Introduction to GUI Building

Contributed by Saleem Gul and Tomas Pavek

This beginner tutorial teaches you how to create a simple graphical user interface and add simple back-end functionality. In particular we will show how to code the behaviour of buttons and fields in a Swing form.
We will work through the layout and design of a GUI and add a few buttons and text fields. The text fields will be used for receiving user input and also for displaying the program output. The button will initiate the functionality built into the front end. The application we create will be a simple but functional calculator.

Expected duration: 15 minutes

Contents

Exercise 1: Creating a Project

The first step is to create an IDE project for the application that we are going to develop. We will name our project `NumberAddition`.

1. Choose `File` > `New Project`. Alternatively, you can click the New Project icon in the IDE toolbar.
2. In the Categories pane, select the Java node. In the Projects pane, choose Java Application. Click Next.
3. Type `NumberAddition` in the Project Name field and specify a path, for example, in your home directory, as the project location.
4. (Optional) Select the Use Dedicated Folder for Storing Libraries checkbox and specify the location for the libraries folder. See Sharing Project Libraries for more information on this option.
5. Deselect the Create Main Class checkbox if it is selected.
6. Click Finish.

Exercise 2: Building the Front End

To proceed with building our interface, we need to create a Java container within which we will place the other required GUI components. In this step we'll create a container using the `JFrame` component. We will place the container in a new package, which will appear within the Source Packages node.

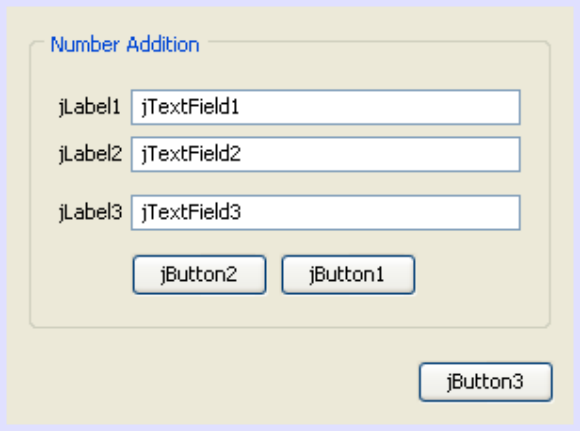
Create a JFrame container

1. In the Projects window, right-click the `NumberAddition` node and choose `New` > `Other`.
2. In the New File dialog box, choose the `Swing GUI Forms` category and the `JFrame Form` file type. Click Next.
3. Enter `NumberAdditionUI` as the class name.
4. Enter `my.numberaddition` as the package.
5. Click Finish.

The IDE creates the `NumberAdditionUI` form and the `NumberAdditionUI` class within the `NumberAddition` application, and opens the `NumberAdditionUI` form in the GUI Builder. The `my.NumberAddition` package replaces the default package.

Adding Components: Making the Front End

Next we will use the Palette to populate our application's front end with a JPanel. Then we will add three JLabels, three JTextFields, and three JButtons. If you have not used the GUI Builder before, you might find information in the Designing a Swing GUI in NetBeans IDE tutorial on positioning components useful.
Once you are done dragging and positioning the aforementioned components, the JFrame should look something like the following screenshot.



If you do not see the Palette window in the upper right corner of the IDE, choose Window > Palette.

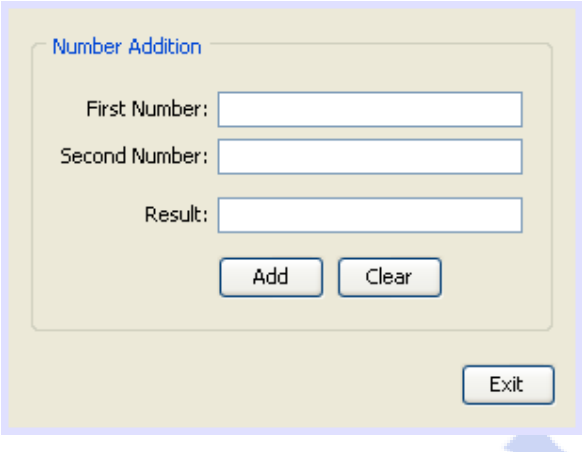
1. Start by selecting a Panel from the Swing Containers category on Palette and drop it onto the JFrame.
2. While the JPanel is highlighted, go to the Properties window and click the ellipsis (...) button next to Border to choose a border style.
3. In the Border dialog, select TitledBorder from the list, and type in `Number Addition` in the Title field. Click OK to save the changes and exit the dialog.
4. You should now see an empty titled JFrame that says Number Addition like in the screenshot. Look at the screenshot and add three JLabels, three JTextFields and three JButtons as you see above.

Renaming the Components

In this step we are going to rename the display text of the components that were just added to the JFrame.

1. Double-click `jLabel1` and change the text property to `First Number`
2. Double-click `jLabel2` and change the text to `Second Number`
3. Double-click `jLabel3` and change the text to `Result`
4. Delete the sample text from `jTextField1`. You can make the display text editable by right-clicking the text field and choosing Edit Text from the popup menu. You may have to resize the `jTextField1` to its original size. Repeat this step for `jTextField2` and `jTextField3`.
5. Rename the display text of `jButton1` to `Clear`. (You can edit a button's text by right-clicking the button and choosing Edit Text. Or you can click the button, pause, and then click again.)
6. Rename the display text of `jButton2` to `Add`.
7. Rename the display text of `jButton3` to `Exit`.

Your Finished GUI should now look like the following screenshot:



Exercise 3: Adding Functionality

In this exercise we are going to give functionality to the Add, Clear, and Exit buttons. The `jTextField1` and `jTextField2` boxes will be used for user input and`jTextField3` for program output - what we are creating is a very simple calculator. Let's begin.

Making the Exit Button Work

In order to give function to the buttons, we have to assign an event handler to each to respond to events. In our case we want to know when the button is pressed, either by mouse click or via keyboard. So we will use ActionListener responding to ActionEvent.

1. Right click the Exit button. From the pop-up menu choose Events > Action > actionPerformed. Note that the menu contains many more events you can respond to! When you select the `actionPerformed` event, the IDE will automatically add an ActionListener to the Exit button and generate a handler method for handling the listener's actionPerformed method.

2. The IDE will open up the Source Code window and scroll to where you implement the action you want the button to do when the button is pressed (either by mouse click or via keyboard). Your Source Code window should contain the following lines:

```
3.   private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
4.       //TODO add your handling code here:
                }
```

5. We are now going to add code for what we want the Exit Button to do. Replace the TODO line with `System.exit(0);`. Your finished Exit button code should look like this:

```
6.   private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
7.       System.exit(0);
                }
```

Making the Clear Button Work

1. Click the Design tab at the top of your work area to go back to the Form Design.

2. Right click the Clear button (`jButton1`). From the pop-up menu select Events > Action > actionPerformed.

3. We are going to have the Clear button erase all text from the jTextFields. To do this, you will add some code like above. Your finished source code should look like this:

```
4.   private void jButton1ActionPerformed(java.awt.event.ActionEvent evt){
5.       jTextField1.setText("");
6.       jTextField2.setText("");
7.       jTextField3.setText("");
                }
```

The above code changes the text in all three of our JTextFields to nothing, in essence it is overwriting the existing Text with a blank.

Making the Add Button Work

The Add button will perform three actions.

1. It is going to accept user input from `jTextField1` and `jTextField2` and convert the input from a type String to a float.
2. It will then perform addition of the two numbers.
3. And finally, it will convert the sum to a type String and place it in `jTextField3`.

Lets get started!

1. Click the Design tab at the top of your work area to go back to the Form Design.
2. Right-click the Add button (`jButton2`). From the pop-up menu, select Events > Action > actionPerformed.
3. We are going to add some code to have our Add button work. The finished source code shall look like this:

```
4.   private void jButton2ActionPerformed(java.awt.event.ActionEvent evt){
5.       // First we define float variables.
6.       float num1, num2, result;
7.       // We have to parse the text to a type float.
8.       num1 = Float.parseFloat(jTextField1.getText());
9.       num2 = Float.parseFloat(jTextField2.getText());
10.      // Now we can perform the addition.
11.      result = num1+num2;
12.      // We will now pass the value of result to jTextField3.
13.      // At the same time, we are going to
14.      // change the value of result from a float to a string.
15.      jTextField3.setText(String.valueOf(result));
                        }
```

Our program is now complete we can now build and run it to see it in action.

Exercise 4: Running the Program

To run the program in the IDE:

1. Choose Run > Run Main Project (alternatively, press F6).

   Note: If you get a window informing you that Project NumberAddition does not have a main class set, then you should select`my.NumberAddition.NumberAdditionUI` as the main class in the same window and click the OK button.

To run the program outside of the IDE:

1. Choose Run > Clean and Build Main Project (Shift-F11) to build the application JAR file.
2. Using your system's file explorer or file manager, navigate to the `NumberAddition/dist` directory.

   Note: The location of the `NumberAddition` project directory depends on the path you specified while creating the project in step 3 of the Exercise 1: Creating a Project section.
3. Double-click the `NumberAddition.jar` file.

After a few seconds, the application should start.

Note: If double-clicking the JAR file does not launch the application, see this article for information on setting JAR file associations in your operating system.

You can also launch the application from the command line.

To launch the application from the command line:

1. On your system, open up a command prompt or terminal window.
2. In the command prompt, change directories to the `NumberAddition/dist` directory.
3. At the command line, type the following statement:

```
java -jar  NumberAddition.jar
```

Note: Make sure `my.NumberAddition.NumberAdditionUI` is set as the main class before running the application. You can check this by right-clicking the NumberAddition project node in the Projects pane, choosing Properties in the popup menu, and selecting the Run category in the Project Properties dialog box. The Main Class field should display `my.numberaddition.NumberAdditionUI`.

How Event Handling Works

This tutorial has showed how to respond to a simple button event. There are many more events you can have your application respond to. The IDE can help you find the list of available events your GUI components can handle:

1. Go back to the file `NumberAdditionUI.java` in the Editor. Click the Design tab to see the GUI's layout in the GUI Builder.
2. Right-click any GUI component, and select Events from the pop-up menu. For now, just browse the menu to see what's there, you don't need to select anything.
3. Alternatively, you can select Properties from the Window menu. In the Properties window, click the Events tab. In the Events tab, you can view and edit events handlers associated with the currently active GUI component.
4. You can have your application respond to key presses, single, double and triple mouse clicks, mouse motion, window size and focus changes. You can generate event handlers for all of them from the Events menu. The most common event you will use is an Action event. (Learn best practices for Event handling from Sun's Java Events Tutorial.)

How does event handling work? Every time you select an event from the Event menu, the IDE automatically creates a so-called event listener for you, and hooks it up to your component. Go through the following steps to see how event handling works.

1. Go back to the file `NumberAdditionUI.java` in the Editor. Click the Source tab to see the GUI's source.
2. Scroll down and note the methods `jButton1ActionPerformed()`, `jButton2ActionPerformed()`, and `jButton3ActionPerformed()` that you just implemented. These methods are called event handlers.
3. Now scroll to a method called `initComponents()`. If you do not see this method, look for a line that says `Generated Code`; click the + sign next to it to expand the collapsed `initComponents()` method.
4. First, note the blue block around the `initComponents()` method. This code was auto-generated by the IDE and you cannot edit it.
5. Now, browse through the `initComponents()` method. Among other things, it contains the code that initializes and places your GUI components on the form. This code is generated and updated automatically while you place and edit components in the Design view.

6. In `initComponents()`, scroll down to where it reads

```
7.   jButton3.setText("Exit");
8.   jButton3.addActionListener(new java.awt.event.ActionListener() {
9.       public void actionPerformed(java.awt.event.ActionEvent evt) {
10.             jButton3ActionPerformed(evt);
11.      }
               });
```

This is the spot where an event listener object is added to the GUI component; in this case, you register an ActionListener to the `jButton3`. The ActionListener interface has an actionPerformed method taking ActionEvent object which is implemented simply by calling your `jButton3ActionPerformed`event handler. The button is now listening to action events. Everytime it is pressed an ActionEvent is generated and passed to the listener's actionPerformed method which in turn executes code that you provided in the event handler for this event.

Generally speaking, to be able to respond, each interactive GUI component needs to register to an event listener and needs to implement an event handler. As you can see, NetBeans IDE handles hooking up the event listener for you, so you can concentrate on implementing the actual business logic that should be triggered by the event.