In this chapter learn you through the process of creating the graphical user interface (GUI) for an application called ***AddTwoNumber*** using the NetBeans IDE GUI Builder. In this chapter you will learn how to: use the GUI Builder Interface, create a GUI Container, add, resize, and align components, adjust component anchoring, set component auto-resizing behavior, edit component properties.

Create Windows Form

The IDE's GUI Builder makes it possible to build professional-looking GUIs without an intimate understanding of layout managers. You can lay out your forms by simply placing components where you want them.

create project

Because all Java development in the IDE takes place within projects, we first need to create a new ***AddTwoNumber*** project within which to store sources and other project files. This is Java applications often consist of several IDE projects, for the purposes of this tutorial, we will build a simple application.

create a new AddTwoNumber application project:

1. Choose ***File > New Project***. Alternately, you can click the New Project icon in the IDE toolbar.
2. In the Categories pane, select the Java node and in the Projects pane, choose Java Application. Click Next.
3. Enter ***AddTwoNumber*** in the Project Name field and specify the project location.
4. Leave the Use Dedicated Folder for Storing Libraries checkbox unselected.
5. Ensure that the Set as Main Project checkbox is selected and clear the Create Main Class field.
6. Click Finish.

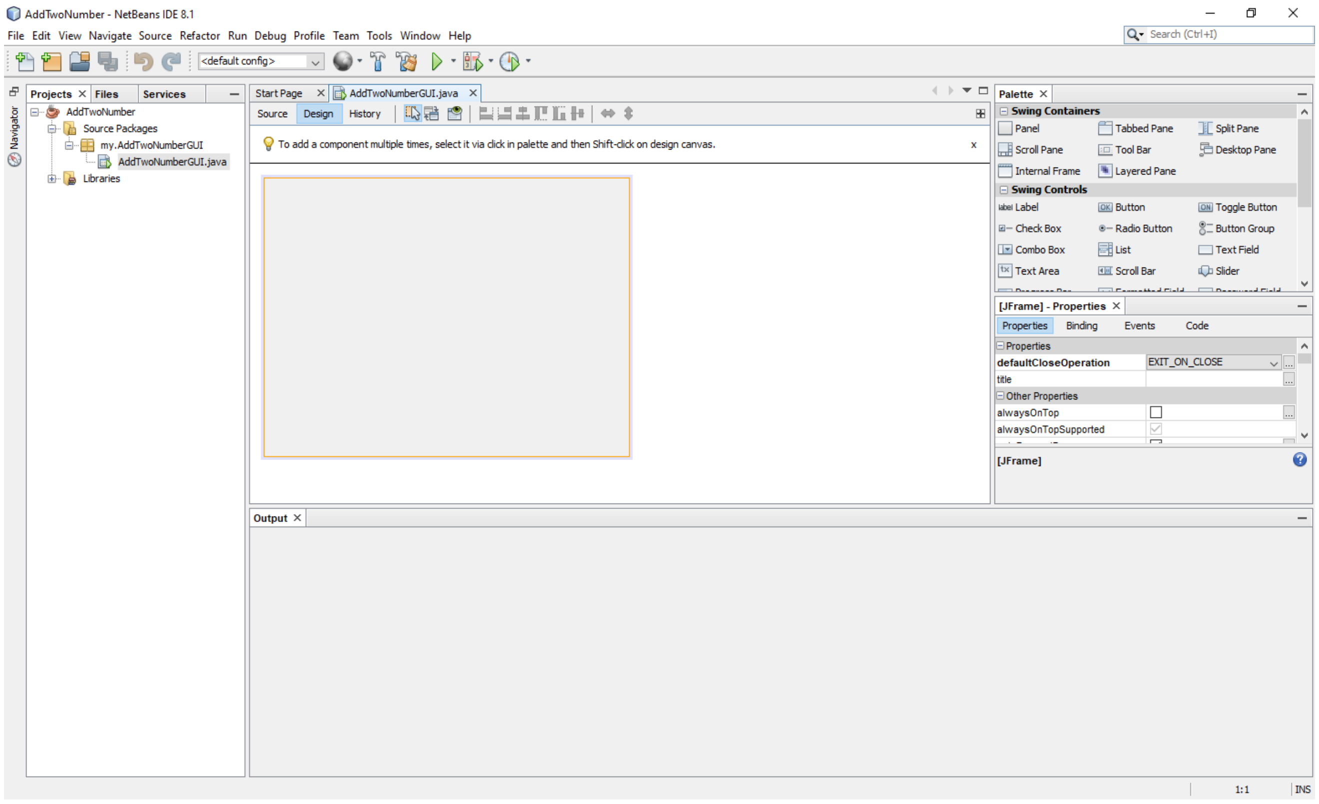
The IDE creates the ***AddTwoNumber*** folder on your system in the designated location. This folder contains all of the project's associated files, including its Ant script, folders for storing sources and tests, and a folder for project-specific metadata. To view the project structure, use the IDE's Files window.

Create JFrame container

After creating the new application, you may have noticed that the Source Packages folder in the Projects window contains an empty <default package> node. 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 and place the container in a new package.

To add a JFrame container:

1. In the Projects window, right-click the ContactEditor node and choose New > JFrame Form.
2. Enter ***AddTwoNumberGUI*** as the Class Name.
3. Enter ***my***. ***AddTwoNumberGU***I as the package.
4. Click Finish and you will see


AddTwoNumber - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
AddTwoNumber
Source Packages
my.AddTwoNumberGUI
AddTwoNumberGUI.java
Libraries
Navigator
Start Page AddTwoNumberGUI.java
Source Design History
To add a component multiple times, select it via click in palette and then Shift-click on design canvas.
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
[JFrame] - Properties
Properties Binding Events Code
Properties
defaultCloseOperation EXIT_ON_CLOSE
title
Other Properties
alwaysOnTop
alwaysOnTopSupported
[JFrame]
Output
1:1 INS

Getting Familiar with the GUI Builder

Now that we've set up a new project for our application, let's take a minute to familiarize ourselves with the GUI Builder's interface.

When we added the JFrame container, the IDE opened the newly-created ***AddTwoNumberGUI*** form in an Editor tab with a toolbar containing several buttons, as shown in the preceding illustration. The ***AddTwoNumber*** form opened in the GUI Builder's Design view and three additional windows appeared automatically along the IDE's edges, enabling you to navigate, organize, and edit GUI forms as you build them.

The GUI Builder's various windows include:

1. **Design Area.** The GUI Builder's primary window for creating and editing Java GUI forms. The toolbar's Source button enables you to view a class's source code, the Design button allows you to view a graphical view of the GUI components, the History button allows you to access the local history of changes of the file. The additional toolbar buttons provide convenient access to common commands, such as choosing between Selection and Connection modes, aligning components, setting component auto-resizing behavior, and previewing forms.
2. **Navigator.** Provides a representation of all the components, both visual and non-visual, in your application as a tree hierarchy. The Navigator also provides visual feedback about what component in the tree is currently being edited in the GUI Builder as well as allows you to organize components in the available panels.

3. **Palette.** A customizable list of available components containing tabs for JFC/Swing, AWT, and JavaBeans components, as well as layout managers. In addition, you can create, remove, and rearrange the categories displayed in the Palette using the customizer.
4. **Properties Window**. Displays the properties of the component currently selected in the GUI Builder, Navigator window, Projects window, or Files window.

If you click the Source button, the IDE displays the application's Java source code in the Editor with sections of code that are automatically generated by the GUI Builder indicated by grey areas (they become blue when selected), called Guarded Blocks. Guarded blocks are protected areas that are not editable in Source view. You can only edit code appearing in the white areas of the Editor when in Source view. If you need to make changes to the code within a Guarded Block, clicking the Design button returns the IDE's Editor to the GUI Builder where you can make the necessary adjustments to the form. When you save your changes, the IDE updates the file's sources.

Concept of window Form (JFrame)

The IDE's GUI Builder solves the core problem of Java GUI creation by streamlining the workflow of creating graphical interfaces, freeing developers from the complexities of Swing layout managers. It does this by extending the current NetBeans IDE GUI Builder to support a straightforward "Free Design" paradigm with simple layout rules that are easy to understand and use. As you lay out your form, the GUI Builder provides visual guidelines suggesting optimal spacing and alignment of components. In the background, the GUI Builder translates your design decisions into a functional UI that is implemented using the new GroupLayout layout manager and other Swing constructs. Because it uses a dynamic layout model, GUI's built with the GUI Builder behave as you would expect at runtime, adjusting to accommodate any changes you make without altering the defined relationships between components. Whenever you resize the form, switch locales, or specify a different look and feel, your GUI automatically adjusts to respect the target look and feel's insets and offsets.

Adding Components: The Basics

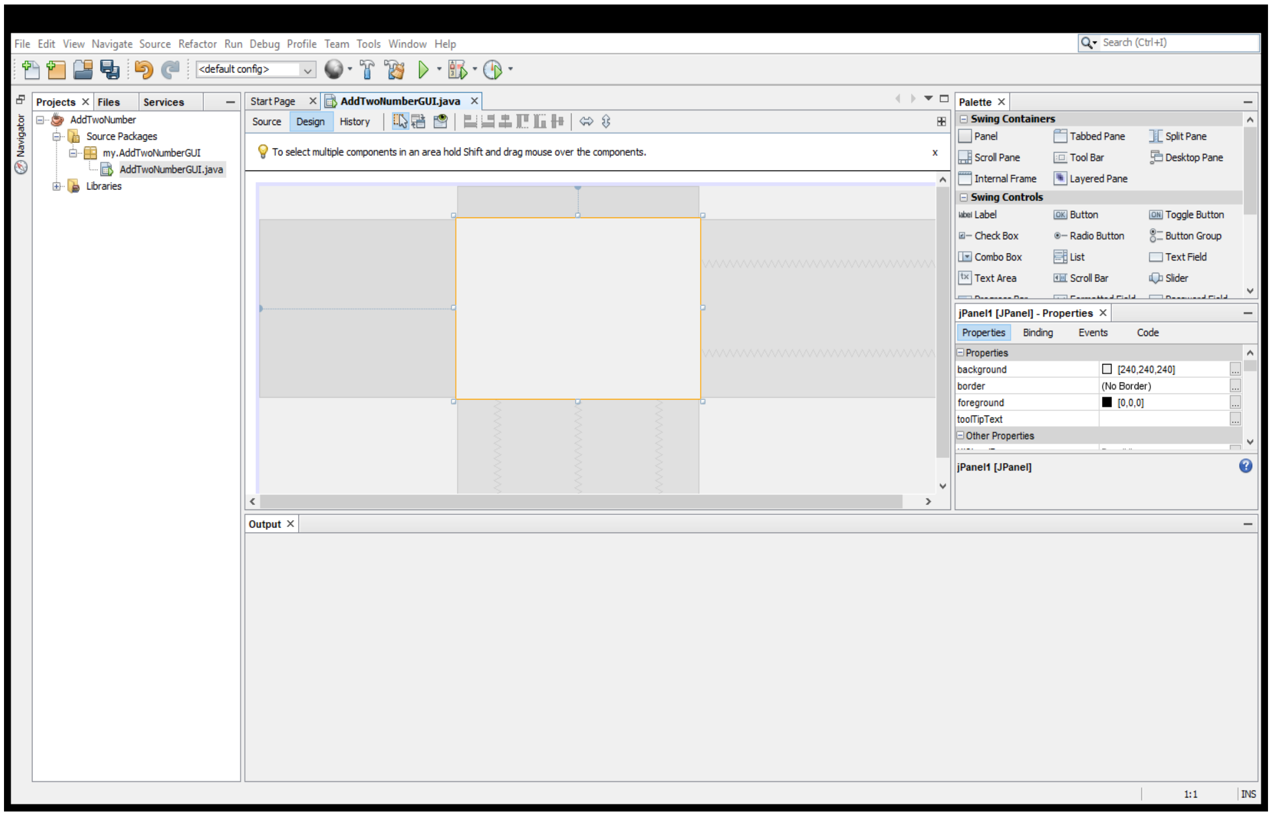
Though the IDE's GUI Builder simplifies the process of creating Java GUIs, it is often helpful to sketch out the way you want your interface to look before beginning to lay it out. Many interface designers consider this a "best practice" technique, however, for the purposes of this tutorial you can simply peek at how our completed form should look by jumping ahead to the Previewing your GUI section.

Since we've already added a JFrame as our form's top-level container, the next step is to add a couple of JPanels which will enable us to cluster the components of our UI using titled borders. Refer to the following illustrations and notice the IDE's "drag and drop" behavior when accomplishing this.

To add a JPanel:

1. In the Palette window, select the Panel component from the Swing Containers category by clicking and releasing the mouse button.
2. Move the cursor to the upper left corner of the form in the GUI Builder. When the component is located near the container's top and left edges, horizontal and vertical alignment guidelines appear indicating the preferred margins. Click in the form to place the JPanel in this location.

The JPanel component appears in the ***AddTwoNumberGUI*** form with orange highlighting signifying that it is selected. After releasing the mouse button, small indicators appear to show the component's anchoring relationships and a corresponding JPanel node is displayed in the Navigator window, as shown in the following illustration.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
AddTwoNumber
Source Packages
my.AddTwoNumberGUI
AddTwoNumberGUI.java
Libraries
Start Page
AddTwoNumberGUI.java
Source Design History
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
Properties Binding Events Code
Output
1:1 INS

To select multiple components in an area hold Shift and drag mouse over the components.

jPanel1 [JPanel] - Properties

Properties

background	[240,240,240]
border	(No Border)
foreground	[0,0,0]
toolTipText	

Other Properties

jPanel1 [JPanel]

To add title borders to the JPanels:

1. Select the top JPanel in the GUI Builder.
2. In the Properties window, click the ellipsis button (...) next to the Border property.
3. In the JPanel Border editor that appears, select the TitledBorder node in the Available Borders pane.
4. In the Properties pane below, enter Name for the Title property.
5. Click the ellipsis (...) next to the Font property, select Bold for the Font Style, and enter 12 for the Size. Click OK to exit the dialogs.

To add a JLabel to the form:

1. In the Palette window, select the Label component from the Swing Controls category.
2. Move the cursor over the Name JPanel we added earlier. When the guidelines appear indicating that the JLabel is positioned in the top left corner of the JPanel with a required margin at the top and left edges, click to place the label.

The JLabel is added to the form and a corresponding node representing the component is added to the Inspector window.

Before going further, we need to edit the display text of the JLabel we just added. Though you can edit component display text at any point, the easiest way is to do this as you add them.

To edit the display text of a JLabel:

1. Double-click the JLabel to select its display text.
2. Type First Number: and press Enter.

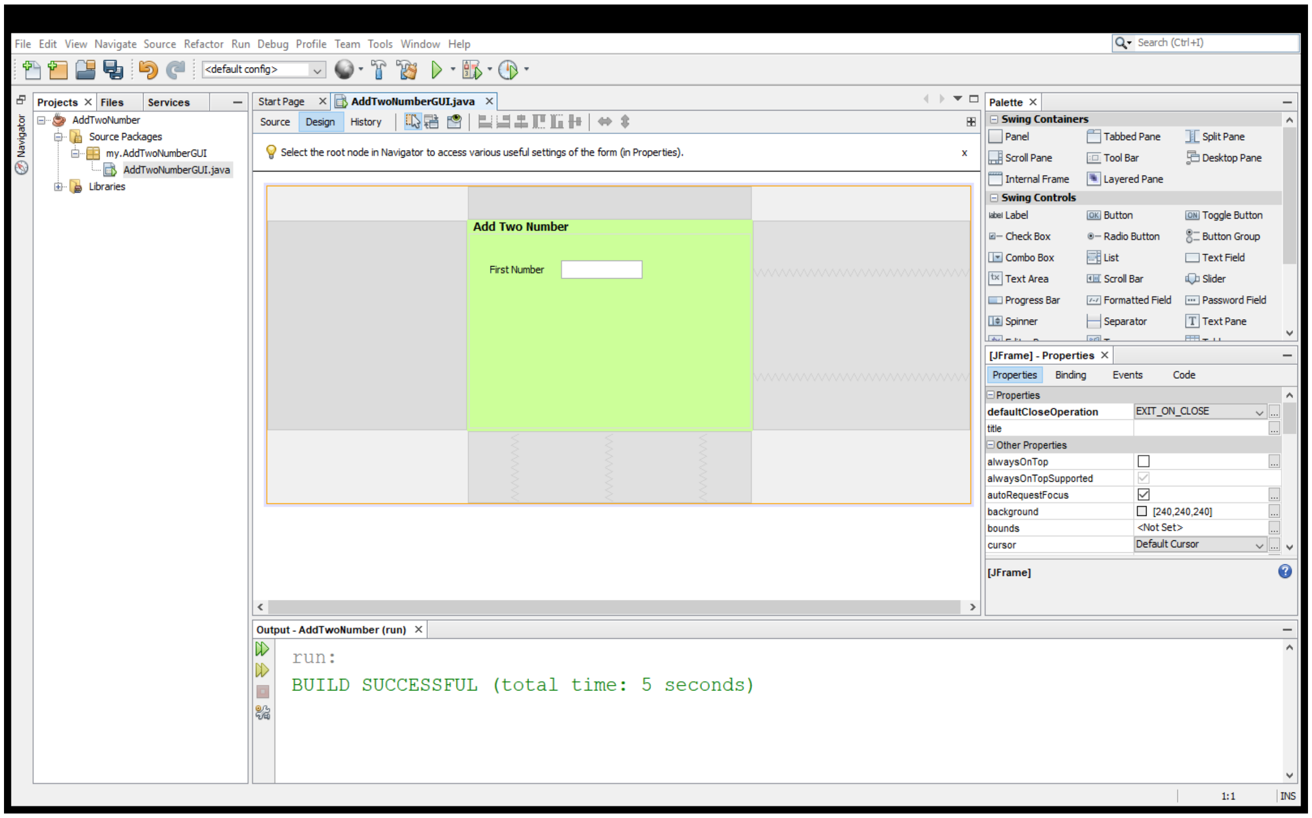
The JLabel's new name is displayed and the component's width adjusts as a result of the edit.

Now we'll add a JTextField so we can get a glimpse of the GUI Builder's baseline alignment feature.

To add a JTextField to the form:

1. In the Palette window, select the Text Field component from the Swing Controls category.
2. Move the cursor immediately to the right of the First Name: JLabel we just added. When the horizontal guideline appears indicating that the JTextField's baseline is aligned with that of the JLabel and the spacing between the two components is suggested with a vertical guideline, click to position the JTextField.

The JTextField snaps into position in the form aligned with the JLabel's baseline, as shown in the following illustration. Notice that the JLabel shifted downward slightly in order to align with the taller text field's baseline. As usual, a node representing the component is added to the Navigator window.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
AddTwoNumber
Source Packages
my.AddTwoNumberGUI
AddTwoNumberGUI.java
Libraries
Navigator
Start Page
AddTwoNumberGUI.java
Source Design History
Add Two Number
First Number
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
Spinner Separator Text Pane
[JFrame] - Properties
Properties Binding Events Code
Output - AddTwoNumber (run)
1:1 INS

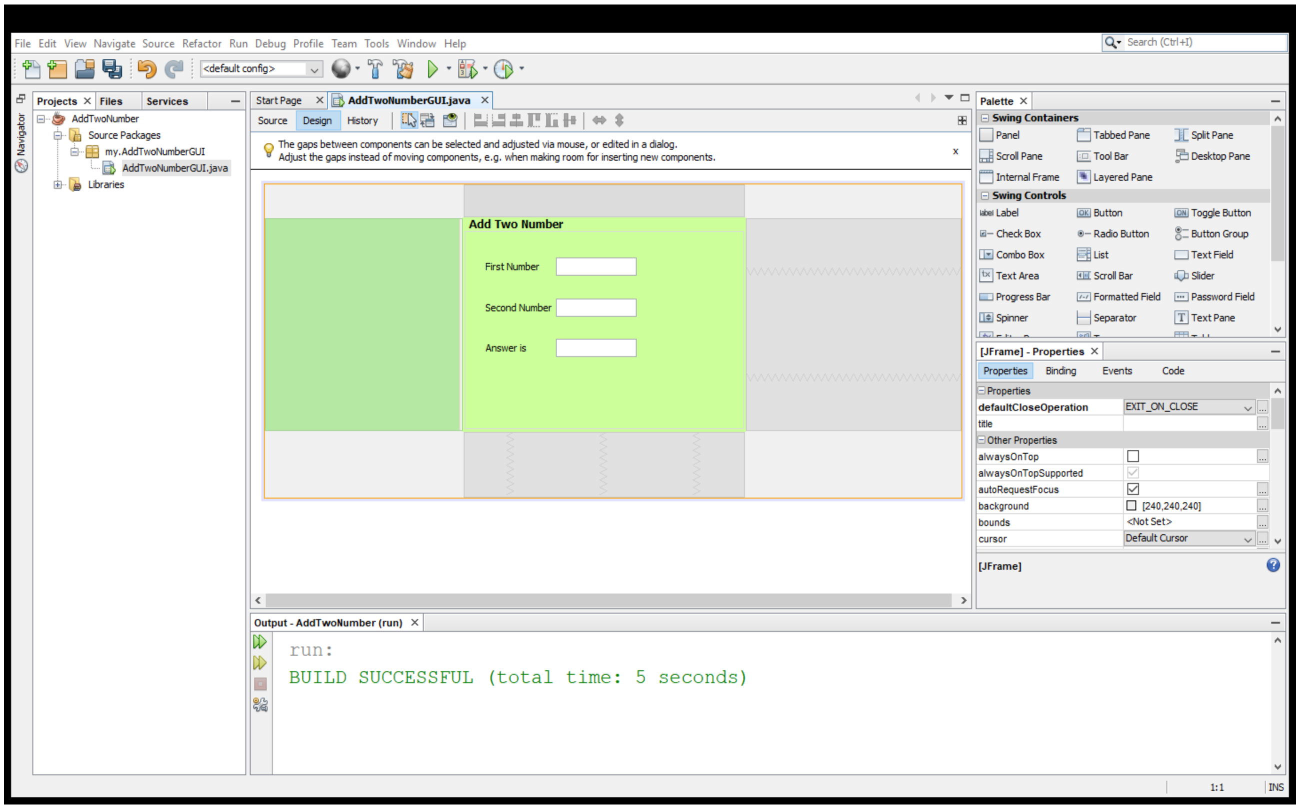
Select the root node in Navigator to access various useful settings of the form (in Properties).

Properties	
defaultCloseOperation	EXIT_ON_CLOSE
title	
Other Properties	
alwaysOnTop	☐
alwaysOnTopSupported	✓
autoRequestFocus	✓
background	[240,240,240]
bounds	<Not Set>
cursor	Default Cursor

[JFrame]

```
run:
BUILD SUCCESSFUL (total time: 5 seconds)
```

Repeat above process for as per your required Jlabel and JTextfield.

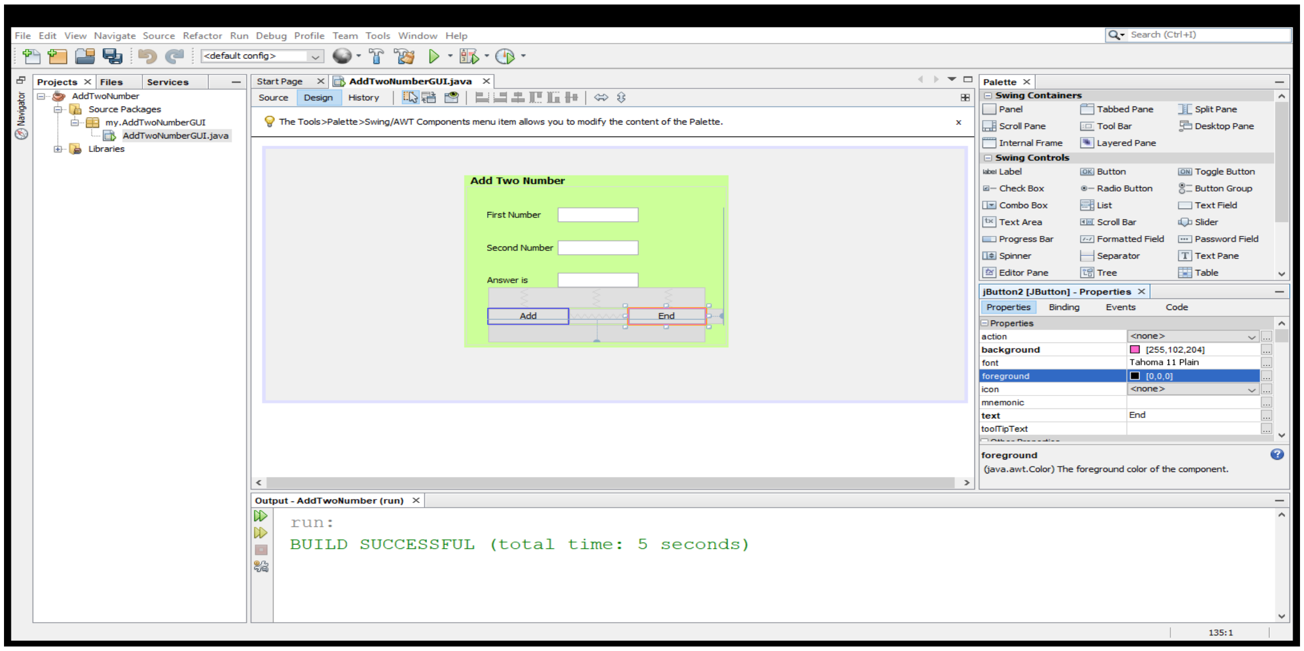

The gaps between components can be selected and adjusted via mouse, or edited in a dialog.
Adjust the gaps instead of moving components, e.g. when making room for inserting new components.

defaultCloseOperation	EXIT_ON_CLOSE
title	
alwaysOnTop	☐
alwaysOnTopSupported	✓
autoRequestFocus	✓
background	[240,240,240]
bounds	<Not Set>
cursor	Default Cursor

```
run:
BUILD SUCCESSFUL (total time: 5 seconds)
```

To add, align, and edit the display text of multiple buttons:

1. In the Palette window, select the Button component from the Swing Controls category.
2. Move the JButton over the right edge of the JPanel.

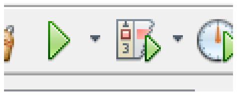

The Tools>Palette>Swing/AWT Components menu item allows you to modify the content of the Palette.

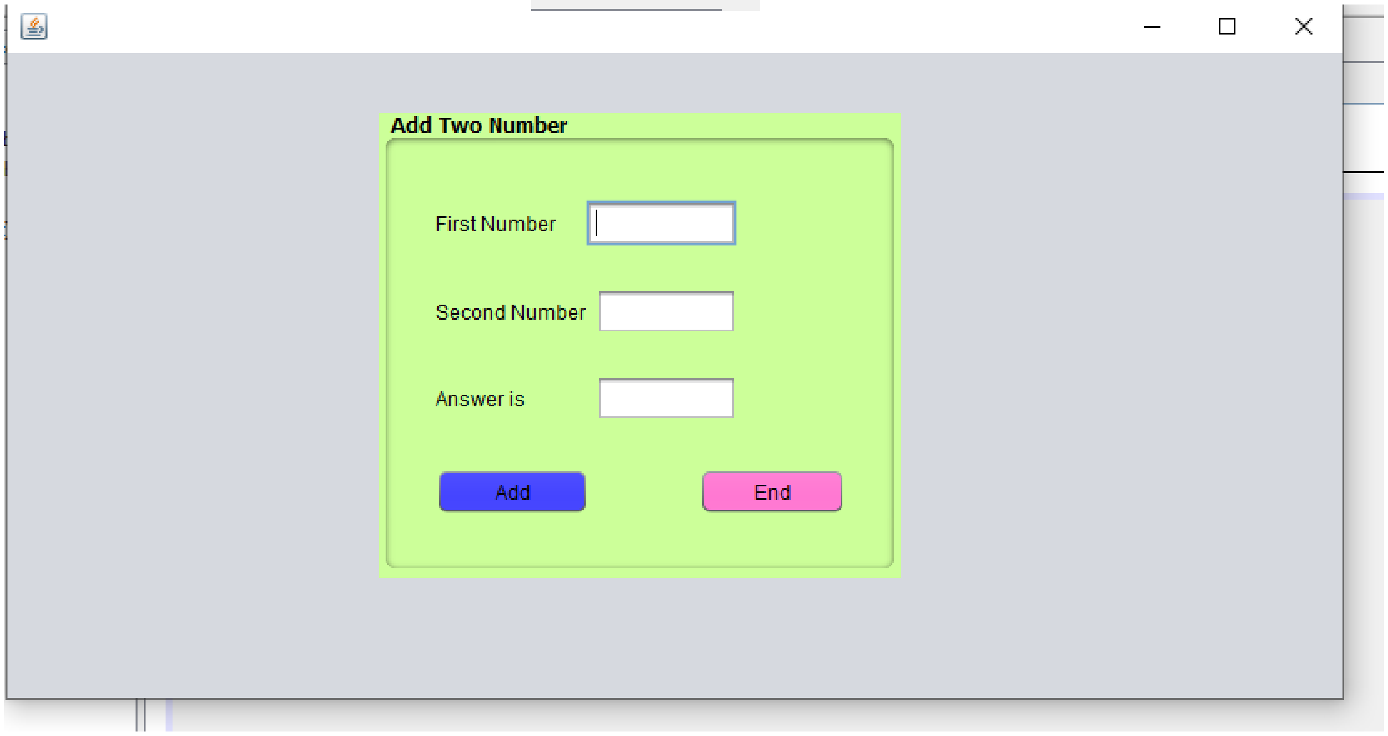
action	<none>
background	[255,102,204]
font	Tahoma 11 Plain
foreground	[0,0,0]
icon	<none>
mnemonic	
text	End
toolTipText	

foreground
(java.awt.Color) The foreground color of the component.

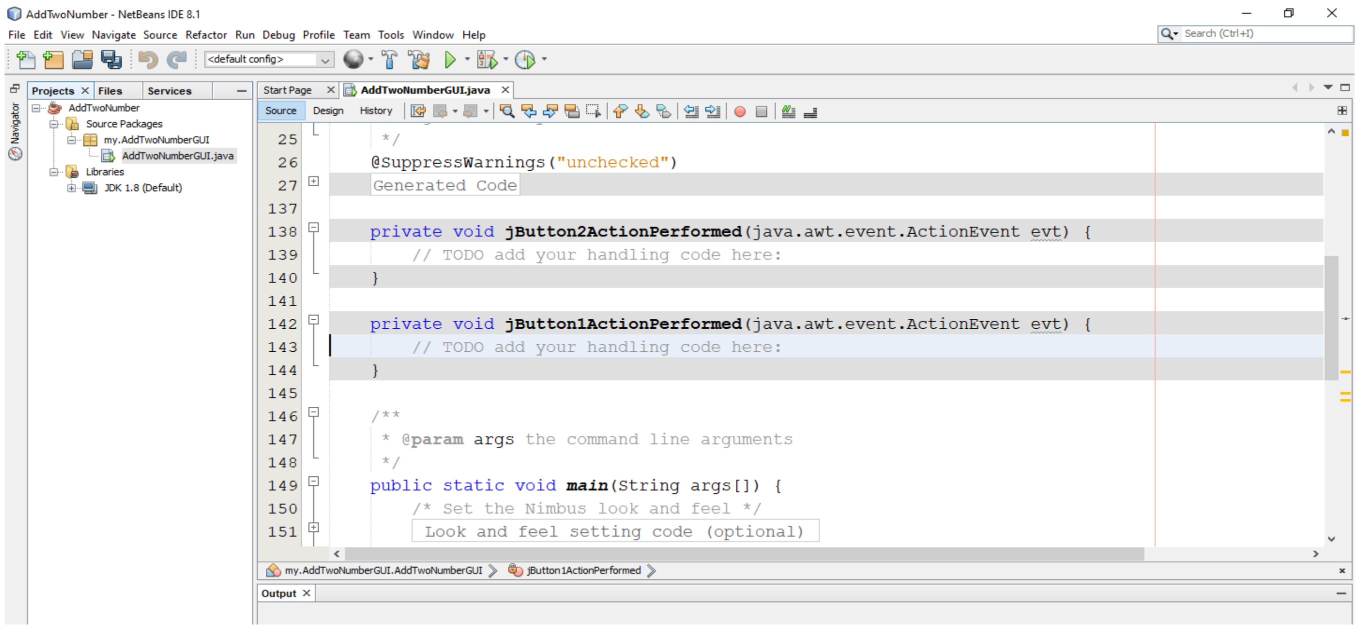
```
run:
BUILD SUCCESSFUL (total time: 5 seconds)
```

Run Application

To run your application, click Play button  where at tool bar then you will see



Now We write behind the code in ***Add*** and ***End*** Button, we also know event handling programming. We double click on add button then we will see as below



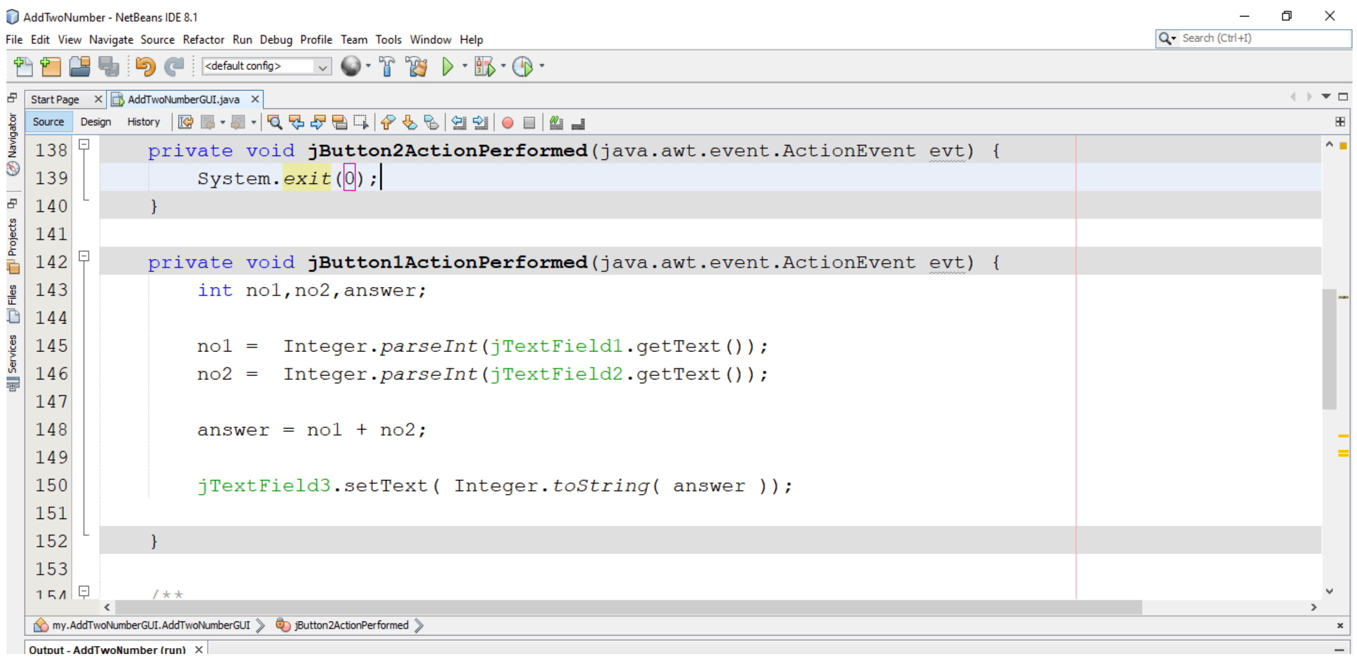
```
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)
```

What is an Event Handling Programming?

Change in the state of an object is known as Event, i.e., event describes the change in the state of the source. Events are generated as a result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

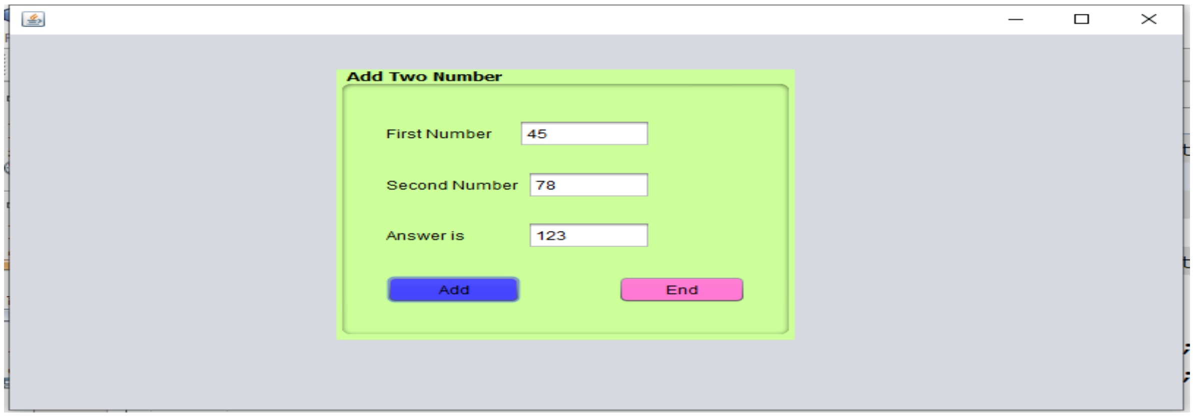
Here ***jButton1ActionPerformed()*** is ***Add*** button and ***jButton2ActionPerformed()*** is ***End*** button Event handling methods. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs. Java uses the Delegation Event Model to handle the events.

Now you double click on Add Button then write this code same as below.



```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        int no1,no2,answer;

        no1 =  Integer.parseInt(jTextField1.getText());
        no2 =  Integer.parseInt(jTextField2.getText());

        answer = no1 + no2;

        jTextField3.setText( Integer.toString( answer ));

    }

    /**
```

After write code you run this window form application and will be work both ***Add*** and ***End*** Button successfully same as below.



This is a simple example for window form application with event handling programming here start from new world of window form designing with event handling programming.