MODULE V
Graphical User Interface and Database support of Java:

Syllabus:

- Swings fundamentals - Swing Key Features, Model View Controller (MVC), Swing Controls, Components and Containers, Swing Packages, Event Handling in Swings, Swing Layout Managers, Exploring Swings –JFrame, JLabel, The Swing Buttons, JTextField.
- Java DataBase Connectivity (JDBC) - JDBC overview, Creating and Executing Queries – create table, delete, insert, select.

Swings

- The Swing-related classes are contained in **javax.swing**.
- Swing is a set of classes that provides more powerful and flexible GUI components than are possible with the AWT.
- All components have more capabilities in Swing.

  Example: A button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.
- Swing library is an official Java GUI tool kit released by Sun Microsystems
- Java Swing provides platform-independent and lightweight components

- Java Swing is a part of Java Foundation Classes(JFC) that is used to create window-based applications.
- It is built on the top of AWT and entirely written in java.

JFC

- The Java Foundation Classes(JFC) are a set of GUI components which simplify the development of desktop applications.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Features of Swing (Advantages of Swings over AWT)

Platform Independent

- It is platform independent, the swing component that are used to build the programs are not platform specific.
- The behavior and appearance of Swing components is consistent across platforms, whereas AWT components will differ from platform to platform. Thus Swing is platform independent whereas AWT is platform dependent.

Swing Components Are Lightweight

- This means that they are written entirely in Java and do not map directly to platform-specific peers.

Swing Supports a Pluggable Look and Feel

- SWING based GUI Application look and feel can be changed at run-time, based on available values.

Swing uses MVC Architecture

- Java's Swing components have been implemented using the **model-view controller** (**MVC**) model.
  - *Any Swing component can be viewed in terms of three independent aspects: what state it is in (its* ***model****), how it looks (its* ***view****), and what it does (its* ***controller****). Suppose the user clicks on a button. This action is detected by the controller. The controller tells the model to change into the pressed state. The model in turn generates an event that is passed to the view. The event tells the view that the button needs to be redrawn to reflect its change in state.*
- High level of separation between view and controller is not beneficial for Swing components. Instead Swing uses a **modified version of MVC** that combines the view and controller into a single entuity called UI(User Interface) delegate. For this reason ,Swing's approach is called either the **Model-Delegate** architecture or **Separable Model** architecture.

Rich Controls

- Swing is the latest GUI toolkit, and provides a richer set of interface components than the AWT.

Customizable

- Swing components can be given their own "look and feel“. Example: A button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

Manageable

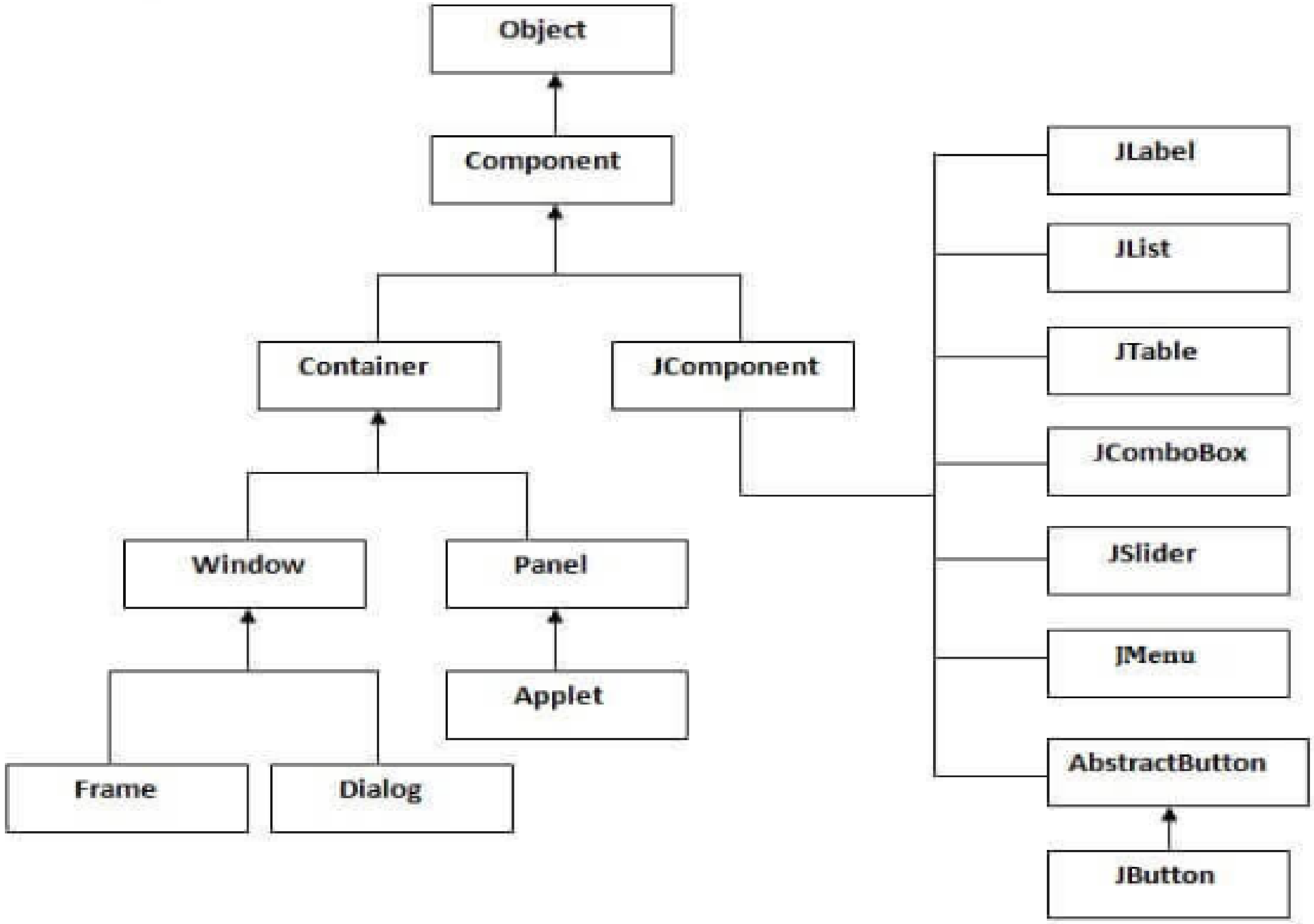
- It is easy to manage and configure. Its mechanism and composition pattern allows changing the settings at run time as well. The uniform changes can be provided to the user interface without doing any changes to application code.

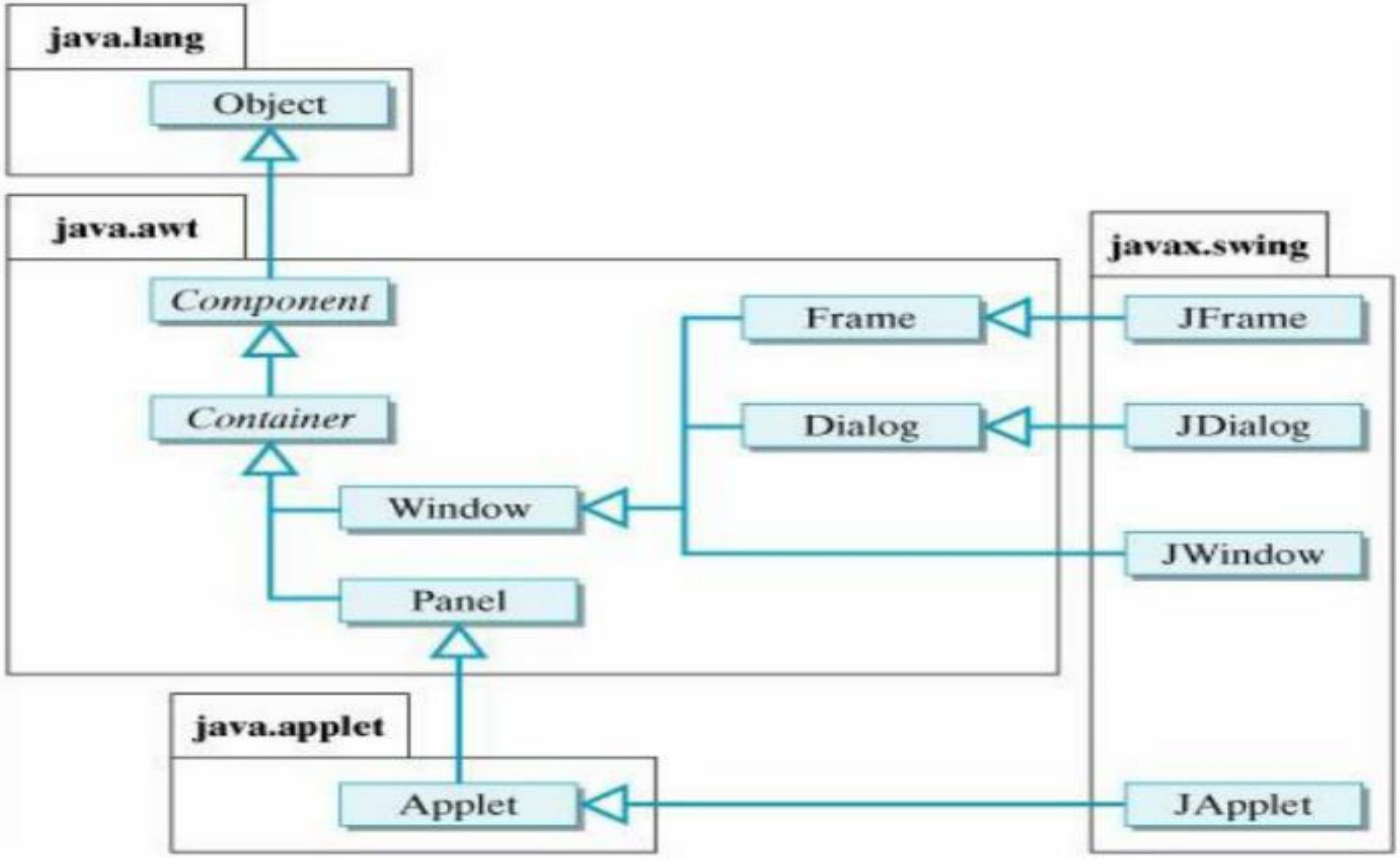
Difference between AWT and Swing

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent.**	Java swing components are **platform-independent.**
2)	AWT components are **heavyweight.**	Swing components are **lightweight.**
3)	AWT **doesn't support pluggable look and feel.**	Swing **supports pluggable look and feel.**
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	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 hierarchy of java swing API is given below.





Java AWT, Swing and Applet Components Hierarchy

COMPONENTS & CONTAINERS

- A component is an independent visual control, such as a push button or slider.
- A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components.
- Swing components inherit from the javax.Swing.JComponent class, which is the root of the Swing component hierarchy.

Swing Components and Containers

Swing Components

- Swing components are derived from the **JComponent** class.
- **JComponent** provides the functionality that is common to all components.
- For example, **JComponent** supports the pluggable look and feel.
- **JComponent** inherits the AWT classes Container and Component.
- Thus a Swing component is built on and compatible with an AWT component.
- All of Swing's components are represented by classes defined within the package **javax.swing**.

Class names for Swing components(including those used as containers)

JApplet
JColorChooser
JDialog
JFrame
JLayeredPane
JMenuItem
JPopupMenu
JRootPane
JList
JProgressBar

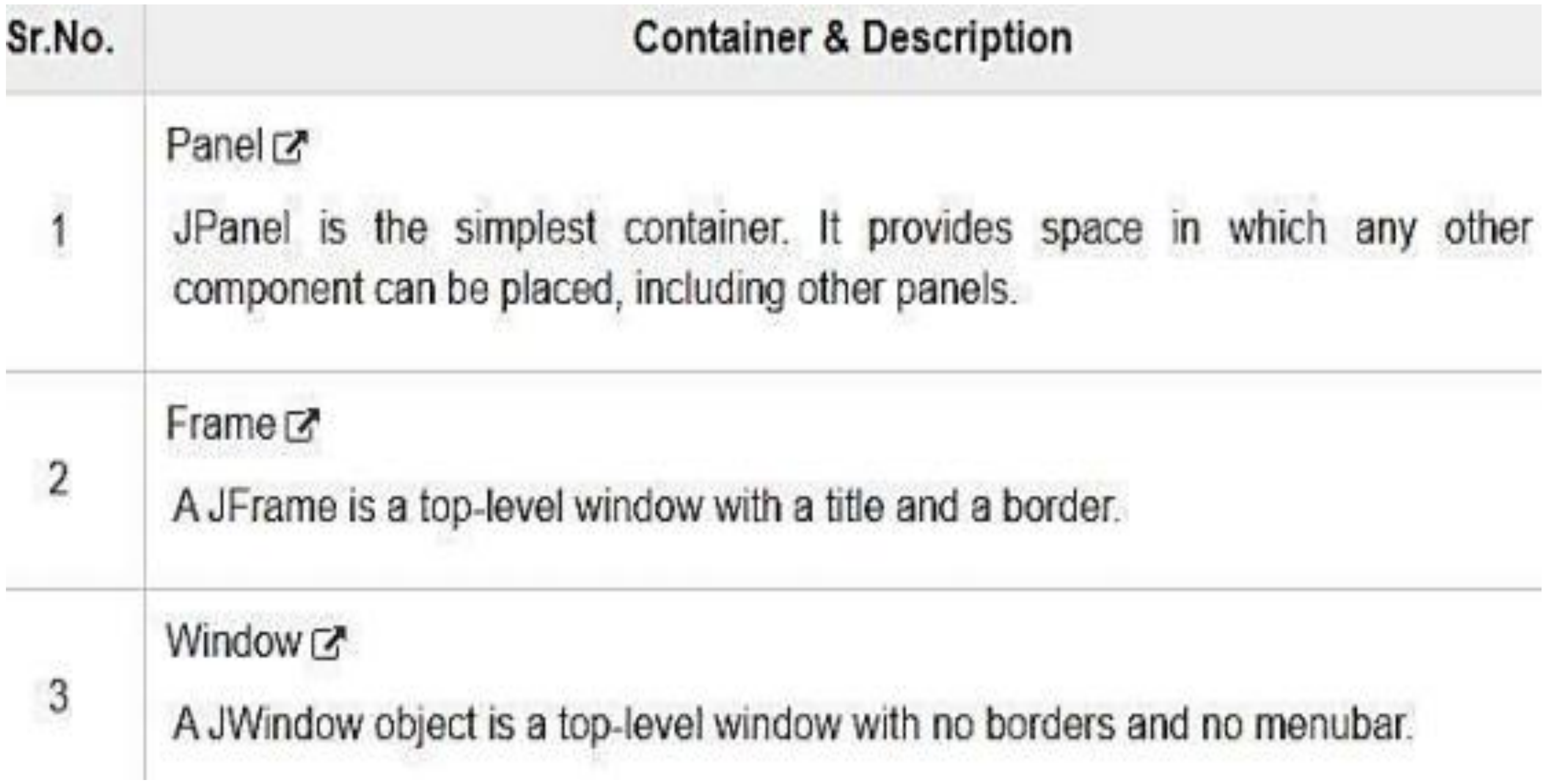
JSlider
JTable
JTogglebutton
JViewport
JButton
JComboBox
JEditorPane
JInternalFrame
JOptionPane

Swing Conainers

- Swing defines two types of containers.
- The first are top-level containers: **JFrame**, **JApplet**, **JWindow**, and **JDialog**.
- They do not inherit **JComponent**. However, they inherit the AWT classes **Component** and **Container**.
- Unlike Swing's other components, which are lightweight, the top level containers are heavyweight.
- The one most commonly used for application is **JFrame** and the one used for Applet is **JApplet**.

- The second type of containers supported by Swing are lightweight containers.
- They inherit the **JComponent**.
- An example of lightweight container is **JPanel**.
- Lightweight container can be contained within another container.

- Following is the list of commonly used containers while designed GUI using SWING.



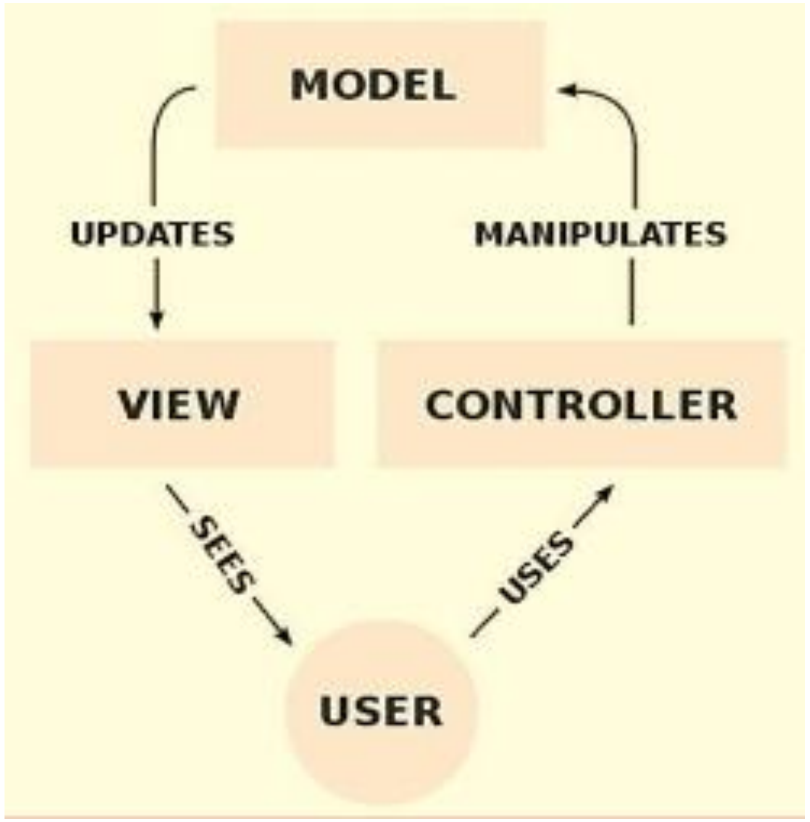
Sr.No.	Container & Description
1	Panel JPanel is the simplest container. It provides space in which any other component can be placed, including other panels.
2	Frame A JFrame is a top-level window with a title and a border.
3	Window A JWindow object is a top-level window with no borders and no menubar.

The Model-View-Controller Architecture

- Swing uses the model-view-controller architecture (MVC) as the fundamental design behind each of its components
- Essentially, MVC breaks GUI components into three elements. Each of these elements plays a crucial role in how the component behaves.
- The Model-View-Controller is a well known software architectural pattern ideal to implement user interfaces on computers by dividing an application intro three interconnected parts

- Main goal of Model-View-Controller, also known as MVC, is to separate internal representations of an application from the ways information are presented to the user.
- Initially, MVC was designed for desktop GUI applications but it's quickly become an extremely popular pattern for designing web 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 to commands for the Model or View.

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



- The Model receives commands and data from the Controller. It 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 to commands for the Model or the View.

EVENT HANDLING IN SWINGS

- The functionality of Event Handling is what is the further step if an action performed.
- Java foundation introduced “Delegation Event Model” i.e describes how to generate and control the events.
- The key elements of the Delegation Event Model are as source and listeners.
- The listener should have registered on source for the purpose of alert notifications.
- All GUI applications are event-driven

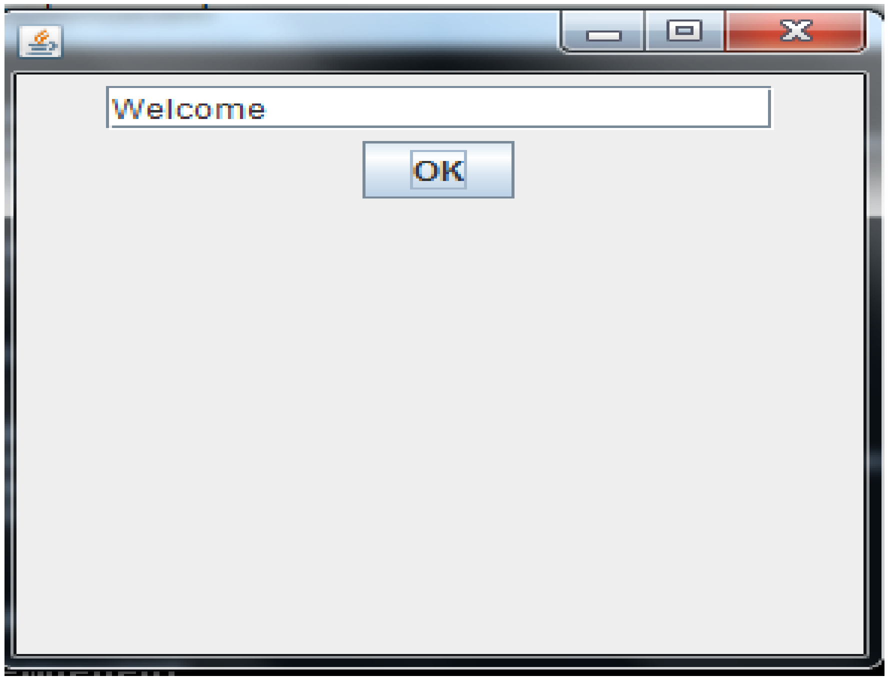
Java Swing event object

- When something happens in the application, an event object is created.
- For example, when we click on the button or select an item from a list.
- There are several types of events, including ActionEvent, TextEvent, FocusEvent, and ComponentEvent.
- Each of them is created under specific conditions.
- An event object holds information about an event that has occurred.

Example

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Swingdemoevent implements ActionListener {
JTextField t=new JTextField(20);
JButton b=new JButton("OK");
public Swingdemoevent()    {
    JFrame f=new JFrame();
    f.add(t);
    f.add(b);
    f.setSize(300,300);
    f.setLayout(new FlowLayout());
    f.setVisible(true);
    b.addActionListener(this);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);    }
public void actionPerformed(ActionEvent e)
{
t.setText("Welcome");
}
public static void main(String args[])  {
Swingdemoevent s=new Swingdemoevent();
}    }
```

Output


Welcome

SWING LAYOUT MANAGERS

- Layout refers to the arrangement of components within the container.
- Layout is placing the components at a particular position within the container. The task of laying out the controls is done automatically by the Layout Manager.
- The layout manager automatically positions all the components within the container.
- Even if you do not use the layout manager, the components are still positioned by the default layout manager. It is possible to lay out the controls by hand, however, it becomes very difficult

- Java provides various layout managers to position the controls. Properties like size, shape, and arrangement varies from one layout 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

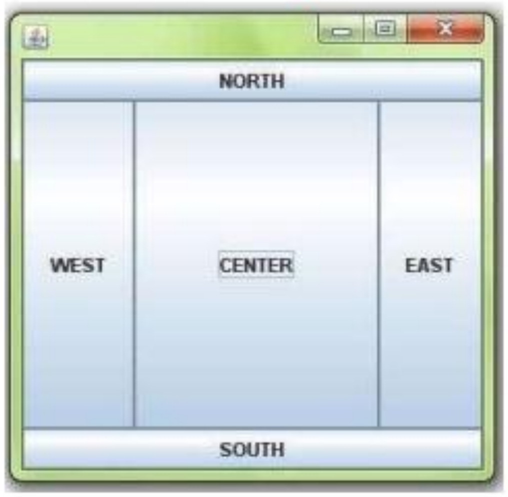
  javax.swing.BoxLayout

  javax.swing.GroupLayout

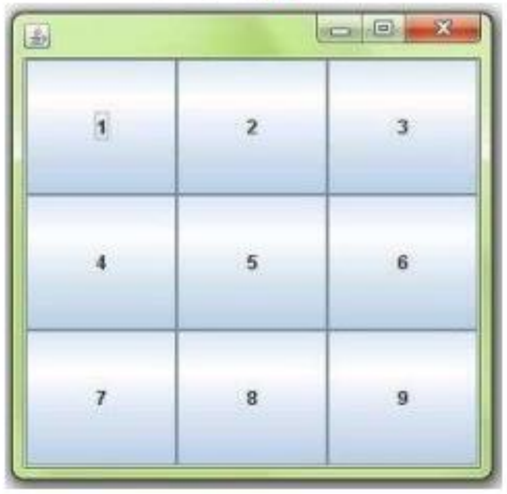
  javax.swing.SpringLayout etc.

- A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area.
- GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

BorderLayout

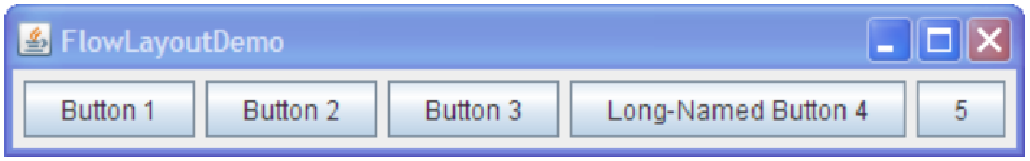


GridLayout



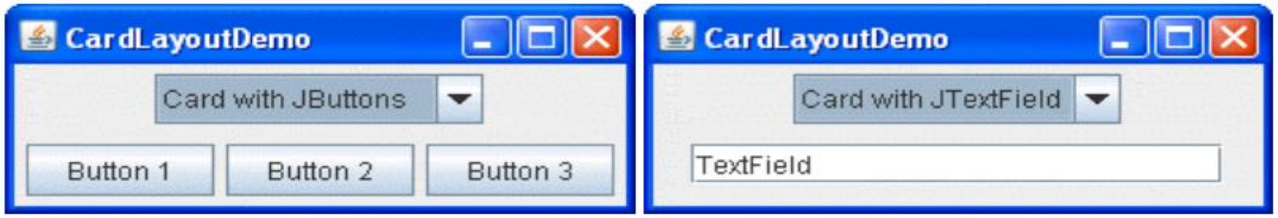
- FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide.

FlowLayout



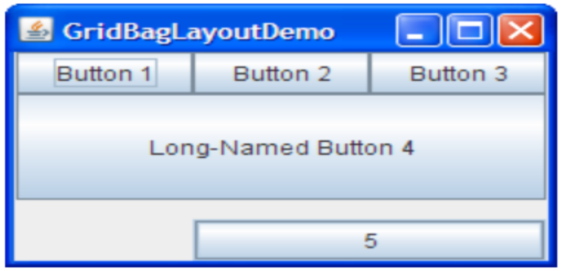
- The CardLayout class lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays.

CardLayout



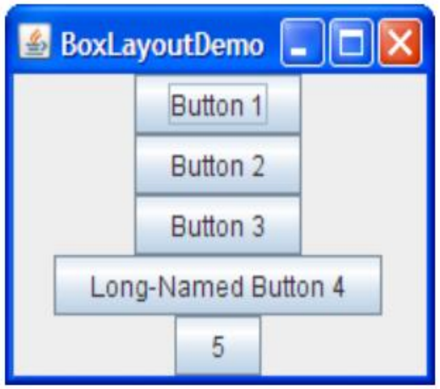
- GridBagLayout is a sophisticated, flexible layout manager. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. The rows in the grid can have different heights, and grid columns can have different widths.

GridBagLayout



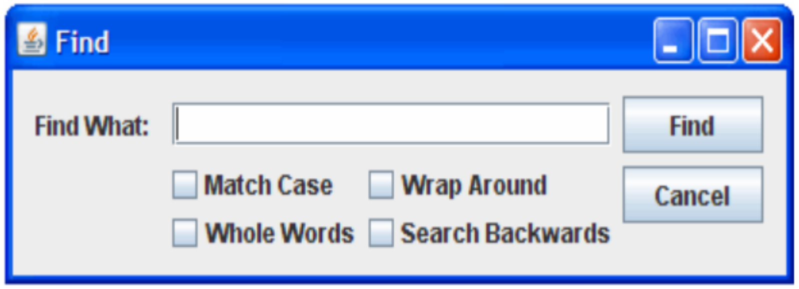
- The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components.

BoxLayout



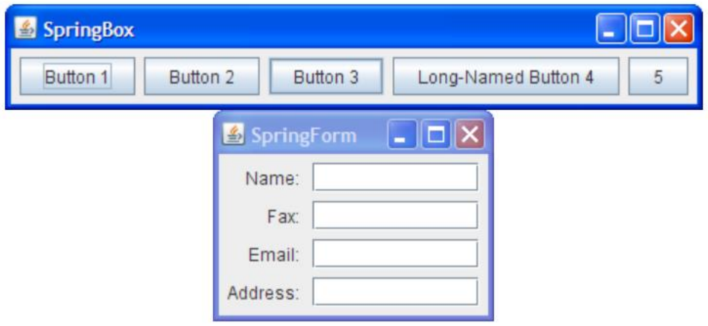
- GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently.

GroupLayout



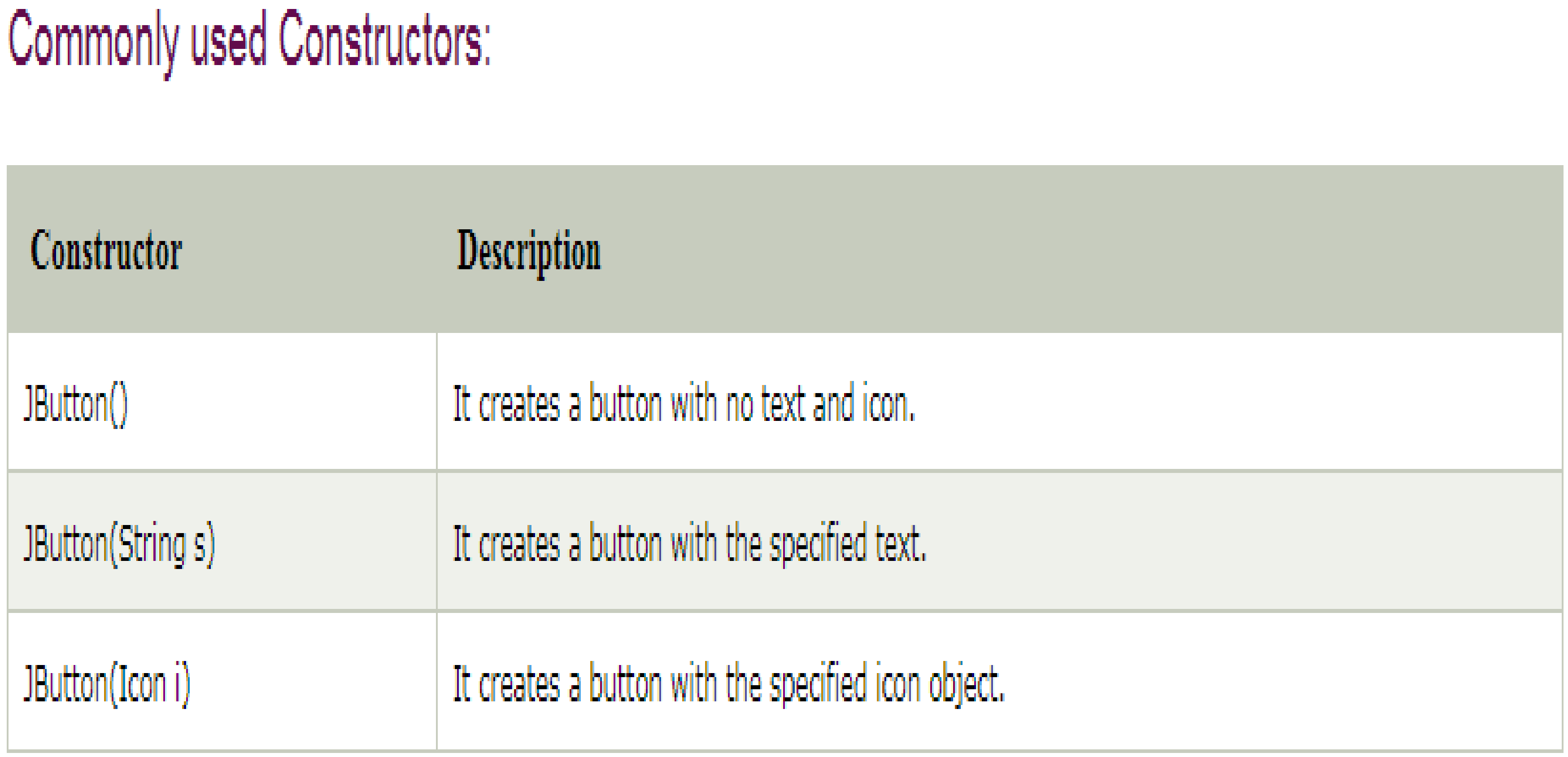
- SpringLayout lays out the children of its associated container according to a set of constraints

SpringLayout



JButton

- JButton extends AbstractButton class



Commonly used Constructors:

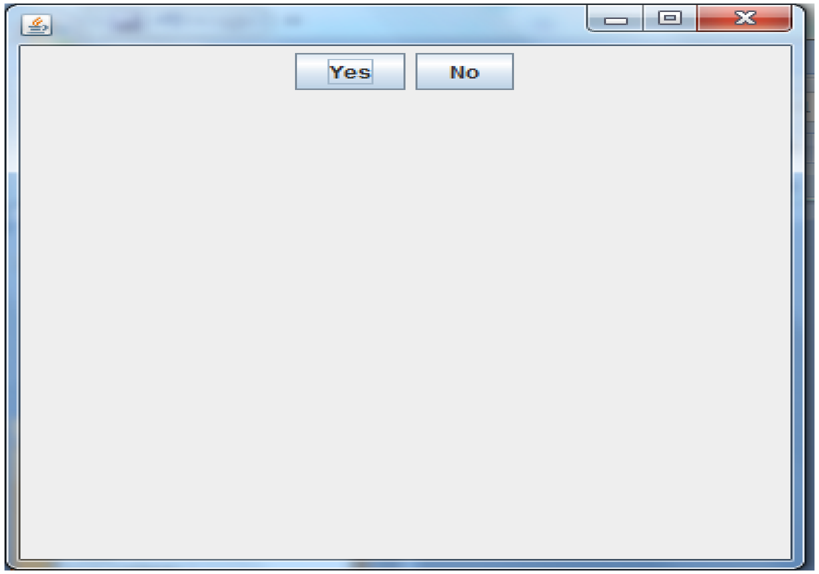
Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

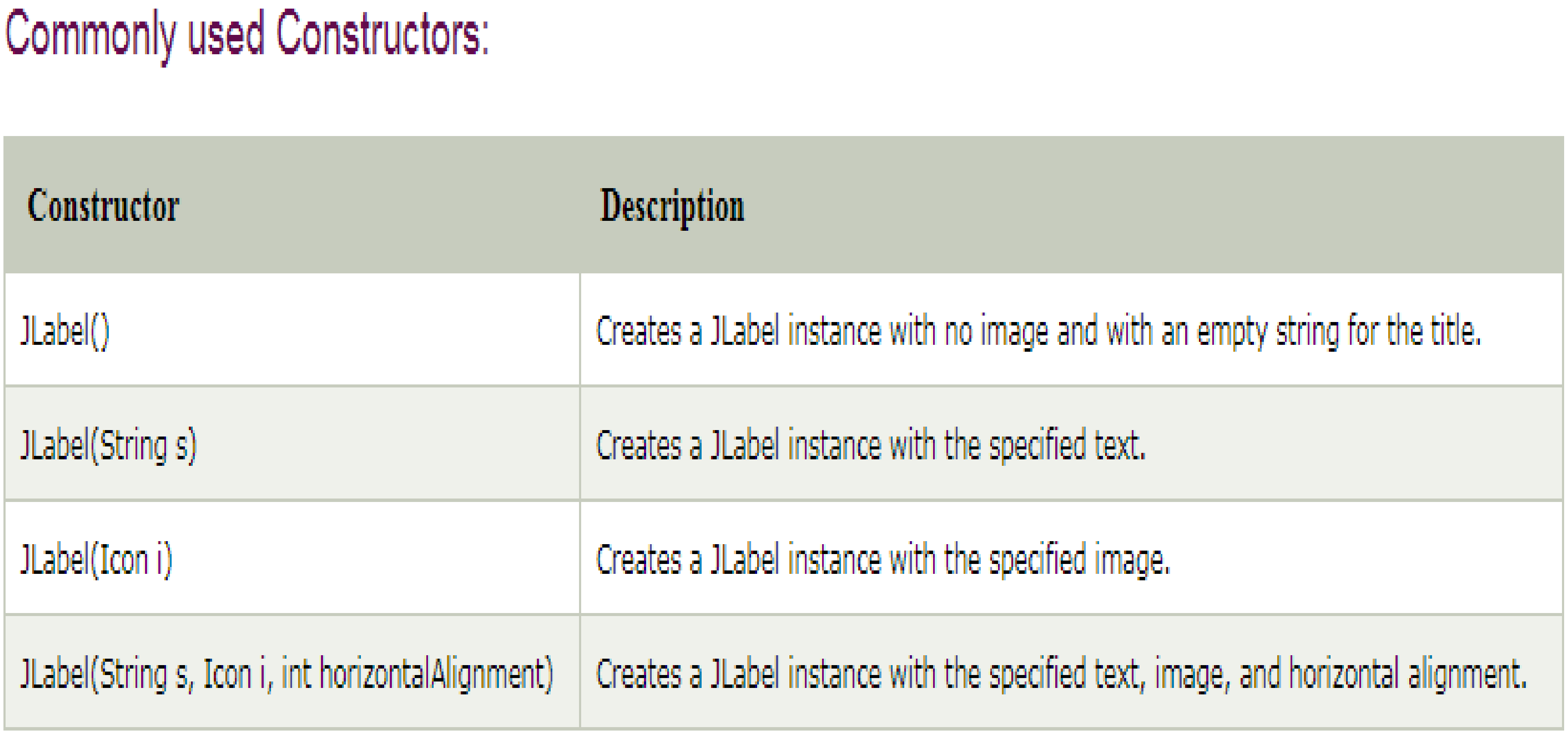
- **void setText(String s):** It is used to set specified text on button
- **String getText():** It is used to return the text of the button.
- **void setEnabled(boolean b):** It is used to enable or disable the button.
- **void setIcon(Icon b):** It is used to set the specified Icon on the button.
- **Icon getIcon():** It is used to get the Icon of the button.
- **void addActionListener(ActionListener a):** It is used to add the action listener to this object.

Example (JButton)

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class testswing extends JFrame{
 testswing(){
  JButton bt1 = new JButton("Yes");
  JButton bt2 = new JButton("No");
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setLayout(new FlowLayout());
  setSize(400, 400);
  add(bt1);
  add(bt2);
  setVisible(true);       }
 public static void main(String[] args)   {
  new testswing();        }    }
```



Jlabel



Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

- Here Icon is abstract class that cannot be instantiated. ImageIcon is a class that extends Icon. So to load images the following statement can be used:

  ImageIcon ic=new ImageIcon(“filename”);

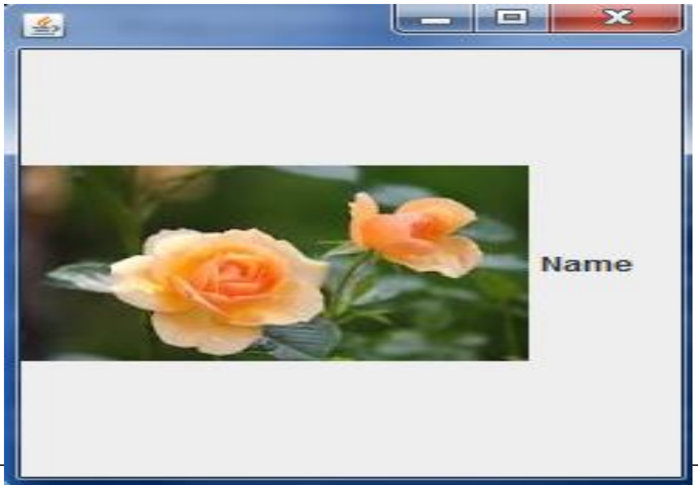
  where filename is a string quantity.

Commonly used Methods of JLabel class:

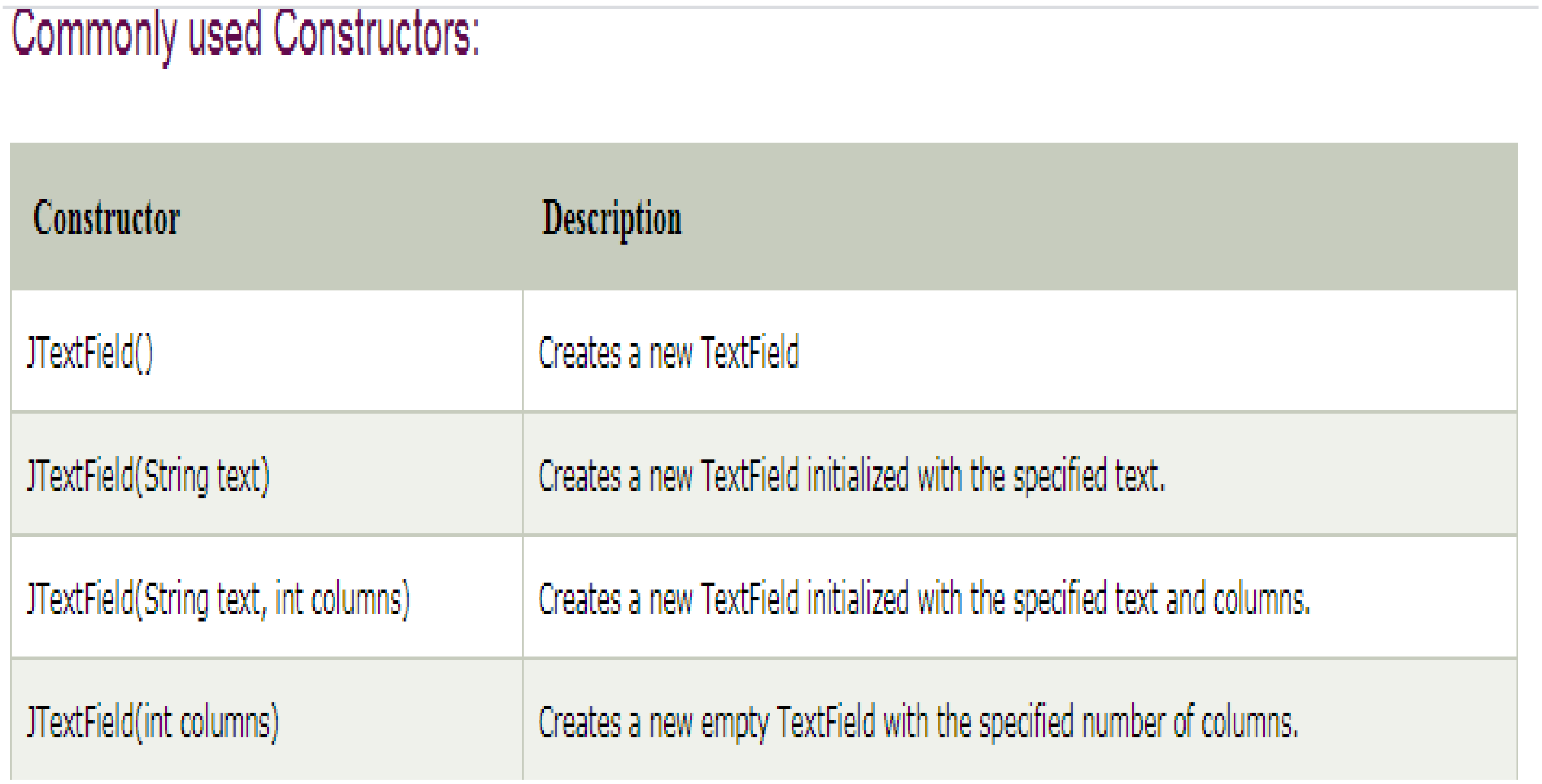
- **String getText():** It returns the text string that a label displays.
- **void setText(String text):** It defines the single line of text this component will display.
- **void setHorizontalAlignment(int alignment):** It sets the alignment of the label's contents along the X axis.
- **Icon getIcon():** It returns the graphic image that the label displays.
- **int getHorizontalAlignment():** It returns the alignment of the label's contents along the X axis.

Example (JLabel)

```
import javax.swing.*;
public class SimpleLabel extends JFrame
{   SimpleLabel()
    {  ImageIcon ic=new ImageIcon("download.jpg");
       JLabel jl=new JLabel("Name",ic,JLabel.LEFT);
       setSize(250,300);
       setVisible(true);
        add(jl);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
         new SimpleLabel();
    }
}
```



JTextField



Commonly used Constructors:

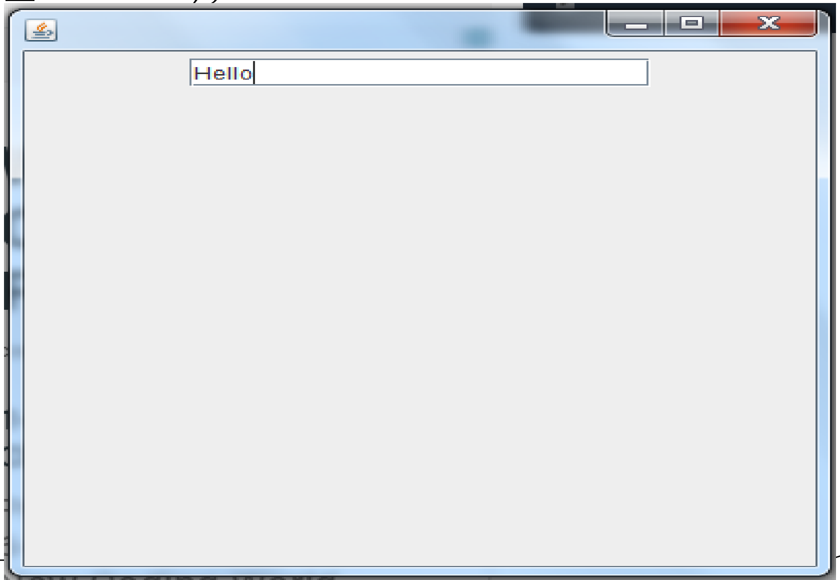
Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods of JTextField class:

- **void addActionListener(ActionListener l):** It is used to add the specified action listener to receive action events from this textfield.
- **void setFont(Font f):** It is used to set the current font.
- **void removeActionListener(ActionListener l):** It is used to remove the specified action listener so that it no longer receives action events from this textfield.

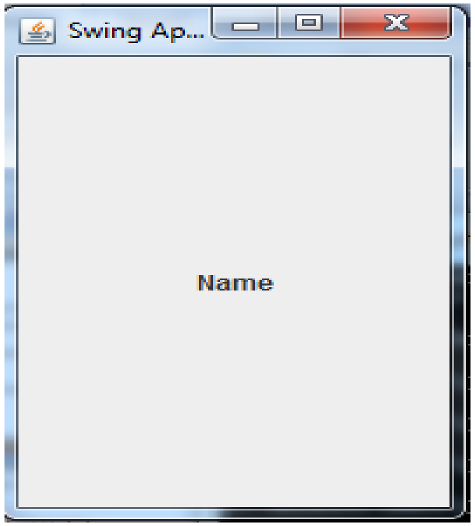
Example (JTextField)

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public 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();
 }     }
```



Example (JFrame and JLabel)

```
import javax.swing.*;
class swingdemo
{   swingdemo()
    {
       JFrame jf=new JFrame("Swing Appl");
       jf.setSize(200,300);
       JLabel jl=new JLabel("Name",JLabel.CENTER);
       jf.add(jl);
       jf.setVisible(true);
       jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //closing window will end the application
    }
}
public class Main
{   public static void main(String args[])
    {
       new swingdemo();
    }
}
```



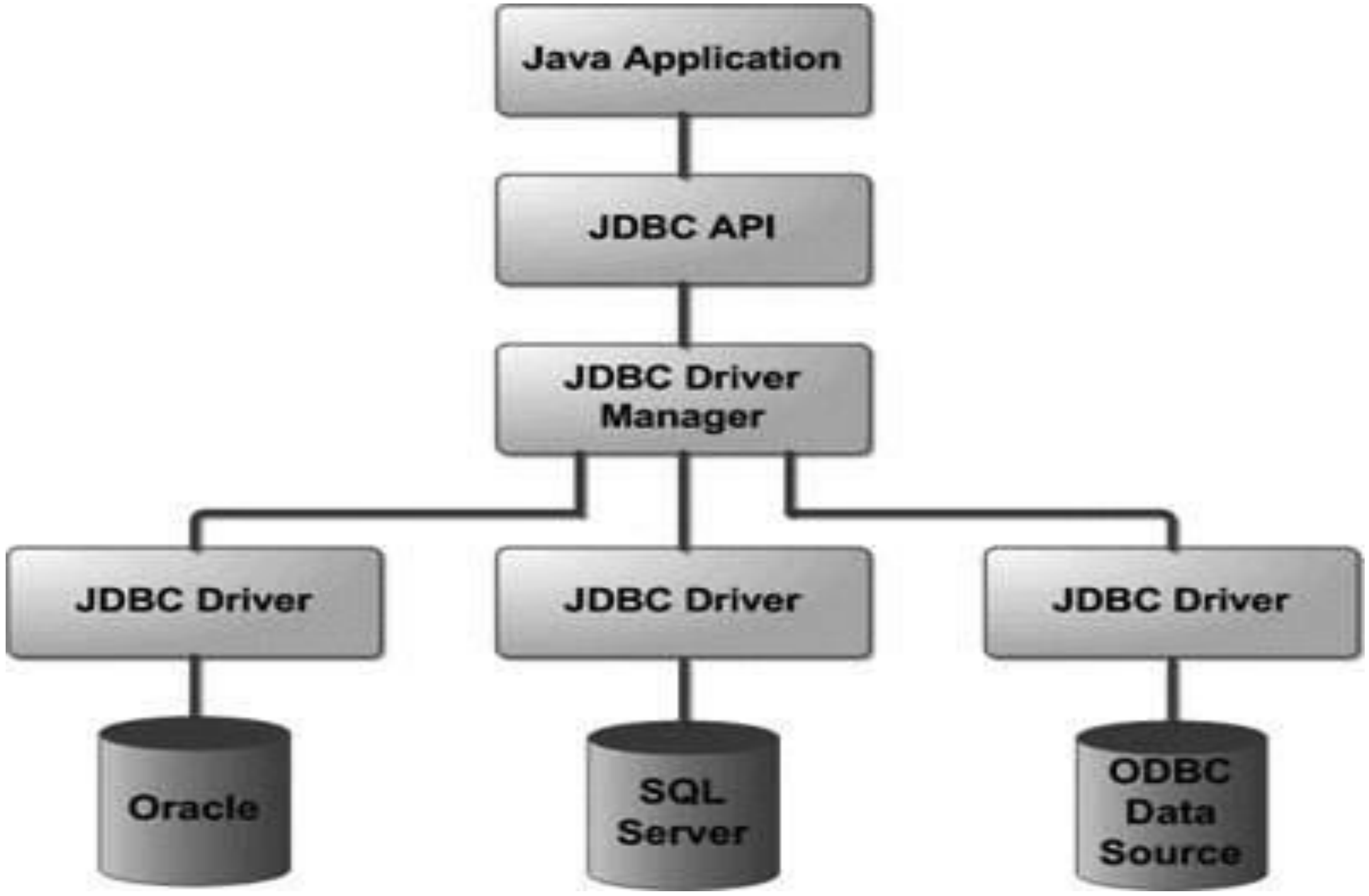
JAVA DATABASE CONNECTIVITY (JDBC)

- **J**ava **D**ata**b**ase **C**onnectivity : It is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

- The JDBC library includes APIs for each of the tasks commonly associated with database usage:
  - Making a connection to a database
  - Creating SQL or MySQL statements
  - Executing that SQL or MySQL queries in the database
  - Viewing & Modifying the resulting records

- JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

JDBC Architecture



- JDBC Architecture consists of two layers
  - **JDBC API:** This provides the application-to-JDBC Manager connection.
  - **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

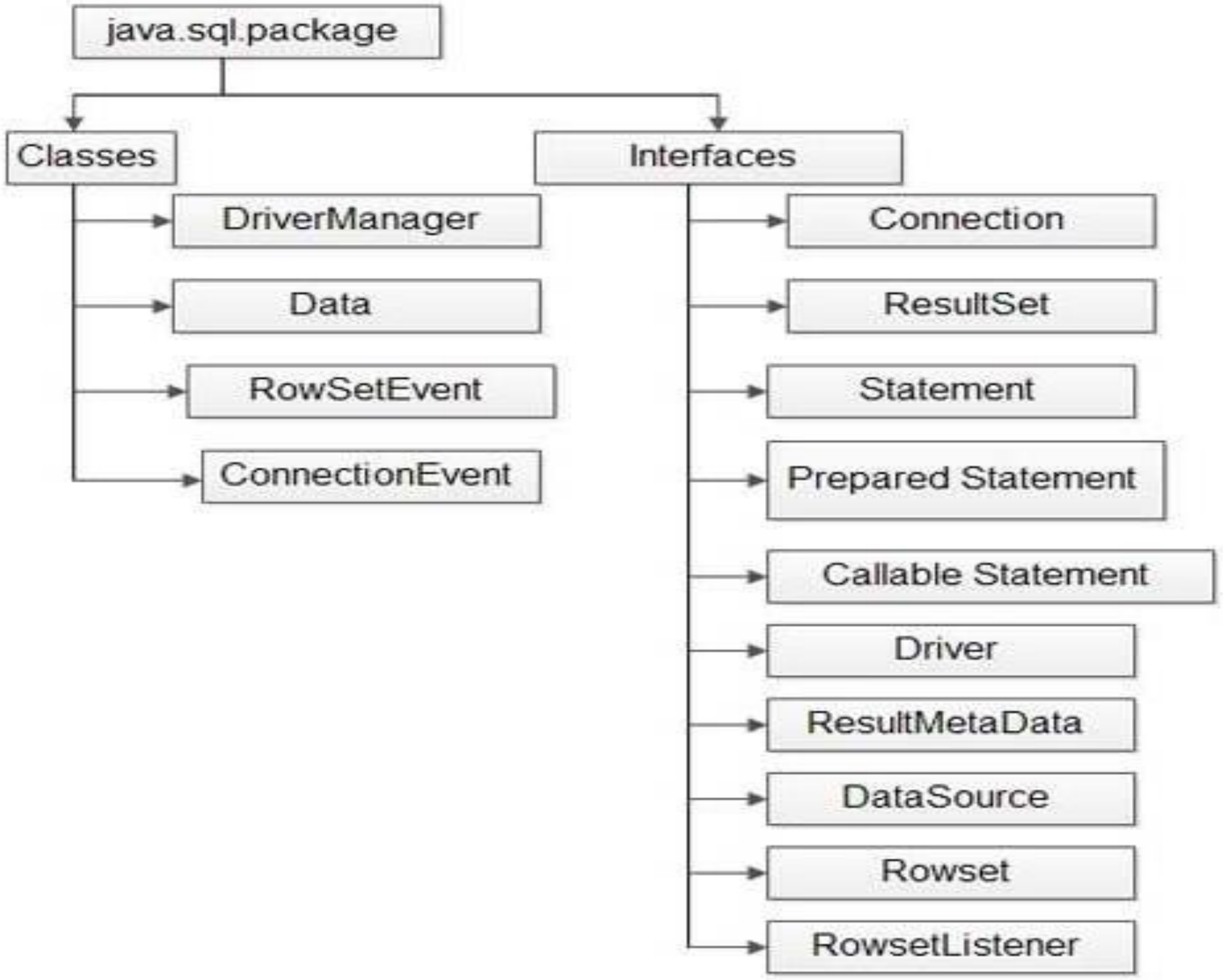
- The JDBC API uses a driver manager to provide transparent connectivity to heterogeneous databases.

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

Common JDBC Components

- The **JDBC API** provides the following **interfaces** and **classes**:
  - **DriverManager:** This class manages a list of database drivers.
  - **Driver:** Handles the communications with the database server
    - A **JDBC driver** is a software component enabling a Java application to interact with a database.
    - To connect with individual databases, JDBC requires drivers for each database.
  - **Connection:** All communication with database is through connection interface object.
  - **Statement:** This interface object is used to submit the SQL statements to the database.
  - **ResultSet:** These objects hold data retrieved from a database . It acts as an iterator to allow you to move through its data.
  - **SQLException:** This class handles any errors that occur in a database application.

- The **forName()** method of **java.lang.Class** is used to register the driver class.
- The **getConnection()** method of DriverManager class is used to establish connection with the database
- The **createStatement()** method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- The **executeQuery()** method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.
- By closing connection object statement and ResultSet will be closed automatically. The **close()** method of Connection interface is used to close the connection.


java.sql.package
Classes
DriverManager
Data
RowSetEvent
ConnectionEvent
Interfaces
Connection
ResultSet
Statement
Prepared Statement
Callable Statement
Driver
ResultMetaData
DataSource
Rowset
RowsetListener

Creating and Executing Queries

- **S**tructured **Q**uery **L**anguage (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.
- SQL is supported by almost any database that is used, and it allows to write database code independently of the underlying database.
- **Create Database**

  Syntax

  CREATE DATABASE *databasename*;

  ```
  CREATE DATABASE emp;
  ```
- **Drop Database**

  Syntax

  DROP DATABASE *databasename*;

  ```
  DROP DATABASE emp;
  ```

- **Create Table**

  Syntax

  CREATE TABLE *table_name* (*column1 datatype(size)*, *column2 datatype(size),…..*);

  CREATE TABLE Employees (id int NOT NULL, age int NOT NULL, first VARCHAR(10), last VARCHAR(10), PRIMARY KEY (id)); /*The NOT NULL constraint enforces a //column to NOT accept NULL values.*/

- **Drop Table**

  Syntax

  DROP TABLE *table_name*;

  DROP TABLE Employees;

- **INSERT Data**

  Syntax

  INSERT INTO *table_name* **VALUES** (*value1*, *value2*, *value3*, …);

  INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');

- **SELECT Data**

  Syntax

  SELECT *column1*, *column2*, …**FROM** *table_name*;

  OR

  SELECT * **FROM** *table_name*;   *// To select the whole table*

  SELECT first, last, age FROM Employees WHERE id = 100

- **UPDATE Data**

  Syntax

  UPDATE *table_name* **SET** *column1* = *value1*, *column2* = *value2*, …
  WHERE *condition*;

  UPDATE Employees SET age=20 WHERE id=100;

- **DELETE Data**

  Syntax

  DELETE FROM *table_name* **WHERE** *condition*;

  DELETE FROM Employee WHERE id=100;

Java Database Connectivity with 6 Steps

1. **Import the packages:** *import java.sql.**
2. **Register the JDBC driver:**

   To open a communication channel with the database.

   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //MS Access

   Class.forName("com.mysql.jdbc.Driver"); //MySQL

   [or Class.forName("org.apache.derby.jdbc.ClientDriver");]//netbeans
3. **Open a connection:**

   Connection connect = DriverManager.getConnection("jdbc:odbc:mydsn");//MSAccess

   Connection connect = DriverManager.getConnection ("jdbc:mysql://localhost:3306/mydatabase","root","root");//MySQL

   [or Connection connect=DriverManager.getConnection("jdbc:derby://localhost:1527/Test1", "Test", "Test");]//netbeans

4. **Execute a query:** build and submit an SQL statement. For that ,

   First create the statement object

   Statement stmt=con.createStatement(); //**OR** PreparedStatement

   Then execute query

   ResultSet rs=stmt.executeQuery("select * from *tablename*");

5. **Extract data from result set:** Use appropriate *ResultSet.getXXX()* to retrieve the data from the result set

   while(rs.next()){

   System.out.println(rs.getInt(1)+" "+rs.getString(2));

   }

6. **Clean up the environment:** closing all database resources

   con.close();

Java Database Connectivity with MySQL

If we are using mysql database, we need to know following informations for the mysql database:

- 1. Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver.
- 2. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/mydatabase where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and mydatabase is the database name. We may use any database, in such case, we need to replace the mydatabase with our database name.

- 3. Username: The default username for the mysql database is **root**.
- 4. Password: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use **root** as the password.
- Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
 create database mydatabase;
use mydatabase;
create table employee(id int(10),name varchar(40),age  int(3));
```

Difference between Statement and PreparedStatement

- **1.Statement :**

  It is used for accessing your database. Statement interface cannot accept parameters and useful when you are using static SQL statements at runtime. If you want to run SQL query only once than this interface is preferred over PreparedStatement.

Eg) //Creating The Statement Object and then execute

```
        Statement st = con.createStatement();
st.executeUpdate("CREATE TABLE STUDENT(ID NUMBER, NAME VARCHAR)");
```

2. PreparedStatement :

It is used when you want to use SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

Eg) //Creating the PreparedStatement object

```
        PreparedStatement pst = con.prepareStatement("update STUDENT set
        NAME = ? where ID = ?"); //Setting values to place holders
        pst.setString(1, "RAM"); //Assigns "RAM" to first place holder
        pst.setInt(2, 512); //Assigns "512" to second place holder
        pst.executeUpdate(); //Executing PreparedStatement
```

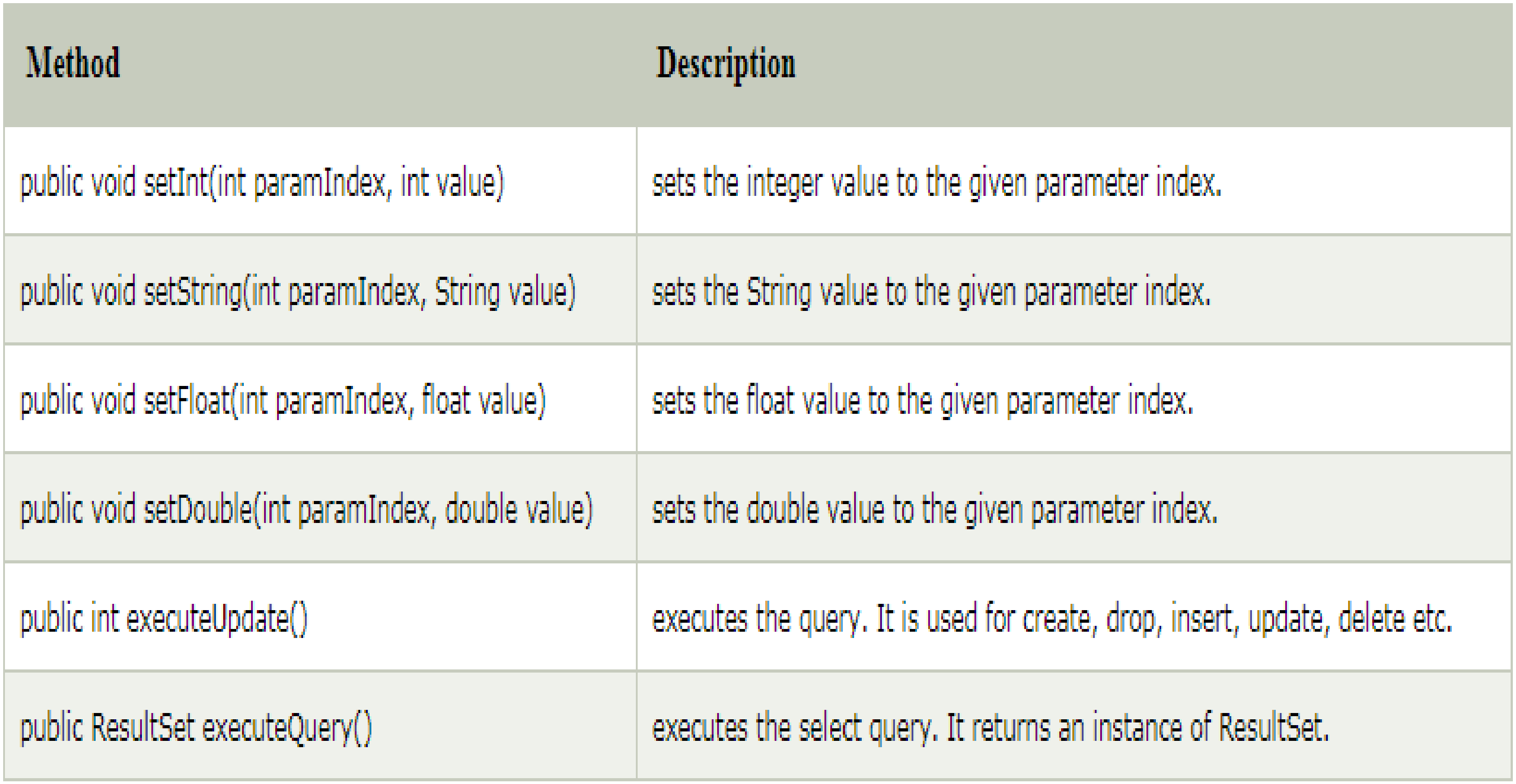
Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

- **1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- **2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- **3) public boolean execute(String sql):** is used to execute queries that may return multiple results.
- **4) public int[] executeBatch():** is used to execute batch of commands.

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:



Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Commonly used methods of ResultSet interface

1) **public boolean next():**is used to move the cursor to the one row next from the current position.
2) **public boolean previous():**is used to move the cursor to the one row previous from the current
3) **public boolean first():**is used to move the cursor to the first row in result set object.
4) **public boolean last():**is used to move the cursor to the last row in result set object.
5) **public int getInt(int columnIndex):**is used to return the data of specified column index of the current row as int.
6) **public int getInt(String columnName):**is used to return the data of specified column name of the current row as int.
7) **public String getString(int columnIndex):**is used to return the data of specified column index of the current row as String.
8) **public String getString(String columnName):**is used to return the data of specified column name of the current row as String.

```
import java.sql.*;
public class FirstExample
{   public static void main(String[] args)
   {    Connection connect = null;
        try
        {    Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
             connect = DriverManager.getConnection(“jdbc:odbc:mydsn”);
             PreparedStatement   pstm=connect.prepareStatement("SELECT id, first, last, age
                                                                              FROM Employees");
             ResultSet rs =pstm.executeQuery();
             while(rs.next())
             {    int id = rs.getInt("id");                         int age = rs.getInt("age");
                  String first = rs.getString("first");       String last = rs.getString("last");
                  System.out.print("ID: " +id+", Age: "+age+", First: "+first+", Last: "+ last);
             }
             rs.close();   pstm.close();   connect.close();
         }
         catch(SQLException se)
           {     System.out.println(“SQL EXCEPTION OCCURRED”);}

        }
}
```

Sample Code – select *

```
void viewall()
{
	try
	{
		PreparedStatement pstm=connect.prepareStatement("select * from 
										Mytab");
		ResultSet rs=pstm.executeQuery ();
		while (rs.next())
		{	// Roll & Name are fields in database
			System.out.println(rs.getInt("Roll")+ " "+rs.getString("Name"));
		}
	}
	catch(Exception e){}
}
```

Sample Code - Search by ID

```
void search()
{
	try
	{
		int r=12;
		PreparedStatement  pstm=connect.prepareStatement("select * from Mytab
		                                                   where Roll=?");
		pstm.setInt(1,r);
		ResultSet rs=pstm.executeQuery ();
		while (rs.next())
		{
			int roll = rs.getInt("Roll");
			String sname = rs.getString("Name");
			System.out.println(roll+ " "+sname);
		}
	}
	catch(Exception e){}
}
```

Sample Code - insert

```
void addstudent(int rollno, String name )
{
        try
        {
                PreparedStatement   pstm=connect.prepareStatement("insert into Mytab
                                                                        (Roll,Name)values(?,?)");
                pstm.setInt(1,rollno);
                pstm.setString(2,name);
                pstm.executeUpdate();
        }
        catch(Exception e){}
}
```

Sample Code - update

```
void edit(int roll, int roll_edit, String name_edit)
{
	try
	{

		PreparedStatement pstm =connect.prepareStatement("update Mytab set
		Roll=?,Name=? where Roll=?");
		pstm.setInt(1,roll_edit);
		pstm.setString(2,name_edit);
		pstm.setInt(3,roll);
		pstm.executeUpdate();
	}
	catch(Exception e){}
}
```

Sample Code - delete

```
void delete(int roll)
{
        try
        {
                PreparedStatement pstm=connect.prepareStatement("delete from Mytab  where
                Roll=?");
                pstm.setString(1,roll);
                pstm.executeUpdate();
        }
        catch(Exception e){}
}
```