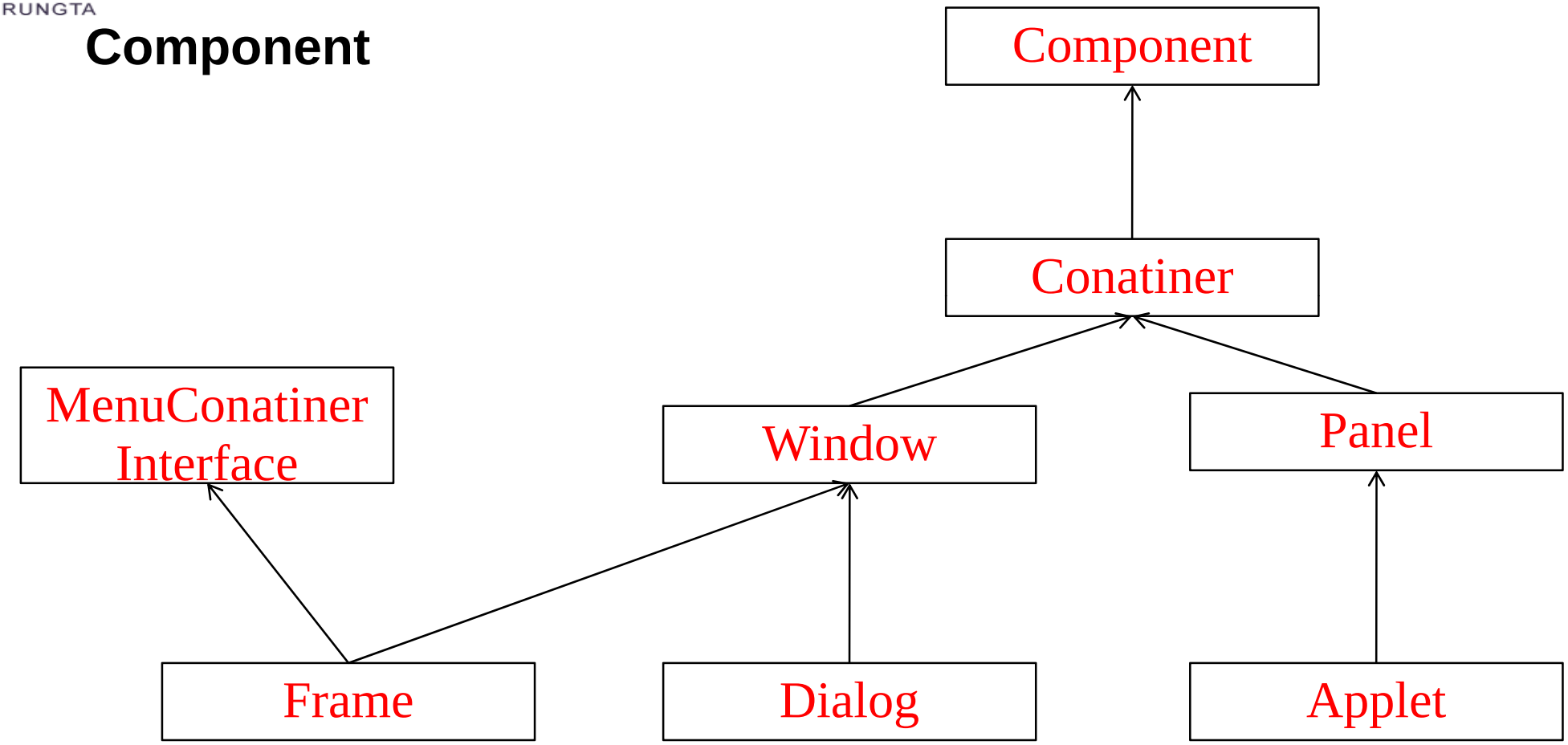
RUNGTA

AWT and Event Handeling

Abstract Window Toolkit (AWT)

- The AWT classes are contained in the java.awt package.
- The AWT defines winodws according to a class hierarchy that adds functionality at each specific level.
- At the top of the AWT hierarchy is the Components.

Component


Component
Conatiner
MenuConatiner Interface
Window
Panel
Frame
Dialog
Applet

Working With Frame

- Here are two of **Frame's constructors:**
  - Frame( )
  - Frame(String *title)*
- **Setting the Window's Dimensions**
  - void setSize(int *newWidth, int newHeight)*
  - void setSize(Dimension *newSize)*
  - Dimension getSize( )
- **Hiding and Showing a Window**
  - void setVisible(boolean *visibleFlag)*
- **Setting a Window's Title**
  - void setTitle(String *newTitle)*

Working with Color

- Color class provide to handle color. This contain following constructors
  - Color(int *red, int green, int blue)*
  - Color(int *rgbValue)*
  - Color(float *red, float green, float blue)*
- *Color Class contain also some color constant*
  - *Color.blue*
  - *Color.red*
  - *Color.green*

Basic User Interface Components

- Label
- Button
- Checkbox
- CheckboxGroup
- Choice
- List
- TextField
- TextArea
- Scrolling
- Scrollbar

Label

- A *label is an object of type* ***Label, and it contains a*** string, which it displays.
- Labels are passive controls that do not support any interaction with the user.
- **Label defines the following constructors:**
  - Label( )
  - Label(String *str)*
  - Label(String *str, int how)*
- The first version creates a blank label.
- The second version creates a label that contains the string specified by *str. This string is left-justified.*
- *The third version creates a label that* contains the string specified by *str using the alignment specified by how.*
- *The value of how* must be one of these three constants:
  - **Label.LEFT, Label.RIGHT, or Label.CENTER.**

Methods of the Label

Method	Action
void setText(String *str)*	Change the text of label
String getText( )	Return a string contatining this label‘s text
void setAlignment(int *how)*	Change the alignment of the label
int getAlignment( )	Return an integer representing the aligment 0 is Label.LEFT 1 is Label.CENTER 2 is Label.RIGHT

RUNGTA

Button

- Button is the component of Java Abstract Window Toolkit and is used to trigger actions and other events required for your application. The syntax of defining the button is as follows :
  - Button( );
  - Button(String name);

Methods of Button

Method	Action
void setLabel(String *str)*	Change the caption of button
String getLabel( )	Return a string caption of button

TextField

- This is the text container component of Java AWT package. This component contains single line and limited text information. This defines the following constructors:
  - TextField( )
  - TextField(int *numChars)*
  - TextField(String *str)*
  - TextField(String *str, int numChars)*

Method of the TextField

Method	Action
String getText()	Returns the text that the text field contains
setText(String str)	Puts the given text string into the field
setColumns(int)	Set the width of text field
int getColumns()	Returns the width of the text field
setEditable(boolean)	True -Enables text to the edited, False-Freezes the text
boolean isEditable( )	Return true or false based on weather the text is editable
void select(int *startIndex, int endIndex)*	Selects the text between the two integer position
String getSelectedText( )	Return the select text
void setEchoChar(char *ch)*	Set the character used for masking or password
char getEchoChar( )	Return the echo character
boolean echoCharIsSet( )	Return true or false
selectAll()	Select all the text in the field

TextArea

- The AWT includes a simple multiline editor called **TextArea.**
- Following are the constructors for TextArea:
  - TextArea( )
  - TextArea(int *numLines, int numChars)*
  - TextArea(String *str)*
  - TextArea(String *str, int numLines, int numChars)*
  - TextArea(String *str, int numLines, int numChars, int sBars)*
- *sBars must be one of these values:*
  - SCROLLBARS_BOTH
  - SCROLLBARS_NONE
  - SCROLLBARS_HORIZONTAL_ONLY
  - SCROLLBARS_VERTICAL_ONLY

Methods of TextArea

- void append(String *str)*
- void insert(String *str, int index)*
- void replaceRange(String *str, int startIndex, int endIndex)*

Checkbox

- This component of Java AWT allows you to create check boxes in your applications. The syntax of the definition of Checkbox is as follows :
  - Checkbox( )
  - Checkbox(String *str)*
  - Checkbox(String *str, boolean on)*
  - Checkbox(String *str, boolean on, CheckboxGroup cbGroup)*
  - Checkbox(String *str, CheckboxGroup cbGroup, boolean on)*

Method of Checkbox

Method	Action
boolean getState( )	Return true or false based on whether the check box is selected
void setState(boolean *on)*	Change the check box state true or false
String getLabel( )	Return a string containing the check box label
void setLabel(String *str)*	Change the text of the checkbox label

CheckboxGroup

- It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time.
- These check boxes are often called *radio buttons*
- CheckboxGroup methods are as follows:
  - Checkbox getSelectedCheckbox( )
  - void setSelectedCheckbox(Checkbox *which)*

Choice

- work as combobox it is used to create popup list of items from which the user may choose.
- Constructor
  - Choice()
- Methods
  - void add(String *name)*
  - String getSelectedItem( )
  - int getSelectedIndex(
  - int getItemCount( )
  - void select(int *index)*
  - void select(String *name)*
  - String getItem(int *index)*

Menu Bars and Menus

- To create a menu and menu bar,
- first create an instance of **MenuBar.** This class only defines the default constructor.
- Next, create instances of Menu that will define the selections displayed on the bar. Following are the constructors for Menu:
  - Menu( )
  - Menu(String *optionName)*
  - Menu(String *optionName, boolean removable)*
- Individual menu items are of type **MenuItem. It defines these constructors:**
  - MenuItem( )
  - MenuItem(String *itemName)*
  - MenuItem(String *itemName, MenuShortcut keyAccel*)

List

- The List class provides a compact, multiple-choice, scrolling selection list. It provide following constructors
  - List( )
  - List(int *numRows)*
  - List(int *numRows, boolean multipleSelect)*
- Methods
  - void add(String *name)*
  - void add(String *name, int index)*
  - String getSelectedItem( )
  - int getSelectedIndex( )
  - String[ ] getSelectedItems( )
  - int[ ] getSelectedIndexes( )
  - int getItemCount( )
  - void select(int *index)*
  - String getItem(int *index)*

Scrollbar

- *Scroll bars are used to select continuous values between a specified minimum and* maximum.
- Scrollbar defines the following constructors:
  - Scrollbar( )
  - Scrollbar(int *style)*
  - Scrollbar(int *style, int initialValue, int thumbSize, int min, int max)*
- *Style can be* **Scrollbar.VERTICAL or Scrollbar.HORIZONTAL**
- **Methods**
  - void setValues(int *initialValue, int thumbSize, int min, int max)*
  - int getValue( )
  - void setValue(int *newValue)*
  - int getMinimum( )
  - int getMaximum( )
  - void setUnitIncrement(int *newIncr)*
  - void setBlockIncrement(int *newIncr)*

Layout Management

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout

FlowLayout

- FlowLayout is the default layout manager for Panel and Applet.
- the constructors for **FlowLayout:**
  - FlowLayout( )
  - FlowLayout(int *how)*
  - FlowLayout(int *how, int horz, int vert)*
- Valid values for *how are as follows:*
  - FlowLayout.LEFT
  - FlowLayout.CENTER
  - FlowLayout.RIGHT
- *horz and vert* specify the horizontal and vertical space left between components.

BorderLayout

- The **BorderLayout class implements a common layout style for top-level windows.**
- **It** has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as north, south, east, and west.
- The middle area is called the center.
- Here are the constructors defined by **BorderLayout:**
  - BorderLayout( )
  - BorderLayout(int *horz, int vert)*
- **BorderLayout defines the following constants that specify the regions:**
  - BorderLayout.CENTER
  - BorderLayout.EAST
  - BorderLayout.NORTH
  - BorderLayout.SOUTH
  - BorderLayout.WEST

GridLayout

- GridLayout lays out components in a two-dimensional grid.
- When you instantiate a GridLayout, you define the number of rows and columns.
- The constructors supported by GridLayout are shown here:
  - GridLayout( )
  - GridLayout(int *numRows, int numColumns)*
  - GridLayout(int *numRows, int numColumns, int horz, int vert)*

CardLayout

- **CardLayout provides these two constructors:**
  - CardLayout( )
  - CardLayout(int *horz, int vert)*

Font

- **Font constructor has this general form:**
  - Font(String *fontName, int fontStyle, int pointSize)*
- *fontName specifies the name of the desired font.*
- The style of the font is specified by *fontStyle.*
- **Font.PLAIN, Font.BOLD, and Font.ITALIC.**
- We can set font in componet to using following method
  - void setFont(Font *fontObj)*

Event Handeling

- An *event is an object that describes a state change in a source.*
- An event *source is an object that generates an event.*
- Each type of event has its own registration method.
- Here is the general form:
  - public void add*TypeListener(TypeListener el)*
- An event *listener is an object that is notified when an event occurs.*

Event Classes

- At the root of the Java event class hierarchy is **EventObject**, which is in java.util.
- It is the superclass for all events. Its one constructor is shown here:
  - EventObject(Object *src)*
- EventObject contains two methods: getSource( ) and toString( ).
- The getSource( ) method returns the source of the event. Its general form is shown here:
  - Object getSource( )
- The class **AWTEvent,** defined within the java.awt package, is a subclass of EventObject.
- The package **java.awt.event defines several types of events that are generated by**
- various user interface elements

Event Classes

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.

Event Classes...

Event Class	Description
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved. (Added by Java 2, version 1.4)
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

The ActionEvent Class

- An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
- The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event:
- ALT_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK.
- ActionEvent has these three constructors:
  - ActionEvent(Object *src, int type, String cmd)*
  - ActionEvent(Object *src, int type, String cmd, int modifiers)*
  - ActionEvent(Object *src, int type, String cmd, long when, int modifiers)*

The AdjustmentEvent Class

- An AdjustmentEvent is generated by a scroll bar.
- The constants and their meanings are shown here:
- BLOCK_DECREMENT -The user clicked inside the scroll bar to decrease its value.
- BLOCK_INCREMENT -The user clicked inside the scroll bar to increase its value.
- TRACK - The slider was dragged.
- UNIT_DECREMENT - The button at the end of the scroll bar was clicked
- to decrease its value.
- UNIT_INCREMENT - The button at the end of the scroll bar was clicked
- to increase its value.

The ComponentEvent Class

- A ComponentEvent is generated when the size, position, or visibility of a component is changed.
- ComponentEvent has this constructor:
  - ComponentEvent(Component *src, int type)*
- The type of the event is specified by *type.*
  - COMPONENT_HIDDEN: The component was hidden.
  - COMPONENT_MOVED: The component was moved.
  - COMPONENT_RESIZED:The component was resized.
  - COMPONENT_SHOWN: The component became visible.

The FocusEvent Class

- A FocusEvent is generated when a component gains or loses input focus.
- These events are identified by the integer constants FOCUS_GAINED and FOCUS_LOST.
- **FocusEvent is a subclass of ComponentEvent and has these constructors:**
  - FocusEvent(Component *src, int type)*
  - FocusEvent(Component *src, int type, boolean temporaryFlag)*
  - Focus Event(Component *src, int type, boolean temporaryFlag, Component other)*

The InputEvent Class

- The abstract class **InputEvent is a subclass of ComponentEvent**
- Its subclasses are **KeyEvent and MouseEvent.**
- InputEvent defines several integer constants that represent any modifiers, such as the control key being pressed, that might be associated with the event.
- **ALT_MASK , BUTTON2_MASK, META_MASK, ALT_GRAPH_MASK , BUTTON3_MASK SHIFT_MASK, BUTTON1_MASK CTRL_MASK, ALT_DOWN_MASK , ALT_GRAPH_DOWN_MASK, BUTTON1_DOWN_MASK, BUTTON2_DOWN_MASK , BUTTON3_DOWN_MASK, CTRL_DOWN_MASK, META_DOWN_MASK , SHIFT_DOWN_MASK**

The ItemEvent Class

- An ItemEvent is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected.

The KeyEvent Class

- A **KeyEvent is generated when keyboard input occurs. There are three types** of key events, which are identified by these integer constants:
- **KEY_PRESSED, KEY_RELEASED, and KEY_TYPED.**
- There are many other integer constants that are defined by **KeyEvent.**
- **VK_ENTER VK_ESCAPE VK_CANCEL VK_UP**
- **VK_DOWN VK_LEFT VK_RIGHT VK_PAGE_DOWN**
- **VK_PAGE_UP VK_SHIFT VK_ALT VK_CONTROL**
- **VK_0 through VK_9**
- **VK_A through VK_Z**

The MouseEvent Class

- MouseEvent is a subclass of InputEvent.
- There are eight types of mouse events.
  - MOUSE_CLICKED :The user clicked the mouse.
  - MOUSE_DRAGGED :The user dragged the mouse.
  - MOUSE_ENTERED :The mouse entered a component.
  - MOUSE_EXITED :The mouse exited from a component.
  - MOUSE_MOVED :The mouse moved.
  - MOUSE_PRESSED: The mouse was pressed.
  - MOUSE_RELEASED: The mouse was released.
  - MOUSE_WHEEL: The mouse wheel was moved

The WindowEvent Class

- WINDOW_ACTIVATED :The window was activated.
- WINDOW_CLOSED :The window has been closed.
- WINDOW_CLOSING :The user requested that the windowbe closed.
- WINDOW_DEACTIVATED :The window was deactivated.
- WINDOW_DEICONIFIED :The window was deiconified.
- WINDOW_GAINED_FOCUS: The window gained input focus.
- WINDOW_ICONIFIED :The window was iconified.
- WINDOW_LOST_FOCUS :The window lost input focus.
- WINDOW_OPENED: The window was opened.
- WINDOW_STATE_CHANGED: The state of the window changed.

RUNGTA

Sources of Events

Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces

- The delegation event model has two parts: sources and listeners.
- Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package.
- When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

RUNGTA

Event Listener Interfaces…

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.

Event Listener Interfaces…

MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved. (Added by Java 2, version 1.4)
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus. (Added by Java 2, version 1.4)
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces and methods

Interface	Method
ActionListener	void actionPerformed(ActionEvent *ae)*
AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent *ae)*
ComponentListener	void componentResized(ComponentEvent *ce)* void componentMoved(ComponentEvent *ce)* void componentShown(ComponentEvent *ce)* void componentHidden(ComponentEvent *ce)*
ContainerListener	void componentAdded(ContainerEvent *ce)* void componentRemoved(ContainerEvent *ce)*
FocusListener	void focusGained(FocusEvent *fe)* void focusLost(FocusEvent *fe)*

Event Listener Interfaces and methods…

Interface	Method
ItemListener	void itemStateChanged(ItemEvent *ie)*
KeyListener	void keyPressed(KeyEvent *ke)* void keyReleased(KeyEvent *ke)* void keyTyped(KeyEvent *ke)*
MouseListener	void mouseClicked(MouseEvent *me)* void mouseEntered(MouseEvent *me)* void mouseExited(MouseEvent *me)* void mousePressed(MouseEvent *me)* void mouseReleased(MouseEvent *me)*
MouseMotionListener	void mouseDragged(MouseEvent *me)* void mouseMoved(MouseEvent *me)*
TextListener	void textChanged(TextEvent *te)*

Event Listener Interfaces and methods…

Interface	Method
WindowFocusListener	void windowGainedFocus(WindowEvent *we)* void windowLostFocus(WindowEvent *we)*
WindowListener	void windowActivated(WindowEvent *we)* void windowClosed(WindowEvent *we)* void windowClosing(WindowEvent *we)* void windowDeactivated(WindowEvent *we)* void windowDeiconified(WindowEvent *we)* void windowIconified(WindowEvent *we)* void windowOpened(WindowEvent *we)*

The Delegation Event Model

- Implement the appropriate interface in the listener so that it will receive the type of event desired.
- Implement code to register and unregister (if necessary) the listener as a recipient for the event notifications.

Adapter Classes

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

JDBC Basics

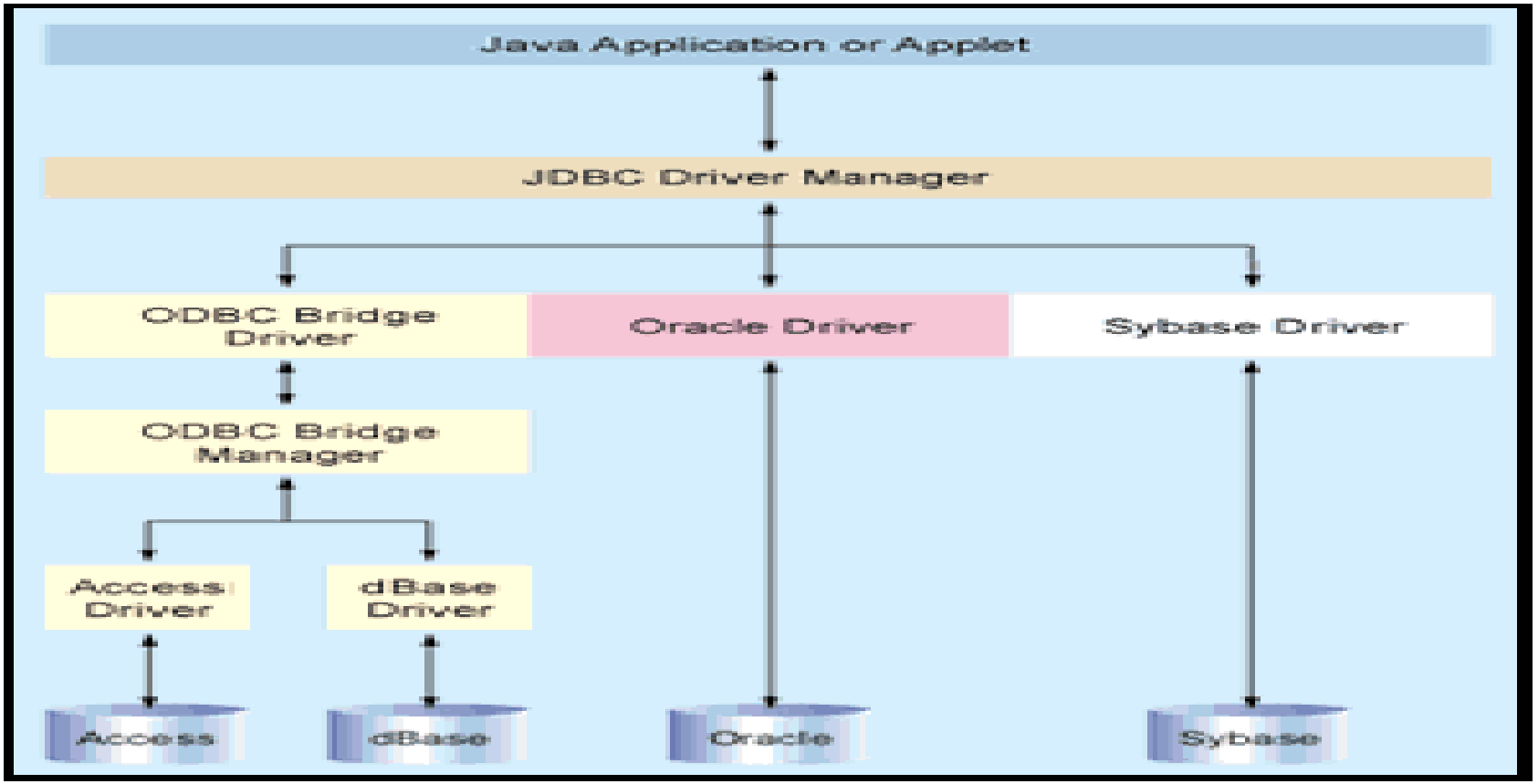


Figure 1. Anatomy of Data Access. The Driver Manager provides a consistent layer between your Java app and back-end database. JDBC works natively (such as with the Oracle driver in this example) or with any ODBC datasource.

JDBC Drivers

1. JDBC-ODBC Bridge
2. A native API partly Java Technology enabled driver
3. A net protocol fully Java Technology enabled driver (I)
4. A net protocol fully Java Technology enabled driver (II)

The Seven Basic Steps to JDBC

- Import java.sql package
- Load and register the driver
- Establish a connection to database server
- Create a statement
- Execute the statement
- Retrieve the results
- Close the statement and connection

Getting Started

- **Loading a driver**
  - E.g Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
- **Making a Connection**
  - Connection con = DriverManager.getConnection(url, “mylogin”, “password”);
  - Note: url identifies the Data Source in Database. E.g String url = “`jdbc:oracle:oci8:@ccdb`”

JDBC Statements

- A JDBC statement object is used to send your SQL statement to the database server
- A JDBC statement is associated with an open connection and not any single SQL statement
- JDBC provides three classes of SQL statement
  - Statement
  - PreparedStatement
  - CallableStatement

Creating JDBC Statement

```
Statement stmt = con.createStatement();
String createCS4400 = “Create table CS4400 “+
    “(SSN Integer not null, Name VARCHAR(32), “+ “Marks Integer)”;
stmt.executeUpdate(createCS4400);
String insertCS4400 = “Insert into CS4400 values “+
    “(123456789,abc,100)”;
stmt.executeUpdate(insertCS4400);
```

Create JDBC Statement (contd..)

```
String queryCS4400 = “select * from CS4400”;
ResultSet rs = Stmt.executeQuery(queryCS4400);
While (rs.next()) {
    int ssn = rs.getInt(“SSN”);
    String name = rs.getString(“NAME”);
    int marks = rs.getInt(“MARKS”);
}
```

Note: column number can also be used in place of column name.Refer to java.sql.ResulSet API for more details

Prepared Statement

- Unlike “Statement” it is given a SQL statement when it is created.
- Used when you want to execute “Statement” object many times

E.g

```
String insert = “Insert into CS4400 (?,?,?)”;
PreparedStatement stmt2 = con.prepareStatement(insert);
stmt2.setInt(1,123456789);
stmt2.setString(2,abc);
stmt2.setInt(3,100);
stmt2.executeUpdate();
```

Prepared Statement (cont..)

- Executing Select Statement

e.g

```
String query =“SELECT Name from CS4400 where SSN=?”;
PreparedStatement stmt2 = con.prepareStatement(query);
Stmt2.setInt(1,SSN);
ResultSet rs = stmt2.executeUpdate();
While (rs.next())
    System.out.println(rs.getString(Name);
```

Callable Statement

- Used for executing stored procedures
- Example

String createProcedure = “Create Procedure ShowGoodStudents” + “as Select Name from CS4400 where Marks > 90)”;

Stmt.executeUpdate(createProcedure);

CallableStatement cs = con.prepareCall(“(call ShowGoodStudents)”);

ResultSet rs = cs.executeQuery();

Result Set Meta Data

- **Stores the number, types and properties of ResultSet's columns.**

```
ResultSetMetaData rsm = rs.getMetaData();
int number = rsm.getColumnCount();
For (int i=0; i< number;i++)
        System.out.println(rsm.getColumnName(i));
```

Transactions and JDBC

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object, default mode is auto-commit, I.e., each sql statement is treated as a transaction
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

Transactions and JDBC

- If we don't want certain changes to be made permanent, we can issue con.rollback();

  Any changes made since the last commit will be ignored – usually rollback is used in combination with Java's exception handling ability to recover from unpredictable errors.

- Example

  Con.setAutocommit(false);

  Statement stmt = con.createStatement();

  Stmt.executeUpdate("INSERT INTO CS4400 VALUES (1234,'John',0)");

  Con.rollback();

  Stmt.executeUpdate("INSERT INTO CS4400 VALUES (1234,'John',0)");

  Con.commit();

  Con.setAutoCommit(true);

Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state.
- In Java statements which are expected to “throw” an exception or a warning are enclosed in a try block.
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements

Handling Errors with Exceptions

- **Example**

```
try{
  stmt.executeUpdate(queryCS4400);
} catch (SQLException e){
System.out.println(e.getMessage())
}
```

JDBC Example

```
 import java.sql.*;
public class Employee {
public static void main (String args []) {
try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     }
 catch(ClassNotFoundException e){
         System.out.println("Class not Found");
 }
try{
         Connection cnn = DriverManager.getConnection("jdbc:odbc:empdsn");
        Statement st = cnn.createStatement();
        ResultSet rs = st.executeQuery("Select * from EmpTable");
while(rs.next()){
        String eid = rs.getString(1);            //rs.getString("EID")
        String ename = rs.getString(2);
        float sal = rs.getFloat(3);
        System.out.println(eid+"  "+ename+"   "+sal);
}
st.close(); cnn.close();
}
catch(SQLException e){ System.out.println(e);
} } }
```