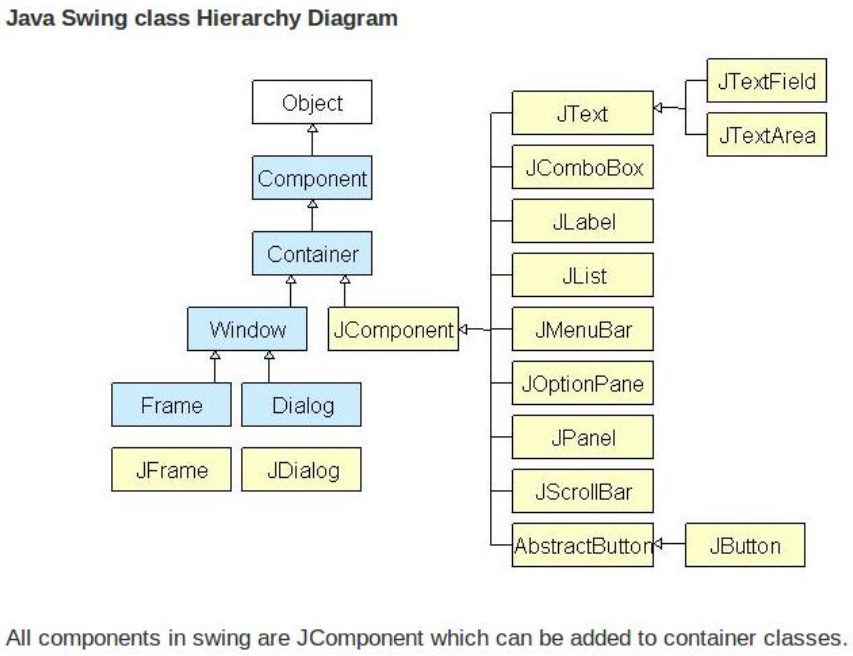
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
CS F213
LAB-8

AGENDA **21/10/2018** **TIME: 02 Hours**

1. **Java Swing, Components, EventHandling, EventClasses, EventListeners**
2. **Application programming using JDBC**

Java Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.



Container class

Container classes are classes that can have other components on it. So for creating a GUI, we need at least one container object. There are 3 types of containers.

1. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. **Frame**: It is a fully functioning window with its title and icons.
3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

Example 1: Adding a button to GUI Application

```
import javax.swing.*;
class gui{
```

```
    public static void main(String args[]){
       JFrame frame = new JFrame("My First GUI");
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(300,300);
       JButton button = new JButton("Press");
       frame.getContentPane().add(button);
       frame.setVisible(true);
    }
}
```

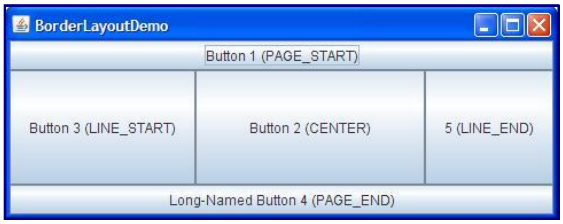
In the above example if you try to add multiple buttons they will get overlapped. Use Layout manager to add multiple components in container.

Java Layout Manger

The Layout manager is used to layout (or arrange) the GUI java components inside a container.There are many layout managers, but the most frequently used are-

Java BorderLayout

A BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java Jframe.



Java FlowLayout

FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.

Java GridLayout

The GridLayout manager is used to lay out the components in a rectangle grid, which has been divided into equal-sized rectangles and one component is placed in each rectangle.

Example 2: Creating panels with GridLayout

```
class CustomerInformation {
     JFrame f;
     JPanel p1, p2, p3;
     JTabbedPane tp;
```

```
     CustomerInformation() {
          f = new JFrame("Customer Form");
          p1 = new JPanel(new GridLayout(5, 2));
          p2 = new JPanel(new GridLayout(6, 2));
          p3 = new JPanel(new GridLayout(2, 2));
          tp = new JTabbedPane();
}
     void dis() {
          f.getContentPane().add(tp);
          tp.addTab("Add Record", p1);
          tp.addTab("Edit Record", p2);
          tp.addTab("Delete Record", p3);

          f.setSize(400, 200);
          f.setVisible(true);
          f.setResizable(true);
     }
     public static void main(String z[]) {
          CustomerInformation pro = new CustomerInformation();
          pro.dis();
     }
}
```

Example 3: Adding multiple button using FlowLayout

```
import javax.swing.*;
class gui{
    public static void main(String args[]){
     JFrame frame = new JFrame("My First GUI");

     //defining the layout of the container
     frame.setLayout(new FlowLayout());
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setSize(300,300);

     JButton button1 = new Jbutton("Press1");
     JButton button2 = new JButton("Press2");

     frame.getContentPane().add(button1);
     frame.getContentPane().add(button2);
     frame.setVisible(true);
    }
}
```

Java Swing Components

JFrame: In Java Swing, most application will be built inside a basic component JFrame, which creates a window to hold other components.

JPanel: JPanel is a popular container to hold different components. It could be set and added by using the code similar to following:

```
JPanel panel = new JPanel();
frame.add(panel);
```

JButton: JButton is an implementation of a "push" button. It can be pressed and configured to have different actions, using the event listener. Note: In example 2 we have created multiple JButton.

JRadioButton: It's a radio button that can be selected or deselected. Use with the ButtonGroup object to create a group of buttons,in which only one button can be selected at a time.

JCheckBox: JCheckBox is used to create checkbox, of which multiple checkbox could be selected at the same time.

JSlider: JSlider is a component that lets the users select a value by sliding a knob within a specified interval. For the knob, it always points to the point which matches the integer values within the interval.

JTable: JTable is used to create a regular two-dimensional tables. The table can display data inside of it. In addition, the user can also edit the data.

JComboBox: JComboBox is a component to select value from a drop-down list. You can choose one and only one element from the list.

JMenu: Used to create menu bar for your application.

Example 4: Adding Textboxes and Buttons to panels in Example 2.

// Add the following lines above the constructor in Example2

```
     JLabel namelabel1, emaillabel1, countrylabel1, genderlabel1,
deletelabel, idlabel2, namelabel2, emaillabel2, countrylabel2,
genderlabel2;
     JTextField name1, email1, gender1, delete_id, name2, email2,
gender2, id2;
     JButton savebtn, resetbtn, editbtn1, editbtn2, deletebtn;
     JComboBox country1, country2;
```

// Add the following lines in the constructor after the line `tp = new JTabbedPane();`

```
          namelabel1 = new JLabel("Customer Name:");
          emaillabel1 = new JLabel("Customer email:");
          countrylabel1 = new JLabel("Customer Country:");
          genderlabel1 = new JLabel("Customer Gender:");
          deletelabel = new JLabel("Enter Customer ID to delete
record:");
          idlabel2 = new JLabel("Customer ID:");
          namelabel2 = new JLabel("Customer Name:");
          emaillabel2 = new JLabel("Customer email:");
          countrylabel2 = new JLabel("Customer Country:");
          genderlabel2 = new JLabel("Customer Gender:");

          name1 = new JTextField(12);
```

```
email1 = new JTextField(12);
gender1 = new JTextField(12);
delete_id = new JTextField(12);

name2 = new JTextField(12);
email2 = new JTextField(12);
gender2 = new JTextField(12);
id2 = new JTextField(12);

country1 = new JComboBox();
country1.addItem("INDIA");
country1.addItem("AMERICA");
country1.addItem("AUSTRALIA");
country1.addItem("PHILLIPHINES");
country1.addItem("SPAIN");

country2 = new JComboBox();
country2.addItem("INDIA");
country2.addItem("AMERICA");
country2.addItem("AUSTRALIA");
country2.addItem("PHILLIPHINES");
country2.addItem("SPAIN");

savebtn = new JButton(" Add ");
resetbtn = new JButton(" Reset");
editbtn1 = new JButton(" Edit ");
editbtn2 = new JButton(" Save");
deletebtn = new JButton("Delete");

p1.add(namelabel1);
p1.add(name1);
p1.add(emaillabel1);
p1.add(email1);
p1.add(countrylabel1);
p1.add(country1);
p1.add(genderlabel1);
p1.add(gender1);
p1.add(savebtn);
p1.add(resetbtn);

p2.add(idlabel2);
p2.add(id2);
p2.add(namelabel2);
p2.add(name2);
p2.add(emaillabel2);
p2.add(email2);
p2.add(countrylabel2);
p2.add(country2);
p2.add(genderlabel2);
p2.add(gender2);
p2.add(editbtn1);
p2.add(editbtn2);
```

```
p3.add(deletelabel);
p3.add(delete_id);
p3.add(deletebtn);
```

SWING Event Classes

AWTEvent: It is the root event class for all SWING events. This class and its subclasses supercede the original **java.awt.Event** class.

ActionEvent: The ActionEvent is generated when the button is clicked or the item of a list is double-clicked.

InputEvent: The InputEvent class is the root event class for all component-level input events.

KeyEvent: On entering the character the Key event is generated.

MouseEvent: This event indicates a mouse action occurred in a component.

Apart from this there are various other events such as; WindowEvent, AdjustmentEvent, ComponentEvent, ContainerEvent, MouseMotionEvent etc.which represent change in the state of a window.

Event Listeners in SWING

Event listeners represent the interfaces responsible to handle events. Every method of an event listener method has a single argument as an object which is the subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

Following are the frequently used listeners in the swing:

ActionListener: The class needs to implemented this interface and an object of this class should be registered with a component by using addActionListener() method. This interface has only one method as:

1. public void actionPerformed(ActionEvent e)

MouseListener: The class which process the mouse event needs to be implemented this interface and an object of this class should be registered with a component by using addMouseListener() method. There are five methods in the interface as:

1. public abstract void mouseEntered(MouseEvent e);
2. public abstract void mouseExited(MouseEvent e);
3. public abstract void mousePressed(MouseEvent e);
4. public abstract void mouseReleased(MouseEvent e);
5. public abstract void mouseClicked(MouseEvent e);

WindowListener: The class which process the window event needs to be implemented this interface and an object of this class should be registered with a component by using addWindowListener() method. Following are the methods of the interface:

1. public abstract void windowActivated(WindowEvent e);
2. public abstract void windowClosed(WindowEvent e);
3. public abstract void windowClosing(WindowEvent e);

4. public abstract void windowDeactivated(WindowEvent e);
5. public abstract void windowDeiconified(WindowEvent e);
6. public abstract void windowIconified(WindowEvent e);
7. public abstract void windowOpened(WindowEvent e);

MouseMotionListener: The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. It has two methods.

1. public abstract void mouseDragged(MouseEvent e);
2. public abstract void mouseMoved(MouseEvent e);

KeyListener: The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. Below are the methods for interface which needs to be implemented:

1. public abstract void keyPressed(KeyEvent e);
2. public abstract void keyReleased(KeyEvent e);
3. public abstract void keyTyped(KeyEvent e);

ItemListener: The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. It has only one method:

1. public abstract void itemStateChanged(ItemEvent e);

JDBC

JDBC stands for Java Database Connectivity. The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.. It is a part of JavaSE (Java Standard Edition). This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use. JDBC API uses JDBC drivers to connect with the database.

Creating a Derby database using Java program

1. Right click on the project and go to Build Path. Select Add External Archives.
2. Navigate to "C:\Program Files\glassfish-4.1\javadb\lib" and open derby.jar.

Steps for connectivity between Java program and database

There are 5 steps to connect Java application with the database using JDBC

2. Register the Driver class
3. Create connection
4. Create Statement
5. Execute queries
6. Close connection

Register the Driver class

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

Example 1: For the default derby database in JAVA.

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
```

Example 2: For connecting to Oracle database.

```
Class.forName(“oracle.jdbc.driver.OracleDriver”);
```

Create the connections

The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. The getConnection() method of DriverManager class is used to establish connection with the database.

```
Connection con = DriverManager.getConnection("jdbc:myDriver:myDatabase",
username, password)
```

username - username from which your sql command prompt can be accessed. password - password from which your sql command prompt can be accessed.
con - is a reference to Connection interface.

Example 5: Establishing a connection to JDBC for derby database and creating a table.

//Add the following method after dis() method in CustomerInformation.java. Note that this method should be run only once. Please comment it after running once.

```
public void createDatabase() throws ClassNotFoundException,
SQLException {

Class.forName("org.apache.derby.jdbc.EmbeddedDriver");

Connection con =
DriverManager.getConnection("jdbc:derby:cust;create=true;user=app;
password=app");

String createString = "create table CUSTOMER_INFO(CUSTOMER_ID
INTEGER PRIMARY KEY,\r\n" +
                    "CUSTOMER_NAME VARCHAR(20),\r\n" +
                    "CUSTOMER_EMAIL VARCHAR(50),\r\n" +
                    "CUSTOMER_COUNTRY VARCHAR(20),\r\n" +
                    "CUSTOMER_GENDER VARCHAR(20))";

Statement stmt = con.createStatement();
stmt.executeUpdate(createString);  }
```

//Add the following method after dis() method in CustomerInformation.java

```
public Connection getConnection() throws ClassNotFoundException,
SQLException {

     Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
     Connection con =
DriverManager.getConnection("jdbc:derby:cust;create=true;user=app;
password=app");
     return con;    }
```

Create a statement

Once a connection is established you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

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

Prepared Statements:

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

```
PreparedStatement stmt=con.prepareStatement("insert into Emp value
s(?,?)");    // ? represents parameters.
stmt.setInt(1,101);//1 specifies the first parameter in the query

stmt.setString(2,"Ratan");
```

Callable Statements:

CallableStatement interface is used to call the stored procedures and functions.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Here, con is a reference to Connection interface used in previous step.

Execute the query

Query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

- Query for updating / inserting table in a database.
- Query for retrieving data.

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

The executeUpdate(sql query) method of Statement interface is used to execute queries of updating/inserting .

ResultSet: The java.sql.ResultSet interface represents the result set of a database query.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories −

- Navigational methods: Used to move the cursor around.
- Get methods: Used to view the data in the columns of the current row being pointed by the cursor.
- Update methods: Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

// Add this code after getConnection () method in CustomerInformation.java

```
private int getCustomerID(Connection con) throws SQLException {
     int value = 0;
     Statement stmt = con.createStatement();
```

```
     ResultSet rs = stmt.executeQuery("Select max(CUSTOMER_ID)
from CUSTOMER_INFO");
     if(rs.next()) value = rs.getInt(1);
          return value+1;
     }
 }
```

Close the connections

By closing connection, objects of Statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

// Add this code after getCustomerID() method in CustomerInformation.java

```
public void closeConnection(Connection con) throws SQLException{
          con.close();
       }
```

Example 6: Action on clicking save button and reset button in Example 4

// This code snippet for creating database and table should be run once i.e. for the first time

```
try {
      createDatabase();
} catch (Exception e1) {
      e1.printStackTrace();
}
```

// Add the following code in constructor of CustomerInformation.java

```
resetbtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
            name1.setText("");
            email1.setText("");
            country1.setSelectedIndex(0);
            gender1.setText("");
      }
});

savebtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
            String value1 = name1.getText();
            String value2 = email1.getText();
            String value3 = (String) country1.getSelectedItem();
            String value4 = gender1.getText();
            try {
                  Connection con = getConnection();
                  int customerid = getCustomerID(con);
                  PreparedStatement st = con.prepareStatement(
                              "insert                          into
CUSTOMER_INFO(CUSTOMER_ID,CUSTOMER_NAME,CUSTOMER_EMAIL,CUSTOMER_CO
UNTRY,CUSTOMER_GENDER) values(?,?,?,?,?)");
```

```
                    st.setInt(1, customerid);
                    st.setString(2, value1);
                    st.setString(3, value2);
                    st.setString(4, value3);
                    st.setString(5, value4);
                    st.executeUpdate();

JOptionPane.showMessageDialog(p1, "Data is successfully inserted
into database.");
             }

             catch (ClassNotFoundException e) {
                    JOptionPane.showMessageDialog(p1,   "Error   in
submitting data!");             }

             catch (SQLException ex) {
      Logger.getLogger(CustomerInformation.class.getName()).log(Le
vel.SEVERE, null, ex); }
       }
});
```

Example 7: Adding Action Events to Edit and Save buttons in second Panel.

// Add the following code in the constructor of CustomerInformation.java

```
editbtn1.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ae) {

int value = Integer.parseInt(id2.getText());
          try {
               Connection con = getConnection();
               PreparedStatement st = con.prepareStatement("select
* from CUSTOMER_INFO where CUSTOMER_ID=?");

               st.setInt(1, value);
               ResultSet res = st.executeQuery();
               res.next();

               id2.setText(Integer.toString(res.getInt(1)));
               name2.setText(res.getString(2));
               email2.setText(res.getString(3));

               country2.setSelectedItem(res.getString(4));
               gender2.setText(res.getString(5));
               con.close();
               } catch (ClassNotFoundException e) {
                    JOptionPane.showMessageDialog(p2, "Can not edit
data");
               } catch (SQLException ex) {
```

```
     Logger.getLogger(CustomerInformation.class.getName()).log(Lev
el.SEVERE, null, ex);
                }
           }
});

editbtn2.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ae) {

     int x = JOptionPane.showConfirmDialog(p2, "Confirm edit? All
data will be replaced");

     if (x == 0) {
          try {
               int value1 = Integer.parseInt(id2.getText());
               String value2 = name2.getText();
               String value3 = email2.getText();
               String value4 = (String) country2.getSelectedItem();
               String value5 = gender2.getText();
               Connection con = getConnection();
               Statement st = con.createStatement();

st.executeUpdate("update CUSTOMER_INFO set CUSTOMER_NAME='" +
value2 + "', CUSTOMER_EMAIL='"+ value3 + "',CUSTOMER_COUNTRY='" +
value4 + "',CUSTOMER_GENDER='" + value5 + "' where CUSTOMER_ID=" +
value1 + "");

     JOptionPane.showMessageDialog(p2, "Updated successfully");
     con.close();
     } catch (ClassNotFoundException ex) {
               JOptionPane.showMessageDialog(p2, "Error in updating
edit fields");
     } catch (SQLException ex) {

     Logger.getLogger(CustomerInformation.class.getName()).log(Lev
el.SEVERE, null, ex);         }
     }
}
});
```

ADDITIONAL EXERCISES:

1. Write an Action Event Listener for Delete Button which takes CUSTOMER_ID from text box in third panel and deletes that record from the Database.

2. Add another panel to the Frame. Create a Textbox and a label “Enter SQL Query” for the Textbox. A query related to customer table is the input of the Textbox. Add a button “Execute” to the panel. Create another label “Results” to display the results. Write an Action Event Listener for Execute Button which takes SQL Query from text box in new panel, executes that query and displays the results in Results Label.

Additional Info.:

Creating a Database
Step 1: Open Microsoft Access and select Blank data base option and give the data base name as File name option
Step 2: Create a table and insert your data into the table
Step 3: Save the table with the desired name; in this article we save the following records with the table name student.

Now Creating DSN of your data base
Step 4: Open your Control Panel and then select Administrative Tools.
Step 5: Click on Data Source(ODBC)-->System DSN.
Step 6: Now click on add option for making a new DSN.select Microsoft Access Driver (*.mdb. *.accdb) and then click on Finish
Step 7: Make your desired Data Source Name and then click on the Select option, for example in this article article we us the name mydsn
Step 8: Now you select your data source file for storing it and then click ok and then click on Create and Finish

Connecting to MS Access:

There are two ways to connect java application with the access database.

1. Without DSN (Data Source Name)

```
String database="student.mdb";//Here database exists in the curren
t directory
String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)}; DB
Q=" + database + ";DriverID=22;READONLY=true";
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection c=DriverManager.getConnection(url);
```

2. With DSN

To connect java application with DSN, create DSN first, here we are assuming your dsn name is mydsn.

```
String url="jdbc:odbc:mydsn";
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection c=DriverManager.getConnection(url);
```

References:

1. https://www.javatpoint.com/example-to-connect-to-the-oracle-database
2. https://www.javatpoint.com/example-to-connect-to-the-mysql-database
3. https://www.javatpoint.com/connectivity-with-access-without-dsn