need to re-format the following line of code:

   ```
   String asStr = String.format ("%." + ( MAX_DISPLAY_LEN ) + "f",
   ```

   So that there are no spaces in between each double quotes and the enclosed text. Most likely copying from the PDF will create additional whitespace which will cause your program to give an error.
4. Add a single line of code to the `actionPerformed` handler for the `quit` button, to make the application exit immediately.
5. Add a single line of code to the `actionPerformed` handler for the `about` button, using `JOptionPane`, to announce your name, the year you wrote this, and that this was made by you (and anything else you might like to say). (Hint: javax.swing.JOptionPane.showMessageDialog).


File Help
0.00000000
1
2
3
+
4
5
6
-
7
8
9
x
.
0
±
÷
√
C
AC
=

Unlike in past Labs, this is not a full Java file to begin with. Instead, you will use `NetBeans` to automatically generate much of `Calculator.java`, then copy and paste the following into the bottom of it to finish the Lab.

```
...

private double result;
private char operation;
private boolean error;
private int displayLen;
private String disp;

public static final int MAX_DISPLAY_LEN = 10;
public static final double MAX_VALUE_EXCLUSIVE = 10000000000.0;


private void allClear()
{
  result = 0.0;
  operation = '+';
  error = false;
  displayLen = 0;
  disp = "";
}

// Avoids "scientific notation" format for large numbers;
// and guaranteed to give us too many digits.  We then just
// truncate away the ones we don't need.
private String toDisplayString(double d)
{
    String asStr = String.format("%." + (MAX_DISPLAY_LEN) + "f",
                                 d);

    int maxLen = MAX_DISPLAY_LEN;
    if(asStr.charAt(0)=='-') {
        maxLen += 1;
    }
    asStr = asStr.substring(0, maxLen);

    return asStr;
}

private void updateDisplay()
{
    if (error)
    {
        display.setText("Err");
    }
    else if(displayLen > 0)
    {
        display.setText(disp);
    }
    else
```

```
        {
            display.setText(toDisplayString(result));
        }
    }

    private void exec(double immediateValue)
    {
        System.out.println(result);
        System.out.println(operation);
        System.out.println(immediateValue);
        switch(operation)
        {
            case '=':
                result = immediateValue;
                break;
            case '+':
                result += immediateValue;
                break;
            case '-':
                result -= immediateValue;
                break;
            case 'x':
                result *= immediateValue;
                break;
            case '  ':
                if (immediateValue == 0)
                {
                    error = true;
                }
                else
                {
                    result /= immediateValue;
                }
                break;
        }
        System.out.println(":=> " + result);
        double check = Math.abs(result);
        if (check >= MAX_VALUE_EXCLUSIVE)
        {
            error = true;
        }

        displayLen = 0;
        disp = "";
    }

    private void handlePress(char code)
    {
        if (code == 'A')
        {
            allClear();
            return;
        }
```

```
if(error)
{
    return;
}

if(code == 'C')
{
    disp = "";
    displayLen = 0;
}
else if(code >= '0' && code <= '9' || code == '.' ) {
    if(code == '.' && disp.indexOf('.') != -1)
    {
        // Just ignore the button press; can't have more than one '.'
        // This is sometimes called "eating" user input.
    }
    else if (displayLen < MAX_DISPLAY_LEN + (code == '.' ? 1 : 0))
    {
        // special case - if disp is "0", just overwrite it.
        if ("0".equals(disp) && code != '.')
        {
            disp = "" + code;
        }
        else
        {
            disp += code;
            displayLen += 1;
        }
    }
}
else if (code == 'r' || code == '±')
{
    double value = displayLen == 0 ? result : Double.valueOf(disp);
    final double intermediate;
    if(code == 'r')
    {
        if(value < 0)
        {
            error = true;
            return;
        }
        intermediate = Math.sqrt(value);
    }
    else
    {
        intermediate = -value;
    }
    disp = toDisplayString(intermediate);
    displayLen = disp.length();
}
else
{
    double value = displayLen == 0 ? result : Double.valueOf(disp);
    exec(value);
```

```
            operation = code;
        }
    }

    private void onButtonPressed(javax.swing.JButton which)
    {
        // Step 1 - Turn `which` into a letter
        String buttonText = which.getText();
        final char letter;
        if("<html>&radic;</html>".equals(buttonText) || "√".equals(buttonText))
        {
            letter = 'r';
        }
        else
        {
            letter = buttonText.charAt(0);
        }

        // Step 2 - Apply the effect of that button (see the "Background"
        //          section in the handout).
        handlePress(letter);

        // Step 3 - change what's shown in the `display` JTextField.
        updateDisplay();
    }

    ...
```