Programming Fundamentals

Learning Objectives

After studying this lesson the students will be able to:

- declare, initialize and assign values to variables.
- write simple applications using variables.
- understand the concept and usage of different data types.
- appreciate the importance and usage of Arithmetic and Assignment operators.
- develop simple applications using different data types,

In the previous chapter, we developed GUI applications with some simple arithmetic operations. Now, we will introduce the concept of variables, which will simplify our efforts of performing complex arithmetic operations. Variables, as the name suggests are those identifiers, which can vary, i.e. can have different values. In programming, variables help us to hold values for some input coming from the user or to hold intermediate result of some calculation or the final result of an operation. In simple terms, variables are like containers that can be used to store whatever values are needed for a specific computation. However, as different materials require different containers, similarly different data types are required to associate the variables to store different types of values. This chapter will give us a good idea of variables and various data types.

Variables

Observe the form given in Figure 5.1 carefully and try to analyze the problem.

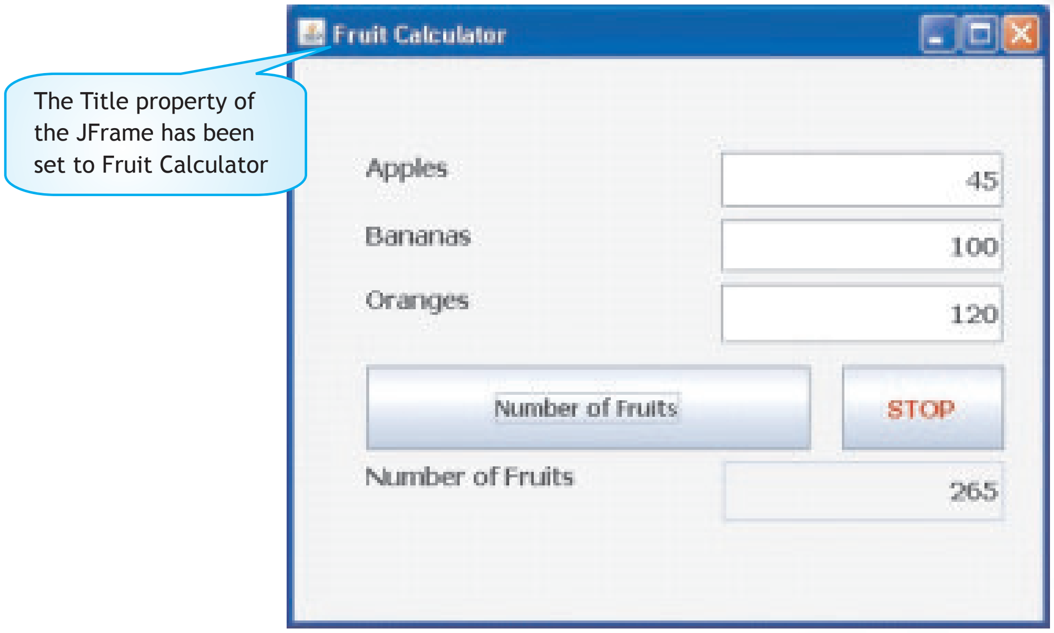


Figure 5.1 A Simple Form to Calculate Total Number of Fruits

After observing the above form, it is clear that we are accepting the number of apples, bananas and oranges in three separate text fields and calculating the total number of fruits by simply adding the three values on the click of a button. The total number of fruits is then displayed in a separate text field. The single line code for this simple application is given in Figure 5.2.

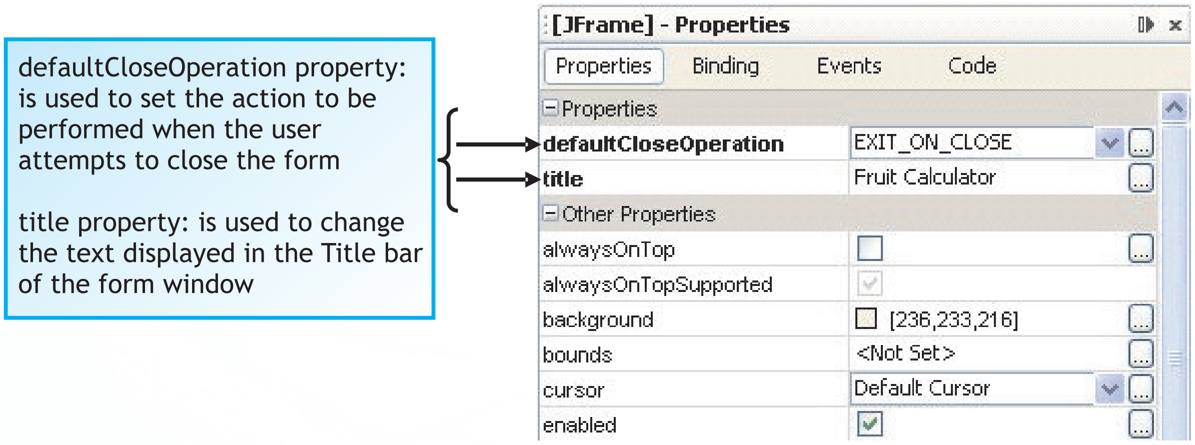
```
jTextField4.setText
(
  Integer.toString
  (
  Integer.parseInt(jTextField1.getText())
  +Integer.parseInt(jTextField2.getText())
  +Integer.parseInt(jTextField3.getText())
  )
);
```

This code can be written in a single line but has been written in 3 lines for better readability and understanding

Figure 5.2 Code to Add Values Accepted in Three Text Fields and Display Result in Fourth Text Field

Now imagine a situation where we have to calculate the total of 20 such fruits. Our one line of code will become very cumbersome and difficult to understand. To avoid such cumbersome code we need to use some containers, which can store the values entered by the user in the different text fields. These values need to be stored only till we add them up. So we need to modify the code given above. To test the same we first need to design the form and then associate code with the click of the button.

Let us first talk about the design of the form. Add a new JFrame form. Go to the Properties tab in the Palette Window and change the title property of the Form as shown in the Figure 5.3.



defaultCloseOperation property: is used to set the action to be performed when the user attempts to close the form

title property: is used to change the text displayed in the Title bar of the form window

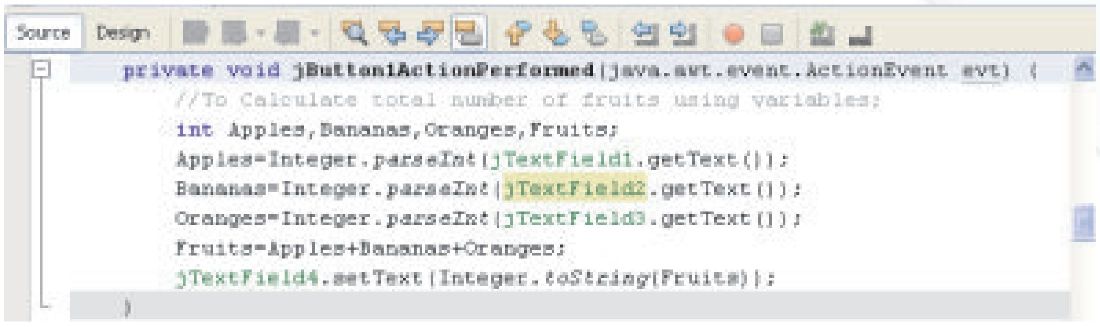
Properties	
defaultCloseOperation	EXIT_ON_CLOSE
title	Fruit Calculator
Other Properties	
alwaysOnTop	☐
alwaysOnTopSupported	✓
background	☐ [236,233,216]
bounds	<Not Set>
cursor	Default Cursor
enabled	✓

Figure 5.3 Changing the Properties of the JFrame Form Component

Now add the following components on the form:

- three editable text fields to accept the number of apples, bananas and oranges
- two buttons - one to calculate & display the total number of fruits and one to exit from the application
- one non-editable text field to display the total number of fruits
- 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 5.1. The next step is to associate code with the button with display text "Number of Fruits". Double click on the button in the design window to reach the point in the source window where the code needs to be written. Rewrite the code given in Figure 5.2 using the concept of containers as shown in Figure 5.4.



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    //To Calculate total number of fruits using variables;
    int Apples,Bananas,Oranges,Fruits;
    Apples=Integer.parseInt(jTextField1.getText());
    Bananas=Integer.parseInt(jTextField2.getText());
    Oranges=Integer.parseInt(jTextField3.getText());
    Fruits=Apples+Bananas+Oranges;
    jTextField4.setText(Integer.toString(Fruits));
}
```

Figure 5.4 Code for Calculating Total Number of Fruits Using the Concept of Variables

Know More

While converting string type of data captured from Text Fields to numbers using Integer.parseInt for Integer values and Double.parseDouble for real numbers (i.e. double type values), remember to put some default numeric value (in most of the cases, it will be 0) to avoid run-time error.

How many containers did we use to solve the above problem? We used four containers. Three of them were used to store the number of individual fruits and the fourth one was used to store the total number of fruits. These four containers need to be identified using separate names. In the beginning of the program, the containers were empty but during the execution, we changed their initial values and allocated different values to them. Such containers are called variables. From the above exercise we infer that these containers/variables have four special characteristics:

1. They have a name.
2. They are capable of storing values.
3. They provide temporary storage.
4. They are capable of changing their values during program execution.

Thus, variables are named temporary storage areas capable of storing values, which can change during the execution of the program. Remember, variables are declared within the code that uses them. In the above code the values are entered by the user in the three textFields and then these values are assigned (stored) to the variables Apples, Bananas and Oranges after converting the values to type Integer using the ParseInt() method. The statement used to assign the values is:

```
Apples=Integer.ParseInt(jTextField1.getText());
```

In this case the value entered in jTextField1 is assigned to variable Apples. Similarly, values entered in jTextField2 is stored in the variables called Bananas and value entered in jTextField3 is stored in the variables called Oranges. The sum total is stored in the variable Fruits using the statement:

```
Fruits = Apples + Bananas + Oranges;
```

Again observe the code given in Figure 5.4 closely and try to find out one extra characteristic about the variables. Note that they all have been used to store numbers without decimals. What if we change the application above to find the total marks obtained by a student? In that case the variables will store numbers with decimals. What if we change the above application to accept the first name, middle name and last name of the user and want to display the full name in the fourth text field? In that case the variables will have to store groups of characters. What do we learn from this? We learn a new characteristic of these variables - each variable can store a specific type of data. The type of data that a variable is capable of storing guides us about how to handle that particular variable and also informs us about what can be done with these variables. So each variable has a specific data type, which determines the kind of value they can store, and also specifies what can be done with these variables. Each programming language has a set of data types that can be used according to the need of the programmer. Now that we are clear about the facts why we need variables and the use of data types, let us try and understand the different data types available in java. In the above example all four variables- Apples, Bananas, Oranges and Fruits are integer type variables as they are storing numbers without decimals. In the code given in Figure 5.4 can you point out the keyword, which identifies these variables as integer numbers?

Data Types

The keyword used to identify the variables as integers is int. These are variables without decimals. Similarly we have data types for handling other varieties of data. The different types of data that we enter need different type of handling. These different types of data can be manipulated through specific data types. The data types that we have used till now can be classified as Numeric data types. Java offers us with other types of data as enumerated below:

Data type states the way the values of that type are stored, and the range for that type.

i) **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 in Figure 5.5 below :

Name	Size	Range	Values	Example
byte	1 byte(8 bits)	-128 to 127(-2^7 to $+(2^7-1)$)	(2^8) = 256	byte rollno;
short	2 bytes(16 bits)	-32768 to 32767(-2^{15} to $+(2^{15}-1)$)	(2^{16})= 65,536	short rate;
int	4 bytes(32 bits)	-2^{31} to $+(2^{31}-1)$	(2^{32}) = 42,94,967,296	int num1;
long	8 bytes (64 bits)	-2^{63} to $+(2^{63}-1)$	(2^{64}) = 1.84467441 $\times 10^{19}$	long amount;

Figure 5.5 Numeric Data Types

The decision about which numeric data type to use should be based on the range of values that a variable can take. For example, to store small values like roll number of students in a class, we should use byte whereas to store admission number of the students in a primary school we may use short as there will be more than 128 students. Similarly, to store large numbers like Roll number of students sitting for a public exam, we should use int. The value assigned to any variable must be in the correct range; otherwise we will receive an error. This means that if we assign a value lower than -128 or higher than 127 to a byte variable, the program will result in an error.

ii) **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)	($3.4\text{x}10^{-38}$) to +($3.4\text{x}10^{-38}$)	float average;
double	Double precision floating point	8 bytes (64 bits)	($1.8\text{x}10^{-38}$) to +($1.8\text{x}10^{-38}$)	double principal;

Figure 5.6 Floating Data Types

Though both float and double are used to store numbers having fractional values but for better accuracy, we normally use double instead of float.

! All numeric data types can store negative as well as positive numbers.

iii) **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).

This becomes clear from the example given in Figure 5.7:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
        char Grade='A';
        jTextField1.setText(""+Grade);
        String Message="Your Grade is Good";
        jTextField2.setText(Message);
}
```

char variable initialised using single quotes

string variable initialized using double quotes

Figure 5.7 Handling Character and String Data Types

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 justify why these variable names are unacceptable.

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

Let us quickly recap the concepts learnt above:

- To store values temporarily we need special containers called variables.
- Each variable must have a name, a data type and a value of the specific type.
- Each variable must be declared before it can be used.
- The name of the variable should be decided according to specific rules and conventions.
- The data type should be decided depending upon the type of the value a variable has to store.

Simple Applications Using the Concept of Variables

Now, let us get back to developing applications to practically understand all the concepts learnt above. First 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 5.8.

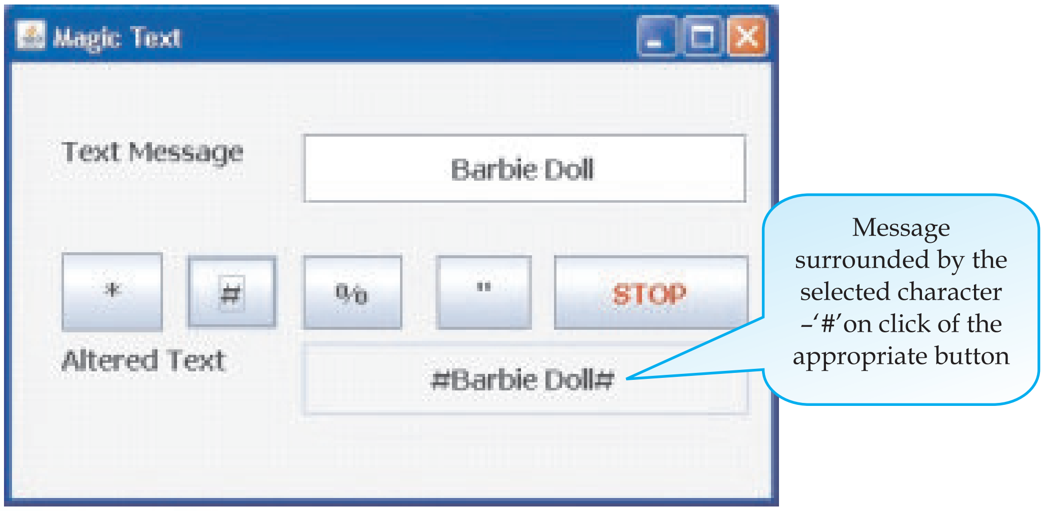

Magic Text
Text Message
Barbie Doll
*
#
%
"
STOP
Altered Text
#Barbie Doll#
Message surrounded by the selected character –'#'on click of the appropriate button

Figure 5.8 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 5.8 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 5.8. 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 as shown in Figure 5.9.

```
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);
}
```

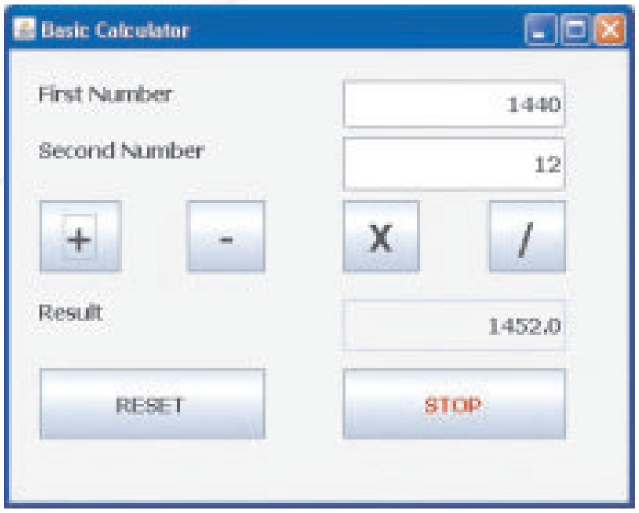
```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
     // Concatenate % to the text in jtextField1:
     char Percent;
     Percent='%';
     jTextField2.setText(Percent+jTextField1.getText()+Percent);
}
```

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
     // Concatenate " to the text in jtextField1:
     char Quotes;
     Quotes='"';
     jTextField2.setText(Quotes+jTextField1.getText()+Quotes);
}
```

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

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:

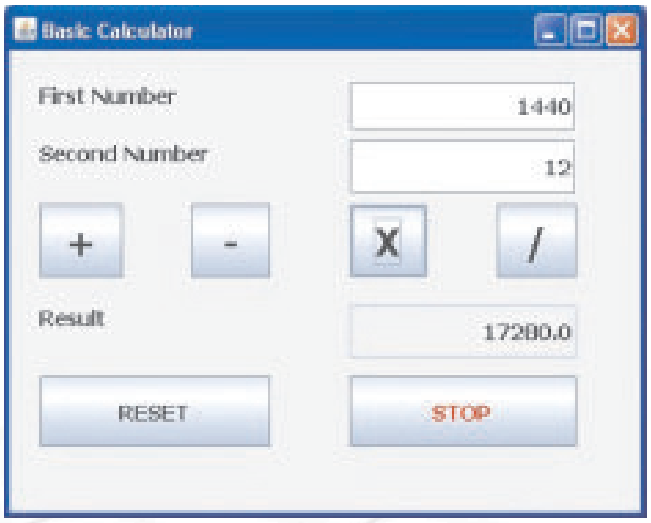
- two editable text fields to accept the two numbers .
- four buttons to decide the operation, one button to reset the fields and one button to exit out of the application.
- one non-editable text field to display the result.
- appropriate labels to direct the user.



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 5.10.

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

Figure 5.10 A Simple Calculator Showing Addition of Two Numbers



When the user enters two numbers and clicks on the X button, the product of the numbers is displayed in the jtextField3 as shown figure 5.11.

Similarly try out the effect of clicking on the other two buttons.

Now write the code for each button of the basic calculator as shown in Figure 5.12

Figure 5.11 A Simple Calculator Showing Product of Two Numbers

```
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 jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
        // Code to subtract Number2 from Number1:
        double Number1,Number2,Result;
        Number1=Double.parseDouble(jTextField1.getText());
        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));
}
```

```
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));
}
```

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

Figure 5.12 Code for Basic Calculator

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

double Number1,Number2,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));

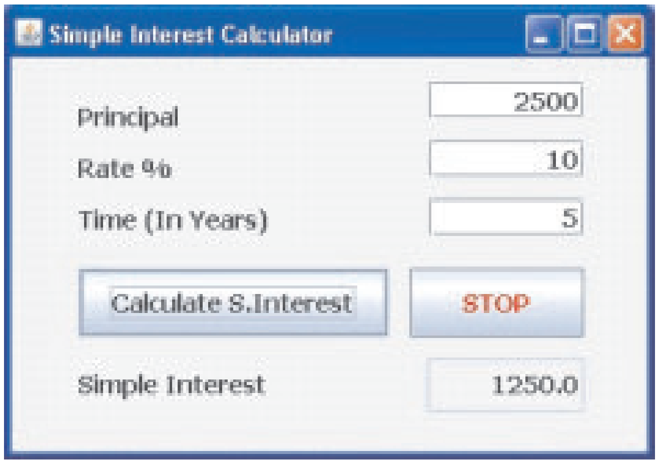
- The value stored in the variable Result is of type double so it is first converted to type string using the toString() method and then the display text of the third text field is set to the converted value using setText().

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.

! While writing commands, variable names are always given without quotes.



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 5.13 and design a similar form.

Figure 5.13 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 5.14.

```
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)
{
    System.exit(0);
}
```

Figure 5.14 Code for Simple Interest Calculator

The above example introduces us to the usage of multiple data types like byte and double in a single application and also teaches us how to handle complex calculations like multiplying Principal, Rate and Time and dividing the result by 100. Did you observe anything common in all the programs we have developed in this chapter? Think.

Operators

In all the programs we have done till now we have used various symbols such as +, - *, / and =. Each of these are used to perform specific operations like addition, subtraction, multiplication, division and assignment. Such symbols that perform specific operations on data are called operators. Operators are symbols that manipulate, combine or compare variables. Each programming language has a specific set of operators. We are going to study about two types of operators here: arithmetic operators and assignment operators.

Assignment Operator

One of the most common operator 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 (also used for String concatenation explained in Chapter 4)
-	subtraction operator
*	multiplication operator
/	division operator
%	remainder operator

The code given in Figure 5.15 displays the use of arithmetic and assignment operators.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
      int result = 4 + 8; // result is now 12
      jTextField5.setText(Integer.toString(result));
      result = result - 8; // result is now 4
      jTextField6.setText(Integer.toString(result));
      result = result * 5; // result is now 20
      jTextField7.setText(Integer.toString(result));
      result = result / 5; // result is now 4
      jTextField8.setText(Integer.toString(result));
      result = result % 3; // result is now 1
      jTextField9.setText(Integer.toString(result));
}
```

Figure 5.15 Usage of Arithmetic and Assignment operators

Now to summarize all that we have learnt in these two chapters let us design an Area Calculator Application. Look at the form design displayed in Figure 5.16

Follow the steps enumerated below to design the form:

1. Add a new JFrame Form and change its title property to Area Calculator.
2. Add three radio buttons on the form - Set the text of each of them as "CIRCLE", "SQUARE" and "RECTANGLE"
3. Add a Button Group in the form.
4. Associate all three Radio Buttons with this Button Group.
5. Add Two Labels - set their initial Text as "" i.e. Empty String.

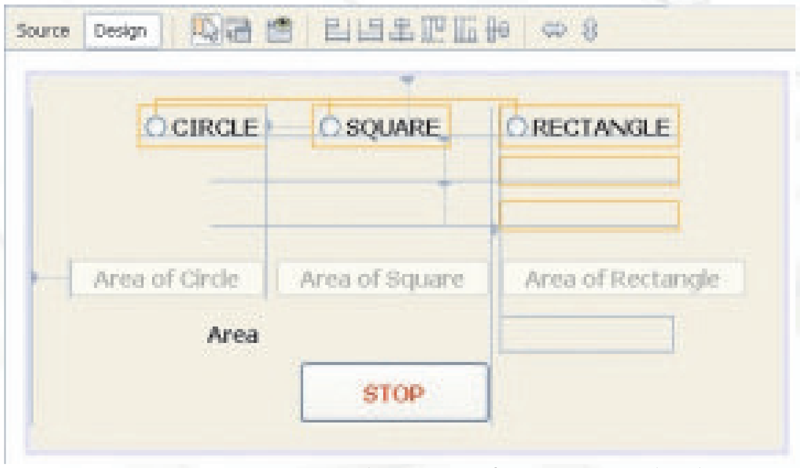
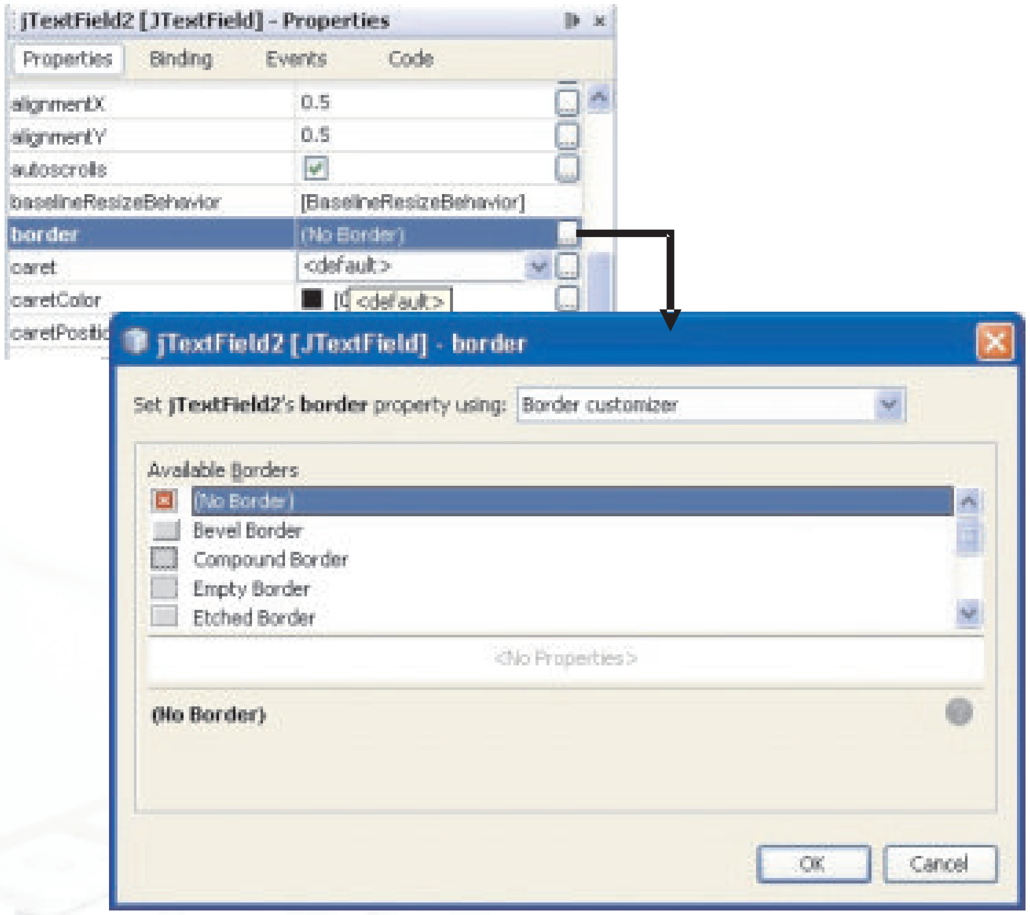


Figure 5.16 Designing the Form for Area Calculator

6. Add Two TextFields - set their initial Text as "", Set Editable property as false, Set Border property as "(No Border)" by selecting the option from the menu that pops up by clicking on the option button(..)as shown in Figure 5.17.


jTextField2 [JTextField] - Properties

Properties	Binding	Events	Code

alignmentX	0.5
alignmentY	0.5
autoscrolls	✓
baselineResizeBehavior	[BaselineResizeBehavior]
border	(No Border)
caret	<default>
caretColor	■ <default>
caretPositio	

jTextField2 [JTextField] - border

Set jTextField2's border property using: Border customizer

Available Borders
- (No Border)
- Bevel Border
- Compound Border
- Empty Border
- Etched Border

<No Properties>

(No Border)

Figure 5.17 Setting the Border Property of a Text Field

7. Add Three Buttons - Set the text of each of them as "Area of CIRCLE", "Area of SQUARE" and "Area of RECTANGLE" and Set their Enable property as false.

8. Add third Label - Set its Text as "Area".

9. Add third TextField - Set its Text as "" i.e. Empty String and Set its Editable property as false.

10. One by one click on all the three radio buttons to associate code with them:

    a. Click on the Radio Button 1 (CIRCLE) and write the CODE as mentioned in Radio Button1 Code shown in Figure 5.18.

b. Click on the Radio Button 2 (SQUARE) and write the CODE as mentioned in Radio Button2 Code shown in Figure 5.18.

c. Click on the Radio Button 3 (RECTANGLE) and write the CODE as mentioned in Radio Button3 Code shown in Figure 5.18.

```
private void jRadioButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
      //To set visible label1 and prompt the user for radius:
jLabel1.setVisible(true);
    jLabel1.setText("Radius");
    jLabel2.setVisible(false);

    jTextField1.setEditable(true);
    jTextField1.setText("");
    jTextField2.setEditable(false);
    jTextField3.setText("");

    jButton1.setEnabled(true);
    jButton2.setEnabled(false);
    jButton3.setEnabled(false);
}
```

```
private void jRadioButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
    //To set visible label1 and prompt the user for Side:
      jLabel1.setVisible(true);
    jLabel1.setText("Side");
    jLabel2.setVisible(false);

    jTextField1.setEditable(true);
    jTextField1.setText("");
    jTextField2.setEditable(false);
    jTextField3.setText("");

    jButton1.setEnabled(false);
    jButton2.setEnabled(true);
    jButton3.setEnabled(false);
}
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
        //To calculate area of Square :
    double Side,Area;
    Side=Double.parseDouble(jTextField1.getText());
    Area=Side*Side;
    jTextField3.setText(Double.toString(Area));
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{
    //To calculate area of Rectangle :
       double Length,Breadth,Area;
    Length=Double.parseDouble(jTextField1.getText());
    Breadth=Double.parseDouble(jTextField2.getText());
    Area=Length*Breadth;
    jTextField3.setText(Double.toString(Area));
}
```

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt)
{
   //To exit the application :
      System.exit(0);
}
```

Figure 5.19 Code for the Buttons of the Area Calculator

Now execute the application and try all options. The initial form and a sample run with the circle option selected is shown in Figure 5.20(a) & Figure 5.20(b).

Note how the visibility, border, enabled and editable properties can be used to make an application more presentable and appropriate.

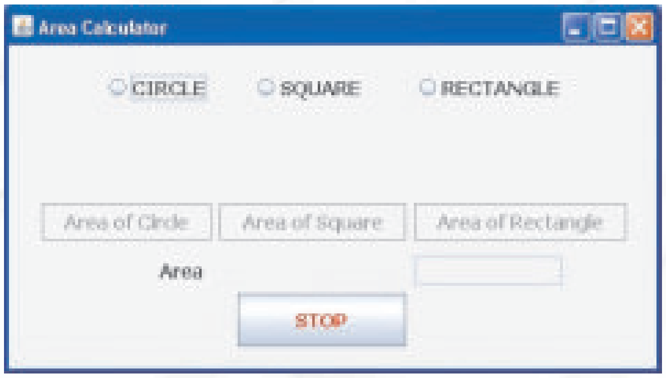


Figure 5.20(a) Area Calculator Initial Run

```
private void jRadioButton3ActionPerformed(java.awt.event.ActionEvent
evt) {
    //To set visible label1 and prompt the user for length & breadth:
      jLabel1.setVisible(true);
    jLabel1.setText("Length");
    jLabel2.setVisible(true);
    jLabel2.setText("Breadth");

    jTextField1.setEditable(true);
    jTextField1.setText("");
    jTextField2.setEditable(true);
    jTextField2.setText("");
    jTextField3.setText("");

    jButton1.setEnabled(false);
    jButton2.setEnabled(false);
    jButton3.setEnabled(true);
   }
```

Figure 5.18 Code for the Radio Buttons of the Area Calculator

11. One by one click on all the three buttons to associate code with them:

    a. Click on the Button 1 (Area of CIRCLE) and write the CODE as mentioned in Button1 Code shown in Figure 5.19

    b. Click on the Button 2 (Area of SQUARE) and write the CODE as mentioned in Button2 Code shown in Figure 5.19

    c. Click on the Button 3 (Area of RECTANGLE) and write the CODE as mentioned in Button3 Code shown in Figure 5.19

    d. Click on the Button 4 (STOP) and write the CODE as mentioned in Button4 Code shown in Figure 5.19.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
       //To calculate area of Circle :
    double Radius,Area;
    Radius=Double.parseDouble(jTextField1.getText());
    Area=3.1416*Radius*Radius;
    jTextField3.setText(Double.toString(Area));
}
```

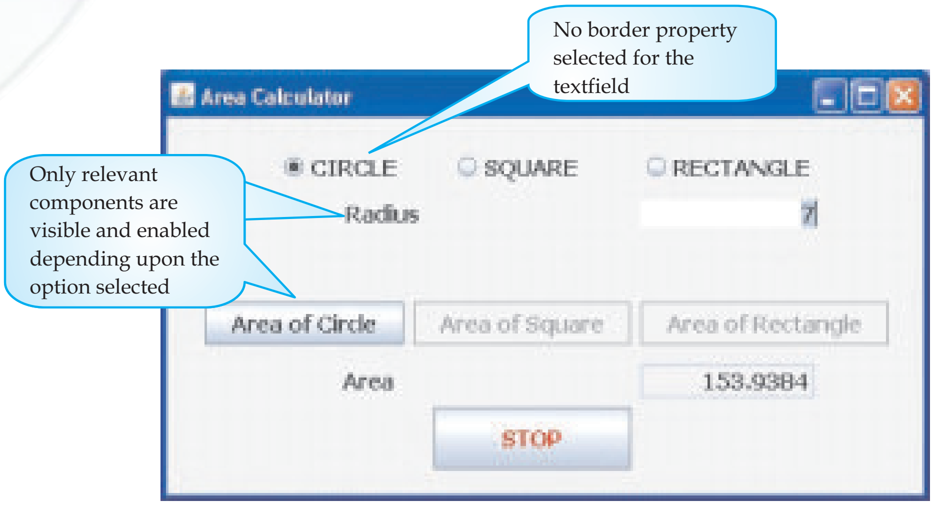

No border property selected for the textfield
Only relevant components are visible and enabled depending upon the option selected
Area Calculator
CIRCLE
SQUARE
RECTANGLE
Radius
7
Area of Circle
Area of Square
Area of Rectangle
Area
153.9384
STOP

Figure 5.20(b) Area Calculator Showing the Area of Circle

Know More

At times, when the size of a text field is not big enough to hold the result of a complicated arithmetic operation, clicking inside the text field and scrolling will help you to see the entire content of it. For example, if a number 3456789.123456235463 is the result obtained, it will be seen as `456235463`. Now, by clicking inside this text field `456235463` and scrolling inside you will be able to see the remaining portion of the content `3456789.1` in the text field. "|" inside the text field is indicating the position of cursor.

Observing the code closely will tell us that we have learnt the following new things in the development of this application:

1. The setVisible() method - to set the visibility of a component at run time. setVisible(true) implies that the component is visible and setVisible(false) implies that the component is hidden.

2. The setEditable() method - to set the editing property of a component at run time. The setEditable(true) implies that the contents of this component can be changed at run time and setEditable(false) implies that the contents of this component cannot be changed at run time.

3. The setEnabled() method - to set the enabled property of a component at run time. The setEnabled(true) implies that this component can trigger a reaction at run time and setEnabled (false) implies that this component cannot trigger a reaction at run time.

In both the previous chapters we have been executing all the part of the code associated with a component in sequence- as they appear one after the other. What happens when we do not want the code to be executed in sequence or do not want certain code to be executed at all? The following chapter will help us in understanding how to achieve this.

Summary

- Variables are named temporary storage locations.
- Variable names in java are case sensitive. That means in the same application Sum and SUM can be treated as two different variables.
- Data Types define the way the values are stored, the range of the values and the operations that can be performed on that type.
- Numeric data types can store positive as well as negative values.
- Assignment operator is used to assign a value.
- Arithmetic operators are used to perform addition, subtraction, multiplication and division.
- Some of the components for which the setVisible() method is applicable are button, textArea, textField, label, CheckBox, RadioButton, ListBox and Hidden.
- A brief summary about all the methods learnt in this lesson is given in the table below:

Method	Syntax	Usage
setVisible()	component.setVisible(boolean)	To set the visibility of a component at run time. setVisible(true) implies that the component is visible and setVisible(false) implies that the component is hidden.
setEditable()	component.setEditable(boolean)	To set the editing property of a component at run time. The setEditable(true) implies that the contents of this component can be changed at run time and setEditable(false) implies that the contents of this component cannot be changed at run time.
setEnabled()	component.setEnabled(boolean)	To set the enabled property of a component at run time. The setEnabled(true) implies that this component can trigger a reaction at run time and setEnabled (false) implies that this component cannot trigger a reaction at run time.

Multiple Choice Questions

1. The storage provided by variables is:

   a) temporary

   b) permanent

   c) volatile

   d) none of the above

2. Which of the following is a valid variable name:

   a) 3firstname

   b) Integer

   c) Char

   d) Number1

3. Which of the following statements refer to the same variable name?

   i) Amount=20 ii) amount=20 iii) Amount="20" iv) Amount=20

   a) Both i) & ii)

   b) Both i) & iv)

   c) Both ii) & iii)

   d) All of the above

4. What will be the output of the following code segment:

```
 String firstName = "Johua ";
String lastName  = "Yacomo";
String fullName  = firstName + lastName;
jTextField1.setText("Full Name: ");
jTextField2.setText (fullName);
```

   a) Full Name:

b) Full Name

Johua Yacomo

c) Johua Yacomo

fullName

d) Full Name:

JohuaYacomo

5. To print the value of a variable "x" of type int, which of the following expressions can be used:

i) jTextField1.setText("x = " + x);

ii) jTextField1.setText ("x = " + "x"));

iii) jTextField1.setText ("x = " + Integer.toString(x));

iv) jTextField1.setText (x = +"x");

a) Both i) & ii)

b) Only i)

c) Both i) and iii)

d) Only iii)

Exercises

1. What is a Variable?
2. Why are data types important?
3. Explain Numeric data types with the help of an example.
4. How are keywords different from variable names?
5. Is Java case sensitive? What is meant by case sensitive?
6. Is a string containing a single character same as a char?
7. Explain the use of different operators with the help of an example.

Lab Exercises

a) Design a GUI desktop application in java to accept the side of a square in a text field and calculate the area and perimeter of the square. Display the results in two separate text fields. Add appropriate labels and an exit button to end the application.

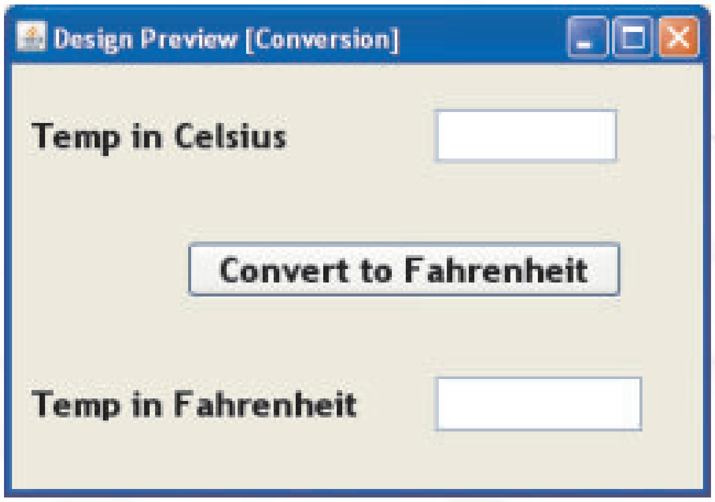
b) Design a GUI desktop application in java to accept marks in 5 subjects in five text fields and calculate the total and average marks. Display the results in separate text fields, which are disabled. Add appropriate labels and an exit button to end the application.

c) Design a GUI desktop application in java to accept sales of a company for four quarters in text fields. Calculate the total yearly sale and display the same in a dialog box. Add appropriate labels and an exit button to end the application.

d) Design a GUI desktop application in java to accept length in kilometers in a text field and display the converted length in meters in a second text field which is disabled. Add appropriate labels and an exit button to end the application.

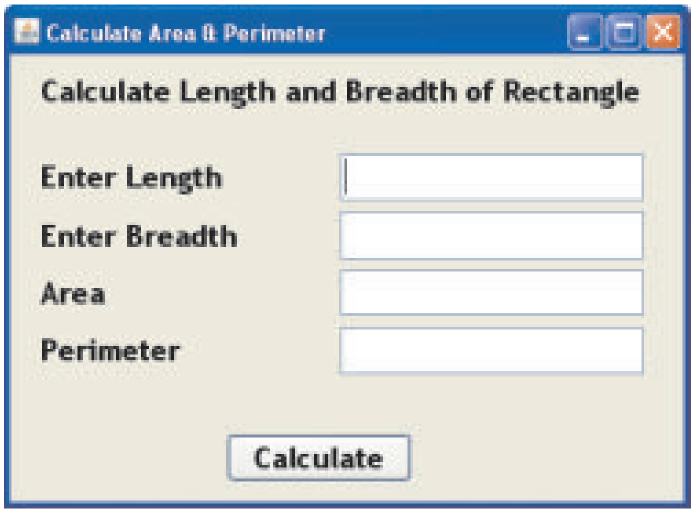
e) Design a GUI desktop application in java to accept two numbers in a in a text field and interchange the values of first number and second number using a temporary variable. Add appropriate labels and an exit button to end the application.

f) Write the code for the following application



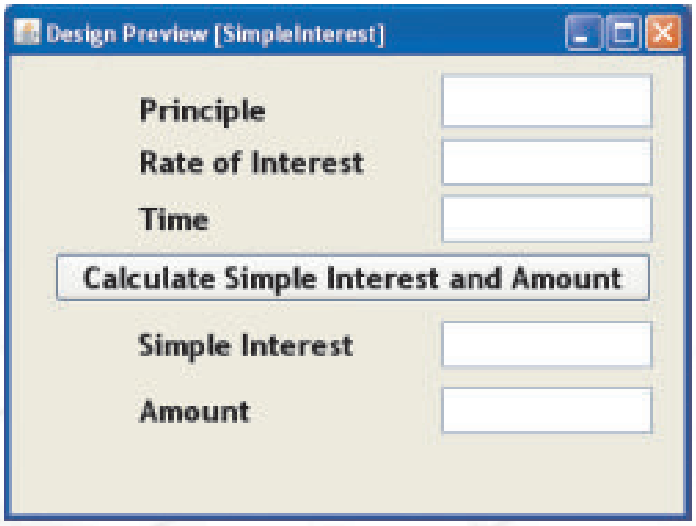
[Hint :Tc = (5/9)*(Tf-32) and Tf = (9/5)*Tc+32 where Tc = temperature in degrees Celsius, Tf = temperature in degrees Fahrenheit]

g) Write the code for the following application :


Calculate Area & Perimeter
Calculate Length and Breadth of Rectangle
Enter Length
Enter Breadth
Area
Perimeter
Calculate

[Hint : Area of Rectangle=Length*Breadth and Perimeter of Rectangle=2*(Length+Breadth)]

h) Write the code for the following application:


Design Preview [SimpleInterest]
Principle
Rate of Interest
Time
Calculate Simple Interest and Amount
Simple Interest
Amount

[Hint : SI [Interest] = (P×R×T)/100 and Amount = Principle + SI]