The University of the West Indies, St. Augustine
COMP 2603 Object Oriented Programming 1

Lab - Week 7

In this lab, we will explore Graphical User Interfaces (GUIs). We will use a new editor, Netbeans, to create a GUI and adjust the design aspects of the interface. The first section introduces the Netbeans editor. The second section focuses on the design and layout of a GUI. The third section focuses on adding functionality to the GUI through Action Listeners.

Download link for the Netbeans IDE (Java SE): https://netbeans.org/downloads/

Part 1: Creating a Java Application using Netbeans

1. Open the Netbeans IDE and choose File > New Project.

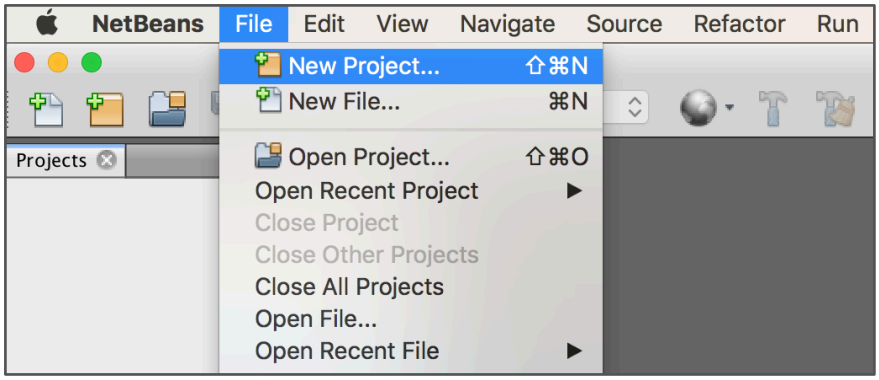
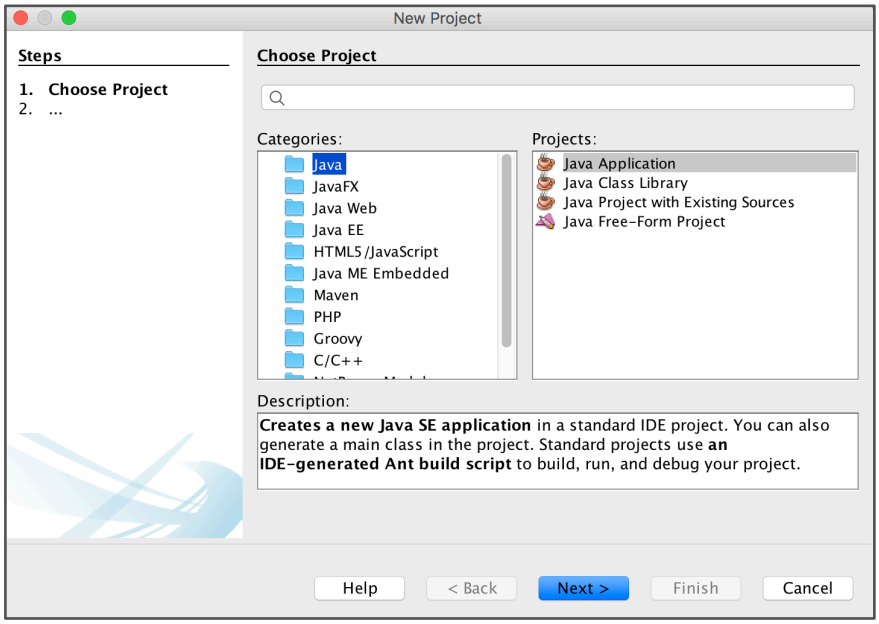


Figure 1. Creating a New Project From The File Menu

2. Create a new **Java Application**. Click Next.



3. Set your project's name as **StudentPortal**. Set the Project location on the **Desktop**. Ensure that the *Create Main Class* option is selected. Click Finish.

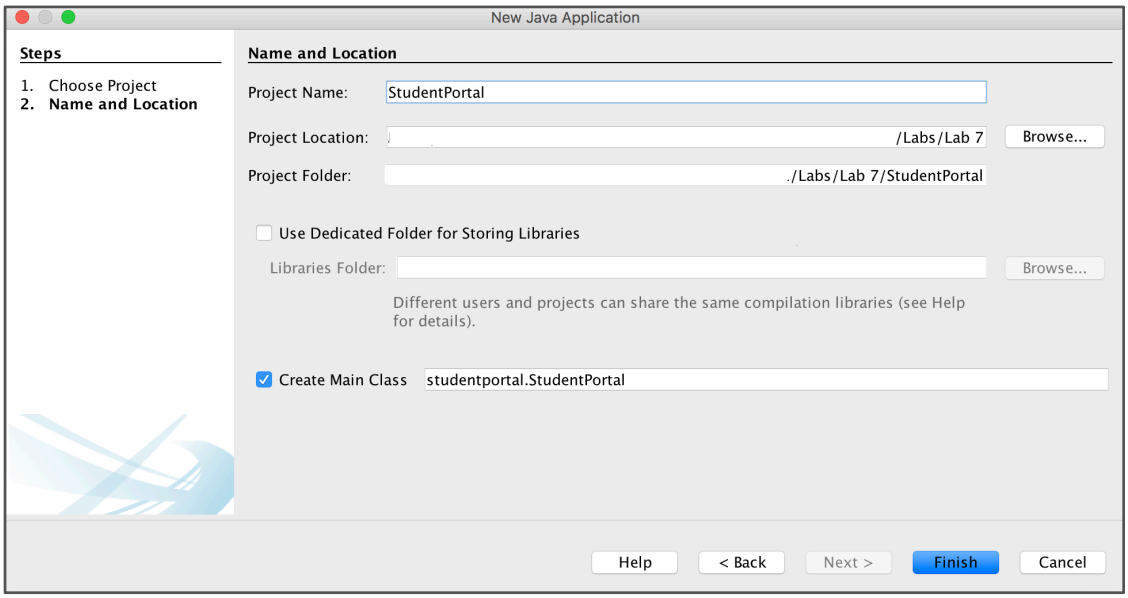


Figure 3. Naming a Project And Setting Where it is Saved

4. Expand the project's file tree in the Navigator (left pane) by clicking the grey arrow ▼ Do you notice the main class, StudentPortal?
   Right click the **studentportal** package and choose New > **JFrame Form**.

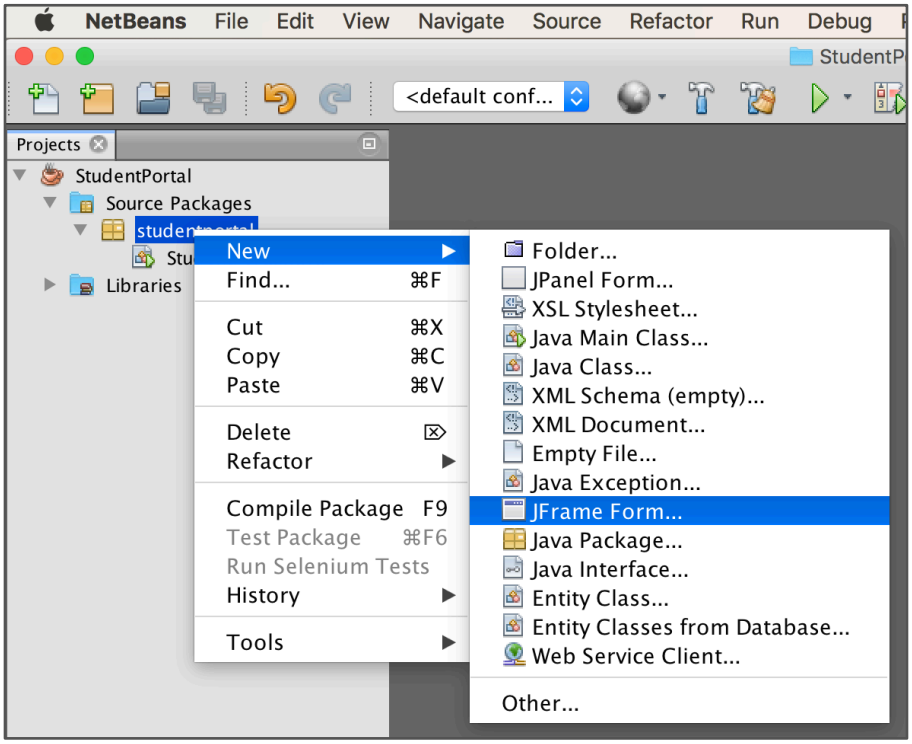


Figure 4. Adding a New File to a Project in Netbeans

5. Set the name of the new JFrame Form to **RegistrationGUI**

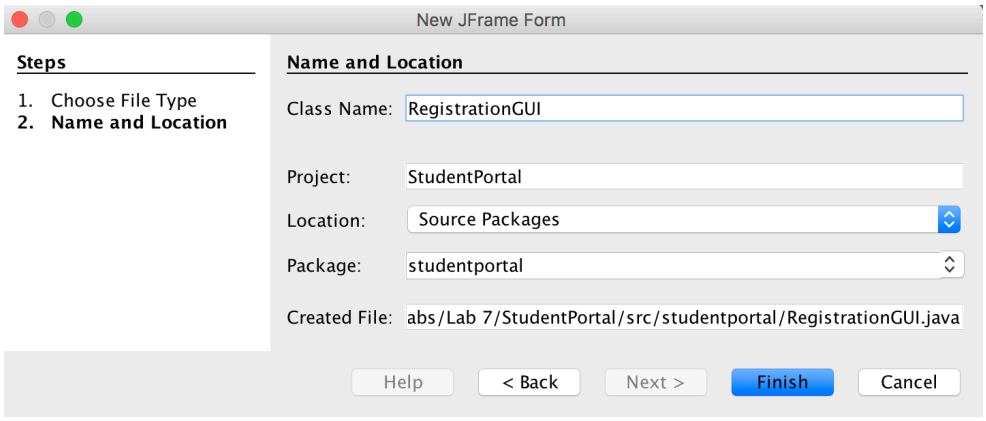


Figure 5. Setting the Name of a new File in a Project

6. The IDE opens the **RegistrationGUI** form in the **GUI Builder Interface** in the **Design View**. The Design View allows us to build Java GUIs by selecting elements from the **Palette**, dragging and dropping them on the canvas in the **Design Area**. We can then set various aspects of these elements in the **Properties Window**. If we have multiple forms, these are listed in the Project's file tree in the Navigator pane.

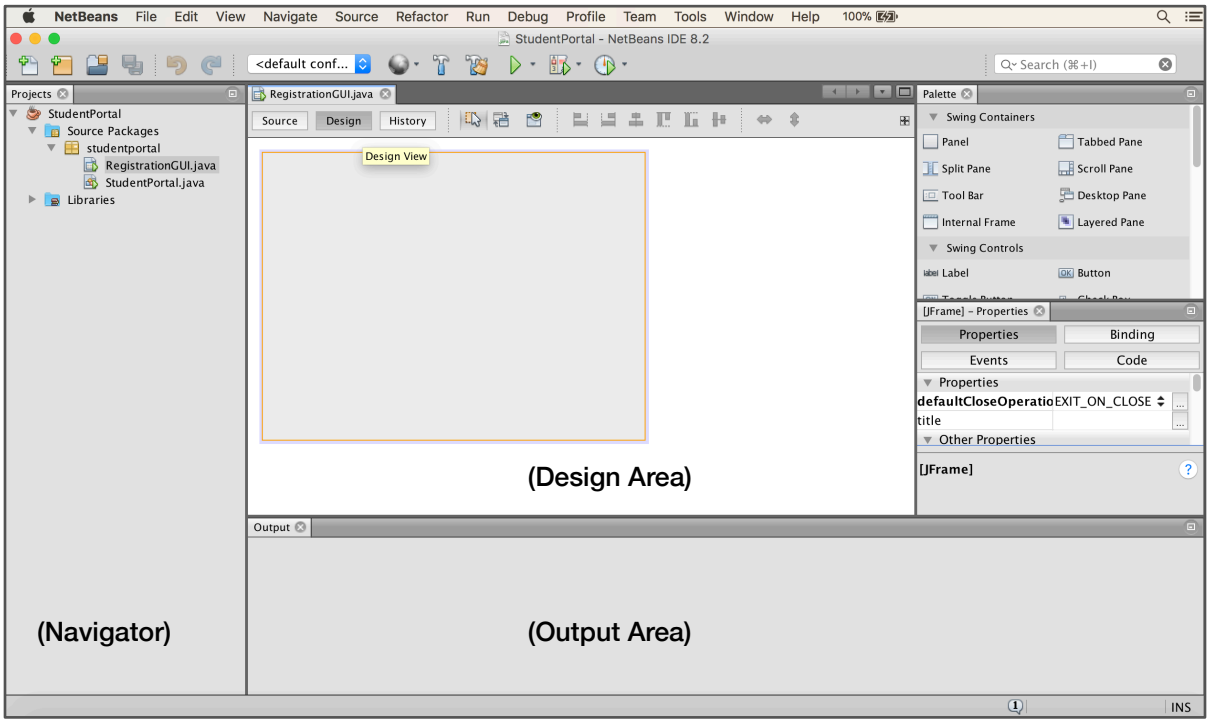


Figure 6. The Design View of the GUI Builder Interface in Netbeans

Good Resource for Learning how to use Netbeans for GUI creation:
https://netbeans.org/kb/docs/java/quickstart-gui.html#getting_familiar

7. Click on the *Source* button to switch to the **Source View**. In the background, Netbeans translates all of our design decisions into a functional user interface implemented in the RegistrationGUI class using the new GroupLayout layout manager and other Swing constructs.

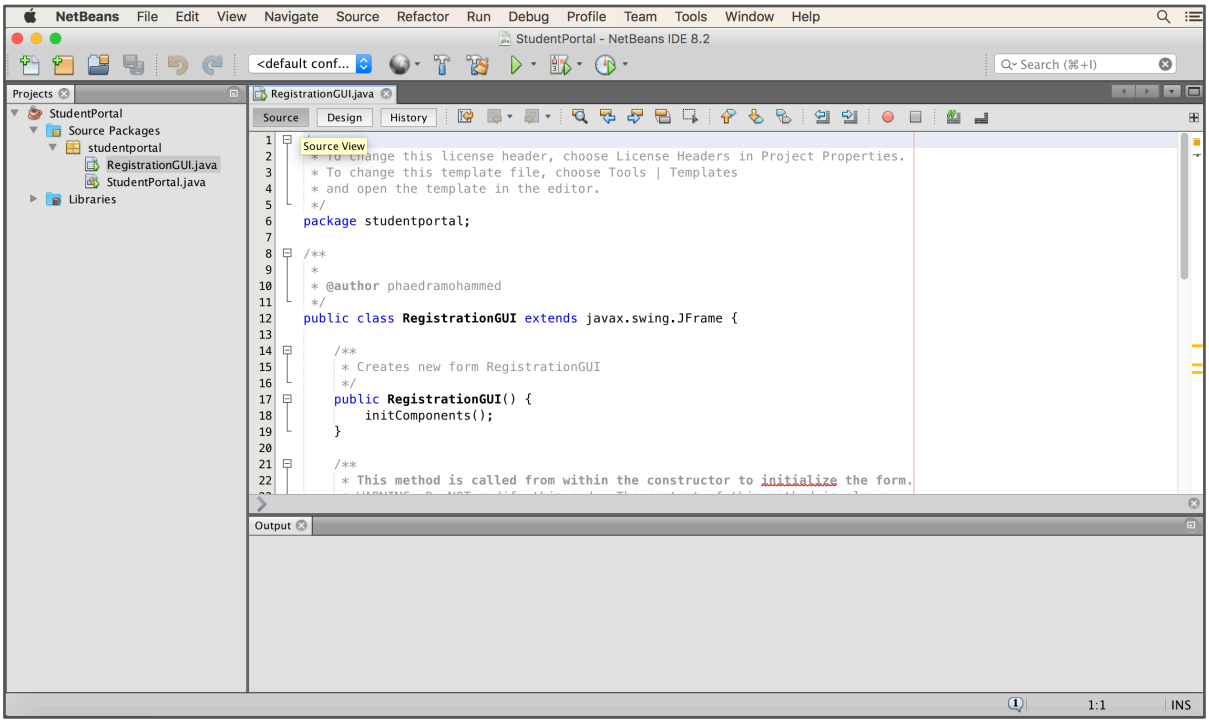
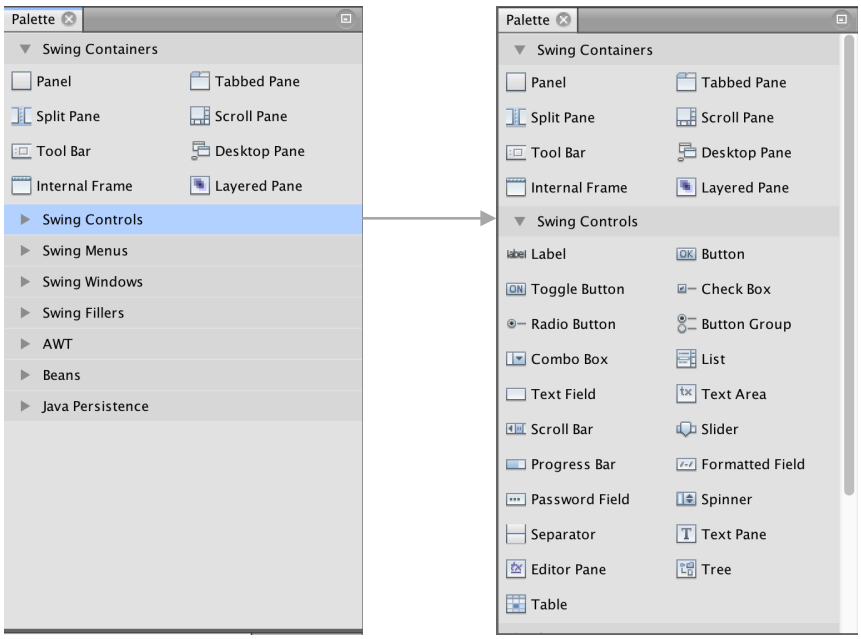


Figure 7. The Source View of the GUI Builder Interface in Netbeans

The Netbeans editor releases developers from the complexities of creating graphical interfaces using the GUI Builder. As we add components to our **RegistrationGUI** form, code is automatically generated in the **RegistrationGUI** class based on where we position each component. Visual feedback is provided in the Design View to help us align components along guidelines for an aesthetic, polished final look and feel.

The Palette
Swing components:
- Containers
- Controls
- Menus



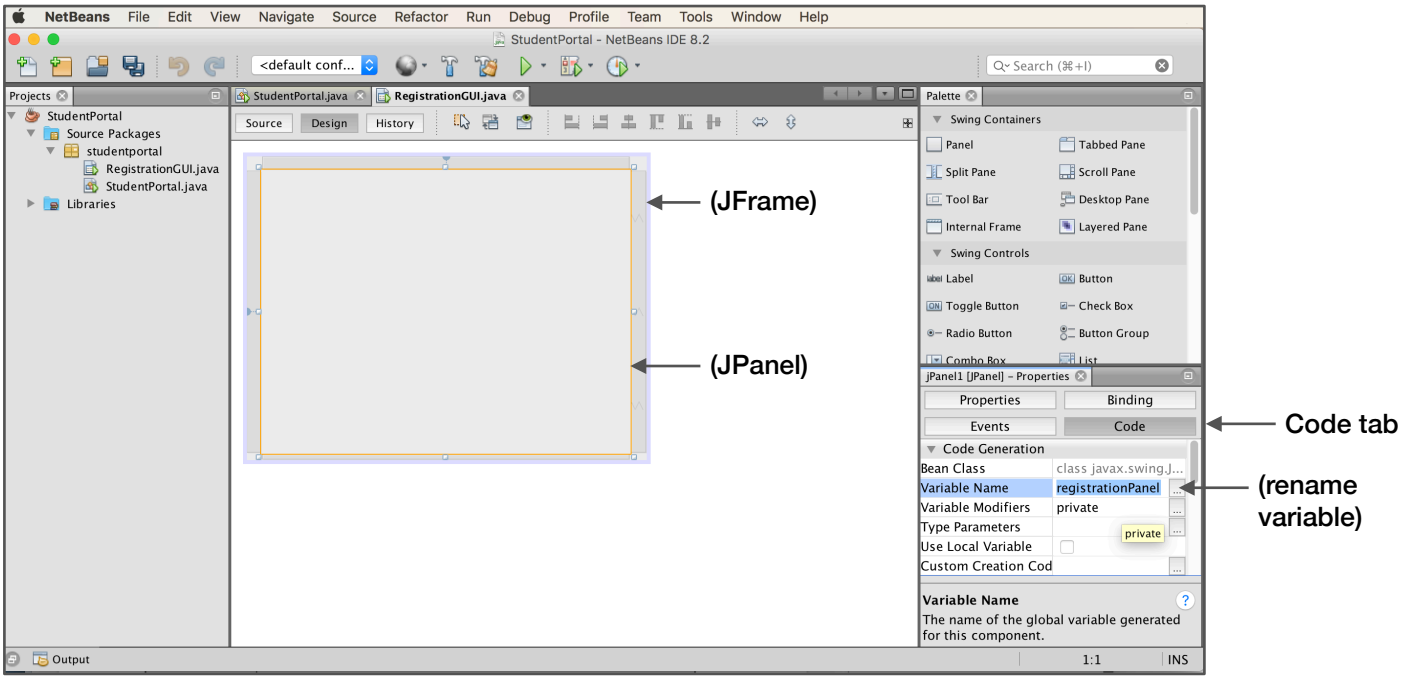
Expanded view of the Swing Controls components

Part 2: Adding GUI components using the Netbeans GUI Builder

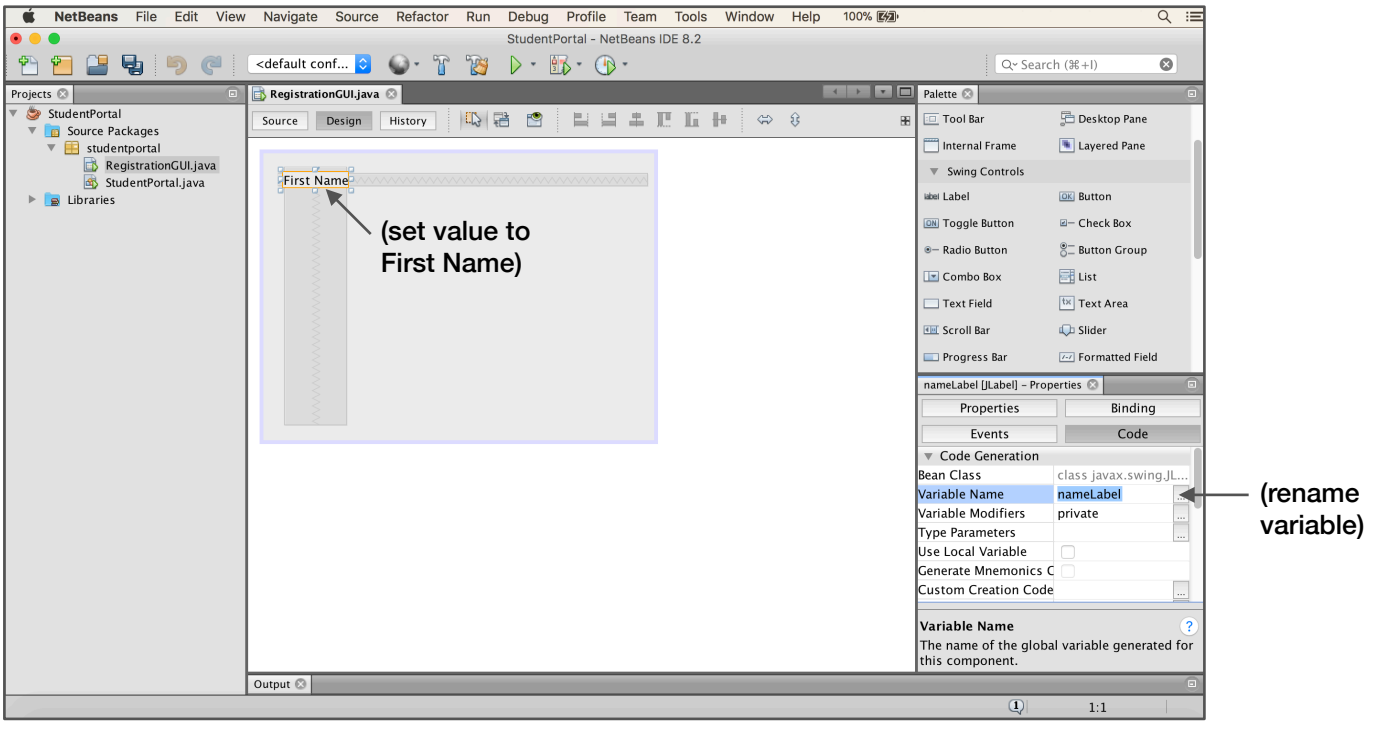
TIP: Restoring Views in GUI Builder if closed accidentally

Window > IDE Tools

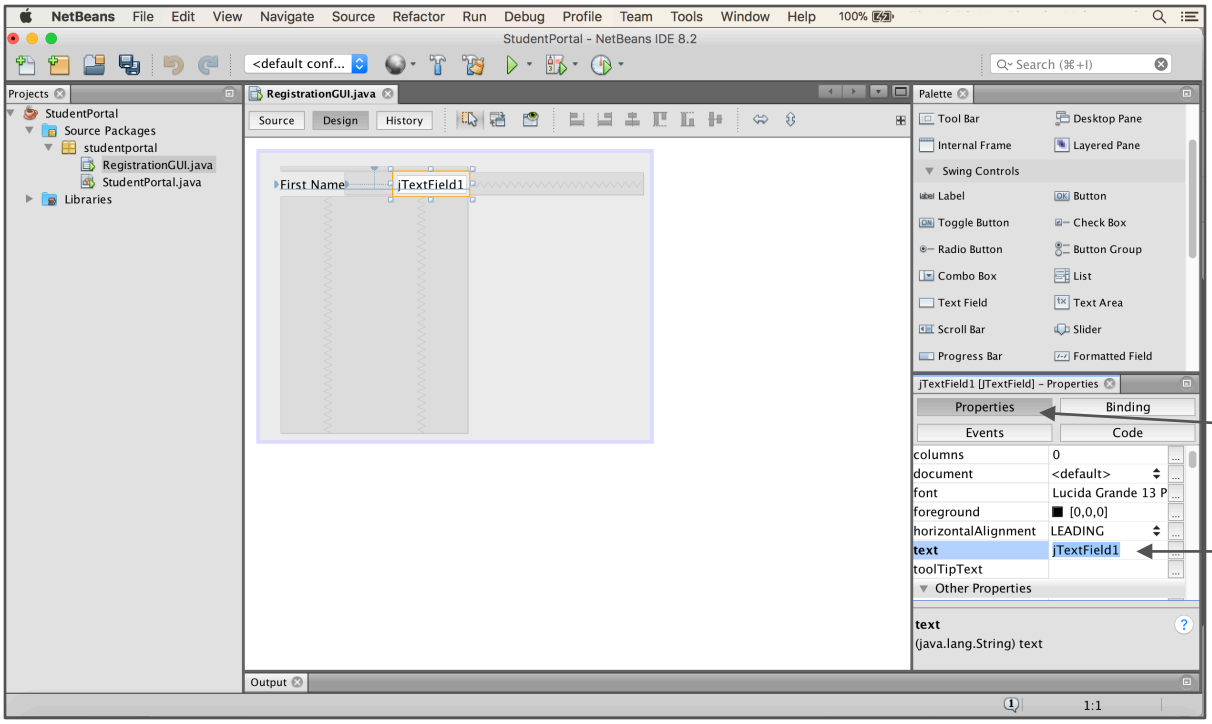
1. The **JFrame** is the top-level container. Let's add a **JPanel** to the **JFrame**. The **JPanel** will be used to cluster upcoming components in neat areas on the form.
   - Ensure that you are in Design View.
   - Select a new **JPanel** from the *Swing Containers* in the Palette.
   - Drag it over to the JFrame and release.
   - Resize it to fill most of the JFrame
   - Change the variable name of the **JPanel** object to **registrationPanel**.



2. Select a new **JLabel** component from the *Swing Controls* in the Palette.
   - Add it to the form and set the value to First Name.
   - Change the variable name of the **JLabel** object to **nameLabel**.



3. Select a new **JTextField** component from the *Swing Controls* in the Palette.
   - Add it to the form and delete the default variable value (JTextField1).
   - Change the variable name of the **JTextField** object to **nameField**.



4. Add components to the form to create the design shown below. Variable names follow:
   - Second **JLabel** object variable name: **lastNameLabel**
   - Second **JTextField** object variable name: **lastNameField**.
   - First Button object variable name: **clearButton**
   - Second Button object variable name: **registerButton**
   - JTextArea object variable name: **displayArea**

5. Switch to the **Source View**. Scroll through the **RegistrationGUI** class and observe the code generated. Where do you notice the names of the GUI components? Where are the components initialised?

TIP: Viewing the full details of the code in a class

Answer:

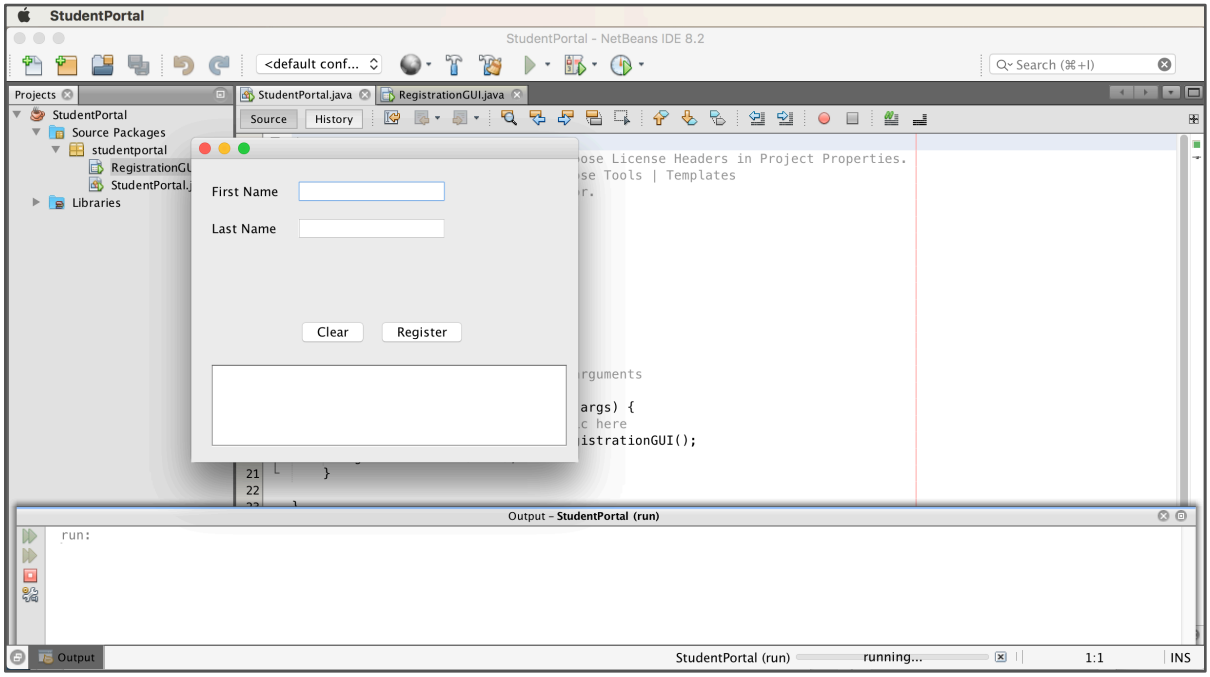
Click on the plus symbol to expand code

6. Let's put in the code to generate a GUI instance in the main class of the project. Switch to the **StudentPortal** class. Type the following code in the main method of the **StudentPortal** class:

```
RegistrationGUI gui = new RegistrationGUI();
gui.setVisible(true);
```

7. From the **StudentPortal** class, run the Project by clicking on the green arrow.

Observe the GUI generated.



8. Try typing some data in the text fields and the text area. Test the buttons. Is anything useful happening when you press the buttons? Why not?

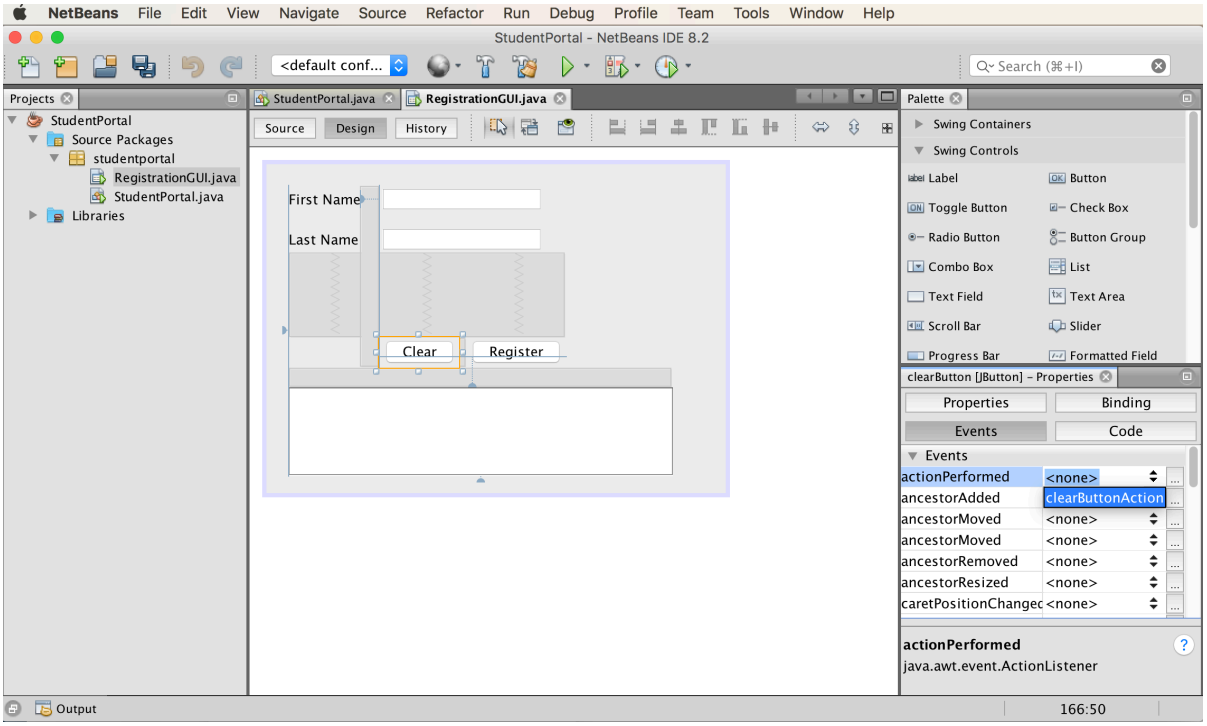
Answer:

9. What should the Clear button do? What is necessary in order to make the Clear button work? Which components will need to be accessed by the Clear button?

Answer:

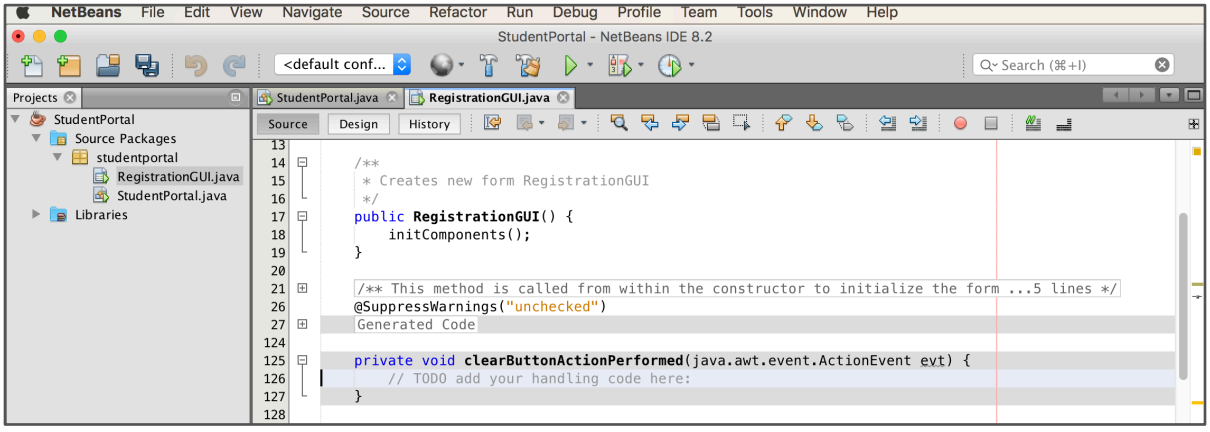
Part 3: Adding Functionality to GUI components

1. Switch to **Design View** in the **RegistrationGUI**.
    - Select the **Clear** button in the Design Area
    - Click on the **Events** tab in the Properties Window.
    - Click the drop-down arrow and select **clearButtonActionPerformed** from the options.

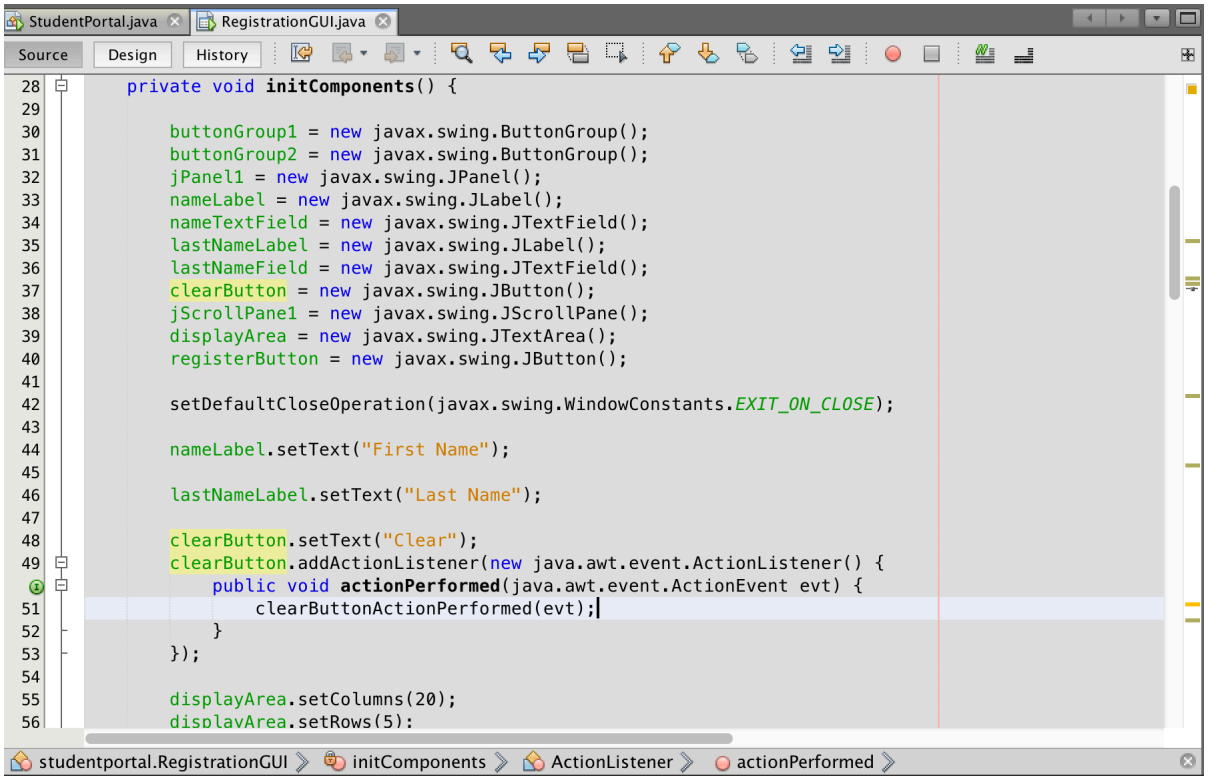


2. The IDE switched to the Source view of the **RegistrationGUI** class automatically after Step 10. Observe the new method that has been inserted:

```
private void clearButtonActionPerformed(…){
}
```



3. Expand the **initComponents( )** method in the **RegistrationGUI**.



What has changed in the code compared to what you observed from Part 2, Step 5? The **clearButton** has some additional code. Explain what the code does.

Answer:

4. Let's make the Clear button a bit more useful. Add the following code to the **clearButtonActionPerformed(..)** method in the **RegistrationGUI**.

```
nameTextField.setText("");
```

5. Run the Project and test whether the **Clear** button works to clear the form. What did it do? Is the button fully functional?

Answer:

6. Add the necessary code to the **clearButtonActionPerformed(..)** method in the **RegistrationGUI** to make the Clear button fully functional. The button should clear the data from both textfields and the text area.

7. Add code to make the textArea non-editable by users.

8. Perform the necessary steps and add code to make the **Register** button work as follows:
    - Capture the first and last names from the text fields
    - Clear the text fields and display a message in the text area:
      Registered: <first name> <last name>

Additional Exercise

Make the **Register** button work as follows:

- Keep a record of the first and last names of every student registered using the GUI.
- Display the messages shown in the figures below when the Register button is clicked for the first time, when a student has been registered successfully, and when registration is full.
- The clear button should work as before (clearing data from all fields)