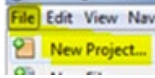





Lab 1: Basics of Java and Graphical interface of Netbeans

The aim of this lab session is to familiarise with syntax of Java and basics of Graphical User-Interface in Netbeans.

Understanding the lab sheet....

	The key features of explanation are highlighted by yellow marker.		Testing tasks
	Compulsory tasks. Requirements for D grade.		Style guide hints

Preparation to the lab work

Complete the following Quizzes, meeting LO1:

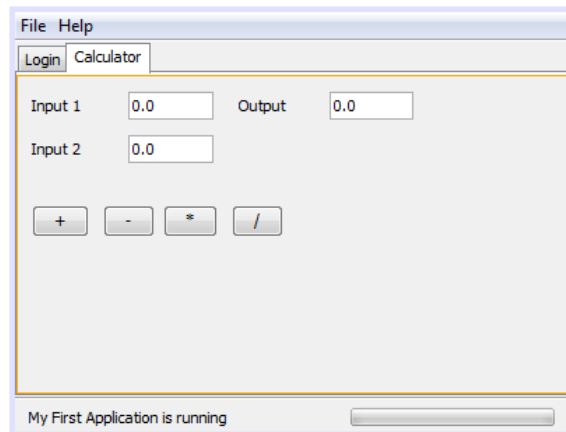
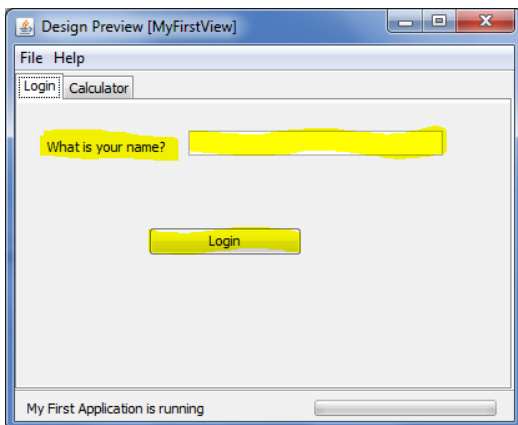
- Week 1-1: Java Basics – loops
- Week 1-2: Java Basics – methods
- Week 1-3: Introduction to Java

In order to pass the coursework, you must achieve a score of 50% or higher on the associated Blackboard quizzes.

Testing of the application created during lab 1:

What will you achieve at the end of lab?

During lab session you will be able to create windows-based application to login into the system and implement simple calculator. At the end of the lab session your developed application may look like this:



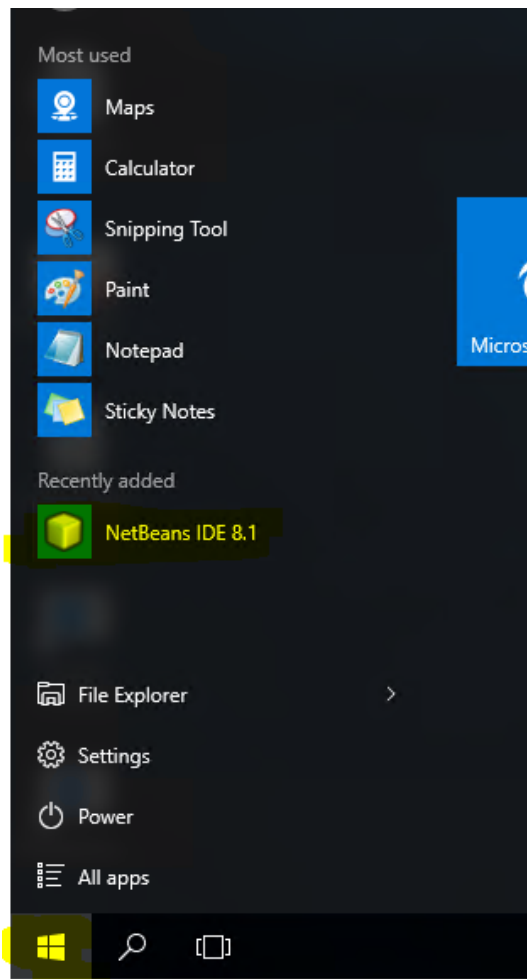
The main skills developed during completion of this lab:

1. Familiarise with programming environment of NETBEANS
2. Independent search for the library methods
3. Skills to re-use methods previously declared and implemented

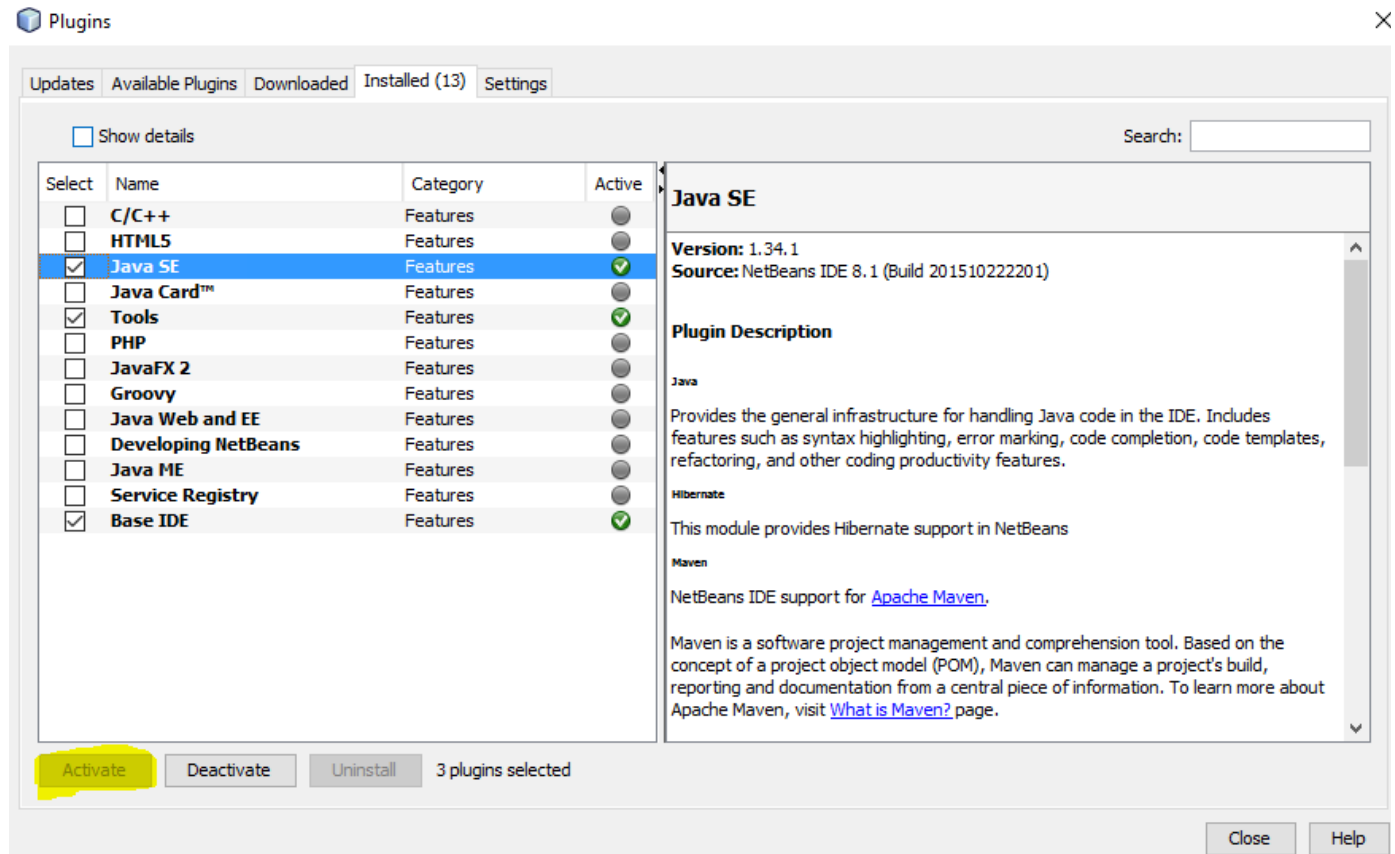
Let's start...

Start NetBeans:

Start Windows -> Netbeans IDE 8.2



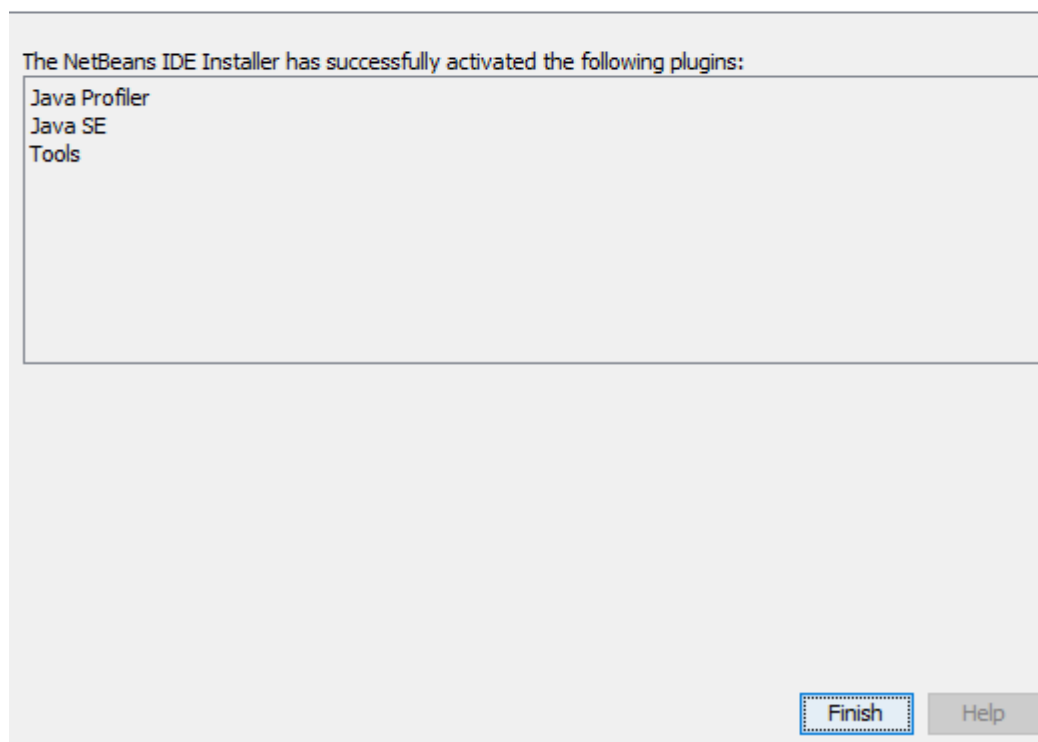
Click "Install plugins", and select the following plugins by clicking "Activate" button:



NetBeans IDE Installer

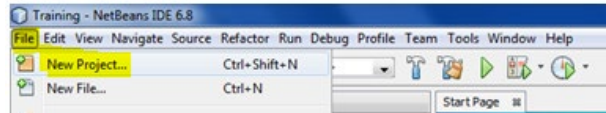
Activation completed successfully

Click Finish to quit the NetBeans IDE installer.

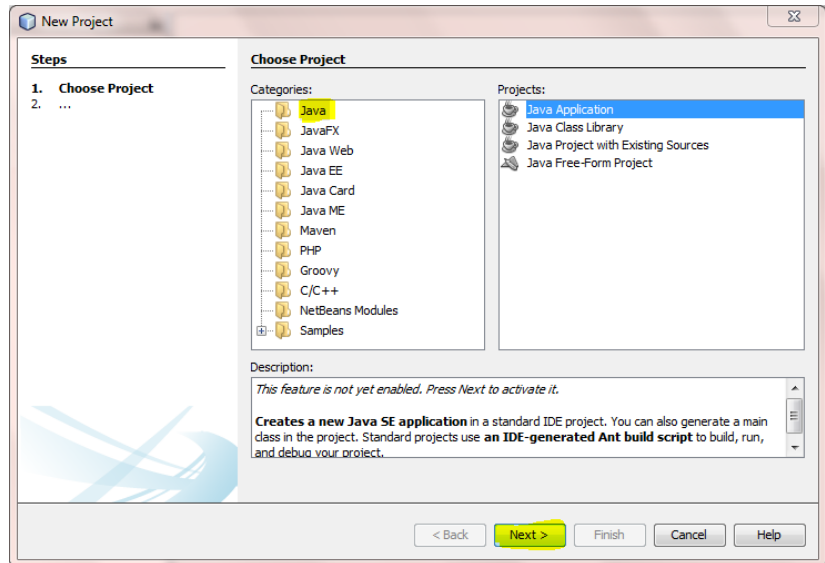


Create a new project:

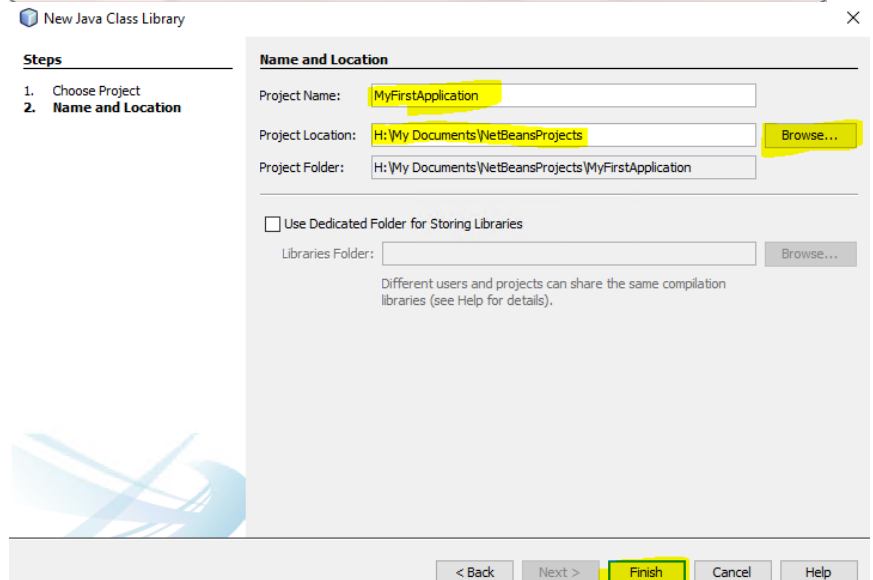
1. Click in the Menu: **File** -> **New Project...**



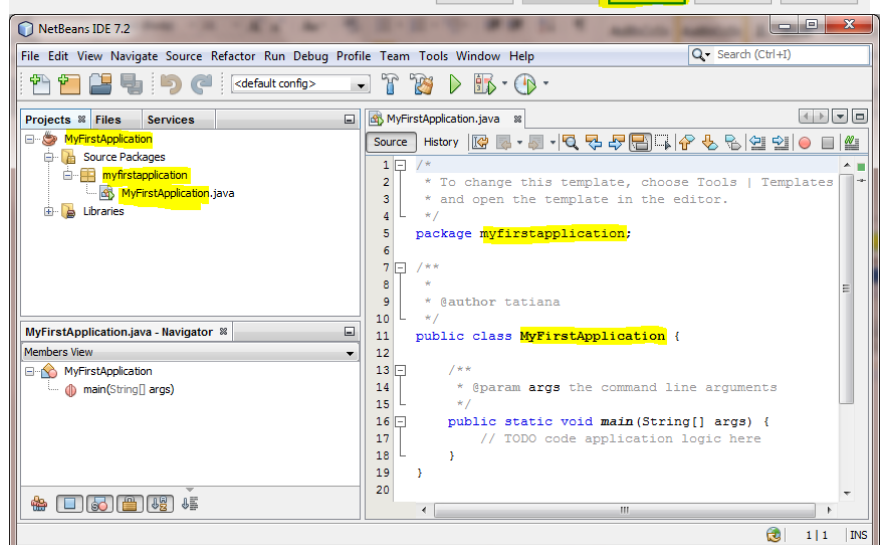
2. New Project window will appear. Select **Java** -> **Java MyApplication** and click **Next**.



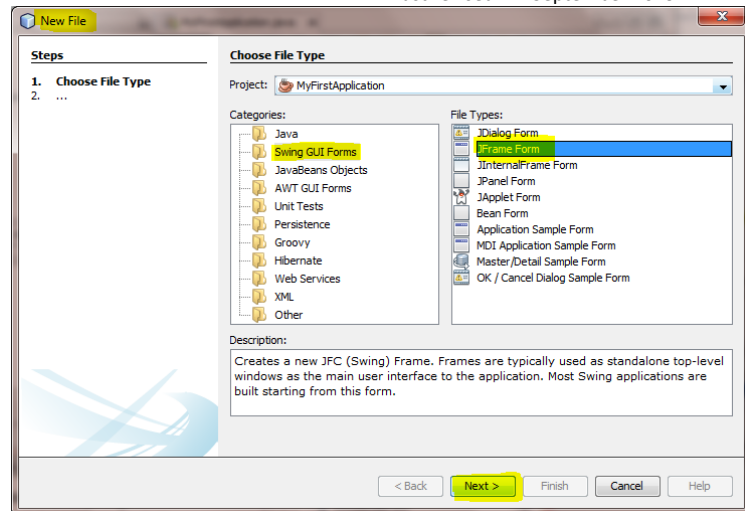
3. In New Desktop Application window, type the **Project Name** and specify **Project Location** using **Browse** button. Click button **Finish**.



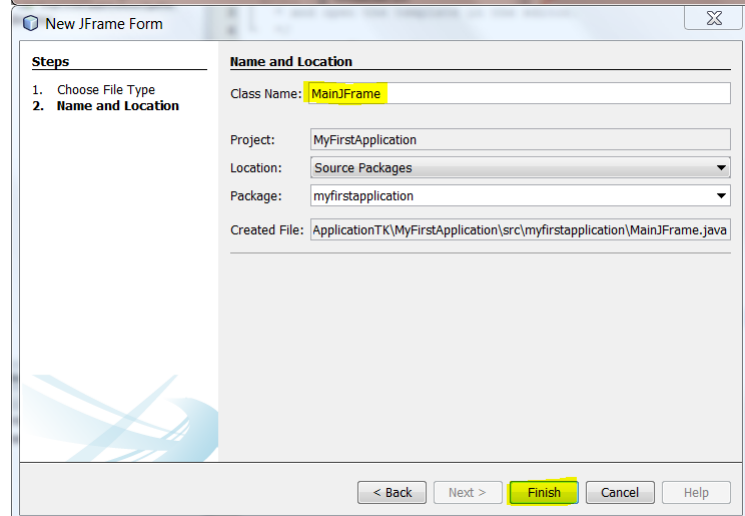
4. The default project is created as shown at left. The initial code is generated automatically.



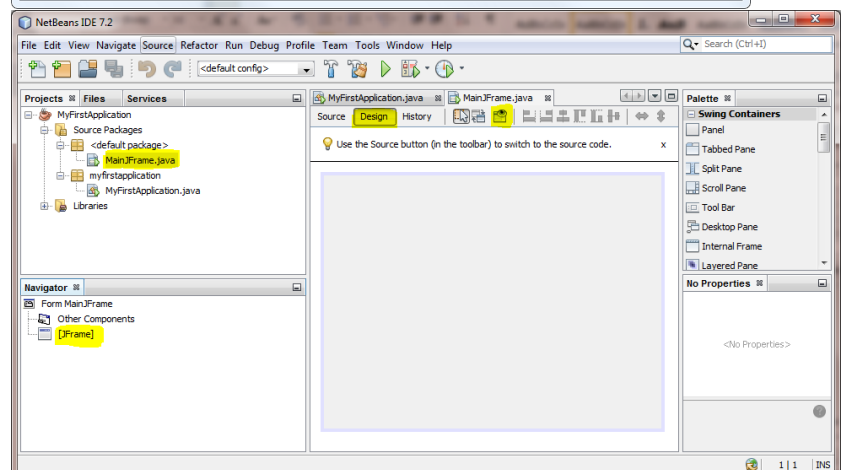
5. The windows-based application is build using JFrames. In order to add JFrame Form to the project, click **File->New File**. Click **Next**, once the **JFrame Form** is selected from **Swing GUI Forms**.



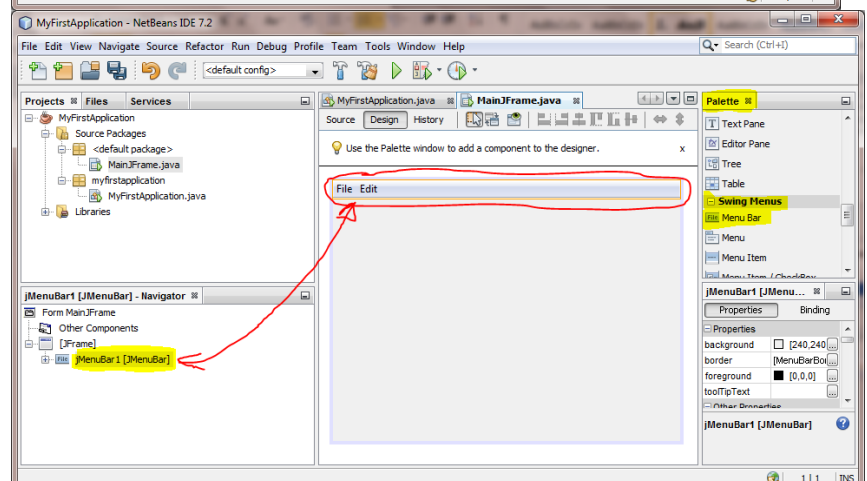
6. Specify the name of the **JFrame**, such as **MainJFrame**. It is good practice to leave the **JFrame** part of the class and add the main purpose of the frame.



7. The **JFrame** can be viewed using Design view. The Navigator shows the elements that belong to the **MainJFrame**.



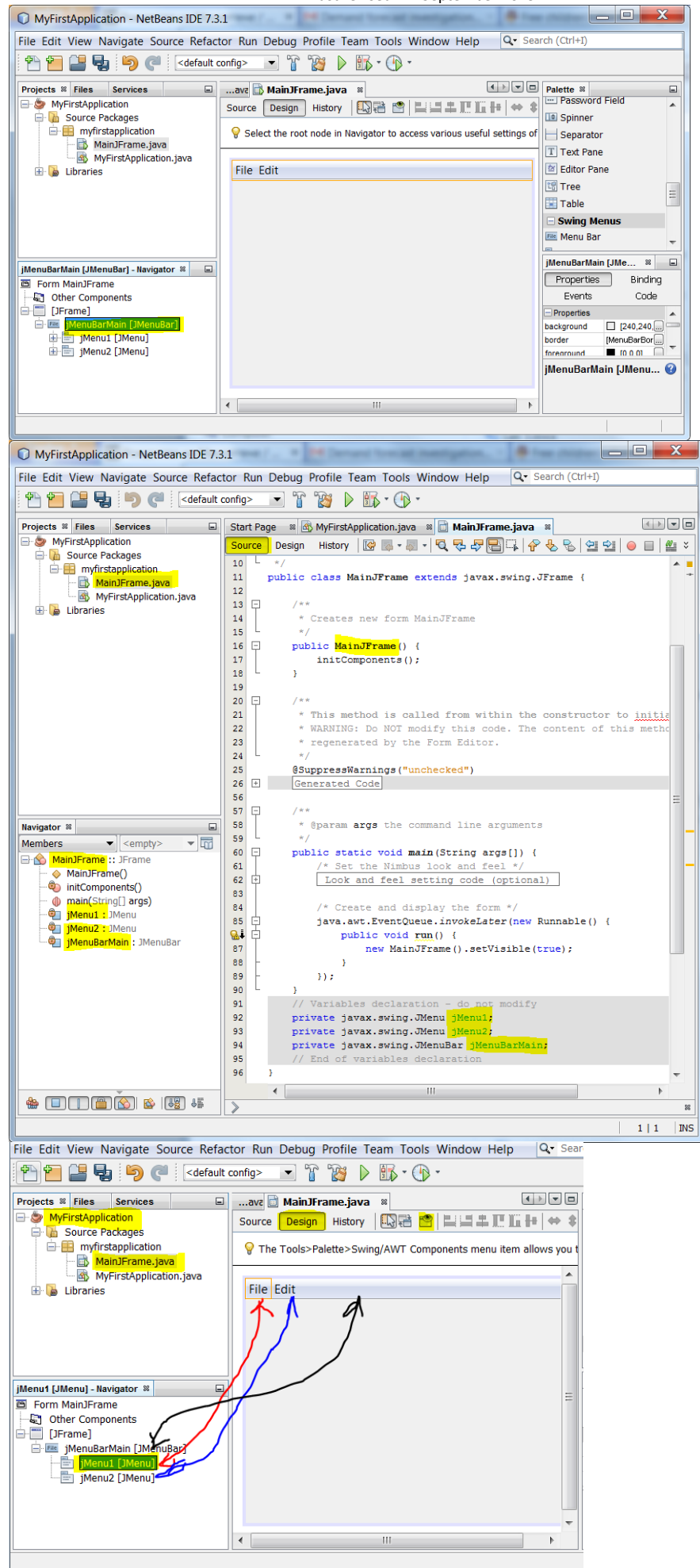
8. In order to add the menu to the frame, in **Palette**, locate **Swing Menus**, Drag and Drop **Menu Bar** to the **JFrame**. Make sure that the design view is activated. Once the menu is added to the **JFrame**, its description is shown in the **Navigator**. **JMenuBar** is the type of menu (name of class). **JMenuBar1** is the name of variable (or object).



9. In order to change name of variable type of **JMenuBar**, select **JMenuBar1[jMenuBar]** in **Navigator**, right click the mouse and select the top option of pop out menu: **"Change Variable Name..."**. Change the name of variable to **jMenuBarMain**. It is good programming practice to name the elements of graphical interface according to their purpose, clearly specifying the types of these elements.

10. Click **Source**. The program code generated automatically is shown. Class **MainJFrame** is declared. Objects (variables) **jMenuBarMain** type of **JMenuBar** (class from the Swing library), **jMenu1** and **jMenu2** type of **JMenu** have been declared automatically. The names of objects correspond to one defined in the graphical interface. Note the names of classes are started with upper case **J**, the names of objects are started with lower case **j**.

11. Consider the figure right. Identify 2 additional graphical elements (shown in blue and black). Identify the type of elements and their names.

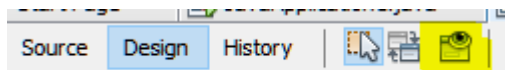


MyFirstApplication project is created as shown in **Projects** window:

1. **MyFirstApplication.java** contains the program code for overall project and contains MyFirstApplication class.
2. **MyJFrame.java** contains the code about actual window, where the GUI for actual program will be implemented, and contains MyJFrame class

Navigator window displays a tree hierarchy of all components contained in the currently opened form. Displayed items include visual components and containers, such as buttons, labels, menus, and panels, as well as non-visual components such as timers and data sources. The current project contains 3 components:

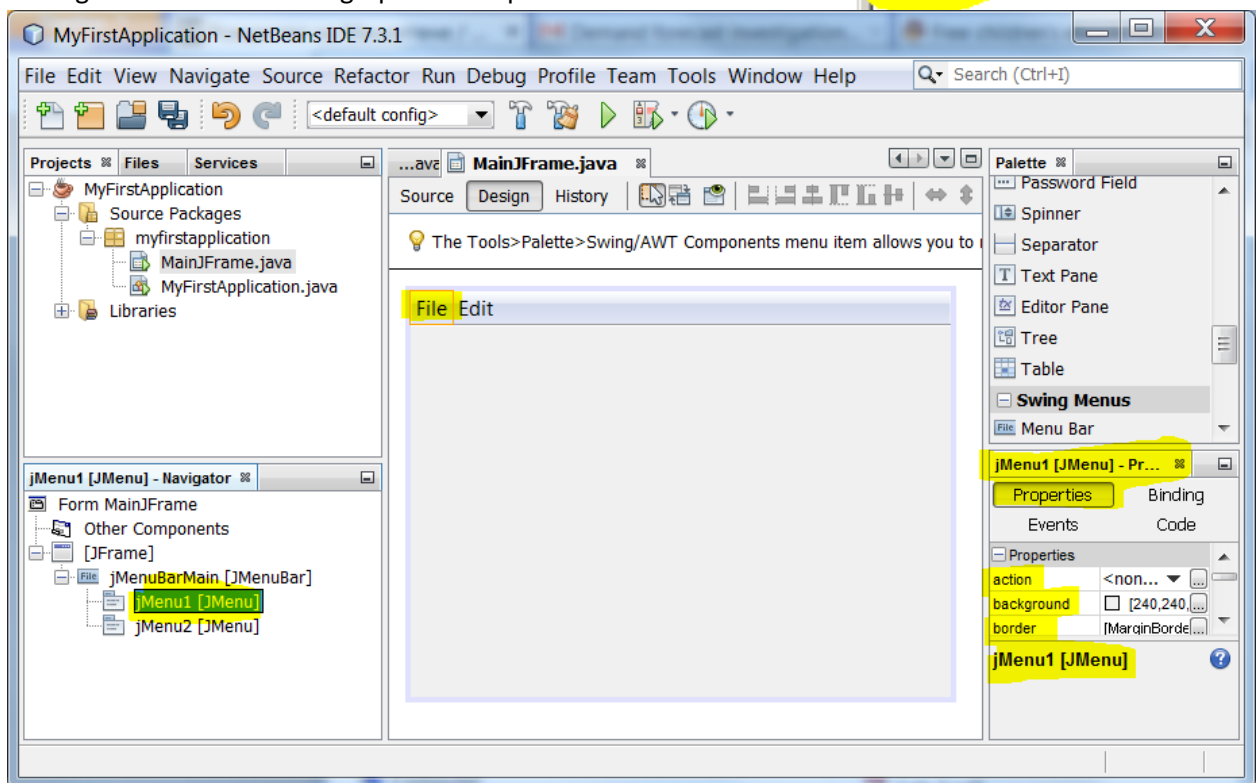
1. **jMenuBarMain** type of **JMenuBar**, shown in Black.
2. **jMenu1** type of **JMenu**, shown in Red.
3. **jMenu2** type of **JMenu**, shown in Blue.



Preview Design shows the view of interface after compilation.

Source view shows the program code related to the chosen .java file.

Design view shows the graphical representation of the project and it allows to change the features of the graphical components in the frame.



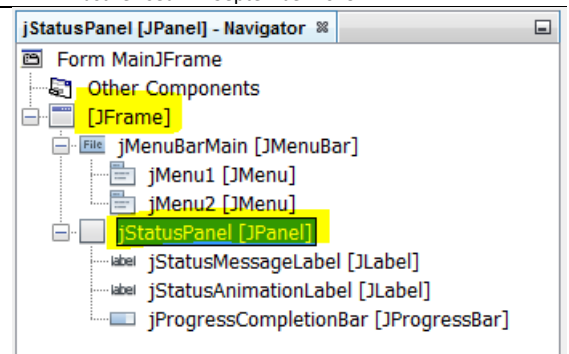
The specific component in the Graphical Interface can be edited using **Properties** window. The **Properties** window displays the property sheet for the currently selected node in the Projects window. Let us consider **jMenu1** type of **JMenu**: In **Navigator**, select the **jMenu1[JMenu]**. The selected item is highlighted in the form using orange box. The features of this panel are shown in the Properties, located in the right bottom corner of the screen.

To open the **Properties** window, choose **jMenu1[JMenu]** object, right click button in mouse and choose **Window > Properties**.



Task: Explore the view of the project. Document the outcome of each step in the report.

1. Click **Preview Design** button.
2. Click **Source view** tab.
3. Modify **border** and **foreground** properties of **JMenuBarMain**.
4. Add **jStatusPanel**. Inside of **jStatusPanel** add the following elements: **jStatusMessageLabel** type of **JLabel**; and **jProgressCompletionBar** type of **JProgressBar**. If the task is completed correctly the **Navigator** will display all elements included into the **jStatusPanel**.
5. Change the **background** feature of **jStatusPanel**, using **Properties**. Use **Preview Design** button to view the compiled view of the frame when the properties of Graphical Interface have been changed.
6. Modify **border** and **font** properties of **jStatusPanel**.
7. Expand **jStatusPanel** as shown below and change the properties of **statusMessageLabel[JLabel]**: **font** and **text** to "My First Application is running."



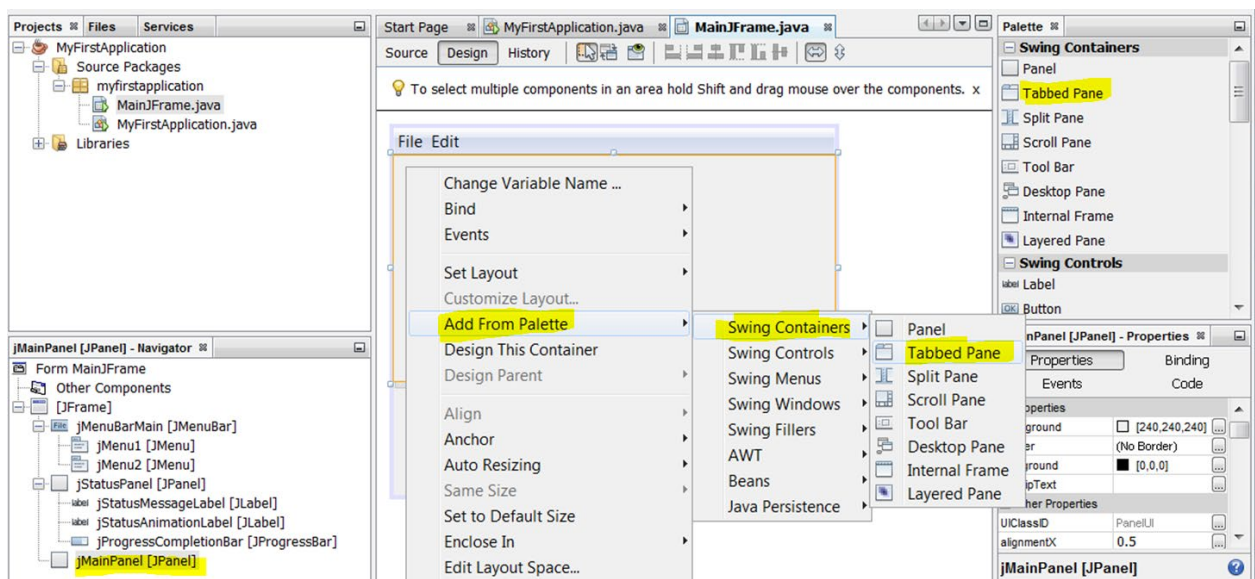
Implementing the GUI using Swing elements

Creating the tabbed control

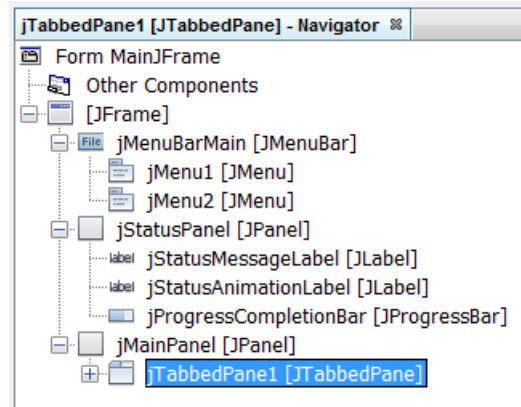
JTabbedPane is used to control the functionality of application using tabs.

Let us create the tabbed control in the frame.

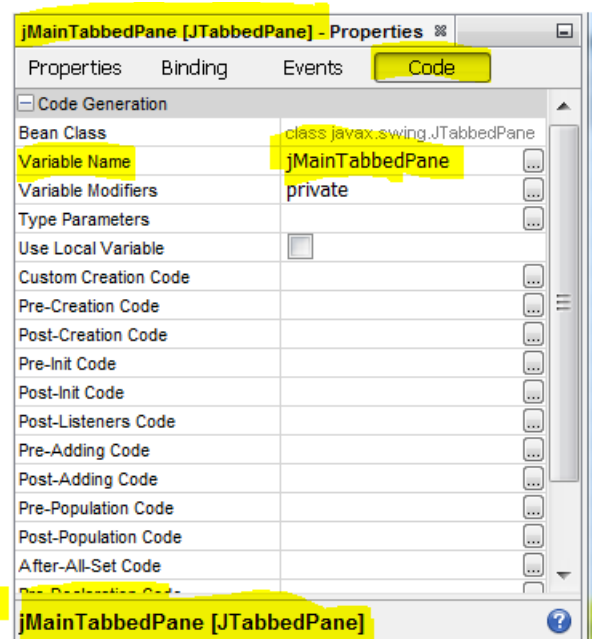
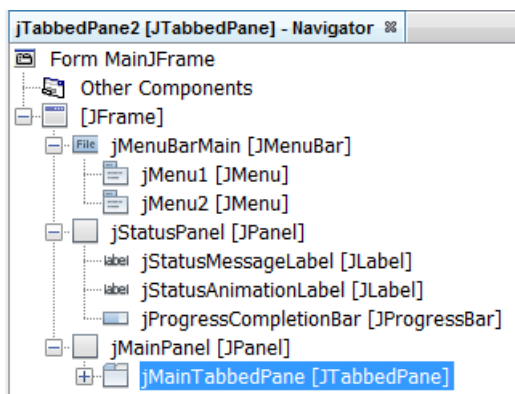
1. Create **jMainPanel** type of **JPanel** in **JFrame**.
2. There are 2 ways to create the tabbed control inside of **jMainPanel**:
 - a. Drag **Tabbed Pane** from **Swing Containers** (located in top left corner of the screen) and drop it to **JFrame**.
 - b. Select **JFrame**. In Design view, right click mouse and select **Add From Palette -> Swing Containers -> Tabbed Pane**.



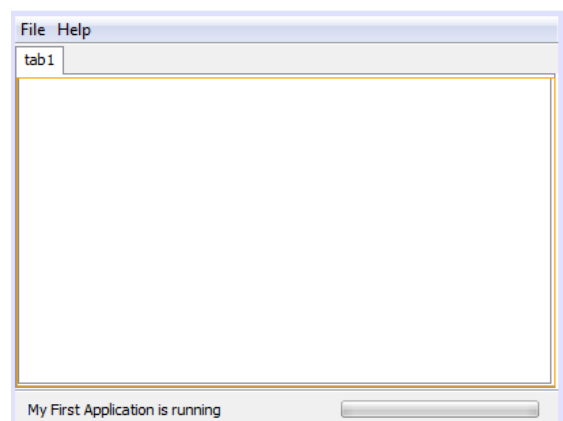
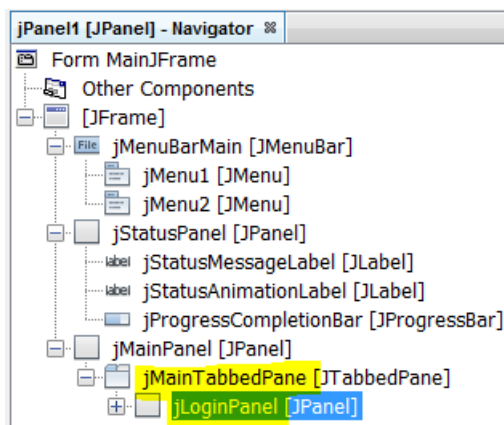
As a result **the *jTabbedPane1****jTabbedPane* is added as a component to ***mainPanel****jPanel*.



3. Change the name of variable ***jTabbedPane1*** to ***jMainTabbedPane***, select ***jTabbedPane1*** in Navigator, right click mouse button and select "**Change Variable Name...**". In the pop up window, type ***jMainTabbedPane*** and click OK. The ***jTabbedPane*** will be renamed as shown in Figure (left). Correspondingly the name property of ***jTabbedPane*** will be changed as shown in Figure (right).



4. Add **Panel** from **Swing Containers** to ***jMainTabbedPane***. Use one of the ways described in Step 1. Rename **Panel** to **LoginPanel** using "**Change Variable Name...**" feature. If the step is completed correctly, the following should be shown in the Navigator (left) and in Frame (right):

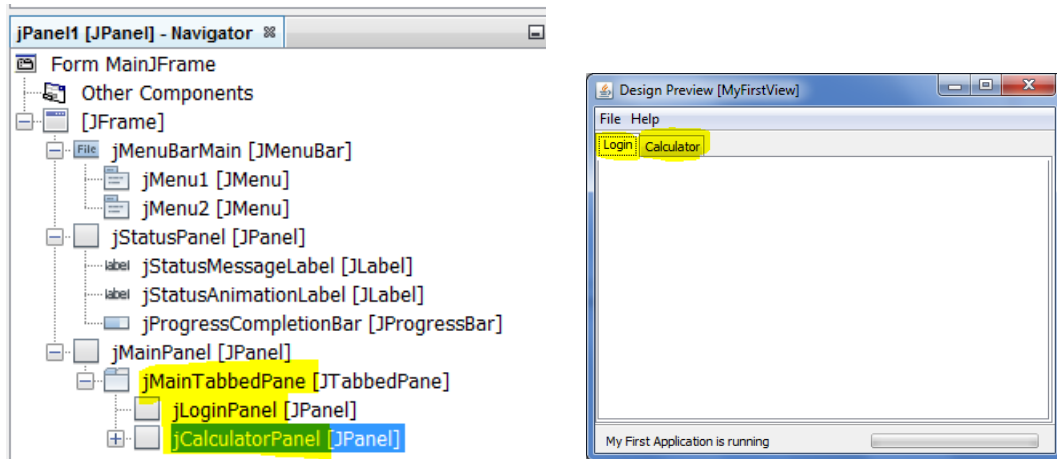




Task: Explore the tabbed control interface. Document the outcome of each step in the report.

1. Explorer the properties of **jLoginPanel** and change at least the title of tab1 to "Login".
2. Add one more tab named "Calculator" to the Frame.
3. Change the name of variable type of **Panel** to **jCalculatorPanel**.

If the tasks are completed correctly, one will achieve the **Navigator** view, given left, and **Design Preview**, given right.



Creating Label control

Label allows displaying text in the window.

1. Select **Login** tab in **Design** view.
2. Drag the **Label** from **Swing Controls** and drop to the **Login** tab.
3. Change the **text** property of the label to "What is your name?"
4. Change the name of variable of **JLabel** to **jNameLabel**. (Refer to Step 2 and Step 3 in Chapter "Created the tabbed control")

Creating TextField

TextField allows to user input the data into the program and consequently display data.

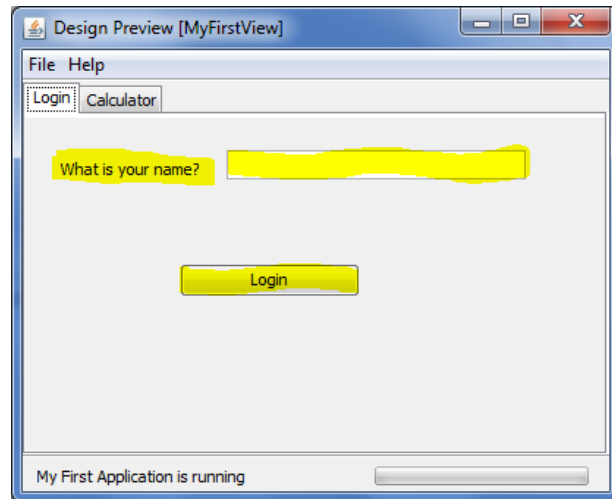
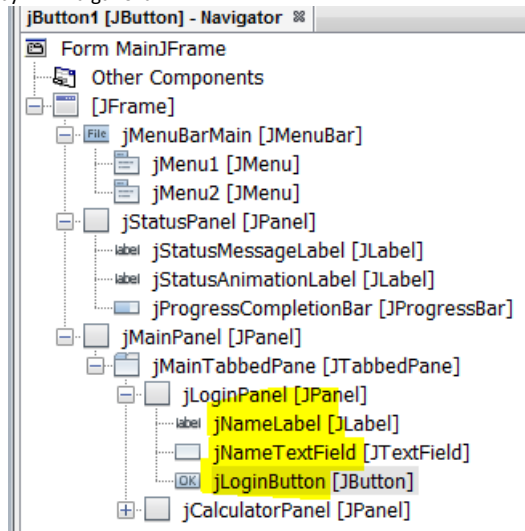
1. Select **Login** tab in **Design** view.
2. Drag the **TextField** from **Swing Controls** and drop to the **Login** tab.
3. Change the **text** property of the label to " ".
4. Change the name of variable of **JTextField** to **jNameTextField**. (Refer to Step 2 and Step 3 in Chapter "Created the tabbed control")

Creating Button

Button allows to perform the specific actions programmed by user.

Adding Button

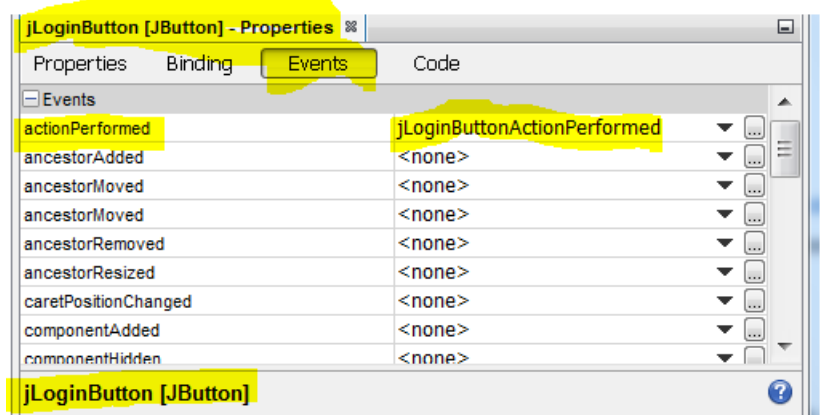
1. Select **Login** tab in **Design** view.
2. Drag the **Button** from **Swing Controls** and drop to the **Login** tab.
3. Change the **text** property of the label to "Login".
4. Change the name of variable of **JButton** to **jLoginButton**. (Refer to Step 2 and Step 3 in Chapter "Created the tabbed control")



Creating action associated with Button

In the current case, the Button does not have any actions to be performed. Let us add some actions to the Button created.

1. Select **jLoginButton** in **Navigator**.
2. Select **Events** tab in **Properties**.
3. From drop down menu for **actionPerformed** select **LoginButtonActionPerformed**.



If the actions are completed correctly, the Source window will be opened and body of function named **LoginButtonActionPerformed** will be created as shown below:

```

172 private void jLoginButtonActionPerformed(java.awt.event.ActionEvent evt) {
173     // TODO add your handling code here:
174 }
175

```

Let us write down the handling code for the button. Once the button is clicked, the program reads the data from **jNameTextField** and writes the message "The user is logged in..." in the **jStatusMessageLabel** located in status bar.

The program code shown below completes writes anything that is entered in the **jNameTextField** into **jStatusMessageLabel**.

```

172 private void jLoginButtonActionPerformed(java.awt.event.ActionEvent evt) {
173     jStatusMessageLabel.setText(jNameTextField.getText() + " is logged in...");
174 }
175

```

Method **getText()** is used to retrieve text property of the component.

Method **setText()** is used to initialise the text property of the component.



Click the **Run Main Project (F6)** button. Once the project is compiled, the application will be running. If after compilation the window of your application has not been appeared, press Shift-F6. Or select the

application in Projects, right click mouse and select "Run File" from pop up Menu. This should sort out your problem.



Test the performance of the application: Enter your name in the TextField and click Login Button. Does the application produce the expected result?

Let us modify the program code further and open the Calculator tab once the user is logged in. In order to do so, you need to change the selected index of **jMainTabbedPane** to 1. The selected index defines the index of active tab and it equal to 0 for **jLoginPanel** and for **jCalculatorPanel** is 1. We use **setSelectedIndex()** function to change the Selected index.

```
172 private void jLoginButtonActionPerformed(java.awt.event.ActionEvent evt) {
173     jStatusLabel.setText(jNameTextField.getText() + " is logged in...");
174     jMainTabbedPane.setSelectedIndex(1);
175 }
```



Test the performance of the application: Enter your name in the JTextField and click Login Button. Does the application produce the expected result? If no, attempt to modify the program in such way that it will produce the required output.

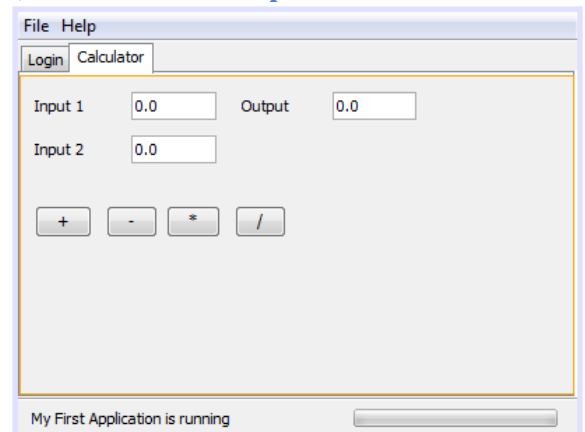
Implementing calculator



Task: Implement interface for basic calculator.

All tasks must be completed to PASS this lab in the threshold assessment. Design the test case for this task. Complete the designed test case, document the output.

1. Add required **Labels** and **TextFields** to the **Calculator** tab. Modify the properties of Labels and TextFields as shown on Figure.
2. Rename variables for **Labels** and **TextFields** in such way that name will reflect their meaning.
3. Add 4 Buttons that correspond to 4 basic operations as shown above
4. Implement the functionality of SUM button. Design the test case for this task.



Complete the designed test case, document the output. Use the following example as an aid:

```
216 private void jSumButtonActionPerformed(java.awt.event.ActionEvent evt) {
217     // read data from the text fields into memory
218     int input1 = Integer.valueOf(jInput1TextField.getText());
219     int input2 = Integer.valueOf(jInput2TextField.getText());
220     // perform calculation
221     int output = input1 + input2;
222     // display the calculation result in the output field
223     jOutputTextField.setText(String.valueOf(output));
224 }
225
```

Lines 218 – 219 save values entered by software user in the **JTextFields** into integer variables **input1** and **input2**. **getText()** method is used to retrieve data from **text** property of

JTextField. This method returns data type of **String**. **Integer.valueOf()** method is used to convert the String to integer.

Line 221 performs calculation of required method.

Line 223 displays the value, stored in output variable into **JOutputTextField**. **String.valueOf()** method converts integer to String.

5. Modify the program code in such way that the program can process **double** values. Design the test case for this task. Complete the designed test case, document the output.
6. Implement the functionality of **subtraction** and **multiplication** buttons. Design the test case for this task. Complete the designed test case, document the output.
7. Implement the functionality of **division** button. Display the message "Cannot divide by zero", if attempted the division by zero. Make sure that division by zero does not appear when the program code is executed. Design the test case for this task. Complete the designed test case, document the output. Complete the designed test case, document the output.

No use of mathematical libraries is allowed in implementation of any functionality summarised below. Use previously implemented methods instead of completing the calculation inside of each button method. It is expected the following independent methods will be created and reused at least once:

- ✓ power();
- ✓ CylinderVolume(),
- ✓ SphereVolume(),
- ✓ SquareBasedPyramidVolume()

8. Calculate **power**, where input 1 is a base and input 2 is a power: $\text{input1}^{\text{input2}}$. Declare and implement separate **power()** method and call it when dedicated button is created.

9. Calculate volume of Cylinder: $\text{volume} = \pi r^2 h$, where r = radius and h = height;

10. Calculate volume of Sphere: $\text{volume} = \frac{4}{3} \pi r^3$



Requirements to the programming style:

1. Each variable must be named according to its meaning.
2. No alphabetical naming of variables.
3. No standard math libraries are allowed to use.
4. Write re-usable program code: the program code must **not** be repeated.
5. Declare **power()** method and later reuse this method in calculation.
6. The data inputted via **JTextField** object must be validated for the corresponding mathematical equations. It may good to implement separate validation method for integer, double numbers, etc.
7. The outcome of each calculation must be compared with calculation produced by ordinary calculation and compared in the report. Does output produced by your program matches the ordinary calculator exactly in all cases? Is precision an issue? Search keywords such as "java string format decimal places" and "formatting double java" might be useful to resolve the problem.