BASICS OF OBJECT ORIENTED PROGRAMMING

Learning Objectives

After studying this lesson the students will be able to:

- Understand the need of object oriented programming
- Define the various terms related to object oriented programming
- Identify the features of an object oriented programming language
- Use features of object oriented programming language to develop simple applications

Over the last lesson, we have reviewed the core Java language. We have learnt how to work with variables and data, perform operations on that data, make decisions based on the data and also loop repeatedly over the same section of code. Now we will move on to learn a concept that is central to Java, namely Object Oriented Programming. Object Oriented Programming is a very user friendly yet a powerful approach to solve basic to difficult problems. The idea was created for developing a language that could be used for system description (for people) and system prescription (as a computer program through a compiler). There are several object-oriented languages. The three most common ones are, Smalltalk, C++ and Java.

Puzzle [4]

A student and a lady are travelling in a train. They get around talking and the lady decides to give a puzzle to the student. She tells him that she has 3 children whose product of ages is equal to the maximum number of runs possible to score in an over without any illegitimate ball being bowled (i.e. NB, Wide, etc). Also, the sum of their ages is equal to her berth number. However, the student isn't able to answer. The lady then gives him a further hint that the eldest of her children has only one eye. At this information, the

student knows the ages of the three children. Without knowing the lady's berth number, can you guess the ages of her children?

Introduction

James Alexander is a resident of a developed nation and works as a freelance consultant. He is hired by one of the corporate houses of a developing nation to plan a strategy to improve production in one of their factories which is located in a remote village named Khabri. The consultant decides to submit a quick action plan and so starts searching for information about the remote village. He has never visited any of the remote locations and so tries to simply imagine the problems faced by remote people. Mohan Swamy is a resident of one of the developing countries and he also is a freelance consultant. He completed his studies from a top notch university and to actually put his theoretical knowledge to practice, he started staying in the remote village Khabri. He wanted to actually experience the hardships faced by people residing in remote areas. To sustain himself he decides to pick up a job in the only factory situated in Khabri. The HR manager impressed with his in-depth knowledge and qualifications requests him to also plan a strategy to improve production of their factory. Who do you think will be able to provide a more viable solution? The obvious answer for most of us would be that the person sitting in the remote village and literate enough to solve the problem will be able to provide a better strategy because he closely understands the real problems of the residents as compared to a person sitting far away. But what does this teach us about programming? This teaches us that in programming also functions/methods/programs written for specific situations are able to manipulate data of their respective entities more efficiently. Now, let us understand a little about the various programming paradigms.

Introduction to Programming

Computer programming is a process of designing, writing, testing, debugging and maintaining the source code of computer programs written in a particular programming language. The purpose of programming is to organize instructions that are capable of solving real life problems. The process of writing source code of programs requires expertise in subject, knowledge of desired application domain, a formal logic and knowledge of syntax of the relevant programming language. Writing instructions in the desired order gives the required results from the program but when these instructions

increase in number, it becomes extremely difficult for the programmer to debug, maintain or troubleshoot codes. For this reason, technology experts kept developing and introducing different programming paradigms and accordingly kept developing languages to support these paradigms. Procedural programming paradigm was one of the major stepping stone for these experts which focused on breaking down a programming task into a collection of small modules known as sub routines or procedures. This paradigm helped the programmers to debug, maintain or troubleshoot codes in a more effective manner. The experts did not stop their research in improving this paradigm and introduced a new paradigm known as object oriented programming paradigm where a programming task was broken into objects. Here each object was capable of encapsulating its own data and methods (subroutines / procedures). The most important distinction is that where procedural programming uses procedures to operate on data, object oriented programming bundles data and methods together and operates as a independent entity of the program. Some languages support one particular programming paradigm while some are developed to support multiple programming paradigms. C++, C sharp, Object Pascal etc. are the languages which support procedural as well as object oriented paradigm. Java supports only object oriented programming. Basic, COBOL support only procedural programming.

Object Oriented Programming

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

The major components of Object Oriented Programming are as follows:

1. Class
2. Object

3. Data Members & Methods

4. Access Specifier and Visibility Modes

Classes and Objects :

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

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

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

Data Members and Methods :

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

In real java programming, this data will be required to conform to a specific data type as in char, int, float or double whereas the methods will be a sequence of steps written together to perform a specific task on the data. Carefully observe the illustration given in Figure 4.1 to reinstate the theoretical concepts learnt above.

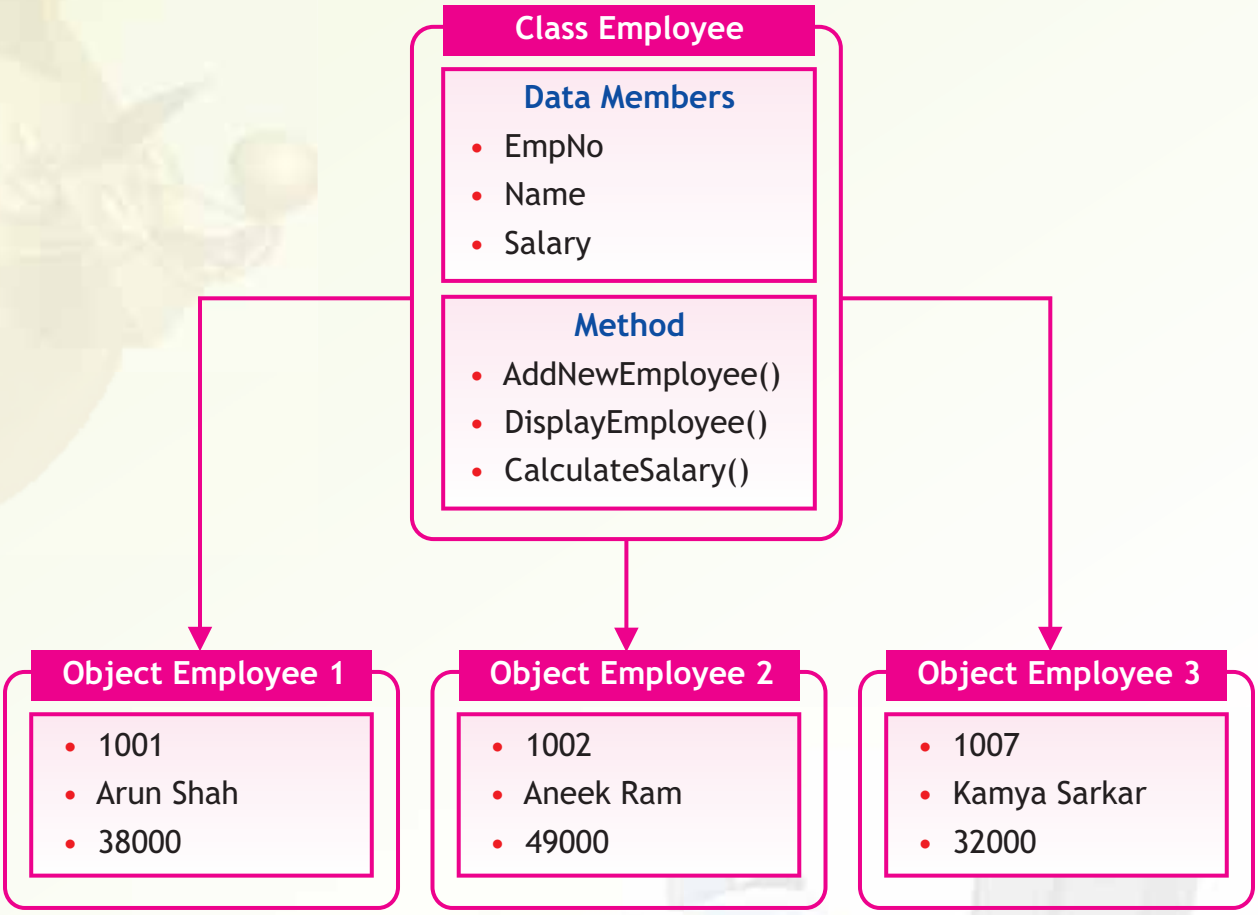


Figure 4.1 Illustration Showing the Class, Object, Members and Methods

Referring to the situation presented in the introduction of the chapter, just like the person residing in the village can efficiently solve problems pertaining to his village, similarly the methods of specific classes are able to manipulate data of their respective classes efficiently resulting in better security of data in an Object Oriented Programming paradigm.

Now that you are clear about the concept of a class and an object, you will be able to appreciate and identify classes and methods that we have already been using throughout our class XI. Do you know JTextField, JLabel, JTextArea, JButton, JCheckBox and JRadioButton are all classes and the jTextField1, jLabel1, jTextArea1, jButton1, jCheckBox1 and jRadioButton1 components are all objects. The setText(), setEnabled(), pow(), substring() are all methods of different classes. This concept is illustrated in Figure 4.2.

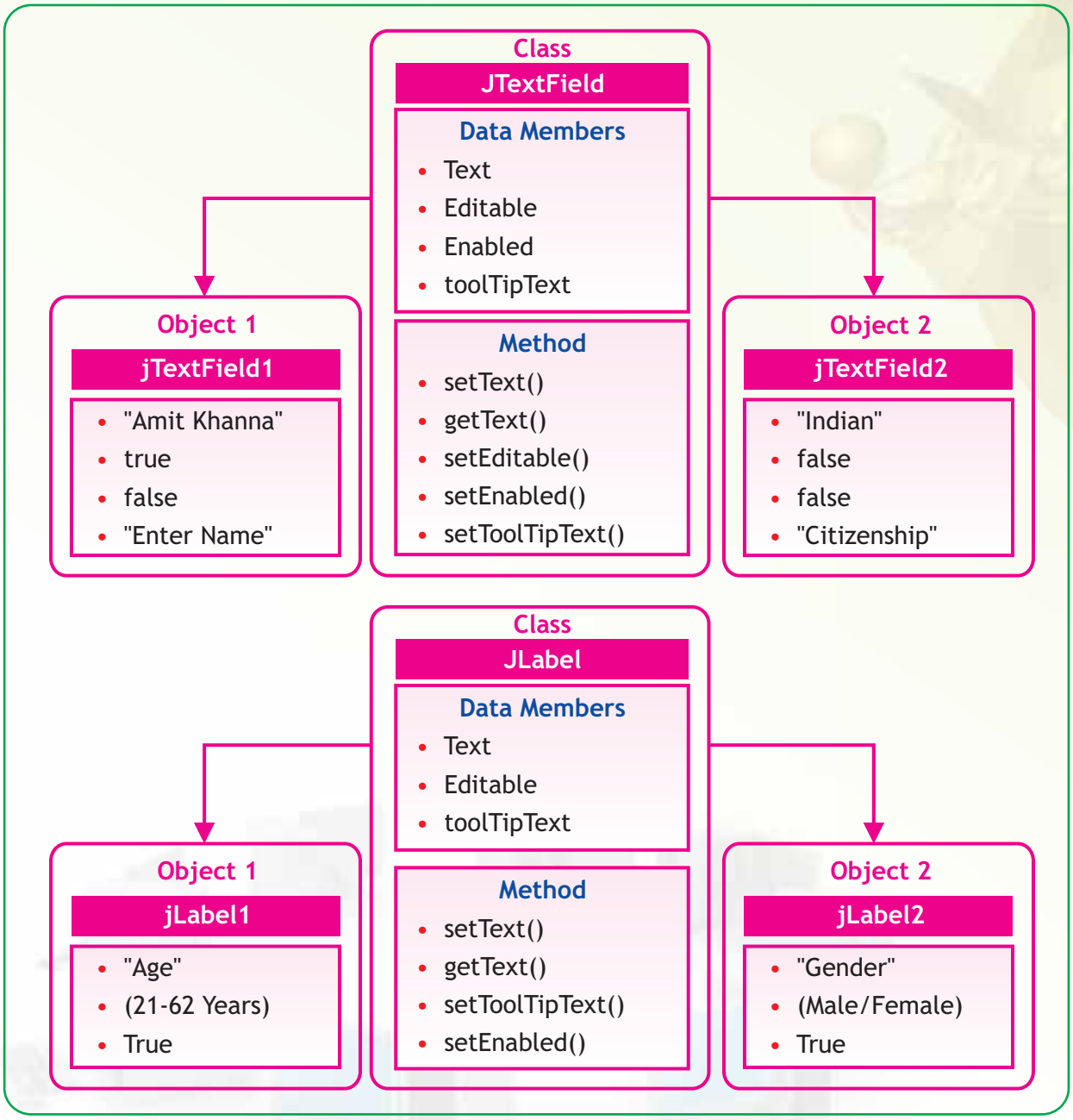


Figure 4.2 JTextField and JLabel Classes

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

The following two tables summarize the data members (properties) and methods of all classes learnt during the course till now:

Class	Objects	Data Members (Properties)	Methods
JTextField	*jTextField1*	• *Text* • *Editable* • *Enabled* • *ToolTipText*	• *setText()* • *getText()* • *setEditable()* • *setEnabled()* • *setToolTipText()*
JLabel	*jLabel1*	• *Text* • *Enabled* • *ToolTipText*	• *setText()* • *getText()* • *setEnabled()* • *setToolTipText()*
JTextArea	*jTextArea1*	• *Columns* • *Editable* • *Font* • *lineWrap* • *Rows* • *wrapStyleWord* • *toolTipText*	• *isEditable()* • *isEnabled()* • *getText()* • *setText()*
JButton	*jButton1*	• *Background* • *Enabled* • *Font* • *Foreground* • *Text* • *Label*	• *getText()* • *setText()*

JCheckBox	*jCheckBox1*	• *Button Group* • *Font* • *Foreground* • *Label* • *Selected* • *Text*	• *getText()* • *setText()* • *isSelected()* • *setSelected()*
JRadioButton	*jRadioButton1*	• *Background* • *Button Group* • *Enabled* • *Font* • *Foreground* • *Label* • *Selected* • *Text*	• *getText()* • *setText()* • *isSelected()* • *setSelected()*
JPasswordField	*jPasswordField1*	• *Editable* • *Font* • *Foreground* • *Text* • *Columns* • *toolTipText*	• *setEnabled()* • *setText()* • *getText()* • *isEnabled()*
JComboBox	*jComboBox1*	• *Background* • *ButtonGroup* • *Editable* • *Enabled*	• *getSelectedItem()* • *getSelectedIndex()* • *setModel()*

		• *Font* • *Foreground* • *Model* • *SelectedIndex* • *SelectedItem* • *Text*	
JList	*jList1*	• *Background* • *Enabled* • *Font* • *Foreground* • *Model* • *SelectedIndex* • *SelectedItem* • *SelectionMode* • *Text*	• *getSelectedValue()*
JTable	*jTable1*	• *model*	• *addRow()* • *getModel()*

Polymorphism :

It is the ability of a method to execute in many forms. In object oriented programming there is a provision by which an operator or a method exhibits different characteristics depending upon different sets of input provided to it. This feature in Object Oriented Programming is known as polymorphism. Two examples of polymorphism are method overloading and operator overloading. Method overloading is where a method name can be associated with different set of arguments/parameters and method bodies in the same class. The round() method of the Math class and the substring() method of the String class are good examples of method overloading. The following examples explain how round() and substring() methods are overloaded.

If the argument passed to the round() method is of type double then it rounds off the double value and returns the closest long value. On the other hand, if the argument passed to the round() method is of type float then it rounds off the float value and returns the closest integer value. This simply means that the round() method is overloaded on the basis of the type of arguments supplied to it. Figure 4.3 illustrates the concept of method overloading as demonstrated by the round()method. For example:

```
float f = 12.5;

double d = 123.6543;

int num1 = Math.round(f);          //num1 will store 13

float num2 = Math.round(d);        //num2 will store 124.0
```

Notice that if the argument is of type double and we try to round it to an integer, then it results in an error because no such method is pre-defined which takes a double argument and rounds it to integer type.

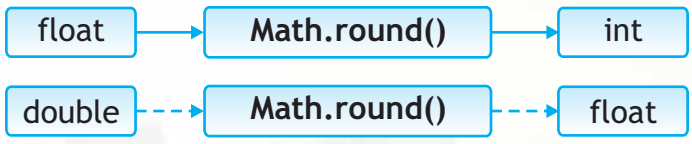


Figure 4.3 Polymorphism demonstrated by round() method

If a single argument is passed to the substring() method then it returns all characters from the start position to the end of the string whereas if two arguments are passed to the substring() method then it returns a substring of the characters between the two specified positions of the string. This simply means that the substring() method is overloaded on the basis of the number of arguments supplied to it. For example:

```
String Message = "Good Morning", Message1, Message2;

Message1 = Message.substring(5);//Message1 will store Morning

Message2 = Message.substring(0,4); //Message2 will store Good
```

Operator Overloading is another example of polymorphism. Just as methods can be overloaded, operators can also be overloaded. A very good example of operator overloading is the + operator which returns the sum of two numbers if the operands are numbers but returns the concatenated string if the operands are characters or strings. For example:

```
String A = "Hello", B = "There";
String C = A + B;              //C will store HelloThere
int num1 = 10, num2 = 20;
int num3 = num1 + num2;        //num3 will store 30
```

Abstract Class :

The classes for which objects are required to be declared are known as concrete classes like JLabel, JTextField, JComboBox etc. The classes for which it is not essential to declare objects to use them are known as abstract classes like JOptionPane. Abstract classes are normally used as base class in inheritance for which no direct object is required to be created. Also abstract classes are used for defining generic methods where there is no requirement of storing results.

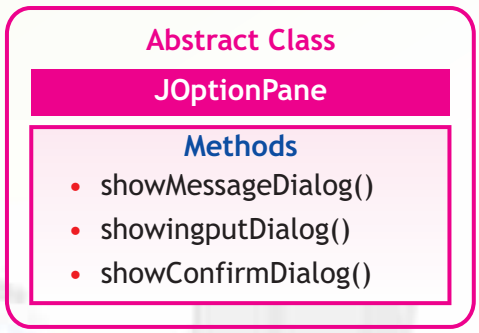


Figure 4.4
Example of an Abstract Class

Notice that the JOptionPane Class which is an abstract class has no data members and so it will not be able to store any values. The reason behind this is that because we cannot create objects of an abstract class, so we will not be able to provide any data to this class. Therefore, there is no point of having a data member.

Inheritance :

Inheritance is the most powerful feature of Object Oriented Programming, after classes themselves. Inheritance is a process of creating new class (derived class or sub class or child class) from existing class (base class or super class or parent class). The derived classes not only inherit capabilities of the base class but also can add new features of

their own. The process of Inheritance does not affect the base class. Observe the base and the derived class shown in Figure 4.5 carefully. Notice that the derived class inherits the data members namely Colour and Height from the base class and has a new data member defined in itself namely the Type. Similarly the derived class inherits the methods namely getColour() and getHeight() from the base class and has a new method defined in itself namely the setType().

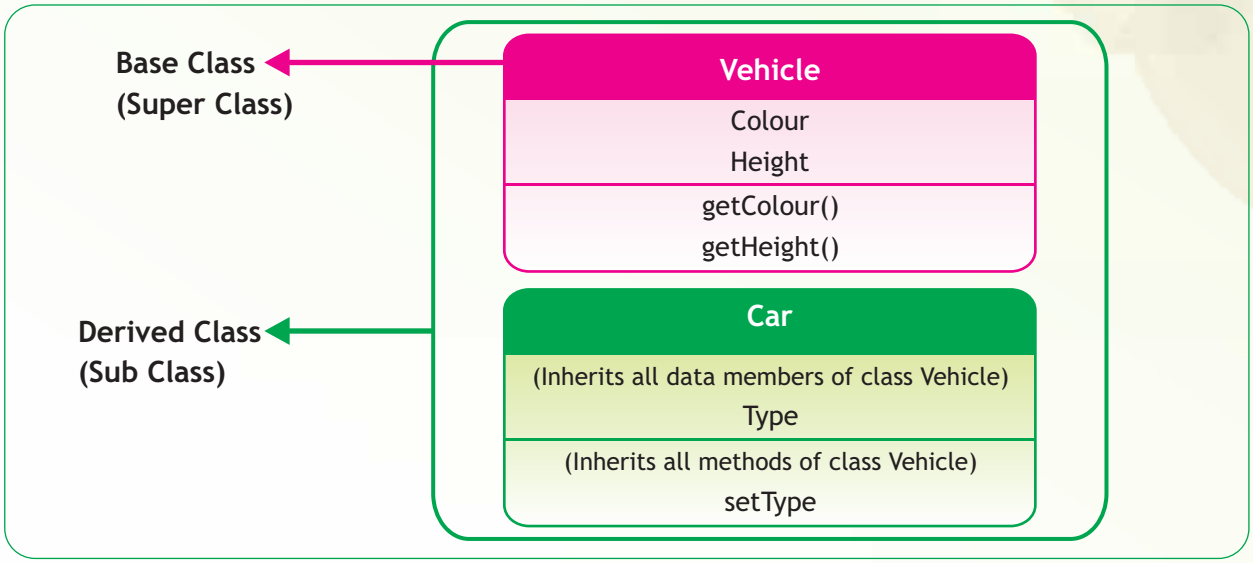


Figure 4.5 Concept of Base Class and Derived Class

The most important aspect of inheritance is that it allows reusability of code, and also a debugged class can be adapted to work in different situations. Reusability of code saves money as well as time and increases program reliability. Inheritance is very useful in original conceptualization and design of a programming problem. The idea of inheritance helps you to include features of already existing class (debugged) in a new class. Properties (data members) and methods included in super class can be invoked/accessed from the subclass. A subclass inherits all the members (data members and methods) from its superclass.

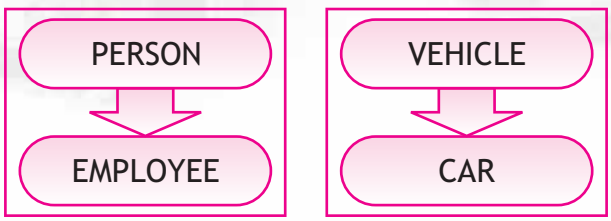


Figure 4.6 Common Examples of Inheritance

The new class can, in turn, can serve as the basis for another class definition. It is important to know that all Java objects use inheritance and every Java object can trace back up the inheritance tree to the generic class Object.

If you look at the pre-fabricated code (code automatically added by the compiler) in any of your application (JFrame Form) created in NetBeans, you will observe that javax.swing.JFrame acts as a base class (super class) and the name given by you for the JFrame Form acts as the name for the derived class (sub class). Each new JFrame Form added to the application inherits all the features of the javax.swing.JFrame class. The screen shot from one such application is shown below to illustrate the same. Here, javax.swing.JFrame is the base class and Example is the derived class.

```
public class Example extends javax.swing.JFrame
{
        /** Creates new form named Example */
        public Example()
    {
        initComponents();
    }
}
```

Figure 4.7 Sample Code Illustrating the Concept of Inheritance in Java

Notice that the extends keyword is used to inherit data members and methods of the JFrame base class for the Example class which is the derived class.

Summary

- Procedural programming paradigm focuses on breaking down a programming task into a collection of small modules known as sub routines or procedures.
- The most important distinction between Procedural and Object Oriented Programming is that where Procedural Programming paradigm uses procedures to operate on data, Object Oriented Programming paradigm bundles data and methods together and operates as an independent entity of the program.
- Class, object, data members and methods are the major components of an Object Oriented Programming paradigm.
- A class is used to encapsulate data and methods together in a single unit and an object is an instance of a class.
- Polymorphism is the ability of a method to execute in many forms.
- Method overloading and operator overloading are two examples of Polymorphism.
- The round() method of the Math class and the substring() method of the String class are good examples of method overloading.
- The + operator is a good example of operator overloading in Java.
- The classes for which it is not essential to declare objects to use them are known as abstract classes. JOptionPane is one of the examples of an abstract class.
- The concept of Inheritance allows reusability of code by including features of already existing class (called the base class) in a new class (called the derived class).
- The extends keyword is used to inherit data members and methods of the base class and allows the derived class to use these methods.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. Which of the following is not a feature of Object Oriented Programming?

   a. Inheritance b. Data Overloading

   c. Polymorphism d. Objects

2. Identify which of the following are not valid examples of method overloading?

   a. round() b. subString()

   c. subStr() d. getText()

3. Which of the following statements about the Java language is true?

   a. Both procedural and OOP are supported in Java

   b. Java supports only procedural approach

   c. Java supports only OOP approach

   d. None of the above

4. Which of the following is an Abstract class?

   a. JTextArea b. String

   c. Math d. JFrame

5. Which of the following statements is false about objects?

   a. Object is an instance of a class

   b. Object is capable of storing data

   c. Each object has its own copy of methods

   d. None of the above

6. A class can have many methods with the same name as long as the number of parameters or type of parameters is different. This OOP concept is known as

   a. Method Overriding b. Method Overloading

   c. Method Invocating d. Method Labelling

7. Which of the following statement is true?

   a. A super class is a subset of a sub class

   b. class Second extends First means First is a sub class

   c. class Second extends First means Second is a super class

   d. None of the above

8. Which feature(s) of Object Oriented Programming is illustrated in the following code snippet:

```
String A = "Hello", B = "There", C;
String C = A + B;
int num1 = 10, num2 = 20;
int num3 = num1 + num2;
C = B.substring(1);
B = A.substring(3);
```

   a. Method Overloading b. Inheritance

   c. Operator Overloading d. None of the above

ANSWER THE FOLLOWING QUESTIONS

1. Define the term Polymorphism. What are the two ways polymorphism is demonstrated in Java?
2. What is the importance of abstract classes in programming?
3. Write a short note on Operator overloading and Method Overloading citing examples for both.
4. Carefully study the code given below. It is giving an error whenever it is compiled:

```
float f = 12.5;
double d = 123.6543;
```

```
int num1 = Math.round(f);          //Statement 1
float num2 = Math.round(d);        //Statement 2
int num2 = Math.round(d);          // Statement 3
```

Identify the statement that will result in an error. Justify.

5. Carefully study the code given below:

```
String Message = "Hello! How are you?", Msg1, Msg2;
Msg1 = Message.substring(7);
Msg2 = Message.substring(0,5);
```

What will be the contents of the variables Msg1 and Msg2 after the above statements are executed?

6. Study the following code and answer the questions that follow:

```
String SMS=jTextArea1.getText();
int L=SMS.length(),Balance;
Balance=160-L;
jTextField2.setText(Integer.toString(L));
jTextField3.setText(Integer.toString(Balance));
```

a. Name any one native class of Java used in the above code.

b. Name the object created of the above mentioned native class.

c. Identify and name two methods of the native class.

d. Name the method used to convert one type of data to another and also mention the data type before and after conversion.

7. Study the following code and answer the questions that follow:

```
public class Example extends javax.swing.JFrame
{
    /** Creates new form named Example */
```

```
        public Example()
        {
            initComponents();
        }
}
```

a. Which feature of object oriented programming is depicted above?

b. Name the base class and the derived class.

c. Name the keyword used for passing on characteristics of the base class to derived class.

8. Compare and contrast the Procedural Programming paradigm and the Object Oriented Programming paradigm by writing a simple program of a mathematical calculator using both approaches. (Note: The teacher may illustrate the Procedural Programming Paradigm using any other simple programming language but should not test the students on it).

LAB EXERCISES†

1. Create a GUI application to accept a string and display it in reverse order using the substring() method.

2. Create a GUI application to create random whole numbers between 2 float numbers input by the user.

3. Create a GUI application to accept 3 numbers in separate text fields and display their sum, average, maximum or minimum after rounding the results on the click of appropriate buttons (There are four separate buttons - one for sum, one for average, one for maximum and one for minimum). The result should be displayed in the fourth text field.

4. Create a GUI application to accept the date (as 1), month (as a number like 3 for March) and year (as 2010) in separate text fields and display the date in the format: dd/mm/yy. Take care of the following points while creating the application:

† The students should be encouraged to design appropriate forms for the applications themselves.

- Verify the date input before displaying it in the suggested format and display error messages wherever applicable
- The date is accepted as 1 (for the first of any month) but should be displayed as 01 in the final format.
- The year is accepted as 2010 but displayed as 10 in the final format.

TEAM BASED TIME BOUND EXERCISES

(Team size recommended: 3 students each team)

1. Divide the class into 6 groups. Assign one class (out of JTextField, JTextArea, JLabel, JCheckbox, JRadioButton and JComboBox) to each of the groups and instruct each group to create applications demonstrating the usage of all methods learnt of that particular class. The groups may also enhance the forms using different properties of the assigned classes.
2. Divide the class into three groups and tell each group to create presentations on one of the following topics (The topics may be allocated using a draw system):
   a) Programming Paradigms
   b) The Philosophy of Object Oriented Programming
   c) Future Trends in Programming

ADVANCED PROGRAMMING CONCEPTS

Learning Objectives

After studying this lesson the students will be able to:

- Define objects and their usage
- Appreciate the usage of native classes Math and String
- Understand the need and use of methods pow() and round() of Math class
- Understand the need and use of methods toUpperCase(), toLowerCase(),substring(), concat(), length() and trim() of String class
- Develop small applications involving the methods learnt of Math and String classes.

In the last lesson, we introduced the concept of Object Oriented Programming and learnt about the different elements of an Object Oriented Programming Language. Now we will move on to learn about two important classes we commonly use in Java - namely Math and String. The lesson focuses on how to use some of the popular methods of these classes appropriately and appreciate how they simplify many programming tasks for us.

Puzzle [5]

Find a 9-digit number, which you will gradually round off starting with the number placed at units, then tens, hundreds etc., until you get to the last numeral, which you do not round off. The rounding alternates (down, up, down ...) which means that the first number from the right is rounded down while the second number from the right is rounded up and so on. After rounding off 8 times, the final number is 500000000. The original number is commensurable by 6 and 7, all the numbers from 1 to 9 are used, and after rounding four times the sum of the not rounded numerals equals 24.

Classes and Objects

As studied in the previous lesson, a class is used to encapsulate data and methods together in a single unit. An object is an instance of a class that is capable of holding actual data in memory locations. Class and objects are related to each other in the same way as data type and variables. If we take a hypothetical case in which human is a class, Mr. Shah and Mr. Kumar will be the objects of this Human class.

Math Class

The Math class contains built-in methods for performing basic numeric operations such as the elementary exponential, rounding of a number, square root, and trigonometric functions. These functions can be used directly by the user in the program. These methods are highly reliable and can tremendously reduce the amount of coding required for an application.

Some of the most commonly used Math class methods are as follows:

Method	Description
pow(double a, double b)	Returns the value of the first argument raised to the second argument.
round(double a)	Returns the closest long to the double argument.
round(float a)	Returns the closest int to the float argument.

It is not necessary to import any package for the Math class because this is already in the java.lang package. Therefore in-built methods of the Math class can be used directly in the application just like the other methods, as we have learnt in the previous class. Let us learn the usage of these methods by building some simple applications. First let us create an application that calculates the power of a number. Observe the following form carefully.

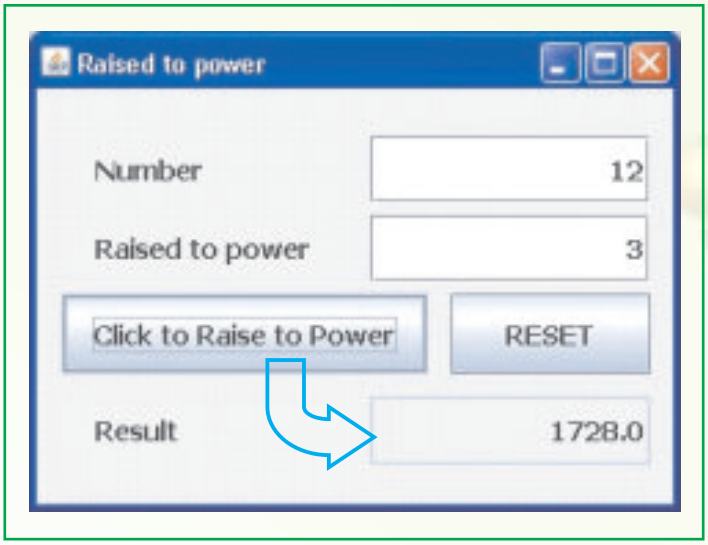


Figure 5.1 Sample Run of the Raised to Power Application

Let us first design the form as shown in Figure 5.1. First add a new JFrame form and set its title property to "Raised to Power". Now, add the following components on the form:

- Two editable text fields to accept the number and the power
- One non-editable text field to display the calculated result
- Two buttons - one to calculate & display the result and one to reset the form components
- Three appropriate labels - one against each of the text field to direct the user.

Change the properties of the components as learnt earlier so that the form looks exactly like the one displayed in Figure 5.1. The next step is to associate code with both the buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 5.2.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Calculate the value of NumberRaise using the pow()
    function
```

```
        double Number,Raise,Result;
        Number=Double.parseDouble(jTextField1.getText());
        Raise=Double.parseDouble(jTextField2.getText());
        Result=Math.pow(Number, Raise);
        jTextField3.setText(Double.toString(Result));
    }
```

```
    private void
    jButton2ActionPerformed(java.awt.event.ActionEvent evt)
    {
        // Clear all text fields by initializing them with blank
        spaces
        jTextField1.setText("");
        jTextField2.setText("");
        jTextField3.setText("");
    }
```

Figure 5.2 Code for the Raised to Power Application

Let us now understand the code in detail line by line:

`double Number,Raise,Result;`

- Declare three variables named Number, Raise and Result of type double.

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

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

- Retrieve the values input by the user from the text fields using the method getText() and store these values in the variables Number and Raise respectively.

```
Result=Math.pow(Number, Raise);
```

- Calculate the number (value stored in variable Number) raised to the value stored in the variable Raise using the pow() method and store the final value in the variable Result.

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

- Display the final result in the third text field using the setText() method after converting it to string type using the toString() method.

Next, 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.

Next let us learn the usage of another method of the Math class namely round(). Observe the following form carefully.

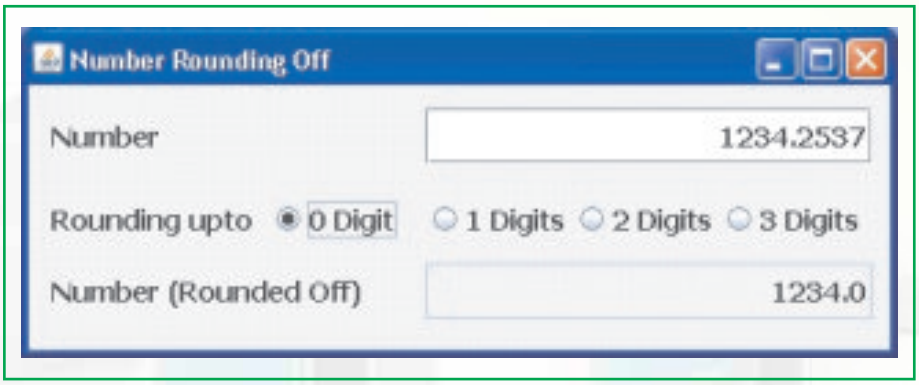


Figure 5.3 Design of the Number Rounding Off Application

Let us first design the form as shown in Figure 5.3. First add a new JFrame form and set its title property to "Number Rounding Off". Now, add the following components on the form:

- One editable text field to accept the number to be rounded.
- One non-editable text field to display the rounded off number

- Four radio buttons -to give a choice to the user for rounding off the number upto 0, 1, 2 or 3 digits
- Three labels - one against each of the text field and one against the radio button group to appropriately direct the user.

Change the properties of the components so that the form looks exactly like the one displayed in Figure 5.3. The next step is to associate code with the radio buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 5.5. A sample run of the application is shown in Figure 5.4.

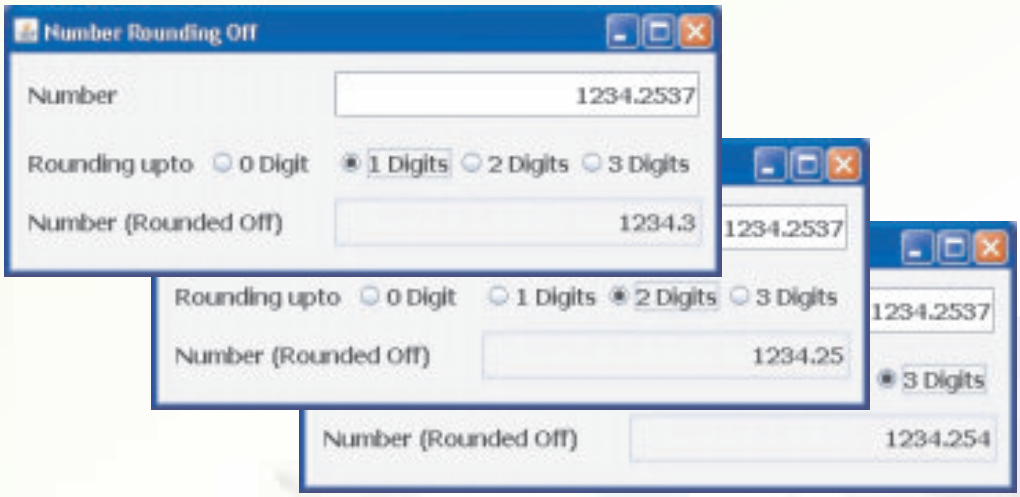


Figure 5.4 Sample Run of the Number Rounding Application

```
private void
jRadioButton1ActionPerformed(java.awt.event.ActionEvent evt)
{ //to round off the input number
    double Number,RoundedNumber;
    Number=Double.parseDouble(jTextField1.getText());
    RoundedNumber=Math.round(Number);
    jTextField2.setText(Double.toString(RoundedNumber));
}
```

```
private void
jRadioButton2ActionPerformed(java.awt.event.ActionEvent evt)
{ //to round off the number to 1 digit
    double Number,RoundedNumber;
    Number=Double.parseDouble(jTextField1.getText());
    //You need to divide by a real number and so 10.0 and
    not 10
    RoundedNumber=Math.round(Number*10)/10.0;
    jTextField2.setText(Double.toString(RoundedNumber));
}
```

```
private void
jRadioButton3ActionPerformed(java.awt.event.ActionEvent evt)
{ //to round off the number to 2 digits
    double Number,RoundedNumber;
    Number=Double.parseDouble(jTextField1.getText());
    RoundedNumber=Math.round(Number*100)/100.0;
    jTextField2.setText(Double.toString(RoundedNumber));
}
```

```
private void
jRadioButton4ActionPerformed(java.awt.event.ActionEvent evt)
{ //to round off the number to 3 digits
    double Number,RoundedNumber;
    Number=Double.parseDouble(jTextField1.getText());
    RoundedNumber=Math.round(Number*1000)/1000.0;
    jTextField2.setText(Double.toString(RoundedNumber));
}
```

Figure 5.5 Code for the Number Rounding Application

Let us now understand the code in detail line by line:

```
double Number,RoundedNumber;
```

- Declare two variables named Number and RoundedNumber of type double.

```
Number=Double.parseDouble(jTextField1.getText());
```

- Retrieve the value input by the user in the first text field using the getText() method and store it in the variable Number after converting it to type double (using the parseDouble() method)

```
RoundedNumber=Math.round(Number*10)/10.0;
```

- Calculate the rounded number using the round method and store it in the variable RoundedNumber. Since the round method does not allow the user to specify the precision digits, so we first multiply the number by 10 (for rounding off to 1 digit and similarly multiply by 100 for rounding off to 2 digits and so on) and then divide the result by 10 to get the closest double number.

```
jTextField2.setText(Double.toString(RoundedNumber));
```

- Display the calculated result in the second text field using the setText() method after converting it to type String using the toString() method.

The coding for the other radio buttons is similar.

Next let us learn a few methods of another important class, namely String class, of Java.

String Class :

The String class includes methods for examining individual characters of a string sequence, for converting strings to uppercase or lowercase, for extracting substrings, for joining two strings together, for calculating the length of a string and also for creating a new string by omitting the leading and trailing whitespaces.

When using most of the String class's methods, it should be kept in mind that a string is just a series of individual characters and that each character has a position or index, a little like a queue. Remember that in strings the first position, or index, is labelled 0 and not 1. So, if we create a string "HAPPY" then the characters will be stored as shown below:

Character Index	0	1	2	3	4
Character stored	H	A	P	P	Y

Some of the String class methods are as follows:

Method	Description
concat(String str)	Concatenates the specified string to the end of this string.
length()	Returns the length of the string.
substring (int beginpos [, int endpos])	Returns a substring of the characters between the two specified positions of the string. The second parameter is optional; if not included then all characters from the start position to the end of the string are included. The character at the ending position (n2) is not included.
toLowerCase()	Returns the string converted to lower case.
toString()	Returns the object as a string.
toUpperCase()	Returns the string converted to upper case.
trim()	Returns the string, after removing the leading and the trailing whitespaces

The most important methods are case conversion methods toUpperCase() and toLowerCase(), so let us first develop a "Case Changer" application. Observe the following form carefully:

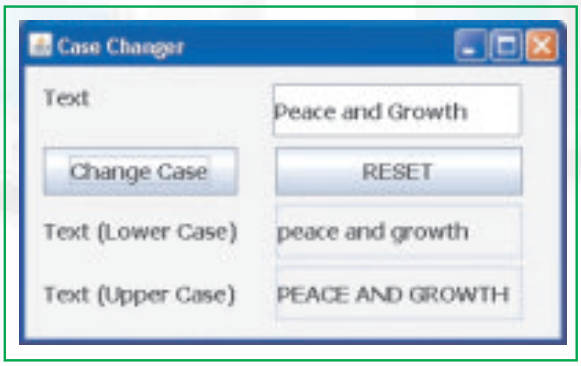


Figure 5.6 Sample Execution of the Case Changer Application

Let us first design the form as shown in Figure 5.6. First add a new JFrame form and set its title property to "Case Changer". Now, add the following components on the form:

- One editable text field to accept the string to be converted to uppercase and lowercase
- Two non-editable text fields - one to display the string converted to uppercase and the other to display the string converted to lowercase
- Two buttons - one to convert & display the converted strings and one to reset the form components
- Three appropriate labels - one against each of the text field to direct the user.

Change the properties of the components so that the form looks exactly like the one displayed in Figure 5.6. The next step is to associate code with both the buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 5.7.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
//Convert an input string to lower case and upper case
    String Str=jTextField1.getText();
    jTextField2.setText(Str.toLowerCase());
    jTextField3.setText(Str.toUpperCase());
}
```

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{ // Clear all text fields by initializing them with blank
spaces
    jTextField1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
}
```

Figure 5.7 Code for the Case Changer Application

Let us now understand the code in detail line by line:

```
String Str=jTextField1.getText();
```

- Declare a variable named Str of type String and initialize it with the value input by the user in the first text field. The value entered in the text field is retrieved using method getText().

```
jTextField2.setText(Str.toLowerCase());
```

- Convert the string named Str to lower case using the toLowerCase() method and then display the converted string in the second text field using the setText() method.

```
jTextField3.setText(Str.toUpperCase());
```

- Convert the string named Str to upper case using the toUpperCase() method and then display the converted string in the third text field using the setText() method.

The coding of the RESET button is exactly the same as learnt in the earlier examples.

Now that we know how to use the methods of the String class, let us next learn two more new methods - one to extract specified number of characters from a string and next to join two strings together. To learn these two methods, let us design an application called

Short Name. The aim of the application is to accept the First Name, Middle Name and the Last Name from the user and display his short name (i.e Last Name followed by his initials). Observe the following form carefully.

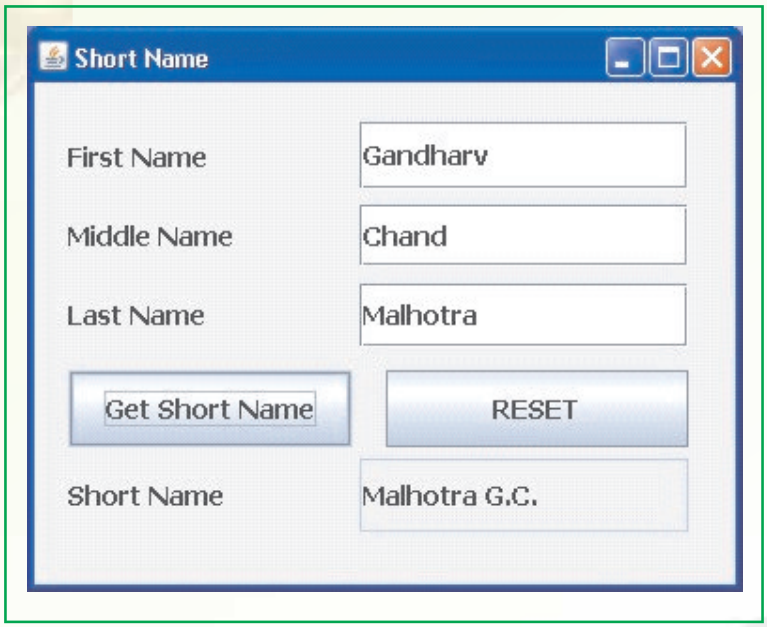


Figure 5.8 Sample Run of the Short Name Application

Let us first design the form as shown in Figure 5.8. First add a new JFrame form and set its title property to "Short Name". Now, add the following components on the form:

- Three editable text fields to accept the first, middle and last name from the user
- One non-editable text field to display the short name
- Two buttons - one to convert & display the short name and one to reset the form components
- Four labels - one against each of the text field to appropriately direct the user.

Change the properties of the components so that the form looks exactly like the one displayed in Figure 5.8. The next step is to associate code with both the buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 5.9.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    // To convert a full name to a short name
    String FirstName,MiddleName,LastName,ShortName="";
    FirstName=jTextField1.getText().substring(0,1);
    MiddleName=jTextField2.getText().substring(0,1);
    LastName=jTextField3.getText();
    ShortName=ShortName.concat(LastName);
    ShortName=ShortName.concat(" "); // to add a blank space
    ShortName=ShortName.concat(FirstName);
    ShortName=ShortName.concat("."); // to add a dot to
    separate FN and MN
    ShortName=ShortName.concat(MiddleName);
    ShortName=ShortName.concat(".");
    jTextField4.setText(ShortName);
}
```

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
    // Clear all text fields by initializing them with blank
    spaces
    jTextField1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
    jTextField4.setText("");
}
```

Figure 5.9 Code of the Short Name Application

Let us now understand the code in detail line by line:

```
String FirstName,MiddleName,LastName,ShortName="";
```

- Declare four variables of type String and initialize them with blanks (to ensure that it is an empty string).

```
FirstName=jTextField1.getText().substring(0,1); and

MiddleName=jTextField2.getText().substring(0,1);
```

- First retrieve the values of First Name and Middle Name entered in the first and second text fields using the getText() method and then extract the first character of both the First Name and Middle Name using the substring() method and store them in the variables FirstName and MiddleName respectively.

```
LastName=jTextField3.getText();
```

- Retrieve the value of the Last Name entered by the user in the third text field using getText() method and store it in the variable named LastName.

```
ShortName=ShortName.concat(LastName); and

ShortName=ShortName.concat(" "); and

ShortName=ShortName.concat(FirstName); and

ShortName=ShortName.concat("."); and

ShortName=ShortName.concat(MiddleName); and

ShortName=ShortName.concat(".");
```

- Initially the variable ShortName is blank. It is first joined together with LastName and then with a blank space after which it is joined together with variable FirstName, dot, variable Middlename and a dot again respectively using the concat() method. Finally, the content of the ShortName will be the LastName followed by the initials of the name entered by the user.

```
jTextField4.setText(ShortName);
```

- The ShortName is then displayed in the fourth text field using the setText() method.

Let us first design the form as shown in Figure 5.10. First add a new JFrame form and set its title property to "SMS Testing". Now, add the following components on the form:

- One editable text field to accept the SMS string
- Two non-editable text fields - one to display the number of characters entered by the user and the other to display how many more characters can be entered (to reach the maximum allowed length)
- Three labels - one against each of the text field to appropriately direct the user.

Change the properties of the components so that the form looks exactly like the one displayed in Figure 5.10. The next step is to associate code with the text field. Double click on the text field in the design window to reach the point in the source window where the code needs to be written. Add the code as given in Figure 5.11.

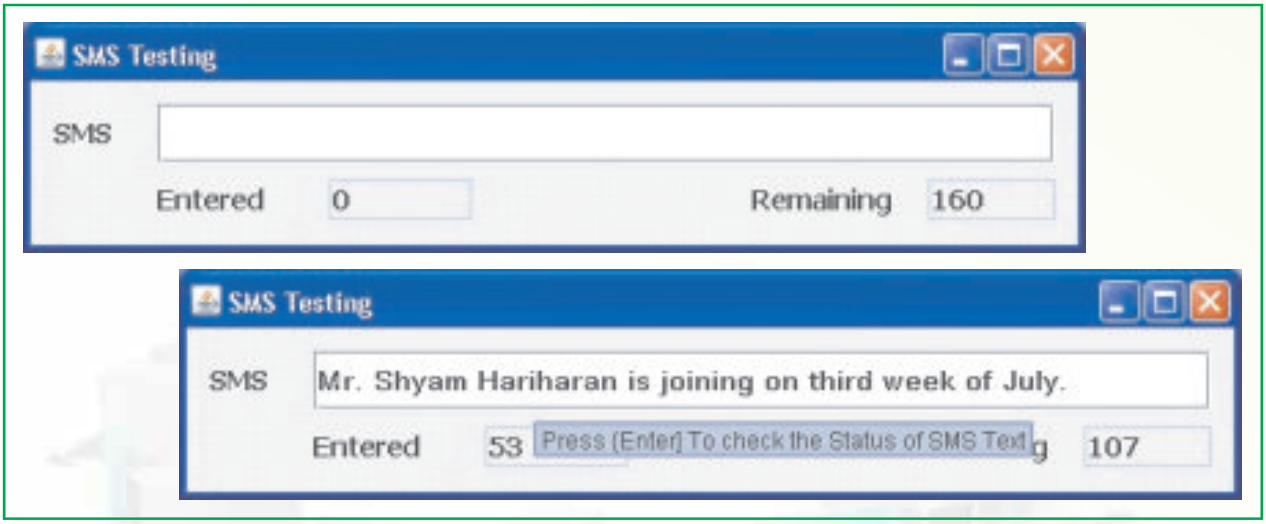


Figure 5.10 Sample Run of the SMS Testing Application

```
private void
jTextField1ActionPerformed(java.awt.event.ActionEvent evt)
{ //Display the total number of characters entered and
remaining to be
    //entered in a SMS
```

```
        String SMS=jTextField1.getText();
        int L=SMS.length();  // calculate the current length of
        the input message
        int Balance;
        Balance=160-L; // calculate the remaining no. of
        characters
        jTextField2.setText(Integer.toString(L));
        jTextField3.setText(Integer.toString(Balance));
    }
```

Figure 5.11 Code for the SMS Testing Application

Let us now understand the code in detail line by line:

```
String SMS=jTextField1.getText();
```

- Declare a variable named SMS of type String and initialize it with the value input by the user in the first text field. The value entered in the text field is retrieved using method getText().

```
int L=SMS.length(),Balance;
```

- Declare two variables - Balance and L of integer type. Initialize the variable named L with the length of the SMS entered by the user. The length is calculated using the in-built method length() of the String class.

```
Balance=160-L;
```

- Calculate the balance number of characters by subtracting the length of the SMS entered from 160 (the maximum number of characters that can be entered)

```
jTextField2.setText(Integer.toString(L)); and
jTextField3.setText(Integer.toString(Balance));
```

- Convert the variables L and Balance to type String using the toString() method and then display these values in the relevant text fields using the setText() method.

The SMS sample application developed above can further be modified with the help of a Text Area to give a better view of the SMS. Let us first design the form as shown in Figure 5.12. First add a new JFrame form and set its title property to "SMS Typist". Now, add the following components on the form:

- One editable text area to accept the SMS message
- Two non-editable text fields - one to display the number of characters entered by the user and the other to display how many more characters can be entered (to reach the maximum allowed length)
- Three labels - one against each of the two text fields and the text area to appropriately direct the user.
- Two buttons - one to reset the form components and the second to exit from the application

Change the properties of the components so that the form looks exactly like the one displayed in Figure 5.12. The next step is to associate code with the Text Area and the buttons. Double click on the text area and the two buttons, one by one, in the design window to reach the point in the source window where the code needs to be written. Add the code for the each of the three components as given in Figure 5.13.

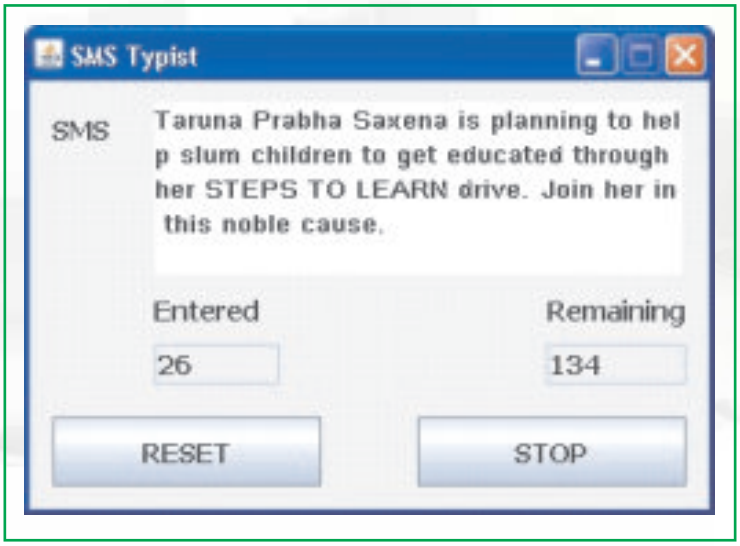


Figure 5.12 Sample Run of the SMS Typist Application

```
private void
jTextArea1CaretUpdate(javax.swing.event.CaretEvent evt)
{
    // Display the total number of characters input and
    remaining for an SMS
    String SMS=jTextArea1.getText();
    int L=SMS.length(),Balance;
    Balance=160-L;
    jTextField2.setText(Integer.toString(L));
    jTextField3.setText(Integer.toString(Balance));
}
```

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{ // Clear all text fields by initializing them with blank
spaces
    jTextArea1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
}
```

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{ //Exit the application
    System.exit(0);  //The parameter 0 indicates a normal
    termination
}
```

Figure 5.13 Code for the SMS Typist Application

The code for the text area is exactly same as the one written in the previous example with just a minor difference. In this case, the SMS is retrieved from the text area instead of the text field. The rest of the code is exactly the same. The code for the two buttons is also similar to all previous applications having a RESET and a STOP button.

Extra Reading

Observe that the event jTextField1ActionPerformed will display the number of characters input and left only after the user presses Enter marking the end of the input. If the user is interested in seeing the status of the number of words input and number of words that can be accepted on the spot while inputting the text, then instead of using the event jTextField1ActionPerformed(java.awt.event.ActionEvent evt), we can use the jTextField1CaretUpdate(javax.swing.event.CaretEvent evt) as shown below:

```
private void
jTextField1CaretUpdate(javax.swing.event.CaretEvent evt)
{
        String SMS=jTextField1.getText();
        int L=SMS.length(),Balance;
        Balance=160-L;
        jTextField2.setText(Integer.toString(L));
        jTextField3.setText(Integer.toString(Balance));
}
```

Note: For writing the code for jTextField1CaretUpdate Event, we need to use the Steps as shown in the following figure:

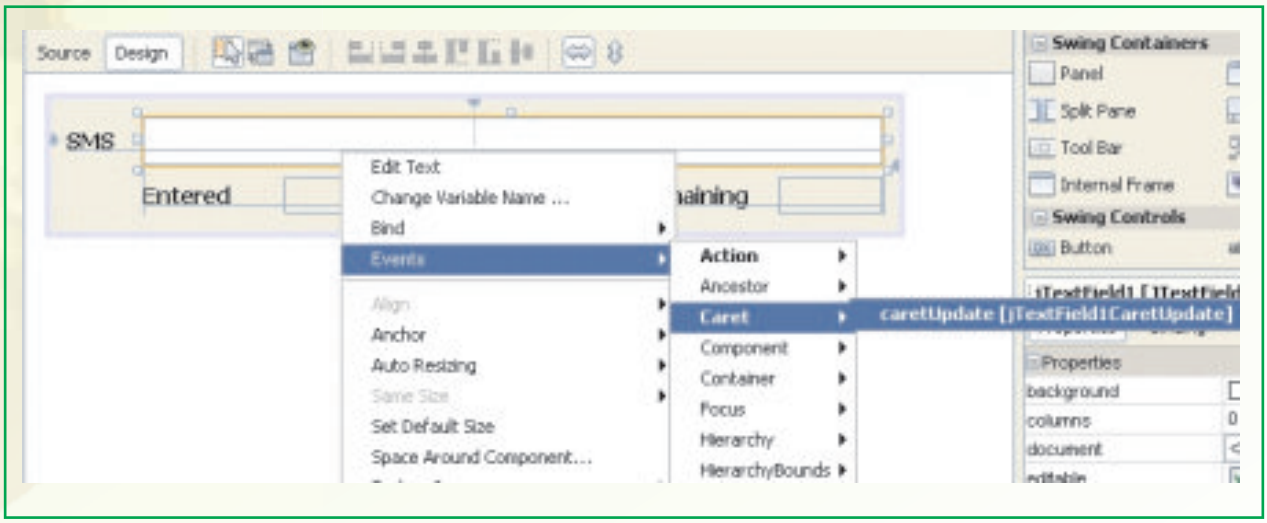


Figure 5.14 Steps for writing the code for jTextField1CaretUpdate Event

What will happen if the user enters more number of characters than the permissible range or enters unnecessary blank spaces before the beginning of the string? The SMS example can further be modified to solve the above problem. We will now develop an application which will remove the unwanted leading and trailing blanks and also truncate the SMS to the permissible range using the trim() and the substring() methods of the String class. Let us first design the form as shown in Figure 5.15. First add a new JFrame form and set its title property to "SMS Generator". Now, add the following components on the form:

- Two editable text areas - one to accept the SMS message and the second to display the final truncated SMS.
- Two non-editable text fields - one to display the number of characters entered by the user and the other to display how many more characters can be entered (to reach the maximum allowed length).
- Four labels - one against each of the two text areas and the two text fields to appropriately direct the user.
- Two buttons - one to truncate and display the SMS in the text area and the other to reset the form components.

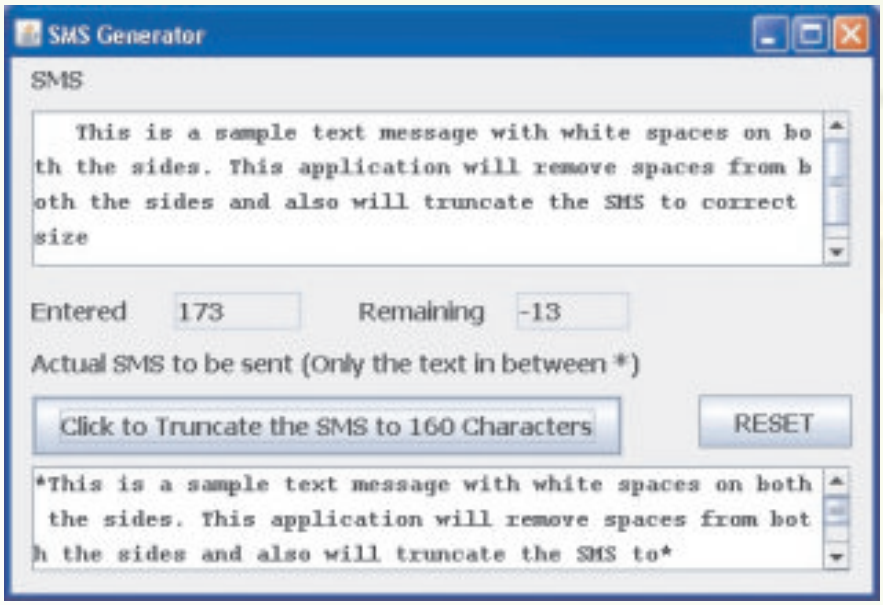


Figure 5.15 Sample Run of the SMS Generator Application

Change the properties of the components so that the form looks exactly like the one displayed in Figure 5.15. The next step is to associate code with both the buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 5.16.

Let us now understand the code in detail line by line:

```
String SMS,FinalSMS;
```

- Declare two variables named SMS and FinalSMS of type String.

```
SMS=jTextArea1.getText();
```

- Retrieve the SMS entered by the user in the first text area using the getText() method and store it in the variable SMS.

```
if (SMS.length()>160)
{
    SMS=SMS.substring(0,159);
}
```

- Check if the length of the SMS entered by the user is greater than 160 (using the length() method), then extract only first 160 characters (using substring() method) and store it in the variable SMS.

```
FinalSMS=SMS.trim();
```

- Remove White Spaces from both sides of the SMS input by the user and store the resultant SMS in the variable named Final SMS.

```
FinalSMS="*"+FinalSMS+"*";
```

- Add the character "*" on both sides of the message

```
jTextArea2.setText(FinalSMS);
```

- Display the final sms in the second text area using the setText() method.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{ // Truncate the SMS to 160 characters and display it
    String SMS,FinalSMS;
    SMS=jTextArea1.getText();
    if (SMS.length()>160)
    {
        SMS=SMS.substring(0,159); // Pick up the first 160
        characters
    }
    FinalSMS=SMS.trim(); //Removes White Space from both sides
    FinalSMS="*"+FinalSMS+"*"; //Add a '*' on both sides of
    the final SMS
    jTextArea2.setText(FinalSMS);
}
```

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{ //Clear all the text fields by initializing them with
blank space
    jTextArea1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
    jTextArea2.setText("");
}
```

```
private void
jTextArea1CaretUpdate(javax.swing.event.CaretEvent evt)
{ // Display the number of characters input and no. of
remaining characters
    String SMS=jTextArea1.getText();
    int L=SMS.length(),Balance;
    Balance=160-L;
    jTextField2.setText(Integer.toString(L));
    jTextField3.setText(Integer.toString(Balance));
}
```

Figure 5.16 Steps for writing the code for jTextField1CaretUpdate Event

Summary

- Java supports native classes like Math and String with predefined methods to make the programming process simpler.
- The pow(d1,d2) method of the Math class returns the value d1raised to d2.
- The round(d) method of the Math class returns the closest integer/float to d.
- The concat(str) method of the String class concatenates the string named str to the string calling the method.
- The length() method of the String class returns the number of characters in the string.
- The toLowerCase() and toUpperCase() methods of the String class return the string converted to lower case and upper case respectively.
- The substring(n1, n2) method of the String Class returns the extracted characters starting from the position n1 till n2.
- The trim() method of the String class returns the string after removing the leading and the trailing blanks.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. The _____ method is used to extract specified number of characters from a string.

   A. subStr() B. subString()

   C. takeString() D. extract()

2. Which of the following method(s) does not accept any arguments?

   A. pow() B. toLowerCase()

   C. round() D. All of the above

3. Which of the following statement is true?

   A. The trim() method is used to trim a string by removing spaces from one side

B. The trim() method is used to trim a number by removing zeroes

C. The trim() method is used to trim a string by removing spaces from both sides.

D. The trim() method is used to trim a string by removing extra characters.

4. Which of the following is not a valid method of the String class?

A. length() B. concat()

C. trim() D. round()

5. _____ is a class which contains methods for basic numerical operations like rounding off the number.

A. Number B. String

C. Math D. Integer

6. What will be the effect of executing the following code:

```
String F1="INTER",F2="MEDIATE";
F1=F1.concat(F2);
```

A. F2 will be concatenated at end of F1 and stored in F1.

B. F1 will be concatenated at end of F2 and stored in F1.

C. F1 will be concatenated at end of F2 and stored in F2.

D. F2 will be concatenated at end of F1 and stored in F2.

7. What will be the contents of L after executing the following code?

```
String F1="I#N#T#E#R",F2;    //# represents a space
F2=F1.trim();
int L=F2.length();
```

A. 5 B. 9

C. 10 D. 6

8. Which of the following is not a primitive data type in Java?

A. float B. int

C. string D. void

ANSWER THE FOLLOWING QUESTIONS

1. What is a class?
2. Name any four native classes of Java and briefly explain the use of each.
3. Give two appropriate advantages of using in-built methods over creating user defined methods.
4. Explain the general syntax of using any method of the String class. Is it in anyway different from using a method of the Math class?
5. What will be the contents of F1 and F2 after the following code is executed?

```
String F1="Hello",F2="Friend";

F1=F1.concat(F2);
```

6. Tanyaman is creating a simple application in java called "Password Checker" in which she needs to convert the characters input by the user in a particular case. Name two methods of the String class that she can use for this purpose.
7. Aryamani is creating a simple application in java called "Name Concatenator" in which he needs to concatenate the first name, middle name and last name input by the user in separate text fields. Name a method and an equivalent operator for this purpose.
8. Study the following code and answer the questions that follow:

```
double Number,RoundedNumber;

Number=Double.parseDouble(jTextField1.getText());

RoundedNumber=Math.round(Number*1000)/1000.0;

jTextField2.setText(Double.toString(RoundedNumber));
```

   a) How many variables have been declared in the above code? Identify and name them.

   b) How many objects have been declared in the above code?

   c) Name any one native class of Java used in the above code.

   d) Identify and name a method of this native class.

e) Name the method used to convert one type of data to another and also mention the data type before and after conversion.

LAB EXERCISES†

1. Create an application to accept two strings - First Name and Last name from the user and display the message Welcome with the complete name of the user.
2. Create an application to accept the radius of a circle, calculate the area and circumference and display the results in a message box after rounding off the area and circumference to an integer number.
3. Modify application to make sure that the user has not input the complete name in the first name text field.
4. Modify the Case Changer application developed in the lesson to display the input text in Title case using the substring(), toLowerCase() and toUpperCase() methods.

TEAM BASED TIME BOUND EXERCISES

(Team size recommended: 3 students each team)

1. Divide the class into groups of 4 students each and tell them to create a GUI application that allows a user to change his password according to the following guidelines:
   a. The form should accept the name and password from the user
   b. Only if the password matches with the pre-input password, the user should be allowed to proceed. (Hint - need to check the length and the contents and also convert the string to a particular case)
   c. Hide the initial text fields and now create form elements to accept input of first name, middle name and last name along with the age.
   d. Ensure that the age input is an integer and the first name, middle name and the last name are not more than 12 characters long.
2. Divide the students into groups of 6 and then give each group an algorithm which is predesigned by the teacher, and tell them to execute it. This algorithm should draw a figure with the children; the first team to find the figure and name it wins.

† The students should be encouraged to design appropriate forms for the applications themselves.

For example :

child[1].AtLeftOf(child[2])

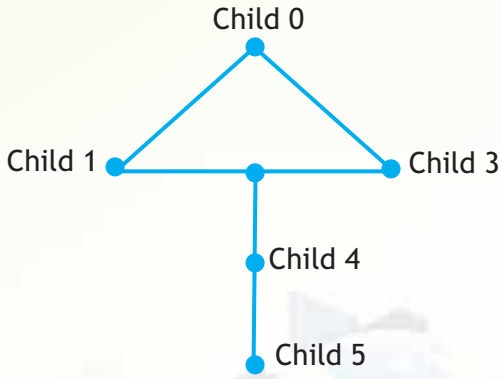
child[0].InFrontOf(child[2])

child[2].InFrontOf(child[4])

child[3].AtRightOf(child[2])

child[4]. InFrontOf(child[5])

The algorithm given above is to form the shape of an arrow as depicted in the following figure:



DATABASE CONNECTIVITY

Learning Objectives

After studying this lesson the students will be able to:

- State the need of saving data in the database
- Connect a GUI interface with data stored in database.
- Use various methods to retrieve data from a database.
- Write a complete GUI application with database connectivity
- State the advantages of developing applications in local languages

In the earlier lessons, you have learnt how to develop a GUI Interface using Netbeans and creating & manipulating data in a table of a database as two independent concepts. This lesson will help you to combine both these concepts together and help you to develop complete applications in which the GUI Interface will act as Front-End or Client Side application and Database will act as Back-End or Server-Side application.

Puzzle [6]

A person wanted to withdraw X rupees and Y paise from the bank. But the cashier made a mistake and gave him Y rupees and X paise. Neither the person nor the cashier noticed that. After spending 20 paise, the person counts the money. And to his surprise, he has double the amount he wanted to withdraw. Find X and Y.

(1 Rupee=100 paise)

Introduction

Imagine a swanky air-conditioned car showroom in prime location of the city with all modern car accessories available in an outlet within the showroom. The showroom has a

wonderful ambience, employs skilled salesperson with excellent communication skills to convince prospective buyers. They have a tie-up with various banks and financial agencies to provide lucrative payment plans to lure customers. What do you think he will sell? Do you think this showroom will be able to cover cost without selling cars? Do you think the skilled salesperson with excellent communication skills will be able to manufacture cars?

On the other hand there is a factory, as usual in a remote location, with abundant raw material, latest machinery, skilled workforce that is technically sound with latest automobile expertise. The factory keeps manufacturing cars but does not have any dealers. Factories are generally set up in remote industrial towns so do you think a remote area in Gujrat will be able to sell cars directly from their factory? Do you think the factory will be able to cover their costs incurred on raw materials without delivering cars? Do you think a skilled workforce can convince people to buy cars?

GUI Application (The Front-End)

As you know that the programmers develop applications for solving real life problems to simplify life of others. That is the reason, they try to develop applications, which require minimum inputs from the user and provide meaningful information in the form of output through their applications. The programmer develops these applications keeping in mind that the user will have a very basic understanding of handling mouse, keyboard and desktop applications. In the earlier lessons of GUI programs, you must have got a good idea of developing simple GUI Applications, in which user was typing in or clicking on certain inputs to produce meaningful results (output). If you see all these applications carefully, you will realize that none of them are capable of retaining the inputs or outputs for future reference or uses. So, typing in any number of inputs in the application carries a life span of one execution only and when the application is executed second time, it requires a new set of inputs from the user. Do you realize why it is like this? It is because; the data entered by you in a front end application was getting stored in temporary memory (RAM) and not getting saved on a hard disk (or any other secondary storage device).

Database (The Back-End)

The database helps you to save data permanently in secondary storage devices and keeps data ready for future reference, modification and addition. In the earlier lessons about database concepts, you have learnt about how to create a table, modify content of a table, add new content in the table, delete content from a table and retrieve content from a table in various forms. But you must have seen that to perform these activities, you were required to learn SQL commands like CREATE, UPDATE, INSERT, DELETE, SELECT and so on and each of these commands had their own syntax structures. Let us review this with the help of the Employee table shown below:

Table: Employee

Empno	Name	Salary
1001	Jasmine Taneja	45000
1009	Ravi Mathwad	34000
1015	Desai Ramakrishnan	58000
1004	Jigyasa Burman	49000

The SQL commands

- to add a new row in the table:

```
INSERT INTO Employee VALUES (1013, "Punya Sarthi", 51000);
```

- to retrieve the entire content from the table:

```
SELECT * FROM Employee;
```

- to increase salary by 10% for the employee whose Empno is 1009:

```
UPDATE Employee SET Salary=Salary*1.1 WHERE Empno=1009;
```

Obviously, you do not expect the user to learn all these commands to perform these activities. User always likes to access the information using the computer with some clicks of mouse or by typing in little textual input as per instructions provided preferably in his/her local language. So, there is a need to provide him/her with an interface to access or modify the information available in the database.

Database Connectivity (Front-End + Back-End)

After looking at the role of the Front-End and the Back-End, it is clear that the Car showroom in our introduction is actually our front end and the factory is nothing but the back end which supplies cars to the showroom. The showroom owner will be able to cover the costs incurred on running and maintaining the showroom only if the cars are delivered to the showroom by the factory. Similarly, the factory needs to send the cars to the showroom set up in an urban area for selling. Setting up an effective link between the two ends to facilitate the transfer of cars is the only solution of all the stated problems.

After going through the above paragraphs, you can understand that there is a requirement of creating an interface (the form), which a user may use for accessing information and also there is a requirement of keeping the information saved in the database (table) in an organized manner for future references and uses. So, we need to combine both these concepts together to develop a complete application for various domain specific problems. Now after learning about the front end and the back end, the next step is to learn how to set up a database connection that allows the front end to communicate with the back end. The two components essential to establish this connectivity are enumerated below and demonstrated in Figure 6.1:

- The JDBC API - software used to provide RDBMS access and execute SQL statements within java code.
- The JDBC Driver for MySQL - software component enabling a java application to interact with a MySQL database.

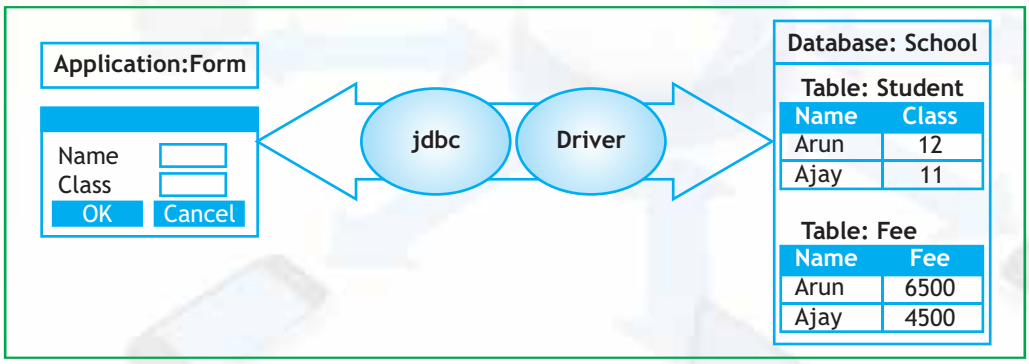


Figure 6.1 Communicating With a Database Using JDBC API and Driver

Know more

An application programming interface (API) is an interface implemented by a software program which enables it to interact with other software. It facilitates interaction between different software programs similar to the way the user interface facilitates interaction between humans and computers.

Adding [MySQL JDBC Driver] Library in NetBeans

The first step for establishing data connectivity is to add the MySQL JDBC Driver Library for use by your project. Remember, that this process has to be repeated every time you start a new project but not for new forms or applications added to an existing project. To add the MySQL JDBC Driver Library follow the given steps.

Step 1: Right click on the Project name and select the Properties option as shown in Figure 6.2.

Step 2: In the Properties dialog box,

i. choose the Libraries option from the Categories pane

ii. click on the Add Library button as shown in Figure 6.3.

iii. From the Add Library dialog box choose the MySQL JDBC Driver

iv. Click on the Create button as shown in Figure 6.3.

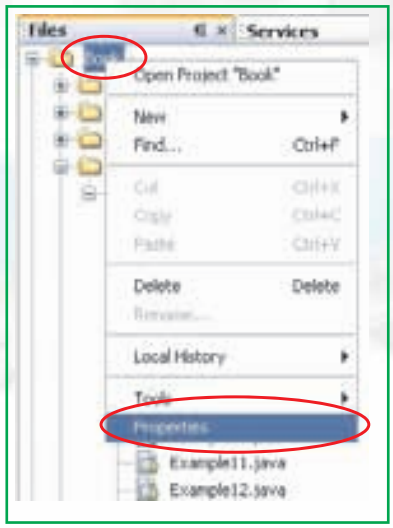


Figure 6.2 Opening Project Properties Dialog Box

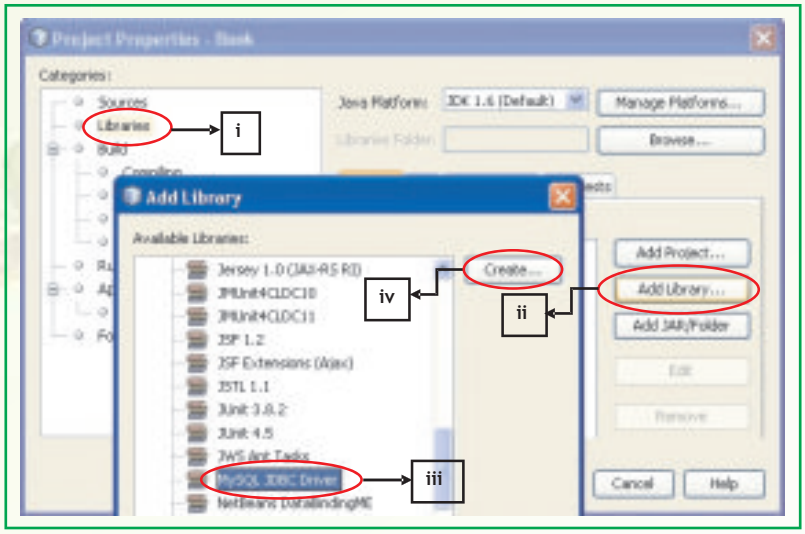


Figure 6.3 Adding the MySQL JDBC Driver From Available Libraries

Know more

The JDBC (Java Database Connectivity) API is a software for executing SQL statements. It provides RDBMS access by allowing you to embed SQL inside Java code. We can create a table, insert values into it, query the table, retrieve results, and update the table with the help of JDBC.

The driver is now added to the compile time libraries and this can be verified by clicking on the Compile Time Libraries tab in the Project Properties dialog box as shown in Figure 6.4. Click on OK to close the Project Properties dialog box.

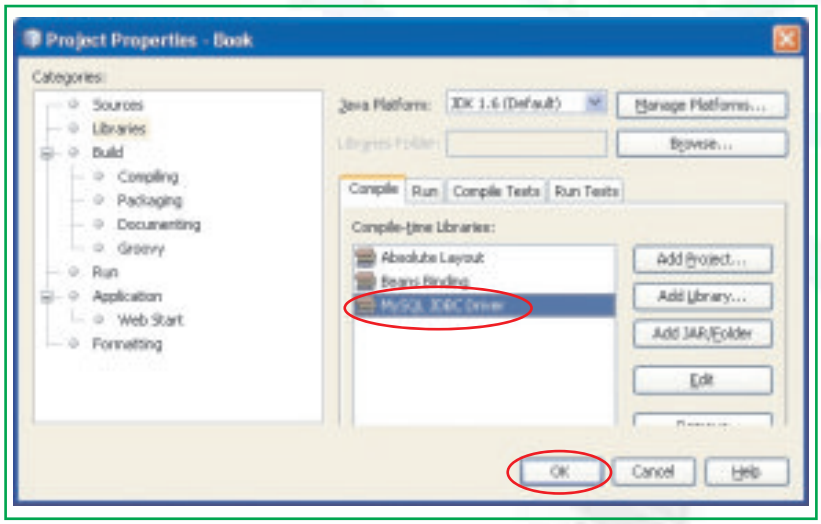


Figure 6.4 Compile-Time Libraries

Basic Libraries Required in the Application for Data Connectivity

Apart from the MySQL JDBC driver, we also need to import some basic class libraries which are essential for setting up the connection with the database and retrieve data from the database - namely DriverManager Class, Connection Class and Statement Class. These class libraries can be added using the following commands in our application code:

```
import java.sql.DriverManager;

import com.mysql.jdbc.Connection;

import com.mysql.jdbc.Statement;
```

Let us quickly understand the basic purpose for each of these class libraries:

- The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager is considered the backbone of JDBC architecture. DriverManager class manages the JDBC drivers that are installed on the system. Its getConnection() method is used to establish a connection to a database. It uses a username, password, and a jdbc url to establish a connection to the database and returns a connection object.
- A JDBC Connection represents a session/connection with a specific database. Connection interface defines methods for interacting with the database via the established connection. An application can have one or more connections with a single database, or it can have many connections with different databases.
- Once a connection is obtained we can interact with the database. To execute SQL statements, you need to instantiate (create) a Statement object using the connection object. A Statement object is used to send and execute SQL statements to a database.

In case any of these libraries are missing, then the execution results in an error as shown in Figure 6.5.

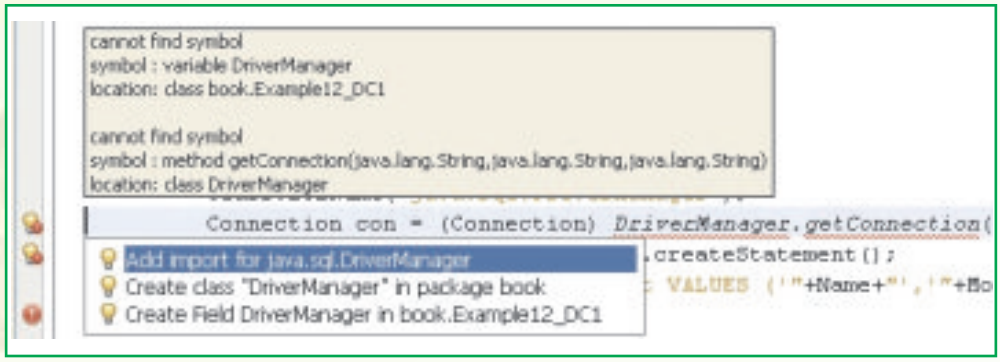


Figure 6.5 Error Window Indicating Missing DriverManager Library

Data Connectivity Application 1: Add Records in a Table Input Through a Form

Now that we have learnt the basics about establishing a connection, let us start developing an application that uses database connectivity. The aim of the first application that we will develop is to add data into a table.

Using MySQL command prompt, create a Table named "Contact" in the Database CBSE, with the specified table structure. The relevant command is shown in Figure 6.6.

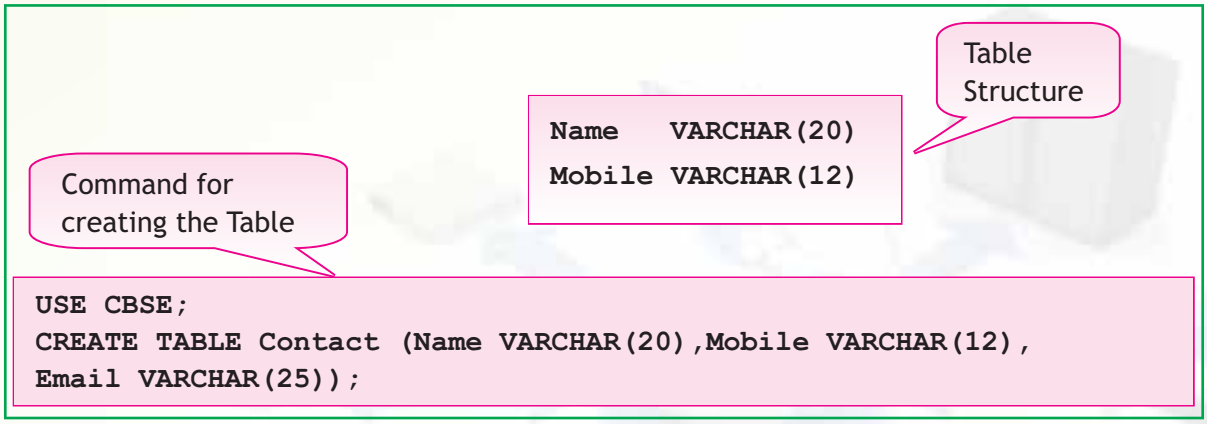


Figure 6.6 Command for creating the back end Table with a specified structure

Now let us design the form as shown in Figure 6.7. First add a new JFrame form and set its title property to "Contact List". Now, add the following components on the form:

- Three editable text fields to accept the name, mobile number and email address from the user.
- Three appropriate labels - one against each of the text field to direct the user.

- Two buttons - one to add the data supplied by the user in the database and one to exit from the application.

Change the properties of the components as learnt earlier so that the form looks exactly like the one displayed in Figure 6.7. The next step is to associate code with both the buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 6.8.

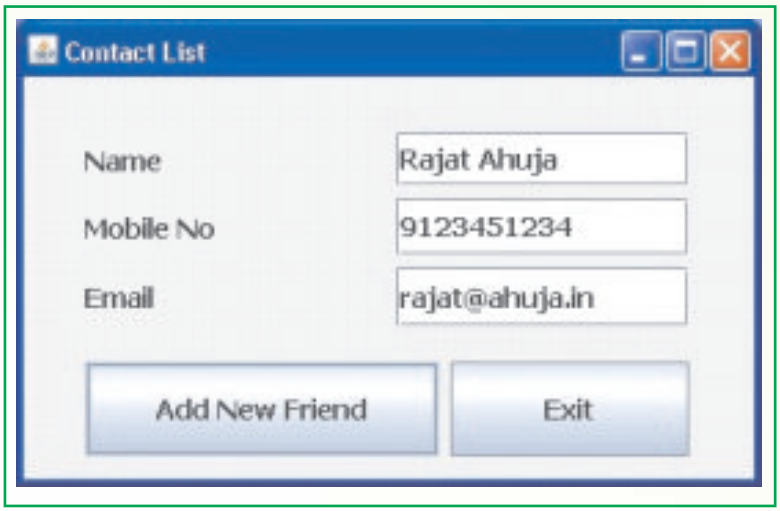


Figure 6.7 Front-end Form for the Contact List Application

```
//NetBeans Application 1 - To add new row(s) in the Contact Table
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
//Declare variables and store values retrieved from the
front end
    String Name=jTextField1.getText();
    String Mobile=jTextField2.getText();
    String Email=jTextField3.getText();
    try
    // this block is executed in case of normal execution
```

```
        {
            Class.forName("java.sql.DriverManager");
            Connection con = (Connection)
            DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/cbse",
            "root", "abcd1234");
            Statement stmt = (Statement) con.createStatement();
            String query="INSERT INTO Contact VALUES
            ('"+Name+"','"+Mobile+"','"+Email+"');";
            stmt.executeUpdate(query);
        }
        catch(Exception e)
        //this block is executed in case of an exception
        {
        //Display an error message in the dialog box for an exception
            JOptionPane.showMessageDialog (this, e.getMessage());
        }
    }
    // This part of the code is used to exit from the application
    private void
    jButton2ActionPerformed(java.awt.event.ActionEvent evt)
    {
        System.exit(0);
    }
```

Figure 6.8 Code for the Contact List Application

Know more

To catch an exception in Java, you write a try block with one or more catch clauses. Each catch clause specifies one exception type that it is prepared to handle. The try block delimits a bit of code that is under the watchful eye of the associated catchers. If the bit of code delimited by the try block throws an exception, the associated catch clauses will be examined by the Java virtual machine. If the virtual machine finds a catch clause that is prepared to handle the thrown exception, the program continues execution starting with the first statement of that catch clause. For example, in the above code if any error occurs while establishing a connection or executing the SQL query then the error message will be displayed in a message box due to the catch block immediately following the try block.

Let us now understand the code in detail line by line:

```
String Name=jTextField1.getText(); and
String Mobile=jTextField2.getText(); and
String Email=jTextField3.getText();
```

- Declare three variables named Name, Mobile and Email of type String and initialize them with the values input by the user in the text fields. The values are retrieved from the text fields using the getText() method.

```
Class.forName("java.sql.DriverManager");
```

- In this step of the jdbc connection process, we load the driver class by calling Class.forName() method with the Driver class name as an argument. Once loaded, the Driver class creates an instance of itself. A client can then connect to the Database Server through JDBC Driver.

```
Connection con = (Connection)
DriverManager.getConnection
    ("jdbc:mysql://localhost:3306/cbse", "root", "abcd1234");
```

- The getConnection() method of the DriverManager class is used to establish a connection to the cbse database. The method uses a username, password, and

a jdbc url to establish a connection to the database and returns a connection object named con.

- In the above command:

  jdbc:mysql - is the Database Driver Connection

  3306 - is the Default Port no on which MySQL runs

  cbse - is the Database Name

  root - is the User Name

  abcd1234 - is the Password

```
Statement stmt = (Statement) con.createStatement();
```

- Instantiate a Statement object called stmt from the connection object (named con created in the previous statement) by using the createStatement() method. A statement object is used to send and execute SQL statements to a database.

```
String query="INSERT INTO Contact VALUES
        ('"+Name+"','"+Mobile+"','"+Email+"');";
```

- Create a variable called query of type String and initialize it with the SQL statement to be executed (in this case the INSERT INTO statement).

```
stmt.executeUpdate(query);
```

- Execute the SQL statement stored in the variable query, using the executeUpdate() method of the Statement class. This results in adding the values stored in the three variables (Name, Mobile & Email) in the table Contact of the cbse database.

```
catch (Exception e)
{
    JOptionPane.showMessageDialog (this, e.getMessage());
}
```

- In case of an exception, retrieve the error message string using the getMessage() method and display it in a dialog box using the showMessageDialog() method.

Note that the forName() method is used to load the class specified as its argument at runtime.

Figure 6.9 shows a sample run of the above application.

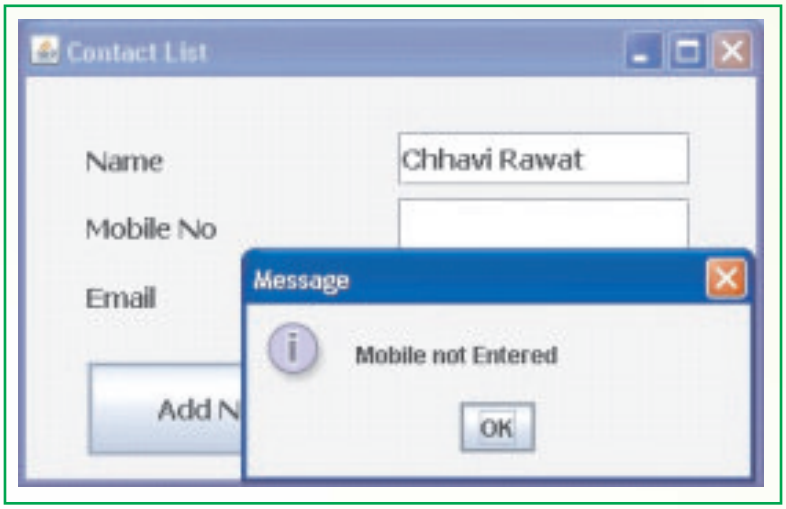


Figure 6.9 Sample Run of the Contact List Application Showing Error Message

The above application can be modified to handle basic data validations as shown in Figure 6.10.

Note that Data validation is the process of ensuring that a program operates on clean, correct and useful data. It uses routines, often called "validation rules" or "check routines", that check for correctness, meaningfulness, and security of data that are input to the system. For example, telephone numbers should include the digits and possibly the characters + & -. A more sophisticated data validation routine may check to see the user has entered a valid country code, i.e., that the number of digits entered match the convention for the country or area specified.

```
//Netbeans Application 1 (with Data Validation)
//- To add new row(s) in the Contact Table
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{  String Name=jTextField1.getText();
   String Mobile=jTextField2.getText();
   String Email=jTextField3.getText();
   /* Check if any of the variable is empty and accordingly
      display
     an appropriate error message */
   if (Name.isEmpty())
      JOptionPane.showMessageDialog(this,"Name not Entered");
   else if (Mobile.isEmpty())
    JOptionPane.showMessageDialog(this,"Mobile not Entered");
  else if (Email.isEmpty())
    JOptionPane.showMessageDialog(this,"Email not Entered");
  else
   { try
      {
          Class.forName("java.sql.DriverManager");
          Connection con = (Connection)
          DriverManager.getConnection
          ("jdbc:mysql://localhost:3306/cbse", "root",
          "abcd1234");
          Statement stmt = (Statement) con.createStatement();
```

```
            String query="INSERT INTO Contact VALUES
                ('"+Name+"','"+Mobile+"','"+Email+"');";
            stmt.executeUpdate(query);
            jTextField1.setText("");
            jTextField2.setText("");
            jTextField3.setText("");
        }
        catch (Exception e)
        {
        JOptionPane.showMessageDialog  (this,  e.getMessage());
        }
}
```

Figure 6.10 Code for the Contact List Application with Data Validations

Let us now understand the purpose of the additional commands added to the above code:

```
if (Name.isEmpty())
    JOptionPane.showMessageDialog(this,"Name not Entered");
else if (Mobile.isEmpty())
    JOptionPane.showMessageDialog(this,"Mobile not Entered");
else if (Email.isEmpty())
    JOptionPane.showMessageDialog(this,"Email not Entered");
else   { …..}
```

- Check if the value of any of the three variables (Name, Mobile, Email) is empty using the isEmpty() method with the help of the if..else conditional statement. In case any of the values is empty then display a error message in a dialog box using the showMessageDialog() method.

```
catch (Exception e)
    { JOptionPane.showMessageDialog (this, e.getMessage()); }
```

- In case of an exception, retrieve the error message string using the getMessage() method and display it in a dialog box using the showMessageDialog() method.

The code for the exit button is the same as all our previous applications.

Note that an exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

Running SQL Commands in Netbeans

After executing the Contact List Application, if you need to verify whether the row has been added to the table or not, then you need not go back to the MySQL prompt. Netbeans allows one to directly run SQL commands from the GUI interface.

Therefore, the content of the table can directly be tested by running SQL in Netbeans. To execute an SQL command in Netbeans perform the following steps:

Step 1: In the Services tab, right click on the Databases and select the New Connection option from the drop down menu.

Step 2: The New Database Connection dialog window opens up as shown in the Figure 6.11. Provide the values as shown in the figure and click OK. Clicking on OK in the above dialog window adds the connection to the existing list as shown in the Figure 6.12. Note that this step is required to be done only once - the first time when creating a connection.

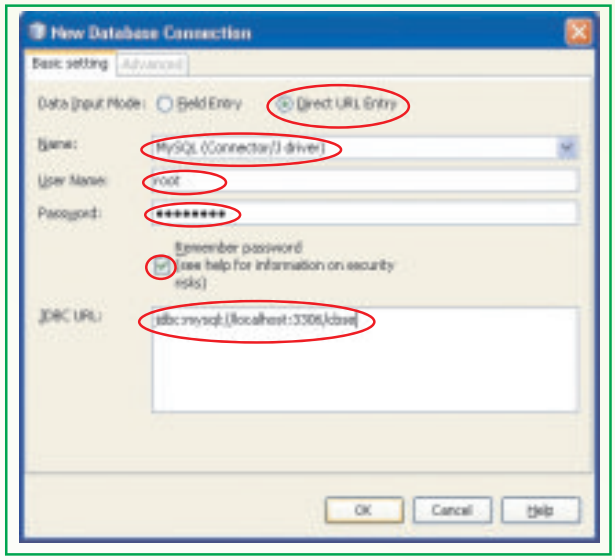


Figure 6.11 Setting Up a New Database Connection

Step 3: Right click on the newly added Connection and select the Connect option from the drop down menu. This step is to be repeated every time you start Netbeans. In the Connect dialog box that opens up, enter the user name and password and select the Remember Password checkbox (Selecting this checkbox ensures that this step is not to be repeated every time you start Netbeans) as shown in Figure 6.12.

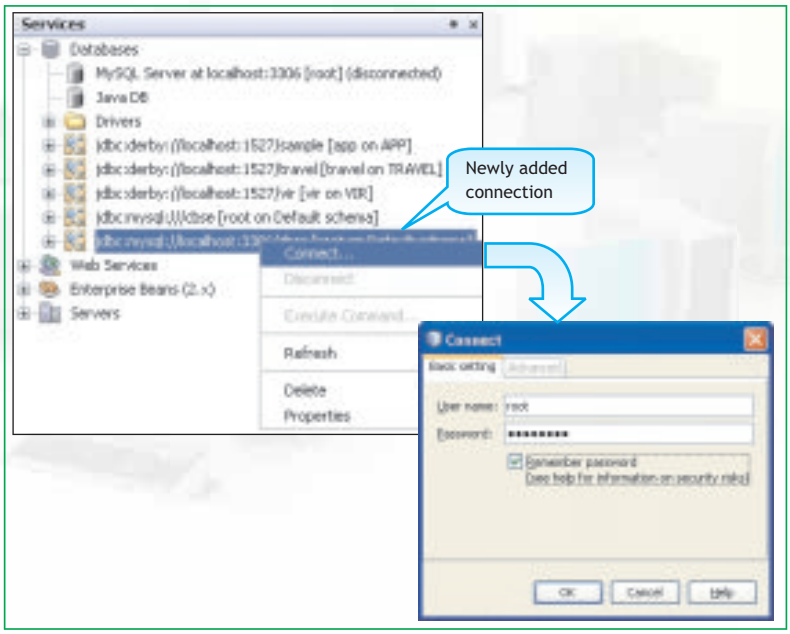


Figure 6.12 Connecting to the Appropriate Database

After establishing the connection, right click on the connection driver and select the Execute Command option from the drop down menu. The screen as shown in Figure 6.13 is displayed. Now, type in any SQL to be executed on the database and execute the command. The result is displayed in the bottom half of the window as shown in Figure 6.13.

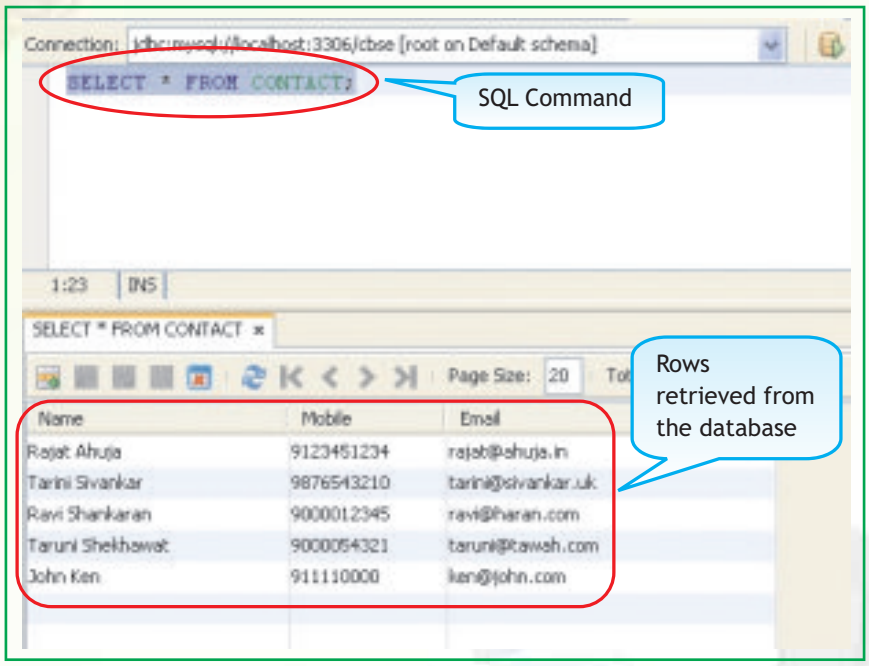


Figure 6.13 Executing SQL Command Directly in Netbeans

Data Connectivity Application 2: To Display Records Retrieved from a Database (Back End) in a Form (Front End) one by one

The next application that we will develop can be used to display data retrieved from a database in a form, one record at a time. Add a few records to the table Contact created above. Now design the form as shown in Figure 6.14. First add a new JFrame form and set its title property to "Search From Contacts". Now, add the following components on the form:

- One editable text field to accept the mobile number from the user.
- Two non-editable text fields to display the details - Name and E-mail of the contact being searched.

- Three appropriate labels - one against each of the text field to direct the user.
- Two buttons - one to search for the details on the basis of a mobile number supplied by the user and one to exit from the application.

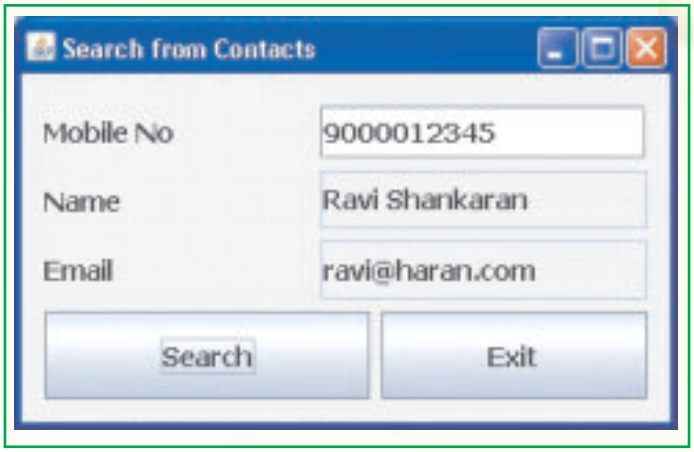


Figure 6.14 Sample Run of Search From Contacts Application

Change the properties of the components as learnt earlier so that the form looks exactly like the one displayed in Figure 6.14. The next step is to associate code with both the buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 6.15.

```
/*Netbeans Application 2 (Basic Part) - To search for a
matching row and display corresponding information from the
table Contact */
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
   // Search the table to find a record matching
   // the input Mobile no.
   String Mobile=jTextField1.getText();
   if (Mobile.isEmpty())
   //Execute this part if text field is blank
```

```
{
   jTextField2.setText("");
   jTextField3.setText("");
   JOptionPane.showMessageDialog
                              (this,"Enter the Mobile No");
}
// This part is executed if a Mobile No is
// input in the text field
else
{
   try
   {
       Class.forName("java.sql.DriverManager");
       Connection con = (Connection)
       DriverManager.getConnection
       ("jdbc:mysql://localhost:3306/cbse",
       "root", "abcd1234");
       Statement stmt = (Statement) con.createStatement();
       String query="SELECT NAME,EMAIL FROM CONTACT
                     WHERE MOBILE='"+Mobile+"';";
       ResultSet rs=stmt.executeQuery(query);
       if (rs.next())
       {
          String Name  = rs.getString("Name");
       //Retrieve the name
```

```
                String Email = rs.getString("Email");
                //Retrieve the email
                jTextField2.setText(Name);
                jTextField3.setText(Email);
            }
            // This part is executed if no matching record is found
            else
            JOptionPane.showMessageDialog
                                (this,"Sorry!No such Mobile No");
        }
        catch (Exception e)
        {
            JOptionPane.showMessageDialog(this, e.getMessage());
        }
    }
}
private void jButton2ActionPerformed
(java.awt.event.ActionEvent evt)
{
  System.exit(0);
}
```

Figure 6.15 Code for the Search From Contacts Application

Let us now understand the code in detail line by line:

```
String Mobile=jTextField1.getText();
```

- Declare a variable called mobile and initialize it with the value input by the user in the first text field which has been retrieved using the getText() method.

```
if (Mobile.isEmpty())
{
    jTextField2.setText("");
    jTextField3.setText("");
    JOptionPane.showMessageDialog(this,"Enter the Mobile No");
}
```

- Check if the mobile variable is empty i.e. in case the user has not input any mobile number then reinitialize both the text fields with blank spaces using the setText() method and show an error message using the showMessageDialog() method.

```
 else
{
    try
    {
        Class.forName("java.sql.DriverManager");
        Connection con = (Connection)
        DriverManager.getConnection
        ("jdbc:mysql://localhost:3306/cbse",root",
        "abcd1234");
        Statement stmt = (Statement) con.createStatement();
        String query="SELECT NAME,EMAIL FROM CONTACT
                WHERE MOBILE='"+Mobile+"';";
```

- If the user has entered a valid mobile number then perform all the steps for setting up a connection and initializing a query as learnt in the first database application.

```
ResultSet rs=stmt.executeQuery(query);
```

- Instantiate an object named rs of the ResultSet class and initialize it with the records returned by executing the SQL statement stored in the query variable. The executeQuery() method is used when we simply want to retrieve data from a table without modifying the contents of the table.

```
if (rs.next())
```

- Check if there is a matching record in the result set using the if construct. The next() method of the result set is used to move to the next record of the database.

```
String Name  = rs.getString("Name");
String Email = rs.getString("Email");
jTextField2.setText(Name);
jTextField3.setText(Email);
```

- Create two variables named Name and Email and initialize them with the values retrieved from the table using the getString() method. Next display these values retrieved from the table in the relevant text fields using the setText() method.

```
else
{ JOptionPane.showMessageDialog
    (this,"Sorry!No such Mobile No"); }
```

- If there is no matching record found in the table then display an error message using the showMessageDialog() method.

```
catch (Exception e)
{
    JOptionPane.showMessageDialog(this, e.getMessage());
}
```

- In case of an exception, retrieve the error message string using the getMessage() method and display it in a dialog box using the showMessageDialog() method.

A sample run of the validation check applied in the above application is shown in Figure 6.16

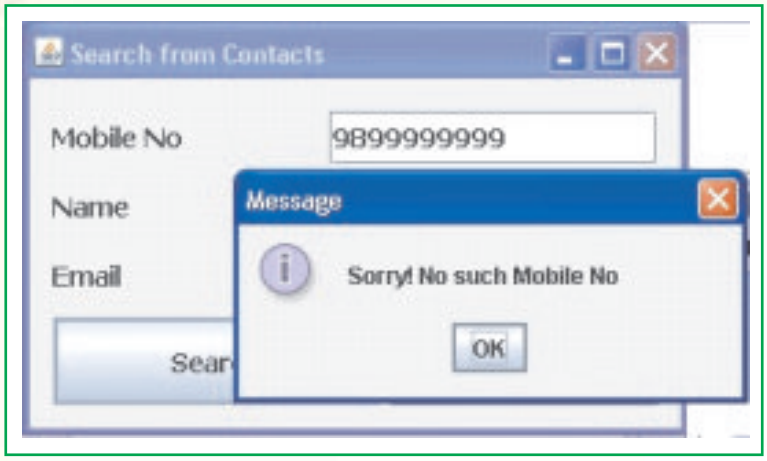


Figure 6.16 Validation Message in Search From Contacts Application

Just like we need to add the DriverManager, Connection and Statement class libraries, similarly we need to import the ResultSet class library when we are retrieving data from a database. In case the above mentioned library is not imported then the error window is displayed as shown in Figure 6.17.

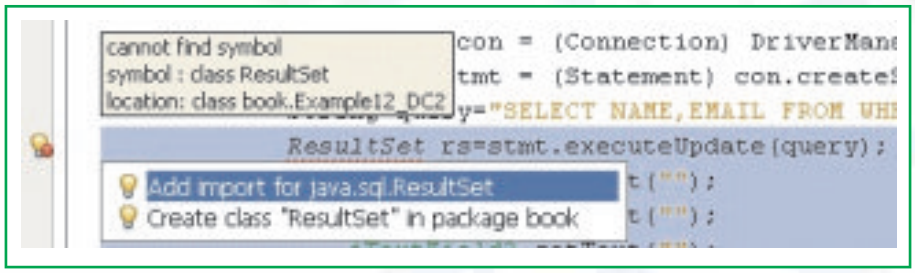


Figure 6.17 Error Message Indicating Missing ResultSet Class Library

Data Connectivity Application 3: To Display the Details Stored in the Back-end Based on the Name Input by the User in a Form.

Observe the sample run of our next application as shown in Figure 6.18 carefully. Can you observe the difference in this application as compared to our previous one?

In the previous application, we are searching on mobile number which will be unique i.e. we will always have a single matching record. What will happen if we perform a search on Name? In this case there is a possibility that we may have more than one matching record. So the major difference between the two applications is that in the new application we will aim at performing a search in which multiple records may be returned.

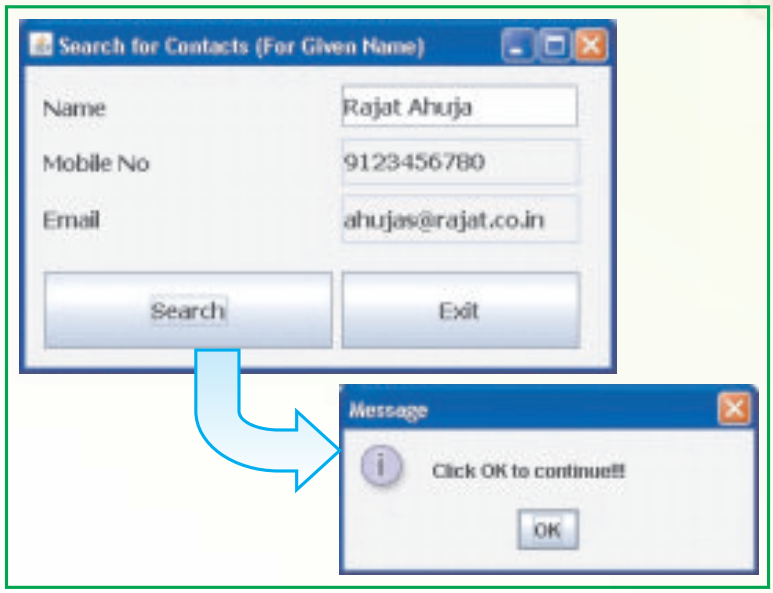


Figure 6.18 Sample Run of the Search For Contacts (For Given Name)

The form design and most of the coding is similar with minor differences. Observe the code given in Figure 6.19 carefully and try to point out the differences.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
   String name=jTextField1.getText();
   if (name.isEmpty())
      JOptionPane.showMessageDialog(this,"Name not Entered");
   else
   {
```

```
try
{
    Class.forName("java.sql.DriverManager");
    Connection con = (Connection)
    DriverManager.getConnection
    ("dbc:mysql://localhost:3306/cbse", "root",
    "abcd1234");
    Statement stmt = (Statement) con.createStatement();
    String query="SELECT Mobile,Email FROM Contact
                  WHERE Name='"+name+"';";
    ResultSet rs=stmt.executeQuery(query);
    int Found=0;
    while(rs.next())
    // Till there are records in the result set
    {
        String mobile = rs.getString("Mobile");
        String email  = rs.getString("Email");
        jTextField2.setText(mobile);
        jTextField3.setText(email);
        JOptionPane.showMessageDialog
                  (null,"Click OK to continue!!!");
        Found++;
        //Increment the variable to indicate a matching
        //record has been found
    }
```

```
            if (Found==0)
                JOptionPane.showMessageDialog
                (this,"Sorry! No such Name in Contact List");
    }
      catch (Exception e)
      {
         JOptionPane.showMessageDialog(this, e.getMessage());
      }
      }
}
```

Figure 6.19 Code for the Search for Contacts (For Given Name) Application

Let us now discuss the changes (additional commands) made to the previous code one by one:

i) The first change is declaration of a new integer variable named Found. This variable is initialized to 0 and is used to keep a track of how many matching records have been found.

ii) The second change is that we have used while loop to traverse through all the records in the table to find all possible matching records.

iii) The third change is the statement used to increment the variable found each time a matching record is encountered.

Data Connectivity Application 4: To Display Records Retrieved From the Back-end in the Table Component

In all the previous applications we have aimed at displaying only one record at a time in a form. What if we want to see all the records in one go? The next application is aimed at solving this problem. The aim of this application is to display all the records in a tabular form.

First, create a new table named Official with the same structure as that of table Contact. Add few records in this table.

Now let us design the form. First add a new JFrame form and set its title property to "Contact List". Now, add the following components on the form:

- One table to display the records retrieved from the table.
- Two radio buttons to select whether the records of the Contact table are displayed or the records of the Official table are displayed.
- Two appropriate labels - one against each of the radio button to direct the user.
- Three buttons - one to refresh data displayed in the table, one to reset the table and one to exit from the application.

Change the properties of the radio buttons, labels and button components as learnt earlier so that the form looks exactly like the one displayed in Figure 6.20. Before associating the code with the buttons, we need to customize the table component.

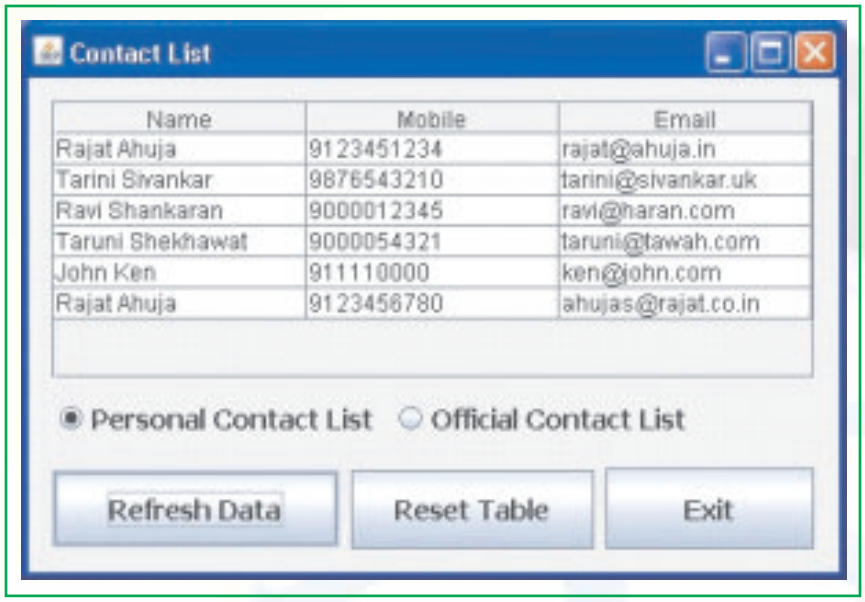


Figure 6.20 Form Design of the Modified Contact List Application

Right click on the table component and select Table Contents option to open the Customizer Dialog window. In this window choose the Columns tab and customize the column names by changing the Title property of each column as shown in Figure 6.21.

Next choose the Rows tab and delete all the four default rows. These rows have to be deleted otherwise they appear as blank rows in the final output and the retrieved data is displayed from fifth row onwards.

Once the form has been totally customized, the next step is to associate code with all the three buttons. Double click on the buttons one by one in the design window to reach the point in the source window where the code needs to be written. Add the code for each of the buttons as given in Figure 6.22.

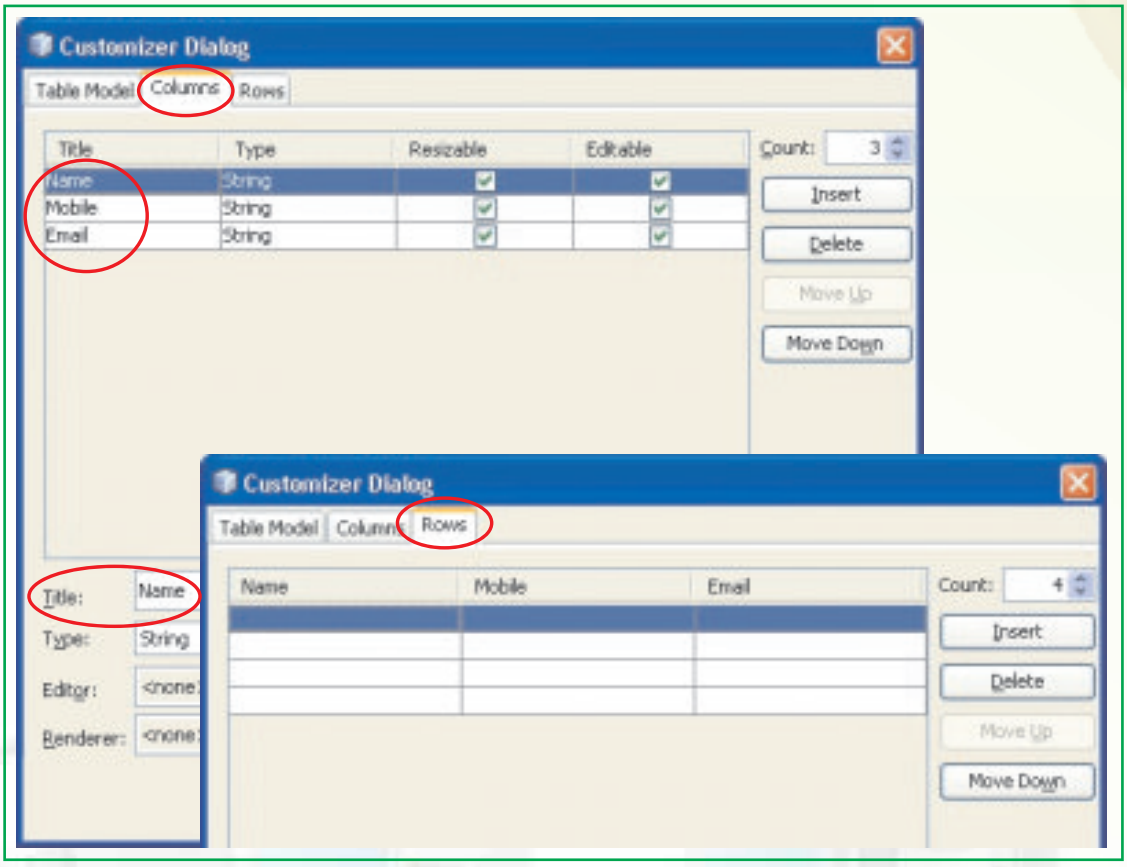


Figure 6.21 Customizing Columns and Rows of the Table Component of the Form

Before proceeding to writing the code, let us understand the purpose of the two buttons - Refresh Data and Reset Table. The purpose of the Refresh Data button is to display the data from the selected table without clearing the previous data. This means that the new data will be displayed in continuation of the previous displayed data. Whereas, if we press the Reset Table button, then it deletes all previous rows of the table component of the form.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
   DefaultTableModel model = (DefaultTableModel)
   jTable1.getModel();
   try
   {
      Class.forName("java.sql.DriverManager");
      Connection con = (Connection)
          DriverManager.getConnection
          ("jdbc:mysql://localhost:3306/cbse",
                        "root", "abcd1234");
      Statement stmt = (Statement) con.createStatement();
      String Tname;
      if (jRadioButton1.isSelected())
          Tname="Contact";
      else
          Tname="Official";
      /* The query is executed on different tables depending
      upon the Purpose */
      String query="SELECT * FROM "+Tname+";";
      ResultSet rs = stmt.executeQuery(query);
      while(rs.next())
      {
          String Name   = rs.getString("Name");
          String Mobile = rs.getString("Mobile");
```

```
            String Email  = rs.getString("Email");
            model.addRow (new Object[] {Name, Mobile,Email});
        }
    }
    catch (Exception e)
    {
        JOptionPane.showMessageDialog (this, e.getMessage());
    }
}
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
    DefaultTableModel model = (DefaultTableModel)
    jTable1.getModel(); int rows=model.getRowCount();
    if (rows>0)
   {
       for (int i=0; i<rows; i++)
            model.removeRow(0);  // To remove all rows from
            current model
    }
}
private void
jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{
    System.exit(0);
}
```

Figure 6.22 Code for the Modified Contact List Application

Let us now understand the code in detail line by line:

```
DefaultTableModel model = (DefaultTableModel)
jTable1.getModel();
```

- To retrieve the Model of the newly created table and type cast it to the Default Model.

```
Class.forName("java.sql.DriverManager");
Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/cbse",
                            "root", "abcd1234");
Statement stmt = (Statement) con.createStatement();
```

- Instantiate a Connection and a Statement object as before

```
String Tname;
```

- Declare a variable named Tname of type String. This variable will be sued to store the name of the table from which the data has to be retrieved.

```
if (jRadioButton1.isSelected())
    Tname="Contact";
else
    Tname="Official";
```

- Check which radio button has been selected and accordingly initialize the variable Tname.

```
String query="SELECT * FROM "+Tname+";";
```

- Create a variable called query of type String and initialize it with the SQL statement to retrieve all records from the specified table. The value of the specified table is stored in the variable Tname and so this variable is concatenated with the normal SQL statement.

```
ResultSet rs = stmt.executeQuery(query);
```

- Execute the SQL query using the executeQuery() method. Instantiate a variable named rs of class ResultSet to store the records returned by the SQL query.

```
while(rs.next())
{
    String Name   = rs.getString("Name");
    String Mobile = rs.getString("Mobile");
    String Email  = rs.getString("Email");
    model.addRow (new Object[] {Name, Mobile,Email});
}
```

- Till there are records in the database, keep retrieving the values of Name, Mobile and Email from the table one by one using the getString() method and store them in the three String variables using the assignment operator.
- After storing the values in the variables, display these values in the table component on the form using the addRow() method.

```
int rows=model.getRowCount();
```

- Declare a variable named rows of type integer and initialize it with the total number of rows in the table component of the form using the getRowCount() method.

```
if (rows>0)
{
    for (int i=0; i<rows; i++)
        model.removeRow(0);
}
```

- If the number of rows is greater than zero, then use a for loop to reset the table component i.e. delete all the rows using the removeRow() method. Note that the parameter passed to the removeRow() method is always zero as we are always deleting the topmost row and this process is repeated rows number of times to ensure that all the rows are deleted.

Data Connectivity Application 5: To Add Records in a Table Accepted Through a Form with Hindi Interface

Now that we are clear about data connectivity, let us learn a new feature - to add multilingual support facility to our Netbeans form and database and then display records in our native language as shown in Figure 6.23.

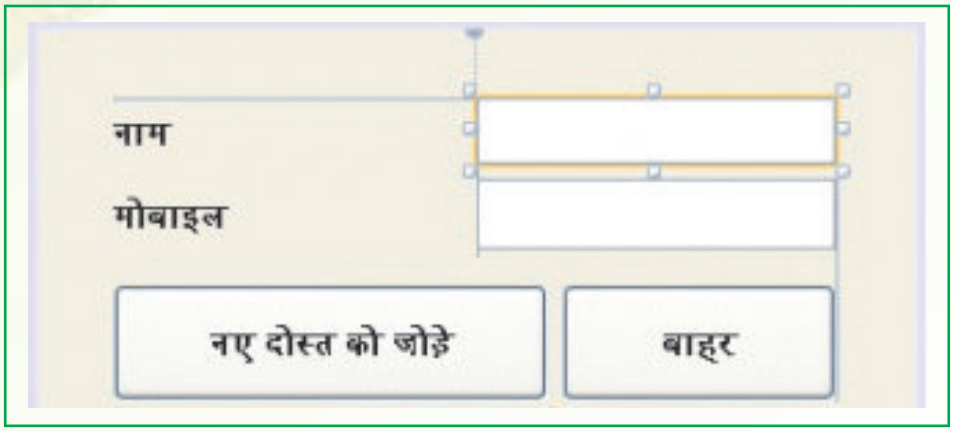


Figure 6.23 Form Design of the Contact List Application with a Hindi Interface

First create a database named *CBSEHINDI* using the following command:

```
CREATE DATABASE CBSEHINDI DEFAULT CHARACTER SET UTF8;
```

Note that the DEFAULT CHARACTER SET UTF8 has been added to the above command to have Unicode support for entering and processing data in HINDI language. Adding this command while creating the database ensures that all tables created within this database support Hindi language processing.

Next create a table named Contact in the CBSEHINDI database using the following command:

```
CREATE TABLE Contact (Name VARCHAR(20),Mobile VARCHAR(12),
                      Email VARCHAR(25));
```

Now while creating the form as shown above for the Netbeans application, remember to use unicode supported fonts for jTextFields, jLabels and jButtons (such as Arial Unicode MS) as shown in Figure 6.24.

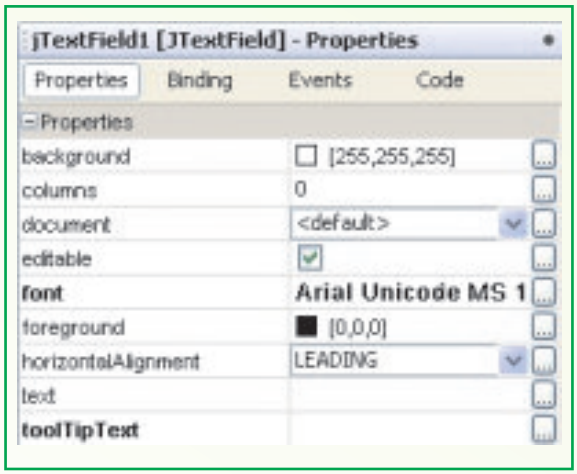


Figure 6.24 Setting the Font Properties for Multilingual Support

The code will be similar as the earlier example shown in Data Connectivity Application 1 (Only change the name of the database to cbsehindi as shown in the following code) and

```
Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/cbsehindi","root",
"abcd1234");
```

A sample run of the Hindi interface application is shown in Figure 6.25.

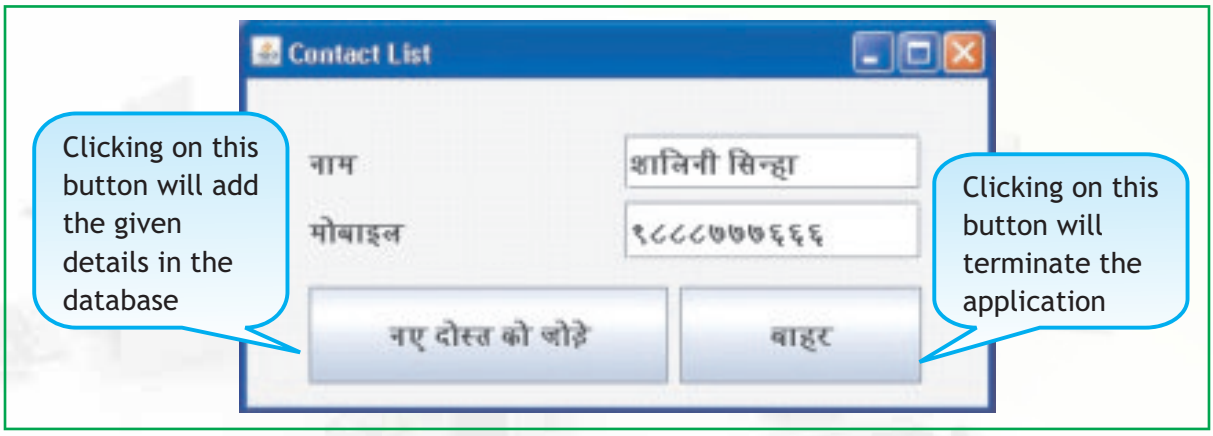


Figure 6.25 Sample Run of the Contact List Application with Hindi Interface

All the records in the cbsehindi database are stored in Hindi. Therefore, if we execute an SQL command to display all the records of the database then the records will be displayed in Hindi as shown in Figure 6.26.

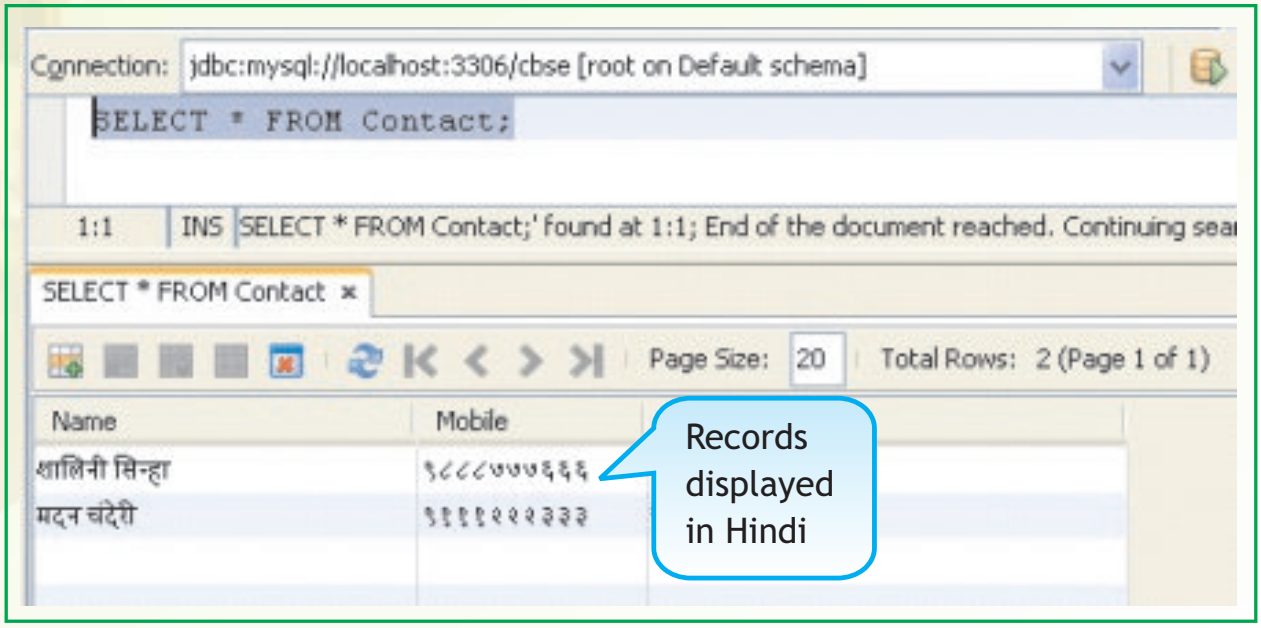


Figure 6.26 Running Simple SQL Statement in Netbeans for Data Display in Hindi

Future Trends

Cloud Computing

Cloud computing is a technology used to access services offered on the Internet cloud. Everything an Informatics System has to offer is provided as a service, so users can access these services available on the "Internet Cloud" without having any previous know-how. The cloud is just one example of a virtualized computing platform, and the next generation of developer tools must enable developers to build software that deploys and performs well in cloud and other virtual environments.

Summary

- A GUI (The Front-End) helps the user to design forms to accept data and provide instructions for retrieving information from the backend.
- A database (The Back-End) helps in storing data in organized manner for future reference and use

- A link needs to be established between the front end form and the back end table using jdbc driver (or an equivalent) to facilitate communication between the two.
- A complete application requires a combination of all three concepts i.e. it requires a user friendly interface, an efficient connection link and a database
- The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver.
- The JDBC Connection interface defines methods for interacting with the database via the established connection.
- To execute SQL statements, you need to instantiate (create) a Statement object using the connection object.
- The getConnection() method of the DriverManager class is used to establish a connection to the database.
- The getMessage() method is used to retrieve the error message string.
- The executeUpdate() method of the Statement class is used to update the database with the values given as an argument.
- The executeQuery() method is used when we simply want to retrieve data from a table without modifying the contents of the table.
- The next() method of the result set is used to move to the next record of the database.
- The getModel() method is used to retrieve the model of a table.
- The getRowCount() method is used to retrieve the total number of rows in a table.
- The removeRow() method is used to delete a row from a table.
- It is possible to add Multilingual support in MySQL database by setting the appropriate character set while creating the database.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. We may use _____ to develop the Front-End of an application.

   A. GUI B. Database

   C. Table D. None of the above

2. The _____ method is used when we simply want to retrieve data from a table without modifying the contents of the table

   A. execute() B. queryexecute()

   C. query() D. executeQuery()

3. Which of the following class libraries are essentially required for setting up a connection with the database and retrieve data?

   A. DriverManager B. Connection

   C. Statement D. All of the above

4. The _____ is required to establish connectivity between the Java code and the MySQL database.

   E. JDBC API F. JDBC Driver

   G. Both A and B

5. To execute SQL statements, you need to instantiate a _____ object using the _____ object.

   A. Connection , Statement B. Statement, Connection

   C. JDBC Driver, Statement D. JDBC DriverManager, Statement

6. Which of the following statements are false?

   A. The getConnection() method is used to establish a connection.

   B. An application can have more than one connection with a database.

   C. An application cannot have more than one connection with a database.

   D. An application can have many connections with different databases.

7. The _____ method is used to instantiate a statement object using the connection object

   A. getConnection() B. getStatement()

   C. createConnection() D. createStatement()

8. The _____ method is used to move to the next row of the result set.

   A. next() B. findNext()

   C. last() D. forward()

ANSWER THE FOLLOWING QUESTIONS

1. What is a GUI?
2. Explain the terms ResultSet with respect to a database.
3. Explain the usage of the following methods:

   a) next()

   b) removeRow()

4. Give two appropriate reasons of connecting GUI form with a Database.
5. What is JDBC? What is the importance of JDBC in establishing database connectivity.
6. Name the 3 essential class libraries that we need to import for setting up the connection with the database and retrieve data from the database.
7. Explain the importance of:

   a) JDBC b) forName() method

   c) getModel() method

8. Consider the following code and answer the questions that follow:

```
String query="SELECT NAME,EMAIL FROM CONTACT"
ResultSet rs=stmt.executeQuery(query); //Statement 1
if (rs.next())                          //Statement 2
{
```

```
        String Name  = rs.getString("Name");
        String Email = rs.getString("Email");
        jTextField2.setText(Name);
        jTextField3.setText(Email);
    }
```

a) Name the table from where the data is being retrieved

b) How many objects of the native String class have been declared in the code? Name the objects.

c) Identify the object name, class name and method name used in the statement labeled as Statement 1.

d) Explain the function of the statement labeled as Statement 2.

LAB EXERCISES†

1. Modify the SMS application developed in Chapter 5 to store the messages in a table along with the number of characters in a message.
2. Modify the password application developed in Chapter 5 to store the name and password of users in a table. The application should also update the password whenever the user modifies his password and should keep a count on how many times the password is being updated.
3. Design a GUI Interface for executing SQL queries for the above developed Password Application to accomplishing the following tasks:
   A. Retrieve a list of all the names and display it in a table form.
   B. Retrieve a list of names where name begins with a specific character input by the user and display it in a table form.
   C. Display all records sorted on the basis of the names.
4. Modify the password application developed above to handle processing in Hindi language.

TEAM BASED TIME BOUND EXERCISES

(Team size recommended: 3 students each team)

1. Divide the class in groups of four and tell each group to develop a phone book application to store and retrieve information about their friends on the basis of name, phone number or birthday. The information stored may include name, phone number, address, birthday, category etc.

2. Each group should next write SQL queries (to be executed on the above created table) for the following(and design appropriate interface element):

   A. Retrieve a list of all the users and display it in a table.

   B. Retrieve a list of users whose name begins with a specific character input by the user and display it one at a time in a form.

   C. Search and display records of the users whose birthday is in a particular month where month is input by the user.