



Getting Started with IDE Programming

Learning Objectives

After studying this lesson the students will be able to:

- ❖ create a project
- ❖ create a new form
- ❖ appreciate the concept and importance of a Graphical User Interface and an Integrated Development Environment
- ❖ understand the need and use of components like Button, Text Field, Labels, Password Field, Text Area and Radio Buttons.
- ❖ add components to the form and change the properties of the components
- ❖ attach code with components of the form
- ❖ develop small applications involving simple calculations with integers and decimal numbers.

In our day to day life, we have to give information innumerable times like fill up bank deposit slips to deposit money or type in username and password to sign in to our mail account and many more. Forms are means to accept data (input) from us and respond as soon as we perform an action like clicking on a button or submitting the form. Have you ever wondered how they are created and wanted to create them yourselves? Well, our journey is a quest in this direction. This chapter deals with teaching the basic process of designing forms in Netbeans and using them to perform simple manipulations using Java.

Introduction

Observe Figure 4.1 carefully. We all may have come across some of the following but have we ever given a thought about what these are? Let us try and analyze their utility. The Google window displayed returns names of websites matching the search text entered by the user in the input area. The Gmail window displayed helps one to log in to



their mail account and view their inbox based on the username and password entered by the user. The IRCTC window displayed helps the user to log in to the Indian Railway site to book train tickets or retrieve specific train information.

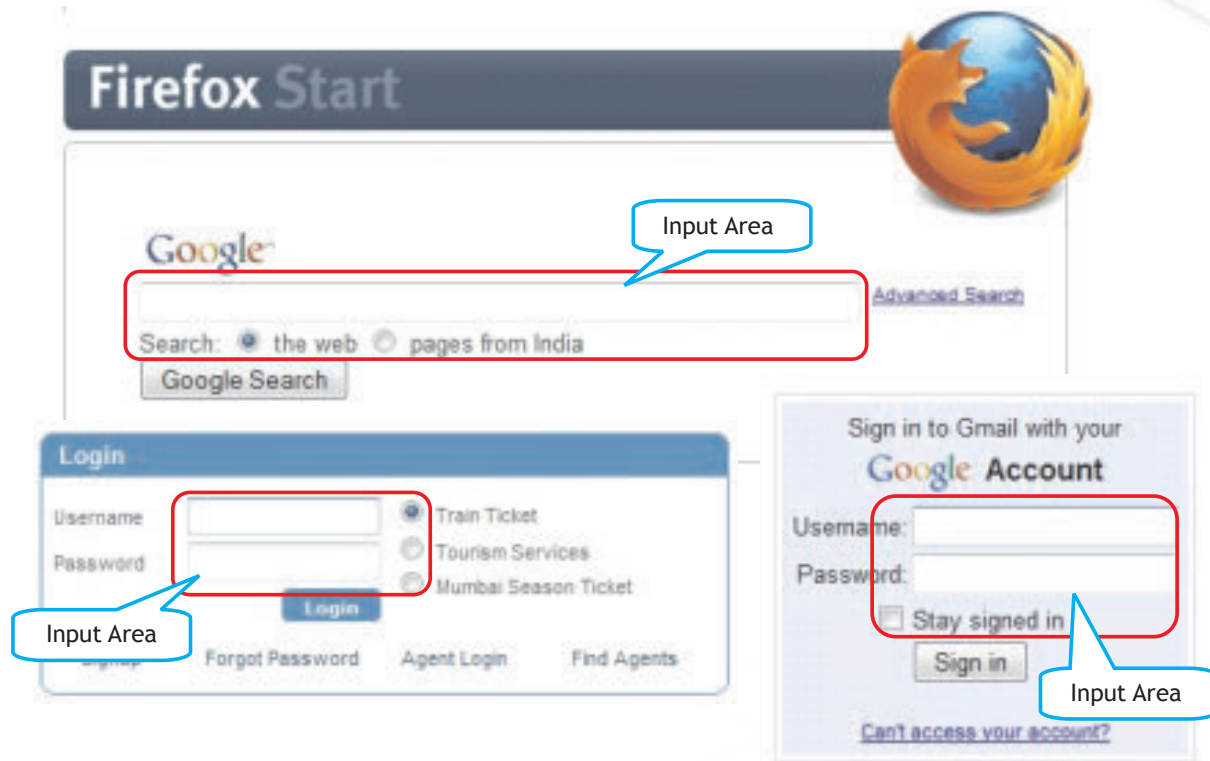


Figure 4.1 Common Ways to Accept Input

From the above discussion we infer that all these are means of entering data. The data entered may be sent to some external agent for processing or simply stored somewhere. Such means of collecting data are known as forms. Forms are used to accept data (input) from users and respond to actions like clicking on a button. In the above figure, the enclosed area is used to accept input and clicking on the button results in some output. Now that we understand what a form is, let us start with a little fun and try to create our very own form. Since this will be our first application so we will do something very simple. We will create a form with a single button and on the click of this button, we will simply exit from the application. We will use Netbeans to create this form, so first start Netbeans (Refer to Appendix 5 for installation and starting of Netbeans). Since all development in the Netbeans takes place within projects, we first need to create a new project within which we will store codes and other related files. The project will act as a storage place for all the forms and codes created by us.



Creating a new Project

To create a new application project called "Book":

1. Choose File > New Project. Alternately, click the New Project icon in the toolbar.
2. From the Categories pane select Java and in the Projects pane, choose Java Application. Click Next.
3. Enter a name (in this case Book) in the Project Name field and specify the project location by clicking on the Browse button. By default the project is saved in the NetBeansProjects folder in My Documents and so this is the default Project location displayed in this field.
4. Ensure that the Set as Main Project checkbox is selected and clear the Create Main Class field.
5. Click Finish.

Netbeans creates the Book folder on your system in the designated location. This folder will contain all of the associated files of the project. The next step is to create a form. To proceed with building our form, we need to create a container within which we will place the other required components of the form like a button. For all our applications we will choose the JFrame Form as the container to place other components.

Creating a new Form

To create a JFrame Form container:

1. In the Projects window, right-click the Book node and choose New > JFrame Form as shown in Figure 4.2.
2. Enter Form Example 1 as the Class Name. This will be the name of your form.
3. Enter Book as the package. This should be the name given while creating the Project.
4. Click Finish.



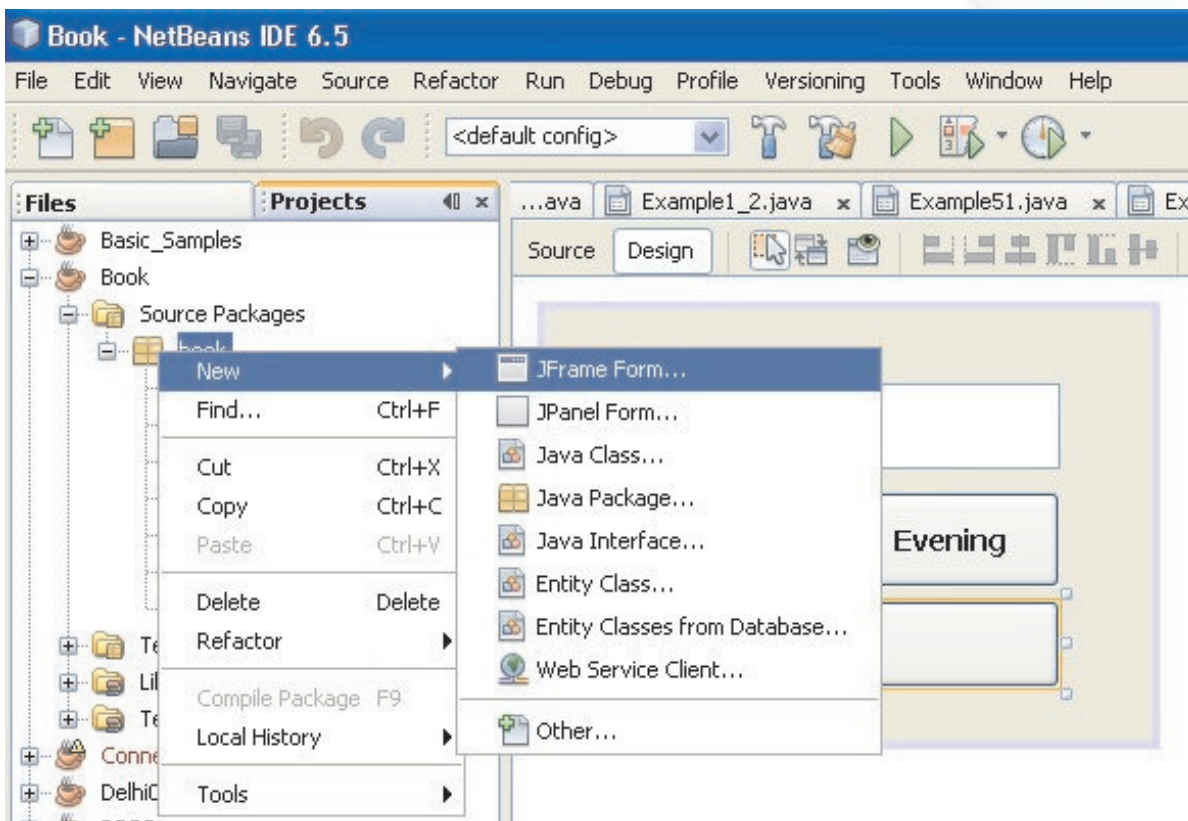


Figure 4.2 Adding a new JFrame Form

Netbeans creates The Form Example1 form within the application and opens the form in the Builder. Now we are ready to add components to our form.

Adding a Button Component to a Form

We want to add a button so follow the given steps to add a JButton to the form:

1. In the Palette window, select the JButton component from the Swing Controls category (displayed in Figure 4.3).
2. Move the cursor over the Form. When the guidelines appear (as displayed in Figure 4.3) indicating that the JButton is positioned in the desired location, click to place the button.

The JButton is added to the form as displayed in Figure 4.3. Note that as soon as the button is added on the form, a corresponding node representing the component is added to the Inspector window.



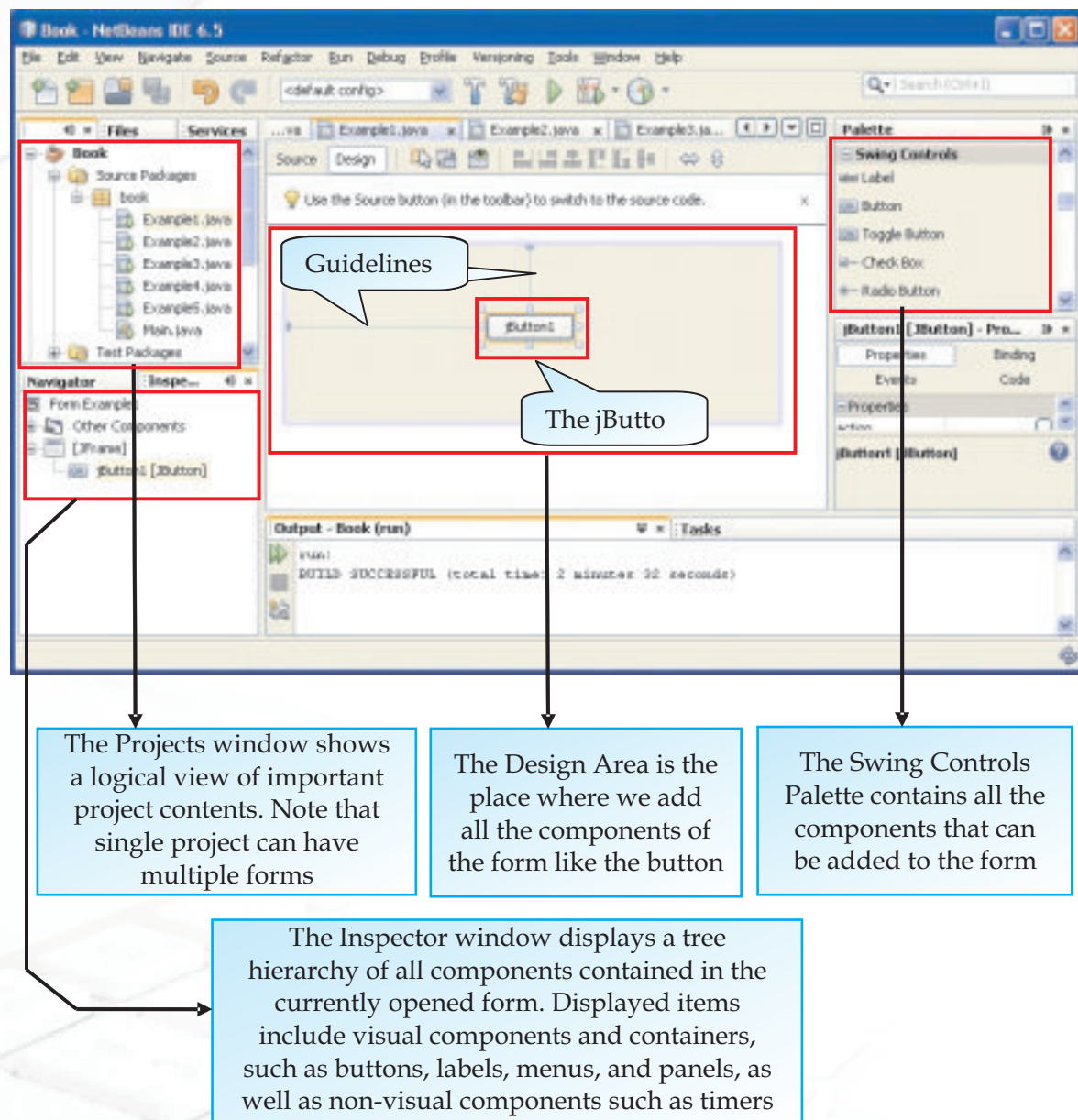
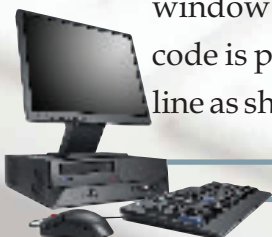


Figure 4.3 Adding a Button and Understanding the Different Windows

Attaching Code to a Form Component

After placing the button, the next step is to write a code to exit from the application on the click of this button. To do the same, double click on the button to attach a code with the event i.e. click of the button. Double clicking on the component opens up the source window and places the cursor on the point where code is to be added. Note that certain code is pre generated and cannot be changed. In the Source window add the single code line as shown in Figure 4.4.



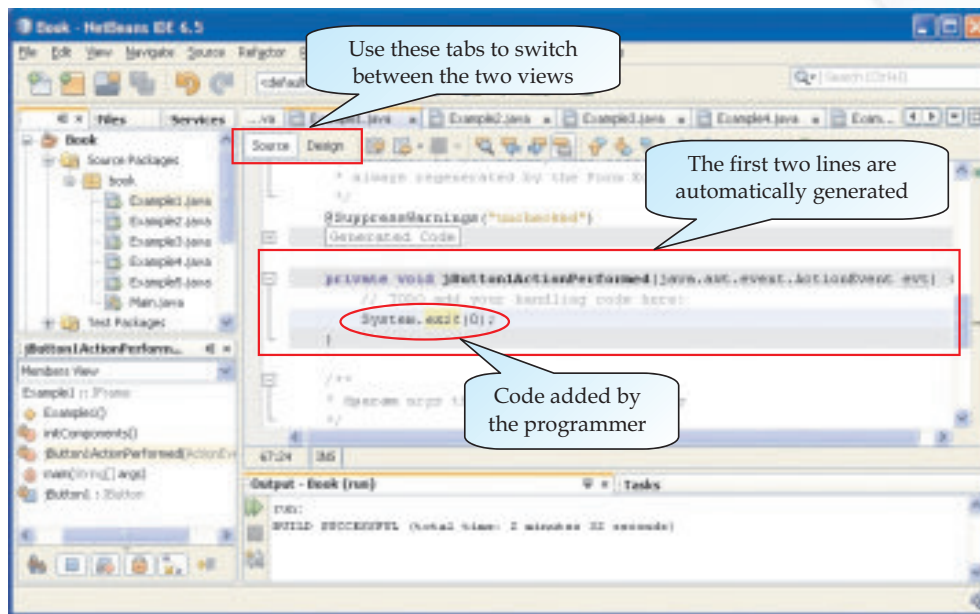


Figure 4.4 Code to exit from an application

Know more

When we click the Source button, the application's Java source code in the Editor is displayed with sections of code that are automatically generated by the Netbeans Builder indicated by gray/blue areas, called Guarded Blocks. Guarded blocks are protected areas that are not editable in Source view. Note that we can only edit code appearing in the white areas of the Editor when in Source view.

Executing a File

Now that the code for the first application is ready let us test our first application. To execute the application simply select Run>Run File or press Shift+F6 as shown in Figure 4.5.

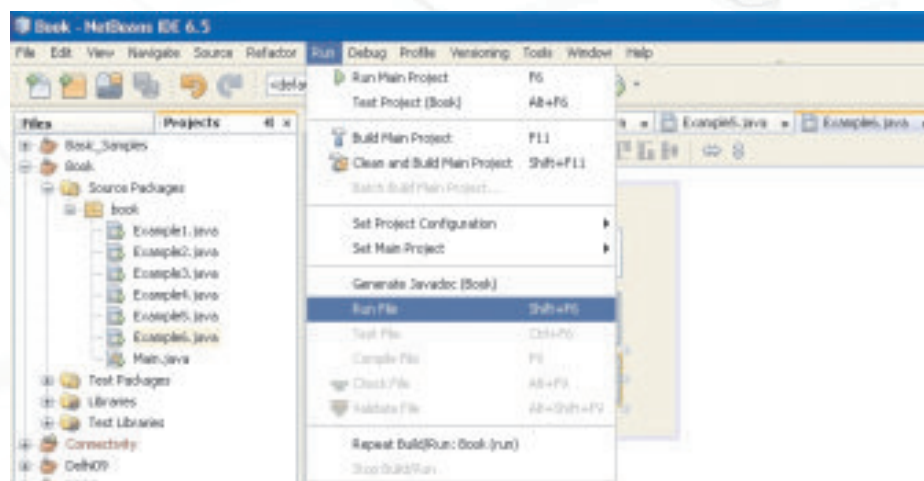


Figure 4.5 Executing a File



On executing the first example, the window shown in Figure 4.6 will appear. Click on the button and observe the result.

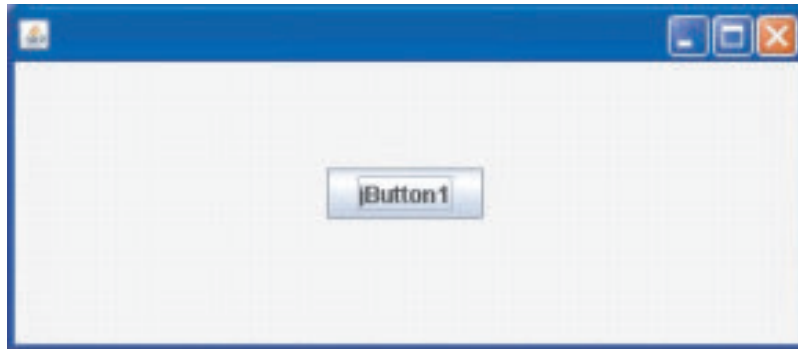


Figure 4.6 Simple Button Application

As soon as we click on the button, the application ends and we return back to the Netbeans design window. So what did we learn? We learnt that the one line of code `system.exit(0)` causes the application to terminate successfully.

Know more

Every object placed on a form supports one or more events that recognize when anything happens to the object. For example, a button placed on the form can be clicked, can be double clicked, can be activated or can be deactivated. To know more about the events associated with the different objects refer to Appendix1.

Isn't it simple? Just one button and one line of code and a wonderful achievement - Closing an application. Did you ever think it was that simple? Well now you know that programming is not really tough. Let us test our observation. Go back and carefully observe Figures 4.3 and 4.4 once again. What do you notice? Did you notice that the window in which the form has been designed is different from the window in which we have written the code? Let us understand this carefully. The window in which we have designed our form is called the Design window and the window in which we have written the code is called the Source window. We can easily switch between the two views by simply clicking on the relevant tab as displayed in Figure 4.4.

We have had an interesting start towards building our very first form and also learnt to write code. Now let us quickly recap the basic steps used while developing the above application before moving on.



Quick Recap - Steps for developing a Simple application

- Step 1: Create a new Project
- Step 2: Add a JFrame form
- Step 3: Add the desired component from the Palette window using drag and drop feature
- Step 4: Associate code with the component by double clicking the component.
- Step 5: Add the source code.
- Step 6: Test the form by pressing Shift+F6.

The above form looks good but it would have looked great if the button had shown the text STOP instead of jButton1. The text STOP on the button would have easily explained to the user that clicking the button will stop the application run. So now let us try and achieve this.

Changing Properties of Components

Each component of our application including the form has certain attributes associated with it. The Properties Window displays the names and values of the attributes (properties) of the currently selected component. We can edit the values of most properties in the Properties window.

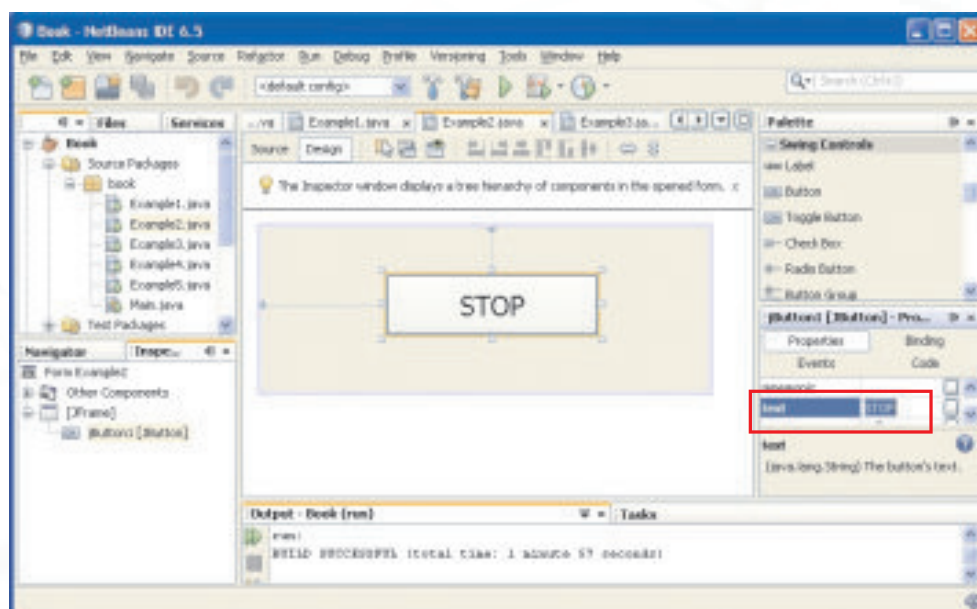


Figure 4.7: Using the text property of a button to change the display text



We want to change the text displayed on the button. There are four ways of doing the same in the design view:

- ❖ Select the button component by clicking on it. In the Properties window highlight the text property and type STOP in the textbox adjacent to it as displayed in Figure 4.7.
- ❖ Alternatively select the object. Left click on the button to highlight the display text. Type STOP and press Enter.
- ❖ Select the object > Press F2 - to make the display text editable. Type in the new text and press Enter.

Right click on the button component and select Edit Text from the Drop down menu to make the display text editable. Type in the new text and press Enter.

Using the Properties window, it is also possible to change the Font and Foreground property of the button as displayed in Figure 4.8

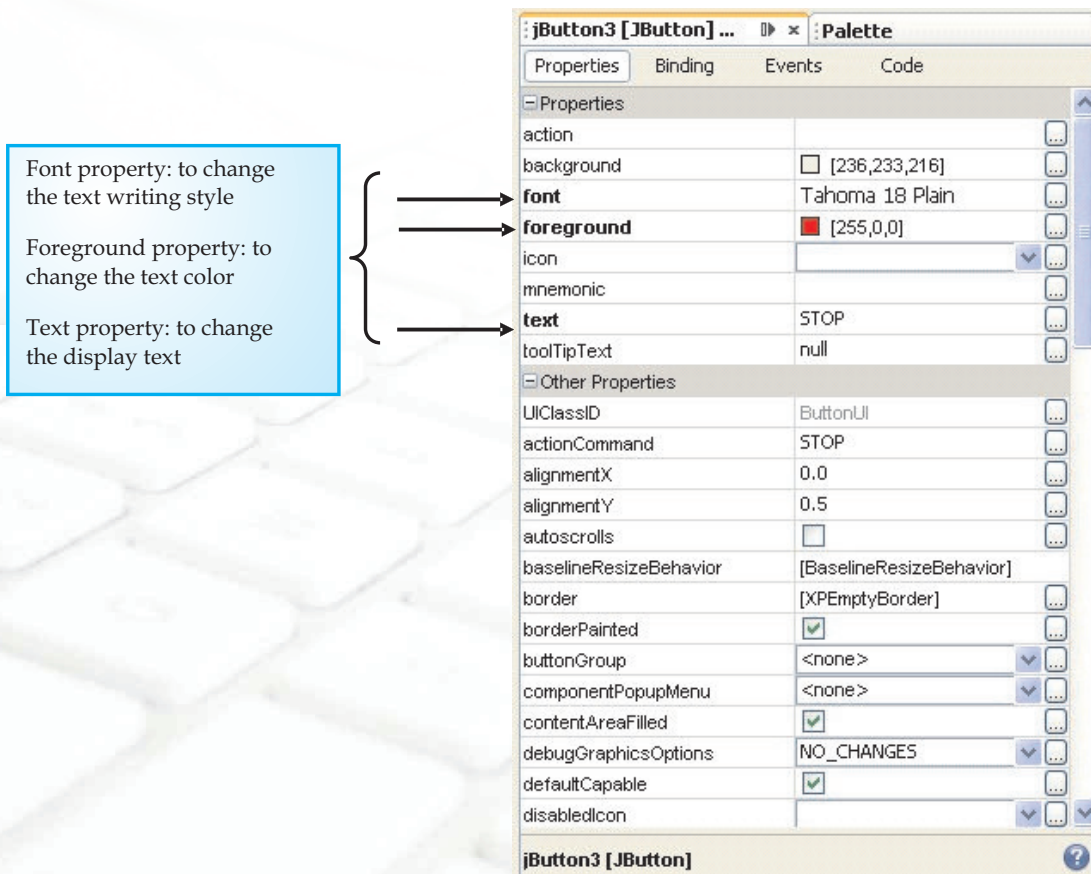


Figure 4.8 Changing Properties of a Button Using the Properties Window

Now when we execute the file the button with the changed text appears as shown in Figure 4.9.

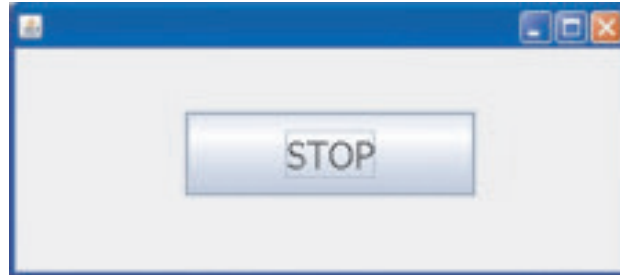


Figure 4.9 The Button with an Appropriate Display Text

! Whenever there is a change in the display text of a component, the new name is displayed and the component's height and width gets adjusted as a result of the change. Use the resize handles to increase the size of the component according to your requirement.

Displaying a Message in a Dialog Box

Now, that we are comfortable with the creation process, let us experiment further and try to display a message on the click of the button. Follow the same process to create a fresh form with a simple button as shown in Figure 4.10. Modify the properties of the button as desired and change the text property to "Wish Me".

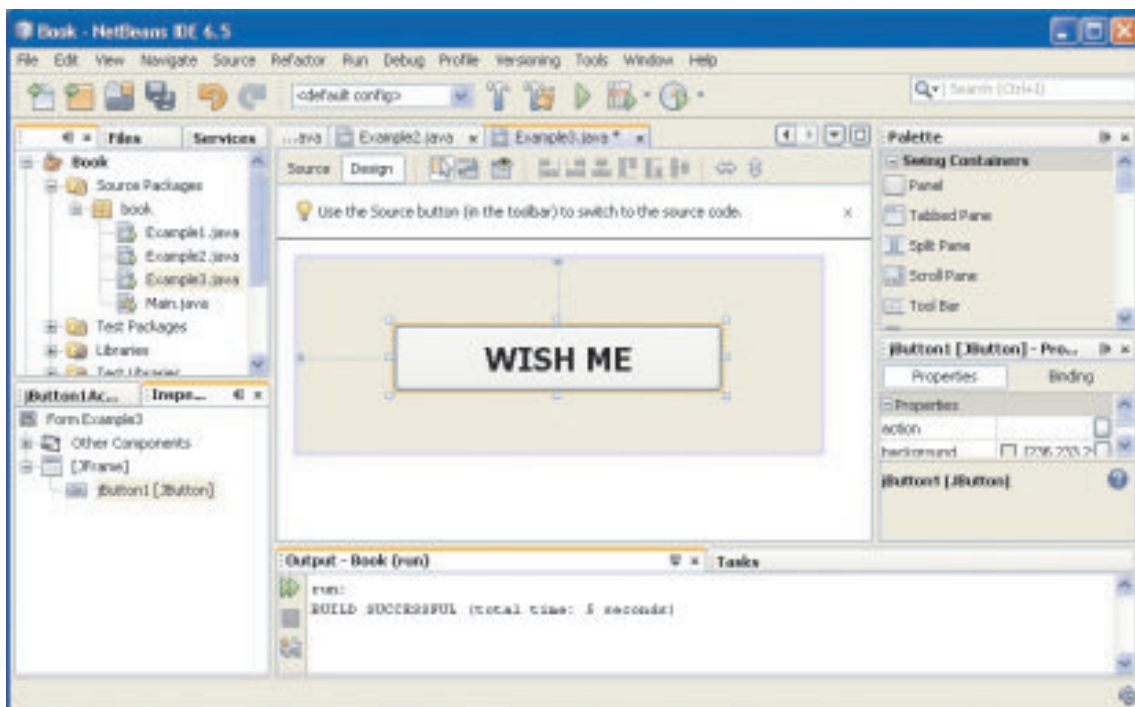


Figure 4.10 A Button with Modified Properties



Know more

In graphical user interfaces, a dialog box is a special window, used in user interfaces to display information to the user, or to get a response if needed. They are so-called because they form a dialog between the computer and the user - either informing the user of something, or requesting input from the user, or both. It provides controls that allows the programmer to specify how to carry out an action.

Switch to the source window and add the single line code as shown in Figure 4.11.

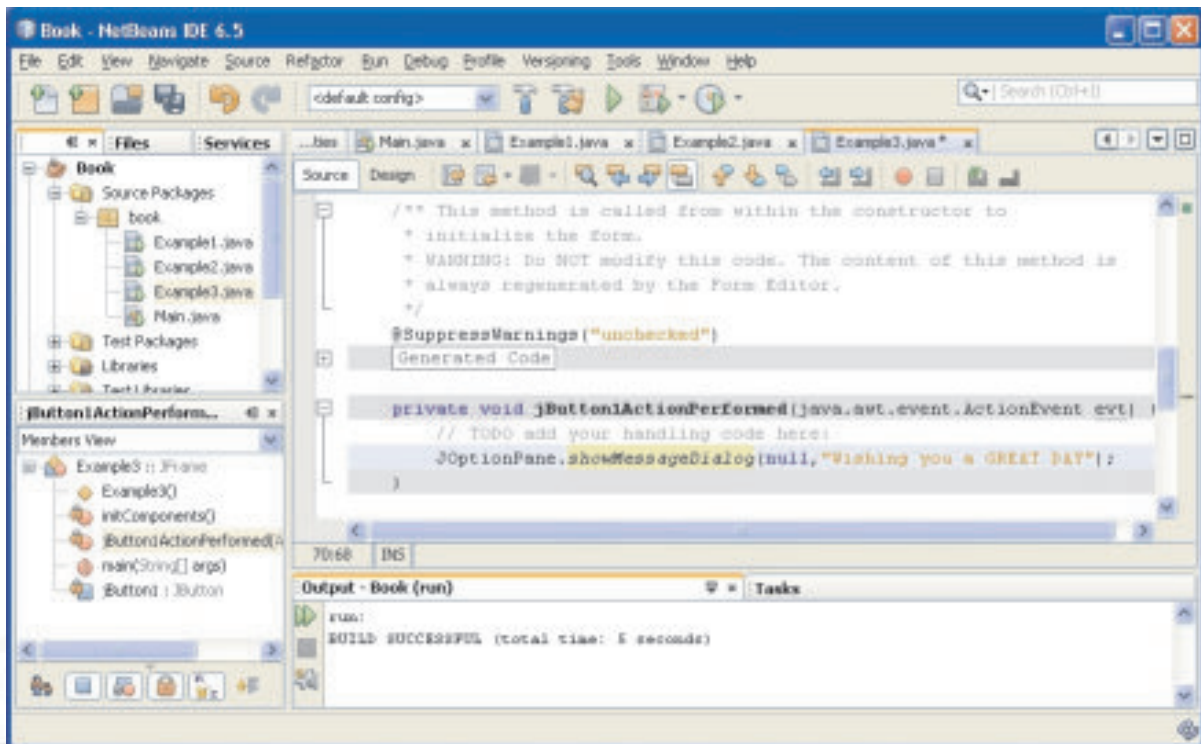


Figure 4.11 Code to Display a Message on the Click of a Button

As soon as you enter the single line code given above, an error indicator and the error message "cannot find symbol" will be displayed. This error simply means that the JOptionPane component is missing from our application. To fix this error we need to add this component. Left click on the error indicator to display a menu with 3 different options and select the option Add import for javax.swing.JOptionPane from the menu as shown in Figure 4.12.



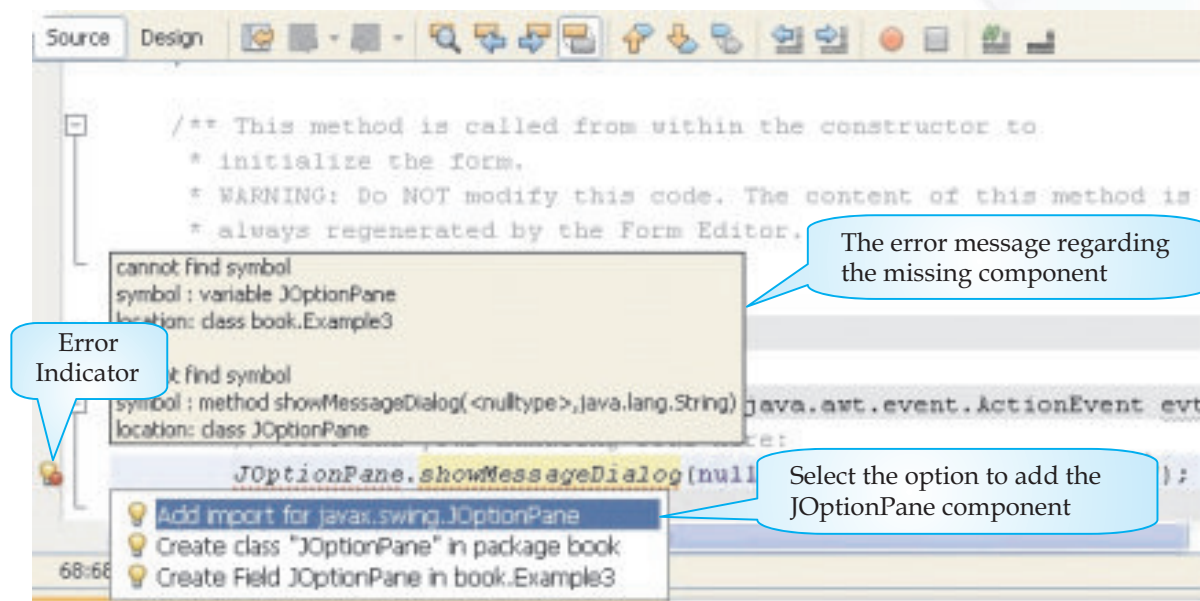


Figure 4.12 Adding the JOptionPane Component

Now execute the file by pressing Shift+F6. Click on the button to see the message. The execution is shown in Figure 4.13

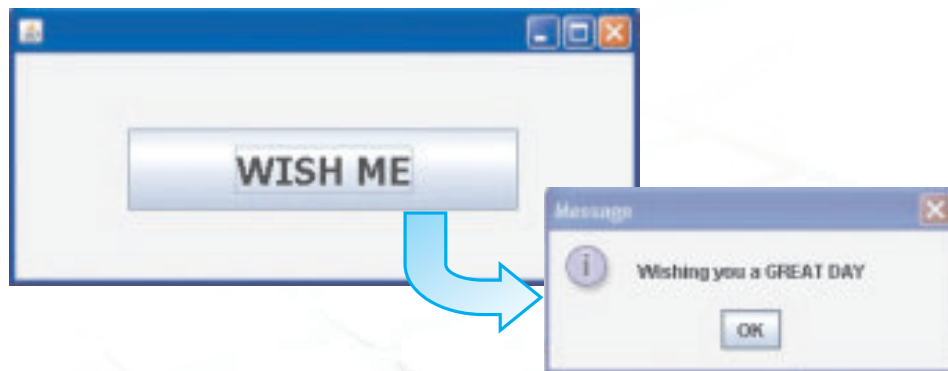


Figure 4.13 Displaying a Message in a dialog Box on the Click of a Button

In this example we learnt that the `showMessageDialog` method can be used to display a specified message in a Dialog box. Till now we have learnt how to use:

- ❖ the Design Window to create a Form and add components to it
- ❖ the Palette to add Swing Controls on the form
- ❖ the Inspector window to view the hierarchical relation among the different components
- ❖ the Properties Window to change the attributes of a selected component



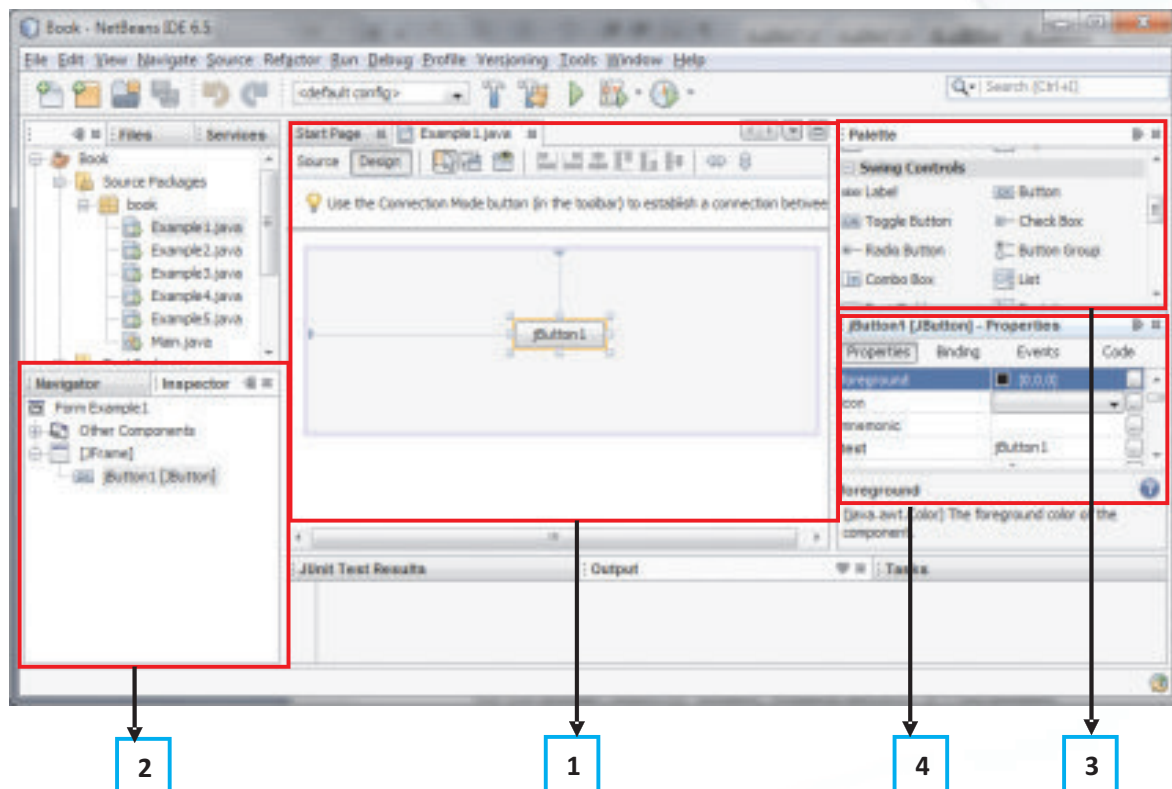
So before proceeding further, let us take a minute to further familiarize ourselves with these different parts of the Netbeans Builder interface. It is important to familiarize ourselves with the Netbeans Interface as it allows us to communicate with the different parts of Netbeans easily and makes our task easier.

Familiarizing with the Netbeans GUI Interface

As experienced above, Netbeans is a GUI. A GUI is an acronym for the term Graphical User Interface. It is known as a GUI as it allows us to interact with the various components through visual elements including pictures, graphical icons, symbols and visual indicators. For example to add a button we simply have to drag the button icon from the Swing Controls tab in the Palette. Similarly as we are writing code, small visual indicators appear informing us about the errors in the code. The Netbeans GUI Builder's various windows include four main components as displayed in Figure 4.14. These components are explained below:

1. **Design Area.** The GUI Builder's primary window for creating and editing Java GUI forms. The toolbar's Source and Design toggle buttons enable us to view the source code or a graphical view of its GUI components. The additional toolbar buttons provide convenient access to common commands, such as aligning components, setting component auto-resizing behaviour, and previewing forms.
2. **Inspector.** Provides a graphic representation of all the components, both visual and non-visual, in our application as a tree hierarchy. The Inspector also provides visual feedback about what component in the tree is currently being edited in the GUI Builder as well as allows us to organize components in the available panels.
3. **Palette.** Contains a customizable list of available components containing tabs for JFC/Swing, AWT, and JavaBeans components, as well as layout managers. In addition, we 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, Inspector window, Projects window, or Files window.





Components: 1. Design Area 2. Inspector Window 3. Palette 4. Properties Window

Figure 4.14 Netbeans GUI Interface

The GUI Builder makes it possible to build professional-looking GUIs without an intimate understanding of layout managers. We can lay out our forms by simply placing components where we want them. Another interesting feature of Netbeans is that it provides comprehensive facilities for software development. It is very helpful as it maximizes the programmer productivity by providing tightly-knit components with similar user interfaces. This kind of an environment where all the tools required for developing an application are available in a single place is known as an Integrated Development Environment or simply IDE. Using an Integrated Development Environment (IDE) for developing applications saves our time by managing windows, settings, and data. In addition, an IDE can store repetitive tasks through macros and abbreviations. Drag-and-drop features make creating graphical user interface (GUI) components or accessing databases easy, and highlighted code and debugging features alert us to errors in our code.



Know more

Because the NetBeans IDE is open source, it is undergoing continual improvement, you may notice slight differences between the screen captures in this book and the latest download. This book is based on NetBeans 6.5 and may vary slightly from later versions as they become available. You can download the latest version from <http://netbeans.org/>

Adding More Components to a Form

Great, now that we are comfortable with the interface, let us get back to the programming journey. In the last example we had displayed a message on the click of a button. Now what next? All the previous examples had only one component. Let us now delve further and try adding more than one component to our form. Adding more components means that we will have multiple code lines. So, first let us try and add more of similar components i.e. more buttons. So we will design an application with 3 separate buttons and display a different message on the click of all the three buttons.

Think what should be the first step?

Right, the first step is to add a new form and then we will add three buttons on the newly created form. Drag and drop three buttons from the Swing Controls tab to complete the form design as shown in Figure 4.15. Don't forget to change the properties and use the resize handle to make the form appear exactly as shown in the Figure 4.15.

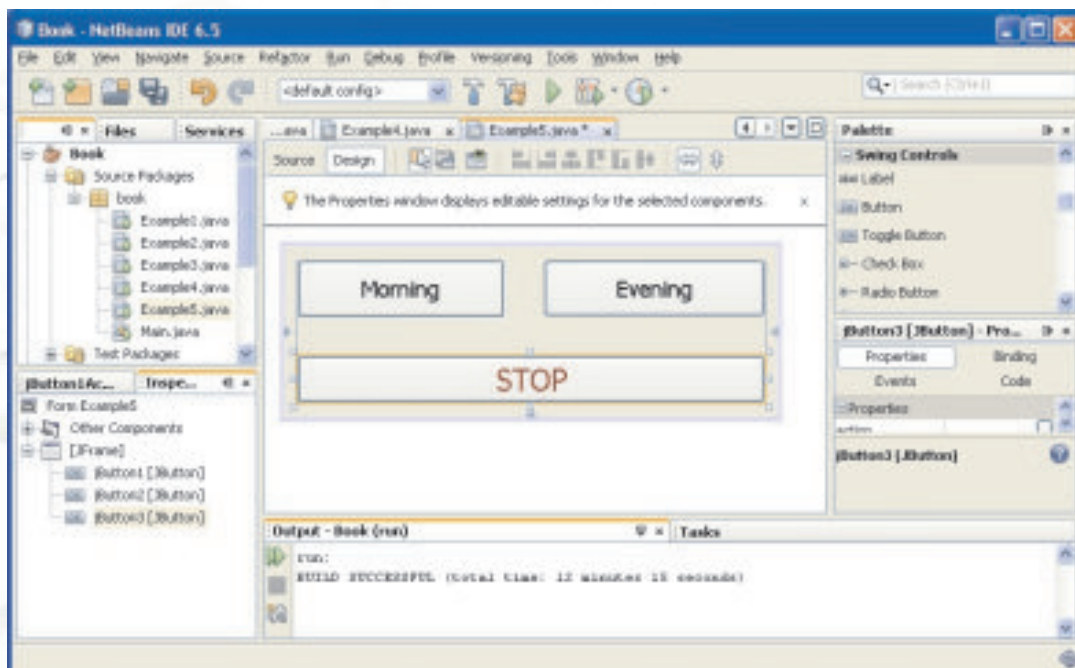


Figure 4.15 A Form with three buttons

Now, just think how to associate different code with each of the three buttons.

Remember, double clicking on a particular button opens up the source window with the cursor placed at the point where code is to be added. So just do the same for all three buttons. Double click on each of the three buttons one by one and keep on adding the relevant code for each one of them. We are going to use the commands we have already learnt in our previous examples to:

- ❖ Display the message "Good Morning" on the click of the Morning button
- ❖ Display the message "Good Evening" on the click of the Evening button
- ❖ End the application on the click of the STOP button.

The complete code for all three buttons is displayed in Figure 4.16

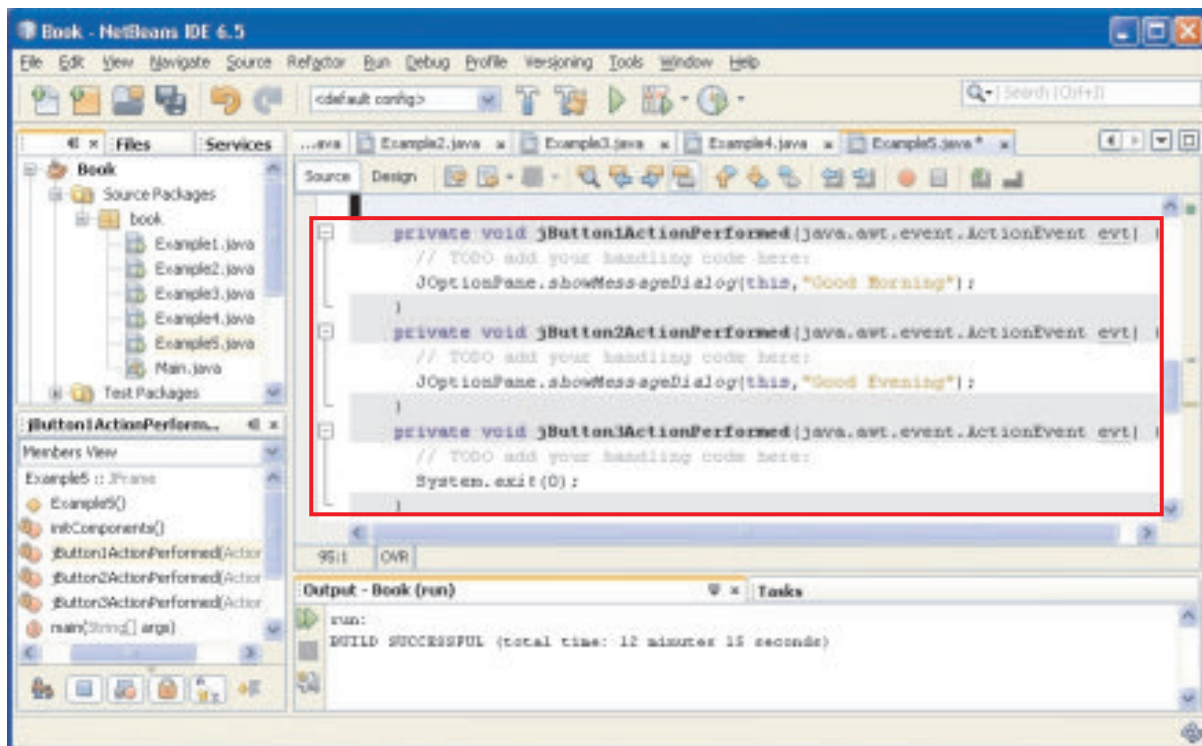


Figure 4.16 Code to Add Functionality to the Form designed in Figure 4.15

Now execute the Example and observe the result of clicking Morning and Evening Buttons. One of the outputs is displayed in Figure 4.17



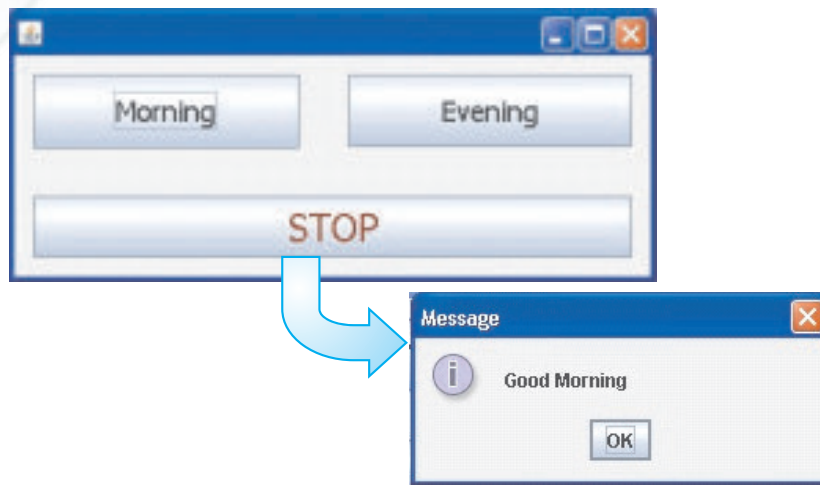


Figure 4.17 Clicking on the Morning button displays the message "Good Morning"

! As we create applications and add to them new objects such as buttons and textboxes, they are automatically assigned names such as `jButton1`, `jButton2` and so on by the IDE. But it is good practice to give names that better match the functionality, such as `BExit` and `BMorning`. Remember that objects on the same form cannot have same name, but two forms might contain objects with the same name.

Using a Text Field Component to Display Messages

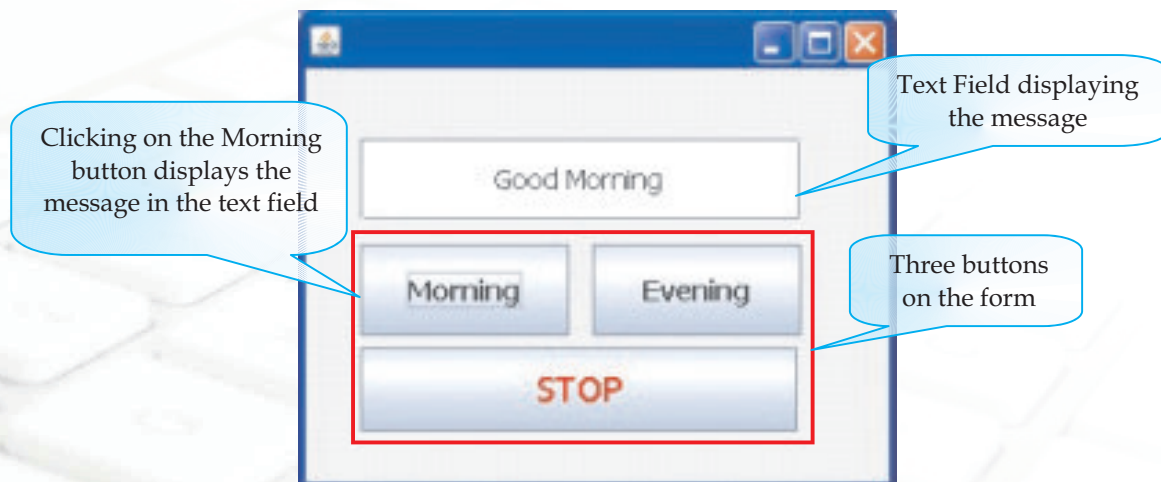


Figure 4.18 Display message in a Text Field on the click of a button

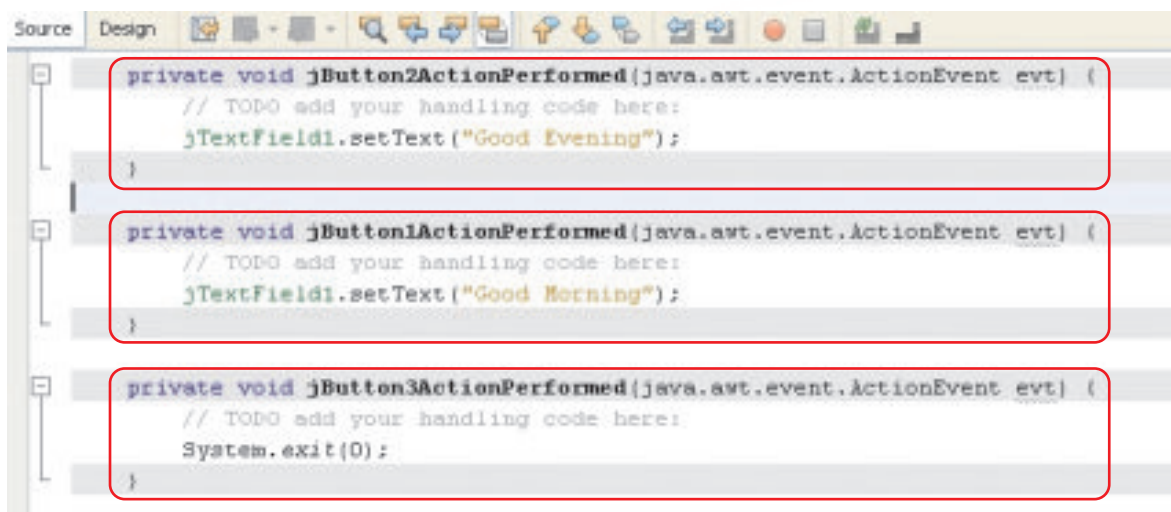



In all the above examples we have displayed all the messages in dialog boxes. But In real life applications we might have to display messages in Text fields too. So we will try and learn about the text field component in our next example. The Text Field component is a text input field in which users can enter single line of text.

We will make a slight modification to the above example by displaying the message in a text field rather than in the dialog window. First, make an attempt to design the form displayed in Figure 4.18 by dragging a Text Field component from the Swing Control Palette to our previous form. Now changing the strategy a bit, let us first look at the sample run of the form designed as shown in Figure 4.18. On the click of the Morning button, the message "Good Morning" should be displayed in the Text Field and similarly on the click of the Evening button, the message "Good Evening" should be displayed in the Text Field.

Think how to achieve this. Don't worry we are providing the solution in the Figure 4.19. But giving a thought and trying is important.

Let us break the suspense and look at the coding given below:



```
Source Design 
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    jTextField1.setText("Good Evening");
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    jTextField1.setText("Good Morning");
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}
```

Figure 4.19: Code to Display message in a Text Field on the click of a Button

The above code introduces us to a new method called `setText()`. This method is used to change the display text of a component (label, text field or button) during run time. The syntax of this method is given below:



Syntax:

```
component.setText("text")
```

The "text" is the display text to be shown for the mentioned component.

Do you remember how we can change the display text of a component during design time? Look at the Figure 4.20 which displays a list of a few editable properties of a Text Field component and try to find an answer of the above question.

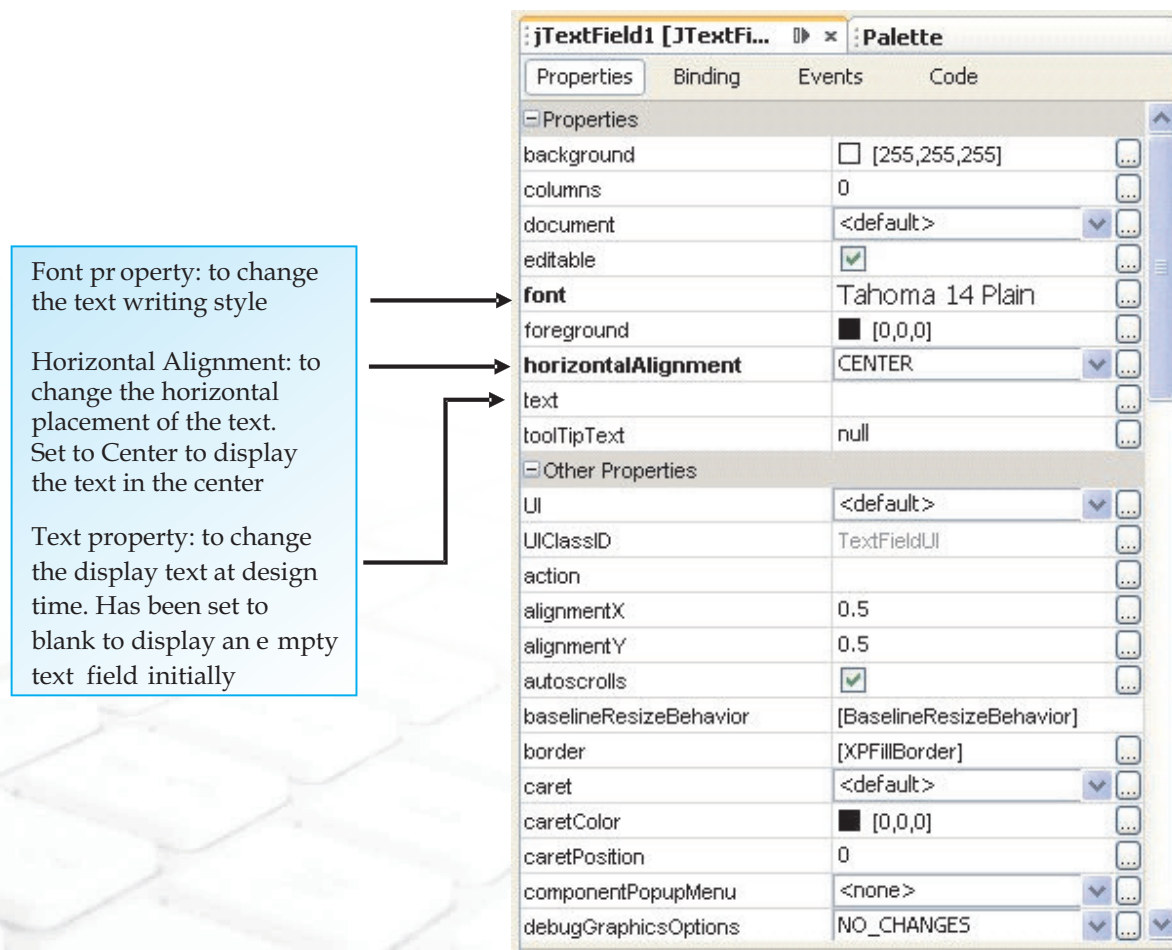


Figure 4. 20 Few Text Field Properties

Using a Text Field Component to Accept Input

In the above example we used a text field to simply display a message but in real life applications, we use a text field to accept input from the user. So in the next example we will use two text fields, one to accept input and a second one to display a message. Let us



first design the form as displayed in the Figure 4.21. The purpose of this form is to accept the name of the user in the Text Field placed at the top and then display a personalized greeting (greeting along with the name of the user) in the Text Field placed at the bottom. Just like there is the `setText()` method to change the display text of a component at run time, there is a `getText()` method to retrieve the display text of a component (label, text field or button) at run time. Think how we can make use of this method in our above stated problem.

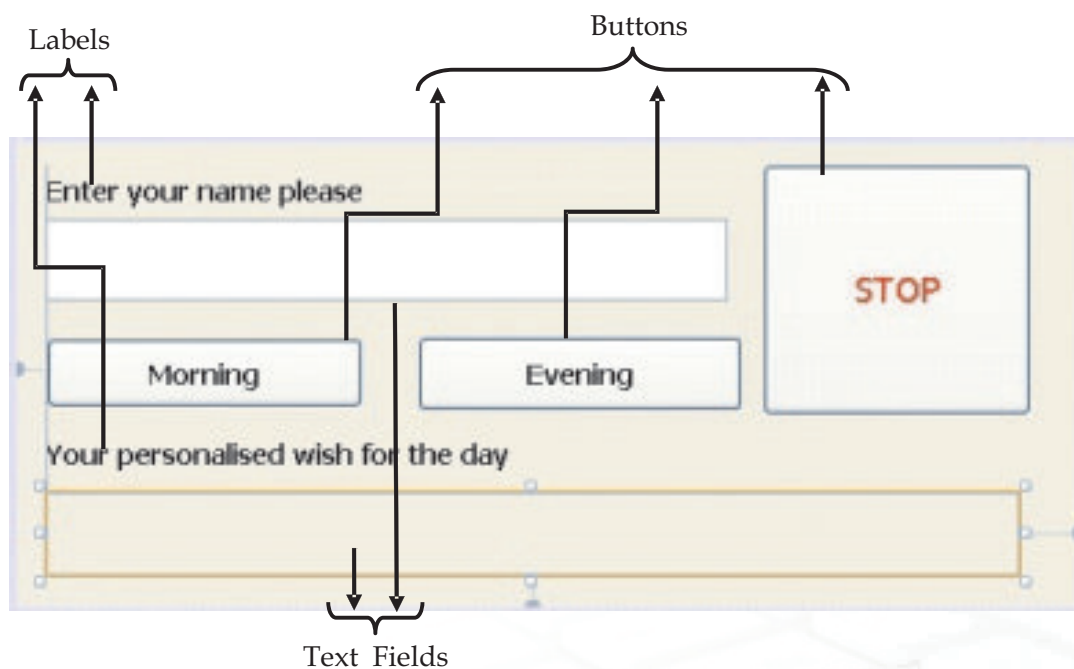
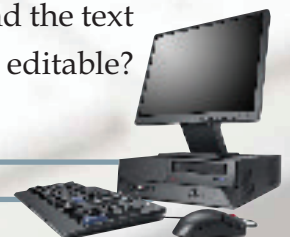


Figure 4.21 Form Design to Display a Personalized Time Based Greeting on the Click of a Button

Observe the Figure 4.21 carefully. What is new for us in this form? First we have used a new component - a label and second is the difference between the two text fields. A label is a component which is used to display simple text or as a label for another component. Can you spot what is the difference between the two text fields? One of them has a white background while the other has the same background colour as the form. The difference in the background colour tells us that one of the text field is editable while the other is not. In simple words editable means that the user can change the text displayed in the text field at run time. The text field at the top has to accept the name of the user and the text field at the bottom has to display the greeting. So which one should be editable?



Obviously the one which has to accept the input should be editable. So, the one with the white background is editable while the other is not. Figure 4.22 displays the properties of both the text fields. Can you guess which property is used to control the editing nature of a text field?

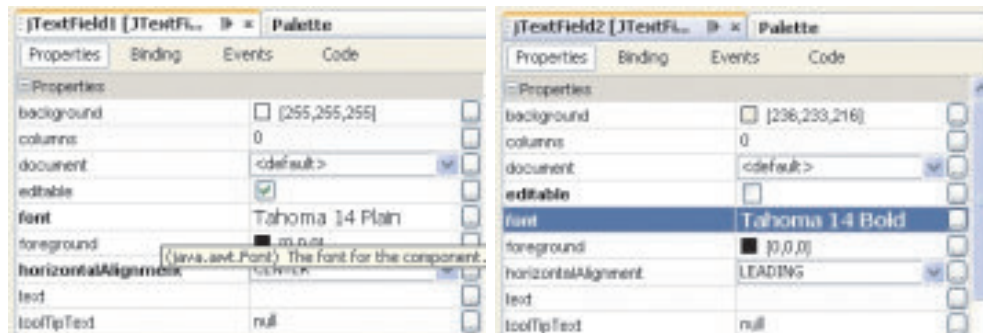


Figure 4.22 Setting Text Field Properties

The editable property is used to control the editing nature of a text field at run time. Therefore the first text Field's check box is selected (indicating that it can be edited at run time) while the second one is non-editable. Now select the label components one by one and change their properties using the Properties window as shown in Figure 4.23

Font property: to change the text writing style

Horizontal Alignment: to change the horizontal placement of the text. Set to Center to display the text in the center of the label

Text property: to change the display text at design time.

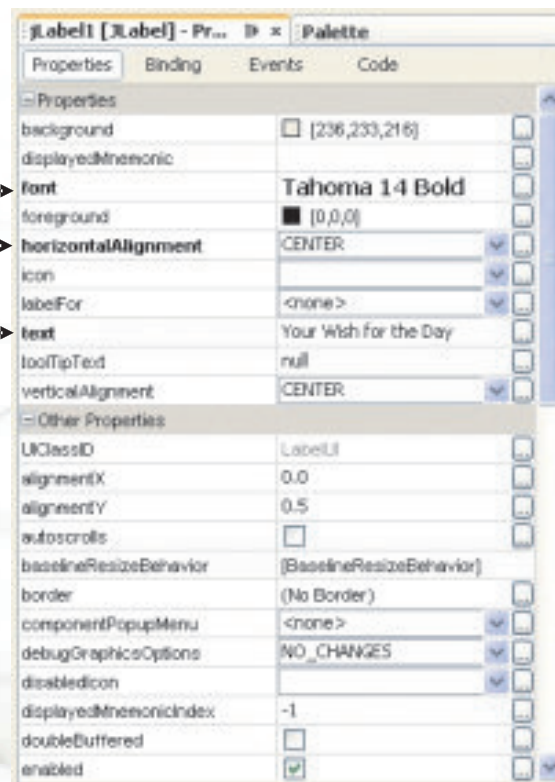


Figure 4.23 Few Properties of the Label Component

After completing the designing of the form, now we are ready to add the code. Remember that we had to use the `getText()` method in our code. Again double click on the three separate buttons one by one to attach relevant code to each one of them. Observe the coding given in Figure 4.24 and try to figure out what's happening.

The code teaches us another useful method - `getText()`. This is used to return the text contained in the referred text component. It is generally used to retrieve the value typed by the user in a textbox or label. The syntax for this method is given below:

Syntax:

`jtextField1.getText()`

This command is used to retrieve the value of the text Field named `jtextField1`.

Let us now understand the code. We want to display the message in the second text field along with the name of the user which has been entered in the first text field.

`jTextField1.getText()`

- ❖ retrieves the name entered by the user in the first text field using `getText()`.

`"Good Morning" + jTextField1.getText()`

- ❖ The message "Good Morning" is concatenated with the name retrieved from the first text field using the `+` symbol.

`jTextField2.setText("Good Morning" + jTextField1.getText())`

- ❖ The display text of the second text field is set to the concatenated message using `setText()`.

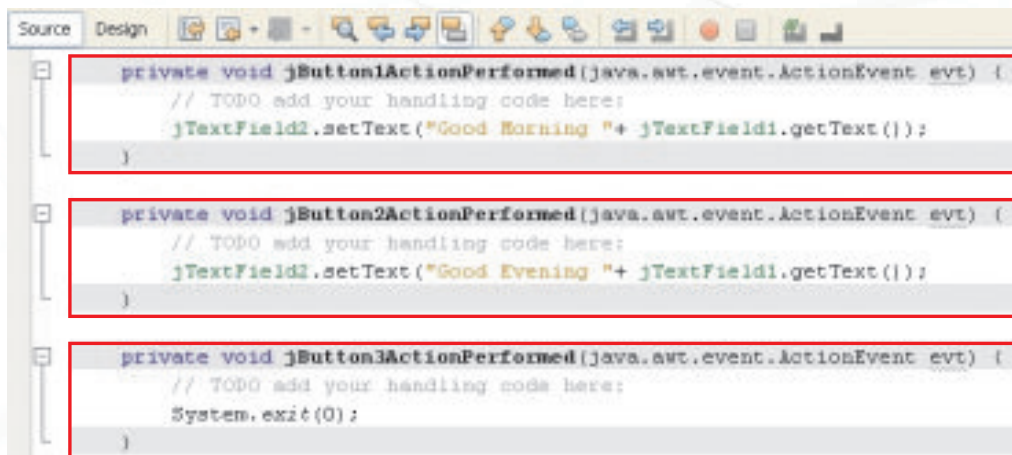
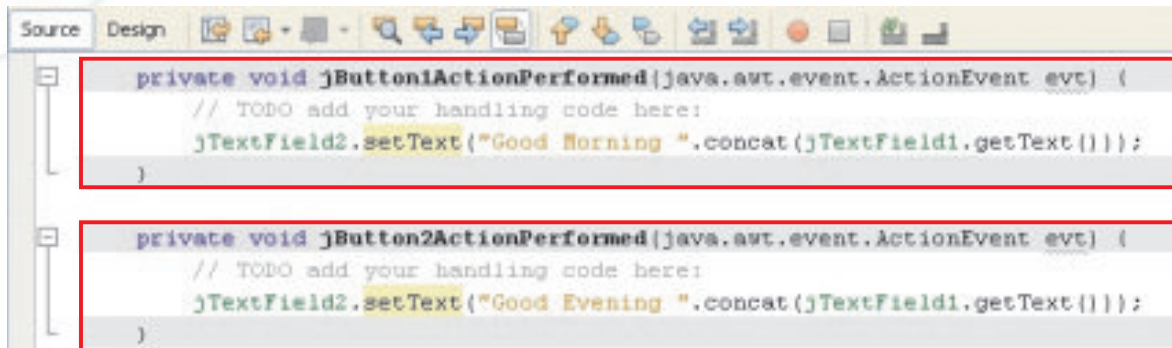


Figure 4.24 Code to Display Personalized Time Based Greeting on Click of a Button using the string concatenator operator (+)



Figure 4.25 displays an alternative method of concatenating the message and the contents of the text field.



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField2.setText("Good Morning ".concat(jTextField1.getText()));  
}  
  
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField2.setText("Good Evening ".concat(jTextField1.getText()));  
}
```

Figure 4.25 Code to Display Personalized Time Based Greeting on Click of a Button using concat() method

This alternate uses the concat() method to add the two strings together. The syntax of this method is:

Syntax:

string1.concat(string2)

This will result in adding the string2 at the end of the string1. For example:

"sham".concat("poo") returns shampoo

and

"to".concat("get").concat("her") returns together

Finally, our code is ready for execution. Figure 4.26 displays the output when the user enters the name and clicks on the Morning button.

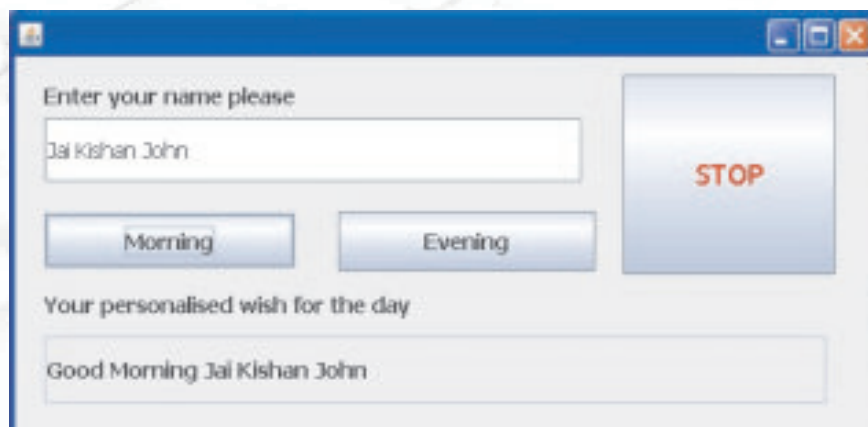


Figure 4.26 Execution of Time Based Personalized Greeting Code

Handling the Radio Button Component

By now we have completely familiarized ourselves with the working of text field, buttons, labels and message box. Let us now delve further and try to explore the utility of other components. Let us first try and modify the above example a bit. Supposing instead of displaying a message, we need to display the title of the user (Mr. or Ms.) along with the name input in the textbox. How to go about it? The simple answer would be to accept the title in a separate textbox and then concatenate it with the name. But do you think it is the right approach? Using the textbox for accepting the title will cause ambiguity thereby making the code complex as we will have to cater to the different inputs. Different users will have different ways of entering the title. Some might write MR. or some might write Mr. or some might write MR (without the dot). Then how do we avoid this ambiguity? A simple solution is to use a radio button component to accept the gender input. Radio buttons are groups of buttons in which, by convention, only one button at a time can be selected. First design the form with the following components:

- ❖ one editable text field to accept the name
- ❖ a group of 2 radio buttons to accept the gender
- ❖ one non-editable text field to display the name along with the title
- ❖ appropriate labels to direct the user

As a first step drag a text field from the Swing Control tab of the Palette. Next drag and place two radio buttons as shown in the following figure. Remember that out of several radio buttons belonging to a group, only one can be selected. Therefore, the next step is to associate the two radio buttons to each other. This is achieved by linking both the radio buttons with a ButtonGroup. For each group of radio buttons, we need to create a ButtonGroup instance and add each radio button to it. It is necessary to associate all the radio buttons in a group to one ButtonGroup. The ButtonGroup takes care of unselecting the previously selected button when the user selects another button in the group. So drag a Button Group component from the Swing Controls tab and drop it anywhere on the form. This is an invisible component which is just used to associate several radio buttons. Now to associate them to same button group, select the first radio button and edit the buttonGroup property of this radio button using the Properties Window as shown in Figure 4.27. Repeat the same procedure for the second radio button of this group to associate them to same button group. Select the same Button Group from the drop down menu in the buttonGroup property for the second radio button.



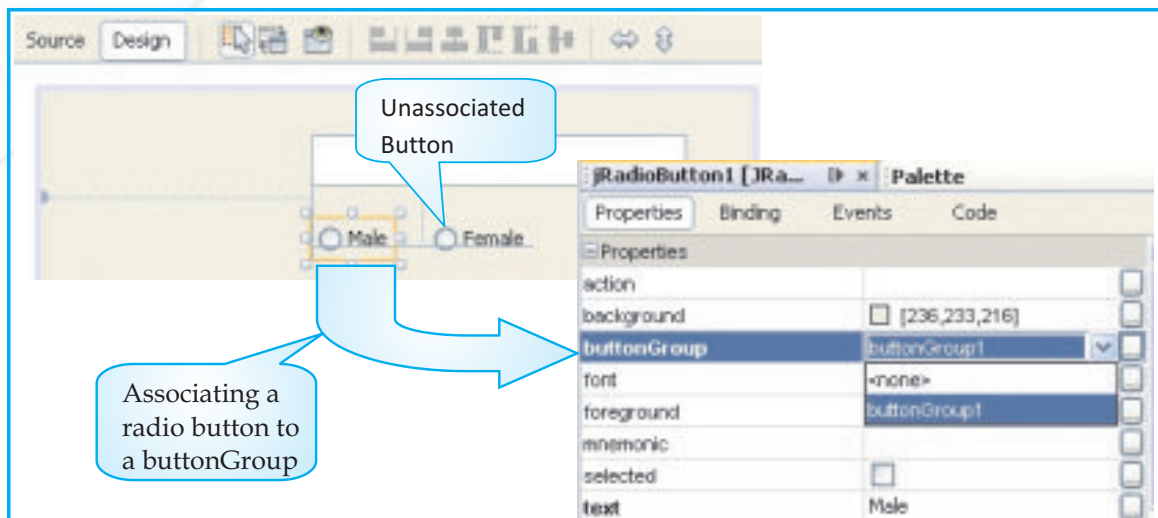


Figure 4.27 Associating First Radio Button with a buttonGroup

After both the radio buttons have been associated together, clicking on any one of them will show an association between them informing us that they belong to a group as shown in Figure 4.28.

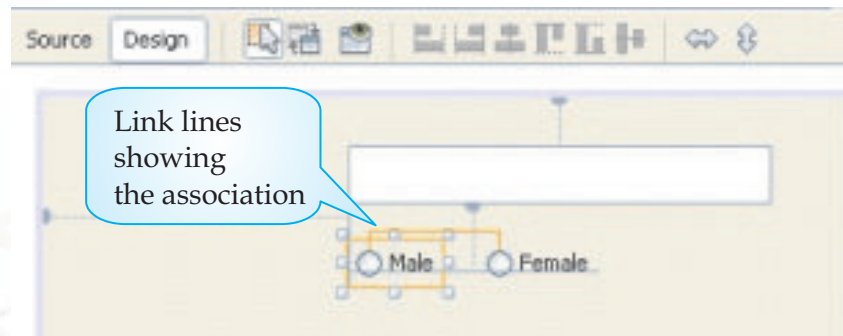


Figure 4.28 Radio Buttons belonging to a Button Group

Add one more non-editable text field to display the name along with the title. Double click on each of the two radio buttons one by one to associate them with the appropriate code displayed in Figure 4.29.



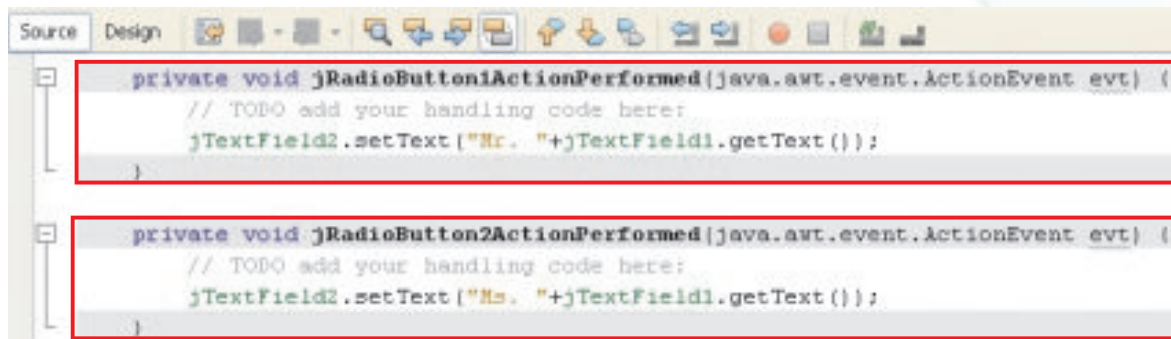


Figure 4.29 Associating Code with the Radio Buttons

Now execute the program and see the output. One sample output is shown in Figure 4.30



Figure 4.30 Sample Execution of Displaying Name with Title

! We should generally initialize a group of radio buttons so that one is selected. However, there is no compulsion regarding this rule - a group of radio buttons can have no initial selection. Once the user has made a selection, exactly one button is selected from then on.

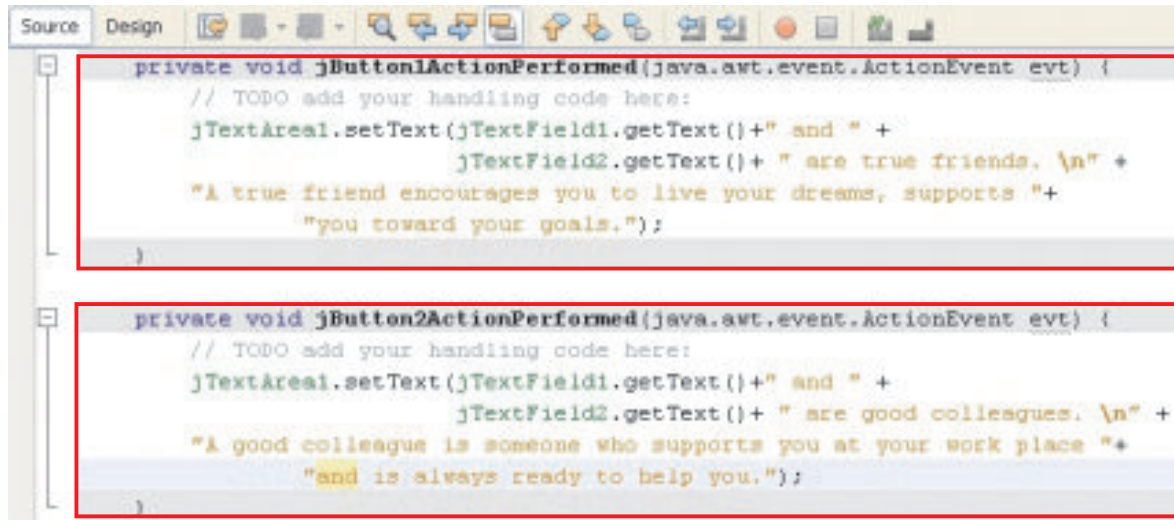
Using the Text Area Component

The text field allows the user to enter a single line of text only. If we want to accept multiline input or want to display multiline output, then what do we do? The Text Area component is a solution for this problem. This component allows us to accept multiline input from the user or display multiple lines of information. This component automatically adds vertical or horizontal scroll bars as and when required during run time. Utilizing the concept of Text Area, let us design an application which accepts names of two people and displays a short message about Friendship or Colleagues depending upon which button is clicked.



Design the form shown in Figure 4.32(a). One new component - the Text Area has been added while the rest of the components are familiar.

Write the code as shown in Figure 4.31 for the two buttons. Add the code for the STOP button.



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    JTextArea1.setText(jTextField1.getText()+" and " +
        jTextField2.getText()+" are true friends. \n" +
        "A true friend encourages you to live your dreams, supports "+
        "you toward your goals.");
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    JTextArea1.setText(jTextField1.getText()+" and " +
        jTextField2.getText()+" are good colleagues. \n" +
        "A good colleague is someone who supports you at your work place "+
        "and is always ready to help you.");
}
```

Figure 4.31 Code for displaying Multiline Text in a Text Area on the click of a Button

Now observe the Figures 4.32 (a), (b) and (c) carefully. These figures show the sample output of the code given in Figure 4.31. Try to point out the difference in the output pattern.

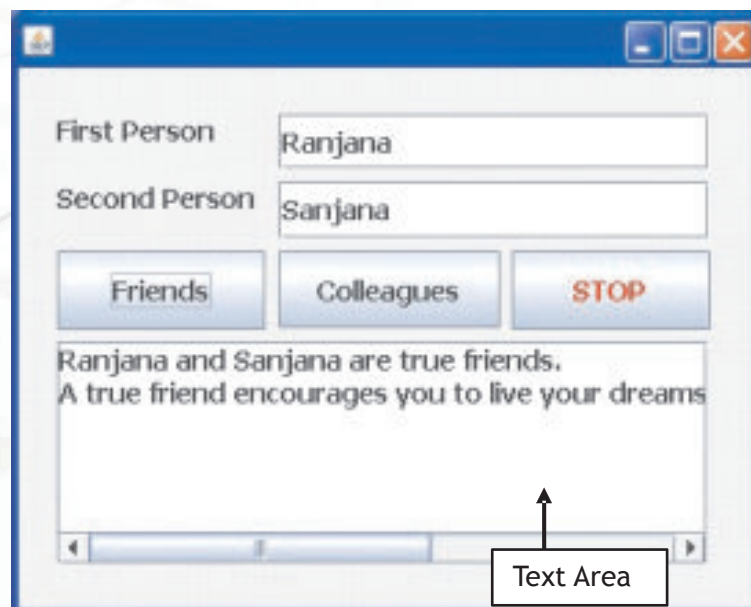


Figure 4.32(a) First Sample Run of the Text Area Application

The main difference in the three sample output windows is about the way the text has been displayed. In the first sample, the text is displayed without any wrapping and so only part of the message is visible. To see the remaining part of the message, the user will have to use the scrollbars. In the second sample, the text is displayed using line wrap without any word wrap. Due to this words are broken although the entire message is visible.

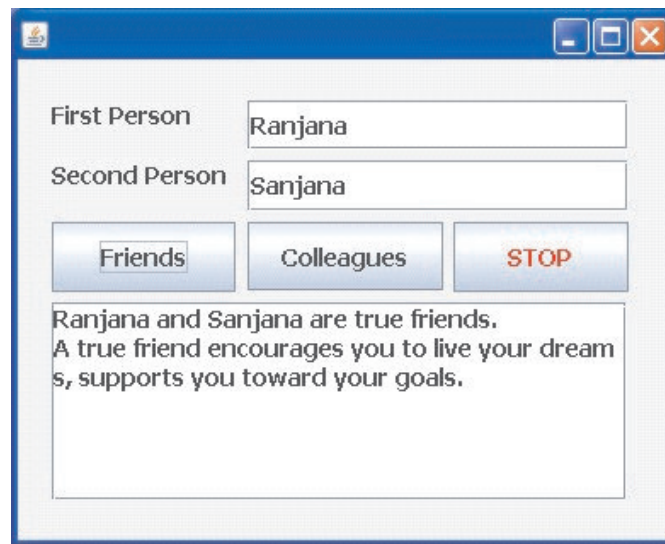


Figure 4.32(b) Second Sample Run of the Text Area Application



Figure 4.32(c) Third Sample Run of the Text Area Application

In the third sample, the text is displayed using both line wrap and word wrap which ensures that no words are broken and also all the lines are neatly wrapped. Let us first understand the exact meaning of line wrap and word wrap and then we will learn how to change these properties. In text display, line wrap is the feature of continuing on a new line when a line is full, such that each line fits in the viewable window, allowing text to be read from top to bottom without any horizontal scrolling.



On the other hand, **Word wrap** is a feature which allows text to be carried over to the next line so that it will stay within a set of defined margins without breaking the words. These properties can be set at design time using the Properties Window as shown in Figure 4.33

Know more

The '\n' is a non printable character that is used to cause explicit line breaks. It can be used with any of the methods which are used to display text like `setText()` and `showMessageDialog`.

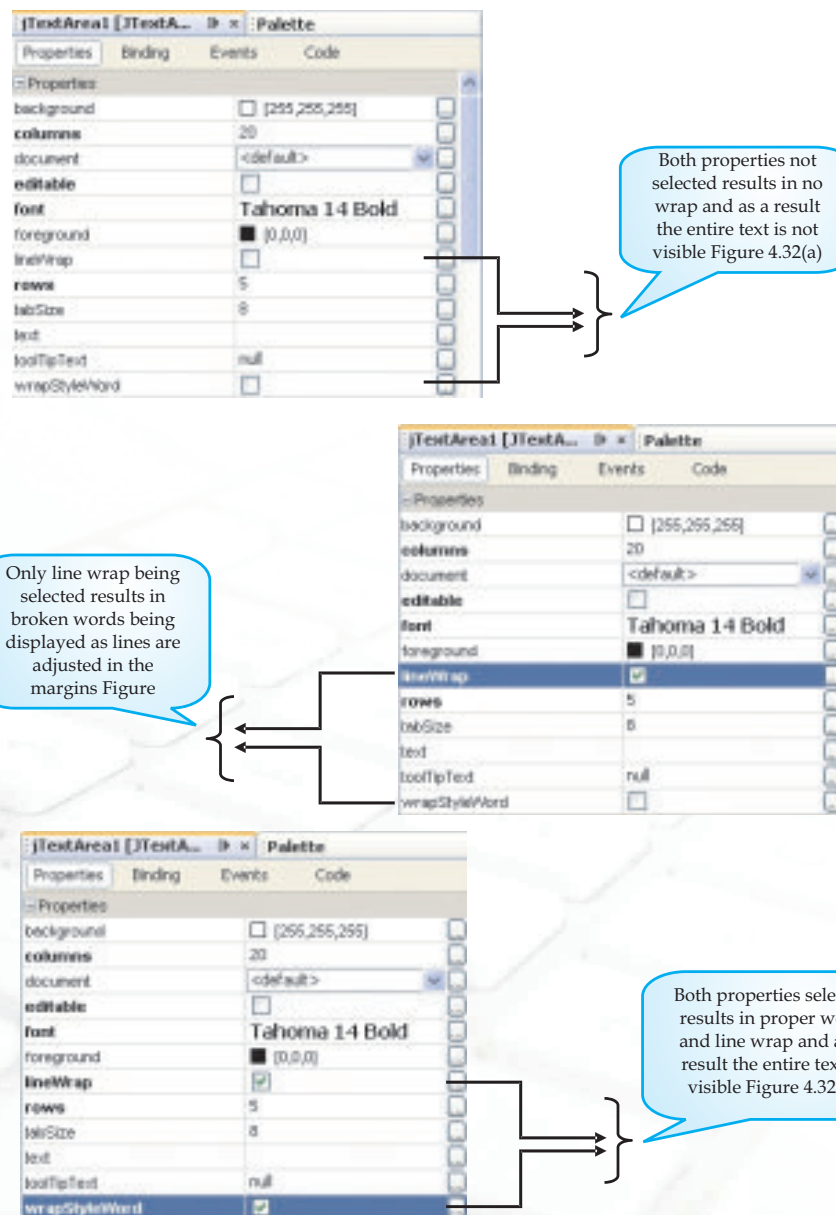


Figure 4.33 Setting the `lineWrap` and `wrapStyleWord` Properties to Study their Effect

Handling a Password Field Component

Now the question arises that what should be done if we want that the text input by the user should not be displayed as characters but as special characters (so that it is not readable by anyone)? The answer is simple. Use the Password Field instead of the normal text field. This component allows confidential input like passwords which are single line. Let us design a simple application which displays a simple message when the user inputs a user name and password. Figure 4.34 displays the sample run of the application. Remember that no checking is being done, rather a simple message is to be displayed on the click of the LOGIN button and the application should be terminated on the click of the CANCEL button. (Validations will be dealt with in Class XII).

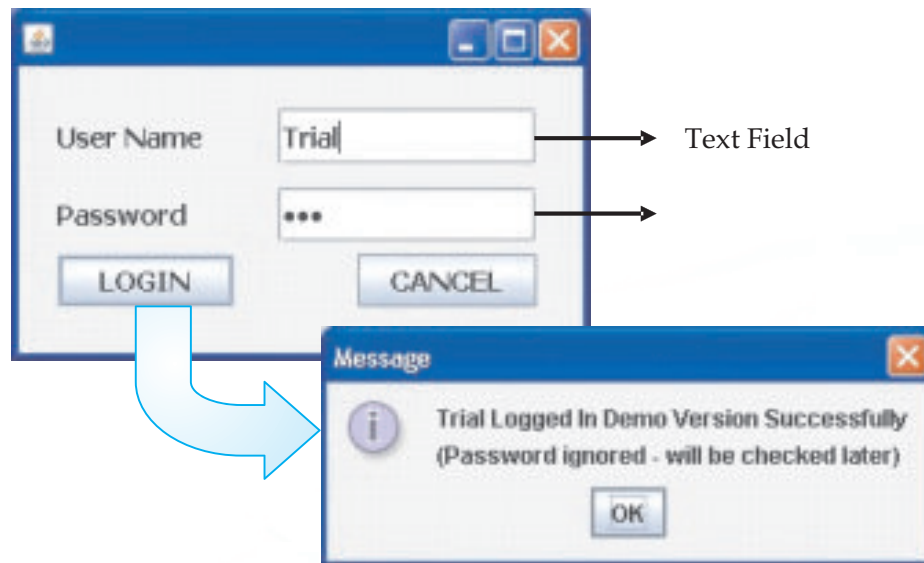


Figure 4.34 Sample run of the Password Application

Figure 4.35 displays the code to display the message on the click of the LOGIN button. Add the code for the CANCEL button also yourself.

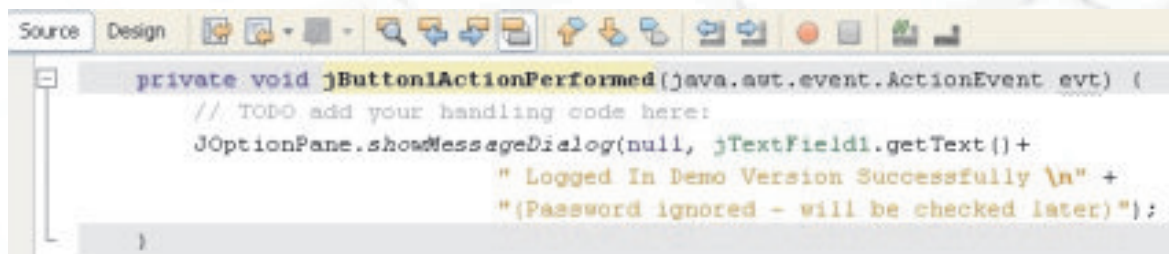


Figure 4.34 Sample run of the Password Application



Writing Code for Performing Simple Calculations Involving Integers

Hope all the components explained above are clear along with their usage and properties. In all the previous examples we have been doing text manipulation. Let us now do some simple computations and calculations. Design the form as shown in Figure 4.36. The form components are:

- ❖ 1 editable text field to input the price per Apple
- ❖ 1 non-editable text field to display the amount to be paid
- ❖ 3 buttons, one for calculating and displaying the price of one dozen apples, one for calculating and displaying the price of two dozen apples and one to exit out of the application.
- ❖ 2 labels to guide the user what information is to be added.

Figure 4.36 Price Calculator

Let us first analyze the problem so that we can easily write the one line code required for all three buttons.

- ❖ The first button with the "One Dozen" display text has to calculate the price of one dozen apples and display it in the second text field. To calculate the price of one dozen apples, we need to know the price of one apple. This is given in the first text field. So we need to retrieve the value of the first text field. Which method should we use to retrieve this value? Think. After retrieving the value we will simply multiply it by 12 and display the answer in the second text field. Which method should we use to display the answer? Think.



- ❖ The second button with the "Two Dozen" display text has to calculate the price of two dozen apples and display it in the second text field. So the process remains similar to the first button but only while calculating we will multiply the price of one apple by 24 and display the answer in the second text field.
- ❖ The third button with the "STOP" display text has to simply end the application. Which method should we use for this purpose?

If you have been able to give an answer for the above three questions then enter the code for each button separately as shown in Figure 4.37 else go back and try to revise a little.

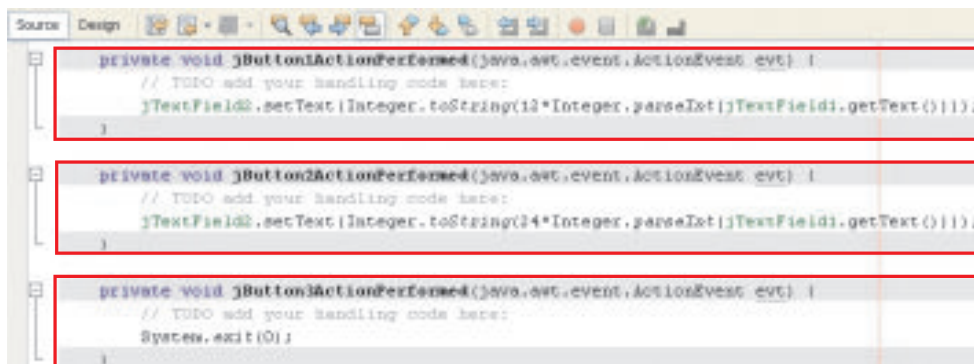


Figure 4.37 Code for the Price Calculator Application

The code has introduced us to two new methods:

- ❖ Integer.toString() - used to convert an Integer value to String type
- ❖ Integer.parseInt() - to convert a value to Integer type

We are already familiar with setText() and getText() so now we are ready to understand the code.

jTextField1.getText()

- ❖ retrieves the value entered by the user in the first text field using getText(). This value by default is treated as a string i.e. a group of characters and not as a number

12*Integer.parseInt(jTextField1.getText())

- ❖ The string value needs to be converted to an integer number and this is achieved using the parseInt() method. After converting it to a number it is multiplied by 12




```
Integer.toString(12 * Integer.parseInt(jTextField1.getText()))
```

- ❖ The value calculated is a number which is to be displayed in a text field. So before displaying it needs to be converted to a string type and this is achieved using the toString() method.

```
jTextField2.setText(Integer.toString(12 * Integer.parseInt(jTextField1.getText())))
```

- ❖ The converted value needs to be displayed in the second text field. This is achieved using the setText() method.

Now test your code and enjoy the result of your hardwork. A sample run is shown in Figure 4.36.

Writing Code for Performing Simple Calculations Involving Numbers with decimals

Let us now do some simple calculations involving numbers with decimals (called double in java). Design the form as shown in Figure 4.38.

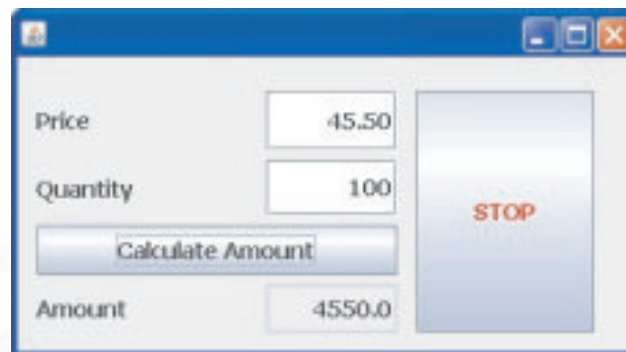


Figure 4.38 Amount Calculator using Numbers with Decimals

The form components are:

- ❖ 2 editable text fields to input the price and quantity
- ❖ 1 non-editable text field to display the amount to be paid
- ❖ 2 buttons, one for calculating and displaying the amount payable and one to exit out of the application.
- ❖ 3 labels to guide the user what information is to be input and displayed

Let us first analyze the problem so that we can easily write the single line code required for the Calculate Amount button.



- ❖ The first button with the "Calculate Amount" display text has to calculate the total amount to be paid and display it in the third text field at the bottom of the screen. To calculate the amount, we need to know the price of one item and also the quantity of the item purchased. These values are given in the first and the second text field respectively. So we need to retrieve these value from the two text fields. Remember that these values will be by default string type so we need to convert them to a suitable type (in this case double) so as to be able to perform calculations on them. After retrieving the value we will simply multiply the two values and convert the value so obtained to string and display the answer in the third text field.

Now add the code for the first button as given in the Figure 4.39

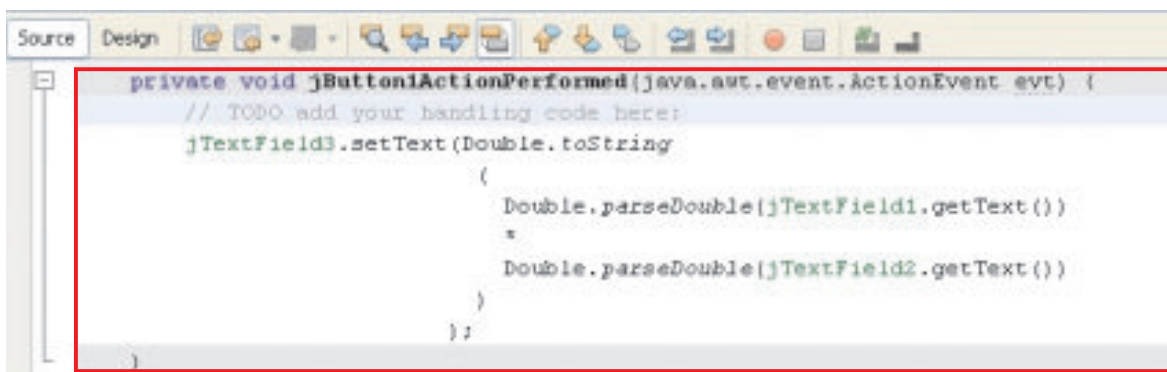


Figure 4.39 Code for the Amount Calculator Using Numbers with Decimals

The code has introduced us to one new method:

- ❖ `Double.parseDouble()` - to convert a value to Double type

We are already familiar with `setText()`, `getText()` and `toString()` so now we are ready to understand the code.

`jTextField1.getText()` and `jTextField2.getText()`

- ❖ retrieves the value entered by the user in the first and second text fields respectively using `getText()`. These values by default are treated as strings i.e. a group of characters and not as numbers

`Double.parseDouble(jTextField1.getText())` and

`Double.parseDouble(jTextField2.getText())`



- ❖ The string values need to be converted to numbers with decimals and this is achieved using the `parseDouble()` method. After converting both the values they are multiplied to get the total amount payable.

```
Double.toString(Double.parseDouble(jTextField1.getText())  
* Double.parseDouble(jTextField2.getText()))
```

- ❖ The value calculated is a number with decimals which is to be displayed in a text field. So before displaying it needs to be converted to a string type and this is achieved using the `toString()` method.

```
jTextField3.setText(Double.toString(Double.parseDouble(jTextField1.getText())  
* Double.parseDouble(jTextField2.getText()))
```

- ❖ The converted value is displayed in the third text field using the `setText()` method.

Now before proceeding to the next chapter let us quickly recap the relation between a Project, Form and Components. Remember each project can have multiple forms and this fact is clear from the Projects window as shown in Figure 4.40.

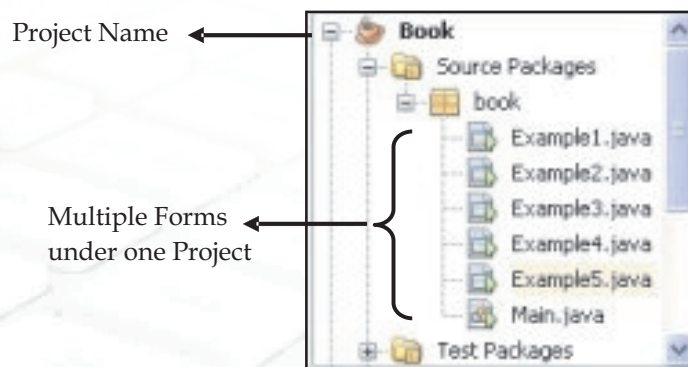


Figure 4.40 Project Window Showing Multiple Forms

Further each form can have one or more elements - some of which may be visible and some invisible. The visible components are all shown under the Frame Component and the non-visible components are part of Other components. The relation of these components is clear from the Inspector window as shown in Figure 4.41



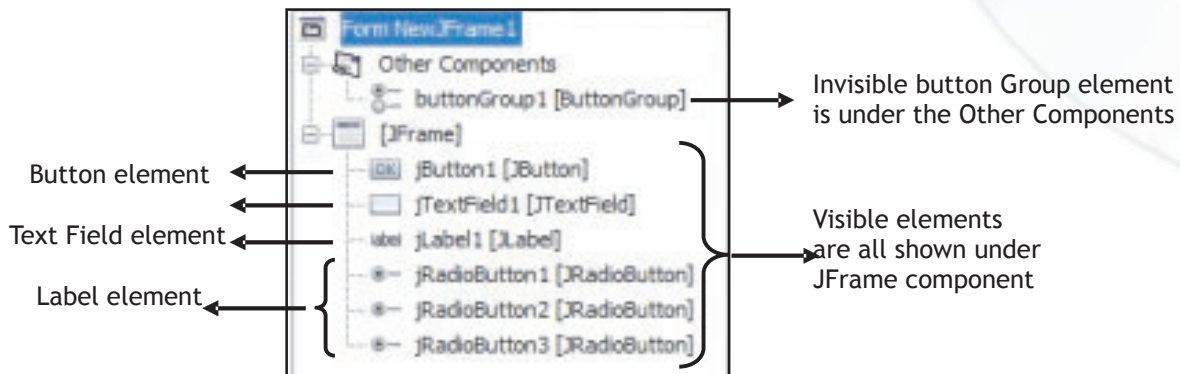


Figure 4.41 Inspector Window Showing Different Components

As we have learnt above, each application is treated as a Project in Netbeans and it can have one or more forms. Each form can have one or more components and this relation between a Project, form and components is depicted in Figure 4.42.

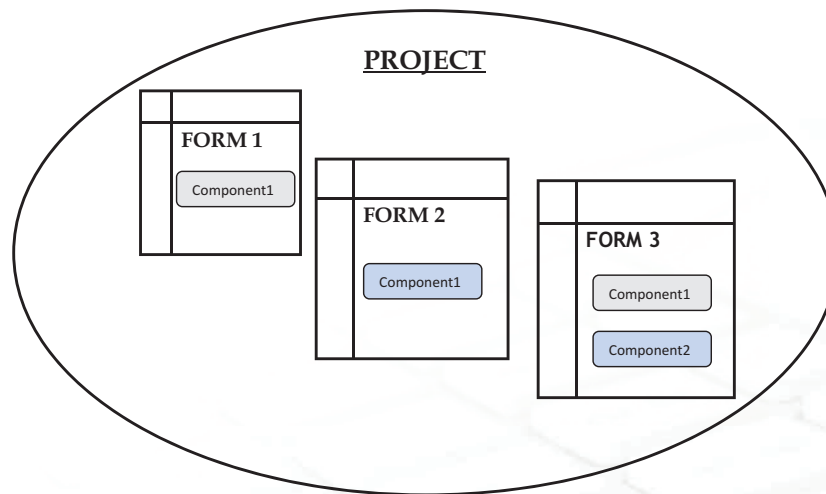


Figure 4.42 Relation Between Project, Form and Components

In the last two examples we have used the concept of String, Integer and Double. We know these are different types of values and we need to understand them further. The next chapter aims at dealing with these in detail. In the process the next chapter will also introduce us to the new components like radio buttons and button group.



Summary

- ❖ Forms are used to accept data (input) and submit data to an external agent for processing.
- ❖ A project in Netbeans acts as a storage place for all the forms and codes created in one application
- ❖ A JFrame Form acts as a container to place other components like the button, text field and text area
- ❖ The Palette Window contains a customizable list of available components containing tabs for JFC/Swing, AWT, and JavaBeans components, as well as layout managers.
- ❖ The Swing Controls can be differentiated from other components as they are preceded with a 'j' like jButton and jTextField
- ❖ The Source tab is used to view the Java source code
- ❖ The Design tab is used to design the form
- ❖ The Properties Window is used to display and edit the attributes of the currently selected component
- ❖ To execute a single file of an application press Shift +F6 or select Run>Run File
- ❖ GUI is an acronym for Graphical User Interface which is an interface that allows us to interact with the various components through visual elements including pictures, graphical icons, symbols and visual indicators
- ❖ The Netbeans IDE consists of The Design Area, the Inspector Window, the Palette and the Properties Window
- ❖ IDE is an acronym for Integrated Development Environment which is a work environment that integrates all tools necessary for Application Development and makes them available as part of one environment



- ❖ A Desktop application creates a template that provides basic application infrastructure such as a menu bar, persisting of window state, and status bar. With this template, you can also generate code to create a GUI interface for a database table (which we will learn in class XII).
- ❖ The various swing components learnt include a button, text field, label, text area, radio button, password field
- ❖ All radio buttons working together must be associated with a single ButtonGroup. The ButtonGroup is an invisible component
- ❖ The general syntax of executing any method is:

`object.method(arguments)`

For example:

1. `Integer.parseInt("10")`

In this example Integer is the object, `parseInt()` the method and 10 is the argument supplied.

2. `textField1.setText("Welcome")`

In this example `textField1` is the object, `setText()` the method and "Welcome" is the argument supplied.

- ❖ The `concat()` method or the string concatenation symbol(+) may be used to add two strings together
- ❖ A brief summary of all the methods learnt in this chapter is given in the table below:



Method	Syntax	Usage
exit() application	System.exit(0)	To successfully terminate an application
showMessageDialog()	JOptionPane.showMessageDialog (parentComponent,message)	To display a specified message in a dialog box
setText()	component.setText("text")	To change the display text of a component (label, text field or button) during run time
getText()	component.getText()	To retrieve the display text of a component (label, text field or button) at run time
concat()	string1.concat(string2)	To concatenate (add) string2 at the end of the string1
toString()	Integer.toString(number)	To convert an Integer value to String type
parseInt()	Integer.parseInt(string)	To convert a string value to Integer type
parseDouble()	Double.parseDouble(string)	To convert a string value to type Double



Multiple Choice Questions

1. **The Form is designed in the**
 - a) Inspector window
 - b) Design window
 - c) Palette window
 - d) Properties window
2. **The Swing Controls components are contained in the**
 - a) Design window
 - b) Inspector window
 - c) Properties window
 - d) Palette window
3. **The most suitable component to accept multiline text is:**
 - a) Text Field
 - b) Password Field
 - c) Text Area
 - d) All of the above
4. **What will be the output of the following command?**
`Learning.concat("Java")`
 - a) Learning Java
 - b) LearningJava
 - c) JavaLearning
 - d) Java Learning
 - e) Will result in an error



5. What will be the output of the following command?

```
"Learning".concat("Java")
```

- a) Learning Java
- b) LearningJava
- c) JavaLearning
- d) Java Learning

Exercises

1. Explain the following terms:

- a) IDE
- b) Inspector Window
- c) Form

2. Explain the usage of the following methods with the help of an example:

- a) setText()
- b) toString()
- c) concat()

3. Differentiate between:

- a) Text field and Text area components
- b) Text field and Password field components
- c) parseInt() and parseDouble() methods
- d) Design and Source tabs

Lab Exercises

- a) Design a GUI desktop application in java to accept the name and favourite sport in two text fields and display an appropriate message including the name and favourite sport in a dialog box using the concat() method. The application must have an exit button to end the application and appropriate labels.



- b) Design a GUI desktop application in java to accept age category using radio buttons and display an appropriate age based message in a text area on selection of a radio button. The application must have an exit button to end the application and appropriate labels.
- c) Design a GUI desktop application in java to accept weight in Kilograms in a text field and convert it into grams and milligrams on the click of two separate buttons. Display the result in a second text field. The application must have an exit button to end the application and appropriate labels.
- d) Design a GUI desktop application in java to accept temperature in Celsius in a text field and display temperature in Fahrenheit in another text field on the click of a button. The application must have an exit button to end the application and appropriate labels.

