MODULE 5

Graphical User Interface & Database support of Java

CHAPTER 1

SWING

SWING FUNDAMENTALS

Java Swing is a GUI Framework that contains a set of classes provide more powerful and flexible GUI components than AWT.

Swing provides the look and feel of modern Java GUI.

Swing library is an official Java GUI tool kit released by Su Microsystems.

It is used to create graphical user interface with Java.

Swing classes are defined in javax.swing package and its su packages.

Java Swing provides platform-independent and lightweig components.

a Swing is a part of Java Foundation Classes (JFC) that is *used ate window-based applications*.

s built on the top of AWT (Abstract Windowing Toolkit) API a irely written in java

ne Java Foundation Classes (JFC) are a set of GUI components whi mplify the development of desktop applications.

he javax.swing package provides classes for java swing API such Button, JTextField, JTextArea, JRadioButton, JCheckbox, JMe ColorChooser etc.

Features of Swing

atform Independent:

t is platform independent, the swing components that are used
build the program are not platform specific.

t can be used at any platform and anywhere.

ghtweight:

Swing components are lightweight which helps in creating the
ighter.

Swings component allows it to plug into the operating system us
nterface framework that includes the mappings for screens
device and other user interactions like key press and mou
movements.

ugging:

t has a powerful component that can be extended to provide th
support for the user interface that helps in good look and feel
the application.

t refers to the highly modular-based architecture that allows it
plug into other customized implementations and framework f
user interfaces.

anageable: It is easy to manage and configure. Its mechanism ar
mposition pattern allows changing the settings at run time
ell. The uniform changes can be provided to the user interfa
ithout doing any changes to application code.

VC:

They mainly follows the concept of MVC that is Model Vie Controller.

With the help of this, we can do the changes in one compone without impacting or touching other components.

t is known as loosely coupled architecture as well.

ustomizable:

Swing controls can be easily customized. It can be changed and th visual appearance of the swing component application independent of its internal representation.

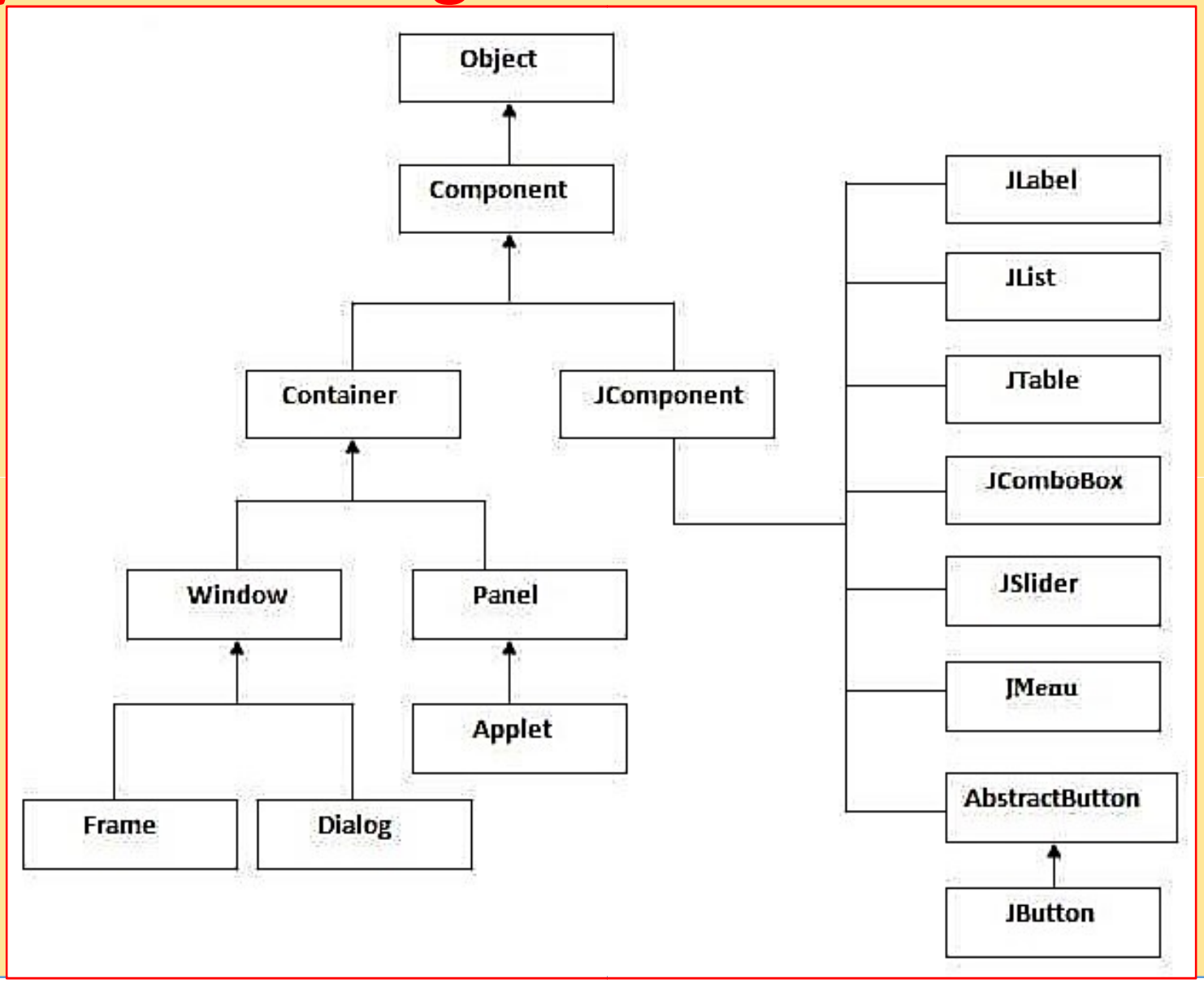
ch Controls :

Swing provides a rich set of advanced controls like Tre TabbedPane, slider, colorpicker, and table controls.

Difference between AWT and Swing

Java AWT	Java Swing
AWT components are **platform-dependent.**	Java swing components are **platform-independent.**
AWT components are **heavyweight.**	Swing components are **lightweight.**
AWT **doesn't support pluggable look and feel.**	Swing **supports pluggable look and feel**
AWT provides **less components** than Swing.	Swing provides **more powerful compone** such as tables, lists, scrollpanes, colorchoo tabbedpane etc.
AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC.**

Hierarchy of Java Swing classes



The Model-View-Controller Architecture

Swing uses the model-view-controller architecture (MVC) as t
fundamental design behind each of its components

Essentially, MVC breaks GUI components into three elements. Ea
of these elements plays a crucial role in how the compone
behaves.

The Model-View-Controller is a well known software architectu
pattern ideal to implement user interfaces on computers
dividing an application intro three interconnected parts

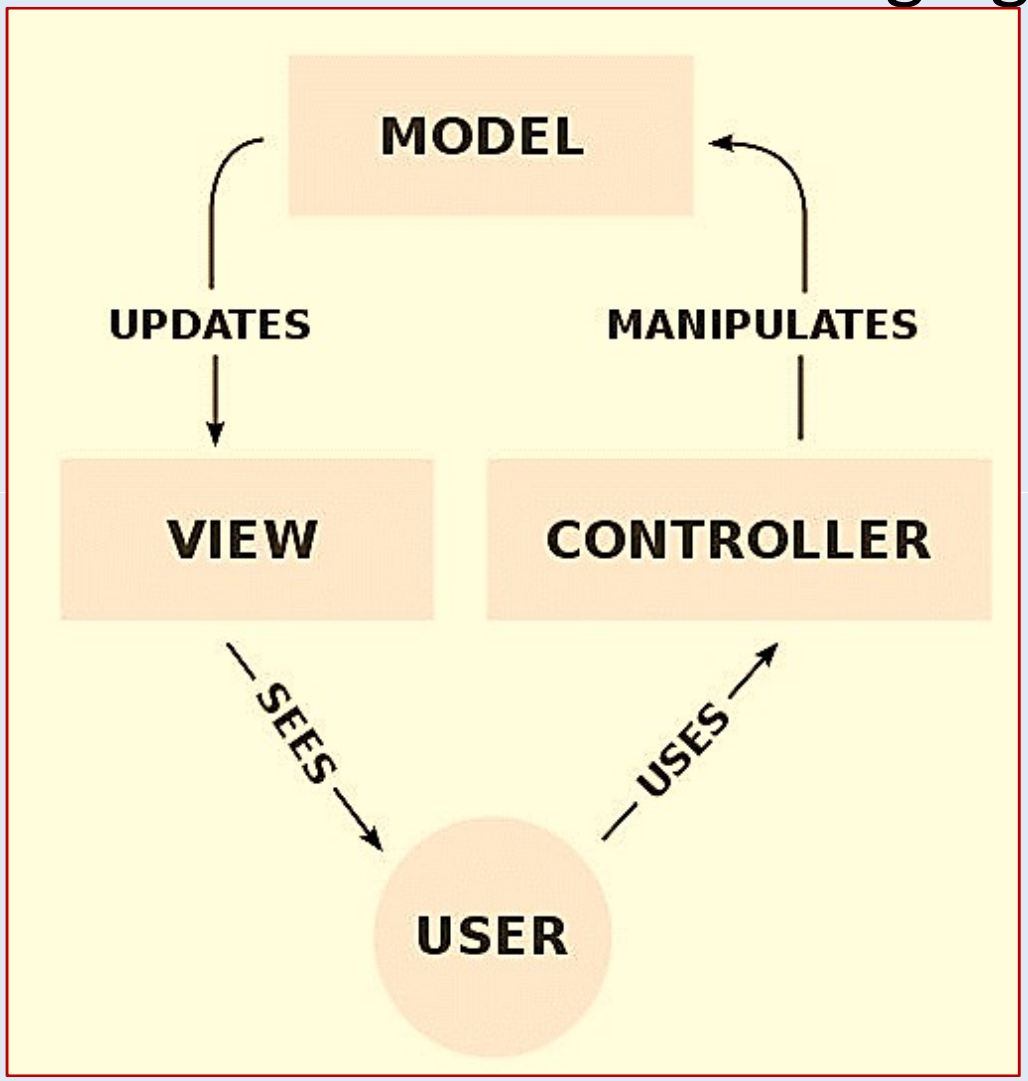
Main goal of Model-View-Controller, also known as MVC, is
separate internal representations of an application from the wa
nformation are presented to the user.

nitially, MVC was designed for desktop GUI applications but i
quickly become an extremely popular pattern for designing w
applications too.

MVC pattern has the three components :

- Model that manages data, logic and rules of the application
- View that is used to present data to user
- Controller that accepts input from the user and converts it
ommands for the Model or View.

the MVC pattern defines the interactions between these thr components like you can see in the following figure :



The Model receives commands and data from the Controller. stores these data and updates the View.

The View lets to present data provided by the Model to the user.

The Controller accepts inputs from the user and converts it commands for the Model or the View.

COMPONENTS & CONTAINERS

A component is an independent visual control, such as a pu button or slider.

A container holds a group of components. Thus, a container is special type of component that is designed to hold oth components.

Swing components inherit from the **javax.Swing.JComponent** clas which is the root of the Swing component hierarchy.

COMPONENTS

Swing components are derived from the JComponent class.

JComponent provides the functionality that is common to components. For example, JComponent supports the pluggab look and feel.

JComponent inherits the AWT classes Container and Componer Thus, a Swing component is built on and compatible with an AW component.

All of Swing's components are represented by classes defin within the package **javax.swing**.

The following table shows the class names for Swing components

JApplet
JColorChooser
JDialog
JFrame
JLayeredPane
JMenuItem
JPopupMenu
JRootPane
JSlider
JTable

- JTogglebutton
- JViewport
- JButton
- JComboBox
- JEditorPane
- JInternalFrame
- JList
- JOptionPane
- JProgressBar

Notice that all component classes begin with the letter J.

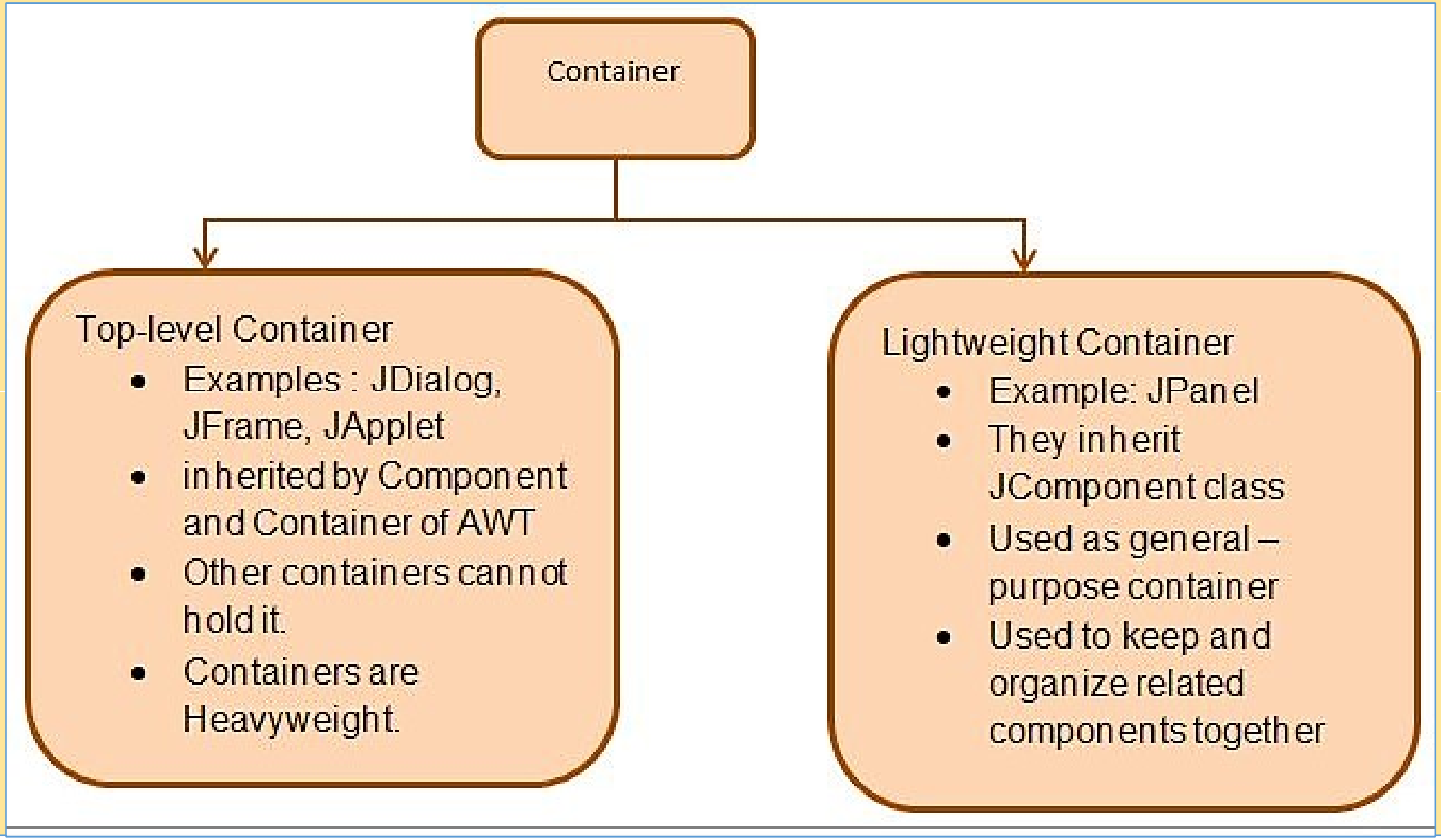
For example, the class for a label is JLabel; the class for a pu button is JButton; and the class for a scroll bar is JScrollBar

CONTAINERS

Swing defines two types of containers. The first are top-lev containers: **JFrame**, **JApplet**, **JWindow**, and **JDialog**. The containers do not inherit **JComponent**. They inherit the AW classes **Component** and **Container**.

The second type container are lightweight and the top-lev containers are heavyweight. This makes the top-level containers special case in the Swing component library.

Java, Containers are divided into two types as shown below:



llowing is the list of commonly used containers while designed UI using SWING.

Sr.No.	Container & Description
1	Panel JPanel is the simplest container. It provides space in which any other component can be placed, including other panels.
2	Frame A JFrame is a top-level window with a title and a border.
3	Window A JWindow object is a top-level window with no borders and no menubar.

wing Example : A window on the screen.

```
mport javax.swing.JFrame;
mport javax.swing.SwingUtilities;

ublic class Example extends JFrame {

    public Example() {
        setTitle("Simple example");
        setSize(300, 200);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        Example ex = new Example();
        ex.setVisible(true);
    }
```

Output

EVENT HANDLING IN SWINGS

The functionality of Event Handling is what is the further step if a action performed.

Java foundation introduced "Delegation Event Model" i.e describ how to generate and control the events.

The key elements of the Delegation Event Model are as source ar listeners.

The listener should have registered on source for the purpose alert notifications.

All GUI applications are event-driven

Java Swing event object

When something happens in the application, an event object created.

For example, when we click on the button or select an item from ist.

There are several types of events, including ActionEvent, TextEver FocusEvent, and ComponentEvent.

Each of them is created under specific conditions.

An event object holds information about an event that h occurred.

SWING LAYOUT MANAGERS

Layout refers to the arrangement of components within t container.

Layout is placing the components at a particular position within t container. The task of laying out the controls is done automatica by the Layout Manager.

The layout manager automatically positions all the componer within the container.

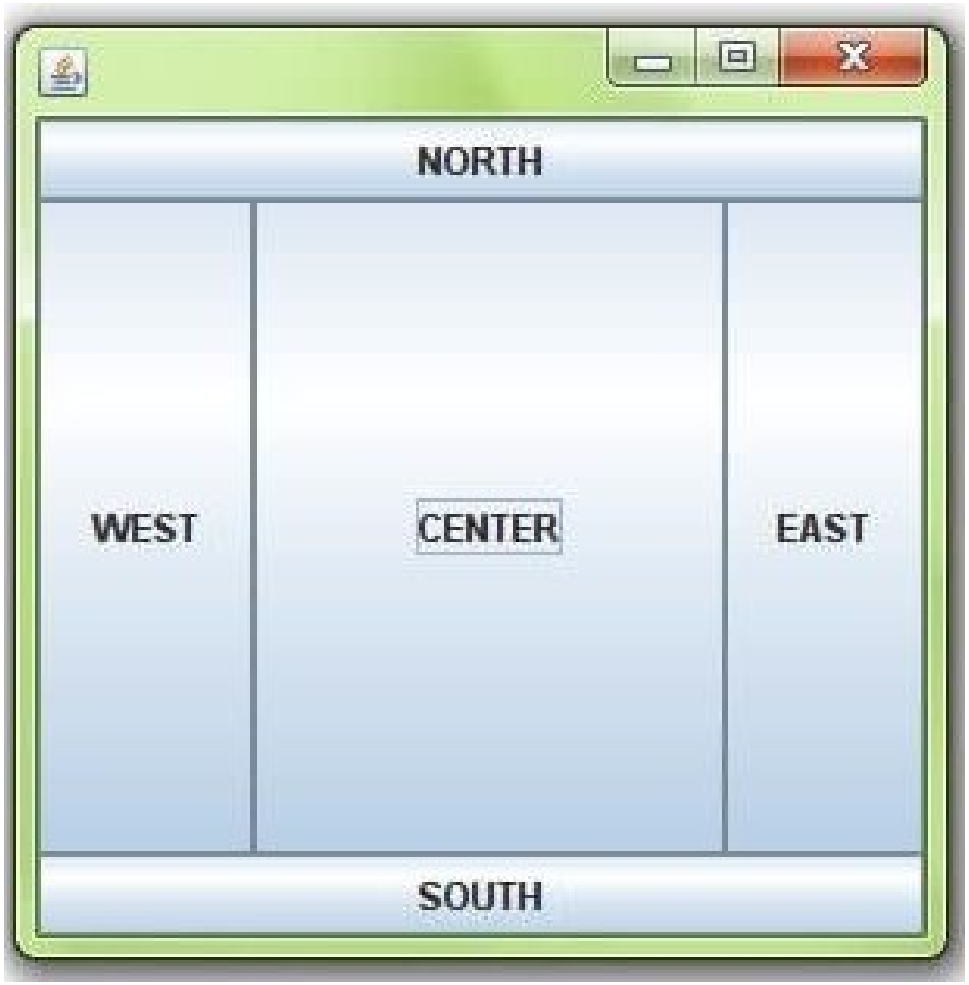
Even if you do not use the layout manager, the components a still positioned by the default layout manager. It is possible to l out the controls by hand, however, it becomes very difficult

Java provides various layout managers to position the contro
Properties like size, shape, and arrangement varies from one layo
manager to the other.

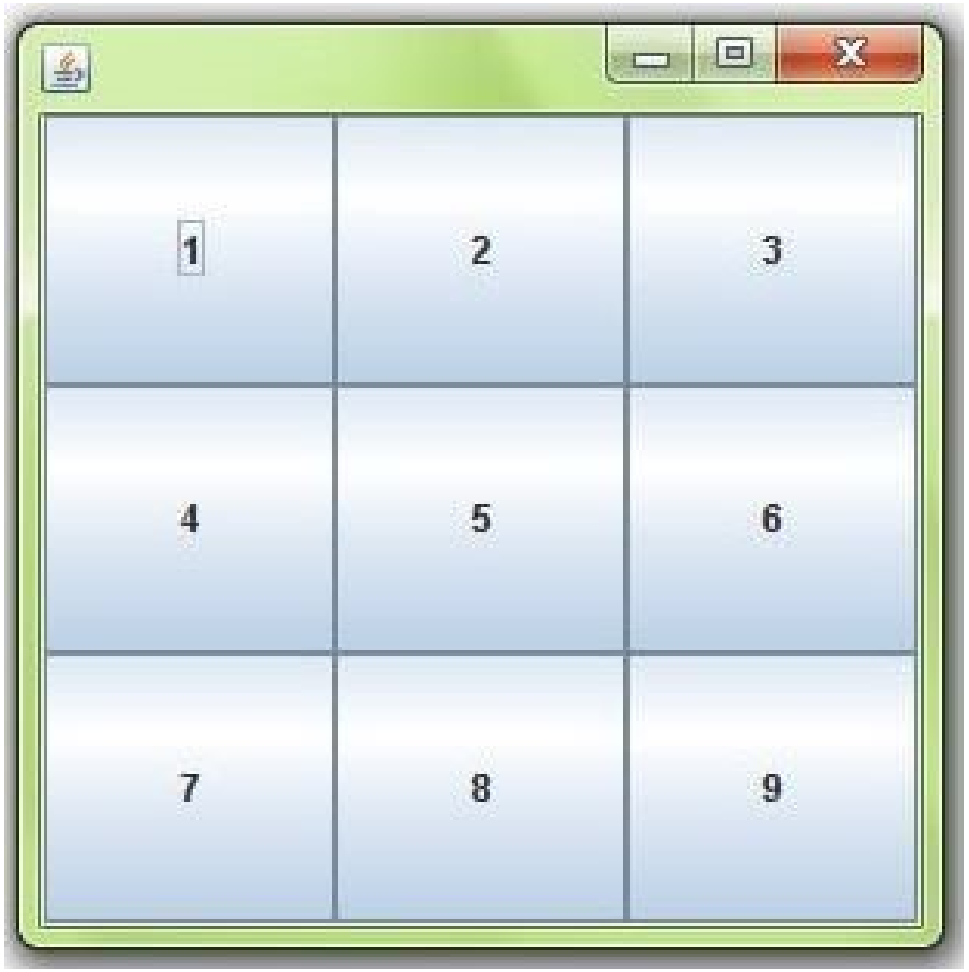
There are following classes that represents the layout managers:

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- javax.swing.SpringLayout etc.

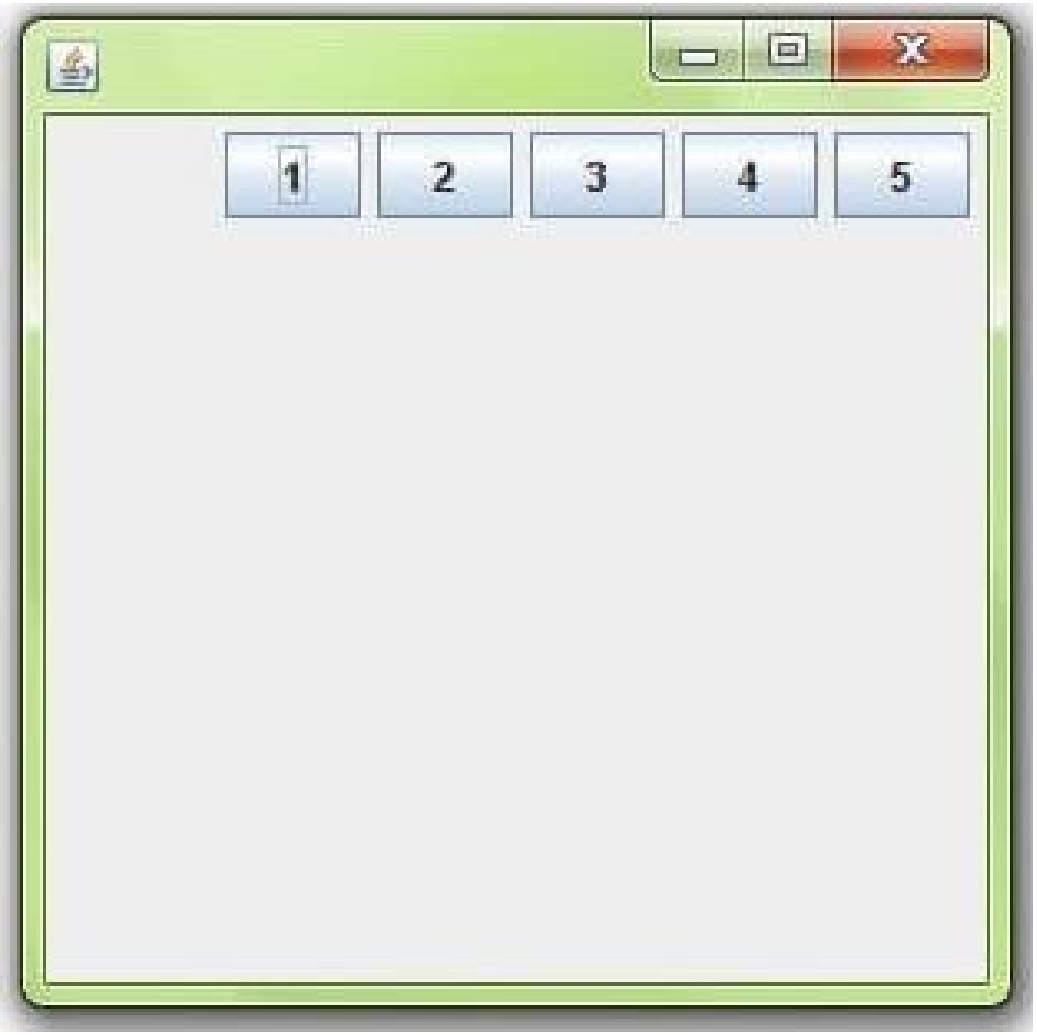
BorderLayout



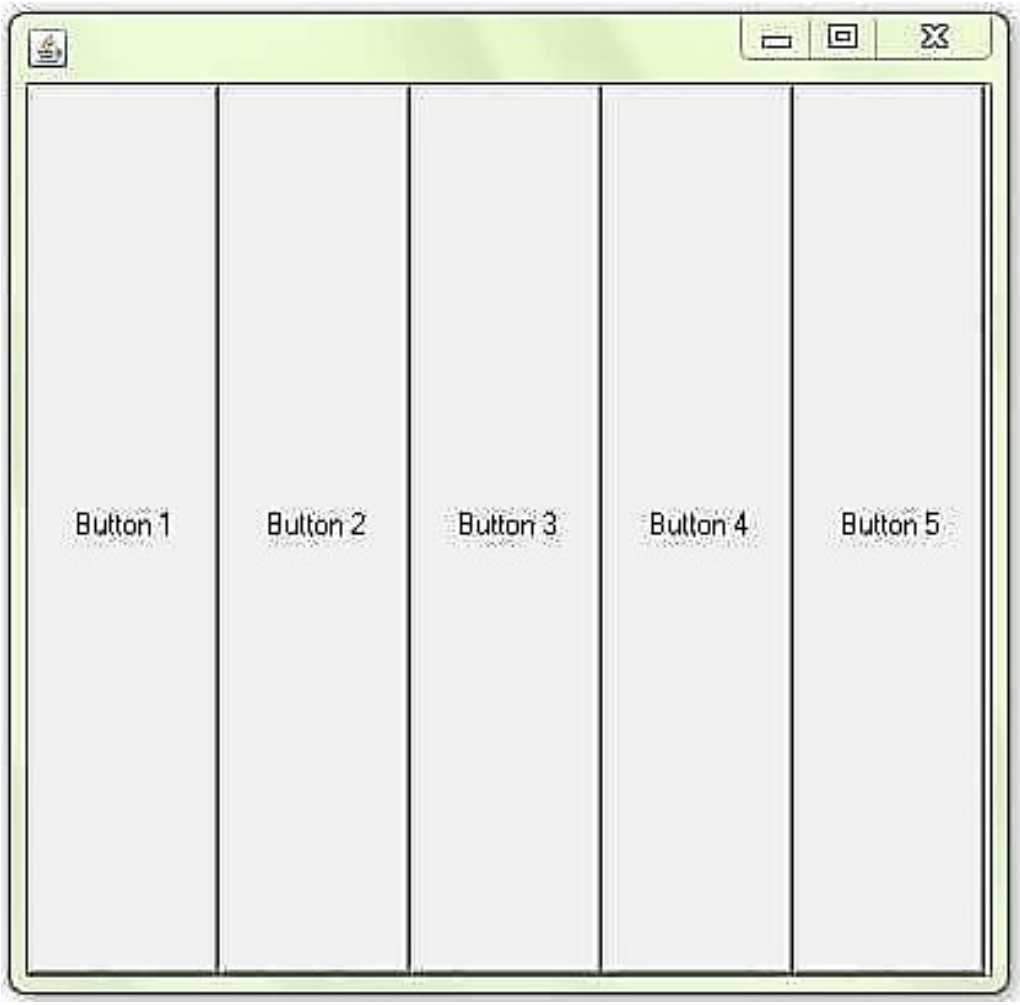
GridLayout



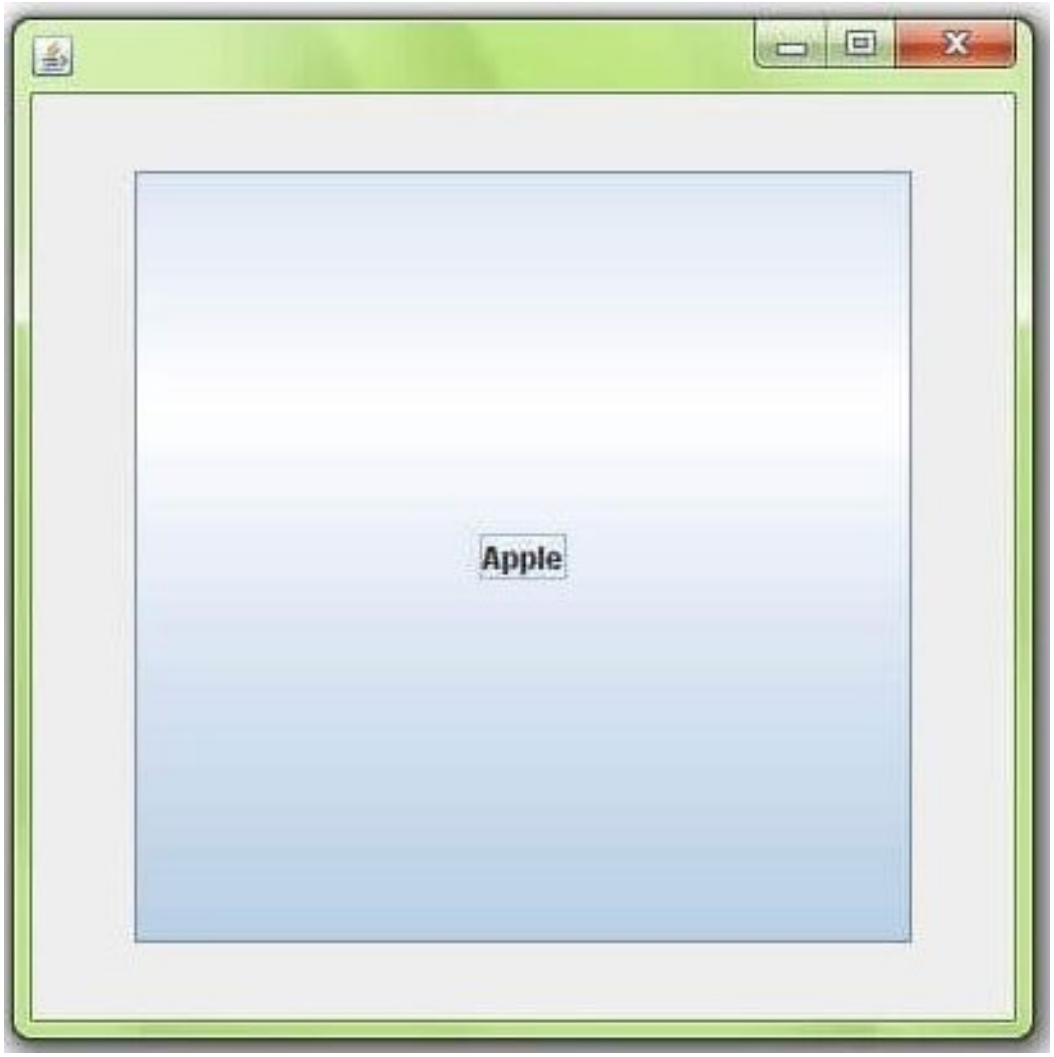
FlowLayout



BoxLayout



CardLayout



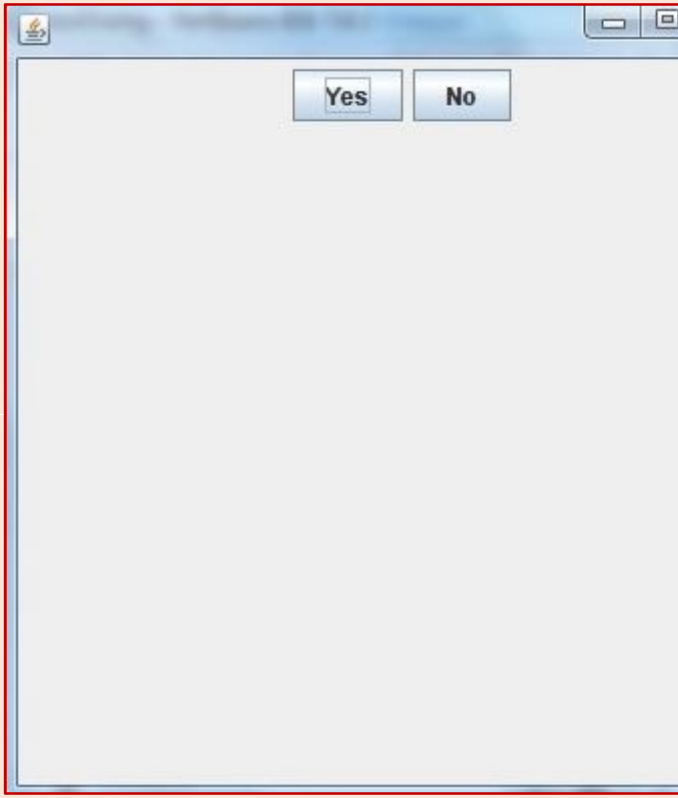
GroupLayout

xample of JButton

```
port javax.swing.*;
port java.awt.event.*;
port java.awt.*;
blic class testswing extends JFrame


testswing()
{
  JButton bt1 = new JButton("Yes");              //Creating a Yes Button.
  JButton bt2 = new JButton("No");               //Creating a No Button.
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)     //setting close operation.
  setLayout(new FlowLayout());             //setting layout using FlowLayout object
  setSize(400, 400);                       //setting size of Jframe
  add(bt1);              //adding Yes button to frame.
  add(bt2);              //adding No button to frame.

  setVisible(true);
}
public static void main(String[] args)
{
  new testswing();
}
```



xample of JTextField

```
mport javax.swing.*;
mport java.awt.event.*;
mport java.awt.*;
ublic class MyTextField extends JFrame

 public MyTextField()
 {
   JTextField jtf = new JTextField(20);  //creating JTextField.
   add(jtf);                             //adding JTextField to frame.
   setLayout(new FlowLayout());
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setSize(400, 400);
   setVisible(true);
 }
 public static void main(String[] args)
 {
   new MyTextField();
 }
```

xample of Jlabel - It is used for placing text in a box

```
mport javax.swing.*;
lass SLabelDemo1

ublic static void main(String args[])

    JFrame label_f= new JFrame("studytonight ==> Label Demo");
    JLabel label_l1,label_l2;
    label_l1=new JLabel("Welcome to studytonight.com");
    label_l1.setBounds(50,50, 200,30);
    label_l2=new JLabel("How are You?");
    label_l2.setBounds(50,100, 200,30);
    label_f.add(label_l1);
    label_f.add(label_l2);
    label_f.setSize(300,300);
    label_f.setLayout(null);
    label_f.setVisible(true);
    }
```

MODULE 5

CHAPTER 2

JDBC

Java DataBase Connectivity (JDBC)

JDBC stands for Java Database Connectivity, which is a standa
Java API for database-independent connectivity between the Ja
programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentione
below that are commonly associated with database usage.

- Making a connection to a database
- Creating SQL or MySQL statements
- Executing SQL or MySQL queries in the database
- Viewing & Modifying the resulting records

JDBC Architecture

JDBC Architecture consists of two layers

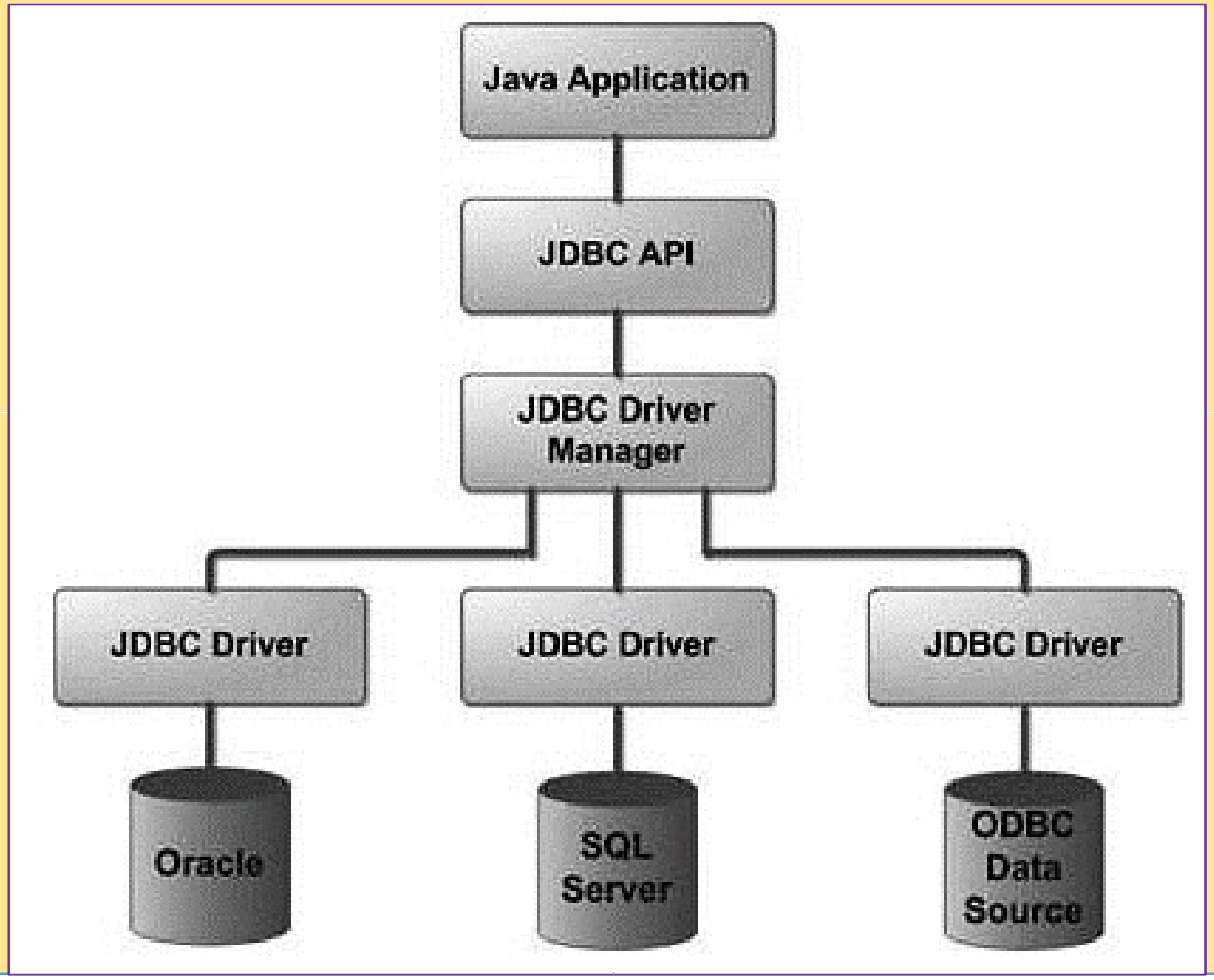
JDBC API: This provides the application-to-JDBC Manag
connection.

JDBC Driver API: This supports the JDBC Manager-to-Driv
Connection.

The JDBC API uses a driver manager and database-specific drive
to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used
access each data source.

Following is the architectural diagram, which shows the location
the driver manager with respect to the JDBC drivers and the Ja
application



Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with t database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

The forName() method is used to register the driver class.

The getConnection() method of DriverManager class is used establish connection with the database

The createStatement() method of Connection interface is used create statement. The object of statement is responsible to execu queries with the database.

The executeQuery() method of Statement interface is used execute queries to the database. This method returns the object ResultSet that can be used to get all the records of a table.

By closing connection object statement and ResultSet will closed automatically. The close() method of Connection interface used to close the connection.

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need follow 5 following steps.

In this example we are using MySql as the database. So we need know following informations for the mysql database:

Driver class: The driver class for the mysql database com.mysql.jdbc.Driver.

Connection URL: The connection URL for the mysql database jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql the database, localhost is the server name on which mysql running, we may also use IP address, 3306 is the port numb and sonoo is the database name. We may use any database, such case, we need to replace the sonoo with our databa name.

Username: The default username for the mysql database is root.

Password: It is the password given by the user at the time of talling the mysql database. In this example, we are going to use ot as the password.

et's first create a table in the mysql database, but before creati able, we need to create database first.

```
create database sonoo;

use sonoo;

create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql databas

```
port java.sql.*;
ss MysqlCon{
olic static void main(String args[]){
{
ss.forName("com.mysql.jdbc.Driver");
nnection con=DriverManager.getConnection(
oc:mysql://localhost:3306/sonoo","root","root");
ere sonoo is database name, root is username and password
tement stmt=con.createStatement();
ultSet rs=stmt.executeQuery("select * from emp");
ile(rs.next())
tem.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
.close();
tch(Exception e){ System.out.println(e);}
```

This example will fetch all the records emp table.

Creating a sample MySQL database

```
eate database SampleDB;
se SampleDB;
REATE TABLE `users` (
 `user_id` int(11) NOT NULL AUTO_INCREMENT,
 `username` varchar(45) NOT NULL,
 `password` varchar(45) NOT NULL,
 `fullname` varchar(45) NOT NULL,
 `email` varchar(45) NOT NULL,
 PRIMARY KEY (`user_id`)
```

Connecting to the database

```
ring dbURL = "jdbc:mysql://localhost:3306/sampledb";
ring username = "root";
ring password = "secret";
y {
    Connection conn = DriverManager.getConnection(dbURL, usernam
ssword);
    if (conn != null) {
    System.out.println("Connected");
}
catch (SQLException ex)

ex.printStackTrace();
```

Once the connection was established, we have a Connection obje
which can be used to create statements in order to execute SQL querie

n the above code, we have to close the connection explicitly after fini
working with the database:

```
conn.close();
```

- **INSERT Statement Example**

et's write code to insert a new record into the table Users wi
llowing details:

username: bill

password: secretpass

fullname: Bill Gates

email: bill.gates@microsoft.com

```
tring sql = "INSERT INTO Users (username, password, fullnam
mail) VALUES (?, ?, ?, ?)";
PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");
statement.setString(2, "secretpass");
statement.setString(3, "Bill Gates");
statement.setString(4, "bill.gates@microsoft.com");
nt rowsInserted = statement.executeUpdate();
f (rowsInserted > 0) {
    System.out.println("A new user was inserted successfully!");
}
```

SELECT Statement Example

```
String sql = "SELECT * FROM Users";

Statement statement = conn.createStatement();
ResultSet result = statement.executeQuery(sql);

int count = 0;

while (result.next()){
    String name = result.getString(2);
    String pass = result.getString(3);
    String fullname = result.getString("fullname");
    String email = result.getString("email");

    String output = "User #%d: %s - %s - %s - %s";
    System.out.println(String.format(output, ++count, name, pass, fullname, email));
}
```

utput

User #1: bill - secretpass - Bill Gates - bill.gates@microsoft.com

UPDATE Statement Example

```
String sql = "UPDATE Users SET password=?, fullname=?, email=? WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "123456789");
statement.setString(2, "William Henry Bill Gates");
statement.setString(3, "bill.gates@microsoft.com");
statement.setString(4, "bill");

int rowsUpdated = statement.executeUpdate();
if (rowsUpdated > 0) {
    System.out.println("An existing user was updated successfully!");
}
```

DELETE Statement Example

The following code snippet will delete a record whose username field contains “bill”

```
String sql = "DELETE FROM Users WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");

int rowsDeleted = statement.executeUpdate();
if (rowsDeleted > 0) {
    System.out.println("A user was deleted successfully!");
}
```