What is Swing?

- The Java Swing provides the multiple platform independent APIs interfaces for interacting between the users and GUIs components.

All Java Swing classes are placed in javax.swing package

Java Foundation Classes (JFC)

- joint effort between SUN & Netscape produced the original swing set of components as part of JFC
- JFC contains
  swing
  cut & paste-(clipboard support)
  Accessibility support
  desktop features
  java 2D

Why Swing?

- offers improved functionality over AWT(new components, expanded components features, better event handling & drag & drop)
- Swing components are rendered by java code rather than native code
- AWT is a part of standard java distribution.
  It has limited implementation ,not designed to provide a serious , main UI.
- AWT component set was not designed for complex programming needs, has lots of bugs, & takes up lot of system resources.

Swing uses fewer system resources, adds lots more sophisticated components, helps to tailor the look & feel of the programs.

swing development has its root in MVC(Model – View- Controller) architecture.

MVC allows swing components to be replaced with different data models & views. Plug gable look & feel is result of MVC architecture.

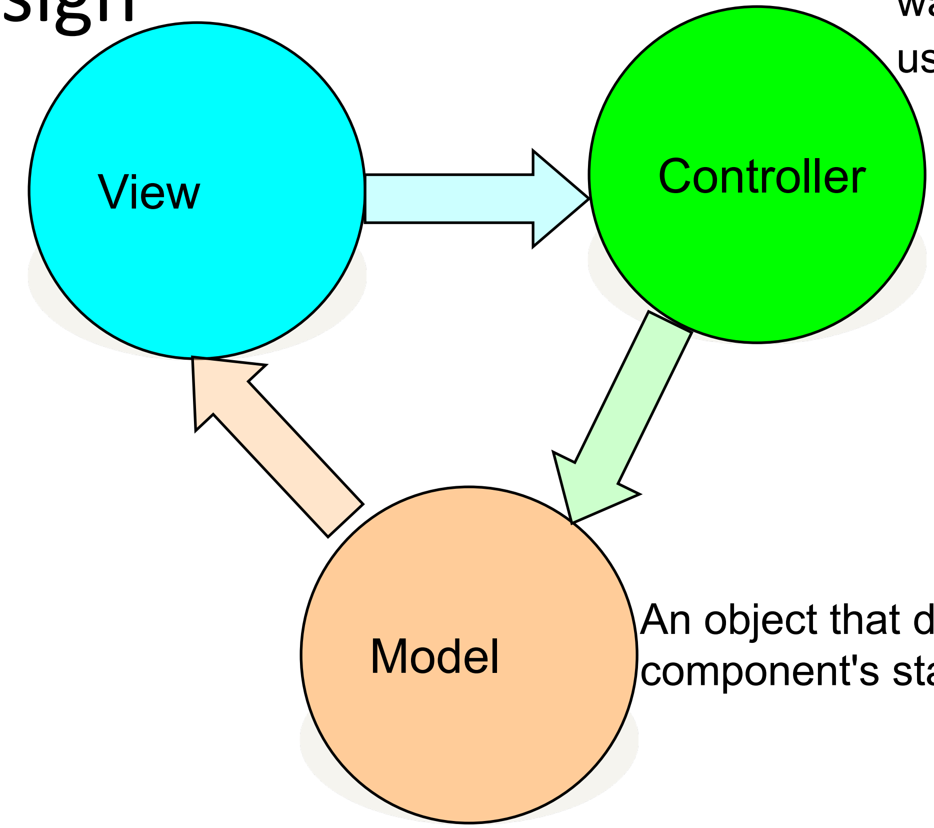
Since java is platform independent & runs on client machine , the look & feel of any platform has to be known

Lightweight components:- in swing most of the components have their own view supported by java look & feel classes (it can't rely on native system classes)

Pluggable look & feel:- supports cross platform look & feel also called java look & feel that remains same across all platforms wherever the program runs.

Swing Overview

- MVC Design



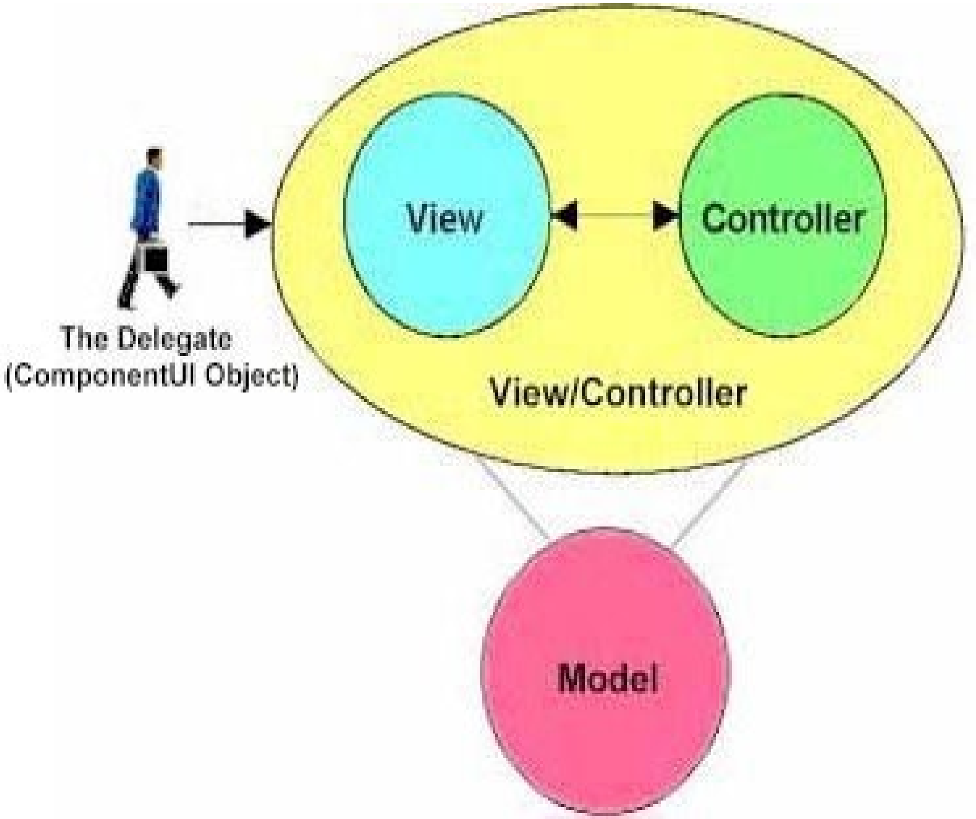
The visual screen representation of a component

An object that controls a component in such a way that it responds to user input.

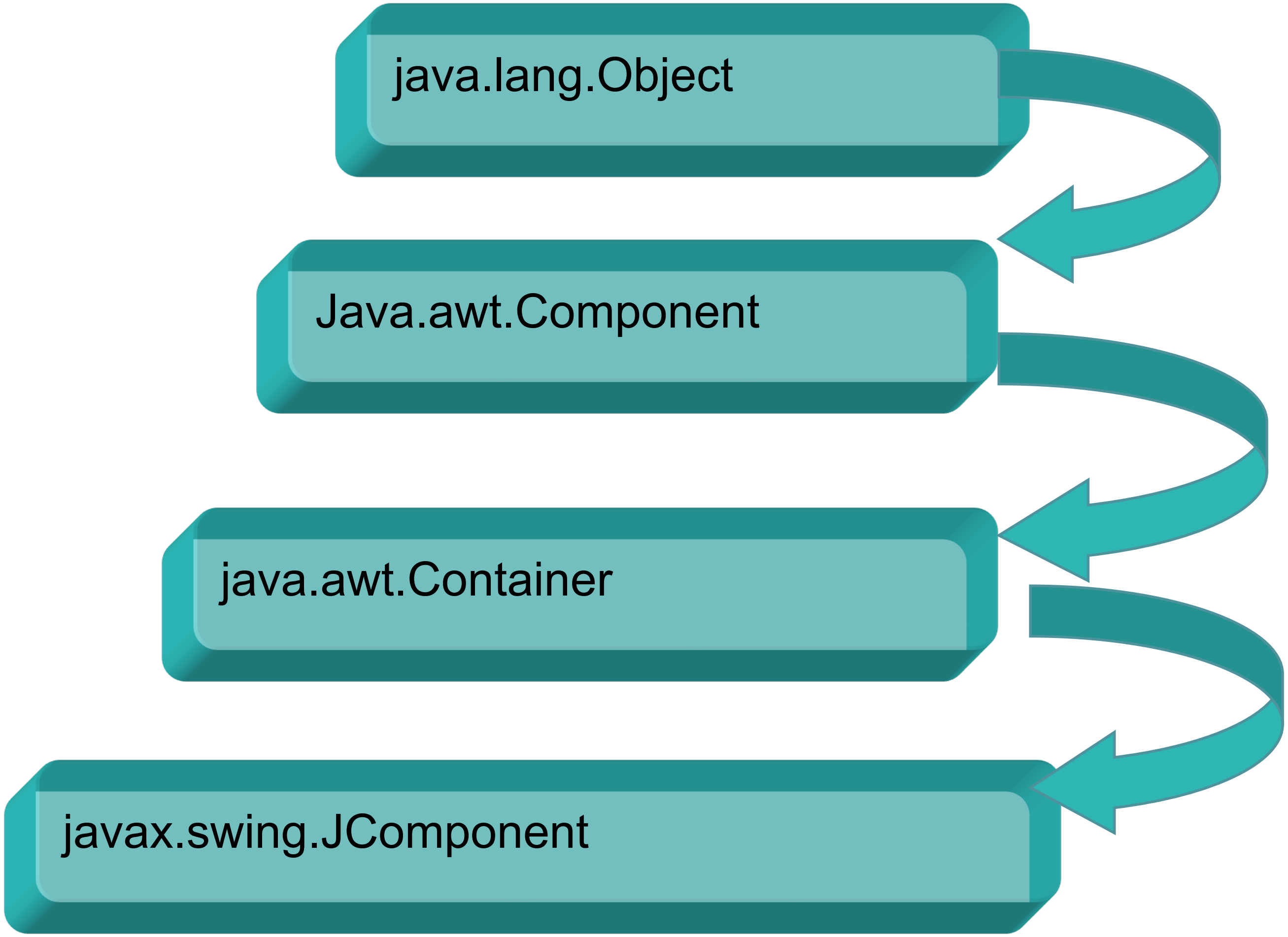
An object that defines the component's state

Swing Overview

- Delegate
  - A vital part of Swing's pluggable L&F mechanism is a user-interface (*UI*) object called a *delegate*.



Swing Component Hierarchy



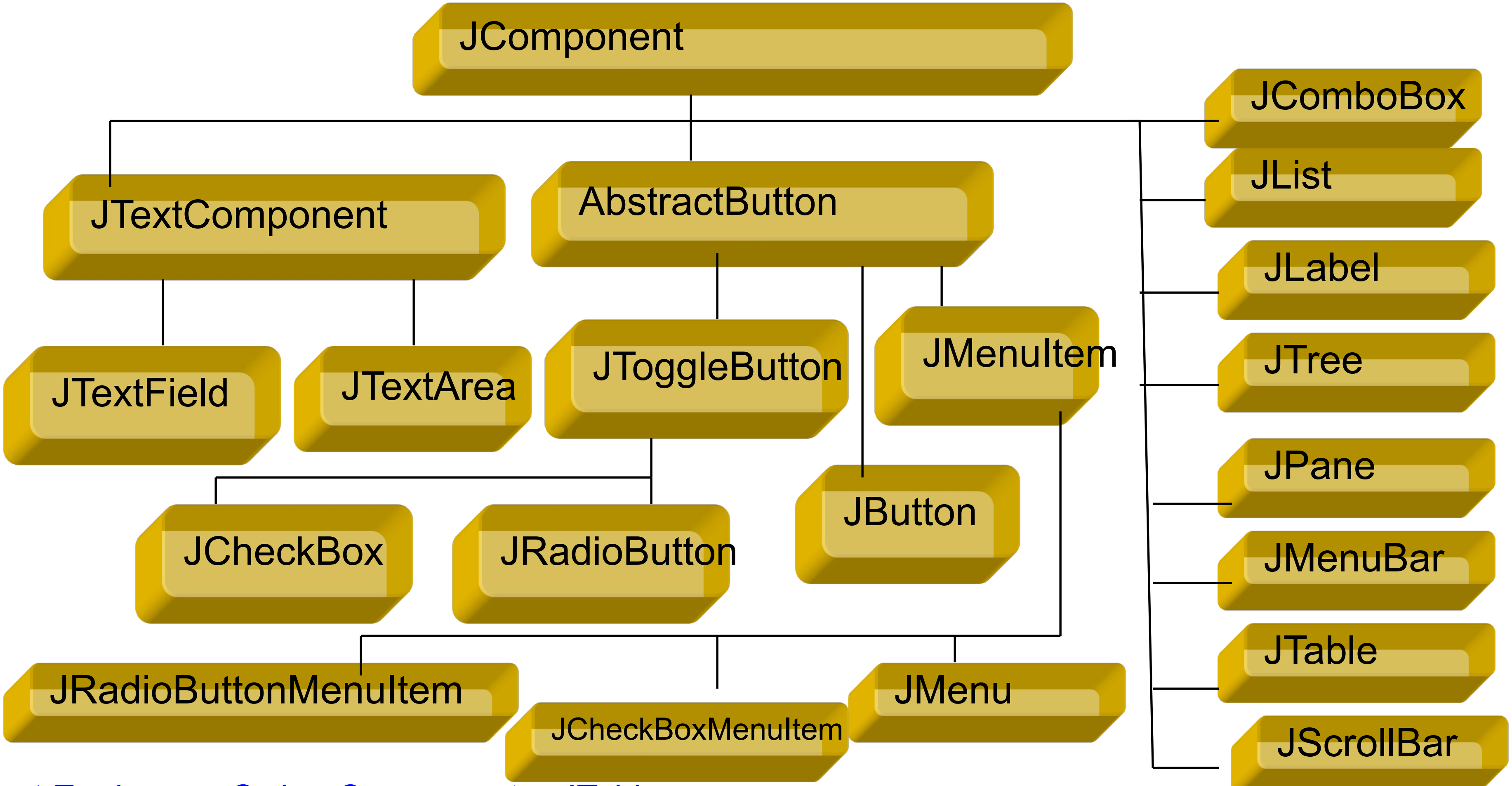
java.lang.Object

Java.awt.Component

java.awt.Container

javax.swing.JComponent

Swing Component Hierarchy



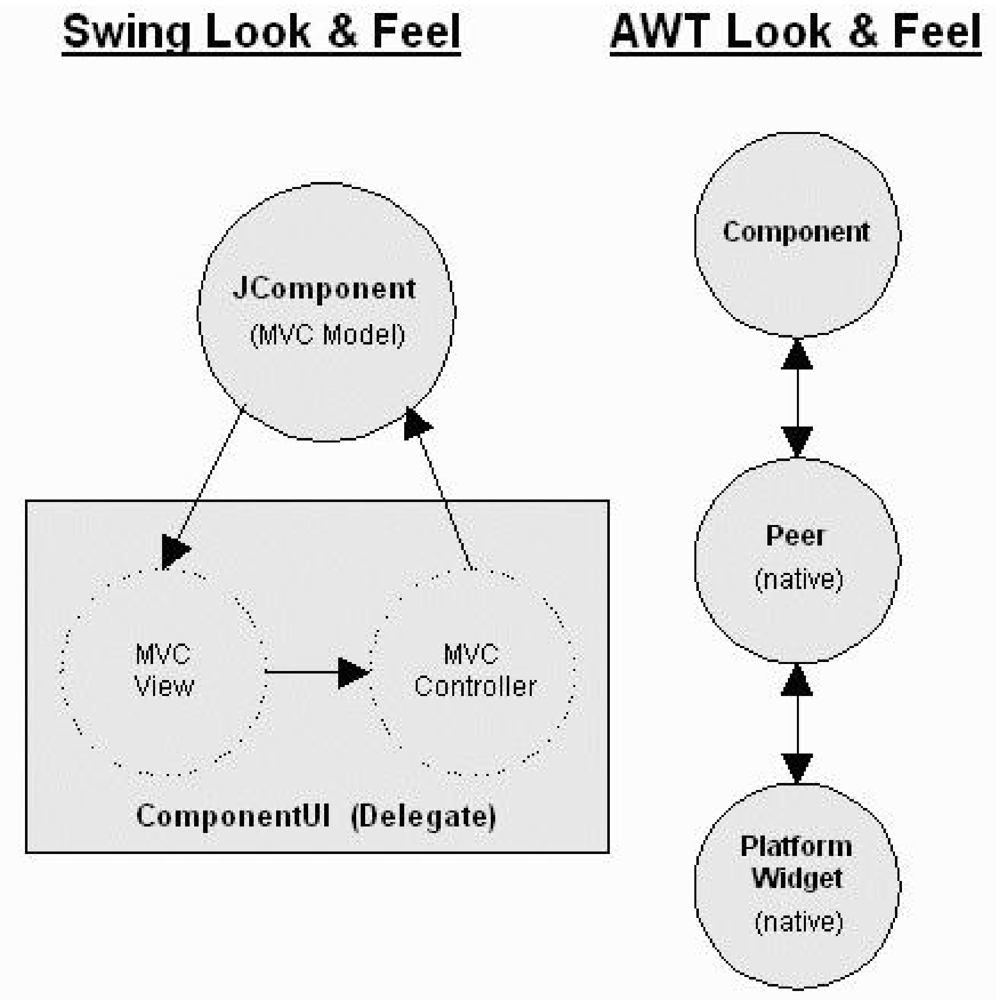
Awt Equiv   Swing Components JTable

SWING V.S AWT

- Components are implemented with absolutely no native code
- More functionality
- Can be shipped as add-on to JDK1.1, JDK1.2
- Lightweight components
- Swing labels & buttons can display images instead of text.
  Can add/ change borders drawn around components
  components don’t have to be rectangular.
  Assertive technologies can read data from swing components.

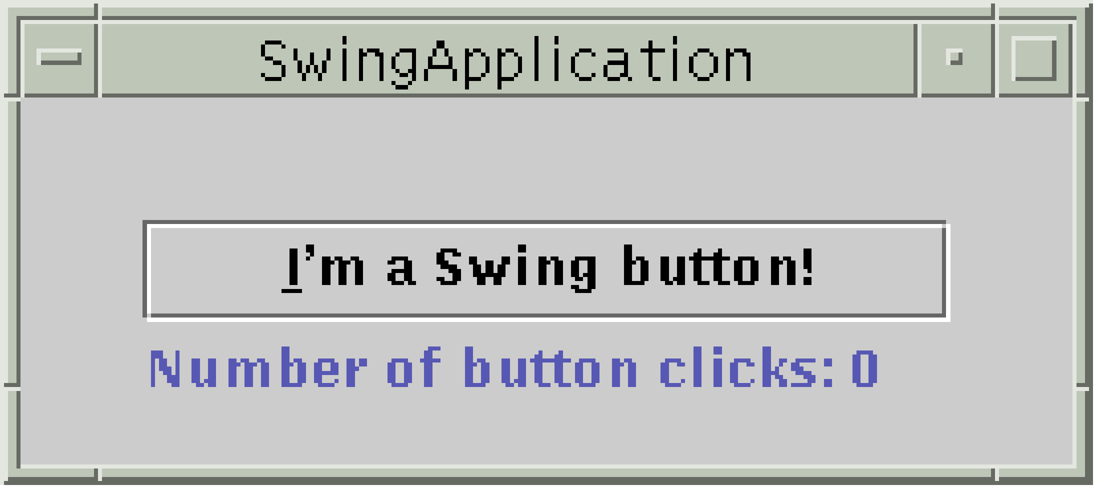
  All swing elements are part of javax.swing package.
  Import javax.swing.*;
  Swing API is powerful, flexible & immense.

Swing vs. AWT


Swing Look & Feel
AWT Look & Feel
JComponent
(MVC Model)
MVC
View
MVC
Controller
ComponentUI (Delegate)
Component
Peer
(native)
Platform
Widget
(native)

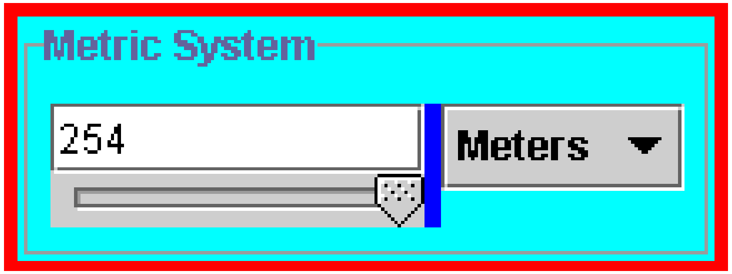
Swing Components and the Containment Hierarchy

- An example:

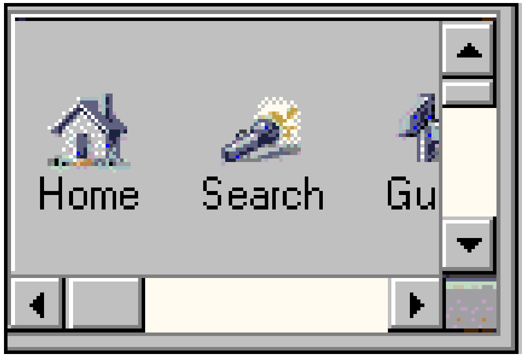


- Four components in this GUI:
  - a frame, or main window (JFrame) --- top-level container
  - a panel, sometimes called a pane (JPanel) --- intermediate container
  - a button (JButton) --- atomic components
  - a label (JLabel) --- atomic components

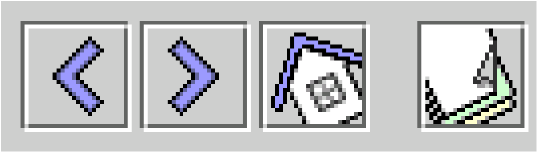
General-Purpose Containers



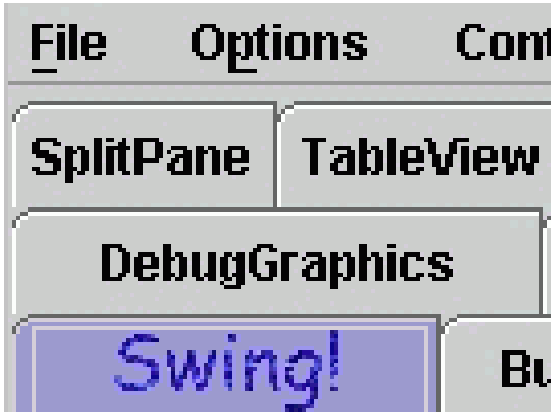
Panel (and JPanel)



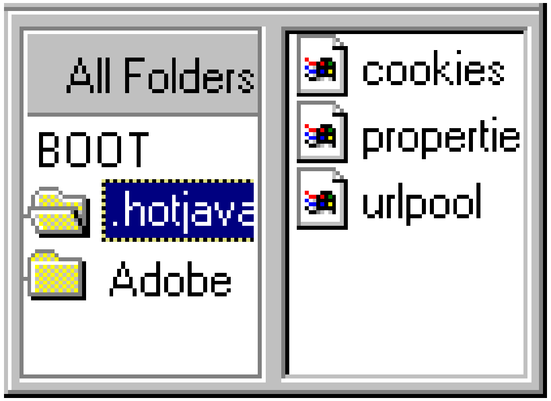
JScrollPane



JToolBar

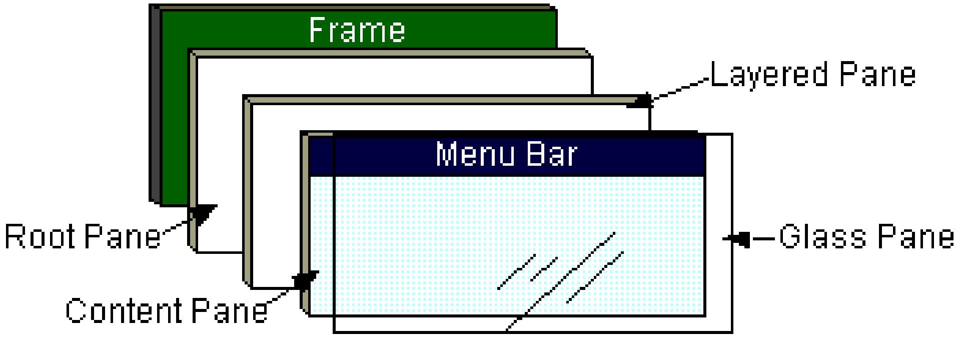


JTabbedPane

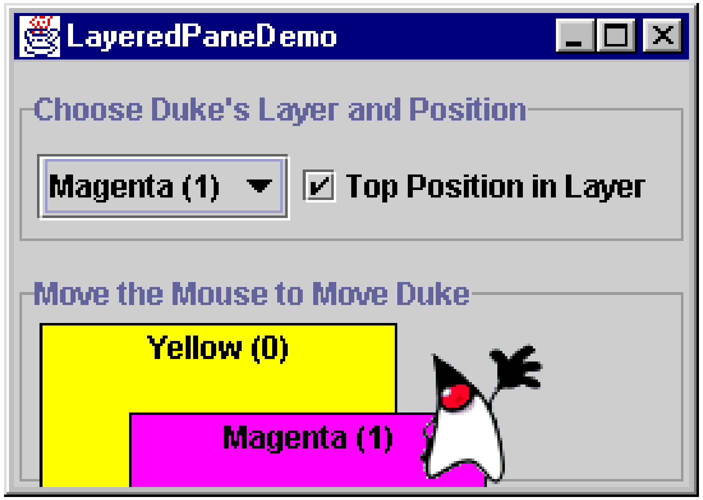


JSplitPane

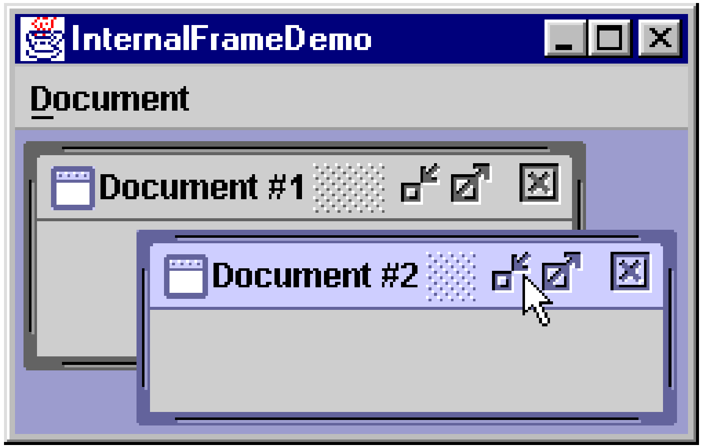
Special-Purpose Containers



Root Pane

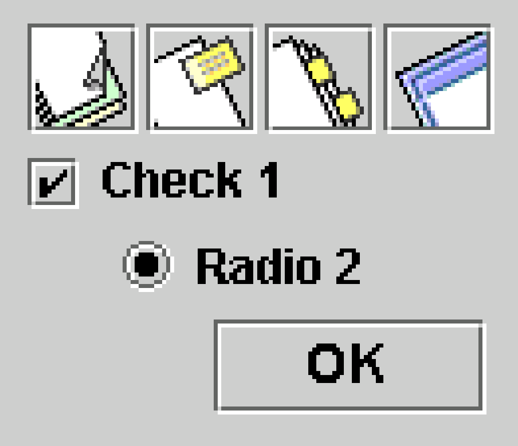


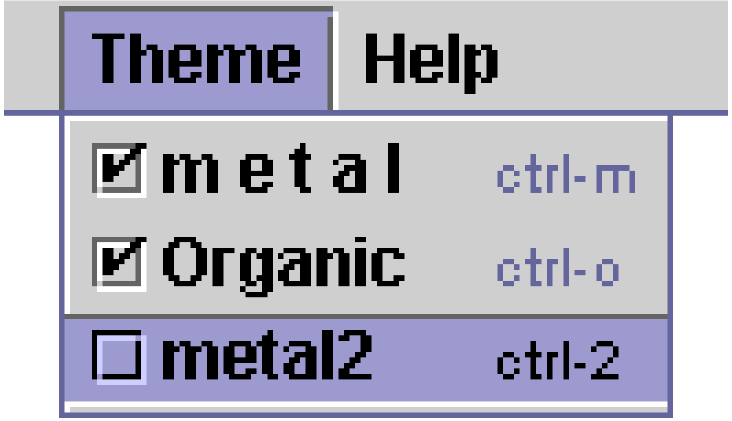
JLayeredPane



JInternalFrames

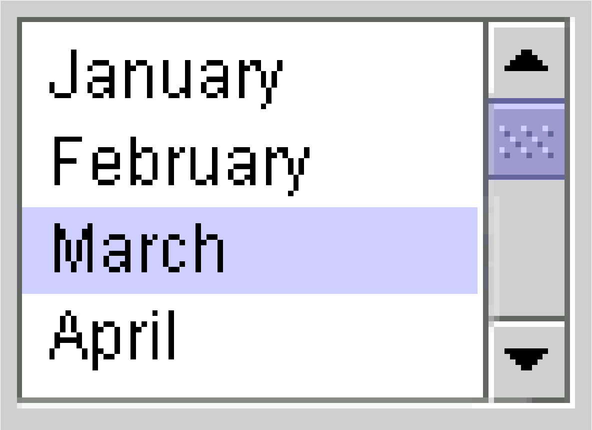
Basic Controls





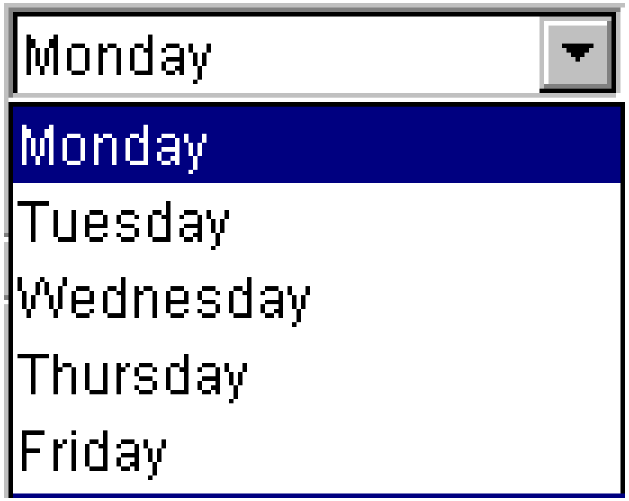
JCheckBox
JRadioButton
JButton

JMenu
JMenuItem

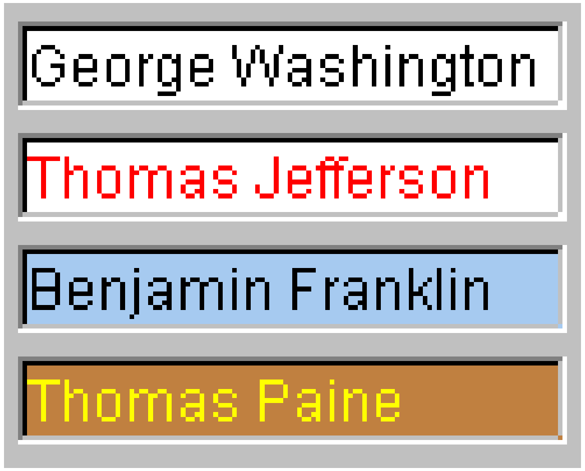


List (and JList)

Basic Controls



Choice (and JComboBox)



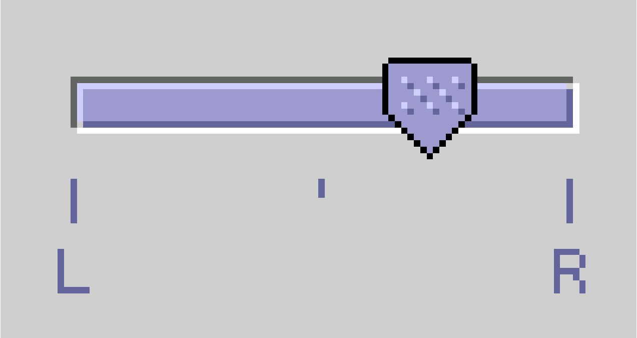
George Washington

Thomas Jefferson

Benjamin Franklin

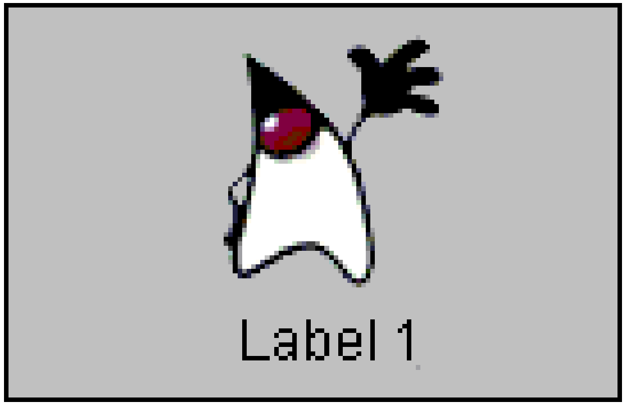
Thomas Paine

JTextField

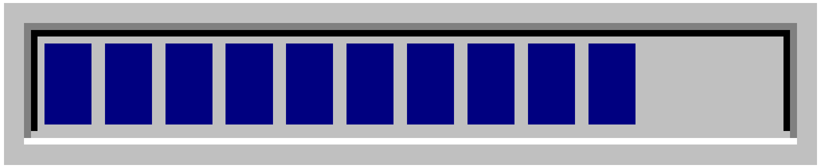


JSlider

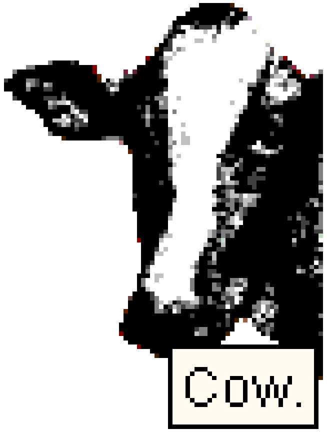
Uneditable Information Displays



Label (and JLabel)

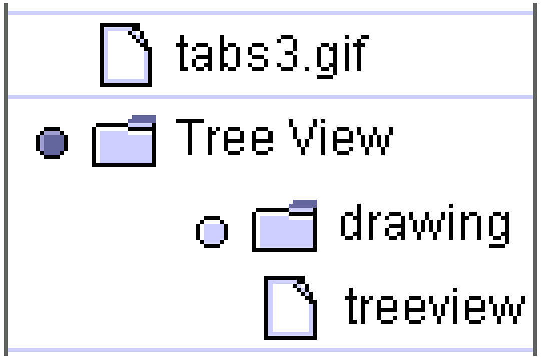


JProgressBar



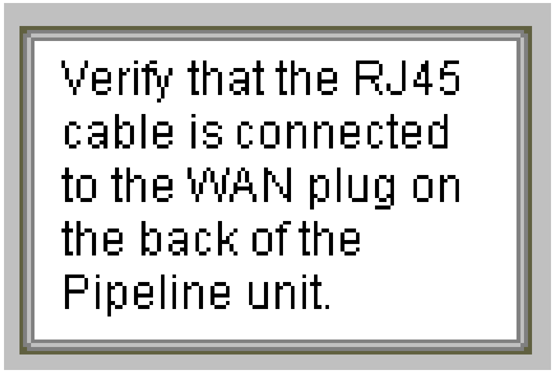
JToolTip

Editable Displays of Formatted Information



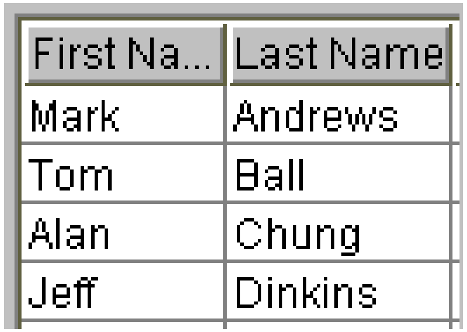
- tabs3.gif
- ● Tree View
  - ○ drawing
    - treeview

JTree



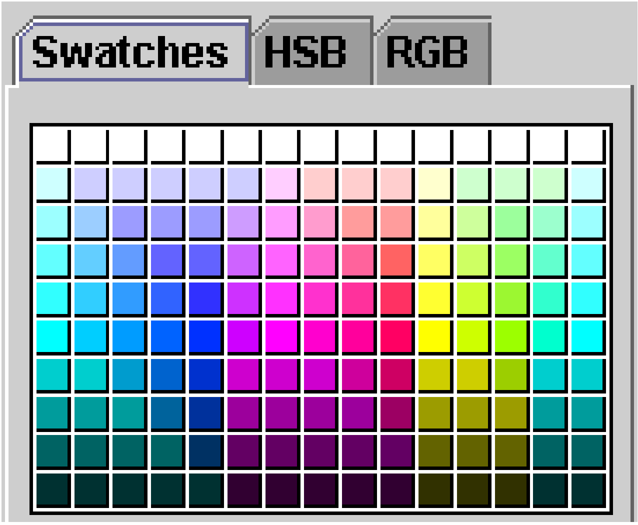
Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.

JText

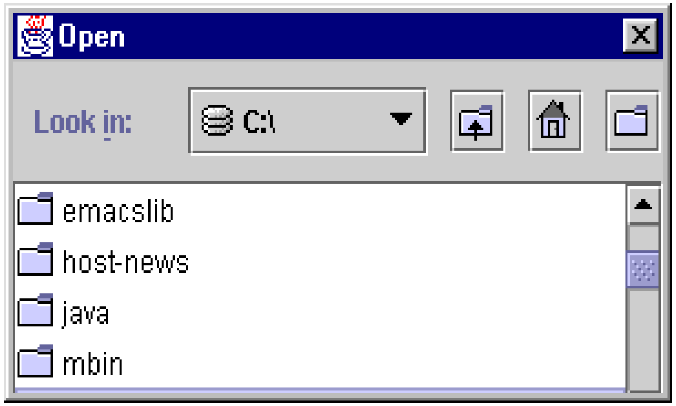


First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

JTable



JColorChooser



Open

Look in: C:\

- emacslib
- host-news
- java
- mbin

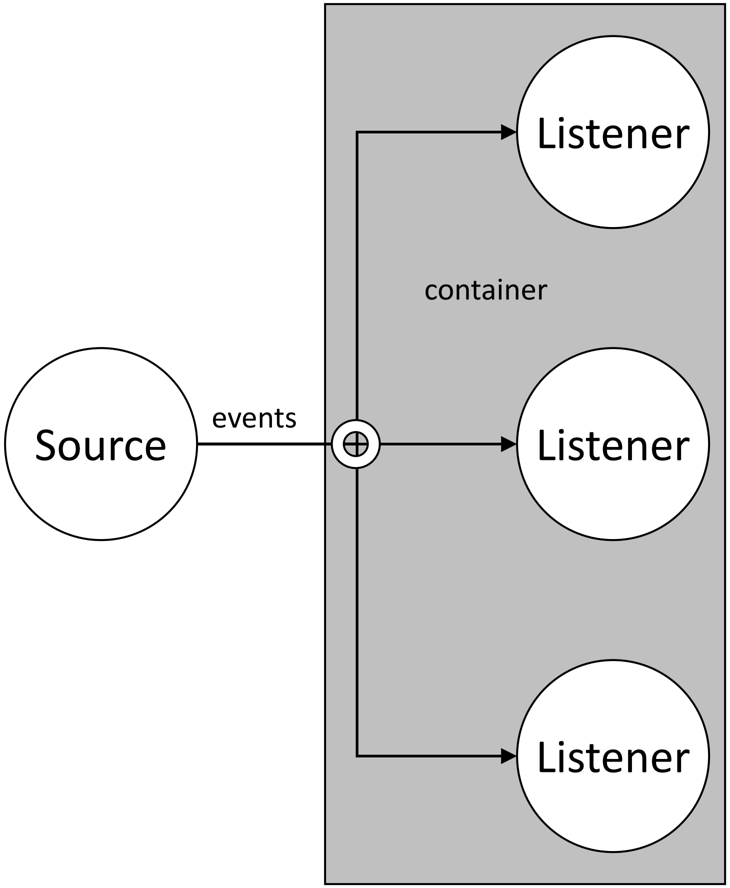
FileDialog (and JFileChooser)

Event Handling

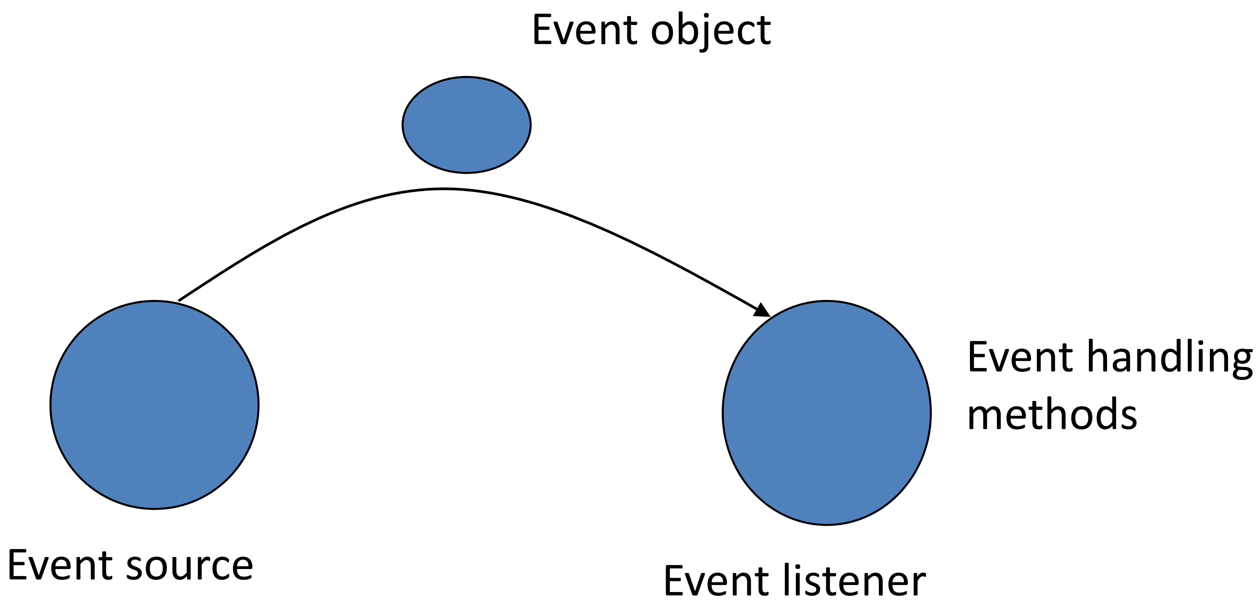
- An event is an object that describes some state change in a source
- Each time a user interacts with a component an event is generated, e.g.:
  - A button is pressed
  - A menu item is selected
  - A window is resized
  - A key is pressed
- An event informs the program about the action that must be performed

The Delegation Event Model

- Provides a standard mechanism for a **source** to generate an **event** and send it to a set of **listeners**
- A source generates events.
- 3 responsibilities of a source:
  - To provide methods that allow listeners to register and unregister for notifications about a specific type of event
  - To generate the event
  - To send the event to all registered listeners.



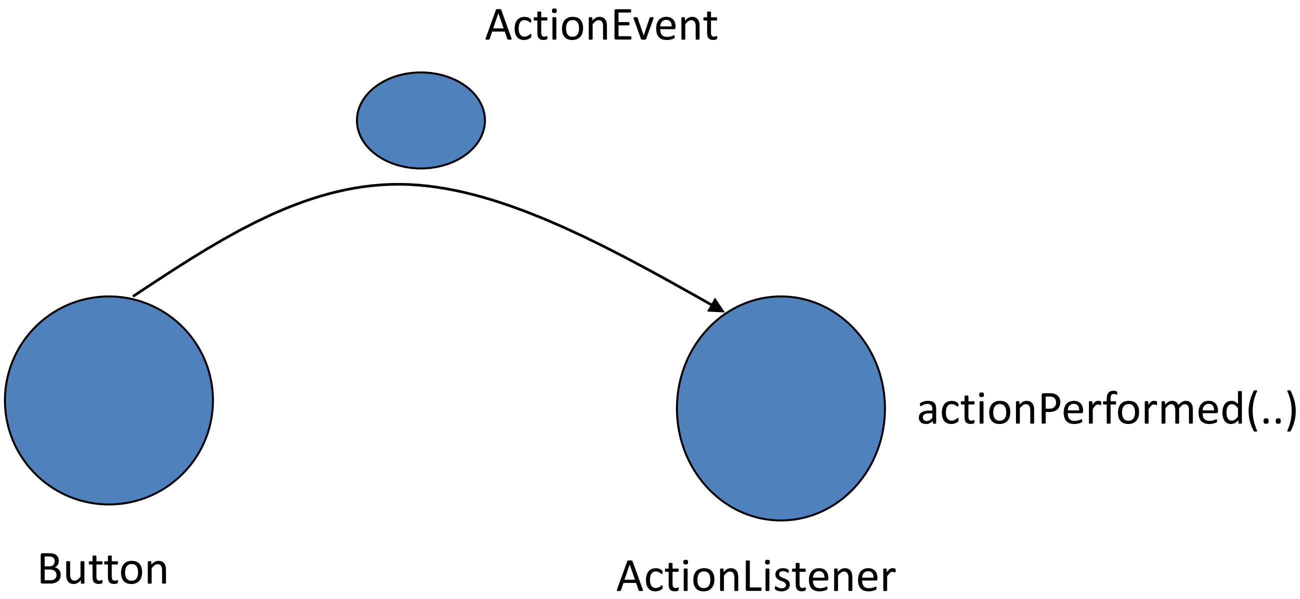
Event Handling Model of AWT



Event Classes

- **The EventObject class has the two methods**
  - **The getSource() method returns the object that generated the event**
  - **toSource() method returns a string equivalent of the event.**
- **The AWTEvent(Object source, int id)**
  - **Source is the object that generates the event and id identifies the type of the event.**
  - **The class has the getID() method that returns the type of the event.**
- **Event Listener (java.util.EventListener) interface does not define any constraints or methods but exists only to identify those interfaces that process events**
- **The Component class has the methods that allow a listener to register and unregister for events:**
  - **void add*Type*Listener(*Type*Listener tl)**
  - **void remove*Type*Listener(*Type*Listener tl)**
  - **Eg:- addKeyListener()**

Action Events on Buttons



Semantic Event Listener

- The semantic events relate to operations on the components in the GUI. **semantic event** classes.
  - An **ActionEvent** is generated when there was an action performed on a component such as clicking on a menu item or a button.
    - Produced by Objects of Type: Buttons
  - An **ItemEvent** occurs when a component is selected or deselected.
    - Produced by Objects of Type: Menus
  - An **AdjustmentEvent** is produced when an adjustable object, such as a scrollbar, is adjusted.
    - Produced by Objects of Type: Scrollbar
- Semantic Event Listeners
  - Listener Interface: ActionListener, Method: void actionPerformed(ActionEvent e)
  - Listener Interface: ItemListener, Method: void itemStateChanged (ItemEvent e)
  - Listener Interface: AdjustmentListener, Method: void adjustmentValueChanged (AdjustmentEvent e)

Using the ActionListener

- Stages for Event Handling by ActionListener
  - First, import event class

    import java.awt.event.*;

  - Define an overriding class of event type (implements ActionListener)

```
class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                // Write what to be done. . .
              label.setText(“Hello World!”);
        }
}
```

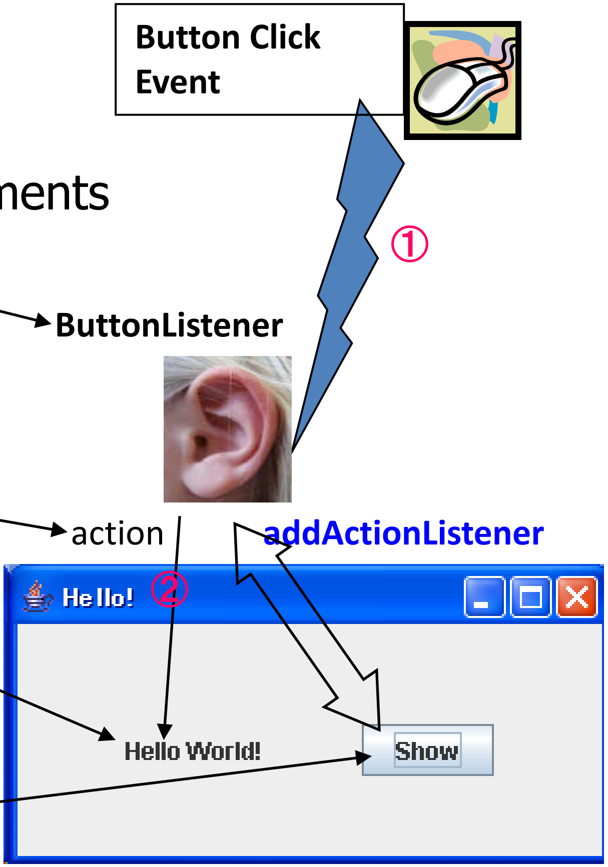
  - Create an event listener object

    ButtonListener bt = new ButtonListener();

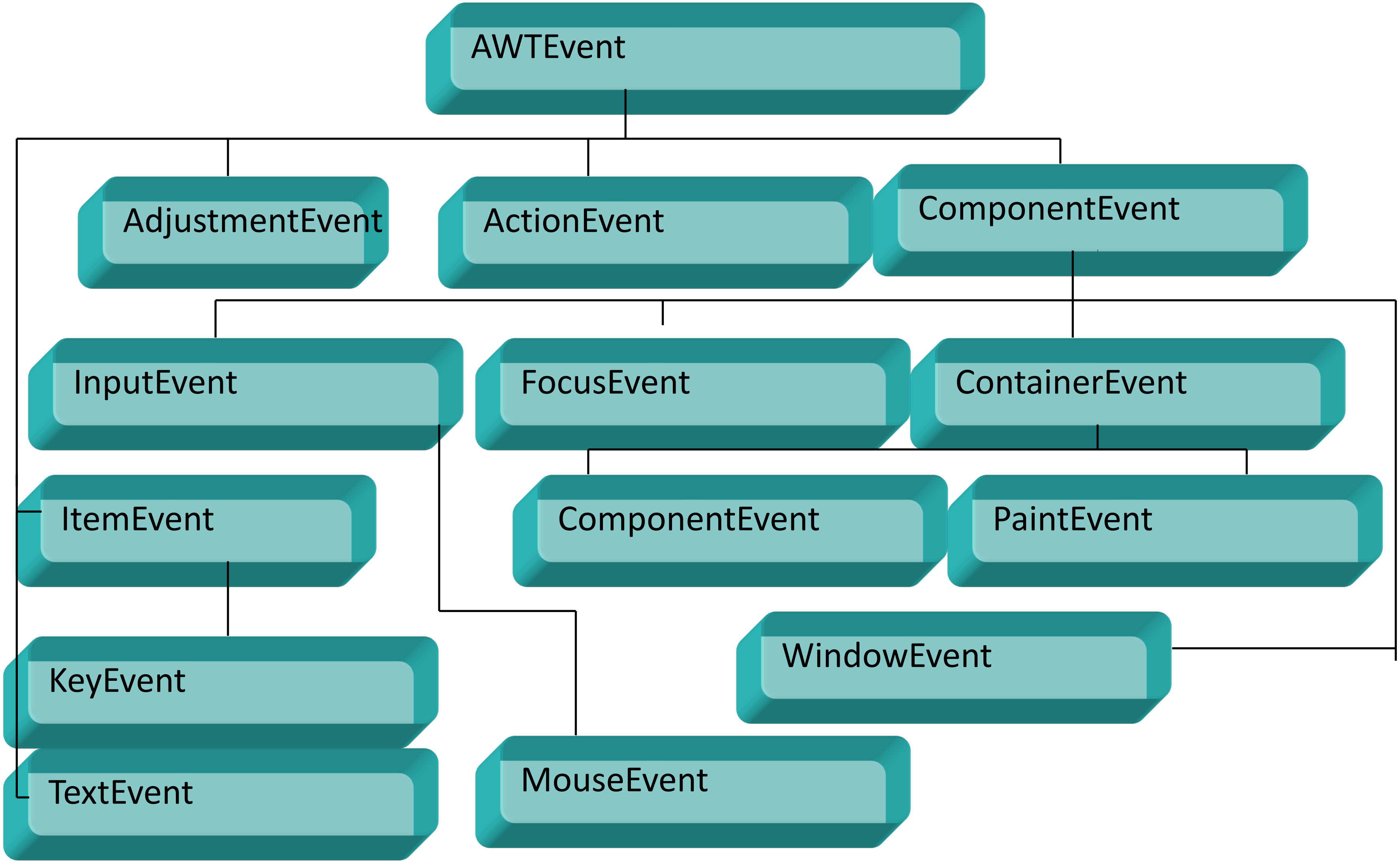
  - Register the event listener object

    b1 = new Button(“Show”);

    b1.**addActionListener**(bt);

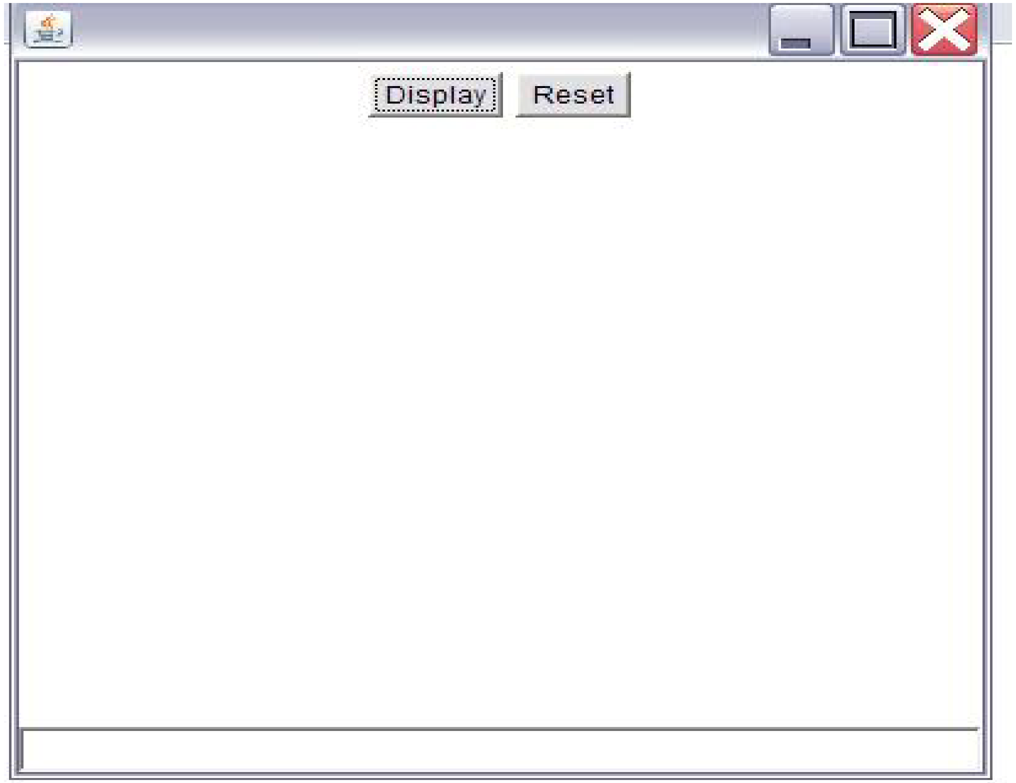


Types of AWT Events

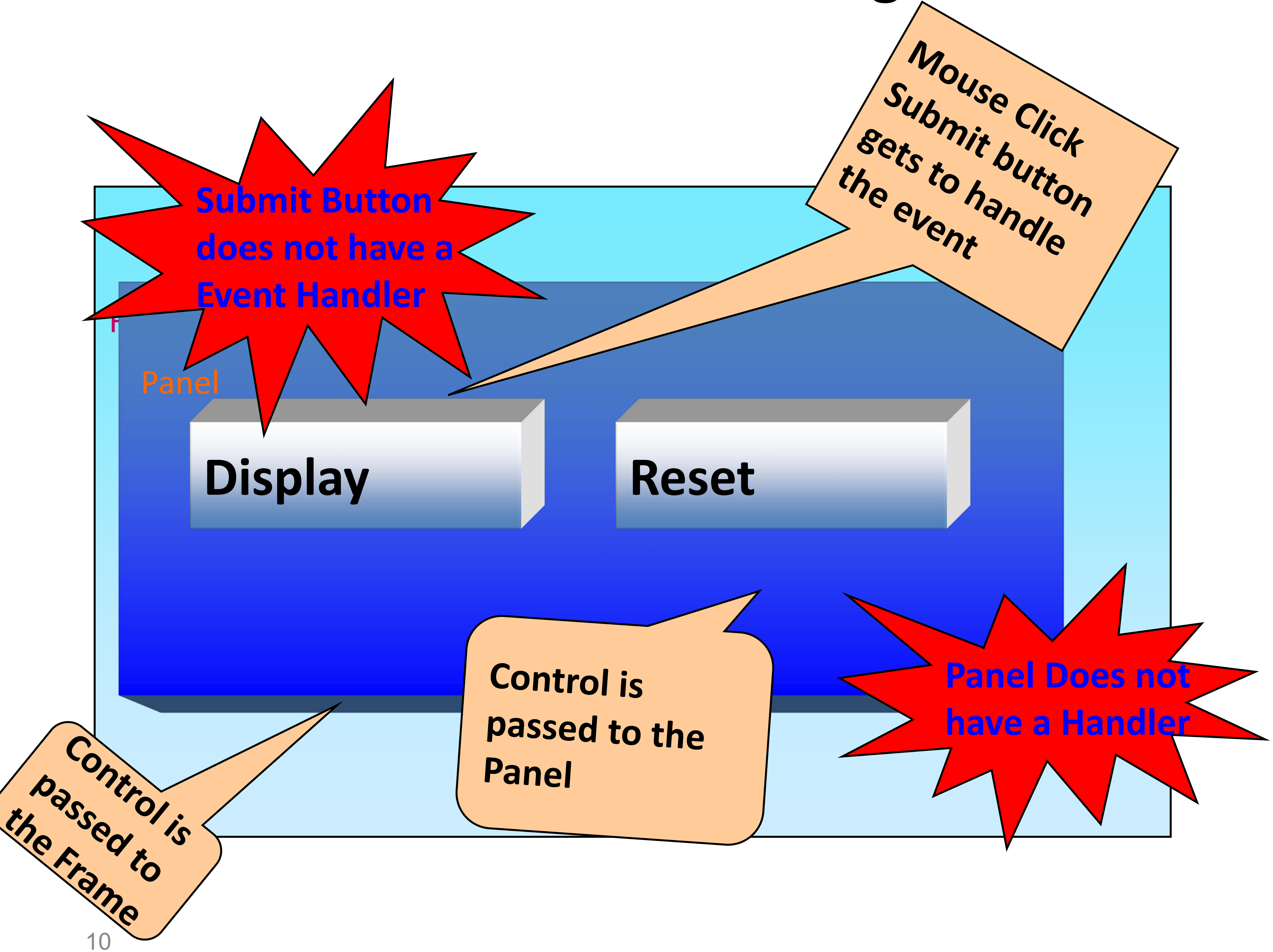


Event Dispatching

Consider the Following



Event Handling



Submit Button does not have a Event Handler

Mouse Click Submit button gets to handle the event

Control is passed to the Panel

Panel Does not have a Handler

Control is passed to the Frame

Event Dispatch and Propogation

- When the user clicks into “submit” button
- java language run-time system gathers
- Event Class
- Component

Event Listener

- An object that would like to be notified of and respond to an event is an *event listener.*
- An object that generates a particular kind of event, called an *event source*, maintains a list of listeners that are interested in being notified when that kind of event occurs.
- When the event source generates an event the event source notifies all the listener objects that the event has occurred.

Event Listeners

- A Listener must be added to a component to react to the events occurring on the component
- An event is a component’s way of letting the listener know about that something has happened
- A component must have a way to register and deregister listeners
- The components must track its Listeners and pass on the events to those listeners
- Multicasting & Unicasting

Some Common EventListeners

Event Listener	Listener methods	Registered On
ActionListener	actionPerformed()	AbstractButton, Button, ButtonModel, ComboBoxEditor, JComboBox, JFileChooser, JTextField, List, MenuItem, TextField, Timer
ItemListener	itemStateChanged()	CheckBox,Choice etc
MouseListener	mouseClicked(),mousePressed(), mouseReleased(),mouseEntered(), mouseExited()	Component
TextListener	textValueChanged()	TextComponent
MouseMotionListener	mouseDragged(),mouseMoved()	Component

Some Common EventListeners

Event Listener	Listener methods	Registered On
WindowListener	**windowActivated(), windowClosed(), windowClosing(), windowDeactivated(), windowDeiconified(), windowIconified(), windowOpened()**	**Window**
FocusListener	**focusGained(),focusLost()**	**Component**
KeyListener	**keyTyped(),keyReleased(),keyPressed()**	**Component**

Example

```
import java.awt.*;
import java.awt.event.*;
public class MyApplication extends Frame
   implements ActionListener
{  Button b1,b2;
   TextField t1;
  Panel p1;
  MyApplication()
   {             b1=new Button(“Display”);
                 b2=new Button(“Clear”);
                 p1=new Panel();
                 t1=new TextField(20);
```

Example contd.

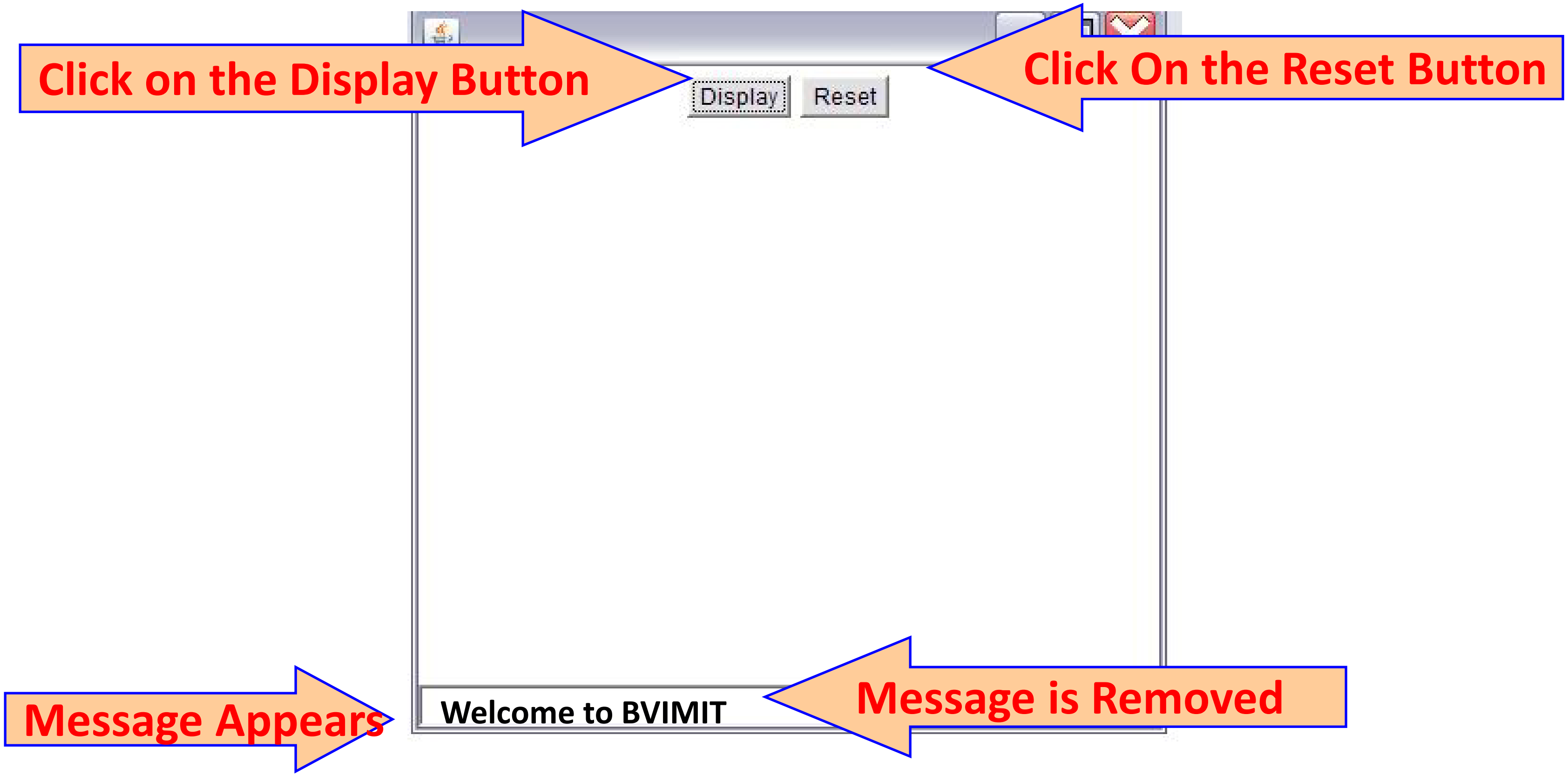
```
        b1.addActionListener(this);
        b2.addActionListener(this);
        p1.add(b1);
        p1.add(b2);
        add(p1,BorderLayout.NORTH);
        add(t1,BorderLayout.SOUTH);
        setSize(400,400);
        setVisible(true);
    }
```

Buttons registered with an ActionListener

Implementing the Action Listener

```
public void actionPerformed(ActionEvent e)
{         if(e.getSource()==b1)
          {         t1.setText(“Welcome to  BVIMIT”);
          }
          else
          {         t1.setText(“ “);
          }
}
public static void main(String s[])
{         new MyApplication();
}
}
```

Output of Example



Welcome to BVIMIT

Click on the Display Button

Click On the Reset Button

Message Appears

Message is Removed

Example

```
import java.awt.*;
import java.awt.event.*;
public class MyApplication extends Frame
   implements MouseListener
{  Button b1;
   TextField t1;
   Panel p1;
   MyApplication()

   {
        b1=new Button("Display");
        t1=new TextField(20);
        p1=new Panel();
```

Implementing MouseListener

```
        b1.addMouseListener(this);
        p1.add(b1);
        add(p1,BorderLayout.NORTH);
        add(t1,BorderLayout.SOUTH);
        setSize(400,400);
        setVisible(true);}
public void mouseEntered(MouseEvent e)
    {
                b1.setBackground(Color.RED);
    }
    public void mouseClicked(MouseEvent e)
    {           t1.setText("Welcome to IBM");
    }
```

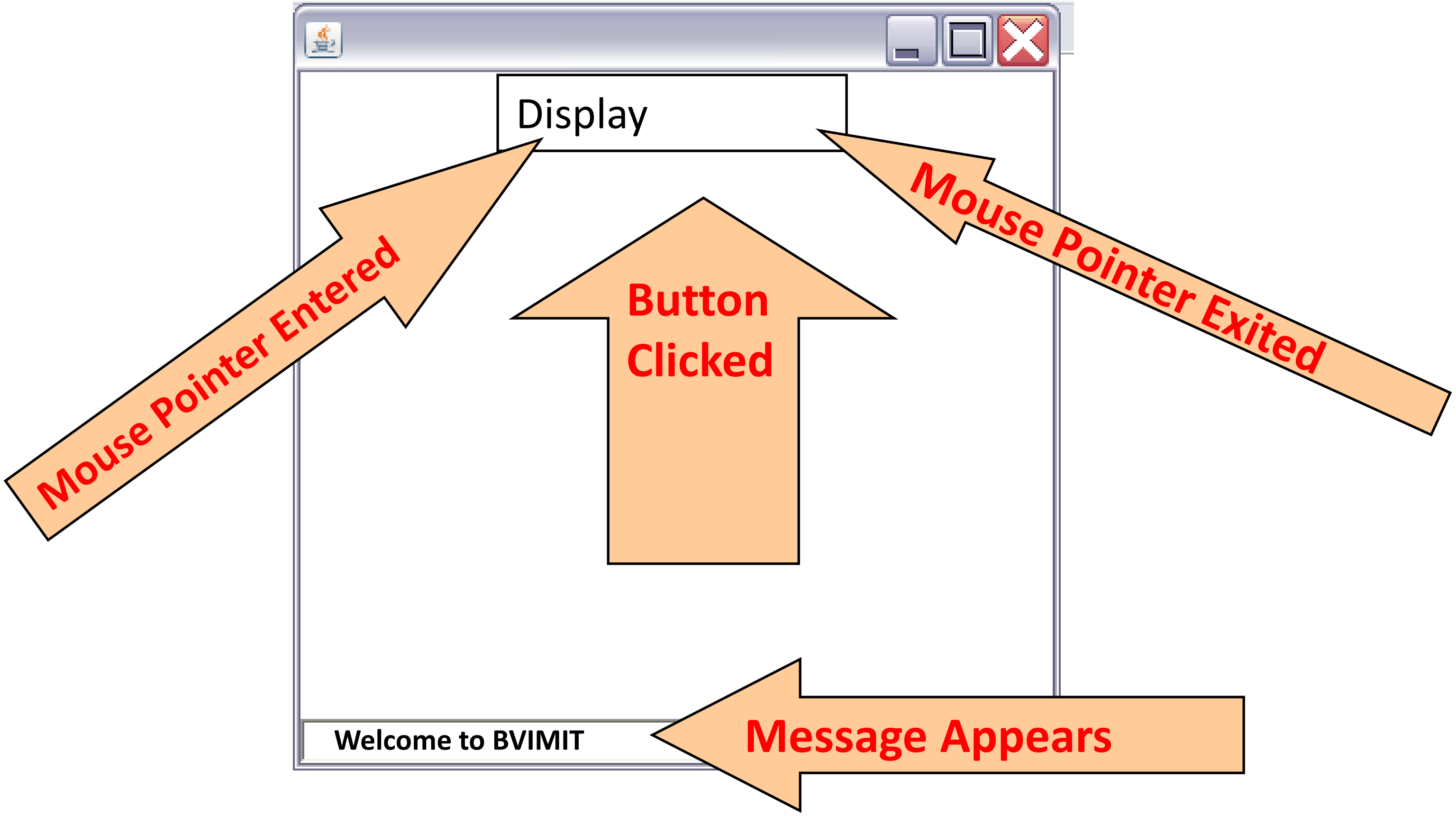
Button is Registered to a MouseListener

Implementing MouseListener

```
public void mouseExited(MouseEvent e)
   {
               b1.setBackground(Color.BLUE);
   }
   public void mousePressed(MouseEvent e)
   {}
   public void mouseReleased(MouseEvent e)
   {}

   public static void main(String s[])
   {
       new MyApplication();
   }}
```

Though the application does not need to respond to these activities, blank implementations must be provided

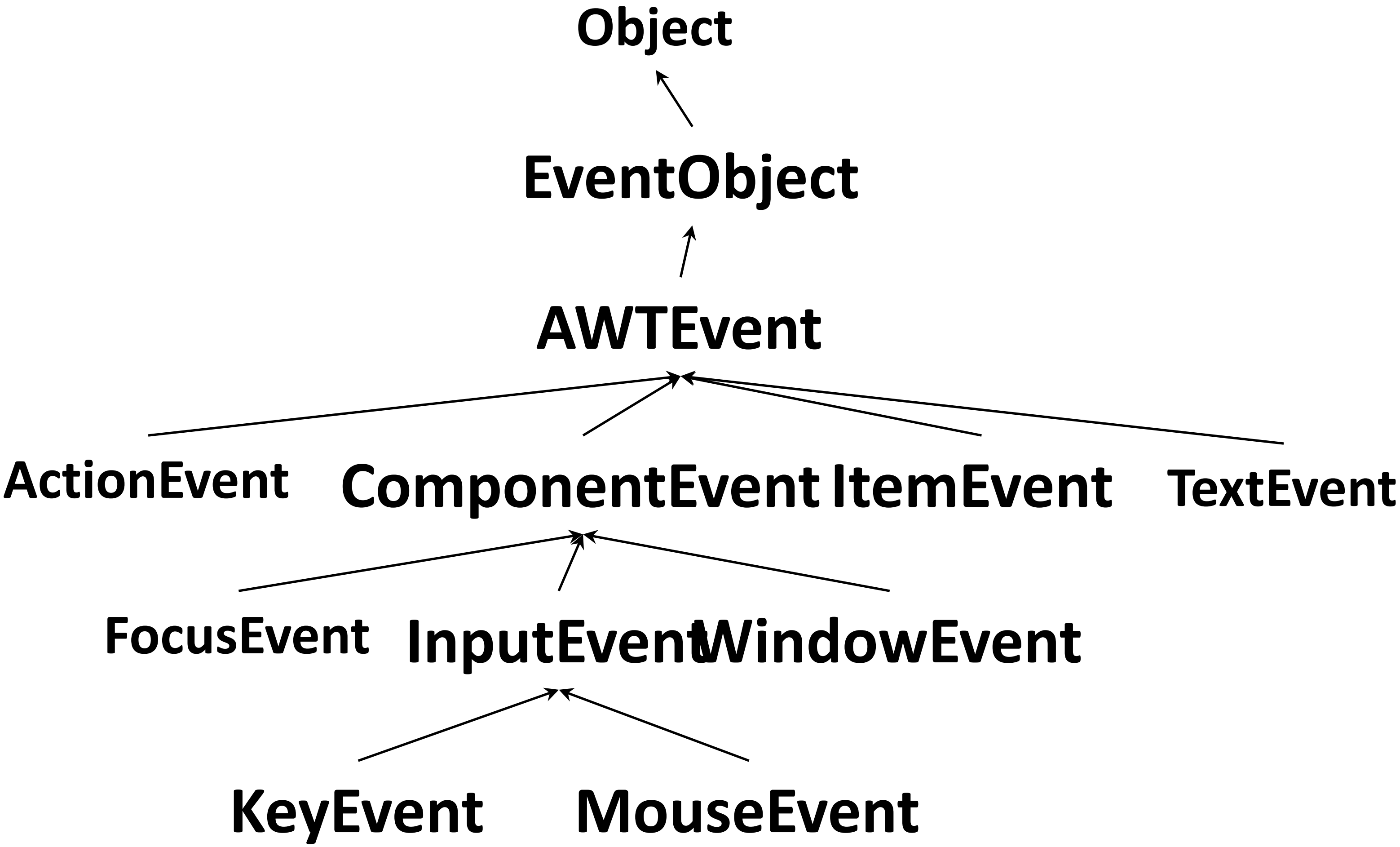
Output of the Example



Welcome to BVIMIT

Back

Event Classes Hierarchy



Event, listener interface and add- and remove-methods	Components supporting this event

ActionEvent
ActionListener ;
addActionListener( )
removeActionListener( )
	Button, List, TextField, MenuItem, CheckboxMenuItem, Menu and PopupMenu

AdjustmentEvent
AdjustmentListener ;
addAdjustmentListener( )
removeAdjustmentListener( )
	Scrollbar, Anything you create that implements Adjustable

ComponentEvent
ComponentListener
addComponentListener( )
removeComponentListener( )
	Component and its derivatives, including Button, Canvas, Checkbox, Choice, Container, Panel, Applet,
	ScrollPane,Window,Dialog,FileDialog,Fra

Event, listener interface and add-and remove-methods	Components supporting this event

ContainerEvent
ContainerListener
addContainerListener( )
removeContainerListener( )
	Container and its derivatives, including Panel, Applet, ScrollPane, Window, Dialog, FileDialog and Frame

FocusEvent
FocusListener
addFocusListener( )
removeFocusListener( )
	Component and its derivatives, including Button, Canvas, Checkbox, Choice, Container, Panel, Applet, ScrollPane, Window, Dialog, FileDialog, Frame Label, List, Scrollbar, TextArea and TextField

KeyEvent
KeyListener
addKeyListener( )
removeKeyListener( )
	Component and its derivatives, including Button, Canvas, Checkbox, Choice, Container, Panel, Applet, ScrollPane, Window,Dialog,FileDialog,Frame,Label, List, Scrollbar, TextArea and TextField

Event, listener interface and add-and remove-methods	Components supporting this event

MouseEvent (for both clicks and motion)
MouseListener;
addMouseListener( )
removeMouseListener( )
	Component and its derivatives, including Button, Canvas, Checkbox, Choice, Container, Panel, Applet, ScrollPane, Window,Dialog,FileDialog,Frame,Label, List, Scrollbar, TextArea and TextField MouseEvent (for both clicks and motion)

MouseMotionEvent
MouseMotionListener
addMouseMotionListener( )
removeMouseMotionListener( )
	Component and its derivatives, including Button, Canvas, Checkbox, Choice, Container, Panel, Applet, ScrollPane, Window,Dialog,FileDialog,Frame,Label, List, Scrollbar, TextArea and TextField

WindowEvent
WindowListener
addWindowListener( )
removeWindowListener( )
	Window and its derivatives, including Dialog, FileDialog, Frame, JFrame,

- Event type: ItemEvent
  - listener interface: ItemListener
  - add-and-remove-methods : addItemListener( ), removeItemListener( )
  - Components supporting this event : Checkbox, CheckboxMenuItem, Choice, List and anything that implements ItemSelectable.
- Event type: TextEvent
  - listener interface: TextListener
  - add-and-remove-methods : addTextListener( ), removeTextListener( )
  - Components supporting this event :Anything derived from TextComponent, including TextArea and TextField

Overview of Adapter Classes

- Adapter classes are used to reduce the code for Event Listeners
- avoids implementing all of the unneeded methods

All adapter classes are in:
java.awt.event package

- An adapter class provides an empty implementation of all methods in an event listener interface.
- are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

- For example, the **MouseMotionAdapter** class has two methods, **mouseDragged( )** & **mouseMoved( ).**
- The signatures of these empty methods are exactly as defined in the MouseMotionListener interface. If you were interested in only *mouse drag* events, then
- you could simply extend MouseMotionAdapter and implement mouseDragged( ).
- The empty implementation of mouseMoved( ) would handle the mouse motion events for you.

Adapter classes

- ComponentAdapter
- ContainerAdapter
- WindowAdapter
- MouseAdapter
- MouseMotionAdapter
- WindowAdapter
- FocusAdapter

MouseAdapter class

```
package java.awt.event;
 import java.awt.*;
 import java.awt.event.*;
 public class MouseAdapter implements MouseListener {
                public void mouseClicked(MouseEvent evt) {}
                public void mousePressed(MouseEvent evt) {}
                public void mouseReleased(MouseEvent evt) {}
                public void mouseEntered(MouseEvent evt) {}
                public void mouseExited(MouseEvent evt) {} }
```

Example using an Adapter class

```
import java.awt.*;
import java.awt.event.*;
public class MyApplication extends Frame
{
    Button b1;
    TextField t1;
    Panel p1;
    MyApplication()

    {
            b1=new Button("Display");
            t1=new TextField(20);
            p1=new Panel();
```

Example(contd)

```
          b1.addMouseListener(new HandleEvent())
          p1.add(b1);
          add(p1,BorderLayout.NORTH);
          add(t1,BorderLayout.SOUTH);
          setSize(400,400);
          setVisible(true);}
 public class HandleEvent extends MouseAdapter
{
          public void mouseEntered(MouseEvent e)
          {
                    b1.setBackground(Color.RED);
          }
```

Inner Class extending the Mouse Adapter

Example(Contd)

```
public void mouseClicked(MouseEvent e)
     {                              t1.setText("Welcome to IBM");
     }
public void mouseExited(MouseEvent e)``
     {
                                    b1.setBackground(Color.BLUE);
     }
}
public static void main(String s[])
     {
               new MyApplication();
     }
}
```

output

Nested Classes

- A class can be defined inside another class
- Benefits:
  - to structure and scope members
  - to connect logically related objects
- A nested class is considered a part of its enclosing class
- They share a trust relationship, i.e. everything is mutually accessible
- Nested types could be:
  - static – allows simple structuring of types
  - nonstatic – defines a special relationship between a nested object and an object of the enclosing class

Inner Class

- Class in the Class
  - Provide the method to define the object type to use in the class
  - Solve the class name conflict to restrict the reference scope of class
  - Information hiding

```
class OuterClass {
        // ...
        class InnerClass {
                // ...
        }
}
```

Inner Class

- Name Reference
  - OuterClass inside : Use InnerClass Simple name
  - OuterClass outside : OuterClass.InnerClass

```
public static void main(String[] args) {
        OuterClass  outObj = new  OuterClass();
       OuterClass.InnerClass   inObj = outObj.new  InnerClass();
}
```

- Access Modifier
  - public, private, protected

Inner class cannot have static variable

Inner classes

```
// This applet does NOT use an inner class.
import java.applet.*;
import java.awt.event.*;
/*
<applet code="MousePressedDemo" width=200
    height=100>
</applet>
*/
public class MousePressedDemo extends Applet {
public void init() {
addMouseListener(new MyMouseAdapter(this));
}
}
class MyMouseAdapter extends MouseAdapter {
MousePressedDemo mousePressedDemo;
public MyMouseAdapter(MousePressedDemo
    mousePressedDemo) {
this.mousePressedDemo = mousePressedDemo;
}
public void mousePressed(MouseEvent me) {
mousePressedDemo.showStatus("Mouse
    Pressed.");
}
}
```

```
// Inner class demo.
import java.applet.*;
import java.awt.event.*;
/*<applet code="InnerClassDemo"
    width=200 height=100>
</applet>*/
public class InnerClassDemo extends Applet
    {
public void init() {
addMouseListener(new
    MyMouseAdapter());
}
class MyMouseAdapter extends
    MouseAdapter {
public void mousePressed(MouseEvent me)
    {
showStatus("Mouse Pressed");
}
}
}
```

Anonymous Inner Classes

- An *anonymous inner class is one that is not assigned a name.*
- an anonymous inner class can facilitate the writing of event handlers.
- Consider the applet shown in the following listing. As before, its goal is to display the string “Mouse Pressed”
- in the status bar of the applet viewer or browser when the mouse is pressed
- There is one top-level class in this program:**AnonymousInnerClassDemo. The init( )** method calls the **addMouseListener( ) method.**
- The syntax **new MouseAdapter( ) { ... } indicates to the compiler that the code** between the braces defines an anonymous inner class.

```
// Anonymous inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
<applet code="AnonymousInnerClassDemo"
    width=200 height=100>
</applet>
*/
public class AnonymousInnerClassDemo extends
    Applet {
public void init() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
showStatus("Mouse Pressed");
}
});
}
}
```

Database Programming

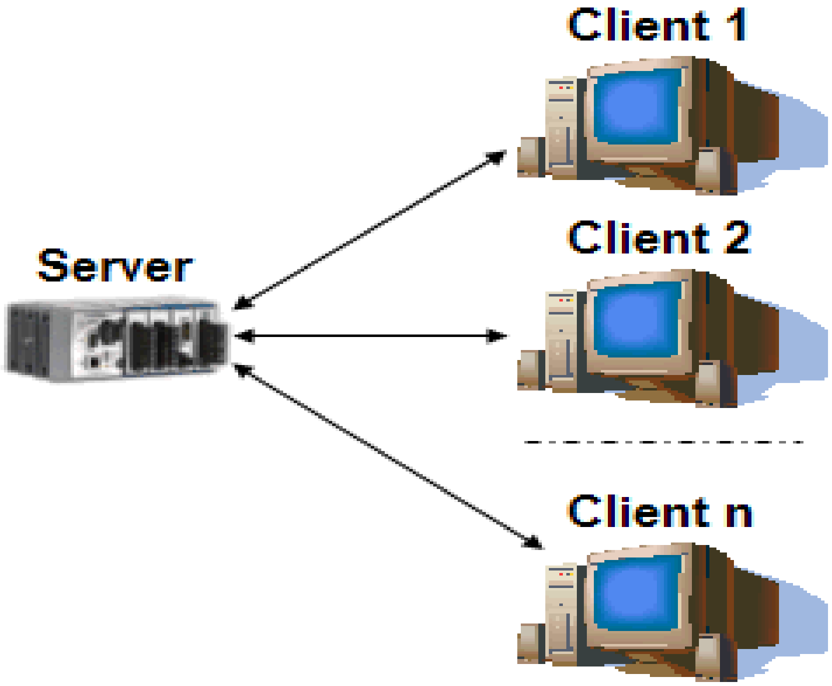
- **JDBC:** Introduction
- JDBC Architecture
- Types of Drivers
- Statement
- ResultSet, Read Only ResultSet, Updatable ResultSet, Forward Only ResultSet, Scrollable ResultSet
- PreparedStatement, Connection Modes, SavePoint, Batch Updations
- CallableStatement, BLOB & CLOB

Database Architectures

- **Two-tier**
- **Three-tier**
- **N-tier**

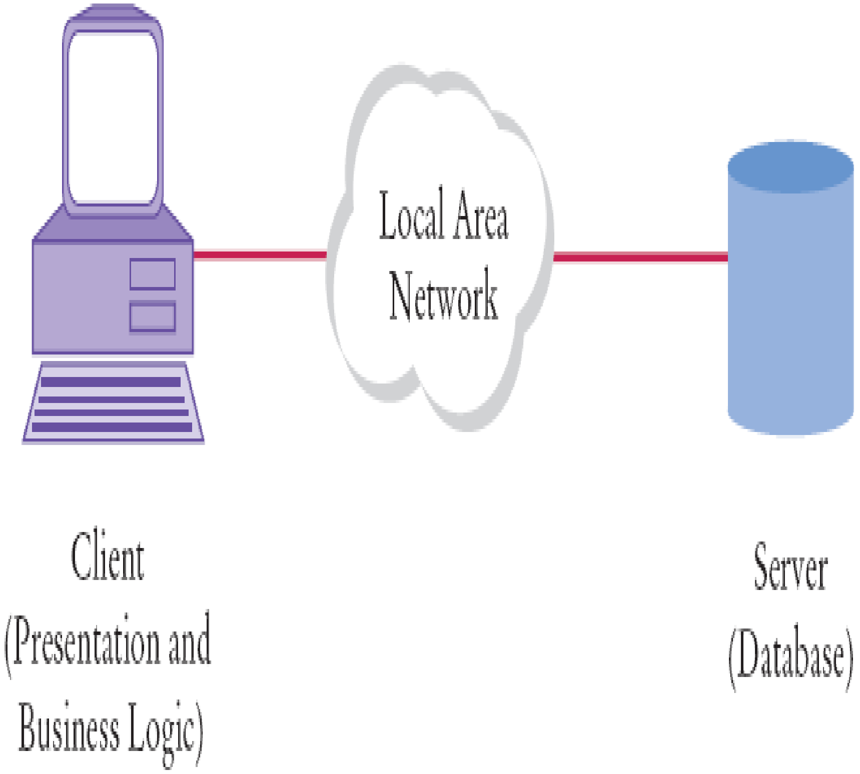
Two-Tier Architecture

- **Client connects directly to server**
- **e.g. HTTP, email**



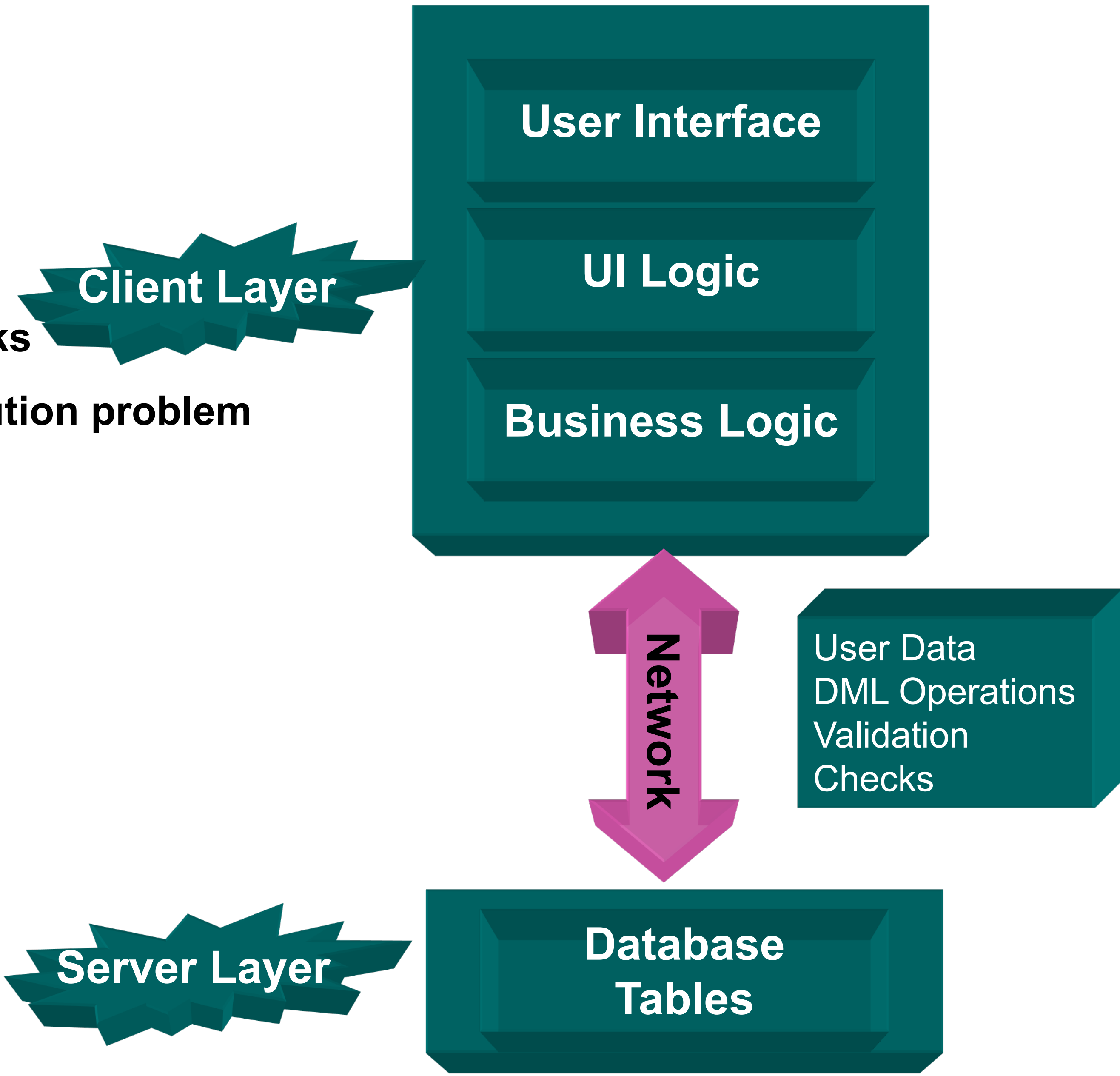
Two-Tier Pros

- **Simple**
- **Client-side scripting offloads work onto the client**



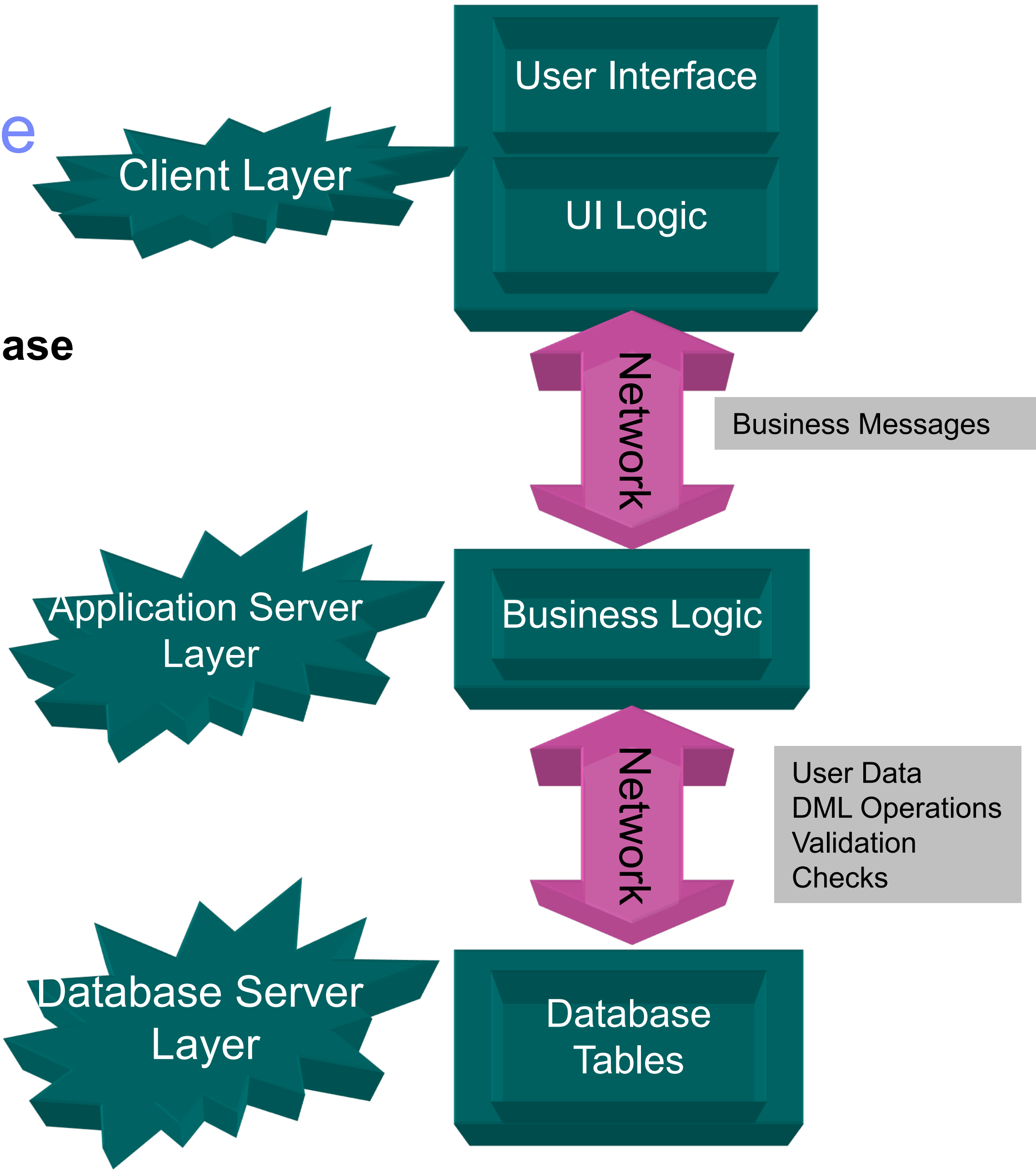
Two-Tier Cons

- **Fat client**
- **Server bottlenecks**
- **Software Distribution problem**
- **Inflexible**



Three-Tier Architecture

- **Application Server sits between client and database**



Client Layer

User Interface

UI Logic

Network

Business Messages

Application Server Layer

Business Logic

Network

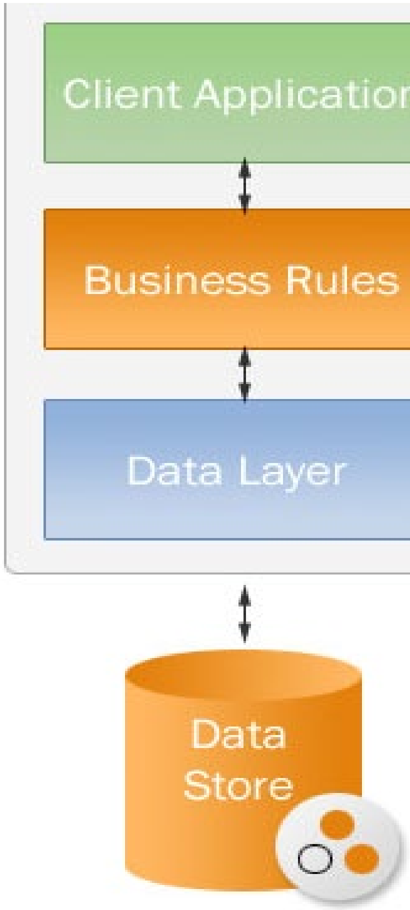
User Data
DML Operations
Validation
Checks

Database Server Layer

Database Tables

Three-Tier Pros

- **flexible: can change one part without affecting others**
- **can connect to different databases without changing code**
- **specialization: presentation / business logic / data management**
- **can cache queries**

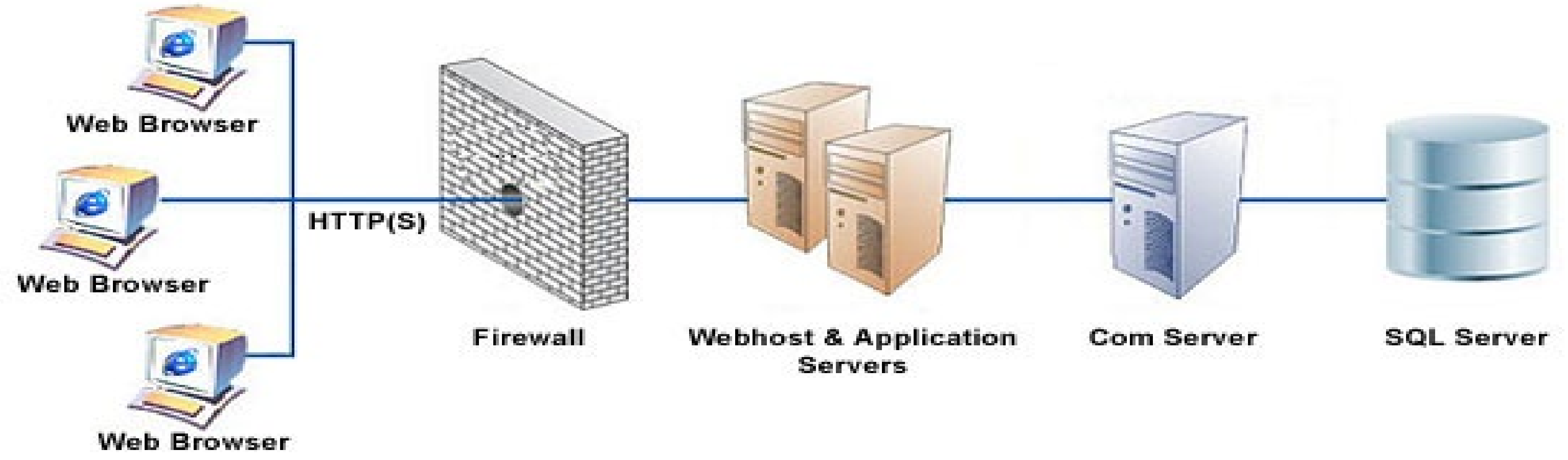


Three-Tier Cons

- **higher complexity**
- **higher maintenance**
- **lower network efficiency**
- **more parts to configure (and buy)**

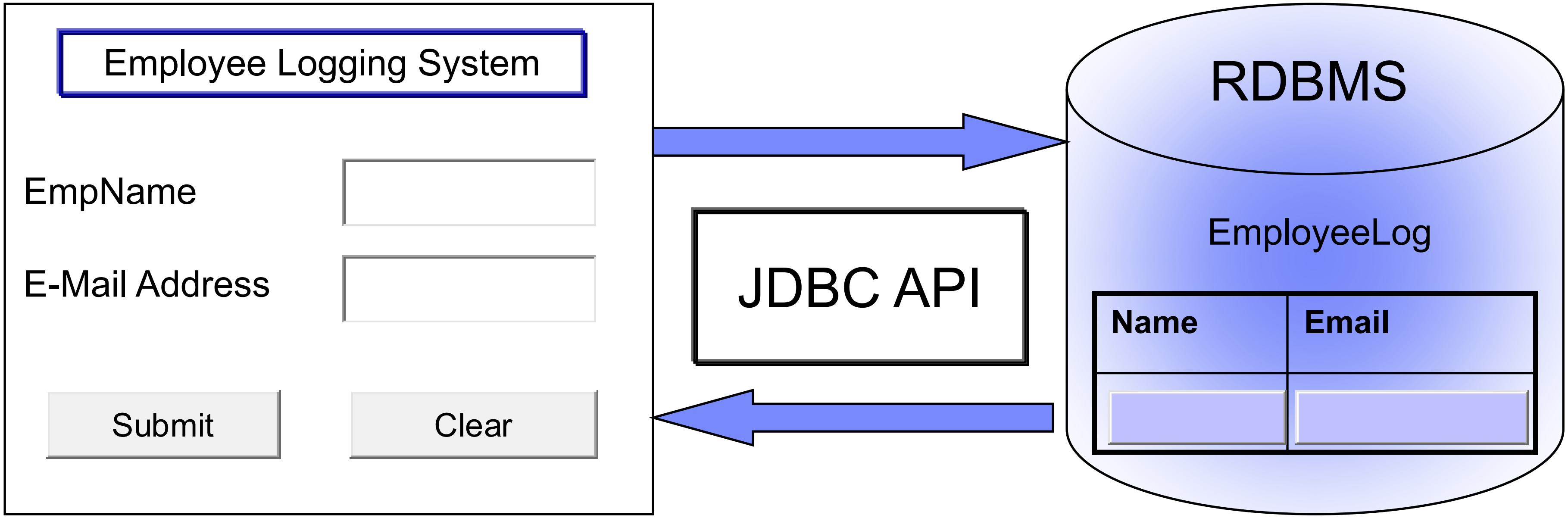
N-Tier Architecture

- **Design your application using as many “tiers” as you need**
- **Use Object-Oriented Design techniques**
- **Put the various components on whatever host makes sense**
- **Java allows N-Tier Architecture, especially with RMI and JDBC**



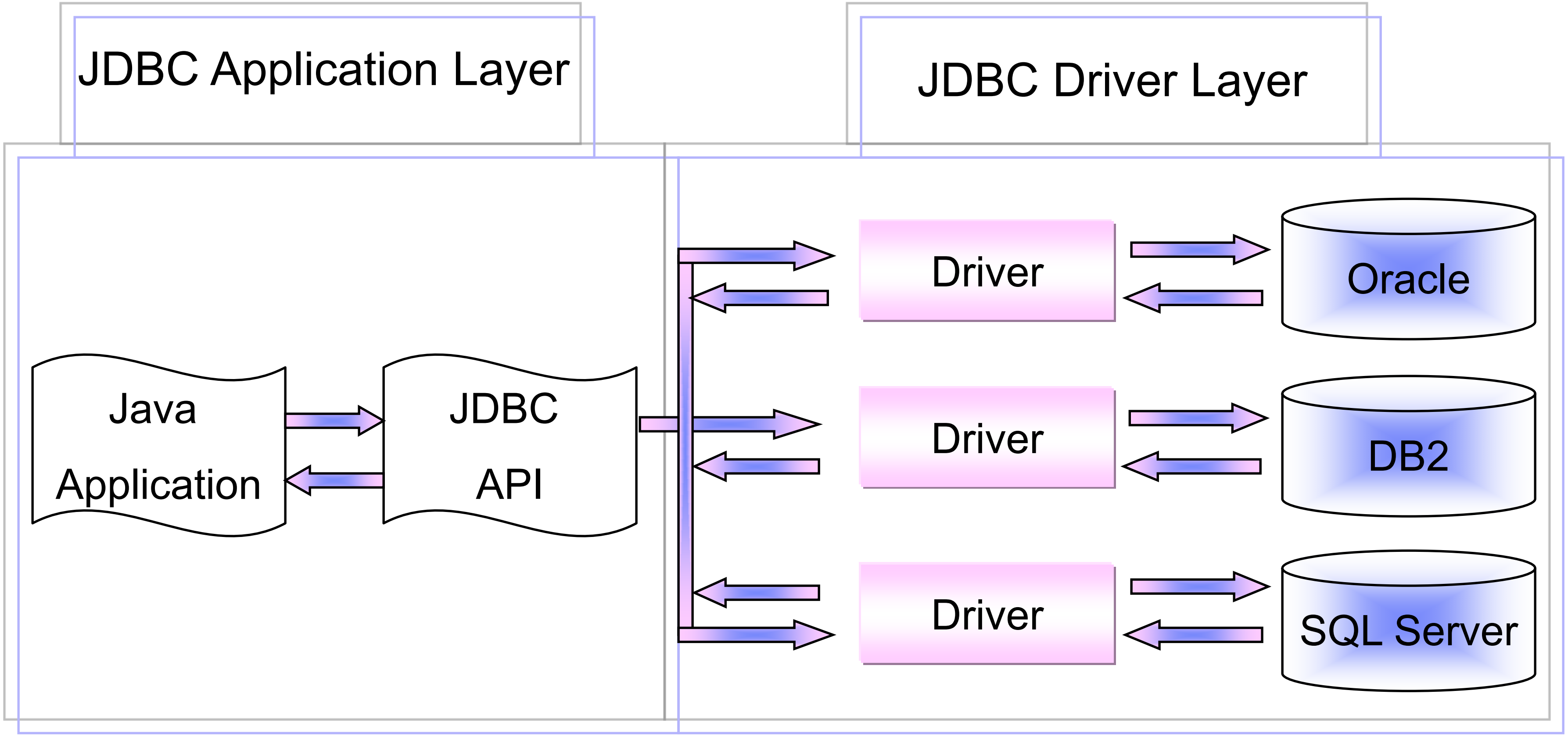
Java Application $\xleftrightarrow{\text{Connects to}}$ Database

- The below given figure shows the Employee Logging System application developed in Java interacting with the Employee database using the JDBC API:

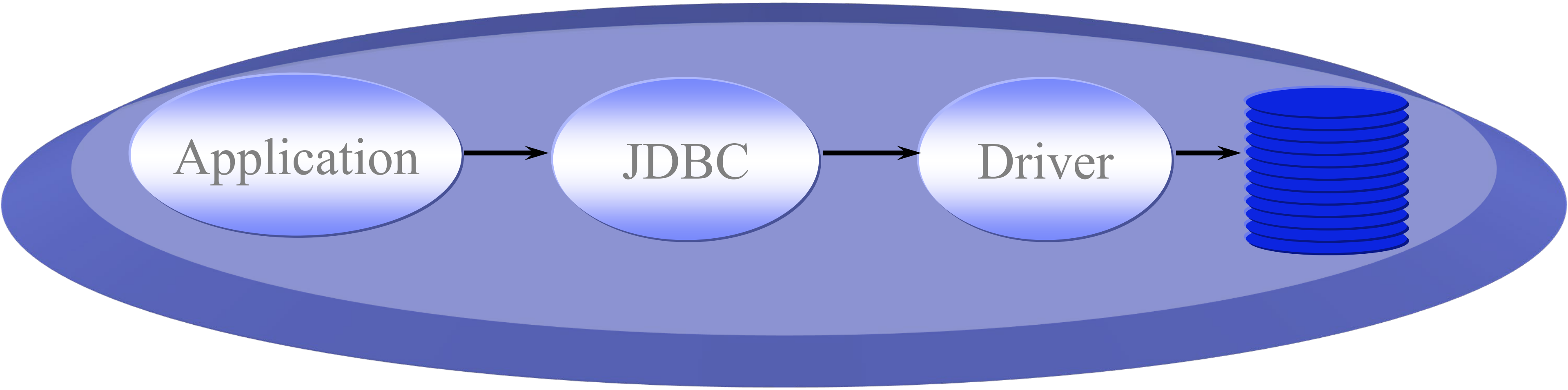


JDBC Architecture

- It can be categorized into into two layers:



JDBC Architecture (Continued)



- **Java code calls JDBC library**
- **JDBC loads a *driver***
- **Driver talks to a particular database**
- **Can have more than one driver -> more than one database**
- **Can change database engines without changing any application code**

JDBC Drivers

- **Type I: “Bridge” -** JDBC-ODBC Bridge Driver
- **Type II: “Native” -** Native-API Partly-Java Driver
- **Type III: “Middleware” -** JDBC-Net Pure-Java Driver
- **Type IV: “Pure” -** Native Protocol Pure-Java Driver

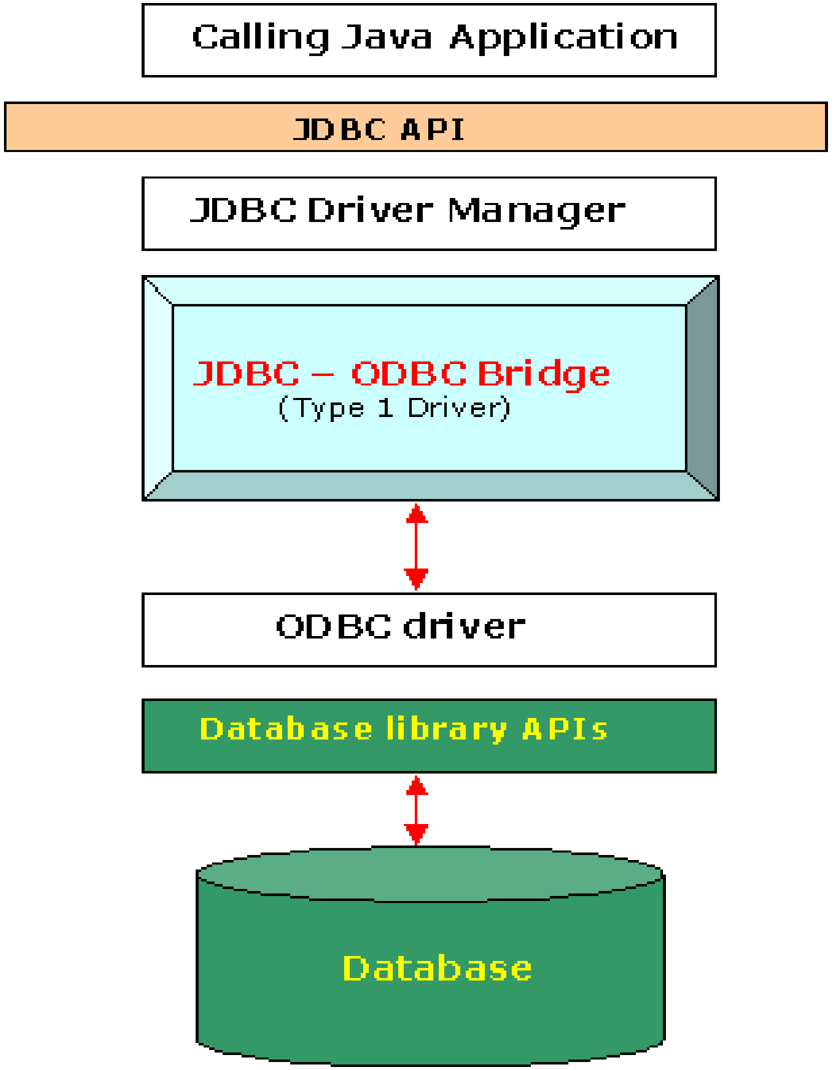
Overview of All Drivers

Type I Drivers

- **Use bridging technology**
- **Translates query obtained by JDBC into corresponding ODBC query, which is then handled by the ODBC driver.**
- **Almost any database for which ODBC driver is installed, can be accessed.**

Go Back To

JDBC Driver List



Disadvantage of Type-I Driver

- **Performance overhead since the calls have to go through the JDBC overhead bridge to the ODBC driver, then to the native db connectivity interface.**
- **The ODBC driver needs to be installed on the client machine.**
- **Not good for Web**

Go Back To

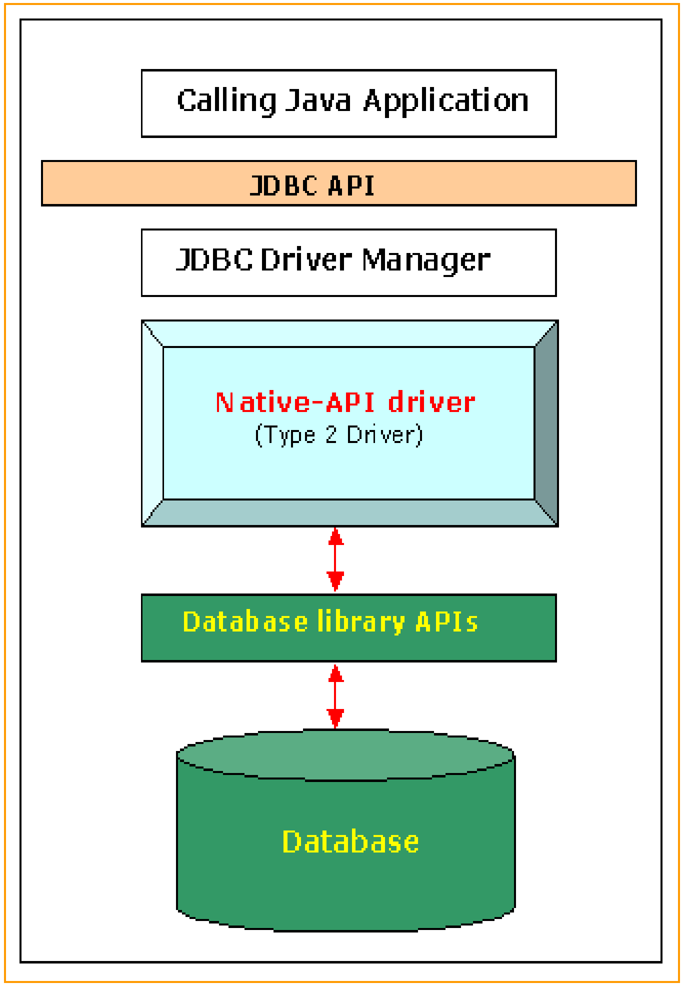
JDBC Driver List

Type II Drivers

- **Native API drivers**
- **Better performance than Type 1 since no jdbc to odbc translation is needed.**
- **Converts JDBC calls into calls to the client API for that database.**

Go Back To

JDBC Driver List



Disadvantage of Type-II Driver

- **The vendor client library needs to be installed on the client machine.**
- **Cannot be used in internet due the client side software needed.**
- **The driver is compiled for use with the particular operating system.**
- **Mostly obsolete now**
- **Not good for Web**

Go Back To

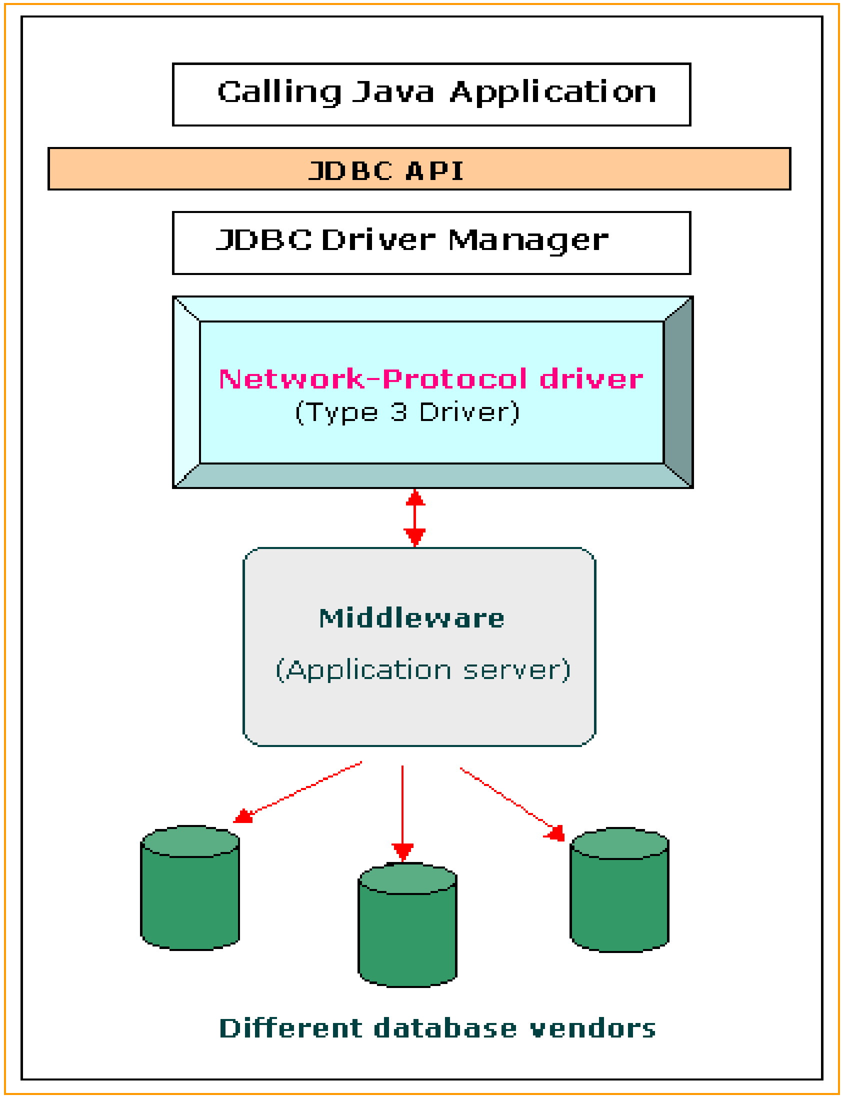
JDBC Driver List

Type III Drivers

- **Follows a three tier communication approach.**
- **Calls middleware server, usually on database host**
- **Very flexible -- allows access to multiple databases using one driver**
- **Only need to download one driver**

Go Back To

JDBC Driver List



Disadvantage of Type-III Driver

- **Requires database-specific coding to be done in the middle tier.**

- **An extra layer added may result in a time-bottleneck.**

Go Back To

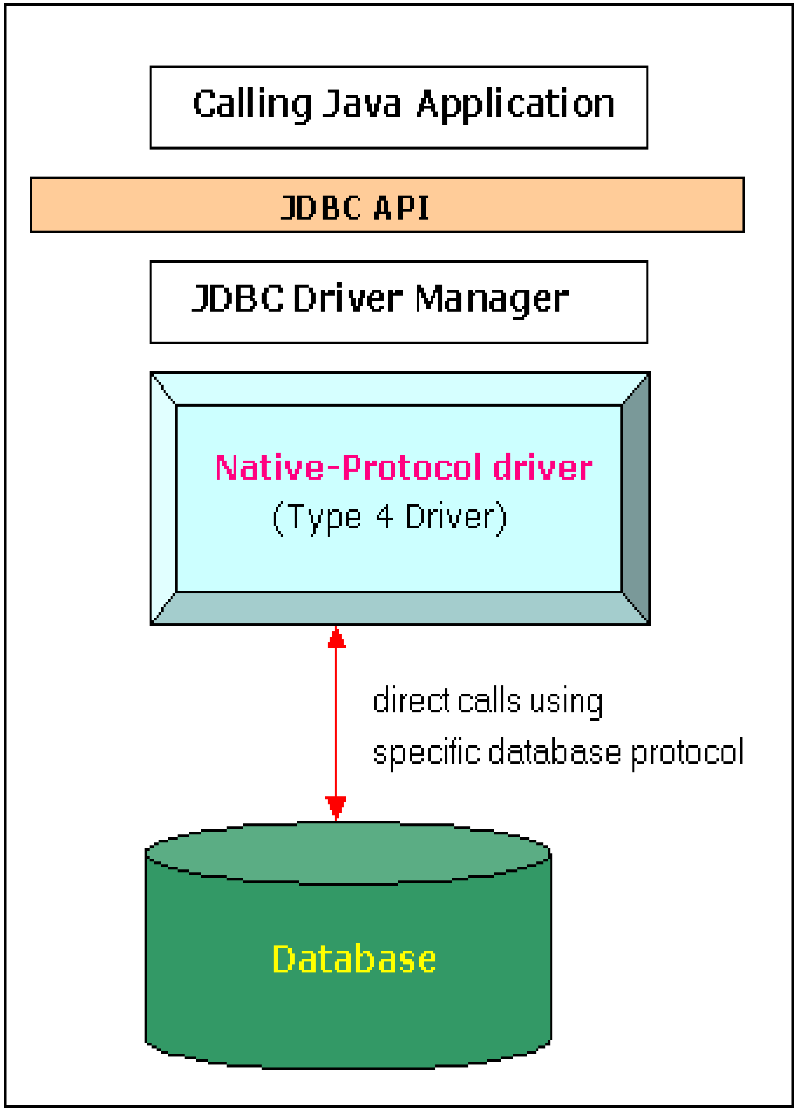
JDBC Driver List

Type IV Drivers

- **100% Pure Java -- the Holy Grail**
- **Communicate directly with a vendor's database through socket connection**
- **Use Java networking libraries to talk directly to database engines**
- **e.g include the widely used Oracle thin driver - oracle.jdbc.driver. OracleDriver**

Go Back To

JDBC Driver List



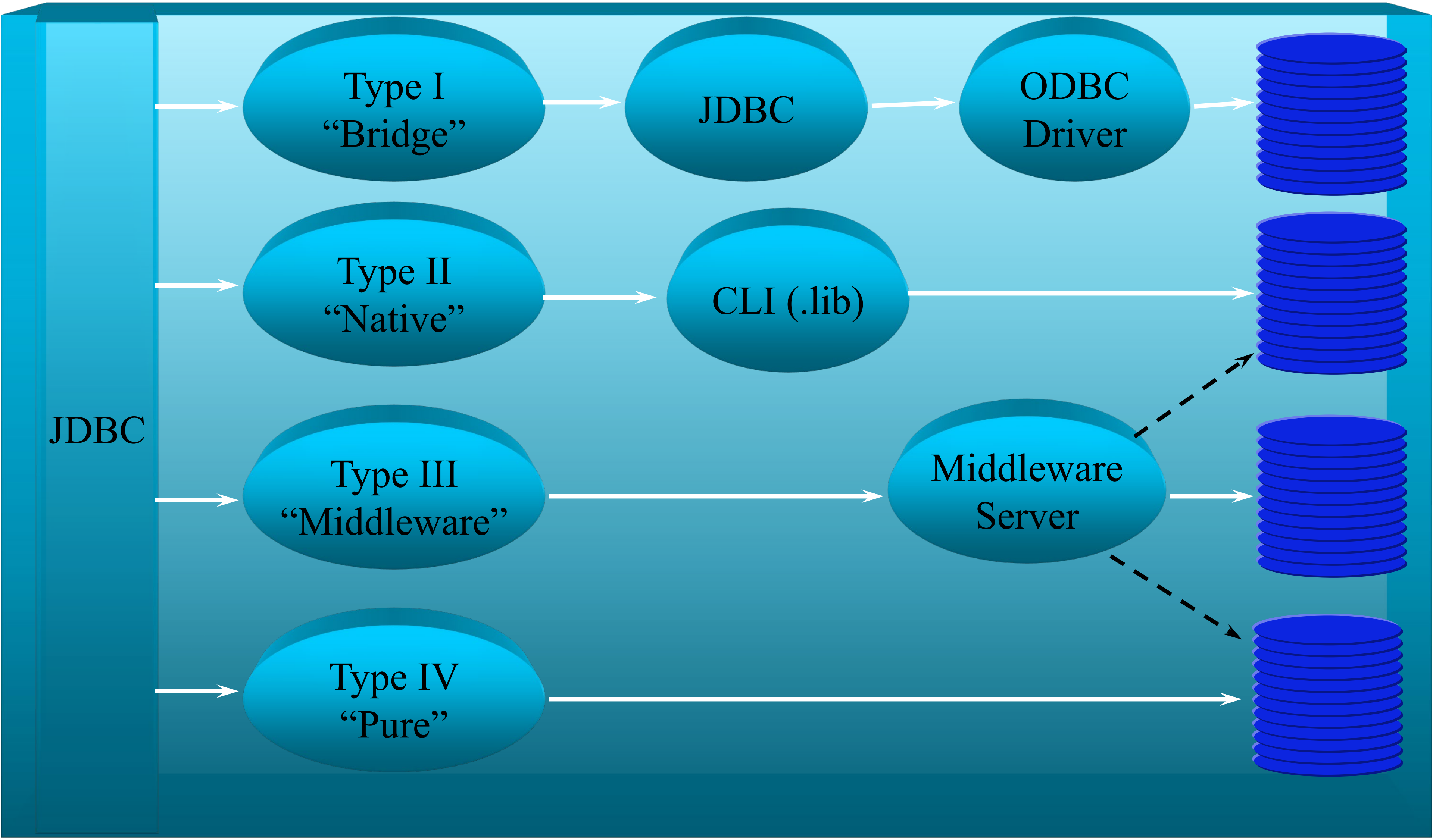
Disadvantage of Type-IV Driver

- **At client side, a separate driver is needed for each database**

Go Back To

JDBC Driver List

JDBC Drivers (Fig.)



Related Technologies

- **ODBC**
  - ✓ Requires configuration (odbc.ini)
- **RDO, ADO**
  - ✓ Requires Win32
- **JavaBlend**
  - ✓ maps objects to tables transparently (more or less)

JDBC API

JDBC API

- The JDBC API classes and interfaces are available in the java.sql and the javax.sql packages.

- The commonly used classes and interfaces in the JDBC API are:

  - ✓ **DriverManager class**: Loads the driver for a database.

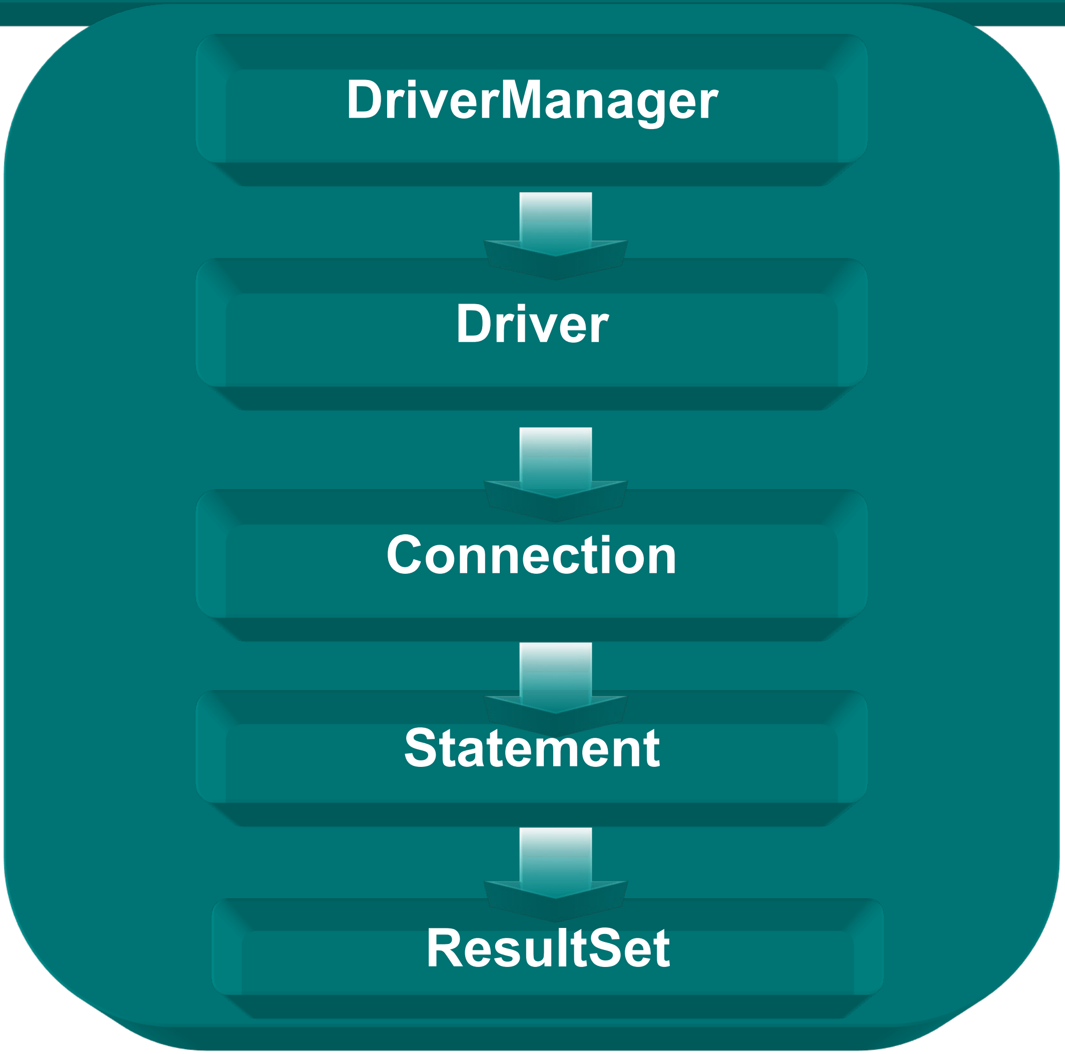
  - ✓ **Driver interface**: Represents a database driver. All JDBC driver classes must implement the Driver interface.

  - ✓ **Connection interface**: Enables you to establish a connection between a Java application and a database.

JDBC API (Continued)

- ✓ **Statement interface**: Enables you to execute SQL statements.
- ✓ **ResultSet interface**: Represents the information retrieved from a database.
- ✓ **SQLException class**: Provides information about the *exceptions* that occur while interacting with databases.

Steps to create JDBC Application



DriverManager

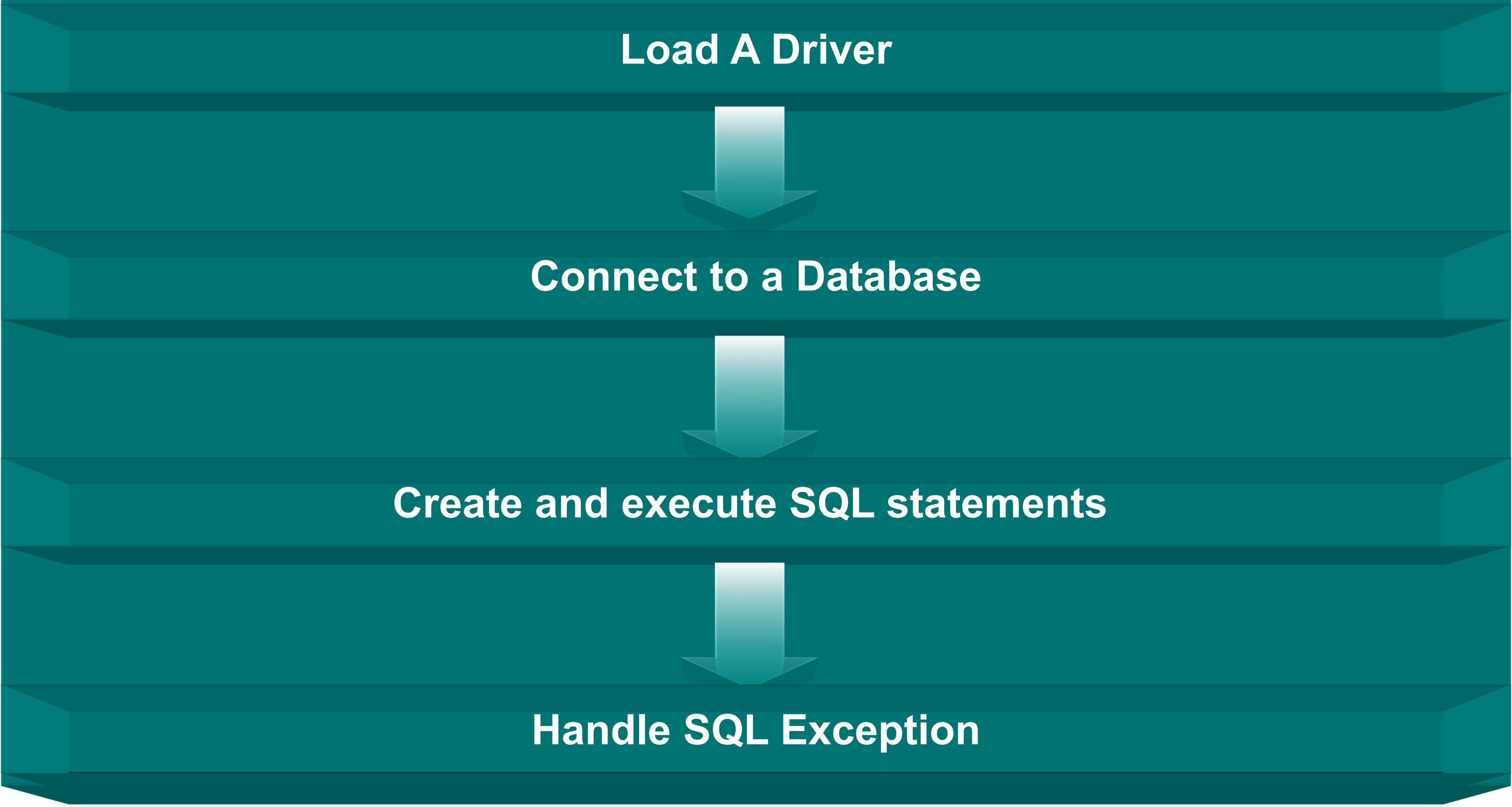
Driver

Connection

Statement

ResultSet

Steps to create JDBC Application (Continued)



Load A Driver

Connect to a Database

Create and execute SQL statements

Handle SQL Exception

JDBC API (Continued)

Load A Driver

- Loading a Driver can be done in two ways:
  - Programmatically:
    - ✓ Using the forName() method
    - ✓ Using the registerDriver()method
  - Manually:
    - ✓ By setting system property

JDBC API (Continued)

Load A Driver (Programmatically)

- Using the forName() method
  - ✓ The forName() method is available in the java.lang.Class class.
  - ✓ The forName() method loads the JDBC driver and registers the driver with the driver manager.
  - ✓ The method call to use the the forName() method is:
  - ✓ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

JDBC API (Continued)

Load A Driver (Programmatically)

- Using the registerDriver() method
  - ✓ You can create an instance of the Driver class to load a JDBC driver.
  - ✓ This instance provide the name of the driver class at run time.
  - ✓ The statement to create an instance of the Driver class is:

    Driver d = new sun.jdbc.odbc.JdbcOdbcDriver();
  - ✓ You need to call the registerDriver() method to register the Driver object with the DriverManager.
  - ✓ The method call to register the JDBC-ODBC Bridge driver is:

    DriverManager.registerDriver(d);

JDBC API (Continued)

Load A Driver (Manually)

- Setting System Property
  - ✓ To load a JDBC driver, add driver name to the jdbc.drivers system property.
  - ✓ Use the –D command line option to set the system property on the command line.
  - ✓ To set the system property the command is:

    ```
    java –D jdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver IBMApp
    ```

JDBC API (Continued)

Connect to a Database

- Connecting to a Database Using DriverManager.getConnection() method:
  - *Connection getConnection (String <url>)*
  - *Connection getConnection (String <url>, String <username>, String <password>)*
    - ✓ Connects to given JDBC URL.
    - ✓ throws java.sql.SQLException
    - ✓ Returns a connection object.

Example:

Connection con=DriverManager.getConnection(“jdbc:odbc:MyDSN”,”scott”,”tiger”);

JDBC API (Continued)

Connect to a Database (Example)

```
String url   = "jdbc:odbc:Northwind";
try {
  Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection(url);
}
catch (ClassNotFoundException e)
  { e.printStackTrace(); }
catch (SQLException e)
  { e.printStackTrace(); }
```

JDBC API (Continued)

Create and Execute SQL Statements

- **The Connection object provides the createStatement() method to create a Statement object.**

`Statement createStatement()`

- ✓ returns a new Statement object

`PreparedStatement prepareStatement(String sql)`

- ✓ returns a new PreparedStatement object

`CallableStatement prepareCall(String sql)`

- ✓ returns a new CallableStatement object

JDBC API (Continued)

Statement Interface

- **A Statement object is used for executing a static SQL statement and obtaining the results produced by it.**

JDBC API (Continued)

Statement Interface Methods

ResultSet executeQuery(String)

- ✓ Execute a SQL statement that returns a single ResultSet.

int executeUpdate(String)

- ✓ Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.

boolean execute(String)

- ✓ Execute a SQL statement that may return multiple results.

JDBC API (Continued)

ResultSet Interface

- **A ResultSet provides access to a table of data generated by executing a Statement.**
- **Only one ResultSet per Statement can be open at once.**
- **The table rows are retrieved in sequence.**
- **A ResultSet maintains a cursor pointing to its current row of data.**
- **The 'next' method moves the cursor to the next row.**
  - ✓ you can’t rewind

JDBC API (Continued)

ResultSet Methods

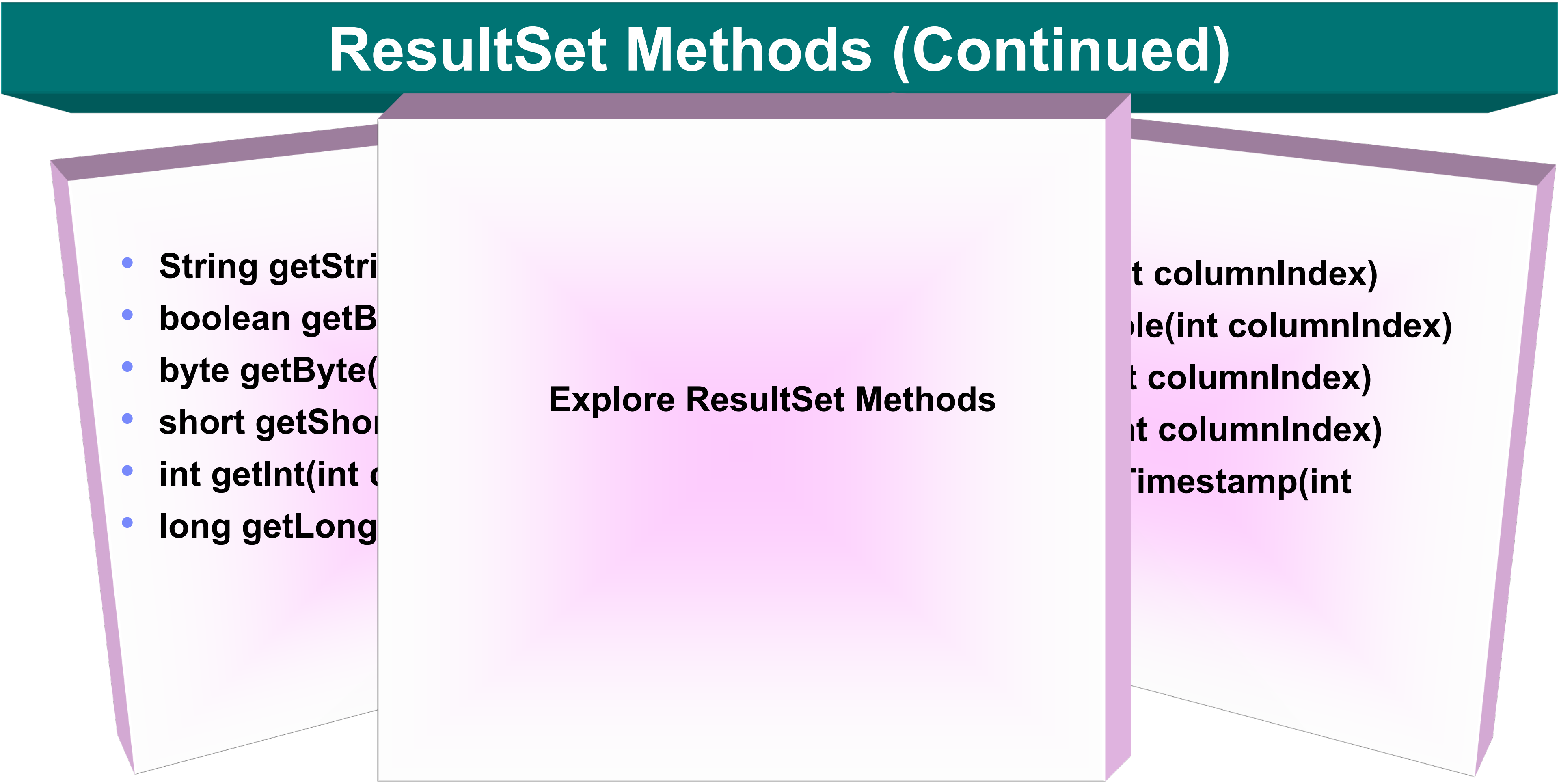
- **boolean next()**
  - ✓activates the next row
  - ✓the first call to next() activates the first row
  - ✓returns false if there are no more rows
- **void close()**
  - ✓disposes of the ResultSet
  - ✓allows you to re-use the Statement that created it

JDBC API (Continued)

ResultSet Methods (Continued)

- *Type* get*Type*(int columnIndex)
  - ✓returns the given field as the given type
  - ✓fields indexed starting at 1 (not 0)
- *Type* get*Type*(String columnName)
  - ✓same, but uses name of field
  - ✓less efficient
- int findColumn(String columnName)
  - ✓looks up column index given column name

JDBC API (Continued)



ResultSet Methods (Continued)

- String getStri
- boolean getB
- byte getByte(
- short getShor
- int getInt(int c
- long getLong

Explore ResultSet Methods

t columnIndex)
ble(int columnIndex)
t columnIndex)
nt columnIndex)
Timestamp(int

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
1. boolean first()	1. Shifts the control of a result set cursor to the first row of the result set.
2. boolean isFirst()	2. checks whether result set cursor points to the first row or not.
3. boolean beforeFirst()	3. moves the cursor before the first row.
4. boolean isbeforeFirst()	4. Checks whether result set cursor moves before the first row.

JDBC API (Continued)

ResultSet Methods (Continued)

	Method Name		Description
5.	boolean last()	5.	Shifts the control to the last row of result set cursor.
6.	boolean isLast()	6.	checks whether result set cursor points to the last row or not.
7.	boolean afterLast()	7.	moves the cursor after the last row.
8.	boolean isAfterLast()	8.	Checks whether result set cursor moves after the last row.

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
9. boolean next()	9. Shifts the control to the next row of result set.
10. boolean previous()	10. Shifts the control to the previous row of the result set.
11. boolean absolute(int rowno)	11. Shifts the cursor to the row number that you specify as an argument.
12. boolean relative(int rowno)	12. Shifts the cursor relative to the row number that you specify as an argument.

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
13. void insertRow()	13. Inserts a row in the current result set.
14. void deleteRow()	14. Deletes a row in the current result set.
15. void updateRow()	15. Updates a row of the current resultset.

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
16. void updateString(col name, String s)	16. Updates the specified column name with the given string value.
17. void updateInt(col name, int x)	17. Updates the specified column name with the given int value.
18. void updateFloat()	18. Updates the specified column name with the given float value.
19. void cancelRowUpdates()	19. Cancels all of the updates in a row.

JDBC API (Continued)

ResultSet Fields

Field Name	Description
1. TYPE_FORWARD_ONLY	1. The ResultSet object can moves forward only from first to last row.
2. TYPE_SCROLL_SENSITIVE	2. Indicates ResultSet is scrollable and it reflects changes in the data made by other user.
3. TYPE_SCROLL_INSENSITIVE	3. Indicates ResultSet is scrollable and does not reflect changes in the data made by other user.

JDBC API (Continued)

ResultSet Fields

Field Name	Description
4. CONCUR_READ_ONLY	4. Does not allow to update the ResultSet object.
5. CONCUR_UPDATABLE	5. Allows to update the ResultSet object .

JDBC API (Continued)

SQL Syntax

INSERT INTO *table* (*field1, field2*) VALUES (*value1, value2*)

- ✓ inserts a new record into the named table

UPDATE *table* SET (*field1* = *value1, field2* = *value2*) WHERE *condition*

- ✓ changes an existing record or records

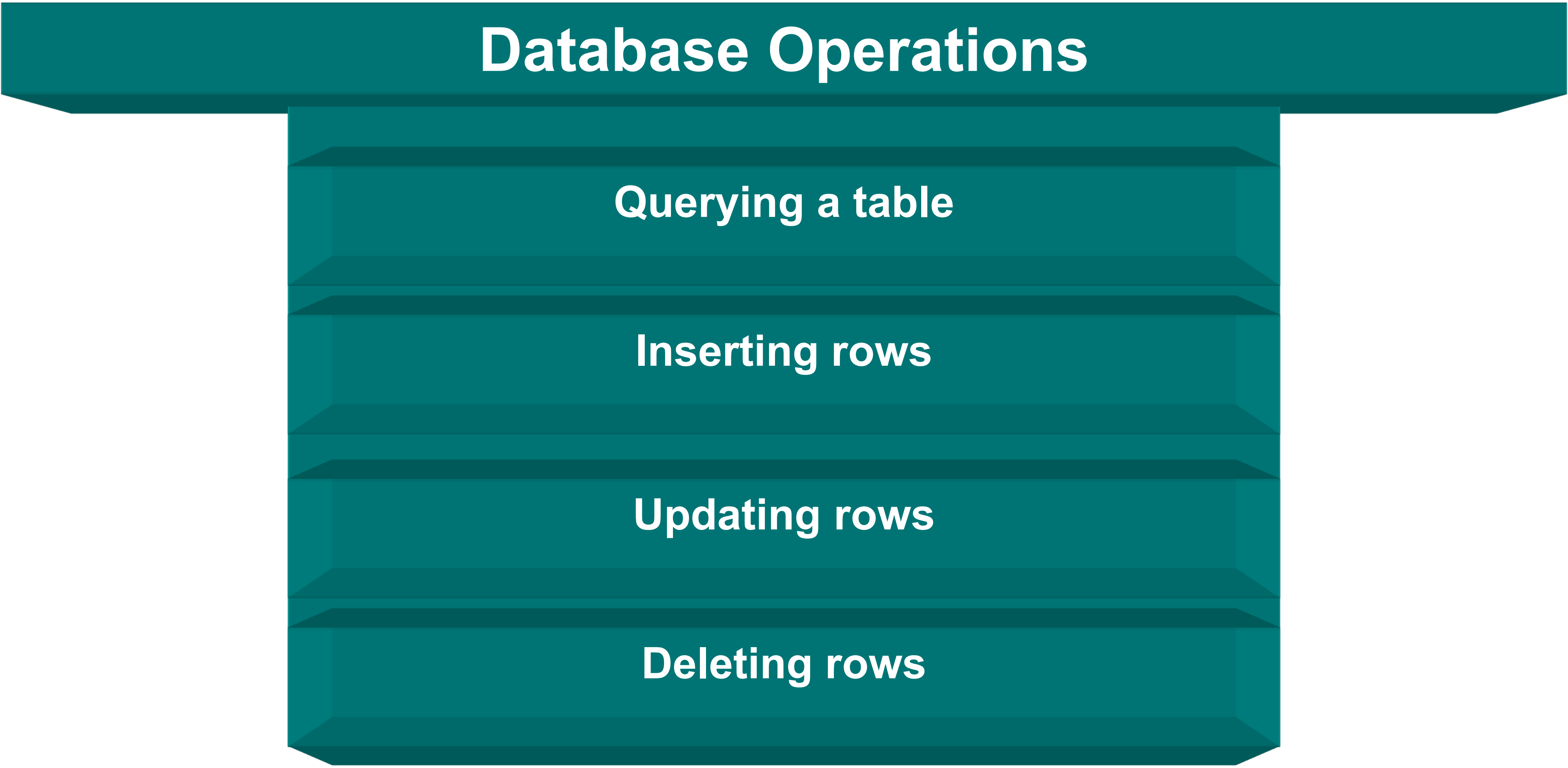
DELETE FROM *table* WHERE *condition*

- ✓ removes all records that match condition

SELECT *field1, field2* FROM *table* WHERE *condition*

- ✓ retrieves all records that match condition

JDBC API (Continued)



Database Operations

Querying a table

Inserting rows

Updating rows

Deleting rows

JDBC API (Continued)

Database Operations

Querying a table

The code snippet to retrieve data from the employees table is:

```
String semp = "SELECT * FROM employees";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(semp);
```

JDBC API (Continued)

Database Operations

Inserting rows

The code snippet to insert rows in employees table is:

```
String semp = “INSERT INTO employees(eid, ename,
basic)  VALUES(1,’A.Sinha’,28000)”;
Statement stmt = con.createStatement();
int noOfInsert = stmt.executeUpdate(semp);
```

JDBC API (Continued)

Database Operations

Updating rows

The code snippet to insert rows in employees table is:

```
String semp = “UPDATE employees SET
basic=basic+2000 where eid=1”;
Statement stmt = con.createStatement();
int noOfUpdate = stmt.executeUpdate(semp);
```

JDBC API (Continued)

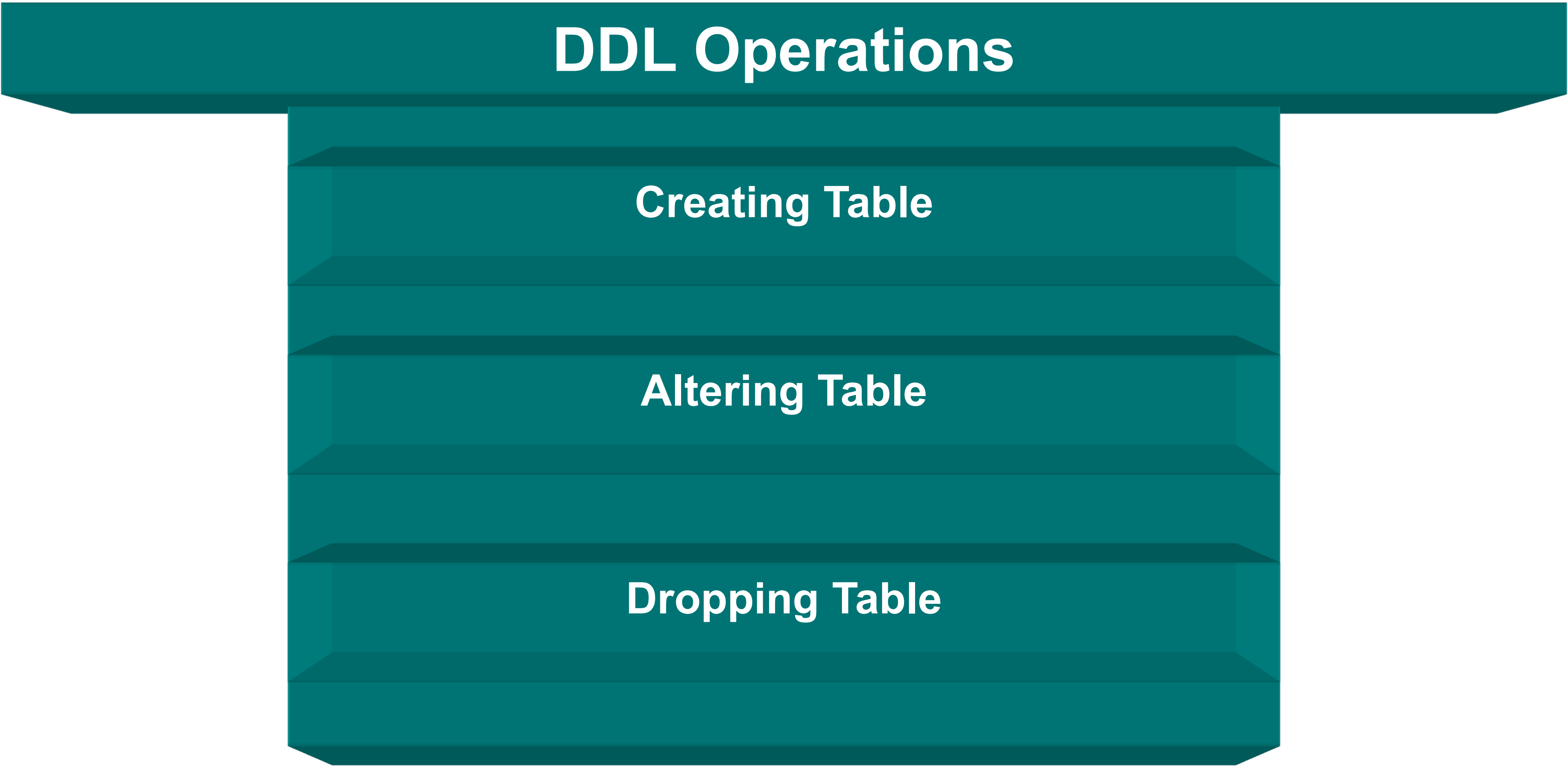
Database Operations

Deleting rows

The code snippet to delete rows in employees table is:

```
String semp = “DELETE FROM employees WHERE
eid=1”;
Statement stmt = con.createStatement();
int noOfDelete = stmt.executeUpdate(semp);
```

JDBC API (Continued)



DDL Operations

Creating Table

Altering Table

Dropping Table

JDBC API (Continued)

DDL Operations

Creating Table

The code snippet to create a *department* table is:

```
Statement stmt = con.createStatement();
stmt.execute(“create table department(eid number(5),
deptno char(10), deptname varchar2(20)” );
```

JDBC API (Continued)

DDL Operations

Altering Table

The code snippet to add a column in *department* table is:

```
Statement stmt = con.createStatement();
stmt.execute(“ALTER TABLE department add depthead
varchar2(15)”);
```

JDBC API (Continued)

DDL Operations

Dropping Table

The code snippet to create a *department* table is:

```
Statement stmt = con.createStatement();
stmt.execute(“DROP TABLE department” );
```

JDBC API (Continued)

PreparedStatement Interface

- The PreparedStatement Interface object:

✓ pass runtime parameters to the SQL statements.

✓ Is compiled and prepared only once by the JDBC.

✓ prepareStatement() method is used to submit parameterized query using a connection object to the database.

JDBC API (Continued)

PreparedStatement Interface (Continued)

Code snippet for preparedStatement:

The code snippet to pass the employee id during runtime using prepareStatement() method:

```
String s=“select * from employee where eid=? “
PreparedStatement pst = con.prepareStatement(s);
pst.setInt(1,100 );
ResultSet rs=pst.executeQuery();
```

It acts as a placeholder

The value of each ‘?’ is set by calling appropriate setXXX() method, In this case setInt()

JDBC API (Continued)

Mapping Java Types to SQL Types

SQL type	Java Type
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Transactions Overview

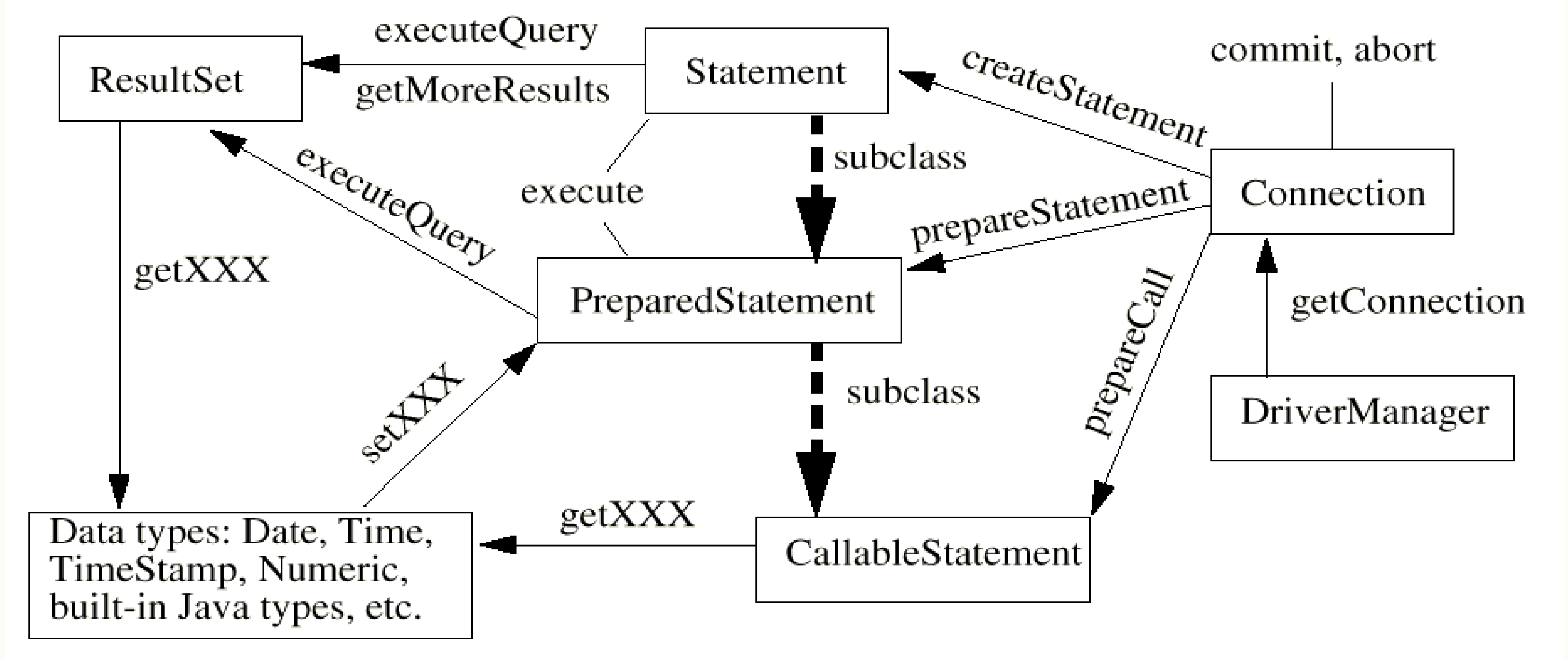
- Transaction = more than one statement which must all succeed (or all fail) together
- If one fails, the system must reverse all previous actions
- Also can’t leave DB in inconsistent state halfway through a transaction
- COMMIT = complete transaction
- ROLLBACK = abort

Transactions (Continued)

Transaction Management

- Transactions are not explicitly opened and closed
- if AutoCommit is true, then every statement is automatically committed
- default case: true
- if *AutoCommit* is false, then every statement is added to an ongoing transaction
- Must explicitly rollback or commit.

JDBC Class Diagram



Batch Processing in JDBC

- Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.
- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Advantage of Batch Processing

- Fast Performance

- void addBatch(String query)It adds query into batch.
- int[] executeBatch()It executes the batch of queries.

```
import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);
 Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,‘palak',40000)");
stmt.addBatch("insert into user420 values(191,‘prakriti',50000)");
stmt.executeBatch();//executing the batch


con.commit();
con.close();
}}
```

Summary

- JDBC Architecture consists of two layers:
  - ✓ JDBC application layer: Signifies a Java application that uses the JDBC API to interact with the JDBC driver manager.
  - ✓ JDBC driver layer: Contains a driver, such as an SQL Server driver, which enables a Java application to connect to a database. This layer acts as an interface between a Java application and a database.
- The JDBC driver manager manages various JDBC drivers.
- The JDBC driver is software that a Java application uses to access a database.

Summary (Continued)

- JDBC supports four types of drivers:
  - ✓ JDBC-ODBC Bridge driver
  - ✓ Native-API Partly-Java driver
  - ✓ JDBC-Net Pure-Java driver
  - ✓ Native Protocol Pure-Java driver
- The JDBC API consists of various classes and interfaces that enable Java applications to interact with databases.
- The classes and interfaces of the JDBC API are defined in the java.sql and javax.sql packages.
- You can load a driver and register it with the driver manager either programmatically or manually.

Summary (Continued)

- Two ways to load and register a driver programmatically are:
  - ✓ Using the Class.forName() method
  - ✓ Using the registerDriver() method
- You can add the driver name to the jdbc.drivers system property to load and register a JDBC driver manually.
- A Connection object establishes a connection between a Java application and a database.
- A Statement object sends requests to and retrieves results from a database.
- You can insert, update, and delete data from a table using the DML statements in Java applications.

Summary (Continued)

- You can create, alter, and drop tables from a database using the DDL statements in Java applications.
- A ResultSet object stores the result retrieved from a database when a SELECT statement is executed.
- You can create various types of ResultSet objects such as read only, updatable, and forward only.

‘JDBC Callable Statement – Execute Store Procedure

WHAT IS CALLABLE STATEMENT IN JDBC?

The CallableStatement Interface is used in accessing and executing Store Procedure and Function. Store Procedure is a group of SQL Statement that encapsulates all the queries and gets compiled. Once store procedure gets compiled it can be executed so many times without compilation. This makes it robust, faster and high performing. It accepts parameter for sql queries that is hidden inside it and gets executed. The Callable Statement in JDBC is used to manipulating these store procedure.

HOW TO USE CALLABLE STATEMENT TO CALL STORE PROCEDURE?

Here, I am going to present a simple callable statement example. This example includes following steps.

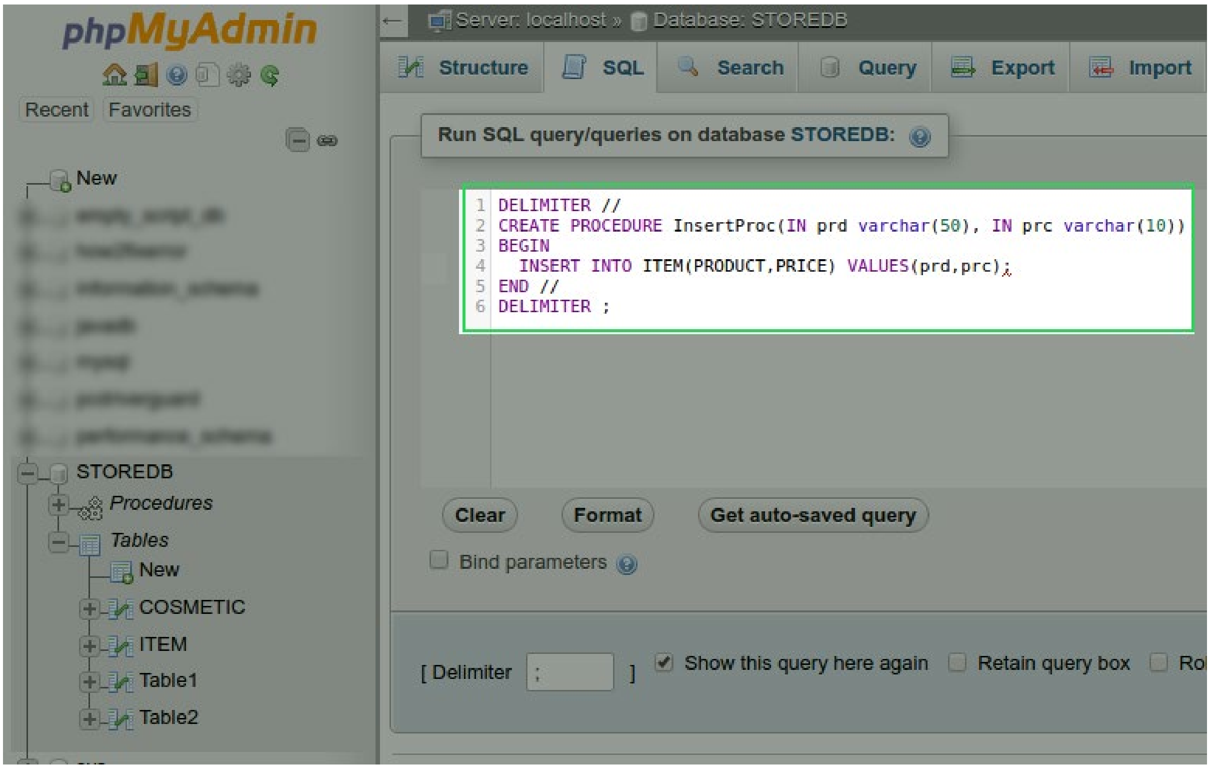
1. First step is creating a store procedure. I will create a store procedure to insert data into ITEM table.
2. Second Step is Using CallableStatement, I will execute this store procedure and insert data to table.

PROGRAMMING EXAMPLE

1. Store Procedure: Creating Store Procedure

You need to execute this statement in your database server sql window. Here, I am using MySQL and used PHPMyAdmin page to create this store procedure.

```
DELIMITER //
CREATE PROCEDURE InsertProc(IN prd varchar(50), IN prc varchar(10))
BEGIN
 INSERT INTO ITEM(PRODUCT,PRICE) VALUES(prd,prc);
END //
DELIMITER ;
```



```
DELIMITER //
CREATE PROCEDURE InsertProc(IN prd varchar(50), IN prc varchar(10))
BEGIN
  INSERT INTO ITEM(PRODUCT,PRICE) VALUES(prd,prc);
END //
DELIMITER ;
```

This statement will create InsertProc Store procedure which takes 2 parameter prd and prc.

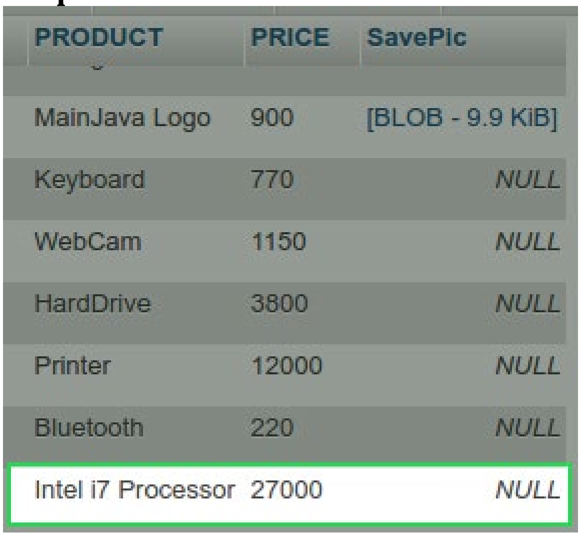
2. Use CallableStatement in JDBC to call and execute this store procedure.

package AdvanceJDBC;

- import java.sql.*;
- import com.mysql.jdbc.CallableStatement;
- public class Callable_StoreProc
- {

```
        static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
        static final String dburl = "jdbc:mysql://localhost/STOREDB";
        static final String dbuser = "root";
        static final String dbpass = "root";

public static void main(String[] args)
{
        Connection con = null;
        CallableStatement csmt = null;

        try
        {
          //Step 1 : Connecting to server and database
          con = DriverManager.getConnection(dburl, dbuser, dbpass);
          //Step 2 : Initialize CallableStatement
          csmt=(CallableStatement)con.prepareCall("{call InsertProc(?,?)}");
          //Step 3 : Execute CallableStatement with Store Procedure
          csmt.setString(1, "Intel i7 Processor");
          csmt.setString(2, "27000");
          csmt.execute();
          System.out.println("Record Inserted Successfully");
        }

        catch (SQLException e)
        {
          System.err.println("Cannot connect ! ");
          e.printStackTrace();
        }

        finally {
          System.out.println("Closing the connection.");
          if (con != null) try { con.close(); } catch (SQLException ignore) {}
        }

}
}
```

Output

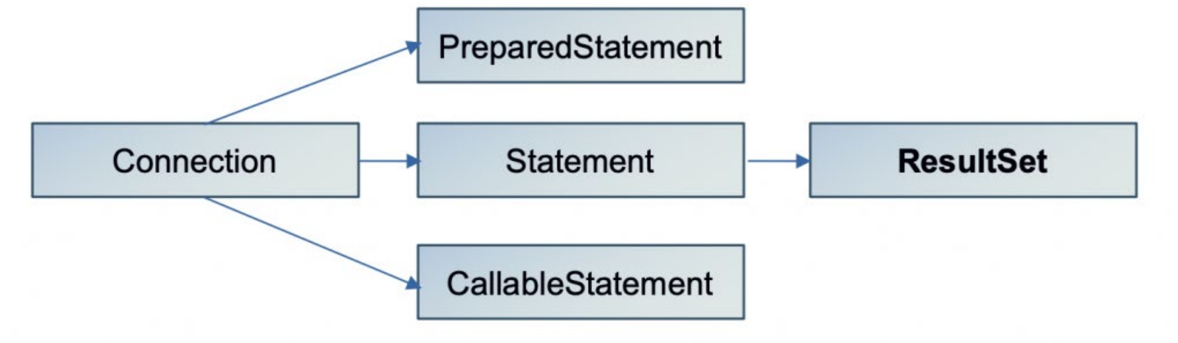


PRODUCT	PRICE	SavePic
MainJava Logo	900	[BLOB - 9.9 KiB]
Keyboard	770	*NULL*
WebCam	1150	*NULL*
HardDrive	3800	*NULL*
Printer	12000	*NULL*
Bluetooth	220	*NULL*
Intel i7 Processor	27000	*NULL*

Java **ResultSet** interface is a part of the java.sql package. It is one of the core components of the JDBC Framework. ResultSet Object is used to access query results retrieved from the relational databases.

ResultSet maintains cursor/pointer which points to a single row of the query results. Using navigational and getter methods provided by ResultSet, we can iterate and access database records one by one. ResultSet can also be used to update data.

Java ResultSet Hierarchy



Java ResultSet Class Hierarchy
The above diagram shows the place of ResultSet in the JDBC Framework. **ResultSet** can be obtained by executing SQL Query using **Statement**, **PreparedStatement** or **CallableStatement**. **AutoCloseable**, **Wrapper** are super interfaces of ResultSet. Now we will see how to work with ResultSet in our Java programs.

ResultSet Example

We will be using MySQL for our example purpose. Use below DB script to create a database and table along with some records.

```
create database empdb;

use empdb;

create table tblemployee (empid integer primary key, firstname
varchar(32), lastname varchar(32), dob date);

insert into tblemployee values  (1, 'Mike', 'Davis',' 1998-11-11');
insert into tblemployee values  (2, 'Josh', 'Martin', '1988-10-22');
insert into tblemployee values  (3, 'Ricky', 'Smith', '1999-05-11');
```

Copy

Let’s have look at the below example program to fetch the records from the table and print them on the console. Please make sure you have the MySQL JDBC driver in the project classpath.

```
package com.journaldev.examples;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Date;


public class ResultSetDemo {

        public static void main(String[] args) {
                String query = "select empid, firstname, lastname, dob 
from tblemployee";
                Connection conn = null;
                Statement stmt = null;
                try {
                        Class.forName("com.mysql.jdbc.Driver");
                        conn = 
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/empdb", 
"root", "root");
                        stmt = conn.createStatement();
                        ResultSet rs = stmt.executeQuery(query);
                        while (rs.next()) {
                                Integer empId = rs.getInt(1);
                                String firstName = rs.getString(2);
                                String lastName = rs.getString(3);
                                Date dob = rs.getDate(4);
                                System.out.println("empId:" + empId);
                                System.out.println("firstName:" + 
firstName);
                                System.out.println("lastName:" + 
lastName);
                                System.out.println("dob:" + dob);
                                System.out.println("");
                        }
                        rs.close();
                } catch (Exception e) {
```

```
            e.printStackTrace();
        } finally {
            try {
                stmt.close();
                conn.close();
            } catch (Exception e) {}
        }
    }
}
```

Output:

```
empId:1
firstName:Mike
lastName:Davis
dob:1998-11-11

empId:2
firstName:Josh
lastName:Martin
dob:1988-10-22

empId:3
firstName:Ricky
lastName:Smith
dob:1999-05-11
```

Explanation:

- ResultSet is obtained by calling the executeQuery method on Statement instance. Initially, the cursor of ResultSet points to the position before the first row.
- The method next of ResultSet moves the cursor to the next row. It returns true if there is further row otherwise it returns false.
- We can obtain data from **ResultSet** using getter methods provided by it. e.g. getInt(), getString(), getDate()
- All the getter methods have two variants. 1st variant takes column index as Parameter and 2nd variant accepts column name as Parameter.
- Finally, we need to call **close** method on **ResultSet** instance so that all resources are cleaned up properly.

ResultSet Types & Concurrency

We can specify type and concurrency of ResultSet while creating an instance of Statement, PreparedStatement or CallableStatement.

statement.createStatement(int resultSetType, int resultSetConcurrency)

ResultSet Types

1) Forward Only (ResultSet.*TYPE_FORWARD_ONLY*)

This type of ResultSet instance can move only in the forward direction from the first row to the last row. ResultSet can be moved forward one row by calling the next() method. We can obtain this type of ResultSet while creating Instance of Statement, PreparedStatement or CallableStatement.

```
Statement stmt =
connection.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select * from tbluser");
```

Copy

2) Scroll Insensitive (ResultSet.*TYPE_SCROLL_INSENSITIVE*)

Scroll Insensitive ResultSet can scroll in both forward and backward directions. It can also be scrolled to an absolute position by calling the absolute() method. But it is not sensitive to data changes. It will only have data when the query was executed and ResultSet was obtained. It will not reflect the changes made to data after it was obtained.

```
Statement stmt =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
       ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select * from tbluser");
```

Copy

3) Scroll Sensitive (ResultSet.*TYPE_SCROLL_SENSITIVE)*

Scroll Sensitive ResultSet can scroll in both forward and backward directions. It can also be scrolled to an absolute position by calling the absolute() method. But it is sensitive to data changes. It will reflect the changes made to data while it is open.

```
Statement stmt =
connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
       ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select * from tbluser");
```

Copy

ResultSet Concurrency

1) Read Only (ResultSet.*CONCUR_READ_ONLY)*

It is the default concurrency model. We can only perform Read-Only operations on ResultSet Instance. No update Operations are allowed.

2) Updatable (ResultSet.*CONCUR_UPDATABLE)*

In this case, we can perform update operations on ResultSet instance.

ResultSet Methods

We can divide ResultSet methods into the following categories.

- **Navigational Methods**
- **Getter/Reader Methods**
- **Setter/Updater Methods**
- **Miscellaneous Methods** - close() and getMetaData()

1. ResultSet Navigational Methods

- **boolean** absolute(**int** row) **throws** SQLException**:** This method moves ResultSet cursor to the specified row and returns true if the operation is successful.
- **void** afterLast() **throws** SQLException**:** This method moves ResultSet cursor to the position after the last row.
- **void** beforeFirst() **throws** SQLException**:** This method moves ResultSet cursor to the position before the first row.
- **boolean** first() **throws** SQLException: This method moves ResultSet cursor to the first row.
- **boolean** last() **throws** SQLException: This method moves ResultSet cursor to the last row.
- **boolean** next() **throws** SQLException: This method moves ResultSet cursor to the next row.
- **boolean** previous() **throws** SQLException: This method moves ResultSet cursor to the previous row.

```
package com.journaldev.examples;
import java.sql.*;

public class ResultSetDemo {

       public static void main(String[] args) {
              String query = "select empid, firstname, lastname, dob
from tblemployee";
              Connection conn = null;
              Statement stmt = null;
              try {
                     Class.forName("com.mysql.jdbc.Driver");
                     conn =
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/empdb",
"root", "root");
                     stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
                     ResultSet rs = stmt.executeQuery(query);
```

```
                        System.out.println("All the rows of table=>");
                        while (rs.next()) {
                                // Go to next row by calling next() method
                                displayData(rs);
                        }
                        System.out.println("Now go directly to 2nd
row=>");
                        rs.absolute(2); // Go directly to 2nd row
                        displayData(rs);
                        System.out.println("Now go to Previous row=>");
                        rs.previous();
                        // Go to 1st row which is previous of 2nd row
                        displayData(rs);
                        rs.close();
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        try {
                                stmt.close();
                                conn.close();
                        } catch (Exception e) {
                        }
                }
        }

        public static void displayData(ResultSet rs) throws SQLException
{
                System.out.println("empId:" + rs.getInt(1));
                System.out.println("firstName:" + rs.getString(2));
                System.out.println("lastName:" + rs.getString(3));
                System.out.println("dob:" + rs.getDate(4));
                System.out.println("");
        }
}
```

Copy

Output:

```
All the rows of table=>
empId:1
firstName:Mike
lastName:Davis
dob:1998-11-11
```

```
empId:2
firstName:Josh
lastName:Martin
dob:1988-10-22

empId:3
firstName:Ricky
lastName:Smith
dob:1999-05-11

Now go directly to 2nd row=>
empId:2
firstName:Josh
lastName:Martin
dob:1988-10-22

Now go to Previous row=>
empId:1
firstName:Mike
lastName:Davis
dob:1998-11-11
```
Copy

2. ResultSet Getter/Reader Methods

- **int** getInt(**int** columnIndex) **throws** SQLException: This method returns value of specified columnIndex as **int**.
- **long** getLong(**int** columnIndex) **throws** SQLException: This method returns value of specified columnIndex as **long**
- String getString(**int** columnIndex) **throws** SQLException: This method returns value of specified columnIndex as **String**
- java.sql.Date getDate(**int** columnIndex) **throws** SQLException: This method returns value of specified columnIndex as **java.sql.Date**
- **int** getInt(String columnLabel) **throws** SQLException: This method returns value of specified column name as **int**.
- **long** getLong(String columnLabel) **throws** SQLException: This method returns value of specified column name as **long**.
- String getString(String columnLabel) **throws** SQLException: This method returns the value of the specified column name as String.
- java.sql.Date getDate(String columnLabel) throws SQLException: This method returns the value of the specified column name as **java.sql.Date**.
- ResultSet contains getter methods that return other primitive datatypes like boolean, float and double. It also has methods to obtain array and binary data from the database.

3. ResultSet Setter/Updater Methods

- **void** updateInt(**int** columnIndex, **int** x) **throws** SQLException: This method updates the value of specified column of current row with **int** value.
- **void** updateLong(**int** columnIndex, **long** x) **throws** SQLException: This method updates the value of the specified column of the current row with long value.
- void updateString(int columnIndex, String x) throws SQLException: This method updates the value of the specified column of the current row with a String value.
- **void** updateDate(**int** columnIndex, java.sql.Date x) **throws** SQLException: This method updates the value of specified column of current row with java.sql.Date value.
- void updateInt(String columnLabel, int x) throws SQLException: This method updates the value of the specified column label of the current row with int value.
- void updateLong(String columnLabel, long x) throws SQLException: This method updates the value of the specified column label of the current row with long value.
- void updateString(String columnLabel, String x) throws SQLException: This method updates the value of the specified column label of the current row with a String value.
- **void** updateDate(String columnLabel, java.sql.Date x) **throws** SQLException: This method updates the value of specified columnLabel of current row with java.sql.Date value.

Note: Setter/Updater Methods doesn't directly update database values. **Database values will be inserted/updated after calling the insertRow or updateRow method**.

```
package com.journaldev.examples;
import java.sql.*;



public class ResultSetUpdateDemo {

       public static void main(String[] args) {
              String query = "select empid, firstname, lastname, dob
from tblemployee";
              Connection conn = null;
              Statement stmt = null;
              try {
                     Class.forName("com.mysql.jdbc.Driver");
                     conn =
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/empdb",
"root", "root");
```

```
            stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
            ResultSet rs = stmt.executeQuery(query);
            System.out.println("Now go directly to 2nd row
for Update");
            if (rs.absolute(2)) {
                // Go directly to 2nd row
                System.out.println("Existing Name:" +
rs.getString("firstName"));
                rs.updateString("firstname", "Tyson");
                rs.updateRow();
            }
            rs.beforeFirst(); // go to start
            System.out.println("All the rows of table=>");
            while (rs.next()) {
            // Go to next row by calling next() method
                displayData(rs);
            }
            rs.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                stmt.close();
                conn.close();
            } catch (Exception e) {
            }
        }
    }

    public static void displayData(ResultSet rs) throws SQLException
{
        System.out.println("empId:" + rs.getInt(1));
        System.out.println("firstName:" + rs.getString(2));
        System.out.println("lastName:" + rs.getString(3));
        System.out.println("dob:" + rs.getDate(4));
        System.out.println("");
    }
}
```

Output:

```
Now go directly to 2nd row for Update
```

```
Existing Name:Josh
All the rows of table=>
empId:1
firstName:Mike
lastName:Davis
dob:1998-11-11

empId:2
firstName:Tyson
lastName:Martin
dob:1988-10-22

empId:3
firstName:Ricky
lastName:Smith
dob:1999-05-11
```

4. ResultSet Miscellaneous Methods

- **void** close() **throws** SQLException**:** This method frees up resources associated with ResultSet Instance. It must be called otherwise it will result in resource leakage.
- **ResultSetMetaData** getMetaData() **throws** SQLException: This method returns ResultSetMetaData instance. It gives information about the type and property of columns of the query output.

Introduction

This article explains how to set up a database connection with a Swing GUI in Java. The NetBeans IDE is used to create the sample examples.

Fetching Records from Database using Swing GUI

For creating this app we need the following files:

1. Java file (SwingDatabaseApp.java)
2. SQL table (emp.sql)

1. SwingDatabaseApp.java

This Java consists of the entire logic. First of all we initialize JFrame components using a constructor then create a database connection and finally set the database value to the textfield and display it using a constructor.

2. emp.sql

For fetching records we need a database table; for that we create an "emp" table in our database.

Syntax

emp.sql

```
create table emp
 (
   uname varchar2(20), umail varchar2(30),
   upass varchar2(20), ucountry varchar2(20)
 );
```

Insert some rows

The following SQL will insert some rows in it:

1. insert into emp values ('sandeep', 'sandy05.1991@gmail.com', 'welcome', 'India');
2. insert into emp values ('rahul', 'rahul@gmail.com' , '123', 'India');

Now let's start creating this app. Use the following procedure to do that in the NetBeans IDE.

Step 1

Open the NetBeans IDE.

Step 2
Choose "Java" -> "Java application" as shown below.

Step 3
Type your project name as "SwingDatabaseApp" as in the following.

Step 4

Now provide the following code in the "SwingDatabaseApp.java" file.

SwingDatabaseApp.java

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.sql.*;

public class SwingDatabaseApp extends JFrame {

//Initializing Components
    JLabel lb, lb1, lb2, lb3, lb4;
    JTextField tf1, tf2, tf3, tf4;

    //Creating Constructor for initializing JFrame components
    SwingDatabaseApp() {
        //Providing Title
        super("Fetching Student Information");

        lb = new JLabel("Fetching Student Information From Database");
        lb.setBounds(20, 50, 450, 20);
        lb.setForeground(Color.red);
        lb.setFont(new Font("Serif", Font.BOLD, 20));
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500, 500);
        lb1 = new JLabel("U_Name:");
        lb1.setBounds(50, 105, 100, 20);
        tf1 = new JTextField(50);
        tf1.setBounds(160, 105, 100, 20);
        lb2 = new JLabel("U_Mail:");
        lb2.setBounds(50, 135, 100, 20);
        tf2 = new JTextField(100);
        tf2.setBounds(160, 135, 200, 20);
        lb3 = new JLabel("U_Pass:");
        lb3.setBounds(50, 165, 100, 20);
        tf3 = new JTextField(50);
        tf3.setBounds(160, 165, 100, 20);
        lb4 = new JLabel("U_Country:");
        lb4.setBounds(50, 200, 100, 20);
        tf4 = new JTextField(50);
        tf4.setBounds(160, 200, 100, 20);
        setLayout(null);

        //Add components to the JFrame
        add(lb);
        add(lb1);
        add(tf1);
        add(lb2);
        add(tf2);
        add(lb3);
        add(tf3);
        add(lb4);
        add(tf4);

        //Set TextField Editable False
        tf1.setEditable(false);
        tf2.setEditable(false);
        tf3.setEditable(false);
        tf4.setEditable(false);

        //Create DataBase Coonection and Fetching Records for uname='sandeep'
        try {
            String str = "sandeep";
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@mcndesktop07:1521",
"sandeep", "welcome");
            PreparedStatement st = con.prepareStatement("select * from emp where uname=?");
            st.setString(1, str);

            //Excuting Query
            ResultSet rs = st.executeQuery();
            while (rs.next()) {
                String s = rs.getString(1);
                String s1 = rs.getString(2);
                String s2 = rs.getString(3);
                String s3 = rs.getString(4);

                //Sets Records in TextFields.
                tf1.setText(s);
                tf2.setText(s1);
                tf3.setText(s2);
                tf4.setText(s3);
            }
        } //Create Exception Handler
        catch (Exception ex) {
            System.out.println(ex);
        }
    }
//Running Constructor

    public static void main(String args[]) {
        new SwingDatabaseApp();
    }
}
```

Step 5

Now our project is ready to run. Right-click on the project menu, then choose "Run". The following output will be generated.

Step 6

Now change the uname and this time we are fetching records for "rahul".

```
String str = "rahul";
```

- fetching records in