

UNIT 5: FUNDAMENTALS TO JAVA PROGRAMMING

Chapter -1 Understand Integrated Development Environment (NETBEANS)

Learning Objectives

- Identify, name and state the usage of the different components of the NetBeans IDE.
- Identify and name the various methods and properties associated with the various form controls

Introduction

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. This chapter deals with teaching the basic process of designing forms in Netbeans and using them to perform simple manipulations using Java.

NetBeans ID

NetBeans IDE is used to create java applications very easily using the efficient GUI builder. It allows us to develop applications by dragging and positioning GUI components from a palette onto a container. The GUI builder automatically takes care of the correct spacing and alignment of the different components relative to each other. Let us go through the different components of the NetBeans IDE (Refer to Fig 5.1):

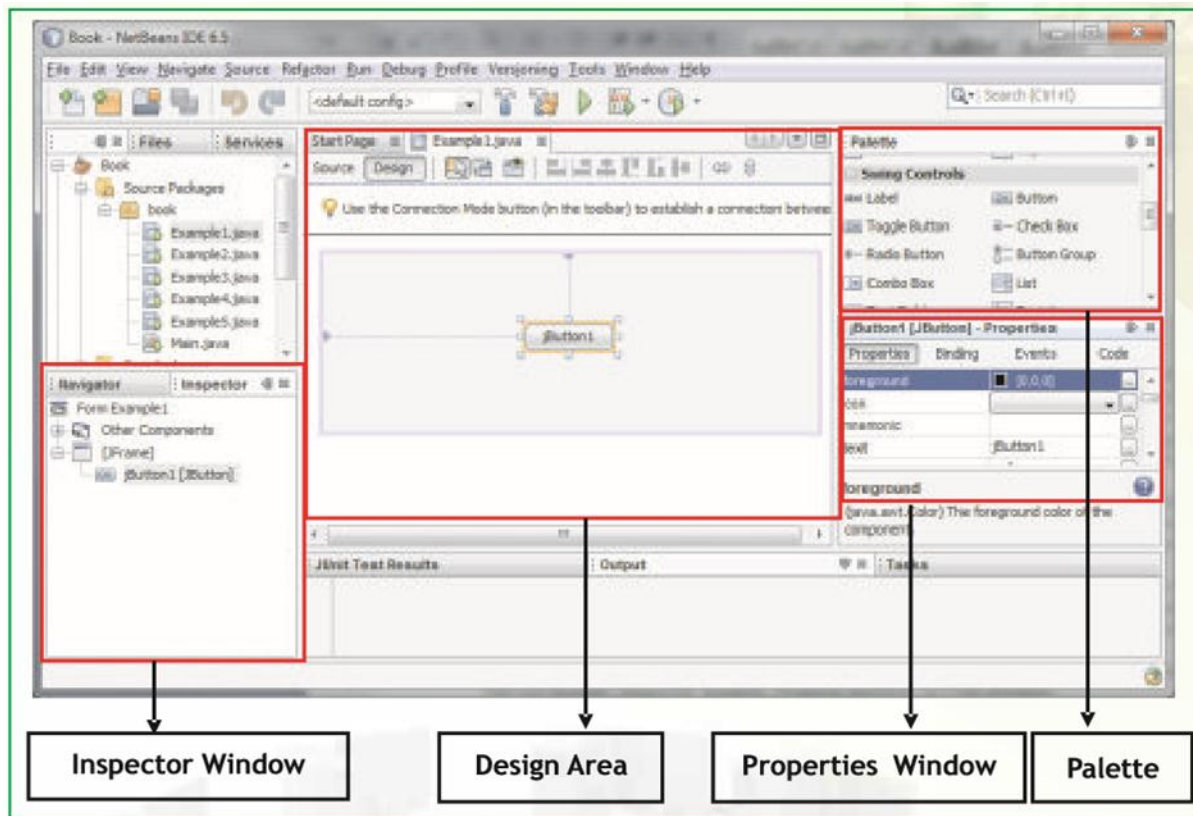


Figure 5.1 NetBeans IDE

1. Title Bar
2. Menu Bar with pull down menus
3. Toolbars
4. GUI builder: It is an area to place components on the form visually. There are two views of the GUI builder- the Design View and the Source View. We can switch over from one view to another by simply clicking on the source and design tabs directly above the Design Area.
5. Palette: Palette contains controls or components used to create GUI applications.
6. Inspector Window: This window is used to display a hierarchy of all the components or controls placed on the current form.
7. Properties Window: Using this window we can make changes in the properties of currently selected control on the form.
8. Code Editor Window: - It is the area where we write code for our java application.

Components

COMPONENTS (ALSO known as "widgets") are the basic interface elements the user interacts with: JLabels, JButtons, JTextFields etc. Components are placed on a container (like the JFrame). There are two types of controls (Refer to Figure 5.2):

- **Parent or container controls:** They act as a background for other controls. For example-Frame. When we delete a parent control, all its child controls get deleted. When we move a parent control all its child controls also move along with it.
- **Child controls:** controls placed inside a container control are called child controls. For example-Text Field, Label, Button etc.

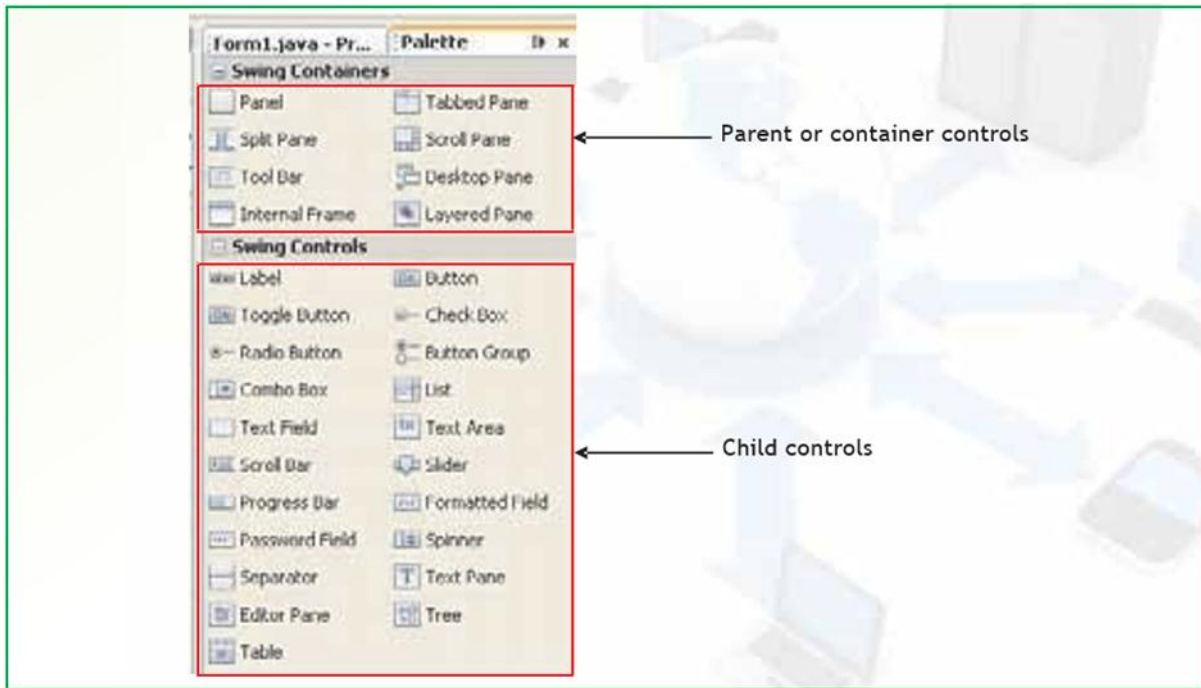


Figure 5.2 Parent and Child controls

Creating a New Project

The steps to create a new project are:

1. Select New Project from the File menu. You can also click the New Project button in the IDE toolbar.
2. In the Categories pane, select the General node. In the Projects pane, choose the Java Application type. Click the Next button.
3. Enter the name of the project in the Project Name field and specify the project location. Do not create a Main class here.
4. Click the Finish button.

Let us recap the relation between a Project, Form and Components. Each application is treated as a Project in NetBeans and each project can have one or multiple forms and this fact is clear from the Projects window as shown in Figure 5.3.

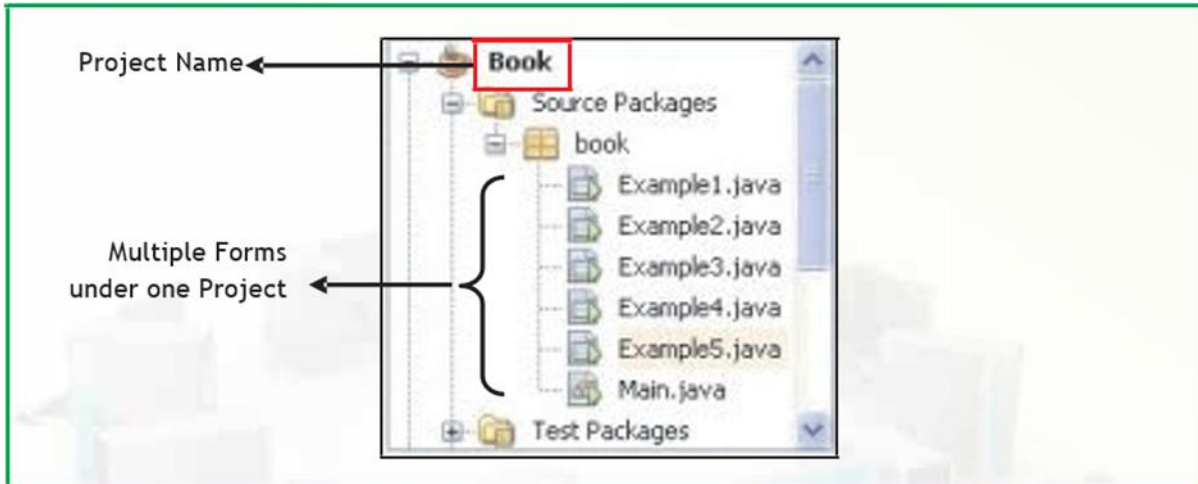


Figure 5.3 Project Window Showing Multiple Forms

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.

We use the drag and drop feature of NetBeans to place components on the form to design an effective interface for our applications. The first step that we undertook while designing our applications was adding a new JFrame form. The JFrame is a window with title, border, (optional) menu bar and is used to contain all other components placed by the user on the form. Some of the properties of the JFrame form are defaultCloseOperation and Title(Refer Figure 5.4).

Property	Description
defaultCloseOperation	Sets action to be performed when the user attempts to close the form.
Title	Sets the text to be displayed in the Title bar of the form window.

Figure 5.4 Properties of the JFrame Form

Any component of GUI front-end (the form itself and the swing containers and controls placed in the form) of an application is an object. Each of these objects belongs to its corresponding class predefined in Java. For example, a form is an object of JFrame class, all the textfields are objects of JTextField class, and so on. Each object has some properties, methods, and events associated with it using which you can control the object's appearance and behaviour.

Properties of an object are used to specify its appearance on the form. For example, to set the **background** colour of a textfield you change its background property; to set its **font** you change its font property; and so on.

Methods are used to perform some action on the object. For example to display something in a textfield you can use its **setText()** method, to extract the contents of a textfield you can use its **getText()** method. Methods can be divided into two categories- *getters and setters*.

- Getters are the methods which extract some information from the object and return it to the program. Getters start with the word get. Examples of getters are: **getText()**, **getForeground()**, **getModel()**, **isEditable** etc.
- Setters are the methods which set some properties of the object so that the object's appearance changes. Setters start with the word set. Examples of setters are: **setText()**, **setForground()**, **setModel()** etc.

Events are the actions which are performed on controls. Examples of events are: **mouseClick**, **mouseMoved**, **keyPressed** etc. When the user performs any action on a control, an event happens and that event invokes (sends a call to) the corresponding part of the code and the application behaves accordingly.

After setting the properties of the JFrame we can start placing components like JButton on the JFrame form. A button is a component that the user presses or pushes to trigger a specific action. When the user clicks on the button at runtime, the code associated with the click action gets executed. The various methods and properties associated with the JButton are summarized in Figure 5.5.

Property	Description
Background	Sets the background color.
Enabled	Contains enabled state of component - true if enabled else false.
Font	Sets the font.
Foreground	Sets the foreground color.
horizontal alignment	Sets the horizontal alignment of text displayed on the button.
Label	Sets the display text.
Text	Sets the display text
Method	Description
getText()	Retrieves the text typed in JButton. String result=<button-name>.getText();

setEnabled	Enables or disables the button. <button-name>.setEnabled(boolean b);
setText()	Changes the display text at runtime. <button-name>.setText(String text);
setVisible	Makes the component visible or invisible - true to make the component visible; false to make it invisible. <button-name>.setVisible(boolean aFlag);

Figure 5.5 Properties and Methods of the JButton

We developed simple real life applications wherein on the click of the button we accepted the data from the user in the JTextField and after processing the data the result was displayed in a JTextField or a JLabel. JTextField allows editing/displaying of a single line of text. JTextField is an input area where the user can type in characters whereas a JLabel provides text instructions or information. It displays a single line of read-only text, an image or both text and image. The various methods and properties associated with the JTextField and JLabel are summarized in Figure 5.6 and 5.7 respectively.

Property	Description
Background	Sets the background color.
Border	Sets the type of border that will surround the text field.
editable	If set true user can edit textfield. Default is true.
enabled	Contains enabled state of component- True if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
horizontalAlignment	Sets the horizontal alignment of text displayed in the textField.
text	Sets the display text
toolTipText	Sets the text that will appear when cursor moves over the component.

Method	Description
---------------	--------------------

getText()	Retrieves the text in typed in jTextField. String result=<textfield-name>.getText();
isEditable()	Returns true if the component is editable else returns false. boolean b=<textfield-name>.isEditable();

isEnabled()	Returns true if the component is enabled,else returns false. boolean b =<textfield-name>.isEnabled();
setEditable	Sets whether the user can edit the text in the textField. true if editable else false. <textfield-name>.setEditable(boolean b);
setText()	Changes the display text at runtime. <textfield-name>.setText(String t);
setVisible()	Makes the component visible or invisible - true to make the component visible; false to make it invisible. <textfield-name>.setVisible(boolean b);

Figure 5.6 Properties and Methods of the jTextField

Property	Description
background	Sets the background color.
enabled	Contains enabled state of component- true if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
horizontalAlignment	Sets the horizontal alignment of text displayed in the component.
text	Sets the display text

Method	Description
---------------	--------------------

getText()	Retrieves the text in typed in JLabel. String result=<label-name>.getText();
isEnabled()	Returns true if the component is enabled,else returns false. boolean b=<label-name>.isEnabled();
setText()	Changes the display text at runtime. <label-name>.setText(String t);
setVisible()	Makes the component visible or invisible - true to make the component visible; false to make it invisible. <label-name>.setVisible(boolean b);

Figure 5.7 Properties and Methods of the JLabel

The Text Area 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. The various methods and properties associated with the JTextArea are summarized in Figure 5.8.

Property	Description
background	Sets the background color.
columns	Sets number of columns preferred for display.
editable	If set true user can edit textfield. Default is true.
enabled	Contains enabled state of component- true if enabled else false.
font	Sets the font.

foreground	Sets the foreground color.
lineWrap	Indicates whether line of text should wrap in case it exceeds allocated width.(Default is false)
rows	Sets number of rows preferred for display.
text	Sets the display text

wrapStyleWord	Sends word to next line in case lineWrap is true and it results in breaking of a word, when lines are wrapped.
---------------	--

Method	Description
append()	Adds data at the end. <textarea-name>.append(String str);
getText()	Retrieves the text in typed in JTextArea. String str = <textarea-name>.getText();
isEditable()	Returns true if the component is editable else returns false. boolean b = <textarea-name>.isEditable();
isEnabled()	Returns true if the component is enabled, else returns false. boolean b = <textarea-name>.isEnabled();
setText()	Changes the display text at runtime. <textarea-name>.setText(String t);

Figure 5.8 Properties and Methods of the JTextArea

The jPasswordField component is used to enter confidential input like passwords which are single line. We can suppress the display of input as this component allows us to input confidential information like passwords. Each character entered can be replaced by an echo character. By default, the echo character is the asterisk, *. The properties of jPasswordField are summarized below:

Property	Description
background	Sets the background color.
font	Sets the font.
foreground	Sets the foreground color.
text	Sets the display text
echoChar	Sets the character that will be displayed instead of text.

Figure 5.9 Properties of jPasswordField

The radio buttons are used to provide the user several choices and allow him to select one of the choices (the radio buttons belong to a group allowing the user to select single option). But radio buttons occupy a lot of space.

Thus, in case of too many options we can use Combo boxes as they help save space and are less cumbersome to design as compared to radio button. We can use check box and list when we want to display multiple options like selecting favourite sports or ordering multiple food items

in a restaurant.

The list is a preferred option over check box in situations wherever multiple options are required to be selected from a large number of known set of options as they help save space and are less cumbersome to design as compared to check boxes. The properties and methods of `JRadioButton` are summarized below:

Property	Description
background	Sets the background color.
buttonGroup	Specifies the name of the group of button to which the <code>JRadioButton</code> belongs.
enabled	Contains enabled state of component -true if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
label	Sets the display text.
text	Sets the display text.
Selected	Sets the button as selected, if set to true, default is false.

Method	Description
<code>getText()</code>	Retrieves the text displayed by radio button. <code>String str = <radiobutton-name>.getText();</code>
<code>isSelected()</code>	Returns true if the component is checked else returns false. <code>boolean b = <radiobutton-name>.isSelected();</code>
<code>setText()</code>	Changes the display text at runtime. <code><radiobutton-name>.setText(String t);</code>
<code>setSelected()</code>	Checks(true) or unchecks the radio button. <code><radiobutton-name>.setSelected(boolean b);</code>

Figure 5.10 Properties and methods of the `JRadioButton`

`JCheckBox` is a small box like component that is either marked or unmarked. When it is clicked, it changes from checked to unchecked or vice versa automatically. The properties and methods of `JCheckBox` are summarized below:

Property	Description
background	Sets the background color.
buttonGroup	Specifies the name of the group of button to which the jCheckBox belongs.
font	Sets the font.
foreground	Sets the foreground color.
label	Sets the display text.
text	Sets the display text
selected	Sets the check box as selected if set to true, default is false.

Method	Description
getText()	Retrieves the text typed in String str = <checkbox-name>.getText();
isSelected()	Returns true if the component is checked else returns false. boolean b = <checkbox-name>.isSelected();
setText()	Changes the display text at runtime. <checkbox-name>.setText(String t);
setSelected()	Checks(true) or unchecks the checkbox. <checkbox-name>.setSelected(boolean b);

Figure 5.11 Properties and methods of the jCheckBox

jComboBox is like a drop down box - you can click a drop-down arrow and select an option from a list whereas jList provides a scrollable set of items from which one or more may be selected. The properties and methods of jComboBox and jList are summarized below:

Property	Description
background	Sets the background color.
buttongroup	Specifies the name of the group of button to which the jComboBox belongs.
editable	If set true user can edit ComboBox. Default is true.
enabled	Contains enabled state of component- True if enabled else false.
font	Sets the font.

foreground	Sets the foreground color.
model	Contains the values to be displayed in the combobox.
text	Sets the display text
selectedIndex	Sets the index number of the element which should be selected by default.
selectedItem	Sets the selected item in the combobox. selectedItem and selectedIndex are in synchronization with each other.

Method	Description
getSelectedItem()	Retrieves the selected item. Object result = <combobox-name>.getSelectedItem();
getSelectedIndex()	Retrieves the index of the selected item. int result = <combobox-name>.getSelectedIndex();
setModel()	Sets the data model that the combo box uses to get its list of elements. <combobox-name>.setModel (ComboBoxModel aModel);

Figure 5.12 Properties and methods of the JComboBox

Property	Description
background	Sets the background color.
enabled	Contains enabled state of component- true if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
model	Contains the values to be displayed in the list.
selectedIndex	Contains the index value of selected option of the control.

selectionMode	<p>Describes the mode for selecting values.</p> <ul style="list-style-type: none"> - SINGLE (List box allows single selection only) - SINGLE_INTERVAL (List box allows single continuous selection of options using shift key of keyboard) - MULTIPLE_INTERVAL (List box allows multiple selections of options using ctrl key of keyboard)
---------------	---

Method	Description
getSelectedValue()	<p>Returns the selected value when only a single item is selected, if multiple items are selected then returns first selected value. Returns null in case no item selected</p> <p>Object result= <list-name>.getSelectedValue();</p>
isSelectedIndex()	<p>Returns true if specified index is selected.</p> <p>boolean b = <list-name>.isSelectedIndex(int index);</p>

Figure 5.13 Properties and methods of the JList

We use JOptionPane when we want to request information from the user, display information to the user or a combination of both. It requires an import statement at the top of the program.

import javax.swing.JOptionPane;

OR

import javax.swing.*;

Either of them is acceptable. The difference is that the latter will import the entire library as denoted by the star whereas the first statement will just import the JOptionPane library.

Method	Description
showMessageDialog()	<p>Shows a one-button, modal dialog box that gives the user some information.</p> <p>Example :</p> <p>JOptionPane.showMessageDialog(this, "Java and NetBeans");</p>

showConfirmDialog()	<p>Shows a three-button modal dialog that asks the user a question. User can respond by pressing any of the suitable buttons.</p> <p>Example: Confirm= JOptionPane.showConfirmDialog(null,"quit?")</p>
showInputDialog()	<p>Shows a modal dialog that prompts the user for input. It prompts the user with a text box in which the user can enter the relevant input.</p> <p>Example : name= JOptionPane.showInputDialog(this,"Name:");</p>

Figure 5.14 Properties and methods of the JOptionPane

Chapter-2 JAVA Programming

- Introduction to Object Oriented Programming
- To understand the need and usage of variables
- To understand various data types (primitive) and purpose of each data type
- To understand usage of operators (assignment, arithmetic, relational, logical, bitwise)
- To understand how to attach a code with components like JButton, JLabel, JTextField and create a simple application on JFrame
- To understand the use of various components like JTextarea, JRadiobutton, JCheckbox, JPasswordField, JListbox, JComboBox, JTable, JOptionPane, JPanel
- To understand when to use selection statements (if, if else and switch case)

Object Oriented Programming

Object Oriented Programming follows bottom up approach in program design and emphasizes on safety and security of data. It helps in wrapping up of data and methods together in a single unit which is known as data encapsulation. Object Oriented Programming allows some special features such as polymorphism and inheritance. Polymorphism allows the programmer to give a generic name to various methods or operators to minimize his memorizing of multiple names. Inheritance enables the programmer to effectively utilize already established characteristics of a class in new classes and applications.

The major components of Object Oriented Programming are as follows:

1. Class
2. Object
3. Data Members & Methods
4. Access Specifier and Visibility Modes

A class is used to encapsulate data and methods together in a single unit. It helps the programmer to keep the data members in various visibility modes depending upon what kind of access needs to be provided in the remaining part of the application. These visibility modes are classified as private, public and protected. Usually, data members of a class are kept in private or protected visibility modes and methods are kept in the public visibility mode.

An object is an instance of a class that is capable of holding actual data in memory locations.

Class and objects are related to each other in the same way as data type and variables. For example, when we declare float variable named marks, the variable marks can be thought of as an object of type float which can be assumed as the class. If we take another hypothetical case in which Human is a class, Mr. Arun Shah, Mr. Aneek Ram will be the objects of this Human class.

Data Members and Methods:

A class contains data members and methods. As discussed in the above example, Mr. Arun Shah is an object of class Human. The phone numbers retained by Mr. Arun Shah in his brain (memory) will be the data. His eyes, ears, nose and mouth can be considered as various methods which allow Mr. Arun Shah to collect, modify and delete data from his memory.

In real java programming, this data will be required to conform to a specific data type as in char, int, float or double whereas the methods will be a sequence of steps written together to perform a specific task on the data. Carefully observe the illustration given in Figure 5.15 to reinstate the theoretical concepts learnt above.

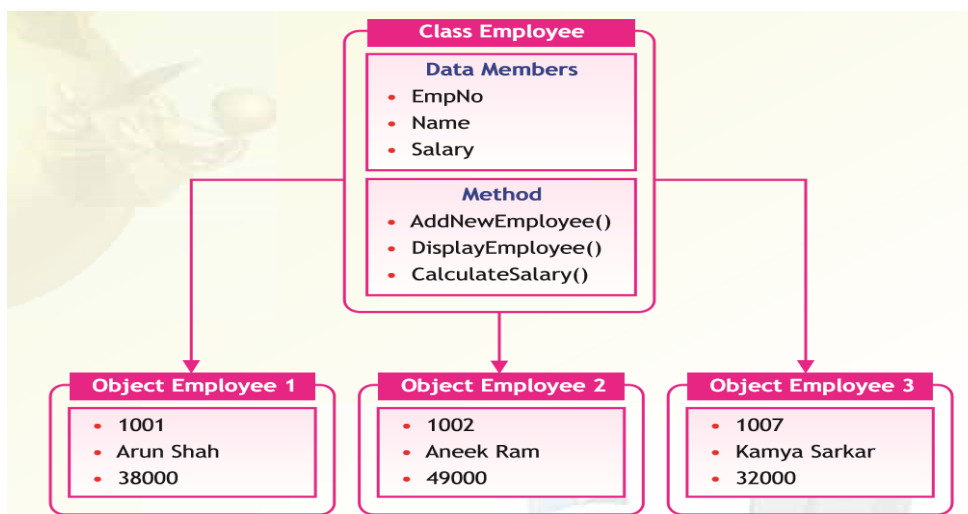


Figure 5.15 Illustration Showing the Class, Object, Members and Methods

The methods of specific classes are able to manipulate data of their respective classes efficiently resulting in better security of data in an Object Oriented Programming paradigm.

The JTextField, JLabel, JTextArea, JButton, JCheckBox and JRadioButton are all classes and the jTextField1, jLabel1, jTextArea1, jButton1, jCheckBox1 and jRadioButton1 components are all objects. The setText(), setEnabled(), pow(), substring() are all methods of different classes. This concept is illustrated in Figure 5.16.

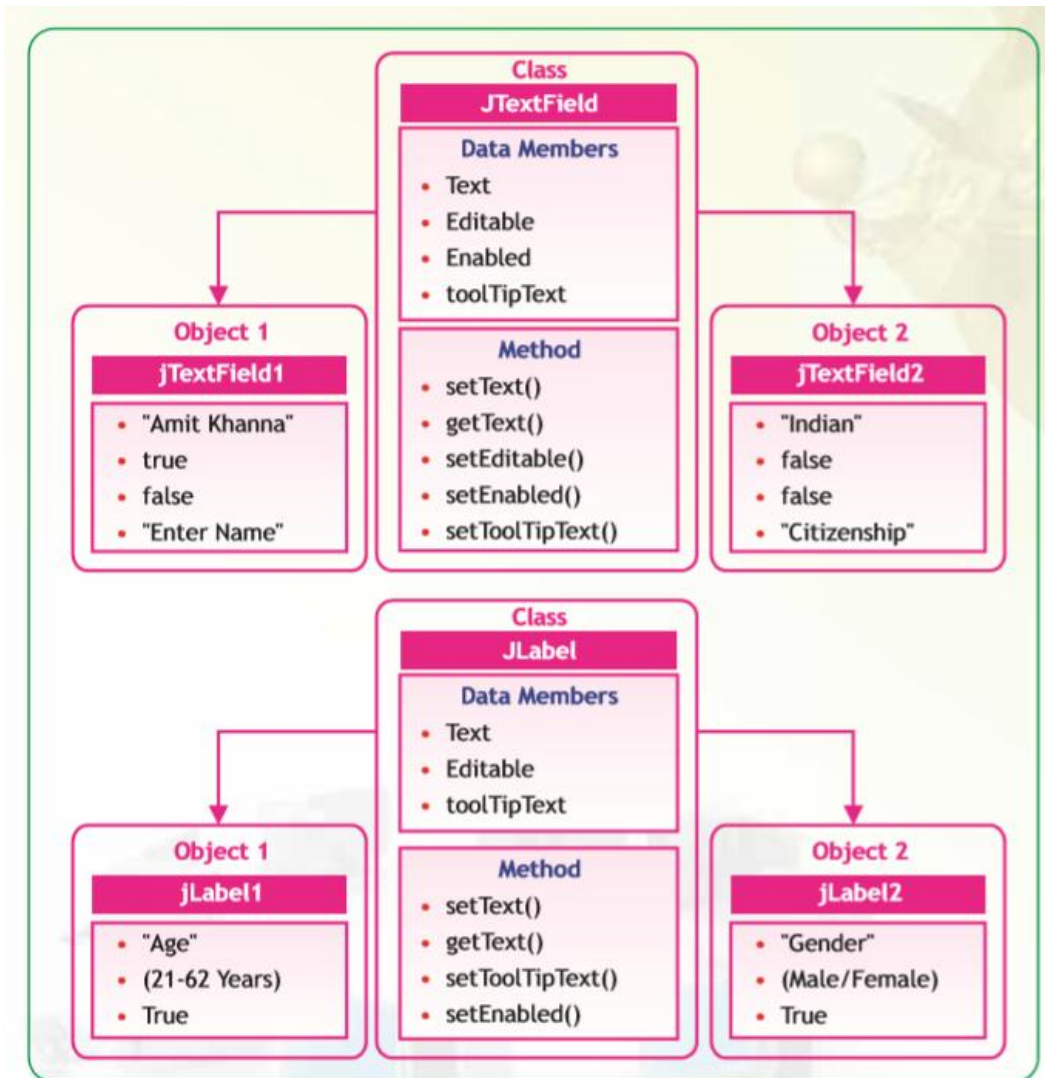


Figure 5.16 JTextField and JLabel Classes

Notice that the properties like Text, Enabled, Editable are actually the data members in the class because they store specific values as data. For example, the property Text of jTextField1object contains the actual text to be displayed in the text field.

Variables

Variables are containers used to store the values for some input, intermediate result or the final result of an operation. The characteristics of a variable are:

- It has a name.
- It is capable of storing values.
- It provides temporary storage.
- It is capable of changing its value during program execution.

However, as different materials require different containers, and so we used different data types to hold different values. Java programming language requires that all variables must first be declared before they can be used.

When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number or to store a single letter or a large number, and they are not going to be interpreted the same way so variables were used along with data types. The data types supported by java are summarized as follows:

Data Types

Data type states the way the values of that type are stored, the operations that can be done on that type, and the range for that type.

Numeric Data Types:

These data types are used to store integer values only i.e. whole numbers only. The storage size and range is listed below:

Name	Size	Range	Example
byte	1 byte(8 bits)	-128 to 127(-2^7 to $+(2^7-1)$)	byte rollno;
short	2 bytes(16 bits)	-32768 to 32767(-2^{15} to $+(2^{15}-1)$)	short rate;
int	4 bytes(32 bits)	-2^{31} to $+(2^{31}-1)$	int num1;
long	8 bytes (64 bits)	-2^{63} to $+(2^{63}-1)$	long amount;

Figure 5.17 Storage size and range of numeric data types

Floating Data Types:

These data types are used to store numbers having decimal points i.e. they can store numbers

having fractional values.

Name	Description	Size	Range	Example
float	Single precision floating point	4 bytes (32 bits)	(3.4×10^{-38}) to $+(3.4 \times 10^{38})$	float average;
double	Double precision floating point	8 bytes (64 bits)	(1.8×10^{-308}) to $+(1.8 \times 10^{308})$	double principal;

Figure 5.18 Storage size and range of floating data types

The decision about which numeric data type to use should be based on the range of values that a variable can take.

Character Data Types:

These data types are used to store characters. Character data types can store any type of values - numbers, characters and special characters. When we want to store a single character, we use char data type and when we want to store a group of characters, we use string data type. For example, to store grades (A, B, C, D, E) of a student we will use char type but to store name of a student, we will use string type. The char data type value is always enclosed inside " (single quotes), whereas a string data type value is enclosed in "" (double quotes).

Operators

With the introduction of variables and constants there arose a need to perform certain operations on them. We performed operations on variables and constants using operators. Operators are symbols that manipulate, combine or compare variables. The operators available in java are summarized below:

Assignment Operator:

One of the most common operators is the assignment operator "=" which is used to assign a value to a variable. We assign the value given on the right hand side to the variable specified on the left hand side. The value on the right hand side can be a number or an arithmetic expression. For example:

```
int sum = 0;
```

```
int prime = 4*5;
```

Arithmetic Operators:

These operators perform addition, subtraction, multiplication, and division. These symbols are similar to mathematical symbols. The only symbol that is different is "%", which divides one

operand by another and returns the remainder as its result.

- +** **additive operator**
- **subtraction operator**
- *** **multiplication operator**
- /** **division operator**
- %** **remainder operator**

Relational Operator:

A relational operator is used to test for some kind of relation between two entities. A mathematical expression created using a relational operator forms a relational expression or a condition. The following table lists the various relational operators and their usage:

Operator	Meaning	Usage
==	equal to	Tests whether two values are equal.
!=	not equal to	Tests whether two values are unequal.
>	greater than	Tests if the value of the left expression is greater than that of the right.
<	less than	Tests if the value of the left expression is less than that of the right.
>=	greater than or equal to	Tests if the value of the left expression is greater than or equal to that of the right.
<=	less than or equal to	Tests if the value of the left expression is less than or equal to that of the right.

Figure 5.19 Relational Operators

Logical Operator:

A logical operator denotes a logical operation. Logical operators and relational operators are used together to form a complex condition. Logical operators are:

Operator	Use	Meaning
&&	a>10 && b<8	a and b are both true
 	a>10 b<8	Either a or b is true
!	! a	a is false

Figure 5.20 Logical Operators

Bitwise Operator:

Bitwise operators are used to perform manipulation of individual bits of a number. They can be used with any of the integral types (char, short, int, etc). They are used when performing update and query operations of Binary indexed tree.

Creating a new Project

Creating a new Form

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 NetBeans Projects 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.

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 5.21.
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.

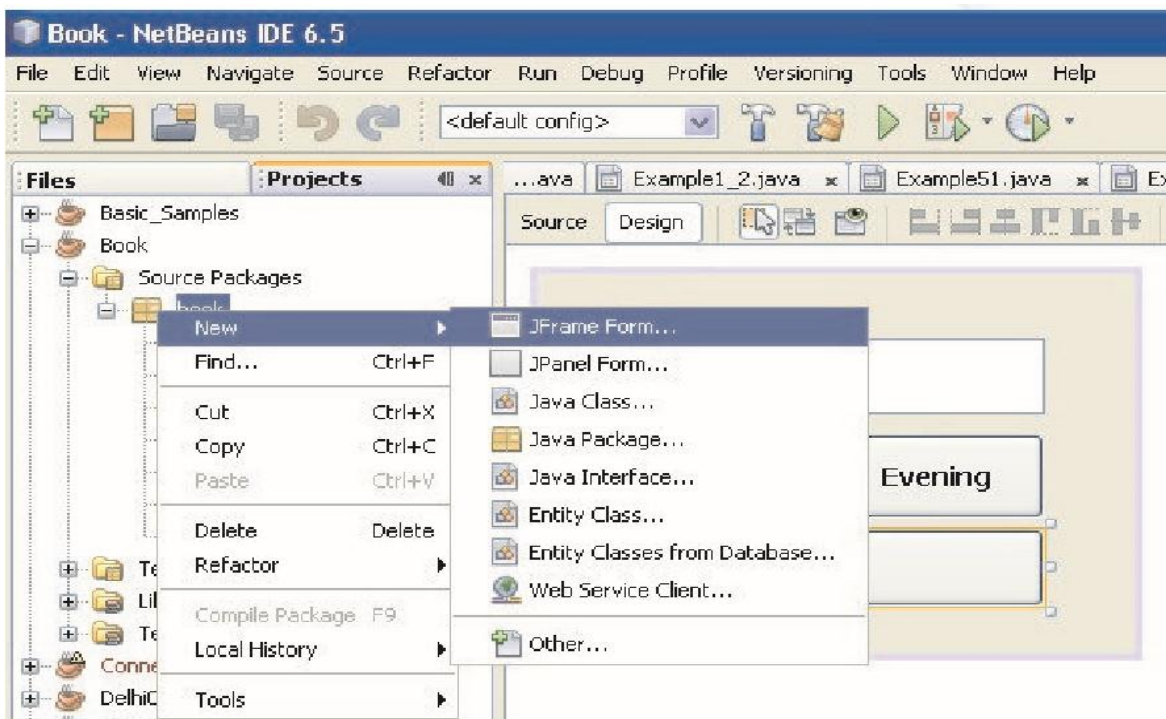


Figure 5.21 Adding a new JFrame Form

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 5.22).
2. Move the cursor over the Form. When the guidelines appear (as displayed in Figure 5.22) 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 5.22. Note that as soon as the button is added on the form, a corresponding node representing the component is added to the Inspector window.

Let us now try and recollect the conversion methods that we have used in java. When a Java program receives input data from a user, it must often convert it from one form (e.g., String) into another (e.g., double or int) for processing.

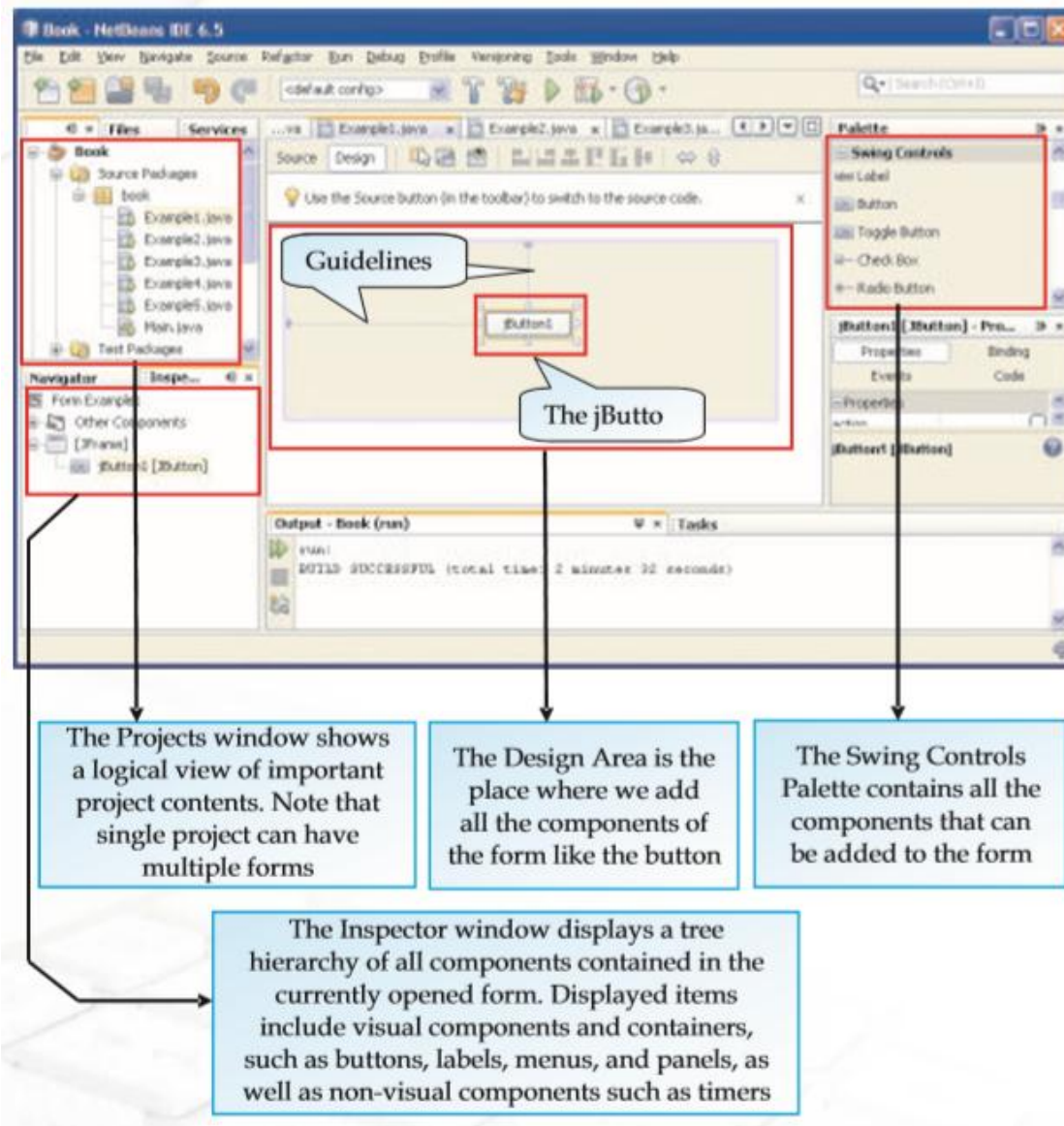


Figure 5.22 Adding a Button and Understanding the Different Windows

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 5.23.

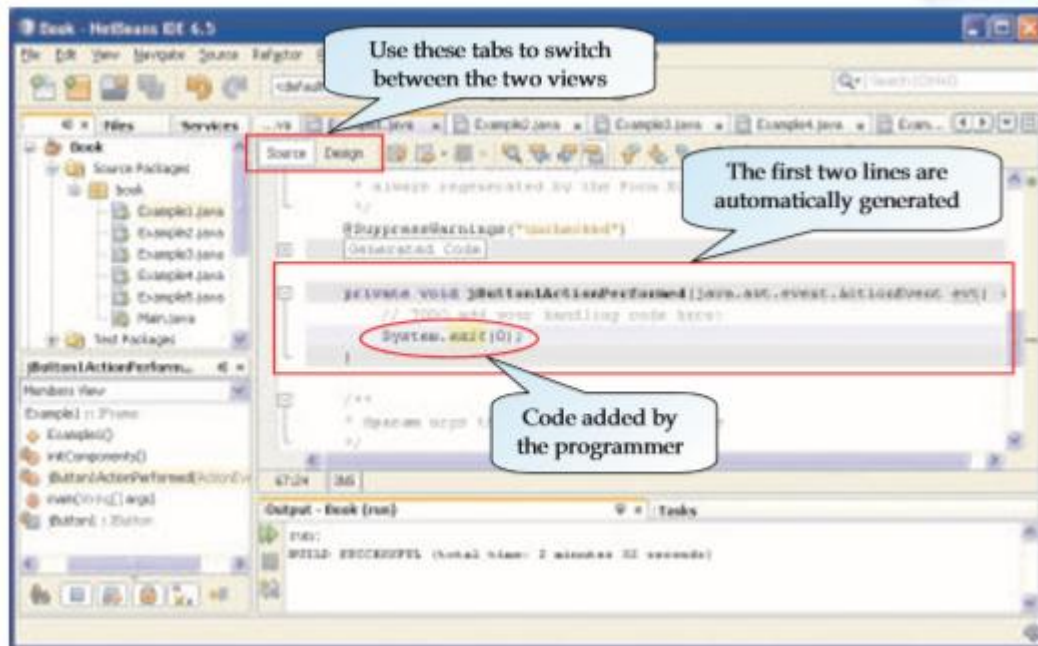


Figure 5.23 Code to exit from an application

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 5.24.

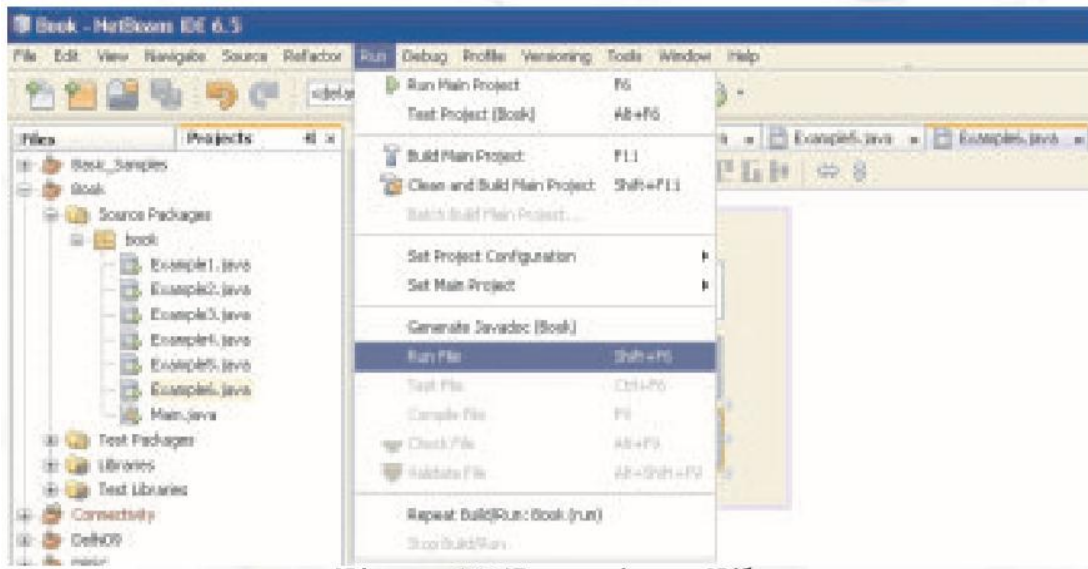


Figure 5.24 Executing a File

On executing the first example, the window shown in Figure 5.25 will appear. Click on the button and observe the result.



Figure 5.25 Simple Button Application

As soon as we click on the button, the application ends and we return back to the Netbeans design window. The one line of code `system.exit(0)` causes the application to terminate successfully.

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 5.22.

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.

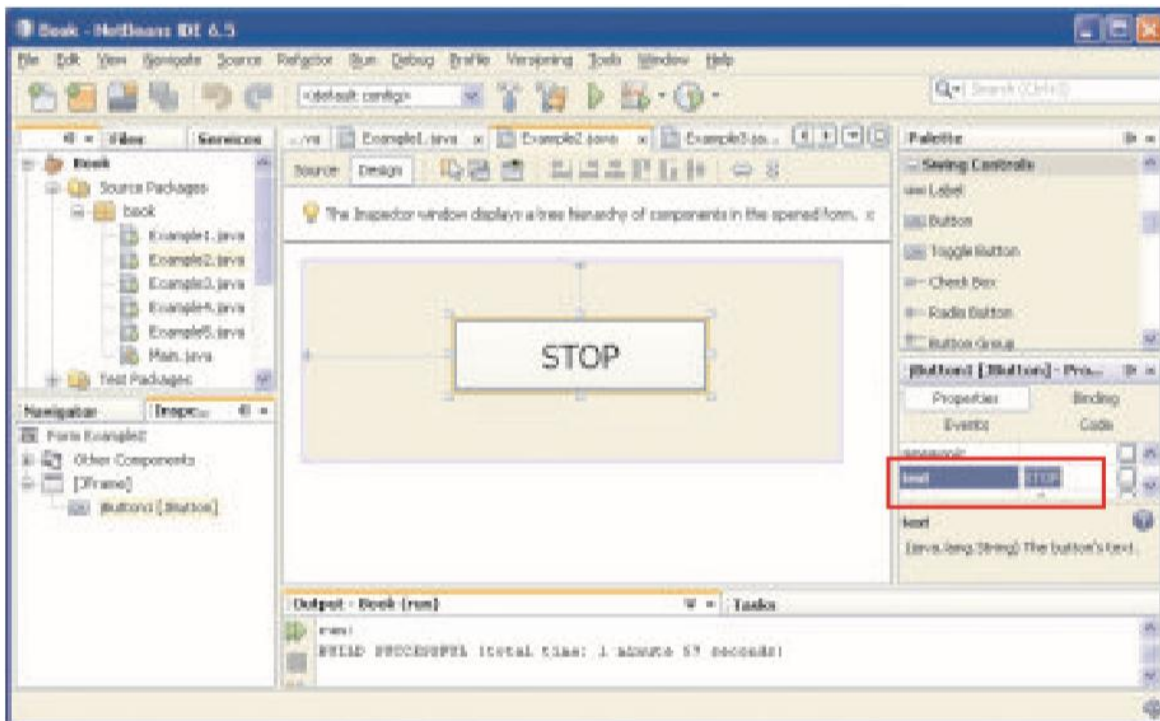


Figure 5.23: Using the text property of a button to change the display text

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 5.23.
- 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 5.24.

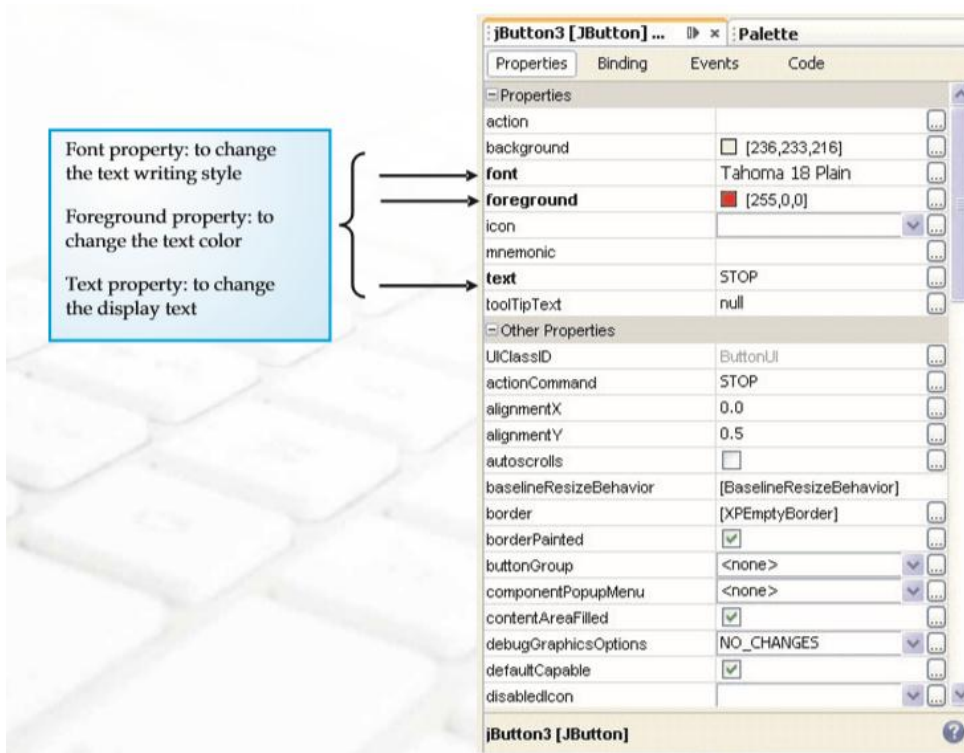


Figure 5.24 Changing Properties of a Button Using the Properties Window

Now when we execute the file the button with the changed text appears as shown in Figure 5.25.



Figure 5.25 The Button with an Appropriate Display Text

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 5.25. Modify the properties of the button as desired and change the text property to "Wish Me".

Switch to the source window and add the single line code as shown in Figure 5.26.

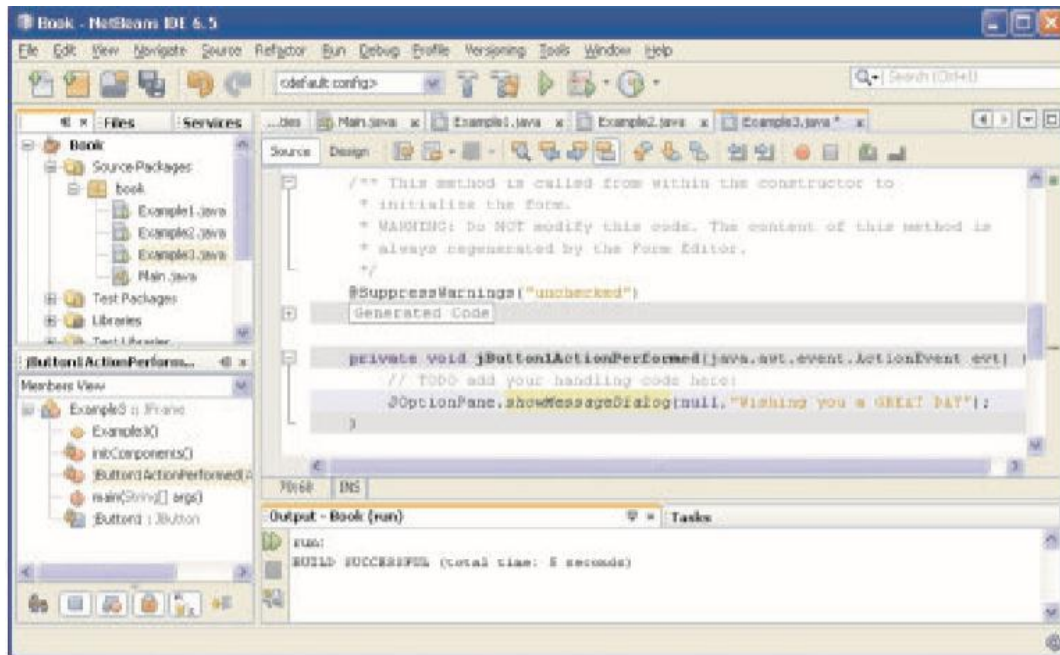


Figure 5.26 Code to Display a Message on the Click of a Button

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.

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 a application with 3 separate buttons and display a different message on the click of all the three buttons. 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 5.27. Don't forget to change the properties and use the resize handle to make the form appear exactly as shown in the Figure 5.27.

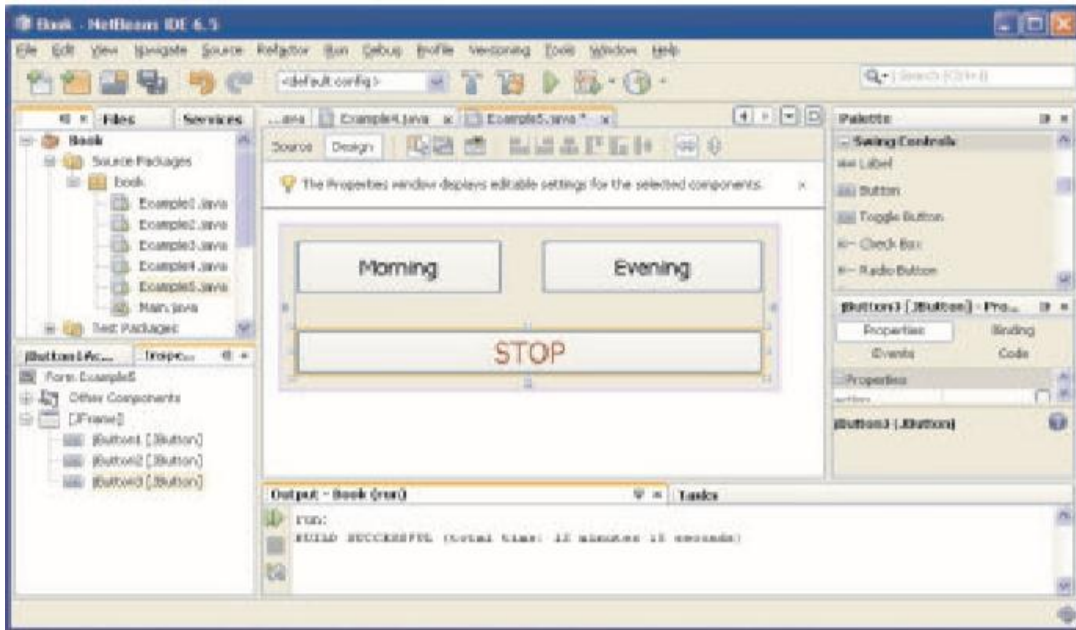


Figure 5.27 A Form with three buttons

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 5.28.

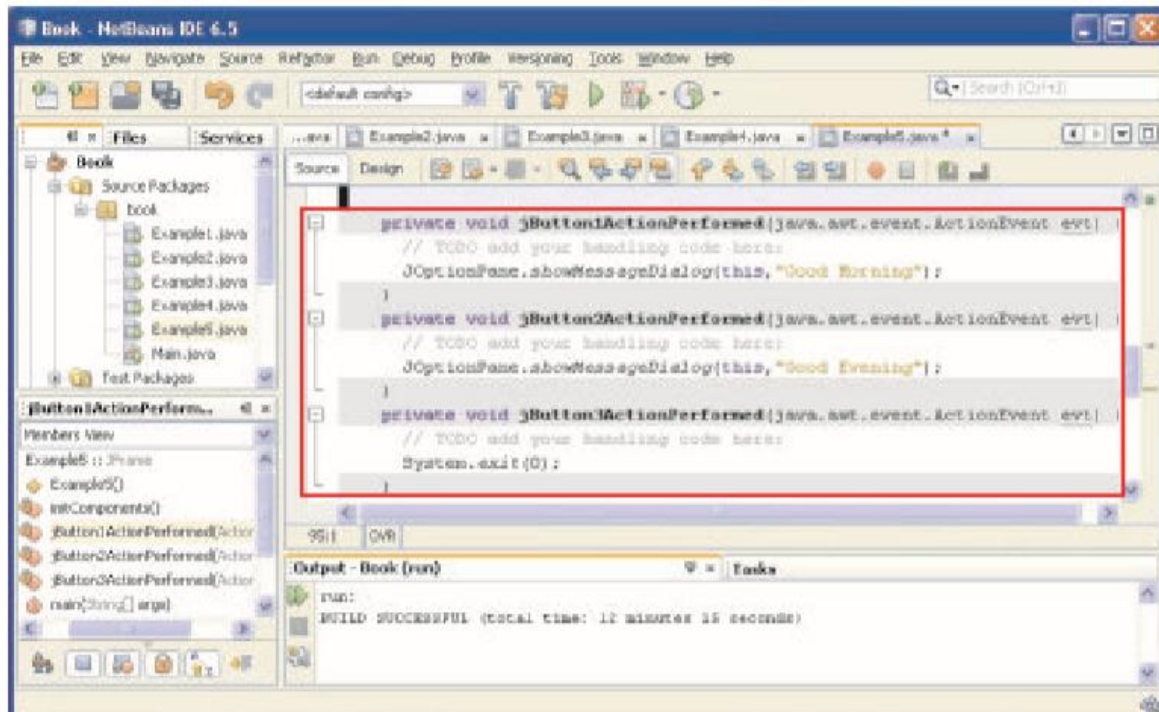


Figure 5.28 Code to Add Functionality to the Form designed in Figure 5.27

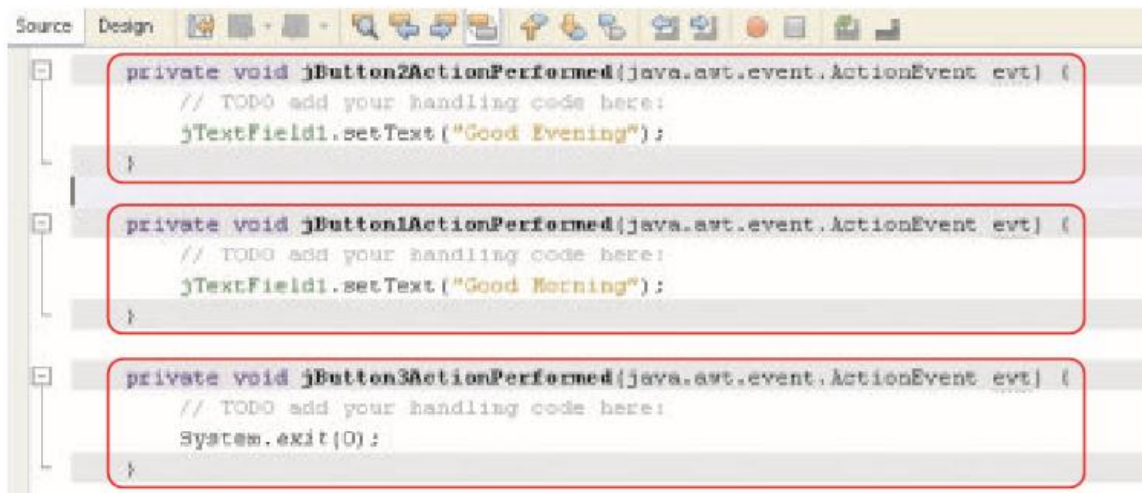
Now execute the Example and observe the result of clicking Morning and Evening Buttons.

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 a message

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.

Let us change the above example so that 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. The code for the same is depicted in the figure 5.29.



```
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 5.29: 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.

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 5.30. 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.

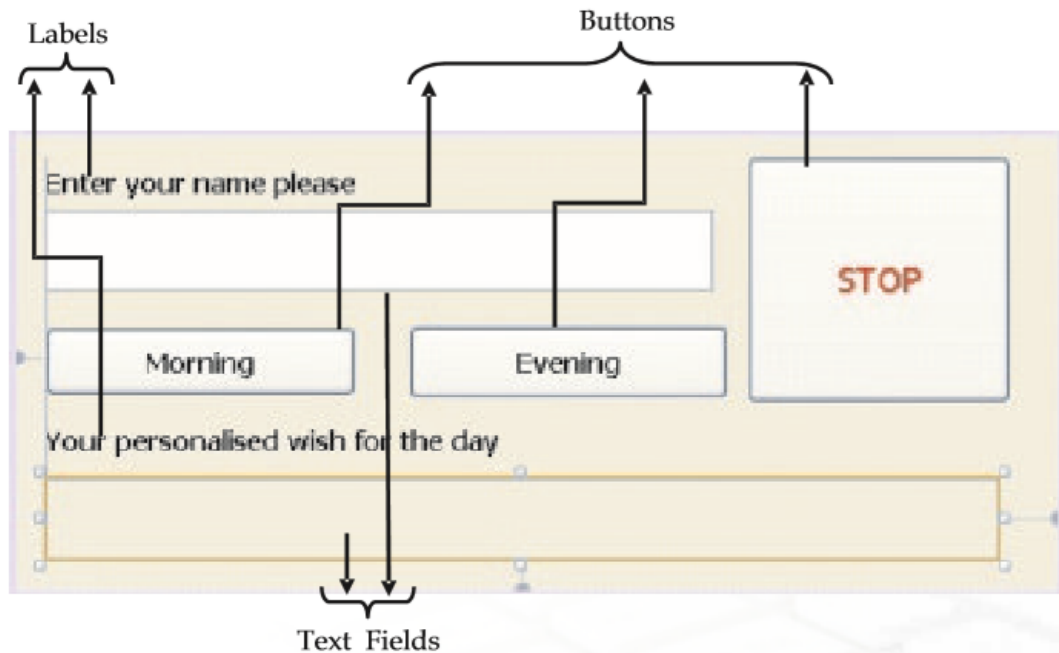


Figure 5.30 Form Design to Display a Personalized Time Based Greeting on the Click of a Button

Observe the Figure 5.30 carefully. we have used a new component - a label and the two text fields. A label is a component which is used to display simple text or as a label for another component. Out of 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 is editable. The text field at the bottom has to display the greeting and is non-editable.

Figure 5.31 displays the properties of both the text fields.

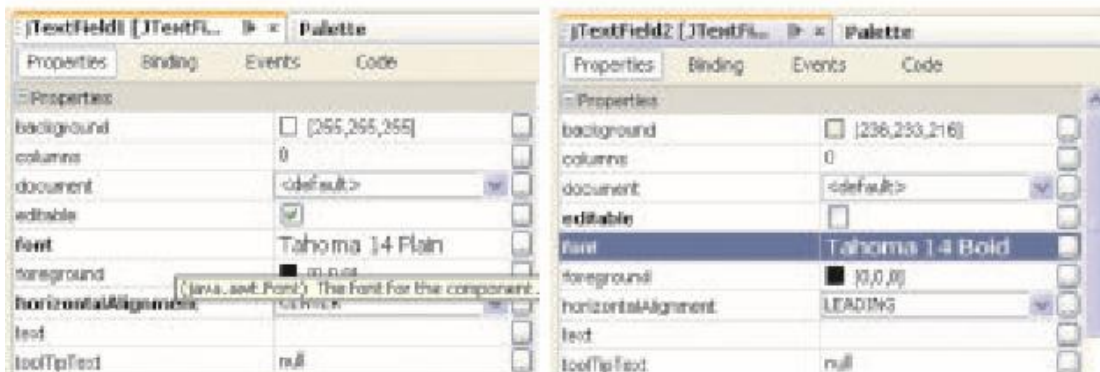


Figure 5.31 Setting Text Field Properties

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 5.32.

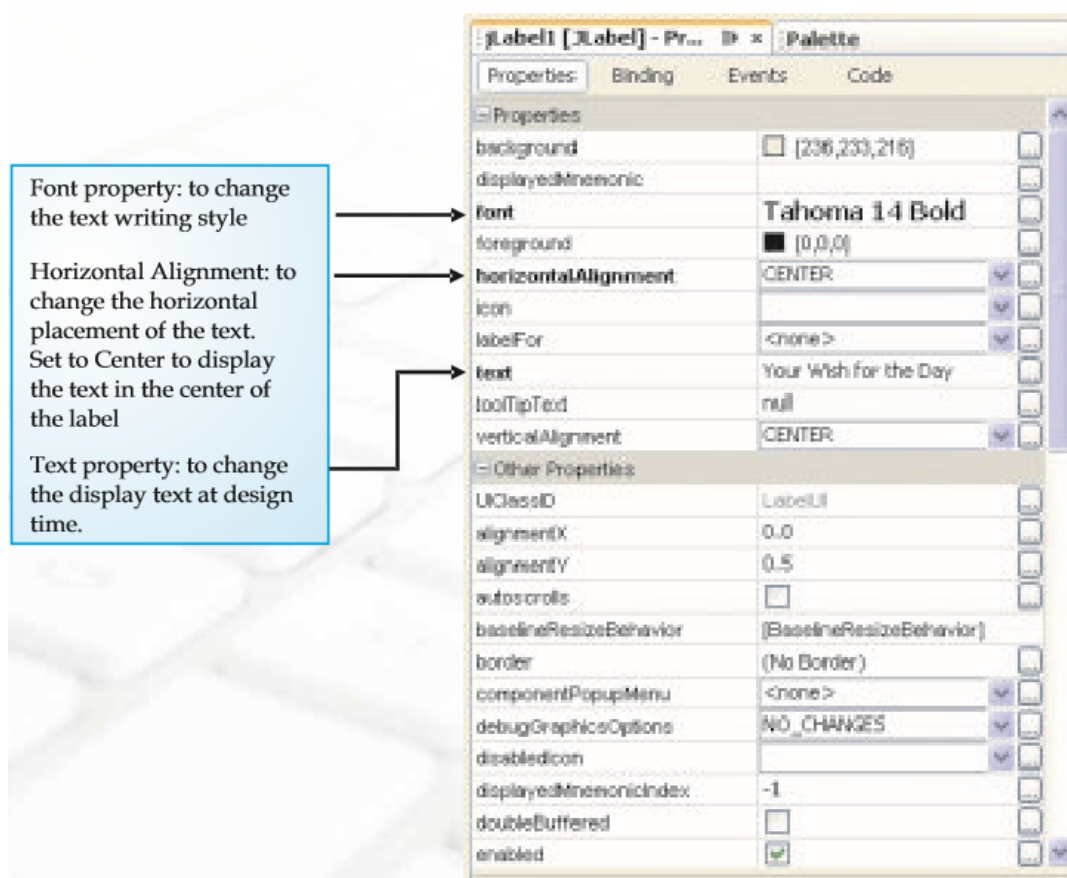


Figure 5.32 Few Properties of the Label Component

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 5.33.

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()`

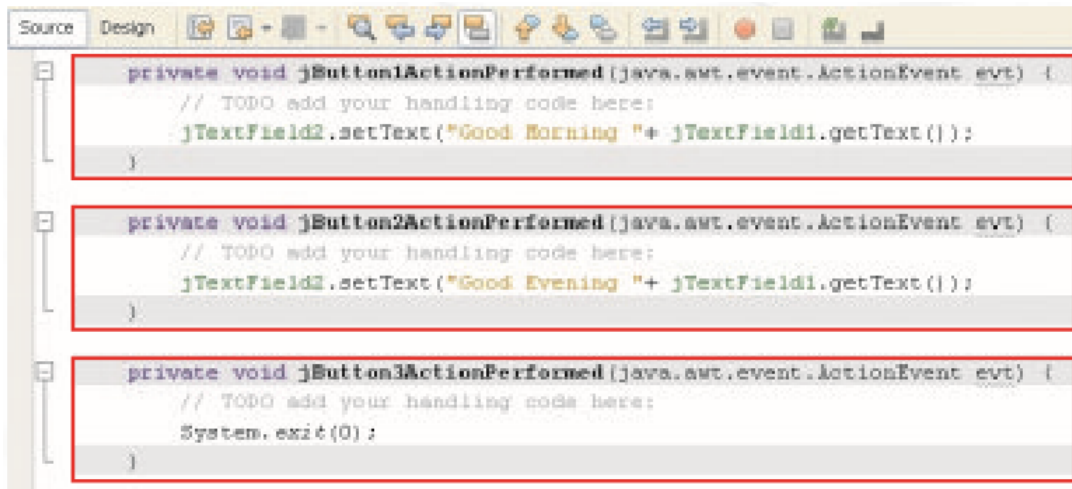
- Retrieves the name entered by the user in the first text field using `getText()`.

"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())
```

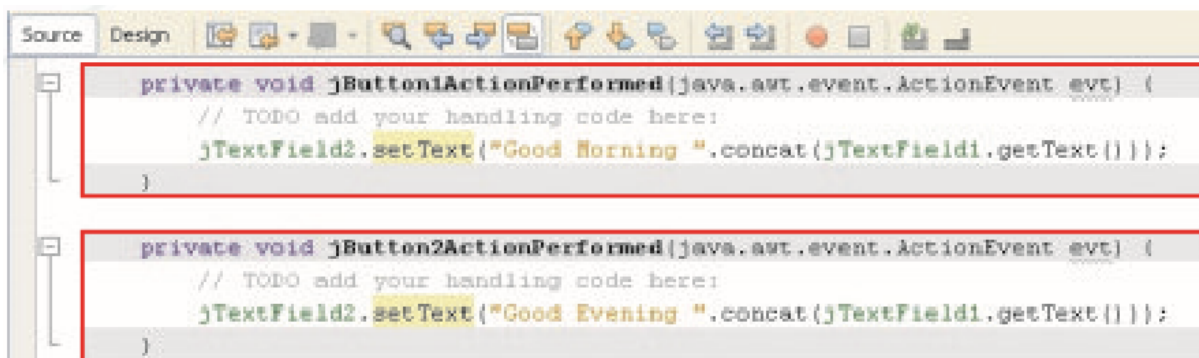
- The display text of the second text field is set to the concatenated message using setText().



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField2.setText("Good Morning" + jTextField1.getText());  
}  
  
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField2.setText("Good Evening" + jTextField1.getText());  
}  
  
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    System.exit(0);  
}
```

Figure 5.33 Code to Display Personalized Time Based Greeting on Click of a Button using the string concatenator operator (+)

Figure 5.34 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 5.34 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 5.35 displays the output when the user enters the name and clicks on the Morning button.

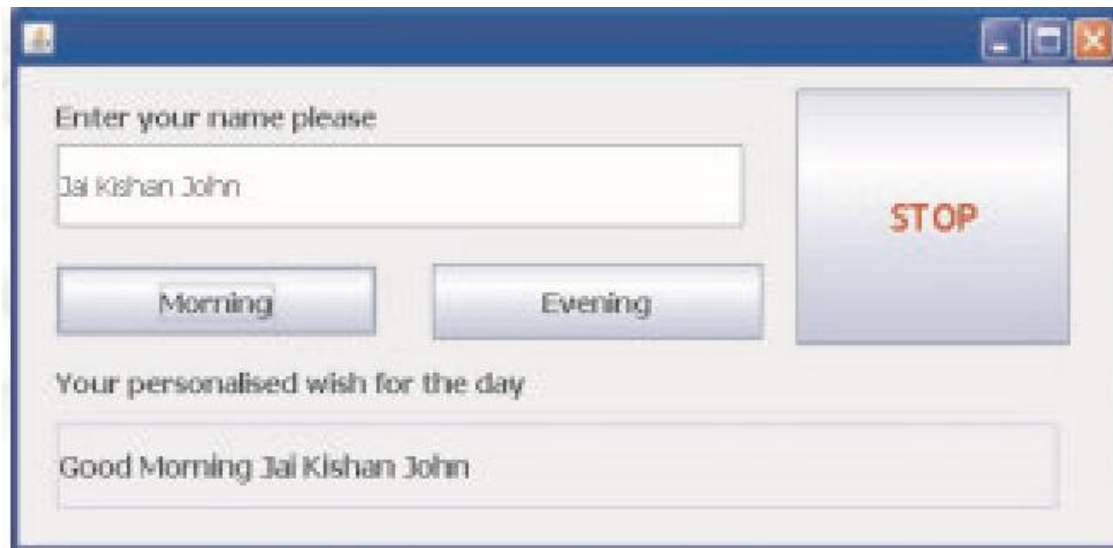


Figure 5.35 Execution of Time Based Personalized Greeting Code

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 5.36. 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.

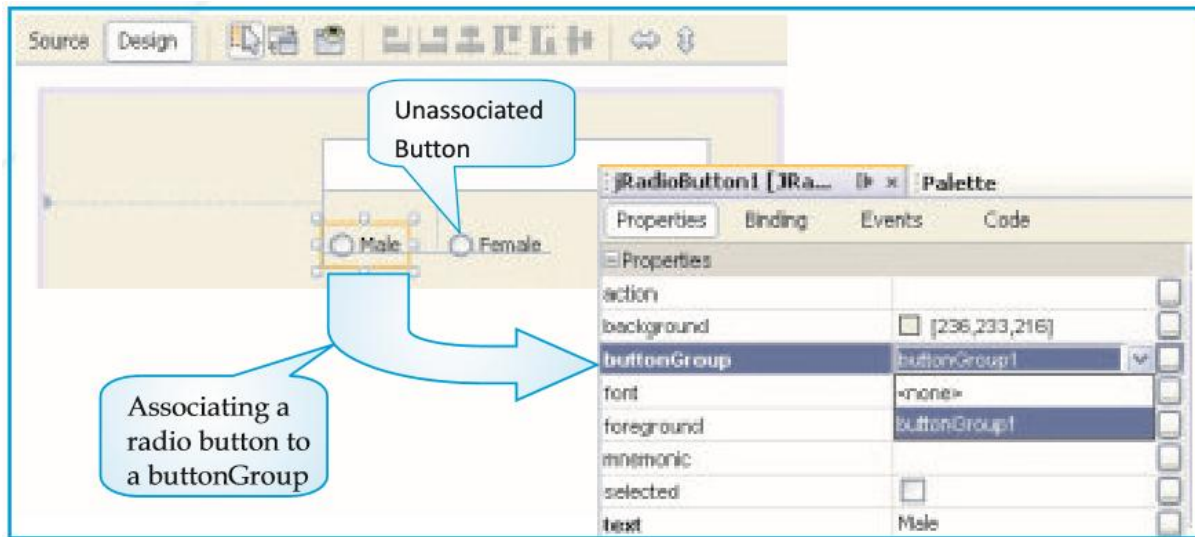


Figure 5.36 Associating First Radio Button with a `buttonGroup`

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. 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 5.37.

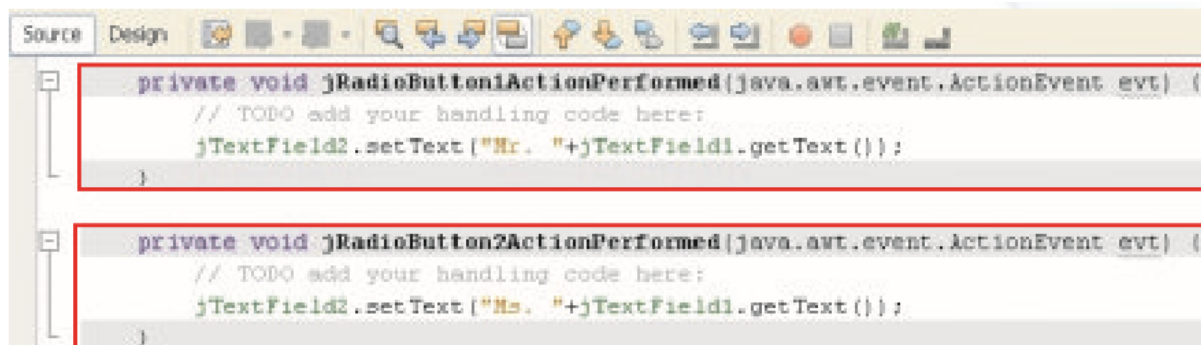


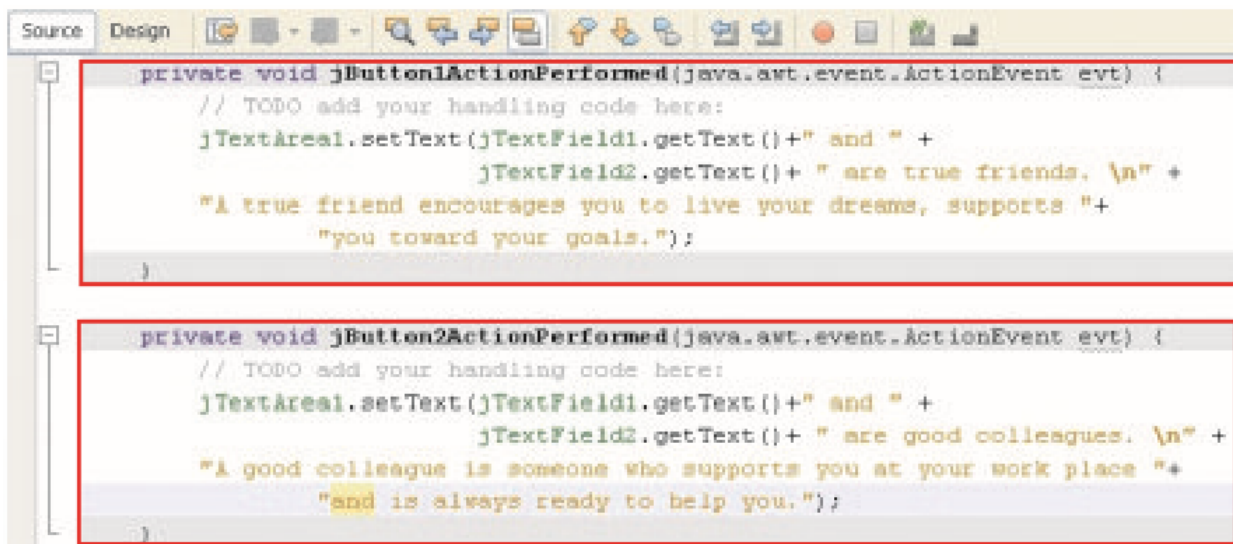
Figure 5.37 Associating Code with the Radio Buttons

Now execute the program and see the output.

Using the Text Area Component

The text field allows the user to enter a single line of text only. The Text Area component is used if we want to accept multiline input or want to display multiline output. 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 5.39. One new component - the Text Area has been added while the rest of the components are familiar. Write the code as shown in Figure 5.38 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 5.38 Code for displaying Multiline Text in a Text Area on the click of a Button

Figure 5.39 shows the sample output of the code given in Figure 5.38.

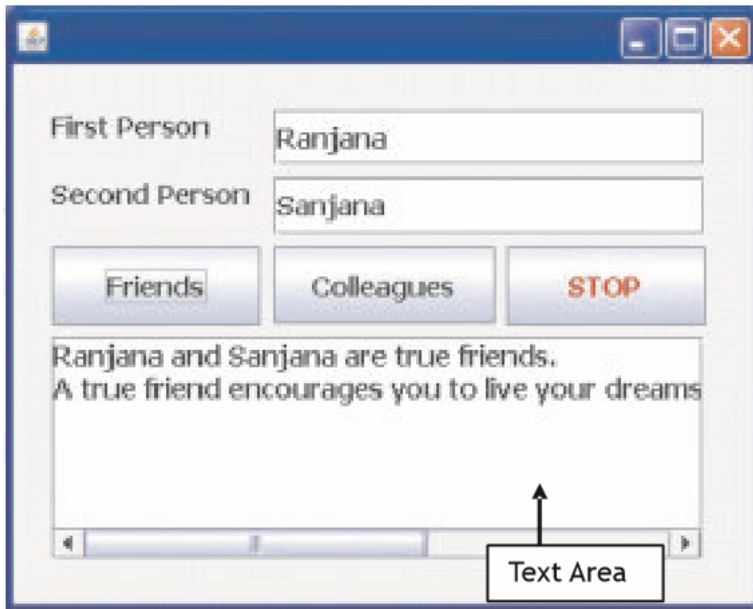


Figure 5.39 Sample Run of the Text Area Application

Handling a Password Field Component

We can use the Password Field 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). 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 5.40 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.

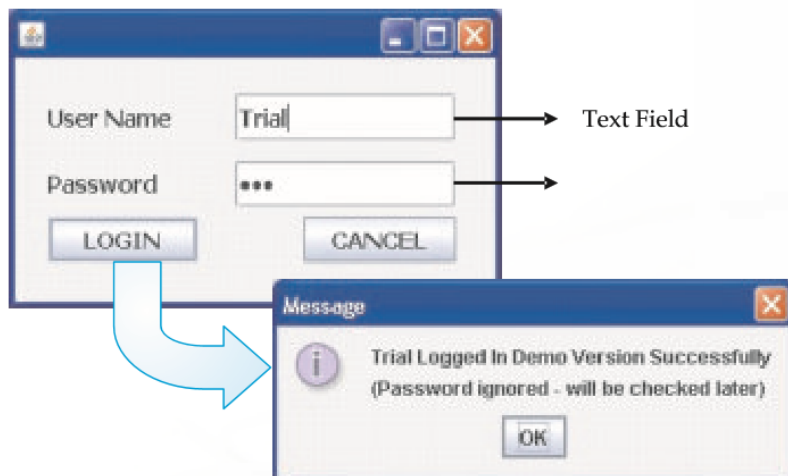


Figure 5.40 Sample run of the Password Application

Figure 5.41 displays the code to display the message on the click of the LOGIN button. Add the code for the CANCEL button also yourself.

```
Source Design
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    JOptionPane.showMessageDialog(null, jTextField1.getText() +
        " Logged In Demo Version Successfully \n" +
        "{Password ignored - will be checked later}");
}
```

Figure 5.41 code to display the message on the click of the LOGIN button

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 5.42. 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.

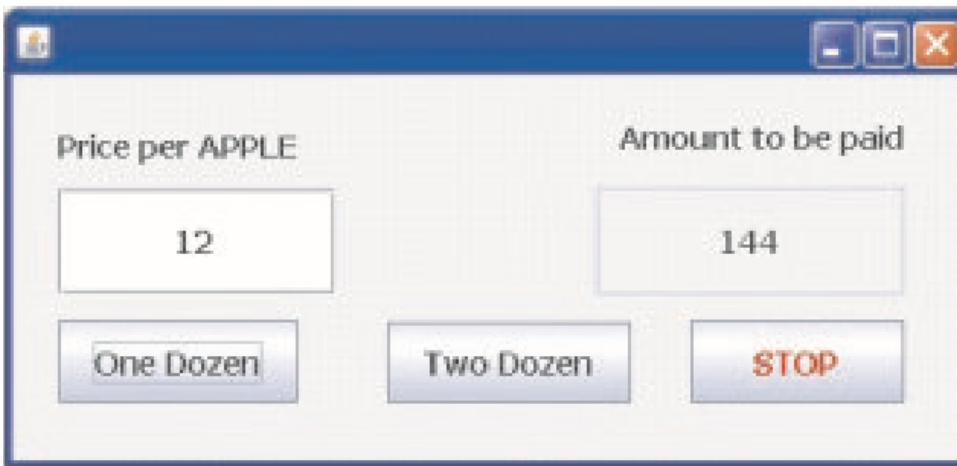
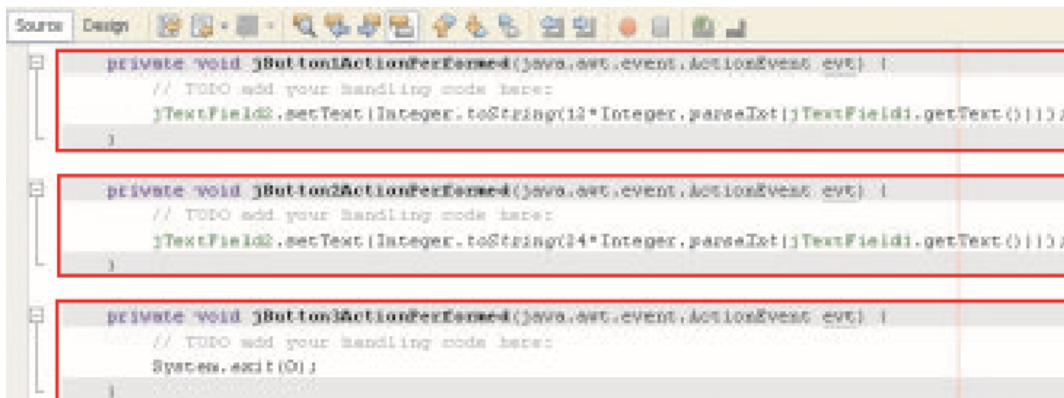


Figure 5.42 Price Calculator

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. After retrieving the value we will simply multiply it by 12 and display the answer in the second text field.
- 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.

Enter the code for each button separately as shown in Figure 5.43.



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField2.setText(Integer.toString(12 * Integer.parseInt(jTextField1.getText())));  
}  
  
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField2.setText(Integer.toString(14 * Integer.parseInt(jTextField1.getText())));  
}  
  
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    System.exit(0);  
}
```

Figure 5.43 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 5.42.

Let us now do some simple calculations involving numbers with decimals (called double in java). Design the form as shown in Figure 5.44.



Figure 5.44 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 values 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 5.45

```
Source Design [Icons]
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    jTextField3.setText(Double.toString
        (
            Double.parseDouble(jTextField1.getText())
            *
            Double.parseDouble(jTextField2.getText())
        )
    );
}
```

Figure 5.45 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.

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 5.46.

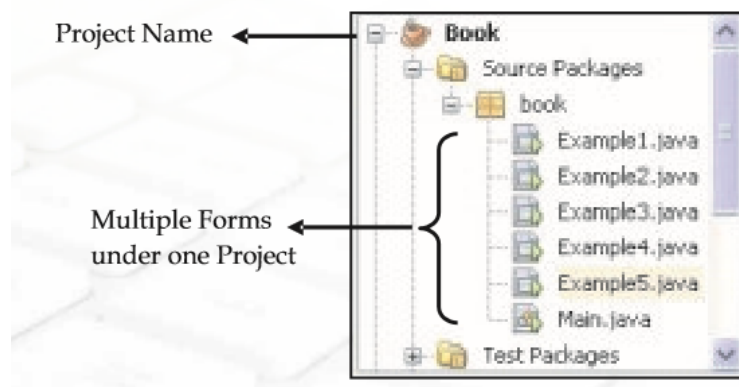


Figure 5.46 Project Window Showing Multiple Forms

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.

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 5.47.

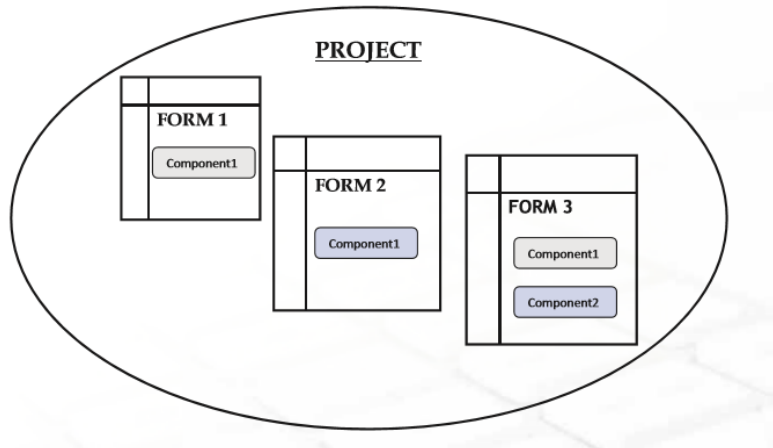


Figure 5.47 Relations Between Project, Form and Components

Variable Declaration

We have learnt that variables are capable of storing values, which we need to use. To reference a variable, it should have a name. Moreover, variables in java can only accept a value that matches its data type. So before we use a variable we must decide on its name and its data type. Giving this information to the language compiler is called variable declaration. Thus, the declaration of a variable tells us about the name of the variable which is necessary to reference it, the type of data it will store and optionally an initial value. Given below are some commonly used ways of variable declaration.

Declaration Example Comment

`int Apples;` Simple declaration of an integer variable named Apples.

`float Sum = 4;` Declaration of a float variable named Sum which has an initial value of 4.0.

Variable Naming Conventions

As mentioned above, each variable needs to have a name so that it can be referenced anywhere during the application. Each programming language has its own set of rules for naming variables. The rules and conventions for naming variables in Java are summarized below:

- Variable names are case sensitive.
- Keywords or words, which have special meaning in java, should not be used as the variable names.
- Variable names should be short and meaningful.

- All variable names must begin with a letter, an underscore (_) or a dollar sign (\$). The convention is to always use a letter and avoid starting variable names with underscore (_) and dollar sign (\$).
- After the first initial letter, variable names may contain letters and digits (0 to 9) and (_, \$), but no spaces or special characters are allowed.

Using the above conventions and rules following is an indicative list of acceptable and unacceptable variable names.

Acceptable Variable Names - Grade, Test_Grade, TestGrade

Unacceptable Variable Names - Grade(Test), 2ndTestGrade, Test Grade, Grade_Test#2 Try to

Java variable names are case sensitive, so sum1 and SUM1 aren't the same variable.

Simple Applications Using the Concept of Variables

Now, let us develop a simple application to learn the use and handling of char data type. Suppose we want to display the message entered by the user surrounded by four different characters. See the sample execution of the application as shown in Figure 5.48.

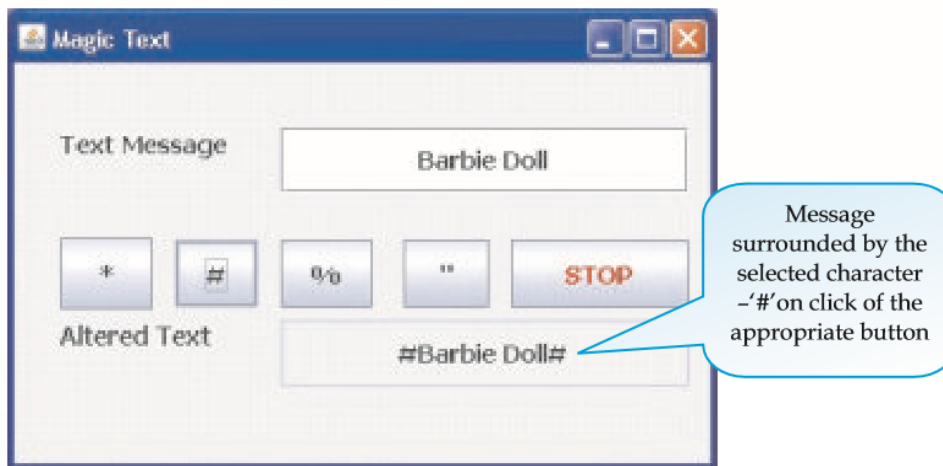


Figure 5.48 Handling Character Variables

As is clear from the sample run, we need to concatenate the message and the selected character depending upon the button clicked by the user. Let us now design the application:

First add a new JFrame form and set its title property to "Magic Text". Design the form as shown in Figure 5.48 with the following components:

- one editable text field to accept the message
- five buttons - four to concatenate message with different characters and one to exit from the application
- one non-editable text field to display the concatenated message
- appropriate labels to direct the user

Change the properties of the components as learnt in the previous chapter so that the form looks exactly like the one displayed in Figure 5.48. The next step is to associate code with the all the buttons. Double click on the buttons one by one in the design window to reach at the point in the source window where the code needs to be written. Add the code for each of the buttons shown as follows.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Concatenate * to the text in
    jTextField1: char Star;
    Star='*';
    jTextField2.setText(Star+jTextField1.getText()+Star
);
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Concatenate # to the text in
    jTextField1: char Hash;
    Hash='#';
    jTextField2.setText(Hash+jTextField1.getText()+Hash
);
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // Concatenate % to the text in
    jTextField1: char Percent;
    Percent='%';
    jTextField2.setText(Percent+jTextField1.getText()+Percent);
}
```

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt)
{
    //To STOP the
    application:
    System.exit(0);
}
```

```

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // Concatenate " to the text in
    jTextField1: char Quotes;
    Quotes="";
    jTextField2.setText(Quotes+jTextField1.getText()+Quotes);
}

```

Now try to develop a similar application with four buttons to perform the basic mathematical operations of addition, subtraction, multiplication and division of any two numbers entered by the user. First design the form with the following components:

- two editable text fields to accept the two numbers .
- four buttons to decide the operation, one button to reset the fields and one button to exit out of the application.
- one non-editable text field to display the result.
- appropriate labels to direct the user.

When the user enters two numbers and clicks on the + button, the sum of the numbers is displayed in the jTextField3 which has been disabled (by setting its editable property to false) as shown in Figure 5.49.

When the user clicks on the RESET button the contents of all the Text Fields are cleared.



Figure 5.49 A Simple Calculator Showing Addition of Two Numbers

Now write the code for each button of the basic calculator shown as follows:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Code to add Number1
    and Number2: double
    Number1,Number2,Result;
    Number1=Double.parseDouble(jTextField1
    .getText());
    Number2=Double.parseDouble(jTextField2.getText()
    ); Result=Number1+Number2;
    jTextField3.setText(Double.toString(Result));
}
```

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent
evt)
{
    // Code to clear the contents of the
    text field: jTextField1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
}
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Code to subtract Number2 from
    Number1: double Number1,Number2,Result;
    Number1=Double.parseDouble(jTextField1.g
    etText());
    Number2=Double.parseDouble(jTextField2.getText());
    Result=Number1-Number2;
    jTextField3.setText(Double.toString(Result));
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent
evt)
{
    // Code to multiply Number1
    and Number2: double
    Number1,Number2,Result;
    Number1=Double.parseDouble(jTextField1
    .getText());
    Number2=Double.parseDouble(jTextField2.getText()
    ); Result=Number1*Number2;
    jTextField3.setText(Double.toString(Result));
}
```

Let us now understand the code. We want to display the result of a computation involving numbers entered in the first and second text field in the third text field based on the button clicked. So only the operator is being changed while the basic

```
private void jButton6ActionPerformed(java.awt.event.ActionEvent
evt)
{
    System.exit(0);
}
```

steps of computation remain the same. So we will explain one (coding for the first button) in detail here:

```
double Number1,Number2,Result;
```

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent
evt)
{
    // Code to divide Number1
    by Number2: double
    Number1,Number2,Result;
    Number1=Double.parseDouble(jTextField1
.getText());
    Number2=Double.parseDouble(jTextField2.getText()
); Result=Number1/Number2;
    jTextField3.setText(Double.toString(Result));
}
```

- declares three variables of type double

```
Number1=Double.parseDouble(jTextField1.getText()); and
```

```
Number2=Double.parseDouble(jTextField2.getText());
```

- retrieves the value entered by the user in the first and second text field using `getText()`. These values by default are treated as strings i.e. a group of characters and not as a number so the string values need to be converted to a double type and this is achieved using the `parseDouble()` method. After converting it to a double type the values are assigned to the variables declared in the first line of code

```
Result=Number1+Number2;
```

- The two values stored in the variables are added and the calculated value is stored in the variable `Result`.

```
jTextField3.setText(Double.toString(Result));
```

- The value stored in the variable `Result` is of type double so it is first converted to type string using the `toString()` method and then the display text of the third text

field is set to the converted value using `setText()`.

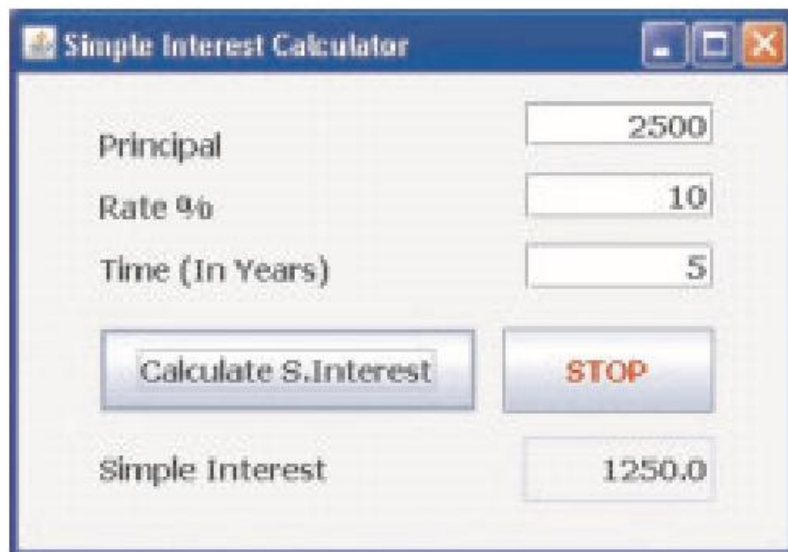
The working of the other three buttons (second, third and fourth) is similar to the one explained above. We are already familiar with the working of the STOP button so let us give a quick look to the coding of the RESET button

```
jTextField1.setText("");                                and  
jTextField2.setText(""); and  
jTextField3.setText("");
```

- The display text of all the three buttons is set to an empty string (i.e. blank) using the `setText()` method.

In all the applications developed so far we have used a single type of data and done simple calculations. Next let us explore the use of multiple data types and using these data types try to perform complex calculations.

Observe the form shown in Figure 5.50 and design a similar form.



Principal	2500
Rate %	10
Time (In Years)	5
Calculate S.Interest	STOP
Simple Interest	1250.0

Figure 5.50 Simple Interest Calculator

The aim of the application is to accept the principal amount, rate and time in three separate text fields and calculate the simple interest on the click of a button. The calculated interest is displayed in a disabled text field. The coding for the same is given in Figure 5.51 .


```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    double Principal,Rate,SInterest;
    byte Time; //Expected value not more than 127 Years
    Principal=Double.parseDouble(jTextField1.getText());
    Rate=Double.parseDouble(jTextField2.getText());
    Time=Byte.parseByte(jTextField3.getText());
    SInterest=(Principal*Rate*Time)/100; //Formula to calculate SI
    jTextField4.setText(Double.toString(SInterest));
}

```

```

private void jButton2ActionPerformed(java.awt.event.ActionEvent
evt)
{

```

FFigure 5.51 Code for Simple Interest Calculator

Control Structures

We use control structures when we want to control the flow of the program. There are types of control structures: Selection statements and Iteration statements.

Selection Statements:

A selection statement selects among a set of statements depending on the value of a controlling expression. The selection statements are the if statement and the switch statement, which are discussed below:

Simple if Statement - The if statement allows selection (decision making) depending upon the outcome of a condition. If the condition evaluates to true then the statement immediately following if will be executed and otherwise if the condition evaluates to false then the statements following the else clause will be executed. The selection statements are also called conditional statements or decision statements.

The syntax of if statement is as shown below:

Syntax:

```

if (conditional expression)
{
    Statement Block;
}
else
{
    Statement Block;
}

```

Points to remember about if statement:

- The conditional expression is always enclosed in parenthesis.
- The conditional expression may be a simple expression or a compound expression.
- Each statement block may have a single or multiple statements to be executed. In case there is a single statement to be executed then it is not mandatory to enclose it in curly braces ({}), but if there are multiple statements then they must be enclosed in curly braces ({}).
- The else clause is optional and needs to be included only when some action is to be taken if the test condition evaluates to false.

Let us now design another application: "Vote Eligibility Checker" where we are accepting the age from the user and we want to validate whether the person is eligible to vote or not. We are accepting the age of the user in a text field and testing whether the age entered by the user is greater than 18 or not. If the age is greater than 18 then the message "You are eligible to VOTE" is displayed. If the age is less than then the message "You are NOT eligible to VOTE" is displayed. In such situations when we have to take action on the basis of outcome of a condition, we need to use a Selection statement. Design the form and set the properties of the components so that the form looks exactly like the one displayed in figure 5.52.



Figure 5.52 Sample Run of The Vote Eligibility Checker Application

The code for this application is given as follows:

```
private void  
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // Code to check eligibility to vote with else condition: if  
    (Integer.parseInt(jTextField1.getText()) >= 18)  
        JOptionPane.showMessageDialog(null, "You are eligible To  
    VOTE");  
    else  
        JOptionPane.showMessageDialog(null, "You are NOT eligible  
    To VOTE");  
}
```

Let us now understand the single line code in detail.

```
Integer.parseInt(jTextField1.getText())
```

- retrieves the value entered by the user in the text field using `getText()`. This value by default is treated as a string and not as a number so it needs to be converted to an integer type and this is achieved using the `parseInt()` method.

```
if (Integer.parseInt(jTextField1.getText()) >=18)
```

- check whether the value retrieved from the text field is greater than or equal to 18 or not. The if statement is used to check the condition and if the condition evaluates to true then we specify what action is to be taken

```
if (Integer.parseInt(jTextField1.getText()) >=18)
```

```
    JOptionPane.showMessageDialog(null, "You are eligible to VOTE")
```

- This if statement is used to check whether the value retrieved from the text field is greater than or equal to 18 or not and if it is then it displays the message "You are eligible to VOTE" using the `showMessageDialog()` method.

```
else
```

```
    JOptionPane.showMessageDialog(null, "You are NOT eligible to VOTE");
```

- The else statement is executed if the value retrieved from the text field is less than 18 and if it is then it displays the message "You are NOT eligible to VOTE" using the `showMessageDialog()` method.

Nested if ... else - These control structures are used to test for multiple conditions as against the simple if statement which can be used to test a single condition. The syntax of nested if else is as follows:

Syntax:

```
if (conditional expression1)
{
    statements1;
}
else if (conditional expression2)
{
    statements2;
}
else if (conditional expression3)
{
    statements3;
```

```

}
else
{
    statements4;
}

```

Let us now develop another application called the Week Day Finder in which we will learn how to use if statement when we have multiple test conditions. The Week Day Finder will display the name of the week in a disabled text field depending upon the day selected by the user. The days are displayed as radio button options, which have to be selected. So, the form will have 7 radio buttons and depending on the button selected the day of the week will be displayed.

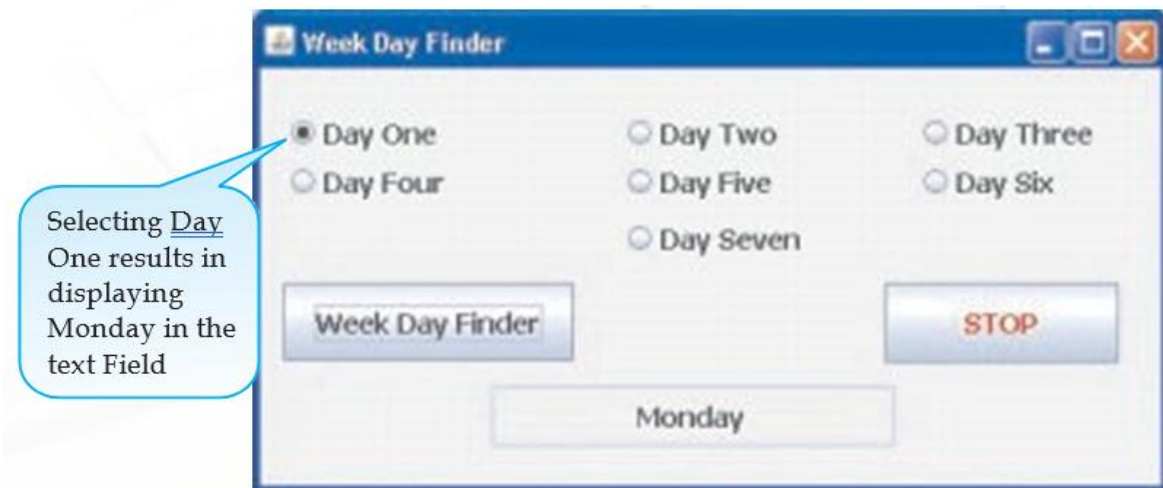


Figure 5.53 Sample Run of the Week Day Finder

Design the form as shown in Figure 5.53. and set the properties of the components according to the functionality required as shown in Figure 5.53. Monday is displayed when the radio button corresponding to Day One is selected as shown in Figure 5.53 as it is the first day of the week. If we select the radio button corresponding to Day Six then Saturday is displayed, as it is the sixth day of the week.

It is clear from the above form that we have to test for multiple conditions. If `jRadioButton1` is selected then Monday will be displayed and if `jRadioButton2` is selected then Tuesday will be displayed and so on. All the select conditions will be checked from top to bottom and wherever the condition evaluates to true, the statements corresponding to that `jRadioButton` will get executed. What happens in case none of the `jRadioButton` is selected?

After understanding the working let us now write the code for the Week Day Finder application as shown in Figure 5.54.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    // To find the day of the
week if
(jRadioButton1.isSelected())
    jTextField1.setText("Monday");
else if (jRadioButton2.isSelected())
    jTextField1.setText("Tuesday");
else if (jRadioButton3.isSelected())
    jTextField1.setText("Wednesday"
); else if
(jRadioButton4.isSelected())
    jTextField1.setText("Thursday")
; else if (jRadioButton5.isSelected())
    jTextField1.setText("Friday");
else if (jRadioButton6.isSelected())
    jTextField1.setText("Saturday")
; else if (jRadioButton7.isSelected())
    jTextField1.setText("Sunda
y"); else
    jTextField1.setText("Day - Not Selected");
}

```

Figure 5.54 Code for the Week Day Finder Application

The above code introduces us to a new method called `isSelected()`. This method is used to check whether a particular radio button is selected or not. The syntax of this method is given below:

Syntax:

`jRadioButton.isSelected()`

This method returns a boolean value i.e. true or false. The true indicates that the radio button is selected and false indicates that the radio button is not selected.

Let us now understand the code in detail. Since the code in each subsequent else is almost the same except the display text, so we will try and understand the first three lines.

`if (jRadioButton1.isSelected())`

- check whether the first radio button is selected or not

`if (jRadioButton1.isSelected()) jTextField1.setText("Monday")`

- Display "Monday" in the text field if the first radio button is selected

`if (jRadioButton1.isSelected()) jTextField1.setText("Monday")`

else if (jRadioButton2.isSelected())

- If the first radio button is not selected then check whether the second radio button is selected or not

Note that to handle multiple conditions, we have used a series of if-else statements. Such a if else statement is called nested if else statement. In this form the if statement checks each of the conditions one by one from top to bottom until it finds one that is true. In case none of the conditions are true then the statement corresponding to the last else is executed. Therefore, in case none of the jRadioButton is selected then "Day - Not Selected" will be displayed.

Switch Statement -

This selection statement allows us to test the value of an expression with a series of character or integer values. On finding a matching value the control jumps to the statement pertaining to that value and the statement is executed, till the break statement is encountered or the end of switch is reached. The expression must either evaluate to an integer value or a character value. It cannot be a string or a real number. The syntax of the switch statement is as follows:

```
switch (Variable/Expression)
{
    case Value1:statements1 ;
        break ;
    case Value2:statements2 ;
        break ;
    .
    .
    default:statements3 ;
}
```

After understanding the working of switch statement, let us now develop a discount calculator using the switch statement. Design the form as shown in Figure 6.30. The Customer is given a discount on the Bill Amount depending upon the Customer Type selected from the combo

box. Discount is calculated as follows:

Customer Type	Discount
Platinum	30%
Gold	20%
Silver	10%
New Customer	No Discount

When the application is executed the discount amount is deducted from the Bill Amount depending upon the Customer Type selected by the user.

When Customer Type is Silver the customer gets a discount of 10% as shown in figure 5.55.

When Customer Type is Gold the customer gets a discount of 20% and when Customer Type is Platinum the customer gets a discount of 30% on the Bill Amount.

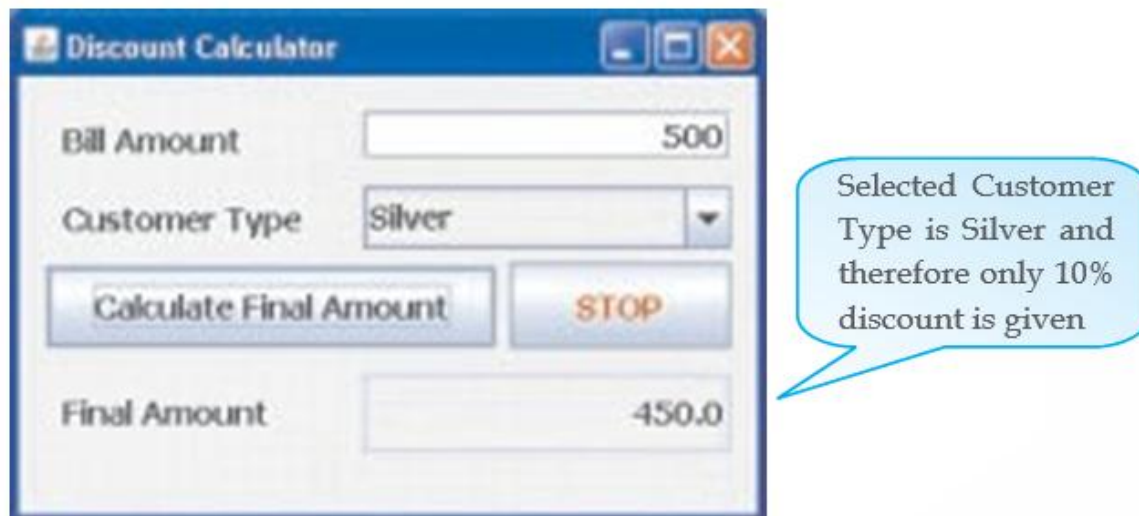


Figure 5.55 Discount of 10% for Customer Type Silver

Let us now write the code for the discount calculator as shown in 5.56.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Code to calculate discount depending upon customer type:
    double FinalAmount=0;
    double BillAmount = Double.parseDouble(jTextField1.getText());
    switch(jComboBox1.getSelectedIndex())
    {
        case 0: FinalAmount=BillAmount; //No Discount for
new customer break;
        case 1: FinalAmount=0.90*BillAmount; //10% Discount
for silver break;
        case 2: FinalAmount=0.80*BillAmount; //20%
Discount for gold break;
        case 3: FinalAmount=0.70*BillAmount; //30%
Discountfor platinum break;
        default:FinalAmount=BillAmount;
    }
    jTextField2.setText(Double.toString(FinalAmount));
}
}

```

Figure 6.31 Code for Discount Calculator Using switch Statement

Now let us understand the code in detail.

```
double FinalAmount=0;
```

- Declare a variable FinalAmount of type double and initialize it to 0.

```
double BillAmount = Double.parseDouble(jTextField1.getText());
```

- Declare a variable BillAmount of type double and initialize it with the value retrieved from the text field (using the getText() method) after converting it to type double (using the parseDouble() method)

```
switch(jComboBox1.getSelectedIndex())
```

- The index of the selected item is retrieved using the getSelectedIndex() method and on the basis of this value the control is transferred using switch statement

```
case 1: FinalAmount=0.90*BillAmount;
```

- If the second value in the combo box is selected then the FinalAmount is calculated by multiplying the BillAmount by 0.90 (to give a discount of 10%)

```
break;
```

- Stop the execution of the switch statement and transfer the control to

the statement immediately following the closing brace of the switch statement. It has to be included as the last statement of each case.

default:FinalAmount= BillAmount

- When `getSelectedIndex()` is not equal to either 1,2 or 3 then the code moves to default statement and no discount is given to the customer.

Comparing Switch and If..else Statements - Switch is used to select sections of code depending on specific integer or character values. If we are handling specific coded values (eg, the number of the button that was clicked in a `JOptionPane`), or processing characters(whose codes are treated like numbers), then switch is useful. The limitations of switch are as follows:

- It doesn't allow ranges, eg case 90-100.
- It requires either integers or characters and doesn't allow useful types like `String`.

```
        String comment; // The generated result.
int choice = Integer.parseInt(jTextField.getText);
        //Enter 0, 1, or 2.
switch (choice)
{
    case 0: comment = "You look so much better
        than usual."; break;
    case 1: comment = "Your work is up to its usual
        standards."; break;
    case 2: comment =
        "You're quite competent for so little
        experience."; break;
    default: comment =
        "Oops -- something is wrong with this code.";
}
```

Equivalent if statement

```
String comment; // The generated result.
int choice= Integer.parseInt(jTextField.getText);
```

```

        //Enter is 0,
1,or 2. if (choice == 0)
    comment = "You look so much better than
usual."; else if (choice == 1)
    comment = "Your work is up to its usual
standards."; else if (choice == 2)
    comment="You're quite competent for so little
experience"; else
    comment = "Oops -- something is wrong with this code.";

```

A switch statement can often be rewritten as an if statement. Let us look at the example given above, when a selection is to be made based on a single value, the switch statement is generally easier to read. The switch is useful when you need to manage a lot of **if/else if/else**. It has a shorter syntax and is more appropriate in this case.

Points to Remember:

- NetBeans is an IDE using which we can develop GUI applications in Java.
- NetBeans provides various components used to create a GUI front-end interface.
- GUI components' appearance and behaviour is controlled by their properties and methods.
- We should use meaningful names for controls on the form and variables in the code. It makes programming convenient.
- Some useful Data Types supported in Java are: int, double, char and boolean.
- String is an Object (reference) type supported in Java.
- A variable must be declared before it can be used.
- Different types of operators are available in Java. Operators are used to perform various operations on data.
- The if statement selects among a set of statements depending on the value of a controlling expression.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. What will be the final value of sum1 after the execution of the program given below?

```
int sum1 = 3; sum1=sum1+1;
jTextField1.setText(""+sum1);
sum1=sum1+1; jTextField2.setText(""+sum1);
sum1=sum1+1;
jTextField3.setText(""+(sum1));
sum1=sum1+1;
jTextField4.setText(""+sum1);
jTextField5.setText(""+sum1);
```

- 5 b. 6
c. 4 d. 7

Consider the following code snippet :

```
int anumber=14;
if (anumber >=10)
jLabel1.setText("first string");
else
jLabel1.setText("second string");
jLabel2.setText("third string");
```

What will be the output when anumber=14

- first string b. :ond string
first string d. :ond string
rd string rd string

3. What's wrong with the following statement?

```
if( (ctr < 5) && (ctr > 30) )
```

- a the logical operator && cannot be used in a test condition.
- b the test condition is always false.
- c the test condition is always true.

4. If there is more than one statement in the block of a if statement, which of the following must be placed at the beginning and the ending of the loop block?

- a parentheses ()
- b French curly braces { }
- c brackets []
- d arrows <>

5. Given the following information:

```
int a = 11;  
int b = 22; int c = 33; int d = 11;
```

Which of the following statements are true :

- i) $a = b$
 - ii) $b \neq d$
 - iii) $c \leq b$
 - iv) $a < c$
 - v) $a = d$
 - vi) $c > a$
 - vii) $a \geq c$
- a i), iv) & vii)
 - b ii), iv), v) & vi)
 - c ii), iv), vi) & vii)
 - d iii), v), vi) & vii)

ANSWER THE FOLLOWING QUESTIONS

1. Explain the following terms:

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

2. Differentiate between :

- a) TextField and TextArea

- b) ComboBox and ListBox
 - c) getText() and setText()
3. What is the significance of the following properties in
TextArea ? LineWrap WrapStyleWord
 4. What are list type controls used for ?
 5. How would you determine whether a combo box is editable or not?
 6. List different selection modes of a list.
 7. What is a button group? Which control is generally used with a buttongroup.
 8. Write and explain two methods each of check box and radio button.

LAB EXERCISES

1. Design a GUI application in which the user enters a three digit number in the text field and on clicking the button the sum of the digits of the number should be displayed in a label.

Hint : Suppose user enters 123 the output should be 6(1+2+3).
2. Design a GUI application to accept a number from the user in a text field and print using option pane whether it is a positive even number or not.
3. Design a GUI application to accept the cost price and selling price form the user in two text fields then calculate the profit or loss incurred.
4. Design a GUI application to accept a character in a text field and print in a label if that character is a vowel: a, e, i, o, or u. The application should be case sensitive.
5. Design a GUI application in java to convert temperature from Celsius to Fahrenheit or vice versa using radio buttons and two text fields
6. Design a GUI application in java to convert kilograms into grams, litres into milliliters, rupees into paisa using combobox and text fields.
7. A book publishing house decided to go in for computerization. The database will be maintained at the back end but you have to design the front end for the company. You have to accept book code, Title, Author and Quantity sold from the user. The Price will be generated depending upon the book code. Net price should be calculated on the basis of the discount given.

Book seller - 25%

School - 20%

Customer - 5%

8. A networking company decided to computerize its employee salary . Develop an application to store employee's personal data which will be saved in the back end. The front end should accept Name, Father's Name, Mother's Name, Address, Gender, Basic Salary, Medical and Conveyance. Calculate gross and net salary.

Basic	DA	HRA
>=40000	35%	37%
>=20000	25%	32%
>=10000	25%	30%