Control Structures

Learning Objectives

After studying this lesson the students will be able to:

- understand the concept and usage of selection and Iteration statements.
- appreciate the need and use of Relational and Logical operators.
- analyze the problem, decide and evaluate conditions.
- understand the need to use the Check Box, List and Combo Box components.
- design simple applications using the various selection control structures in an application.
- develop applications integrating GUI components and iterative control structures.

In all the applications that we have designed so far, the execution of the programs followed a sequence. All the commands or statements were executed from the beginning to the end, one after the other. But if we want to govern the flow of control in a program then we will require control statements. Control statements control the order in which the statements are executed. This chapter will introduce us to the two main categories of control statements namely Selection statements and Iteration statements

Selection Statements

Observe the form execution given in Figure 6.1 carefully and try to analyze the problem.

Figure 6.1 Sample Run of The Vote Eligibility Checker Application

Observing the sample run of the above application, it is clear that we have designed an application 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. 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 6.1.

Let us now try to write the code for the "Check Eligibility To Vote" button as given in Figure 6.2. The code for the STOP button is the same as learnt in previous chapters.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{

  // Code to check eligibility to vote:
  if (Integer.parseInt(jTextField1.getText())>=18)
   JOptionPane.showMessageDialog(null,"You are eligible To VOTE");
}
```

Figure 6.2 Code for the Vote Eligibility Checker Application

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.

Can you guess what will happen if the condition evaluates to false in the above application? Understanding the if statement completely will help us answer this question.

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. Thus, if statement is a selection construct as the execution of statements is based on a test condition. The selection statements are also called conditional statements or decision control statements.

The syntax of if statement is as shown below:

Syntax:

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

{

Statement Block;

}

! Do not use a semicolon after the parenthesis of the conditional expression of the if statement.

There are certain points worth remembering about the if statement as outlined below:

- 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.

After familiarizing with the if statement, can you now understand what will happen in the above application if the Age entered by the user is less than 18? Well in that case, the user will not get any message because we have not included the else clause in our code above. So, let us re-write the program to take care of both the cases i.e. Age>= 18 as well as Age <18. The modified code for this application is given in Figure 6.3.

```
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");
}
```

Figure 6.3 Code for the Vote Eligibility Checker Application (with else Condition)

Now, if the user enters the Age as 12, which is less than 18, then the message "You are NOT eligible To VOTE" gets displayed as shown in Figure 6.4.

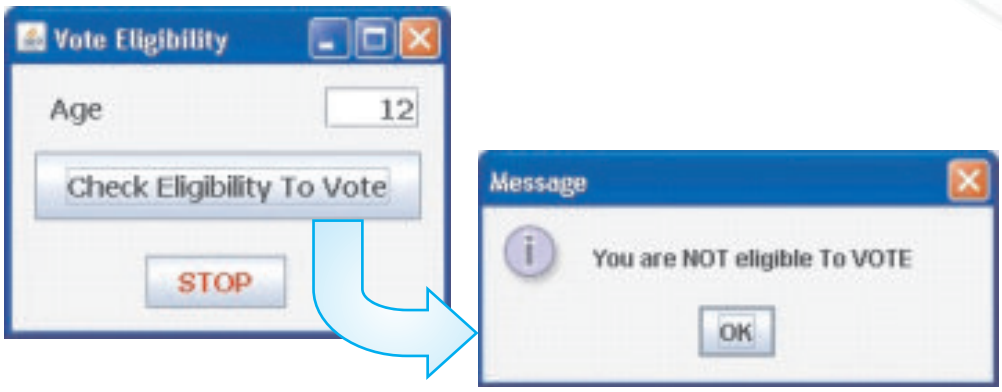


Figure 6.4 Sample Run of The Eligibility Checker Application when condition evaluates to false

Observe the following line extracted from the above code carefully:

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

In the above application we have used an operator >= to establish a relation between the value input by the user and the number 18. Such operators are called relational operators.

! if statement requires curly parenthesis {} for writing more than one number of statements in a statement block.

Know more

If your are comparing two values for equality and one is a constant, put the constant on the left side of the boolean statement. This will prevent accidental assigning of the constant value to the other variable if a single "=" is typed instead of the intended "==". The compilers often do not catch this error and it can lead to very strange problems that are very difficult to track down as shown in the following example.

```
if(x=true)   //assigns the value true to the variable x.  The true clause is always
             //executed, not what was intended.
if(true=x)   //Always generates a compiler error like "attempt to assign value to a
             // constant", error caught right away.
```

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.

Let us now design another application similar to the Vote Eligibility Checker, to further consolidate our understanding of the if statement. The Scholarship Eligibility application is to be developed to check whether a student is eligible for a scholarship or not. The student will be eligible for scholarship only if he/she scores more than or equal to 75 marks in aggregate. In this application, the message will be displayed in the text area instead of the dialog box as shown in Figure 6.5.

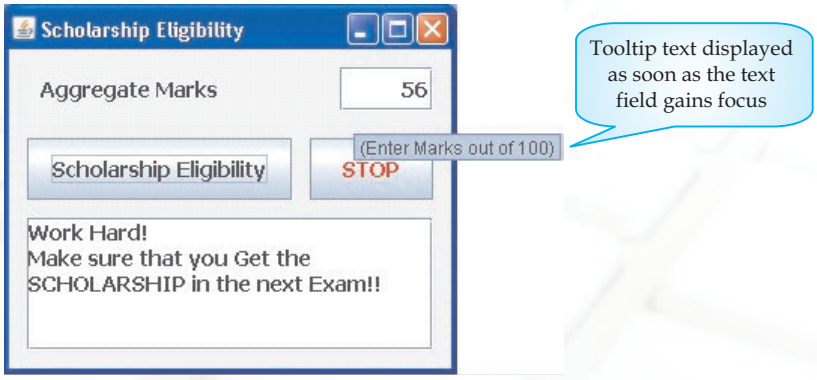


Figure 6.5 Sample Run of the Scholarship Eligibility Application when condition evaluates to false

In Figure 6.5 we can see that a message (Enter Marks out of 100) is displayed when we execute the application. To guide the user about the type of input that is required from them, the toolTipText property of the jTextField has been used.

Know more

The showMessageDialog() method can use null as well as this as the first parameter. Using this ensures that the message window is displayed on top of the window executing the method. On the other hand using null ensures that the message window is displayed in the centre of the screen irrespective of the window executing the method. Try doing it yourself.

ToolTip Text

The tooltip text is the text that appears when the user moves the cursor over a component, without clicking it. The tooltip generally appears in a small "hover box" with information about the component being hovered over. The property can be set using the Properties Window of the jTextField component as displayed in Figure 6.6

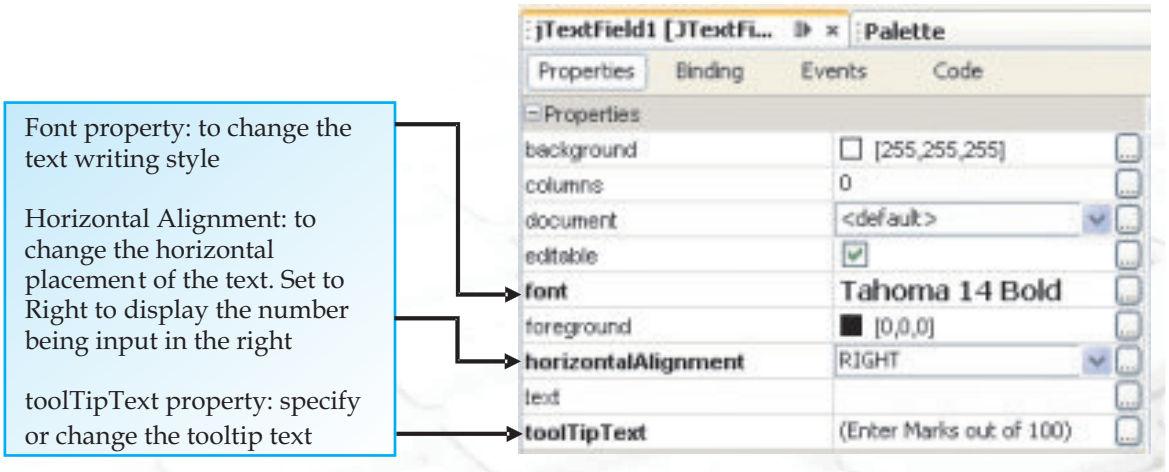


Figure 6.6 Changing the Properties of the JTextField Component

The code for the application is as given below in Figure 6.7:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{

  // Code to check eligibility for scholarship:
  if (Integer.parseInt(jTextField1.getText())>=75)
   jTextArea1.setText("Congratulation!\nYou Get the
SCHOLARSHIP!!");
  else
   jTextArea1.setText("Work Hard!\n" +
    "Make sure that you Get the SCHOLARSHIP in the next Exam!!");
}
```

Figure 6.7 Code for the Scholarship Eligibility Application

Let us now understand the code in detail.

If (Integer.parseInt(jTextField1.getText()) >=75)

jTextArea1.setText("Congratulation!\n You Get the SCHOLARSHIP!!")

- check whether the value retrieved from the text field is greater than or equal to 75 or not. The if statement is used to check the condition and if the condition evaluates to true then the message is displayed in the text area using the setText() method. The '\n' is used to explicitly insert a line break.

else

jTextArea1.setText("Work Hard!\n" +

"Make sure that you Get the SCHOLARSHIP in the next Exam!!")

- if the test condition evaluates to false i.e. in case aggregate marks are less than 75, then two messages - "Work Hard!" and "Make sure that you Get the SCHOLARSHIP in the next Exam!!" are concatenated (using + operator) and displayed in the text area using the setText() method. The '\n' is used to explicitly insert a line break.

After understanding the code clearly, we can easily predict that on entering aggregate marks as 89 the message "Congratulation! You Get the SCHOLARSHIP" gets displayed in the text area as shown in Figure 6.8.

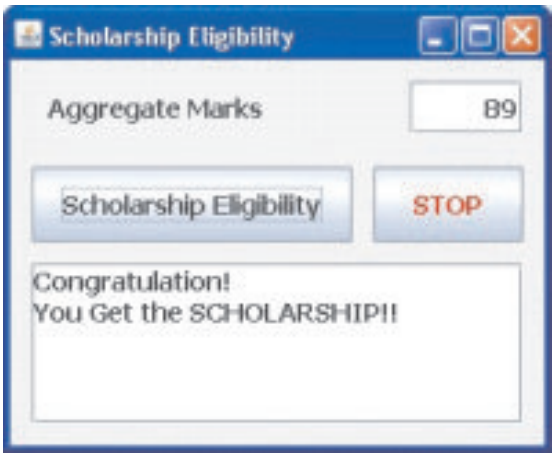


Figure 6.8 Sample Run of the Scholarship Eligibility Application when condition evaluates to true.

In both the applications above, a single test condition was taken based on the input accepted in a text field. What will we do if there are multiple conditions to be checked?

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.

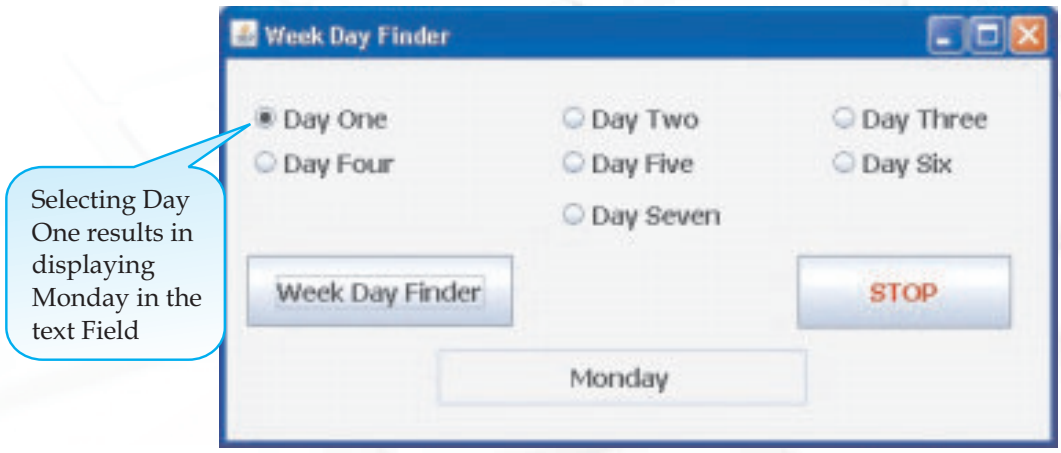


Figure 6.9 Sample Run of the Week Day Finder

Design the form as shown in Figure 6.9. and set the properties of the components according to the functionality required as shown in Figure 6.9. Monday is displayed when the radio button corresponding to Day One is selected as shown in Figure 6.9 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 6.10.

```
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("Sunday");
      else
       jTextField1.setText("Day - Not Selected");
}
```

Figure 6.10 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.

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;
}
```

Firstly, the conditional expression1 will be tested, if the condition evaluates to true then the statements1 block will be executed but if it evaluates to false then conditional expression2 will be tested. If the conditional expression2 evaluates to true then the statements2 block will be executed and so on. Whenever a condition is false, the program will continue examining the subsequent conditions until it finds the true one. On finding a true condition, its corresponding statement block is executed, and then the control is transferred outside the if statement. If none of the condition is true then the statement corresponding to else will be executed.

We have used radio buttons in the application designed above. Well radio buttons are a way of visually providing 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 the radio button occupies lot of space. So if there are too many options then it is advisable to use Combo box as they help save space and are less cumbersome to design as compared to radio button. But supposing we want to allow the user to select multiple options like while selecting favourite sports or ordering multiple food items in a restaurant. In such cases, we will use components like check box and list. 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. Now we will study each of these three components (Check Box, List and Combo box) one by one and side by side design applications to understand the working of each.

Check Box

Check boxes are similar to radio buttons but their selection model is different. Each Check box component works independently of each other and so the user can select any number of check boxes from an interface. A group of radio buttons, on the other hand, can have only one button selected. A Check box can be added from the Swing Control menu as shown in Figure 6.11.

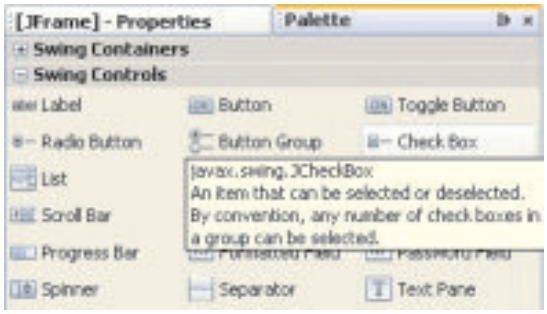


Figure 6.11 Check Box Element of the Swing Control

! The Add() Property is used to add a button (radio button or check box) to the button group at run time.

The properties window is used to change the properties of a Checkbox as shown in Figure 6.12.

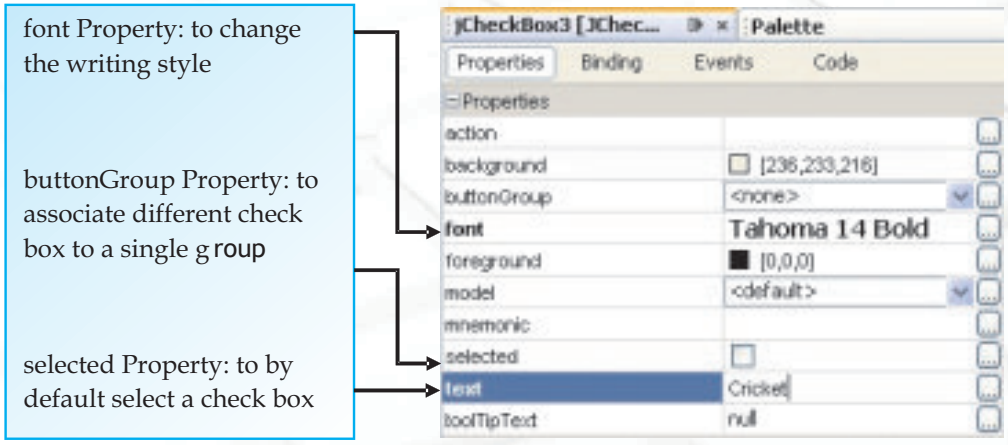


Figure 6.12 Common Properties of the JCheckBox Component

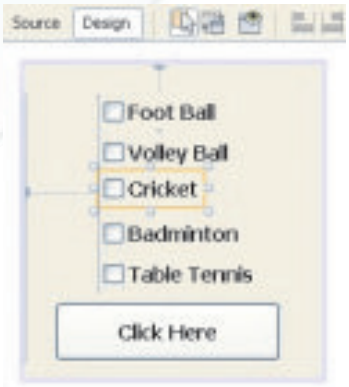


Figure 6.13 Aligning Check Box

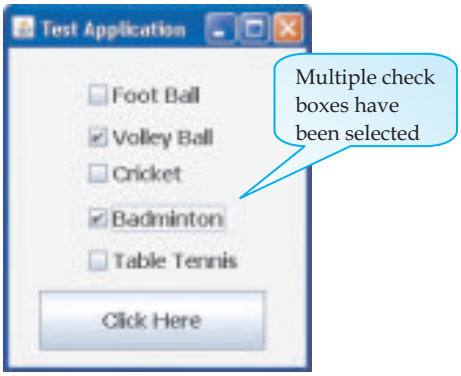


Figure 6.14 Selecting Multiple Check Boxes

! A rectangle with a tick mark means check box is selected.

Some commonly used methods of check box control are as follows:

Method	Description
getText()	Returns the text displayed by the checkbox
setText(String s)	Sets the text displayed by the check box to the String value specified in parenthesis.
isSelected()	Returns the state of check box - true if selected else returns false.
setSelected()	Sets the state of the button - true if the button is selected, otherwise sets it to false.

Figure 6.15 Form for the Sports Charges Application

Now let us try and develop a Sports Charges application to learn manipulation of check boxes. Design a form as shown in Figure 6.15. The aim of the application is to calculate the total charges payable by the user based on the sports selected. Note that all the text fields are disabled because they are just displaying the results and are not accepting any input from the user. The input is being taken in the form of selection of check boxes. On the selection of a particular sport check box, its charges are displayed in the

adjacent text field and on the click of the Calculate Charges button, the charges for all the selected sports are added and displayed in the text field.

! The check box components by default work independent of each other and so the user can select any number of checkboxes on an interface. But if the check boxes are grouped together under one single ButtonGroup then the check box component works like a radio button allowing only one single selection.

Now double click on the two buttons and enter the code as shown in Figure 6.16

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
        double Amount=0;
        if (jCheckBox1.isSelected())//Foot Ball Charges
        {
           jTextField1.setText("2500");
           Amount=Amount+2500;
        }
        if (jCheckBox2.isSelected())//Volley Ball Charges
        {
           jTextField2.setText("1500");
           Amount=Amount+1500;
        }
        if (jCheckBox3.isSelected())//Cricket Charges
        {
           jTextField3.setText("2000");
           Amount=Amount+2000;
        }
        if (jCheckBox4.isSelected())//Badminton Charges
        {
           jTextField4.setText("3000");
           Amount=Amount+3000;
        }
        if (jCheckBox5.isSelected())//Table Tennis Charges
        {
           jTextField5.setText("3500");
           Amount=Amount+3500;
        }
        jTextField6.setText(Double.toString(Amount));
    }
```

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

Figure 6.16 Code for the Sports Charges Application

Let us now understand the code in detail. Since the code in each if is almost the same except the amount being added, so we will understand a single if block.

double Amount=0;

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

if (jCheckBox1.isSelected())

- check whether the first check box is selected or not

if (jCheckBox1.isSelected())

{ jTextField1.setText("2500");

Amount=Amount+2500;}

- If the first checkbox is selected then display the coaching charges for the selected sport in the adjacent text field using the setText() method and then add these charges to the variable amount to calculate the total amount payable.

jTextField6.setText(Double.toString(Amount))

- The variable Amount contains the total amount which is a number. To display it in the text field, it needs to be converted to a string. This is achieved using the toString() method. It is then displayed in the text field using the setText() method. The calculated Amount is displayed outside all the if statements because we want to display it only once after the user has selected all the possible options. If we want to display the total amount after each selection then we need to include this statement inside each if statement.

Know more

*The expression [Amount = Amount + 3500;] can also be written as [Amount+=3500;]. In the same way -=, *= and /= can also be used to simplify the expressions*

List

A List(also called list box) component displays a list of values/options from which single or multiple values/items can be selected. When we place a list on JFrame form the default model property of the list (default values in the list) has values as Item1, Item2 and so on as shown in Figure 6.17. The selectionMode property is set to MULTIPLE_INTERVAL by default ensuring that a user can select multiple items from the list. These properties can be changed using the properties window as shown in Figure 6.18

Figure 6.17 Default Values of a List

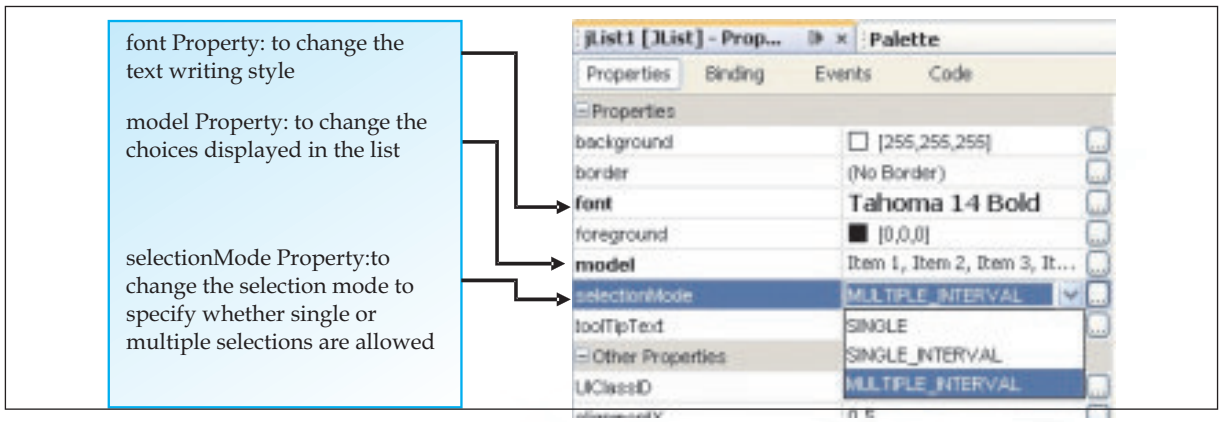


Figure 6.18 Common Properties of the List Component

As shown in the figure 6.18, the selectionMode property has three possible values. The usage of each of these values is explained below:

- SINGLE implies that List box will allow only a single value to be selected.
- SINGLE_INTERVAL implies that List box allows single continuous selection of options using shift key of keyboard (i.e. values which occur in succession).
- MULTIPLE_INTERVAL implies that List box allows multiple selections of options using ctrl key of keyboard.

The model property is used to change the choices displayed in the list. The values can be updated by clicking on the ellipsis(..) next to the property in the properties window as displayed in Figure 6.19 .

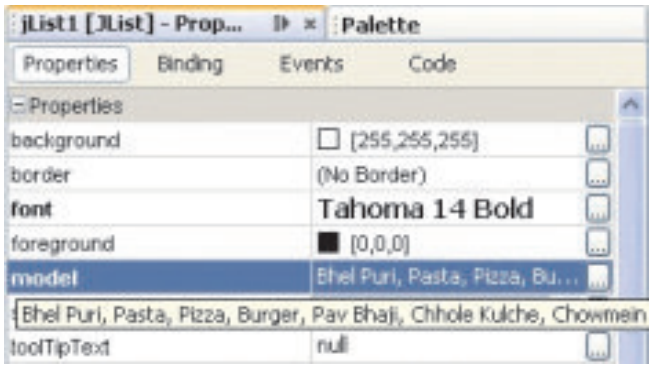


Figure 6.19 Modifying the Model Property

Modifying the model property of the jList component results in a change in the values displayed in the list as shown in Figure 6.20.

Figure 6.20 A simple List

Let us now design an application Restra Order using the food list created to help us understand how to use a list component in an application. Design the form as shown in the Figure 6.21. The form consists of a list, a button, a text field and two labels - one for explaining the selection process and one for indicating that the payable amount is in rupees.

The aim of the application is to allow the user to place an order for multiple items displayed in the list and display the bill amount in the text field which will be calculated on the basis of the items selected. The menu options are shown in Figure 6.21.

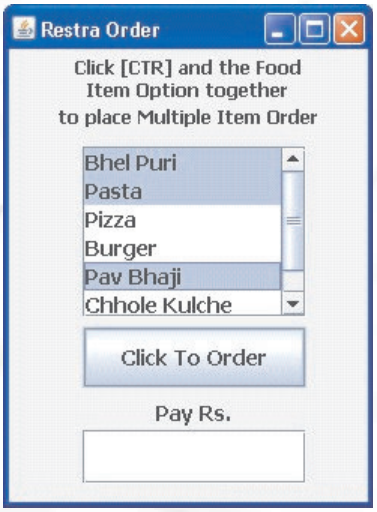


Figure 6.21 Form Design of Restra Order Application

When the user clicks on the Click to Order button the total amount is calculated and displayed in the text field along with the price message boxes for each individual item ordered as shown in figure 6.22.

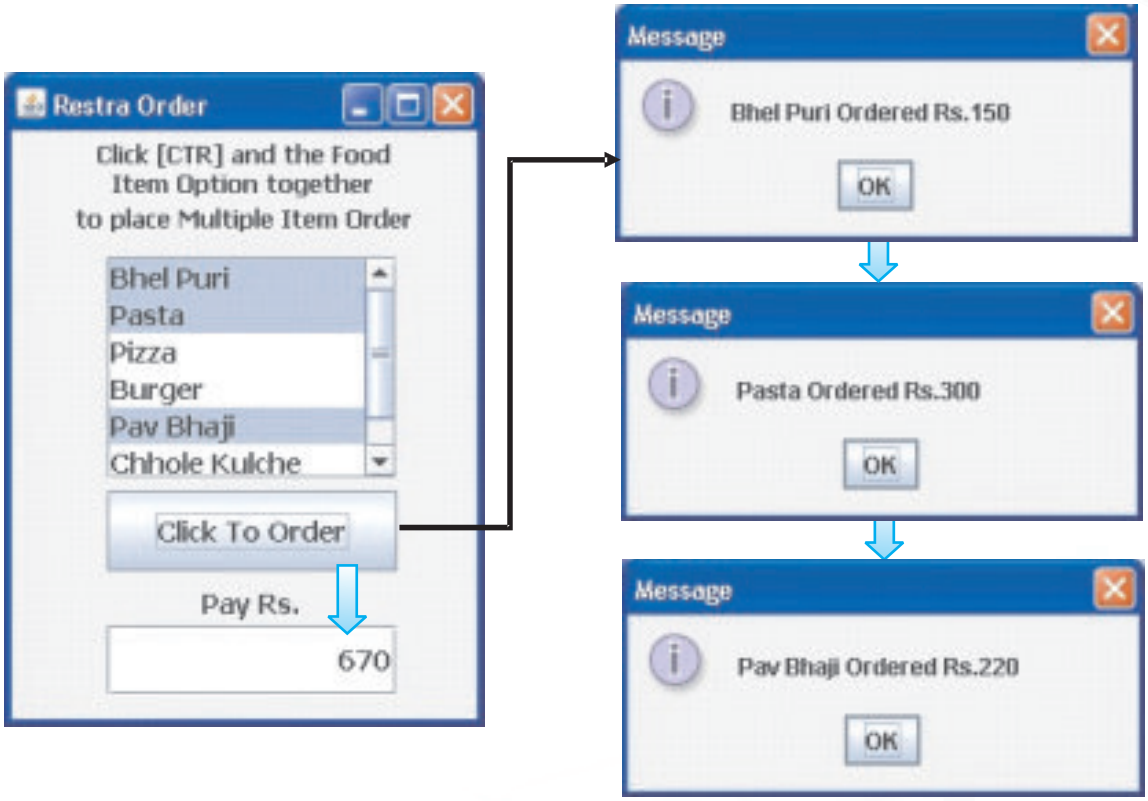


Figure 6.22 Sample Run of the Restra Order Application

! Items/Values in a list have an index value associated with them. First Item/Value in the list has the index 0, second item/value has index 1 and so on. Thus, the index of an item is one less than its position in the list.

! If there is one statment, many programs use braces to make the code more robust. This is a safer practice because any later addition of a statement to one of the clauses will require braces. If you don't have the braces with multiple statements, the compiler may not give any error message, but your code will not do what is expected.

Let us now write the code for the above application as shown in figure 6.23.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
   int Total=0;
//Bhel Puri, Pasta, Pizza, Burger, Pav Bhaji, Chhole Kulche,
Chowmein
   if (jList1.isSelectedIndex(0)==true)
   {
     Total=Total+150;
     JOptionPane.showMessageDialog(this,"Bhel   Puri   Ordered
Rs.150");
   }
   if (jList1.isSelectedIndex(1)==true)
   {
     Total=Total+300;
     JOptionPane.showMessageDialog(this,"Pasta   Ordered
Rs.300");
   }
   if (jList1.isSelectedIndex(2)==true)
   {
     Total=Total+200;
            JOptionPane.showMessageDialog(this,"Pizza  Ordered
Rs.200");
   }
   if (jList1.isSelectedIndex(3)==true)
   {
     Total=Total+180;
           JOptionPane.showMessageDialog(this,"Burger  Ordered
Rs.180");
   }
   if (jList1.isSelectedIndex(4)==true)
   {
     Total=Total+220;
        JOptionPane.showMessageDialog(this,"Pav Bhaji Ordered
Rs.220");
   }
   if (jList1.isSelectedIndex(5)==true)
   {
```

```
      Total=Total+260;
      JOptionPane.showMessageDialog(this,
                                          "Chhole Kulche Ordered
Rs.260");
    }
    if (jList1.isSelectedIndex(6)==true)
    {
      Total=Total+260;
      JOptionPane.showMessageDialog(this,"Chowmein   Ordered
Rs.260");
    }
    jTextField1.setText(Integer.toString(Total));
```

Figure 6.23 Code for the Restra Order Application

The above code introduces us to a new method - isSelectedIndex() method. This method is used to check whether the index specified in the parenthesis has been selected or not. The syntax of this method is given below:

Syntax:

```
jList.isSelectedIndex(int num)
```

The num is an integer value and represents the index value to be checked. The index numbering starts at 0. This method returns a boolean value i.e. true or false. The true indicates that the value at the specified index is selected and false indicates that the value is not selected.

Now let us understand the code in detail. The code for checking the List items selection is similar so we will concentrate on understanding one of them and the last line.

int Total=0

- Declare a variable of type integer to store the total amount payable by the user and initialize it to 0. This variable has been initialized to 0 as initially the user has not selected any item and so the total amount payable by him is 0.

if (jList1.isSelectedIndex(0)==true)

- check whether the first option in the list is selected or not

if (jList1.isSelectedIndex(0)==true)

```
{
  Total=Total+150;
  JOptionPane.showMessageDialog(this,"Bhel Puri Ordered Rs.150");
}
```

- If the first option in the list is selected, then the rate of the selected item is added to the total rate. A message is then displayed with the name of the selected item and its rate.

```
jTextField1.setText(Integer.toString(Total));
```

- After the check is evaluated for all the items, the total amount to be paid by the customer is displayed in the text field using the setText() method. The total amount is an integer value so before displaying the value it is converted to a string type using the toString() method.

Commonly used methods of List control are as follows:

Method	Description
getSelectedValue()	Returns the selected value when only a single item is selected. If multiple items are selected then it returns the first selected value. Returns null in case no item is selected
isSelectedIndex(int index)	Returns true if specified index is selected.

Combo Box

This control is used to display a list of choices from which the user can choose a single option. The difference between combo box and list box control is that a list box control allows user to make one or more selections whereas a combo box control allows the user to make single selection.

Figure 6.24 A simple Combo Box

When we place a combo box on the JFrame form by default it shows Item1 as the first value as shown in Figure 6.24. A Combo box appears like a text field with a drop down list arrow.

The common properties of the Combo Box can be edited using the properties window as shown in Figure 6.25.

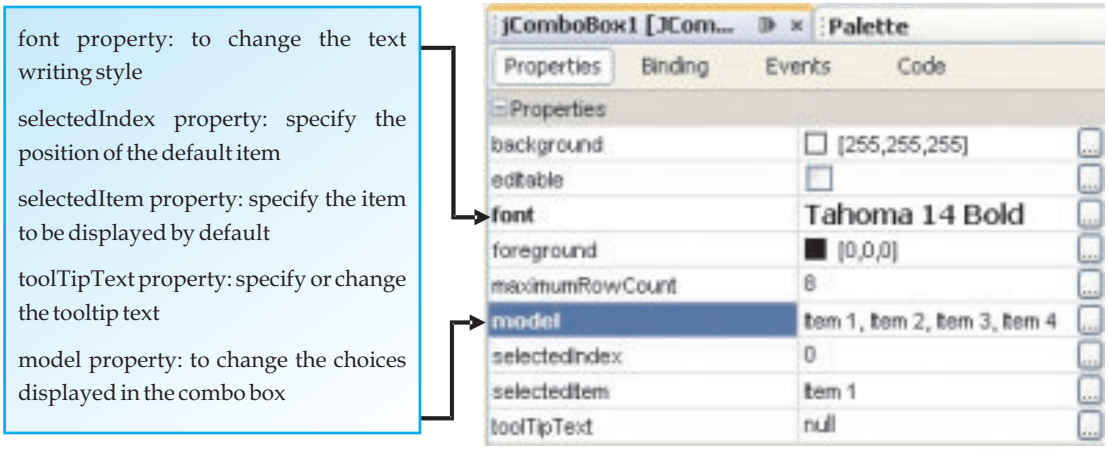


Figure 6.25 Common Properties of the Combo Box Component

The default values displayed in a combo box are Item1, Item 2 and so on. These can be edited by clicking on the ellipse(…) next to the values. Let us create a combo box having the name of cities. Drag the Combo Box component from the Swing Controls tab and then type the items that we want to be displayed in the combo box by clicking on the model property ellipse button. The new values we typed in are shown in Figure 6.26.

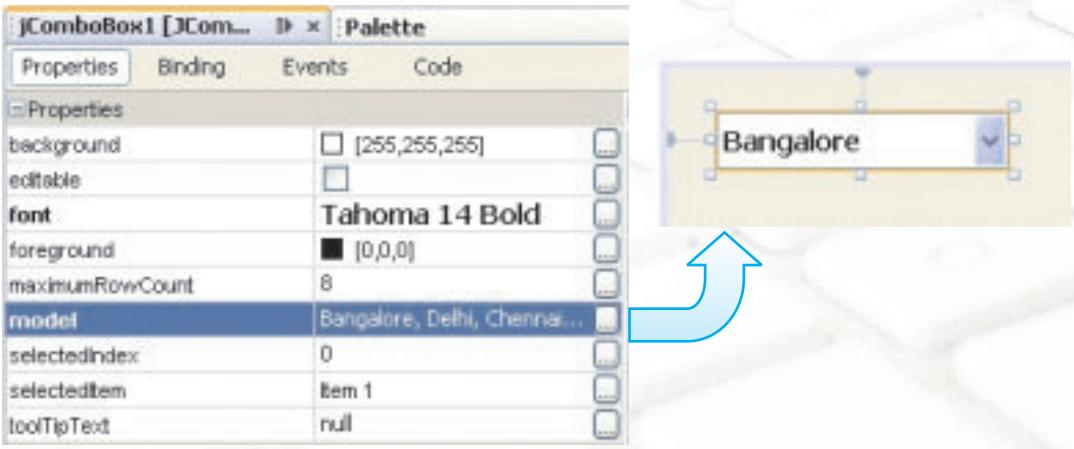


Figure 6.26 Modifying the model Property of a Combo Box

Now Bangalore is the first value in the model property therefore on the form Bangalore will be displayed with a drop down list arrow as shown in Figure 6.27.

Let us design an application called City Highlights to learn the usage of combo box. Design a simple form with a combo box (containing names of 5 cities) and a button with display text as "*Know More*". The required functionality is that on executing the application City Highlights, the user should select a particular city and click on the button to view some additional information about the city. Sample run of the application is shown in Figure 6.27.

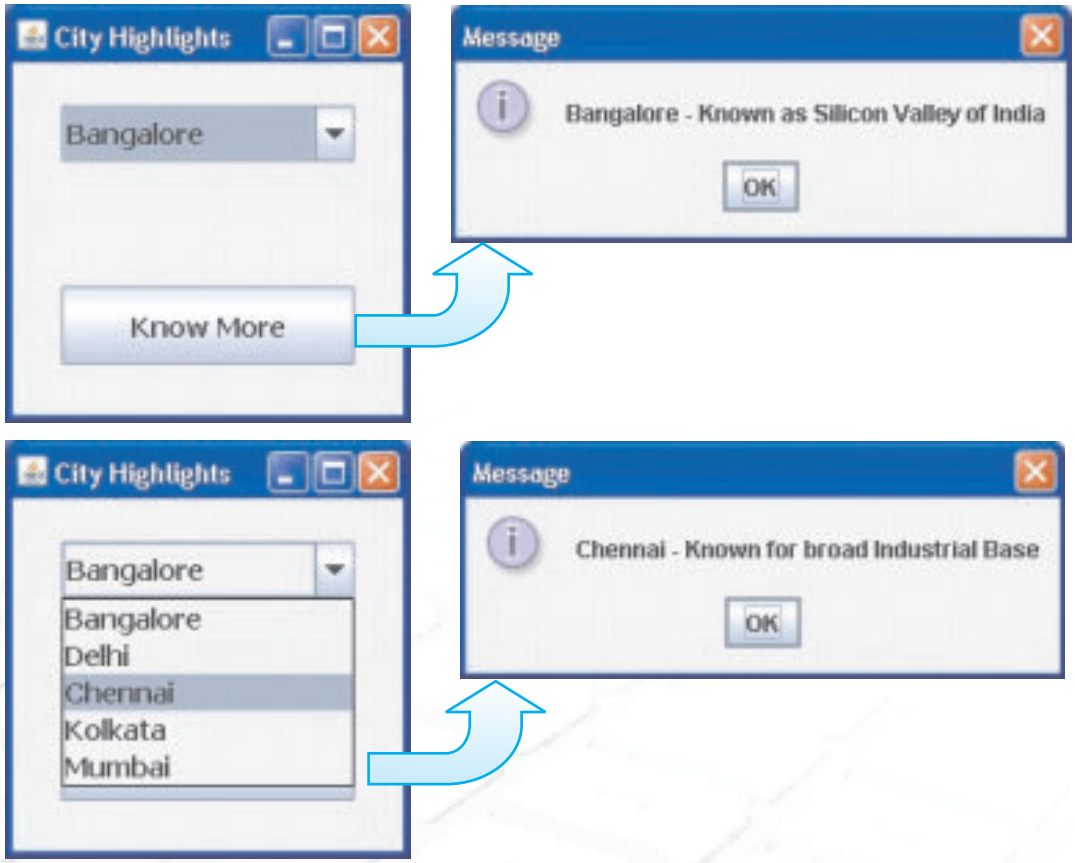


Figure 6.27 Sample Run of the City Highlights Application

The code for the City Highlights application is as shown in Figure 6.28.

```
private void jButton1ActionPerformed
(java.awt.event.ActionEvent evt) {
  if (jComboBox1.getSelectedIndex()==0)
  JOptionPane.showMessageDialog(this,jComboBox1.getSelectedIte
m()+
                                    " - Known as Silicon Valley of
India");
  else if (jComboBox1.getSelectedIndex()==1)
   JOptionPane.showMessageDialog(this,jComboBox1.getSelectedI
tem()+
                              " - Capital City of India");
  else if (jComboBox1.getSelectedIndex()==2)
   JOptionPane.showMessageDialog(this,jComboBox1.getSelectedI
tem()+
                           " - Known for broad Industrial
Base");
  else if (jComboBox1.getSelectedIndex()==3)
   JOptionPane.showMessageDialog(this,
jComboBox1.getSelectedItem()+
   " - Known for literary, artistic and revolutionary heritage");
  else if (jComboBox1.getSelectedIndex()==4)
   JOptionPane.showMessageDialog(this,jComboBox1.getSelectedI
tem()+
                         " - Known for hub of Bollywood");
}
```

Figure 6.28 Code for the City Highlights Application using if else if

This code introduces us to two new methods, the getSelectedIndex() method and the getSelectedItem() method. The syntax and usage of each of these methods is explained below:

1. getSelectedIndex() - This method is used to return the index of the selected item. If an item is selected only then will the getSelectedIndex method return a value else it returns -1. The syntax of this method is given below:

   Syntax:

   ```
   jComboBox.getSelectedIndex()
   ```

2. getSelectedItem() - This method is used to return the selected item. The syntax of this methos is given below:

   Syntax:

   ```
   jComboBox.getSelectedItem()
   ```

Now let us understand the code in detail.

```
if (jComboBox1.getSelectedIndex()==0)
```

- Checks whether the item stored at the first position is selected or not using the getSelectedIndex() method

```
if (jComboBox1.getSelectedIndex()==0)
  JOptionPane.showMessageDialog(this, jComboBox1.getSelectedItem()+
      " - Known as Silicon Valley of India");
```

- If the item stored at the first position is selected then the name of the item is retrieved using the getSelectedItem() method and is concatenated with a message using the concatenation operator(+). The concatenated message is then displayed in a dialog box using the showMessageDialog() method.

```
if (jComboBox1.getSelectedIndex()==0)
  JOptionPane.showMessageDialog(this,jComboBox1.getSelectedItem()+
      " - Known as Silicon Valley of India");
else if (jComboBox1.getSelectedIndex()==1)
```

- If the item stored in first position is not selected then it checks for the item stored in the second position and follows the same procedure.

As is clear from the previous two applications, the nested if else becomes difficult to read and understand as the number of testing conditions goes on increasing. So we introduce a new selection statement - the switch statement. Figure 6.29 shows the code of City Highlights application using switch statement. Observe the code carefully and try to understand the code.

! While deciding between using an if statement and a switch statement always remember that although switch is easier to read but it can only be used to test for equality. The if statement on the other hand can test for equality as well as inequality

```
private  void  jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
  switch (jComboBox1.getSelectedIndex())
  {
  case 0:JOptionPane.showMessageDialog(this,
         jComboBox1.getSelectedItem()+
         " - Known as Silicon Valley of India");
         break;
  case 1:JOptionPane.showMessageDialog(this,
         jComboBox1.getSelectedItem()+
         " - Capital City of India");
         break;
  case 2:JOptionPane.showMessageDialog(this,
         jComboBox1.getSelectedItem()+
         " - Known for broad Industrial Base");
         break;
  case 3:JOptionPane.showMessageDialog(this,
         jComboBox1.getSelectedItem()+
               " - Known for literary, artistic and revolutionary
heritage");
         break;
  case 4:JOptionPane.showMessageDialog(this,
         jComboBox1.getSelectedItem()+
         " - Known for hub of Bollywood");
         break;
  default:JOptionPane.showMessageDialog(this,"No City Selected");
  }
}
```

Figure 6.29 Code for the City Highlights Application using switch

In the above code, we match the value of the test expression jComboBox1.getSelectedIndex()) with the values written in the different case statements. On finding a match, the statement corresponding to that case gets executed. In case none of the case values match the value of the test expression, then the statement corresponding to the default clause is executed.

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 ;
}
```

! Always include a default clause in your switch statement.

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 6.30.

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.

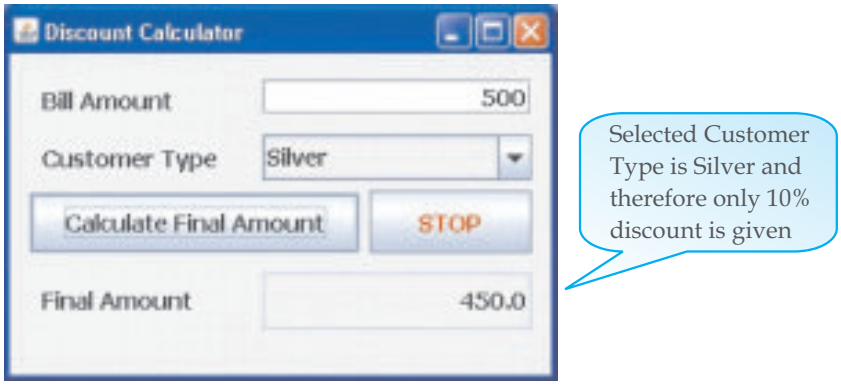


Figure 6.30 Discount of 10% for Customer Type Silver

Let us now write the code for the discount calculator as shown in 6.31.

```
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));
}
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
        // To exit the application:
        System.exit(0);
}
```

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.

In all the above applications, the test condition was a simple expression. Now let us develop another application, the Grade Calculator Application, where we will learn

how to handle a complex test condition. We will calculate grade and check eligibility for an Achiever's Award. If marks in General Knowledge and Analytical Skills are greater than 90 then the child is eligible for Achiever's award. The rules for finding grade are as follows:

Marks	Grade
Above 80	A
Between 79.9 and 70	B
Between 69.9 and 60	C
Between 59.9 and 50	D
Below 50	E

The first step is to design a form as shown in Figure 6.32 with the following components:

- 3 enabled text fields to accept the marks in 3 subjects with appropriate labels
- 3 disabled text fields to display the total, the grade and an indicator for achievers award
- 2 enabled buttons, one to calculate the Total marks and one to exit from the application
- 1 disabled button to find the grade which will be enabled during run time when the total is calculated.

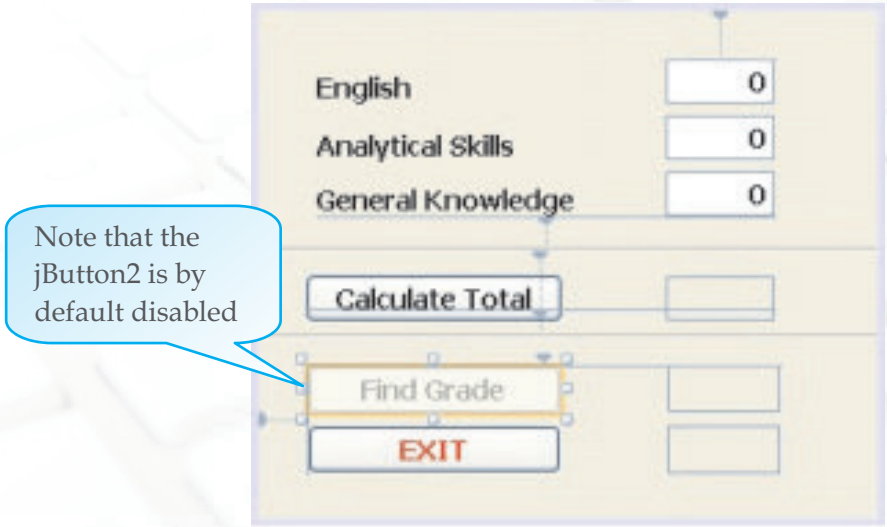


Figure6.32 Design of the Grade Calculator Application

Observe the sample executions of the Grade Calculator to understand the functionality required before writing the code.

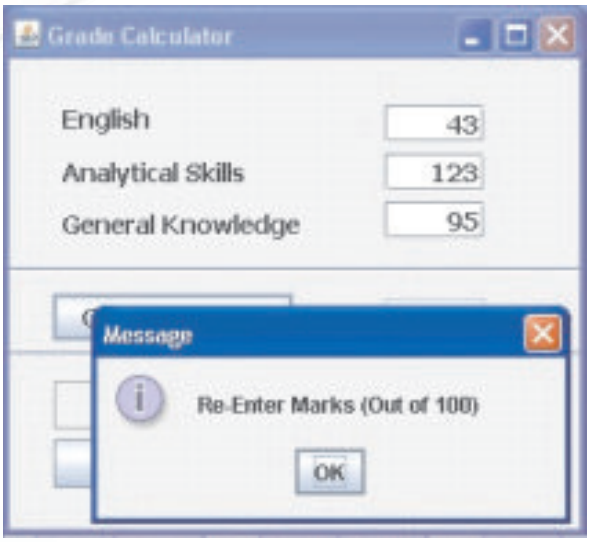


Figure 6.33 Error Handling Message

When the application is executed and the user enters marks greater than 100 either for English, Analytical Skills or General Knowledge, a message box asking the user to re-enter the marks is to be displayed as shown in Figure 6.33.

When the user enters valid marks, and clicks on the Calculate Total button, the total is displayed in the adjacent text field and the Find Grade button is enabled as shown in Figure 6.34.

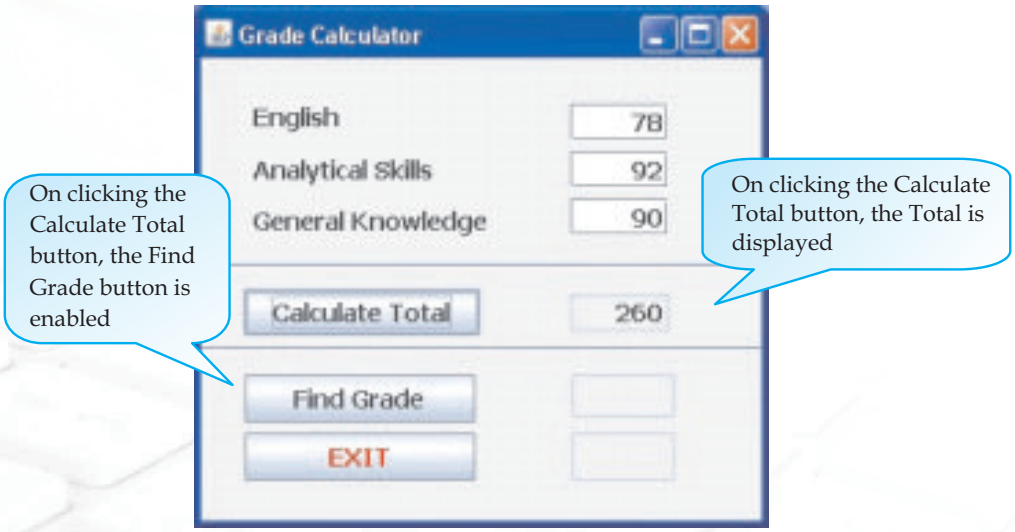


Figure 6.34 Effect of Clicking on the Calculate Total Button of the Grade Calculator

On clicking the Find Grade button, firstly a check is performed to find out whether the child is eligible for Achiever's award and an appropriate message is displayed and a * is displayed in the text field adjacent to the EXIT button as shown in Figure 6.35. Then the grade is calculated according to the criteria mentioned above and is displayed in the adjacent text field.

On clicking the EXIT button the application will terminate.

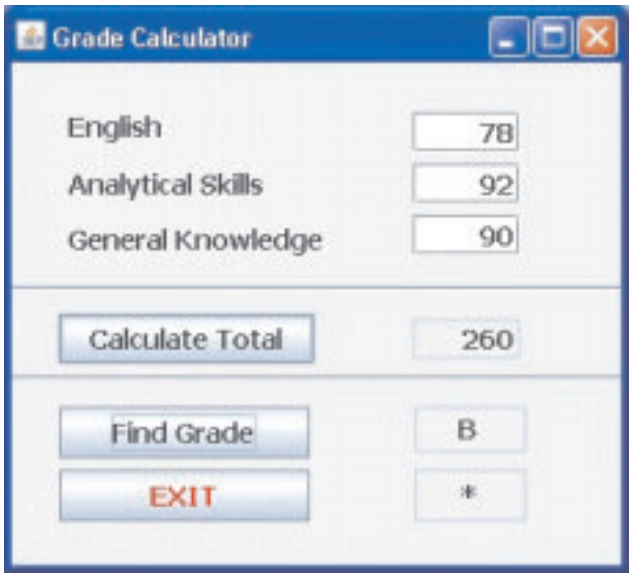


Figure 6.35 Grade and eligibility for Achiever's award displayed

Since the code requires us to join many conditions so we will use a few operators like || and &&. Let us understand their usage briefly before writing the code.

```
if (English>100 || ASkills>100 || GK>100)
JOptionPane.showMessageDialog(this,"Re-Enter Marks (Out of 100)");
```

- Check if the marks input by the user for any of the subjects are greater than 100 or not an if they are then display the message "Re-Enter Marks (Out of 100). Since we have to display the error message if the marks of even one subject are out of limit so we have used the || operator which means OR. So in simple english it means if marks of English are >100 or marks of ASkills >100 or marks of GK > 100, then display the error message. So the message will be displayed even if only one condition evaluates to true.

```
if (ASkills>=90 && GK>=90 )
{
    JOptionPane.showMessageDialog(this,"** Selected for Achiever's Award **");
     jTextField6.setText("*");
}
```

- Check if the marks of ASkills and GK are both >= 90 or not. If they are then display the message "** Selected for Achiever's Award **" and also display a "*" in the text field. Since we have to check that both the marks should be greater than 90 so we have use the && operator which in simple English means AND. So the condition will evaluate to true only if both the conditions are satisfied.

Let us now write the code for the Grade calculator application as shown in Figure 6.38

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
// Variable Declaration and assignment operations
   int Total,English,ASkills,GK;
   English=Integer.parseInt(jTextField1.getText());
   ASkills=Integer.parseInt(jTextField2.getText());
       GK=Integer.parseInt(jTextField3.getText());

       //Validation of Entered Marks
       if (English>100 || ASkills>100 || GK>100)
       JOptionPane.showMessageDialog
(this,"Re-Enter Marks (Out of 100)");
       else
   {
       Total=English+ASkills+GK;
           jTextField4.setText(Integer.toString(Total));
           jButton2.setEnabled(true);
       }
}
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
   // Variable Declaration and assignment operations
       char Grade;
       int ASkills,GK,Total;
ASkills=Integer.parseInt(jTextField2.getText());
       GK=Integer.parseInt(jTextField3.getText());
       Total=Integer.parseInt(jTextField1.getText());
```

```
    //Decision for Achiever's Award
        if (ASkills>=90 && GK>=90 )
        {
        JOptionPane.showMessageDialog
                    (this,"** Selected for Achiever's Award **");
            jTextField6.setText("*");
        }
    //Finding Grade
        if (Total>=80)
        jTextField5.setText("A");
        else if (Total>=70)
            jTextField5.setText("B");
        else if (Total>=60)
            jTextField5.setText("C");
        else if (Total>=50)
        jTextField5.setText("D");
        else
        jTextField5.setText("E");
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent
evt) {
    // To Exit from application
        System.exit(0);
}
```

Figure 6.38 Code for the Grade Calculator Application

Since in this application we had to test for multiple conditions in a if statement, so we had to join the conditions using some operators. Such conditions that are formed by joining simple conditions are called complex conditions and they are usually joined using the logical 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

Now we are quite thorough with the working of the conditional statements. Let us now develop a Natural Number printer wherein we accept a number from the user and print all the natural numbers till that number as shown in the Figure 6.39.

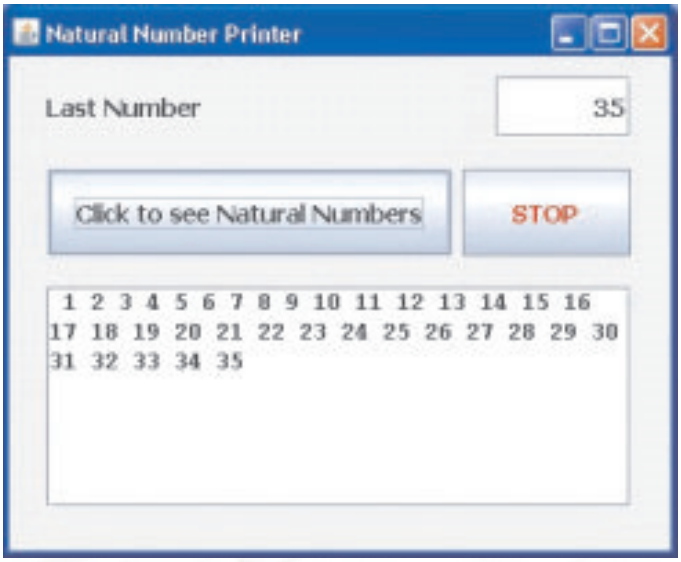


Figure 6.39 Sample Run of the Natural Number Printer

Can you imagine what is happening in this example? We are simply going on concatenating a new natural number to the old contents. How many times we are concatenating depends on the last number input by the user. But are we actually going to repeatedly write the same command multiple times? The answer is no. We are simply going to use Iteration statements as displayed in the code in Figure 6.40.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {

  // Code to print natural numbers :
  int LastNumber=Integer.parseInt(jTextField1.getText());
  for (int I=1;I<=LastNumber;I++)
    jTextArea1.setText(jTextArea1.getText()+
                                    " "+Integer.toString(I));
}
```

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // To Exit from the application
    System.exit(0);
}
```

Figure 6.40 Code for the Natural Number Printer Application using for loop

In this example we have used the Iteration statement - for loop which is the only new concept introduced in this code. Let us understand the code in detail and later we will look at the working and complete syntax of the for loop.

int LastNumber=Integer.parseInt(jTextField1.getText());

- A variable named LastNumber is declared which is of type integer. This variable needs to contain the last number of the series which has been input by the user. The value input by the user in the text field is retrieved using the getText() method. This value is a string and so is converted to an integer value using the parseInt() method. After conversion it is assigned to the variable LastNumber.

for (int I=1;I<=LastNumber;I++)

- The loop control variable is defined and initialized to 1 (int I=1). The loop iterates till the test condition I<=LastNumber evaluates to true and each time at the end of the loop, the loop control variable is incremented by 1 (due to I++).

for (int I=1;I<=LastNumber;I++)

jTextArea1.setText(jTextArea1.getText() + " " + Integer.toString(I));

- Every time the loop executes we convert the number I to a string using the toString() method. This is concatenated with the previous contents of the text area which are retrieved using the getText() method. The empty string (" ") is concatenated in between the two contents to leave a blank space between each consecutive number displayed. Finally the concatenated string is displayed in the text area using the setText() method.

! When you declare a variable inside a for loop, there is one important point to remember: the scope of that variable ends when the for statement ends. (That is, the scope of the variable is limited to the scope of for loop.)

Now let us look at the syntax and working of the for loop in detail.

Iteration Statements

These statements are used to perform a set of instructions repeatedly until the condition is fulfilled. Iteration statements are also called looping statements.

for loop

The for loop operates as follows. The loop starts, with the execution of the initialization portion. This sets the value of the loop control variable, which acts as a counter that controls the loop. Then the condition is evaluated, wherein the loop control variable is checked with a target value. If this expression evaluates to true, then the body of the loop is executed. If it is false, the loop terminates. After one execution of the loop, the iteration portion of the loop is executed. This is usually an expression which increments or decrements the loop control variable. The loop then iterates, until the controlling expression evaluates to false. The syntax of the for loop is:

Syntax

```
for( initialization; test exp; increment/decrement exp)
{
    statements;
}
```

The loop has four different elements that have different purposes. These elements are:

a) Initialization expression: Before entering in a loop, its variables must be initialized. The initialization expression helps to initialize loop variable with its initial value. The initialization expression is executed only once in the beginning of the loop.

b) Test Expression: The test expression decides whether the loop body will be executed or not. If the test condition is true, the loop body gets executed otherwise the loop is terminated. Test expression gets checked every time before entering in the body of the loop.

c) Increment/Decrement Expression: The Increment/Decrement expression changes the value of the loop variable. The increment/decrement expression is executed every time after executing the loop body.

d) The Body of the loop: The statements, which are executed repeatedly till the test expression evaluates to false form the body of the loop.

Know more

The three expressions inside the round braces of for loop are optional. Using this fact an infinite loop can be created as follows:

```
for (;;) // infinite loop
{
    // Your code goes here
}
```

In the above code while changing the value of the loop variable, we have used an operator namely ++. This operator is used to simply increment the loop variable by 1. Such operators, which work on a single operand, are called unary operators.

Unary Operators

The unary operators perform different kind of operations on a single operand .The operations performed are increasing/decreasing a value, negating a value/ expression, or inverting a boolean value.

Symbol	Name of the Operator	Operation	Example
+	Unary plus operator	indicates positive value	num = +1;
-	Unary minus operator	negates an expression	num = - num;
++	Increment operator	increments a value by 1	num = ++ num;
--	Decrement operator	decrements a value by 1	num = -- num;

Increment/Decrement Operators

The increment/decrement (++,--) operators can be a prefix or a postfix. In a pre increment/decrement expression (++ x or -- x), an operator is applied before an operand while in a post increment/decrement expression (x++ or x --) an operator is applied after an operand. In both conditions 1 is added to the value of the variable and the result is stored back to the variable. However, in a prefix expression, value is incremented first

then this new value is restored back to the variable. In postfix expression the current value is assigned to a variable then it is incremented by 1 and restored back to the original variable. The working of the pre increment and post increment is illustrated in Figure 6.41

```
int Number=1000;
Number++;     //Post increment in an independent statement
OR
int Number=1000;
++Number;     //Pre increment in an independent statement
              //will have the same meaning
-------------------------------------------------------------
int Total=50,Number=10;
Total=Total + Number++; //post increment -
                        //first the Total is increased by the
                        //current value of Number and then
                        //Number is incremented by 1
                        //So, after execution of the expression
                        //Total will be 60 and Number will be 11
-------------------------------------------------------------
int Total=50,Number=10;
Total=Total + ++Number; //pre-increment -
                        //first the Number gets incremented
                        //by 1 and then gets added to Total
                        //So, after execution of the expression
                        //Total will be 61 and Number will be 11
```

Figure 6.41 Working of pre increment and post increment

Let us now develop another application to print even numbers or odd numbers depending on the option selected by the user. If the user does not select any option a message is displayed to the user prompting the user to select either even or odd numbers as shown in Figure 6.42.

Figure 6.42 Displaying an error message

When the user enters a number in the text field and selects the odd numbers radio button, all odd numbers till the number entered by the user are displayed as shown in Figure 6.43. Similarly, if the user selects the even numbers radio button, all even numbers till the number entered by the user will be displayed in the text area.

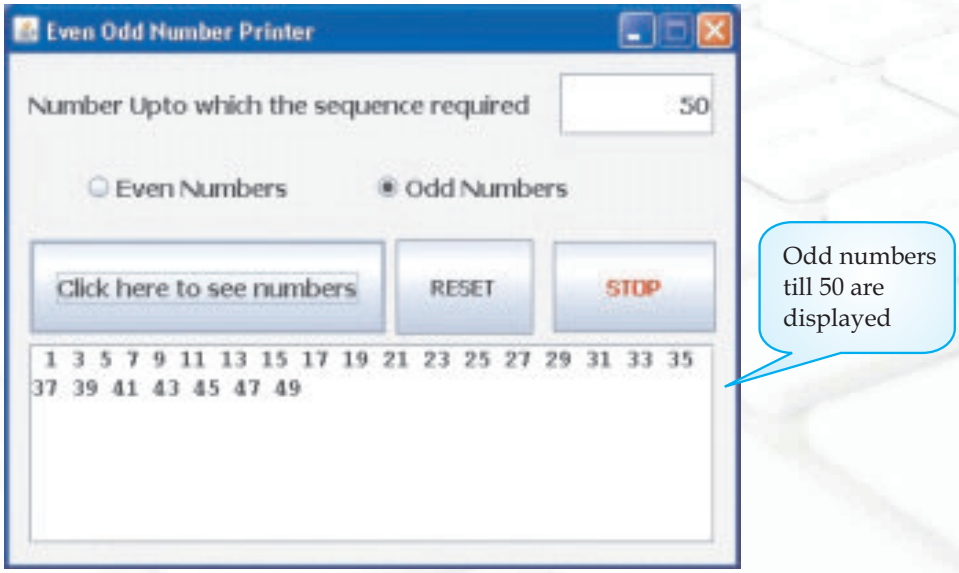


Figure 6.43 Sample run of the Even Odd Number Printer Application

The *RESET* button should clear both the text fields and also deselect both the radio buttons. The *STOP* button should terminate the application. Let us now write the code for each of the three buttons as shown in Figure 6.44.

```
private void jButton1ActionPerformed(
java.awt.event.ActionEvent evt) {

  int LastNumber=Integer.parseInt(jTextField1.getText());
  if (jRadioButton1.isSelected())  //Even Numbers required
  {
   for (int I=2;I<=LastNumber;I+=2)
     jTextArea1.setText(jTextArea1.getText()+
                                  " " +Integer.toString(I));
  }
  else if (jRadioButton2.isSelected())//Odd Numbers required
  {
    for (int I=1;I<=LastNumber;I+=2)
      jTextArea1.setText(jTextArea1.getText()+
                                  " " +Integer.toString(I));
  }
  else
    JOptionPane.showMessageDialog(this,
                "Click to select [Even] or [Odd] Option");
}
```

```
private void jButton2ActionPerformed
(java.awt.event.ActionEvent evt) {
    // Code for Reset button :
     jTextField1.setText("");
     jRadioButton1.setSelected(false);
     jRadioButton2.setSelected(false);
     jTextArea1.setText("");
}
```

```
private void
jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{
   System.exit(0);
}
```

Figure 6.44 Code for the Even Odd Number Printer Application

The above code introduces us to a new method setSelected(). This method is used to set the selection status of a radio button or a check box. The syntax for this method is given below:

Syntax

```
jRadioButton.setSelected(Boolean b)
```

Since the method has to set the state so it needs a boolean value to be supplied. The value b should be true if the button is to be selected and false if the button is to be deselected.

Also note the way in which the loop control variable has been incremented. The statement used is I +=2 which is equivalent to writing I = I + 2. This simply means that the loop control variable is incremented by 2 each time and this has been done to reach the successive odd or even number. The explanation of each line of the above code is left as an exercise.

Now we will learn another two loop statements named while loop and do while loop. The working of both these loops is similar though there is a slight difference between. Observe the code given in Figure 6.40 for the Natural Number Printer Application and then observe the codes given below in Figure 6.45 and Figure 6.46 for the same application.

```
private void jButton1ActionPerformed
(java.awt.event.ActionEvent evt) {

  int LastNumber=Integer.parseInt(jTextField1.getText());
  int i=1 ;                  // loop variable initialized
  while(i<=LastNumber)
  {

jTextArea1.setText(jTextArea1.getText()+""+Integer.toString
(I));
    i=i+1;
  }
}
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
  System.exit(0);
}
```

Figure 6.45 Code for Natural Number Application using while Loop

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {

  int LastNumber=Integer.parseInt(jTextField1.getText());
  int i=1 ;                  // loop variable initialized
  do
  {

jTextArea1.setText(jTextArea1.getText()+""+Integer.toString(I)
);
    i=i+1;
  } while(i<=LastNumber)
}
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
  System.exit(0);
}
```

Figure 6.46 Code for Natural Number Application using do while Loop

On close observation we will see three major differences in the code using for loop and the other two loops which are enumerated below:

1. The loop control variable is declared and initialized outside the loop for both variations.
2. The for keyword has been replaced by the while keyword in the first case and the test condition immediately follows the while keyword. In the second case the for keyword is replaced with do keyword in the beginning of the loop body and the while keyword has been added at the end of the loop body. Again the test condition immediately follows the while keyword.
3. The loop control variable has been incremented inside the loop in both cases.

The rest of the code remains exactly the same. Now let us first understand the syntax of the while statement and the do while statement and then we will develop an application to understand when we will prefer while over for loop.

while Statement

The while loop is an entry-controlled loop. It means that the loop condition is tested before executing the loop body. If the loop condition is initially false, for the first iteration, then loop may not execute even once. The main characteristic of the while loop

is that it can be used in both cases i.e. when the number of iterations is known as well as when it is unknown. The syntax of the while loop is as follows:

Syntax

```
while(test expression)
{
    loop body
}
```

The loop-body can contain a single statement, a compound statement or an empty statement. The loop iterates till the test expression evaluates to true. When the expression becomes false, the program control passes to the statement following the loop. Remember that in while loop, a loop control variable should be initialized before the loop begins and the loop variable should be updated inside the body of the while loop (else it will become an endless loop).

do while Statement

In the do while loop, the test occurs at the end of the loop. This ensures that the do while loop executes the statements included in the loop body at least once. After the first execution of the statement, it evaluates the test expression. If the expression evaluates to true, then it executes the statements of the loop body again. It will go on executing the statements as long as the condition is true. Once the condition becomes false, the loop will terminate. Do while loop is an exit controlled loop. Like if and while statements, the condition being checked must be included between parenthesis. The do while statement must end with a semicolon. The syntax of the loop is as follows:

Syntax

```
do
{
    loop body
}while (test expression);
```

The difference between do-while and while is that do-while evaluates its expression at the end of the loop instead of at the beginning. Therefore, the statements within the do block are always executed at least once.

Let us now develop a Member Counter application in which we accept names from the user and display these names in the text area on the click of a button as shown in Figure 6.47. The first step is to analyze the problem. When the user clicks on the button Click to START an input dialog box prompts the user to enter the Member Name. After entering the member name, when the user clicks on OK the member name is added to the text area and the total number of the members is displayed in the Members Counter text field as shown in Figure 6.48. The user is then asked to confirm whether he wants to continue or not. If the user clicks on Yes then the Enter Member Name dialog box again prompts the user for a new member name. The Member name entered is then appended in the Text Area as shown in Figure 6.48.

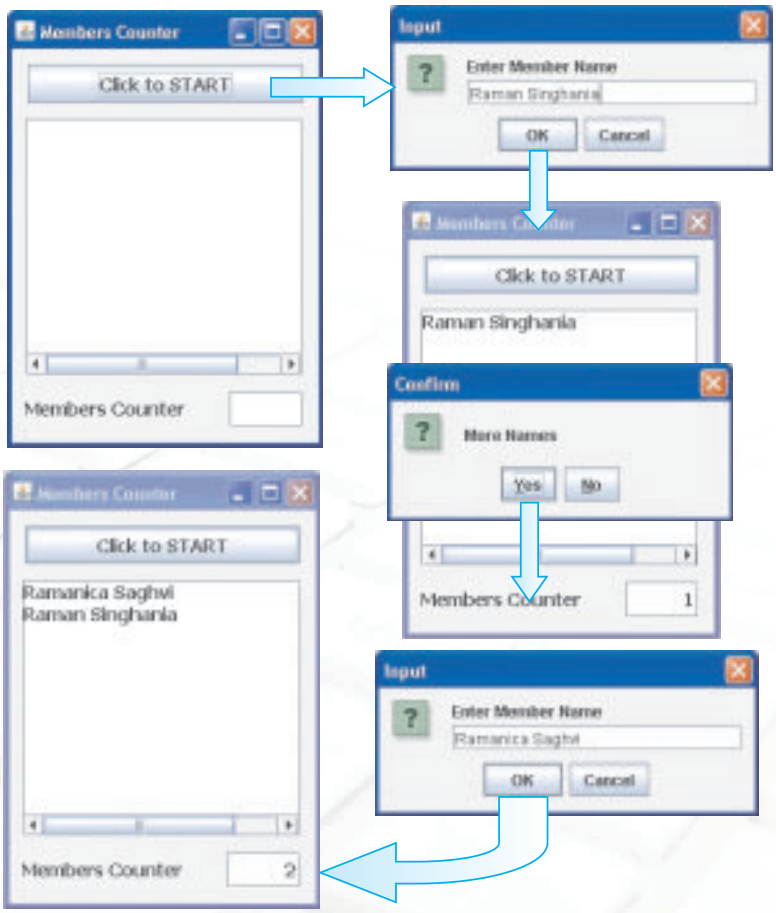


Figure 6.48 Sample Run of Member Counter Explaining the Flow of Operation

The process continues till the user terminates by clicking on the No button in the "More Names" dialog box.

Now think, Can we use for loop for this application? The answer is no because for is a deterministic loop in which the number of iterations should be pre known. So we may use the while loop or the do while instead. Since we want the application to run atleast once so we will use the do while loop.

Let us enter the code for the Click to START button as shown in Figure 6.49.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)

{
   int More,CountName=0;
   String Name;
   do
   {
     //Asking the User to Enter Name of new member
     Name=JOptionPane.showInputDialog("Enter Member Name");

     //Adding New Name in the List
     Name=Name+"\n"+jTextArea1.getText();

     //Displaying new List of Names in TextArea
     jTextArea1.setText(Name);

     //Incrementing in the count of members
     CountName=CountName+1;

     //Re-Displaying the count of Members
     jTextField1.setText(Integer.toString(CountName));

     //Confirm if more names to added or not
     More=JOptionPane.showConfirmDialog(this,
          "More Names","Confirm",JOptionPane.YES_NO_OPTION);
  }
  while (More==0);
}
```

Figure 6.49 Code for the Members Counter Application

The above code introduces us to two new methods namely, showInputDialog() and showConfirmDialog(). The use of both the methods is explained below:

showInputDialog() - This method is used to accept a value from the user using a Dialog Window along with an appropriate message to guide the user about what has to be

entered. The method returns the value input by the user and so can be used on the right side of an assignment statement.

```
Name=JOptionPane.showInputDialog("Enter Member Name");
```

showConfirmDialog() - This method is used to display a message in a Dialog window and prompts the user to select [YES] or [NO]. The method returns 0 if the user clicks on Yes and returns 1 if the user clicks on No in the Dialog Window.

```
More=JOptionPane.showConfirmDialog(this,
"More Names","Confirm",JOptionPane.YES_NO_OPTION);
```

Let us now understand the code in detail.

int More, CountName=0;

- Declare two integer variables named More (which stores whether the user wishes to continue or not) and CountName (which stores the total number of members) and initialize CountName to 0.

String Name;

- Declare a string variable called Name to store the name input by the user. (Later this same variable is used to store all the names of the text area)

Name=JOptionPane.showInputDialog("Enter Member Name");

- The name accepted from the user using a dialog box is stored in the variable name

Name=Name+"\n"+jTextArea1.getText();

- Retrieve the value of the text area using the getText() method (The text area contains all the previous member names) . Then concatenate this value with the name accepted from the user. The "\n" is used so that each name appears on a separate line.

jTextArea1.setText(Name);

- The concatenated string containing all the member names (variable Name) is displayed in the text area using the setText() method

CountName=CountName+1;

- The counter variable containing the total member count is incremented by one

```
jTextField1.setText(Integer.toString(CountName));
```

- The variable CountName is a numeric value so it is converted to a string using the toString() method and then the value is displayed in the text field using the setText() method

```
More=JOptionPane.showConfirmDialog(null,
    "More Names","Confirm",JOptionPane.YES_NO_OPTION);
```

- Ask the user to confirm if the application is to continue

```
while (More==0);
```

- Continue the iteration till the user selects No from the confirm dialog box. When the user selects No, value returned is 1 and so the variable More becomes one, thereby terminating the loop

Summary

- A program statement can execute in three ways: sequence, selection, iteration.
- Selection statements test conditions and selectively execute code depending on the outcome of the test condition.
- The if statement tests a condition. If the condition evaluates to true, the statements in the if part are executed. If the condition is false then the statements in else part get executed.
- Nested if statements - These statements are used to test multiple conditions.
- RadioButton control is used to give user a facility to select or deselect an option. RadioButton controls are dependent on each other (when used as a part of single ButtonGroup), so user can have an option to select only one RadioButton, which are part of the same ButtonGroup. Remember, ButtonGroups when dragged into the form, are invisible swing controls on the form, we require to make RadioButton part of a ButtonGroup from the property window.
- Check box control is used to give user facility to select or deselect one or more

than one option. Check box controls work independent of each other so user can select any number of checkboxes on an interface (when they are not part of a single ButtonGroup).

- Combo box controls are used to select an option from a list of choices.
- List box controls are used to select an option/multiple option from a list of choices.
- A switch is a multiple branch selection statement that can be used as a replacement for if statements testing multiple conditions.
- Iteration statements repeat a set of statements till a test condition is satisfied.
- for and while loops are entry controlled loop.
- do while is an example of exit controlled loop.
- A brief summary about all the methods learnt in this lesson is given in the table below:

Method	Syntax	Usage
isSelected()	component.isSelected()	To check whether a particular radio button or checkbox is selected or not.
setSelected()	component.setSelected (Boolean b)	Sets the state of the button at run time. setSelected(true) should be used if the button is to be set as selected, otherwise use setSelected(false).
isSelectedIndex()	component.isSelectedIndex (int num)	To check whether the index specified in the parenthesis has been selected or not. The num is an integer value and represents the index value to be checked. The index numbering starts at 0.

getSelectedValue()	Component.getSelected Value()	Returns the selected value when only a single item is selected. If multiple items are selected then it returns the first selected value. Returns null in case no item is selected
getSelectedIndex()	component.getSelected Index()	To return the index of the selected item. If an item is selected only then will the getSelectedIndex method return a value else it returns -1.
getSelectedItem() showInputDialog()	component.getSelectedItem() Object.showInputDialog ("text")	To return the selected item. To accept a value from the user using a Dialog Window along with an appropriate message to guide the user about what has to be entered.
showConfirm Dialog()	object.showConfirmDialog (Component parent Component, Object message, String title, int optionType)	To display a message in a Dialog window and prompts the user to select [YES] or [NO]. The method returns 0 if the user clicks on Yes and returns 1 if the user clicks on No in the Dialog Window.

Multiple Choice Questions

1. Which of the following is a selection construct ?

a. do while Loop b. for Loop

c. while Loop d. None of these

2. If there are two or more possible options then we can use:

a. simple if statement b. nested if statement

c. while loop d. None of the above

3. A loop that never ends is called a :

a. continue loop b. infinite loop

c. circle loop d. None of these

4. Statements in a block statement are enclosed in:

a. () Round braces b. [] square braces

c. {} Curly braces d. None of these

5. A group of statements which get executed based on a condition is called:

a. selection b. sequential

c. iteration d. none of these

6. Which of the following is an exit controlled loop?

a. for loop b. do while Loop

c. while loop d. none of these

7. How many times, the following loop gets executed?

```
i=0;
while (i> 20)
{
//Statements
}
```

a. Zero number of times b. Infinite number of times

c. Once d. none of these

8. How many times, the following loop gets executed?

```
i=0;
do
{
    //Statements
}while (i> 20);
```

a. Zero number of times b. Infinite number of times

c. Once d. none of these

Exercises

1. What is the difference between selection and repetition?
2. What is the purpose of if statement? Describe the different forms of if statement.
3. What is the purpose of default clause in a switch statement?
4. What is the main difference between a while loop and a do while loop?
5. What will be the content of jTextField1 after executing the following code:

```
int Num = 6;
Num = Num + 1;
if ( Num > 5)
   jTextField1.setText(Integer.toString(Num));
else
   jTextField1.setText(Integer.toString(Num+5));
```

6. What will be the corresponding outputs of the following code segment if the possible inputs in the jTextField1 are:

(i) 10 (ii) 20 (iii) 30 (iv) 40 (v) 50

```
String str = jTextField1.getText();
Number = Integer.parseInt(str);
Switch (Number)
{
    case 10:jTextField1.setText("Ten Thousand");break;
    case 20:jTextField1.setText ("Twenty Thousand");
    case 30:jTextField1.setText ("Thirty Thousand"); break;
    case 40:jTextField1.setText ("Forty Thousand");
    default:jTextField1.setText ("Not enough!!!");
}
```

7. Find the output of the following code:

```
int First = 7;
int Second = 73;
    First++;
    if (First+Second > 90)
            jlabel1.setText("value is 90 ");
    else
            jlabel1.setText("value is not 90 ");
int Number1 = 7,Number2=8;
int Second = 73;
    if (Number1>0 || Number2>5)
          if (Number1>7)
                jTextField1.setText("Code Worked");
      else
                jTextField1.setText("Code MightWork");
    else
            jTextField1.setText("Code will not Work");
```

8. **What is the main difference between a combo box and a list box?**

9. **How many times will the following loop get executed?**

```
x =  5;
y  = 36;
while ( x <= y)
{
 x+=6;
}
```

10. **What will be the content of the jTextArea1 after executing the following code?**

```
int Num = 1;
do
{
    jTextArea1.setText(Integer.toString(++Num) + "\n");
    Num = Num + 1;
  }while(Num<=10)
```

11. **Explain the use of for statement along with its syntax.**

12. **What are relational operators? Explain with the help of suitable examples.**

Lab Exercises

Design GUI applications for the following:

1. Develop an application to take input from user in a radio button out of the two referring to Area or Perimeter of a circle. Print the Area or Perimeter in a TextField for the value of Radius entered in another TextField.

2. Develop an application to take an input in TextField for a number. If the number is even then Display its square otherwise its cube in a MessageBox.

3. Develop an application to calculate area of a circle, a rectangle or a triangle depending upon the user's choice (from a set of Radio Buttons). Accept the desired input Radius OR Length-Bredth OR Side as per the option selected by the user.

4. An electronic shop has announced the following seasonal discounts on the purchase of certain items

Purchase Amount In Rs	Discount on TV	Discount on Music System
0-25000	5%	10%
25001-50000	10%	20%
More than 50000	15%	30%

Develop an application based on the above criteria, to input amount of purchase and the type of purchase (TV or Music System using JRadioButton) by a customer.

Compute and print the net amount to be paid by a customer along with his name accepted in a text field.

[Hint: Discount = (Discount rate / 100) * Amount of purchase

Net amount = amount of purchase - discount).]

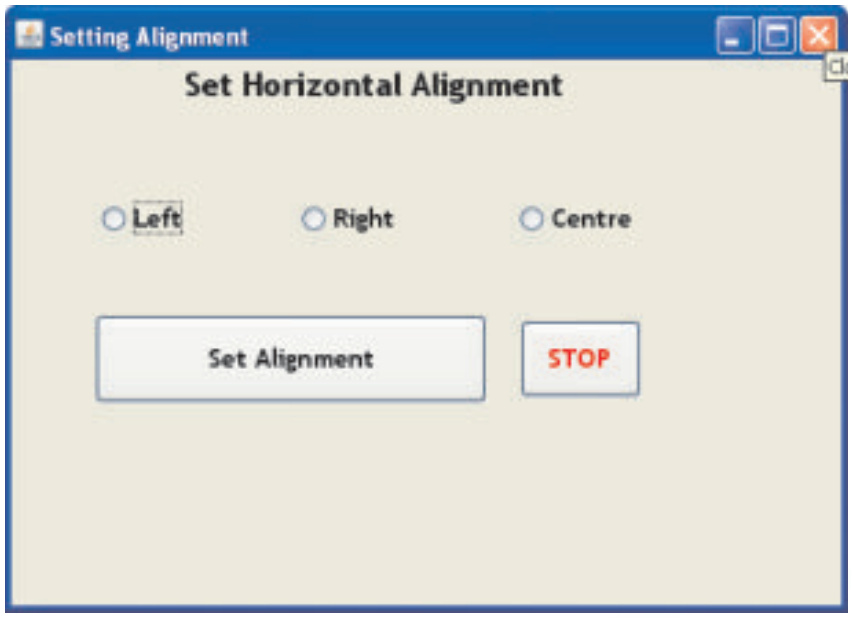
5. Define a GUI application to create a list box ClassName with the following values.

Class Name
XII A
XII B
XII C
XII D

Write a program to print the names of the class teacher according to the class selected based on the following information

Class Name	Class Teacher
XII A	Purnima singh
XII B	Suruchi Oberoi
XII C	Manjula
XII D	Anita Misra

6. Design a GUI application as shown below: On selecting the radio button and clicking the Set Alignment button the alignment of the text on the button gets changed to Left, Right or Centre.



[Hint use the setHorizontalAlignment method. To set the alignment to right we can use setHorizontalAlignment(SwingConstants.RIGHT).