APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KTU SPECIAL

We can together dream the B.tech

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

• KTU STUDY MATERIALS	• SYLLABUS
• KTU LIVE NOTIFICATION	• SOLVED QUESTION PAPERS

JOIN WITH US

WWW.KTUSPECIAL.IN KTUSPECIAL t.me/ktuspecial1

MODULE 4

SOLID Principles in Java (https://www.javatpoint.com/solid-principles-java)
Swings fundamentals – Overview of AWT, Swing v/s AWT, Swing Key Features, Model View Controller (MVC), Swing Controls, Components andContainers, Swing Packages, Event Handling in Swings, Swing Layout Managers, Exploring Swings– JFrame, JLabel, The Swing Buttons,JTextField.
Event handling – Event Handling Mechanisms, Delegation Event Model,Event Classes, Sources of Events, Event Listener Interfaces, Using the Delegation Event Model.
Developing Database Applications using JDBC – JDBC overview, Types,Steps, Common JDBC Components, Connection Establishment, SQL Fundamentals [For projects only] - Creating and Executing basic SQL Queries, Working with Result Set, Performing CRUD Operations with JDBC.

What is AWT?

AWT also known as Abstract window toolkit. It is a platform dependent API used for developing GUI (graphical user interface) or applications that are window based. It was developed by Sun Microsystems In 1995 and is heavy weighted. It is generated by the system's host operating system and contains a large number of methods and classes which are used for creating and managing the UI of an application. The main difference between AWT and Swing is that AWT is purely used for GUI whereas Swing is used for both GUI as well as for making web applications.

What is Swing?

In Java, a swing is a light-weighted, GUI (graphical user interface) that is used for creating different applications. It has platform-independent components and enables users to create buttons as well as scroll bars. It also includes a package for creating applications for desktops. The components of swing are written in Java and are a part of the Java foundation class.

Key Differences Between AWT and Swing

Package:

- AWT: Part of java.awt package.
- Swing: Part of javax.swing package.

Component Design:

- AWT: Uses native system components, hence platform-dependent look.
- Swing: Uses lightweight components, offering a consistent look across platforms.

Performance:

- AWT: Generally slower since it's tied to native GUI components.
- Swing: Faster, more responsive due to lightweight design.

Customizability:

- AWT: Limited customization for components' appearance.
- Swing: Highly customizable with features like 'Look and Feel'.

Difference Between AWT and Swing

AWT	Swing
In Java, awt is an API that is used for developing GUI (Graphical user interface) applications.	Swing on another hand, in Java, is used for creating various applications including the web.
Java AWT components are heavily weighted.	Java swing components are light-weighted.
Java AWT has fewer functionalities as compared to that of swing.	Swing has greater functionalities as compared to awt in Java.
The code execution time in Java AWT is more.	The execution time of code in Java swing is less compared to that of awt.
In Java AWT, the components are platform-dependent.	The swing components are platform-independent.
MVC (Model-View-Controller) is not supported by Java awt.	MVC (Model-View-Controller) is supported by Java swing.
In Java, awt provides less powerful components.	Swing provides more powerful components.
The awt components are provided by the package java.awt	The swing components are provided by javax.swing package
There are abundant features in awt that are developed by developers and act as a thin layer between development and operating system.	Swing has a higher level of in-built components for developers which facilitates them to write less code.

The awt in Java has slower performance as compared to swing.	Swing is faster than that of awt.
AWT Components are dependent on the operating system.	Swing components are completely scripted in Java. Its components are not dependent on the operating system.

Features of Java Swing

1. **Platform Independent**
2. **Customizable**
3. **Plugging**
4. **MVC**
5. **Manageable Look and Feel**
6. **Lightweight**

1. Platform Independent

It is platform-independent, as the swing components used to construct the program are not platform-specific. It works on any platform and in any location.

2. Customizable

Swing controls are simple to customize. It can be changed, and the swing component application's visual appearance is independent of its internal representation.

3. Plugging

Java Swing has a pluggable look and feel. This feature allows users to change the appearance and feel of Swing components without having to restart the application. The Swing library will enable components to have the same look and feel across all platforms, regardless of where the program is running. The Swing library provides an API that allows complete control over the appearance and feel of an application's graphical user interface.

4. MVC

The MVC Relationships:
A visual component is made up of three distinct aspects in general:

- The appearance of the component when it is rendered on the screen.
- The component responds to the user.

- The state information connection with the component.

One component architecture has proven to be exceptionally effective over time.
MVC stands for Model-View-Controller.

- The model agrees with the state information associated with the Component in MVC terminology.
- The view determines how the component appears on screen and any aspects of the view that are influenced by the model's current state.
- The controller is in charge of determining how the component responds to the user.

5. Manageable Look and Feel

It's simple to manage and configure. Its mechanism and composition pattern allows changes to be made to the settings while the program runs. Constant changes to the application code can be made without making any changes to the user interface.

6. Lightweight

Lightweight Components: The JDK's AWT has supported lightweight component development since version 1.1. A component must not rely on non-Java [O/s based] system classes to be considered light. The look and feel classes in Java help Swing components have their view.

You can also read about Why is Java Platform Independent here.

7. Rich Control

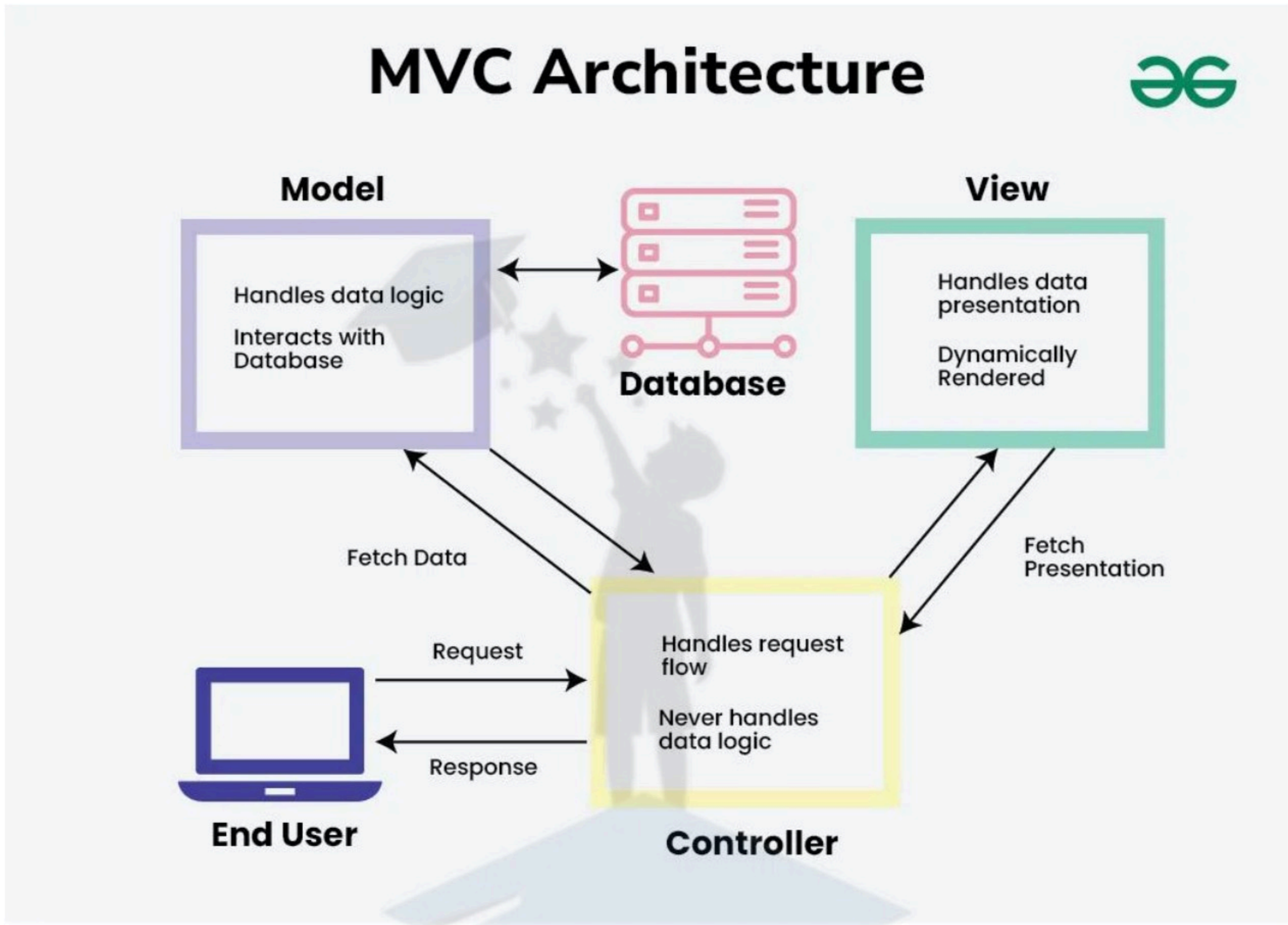
Swing offers a wide variety of rich, interactive controls such as buttons, text fields, tables, sliders, and trees. These components enable the creation of sophisticated user interfaces with advanced features like drag-and-drop support, tooltips, and event handling, enhancing user interaction.

8. Configurable

Swing allows extensive configuration of its components. Developers can set properties such as size, color, font, and event handling, and even define custom behaviours for components. This configurability ensures that the GUI can adapt to various application requirements and user preferences.

What is MVC Architecture?

MVC (Model-View-Controller) architecture is a universal pattern of a structure in which an application is divided into three parts which are all dedicated to certain parts of the whole application. This pattern is normally used in software development to create organized and easy-to-maintain code. Here's a deeper look at each component:



- **Model:** It is worth stating that the Model stands as the data layer for the application. It is directly involved in managing the data as well as the control of the application's logic and rules.
- **View:** The View is in the presentation tier. It plays a role of presenting the information given by the Model to the user and transferring the user commands to the Controller. The View is used to display the data to the user in a readable and manageable way using the interface created by the Controller.
- **Controller:** The Controller CE works in the middle between the Model and the View. It takes the input from the View, sometimes modifies it with the help of the Model, and sends it back to the View. the results back to the View.

The MVC architecture is significant in system design for the following reasons:

- **Separation of Concerns:** MVC structures an application into three integrated elements and that really separates concern. Due to the clear division of

responsibilities among each of the components, the functioning of the application becomes more logical and comprehensible.

- **Reusability:** Due to the fact that Model, View, and Controller are all distinct entities, components can be utilized in the various sections of the application or different projects. For example, a Model class that contains user data can be used many times in the views and the controllers.
- **Scalability:** MVC amplifies the creation of applications that can be developed further. When the application advances, new functionalities of the application can be added without significant alterations to some parts of the application because they are elusive.
- **Testability:** This separation of concerns makes it easier for the testing of each part from the other as we do from other problems. One of the testing strategies is to have separate tests for Model, View, and Controller parts, in this way, one is sure that each part is functioning properly before combining them.

Key Components of MVC Architecture

Here are detailed explanations and examples for each of the key components of the MVC architecture:Here are detailed explanations and examples for each of the key components of the MVC architecture

:**Model:** The Model component brings together all the core details and business knowledge. Therefore, it is concerned with the management of data, states and rules of the application. For example, in a booking system the Model might deal with user data, booking details and with calculations of prices.

Control	Class	Description
Label	JLabel	Displays a short string or an image icon.
Button	JButton	A push button that can trigger an action.
Text Field	JTextField	Allows the user to enter a single line of text.

Password Field	JPasswordField	A text field that hides input (for passwords).
Text Area	JTextArea	Multi-line text input area.
Check Box	JCheckBox	Represents an on/off toggle. Multiple boxes can be selected.
Radio Button	JRadioButton	Allows single selection among grouped options.
Combo Box	JComboBox	A drop-down list of items.
List	JList	Displays a list of items.
Panel	JPanel	A generic container for grouping components.
Frame	JFrame	A top-level window with a title and border.
Dialog	JDialog	A pop-up window for messages or input.
Scroll Pane	JScrollPane	Provides scrollable view for components.
Tree	JTree	Displlays Hierarchial data
Table	JTable	Displays data in a tabular format.

View: The View component is responsible for the presentation layer. It takes the data provided by the Model and presents it to the user in a readable format.

Controller: The Controller component handles user input and interactions. It processes the input, interacts with the Model, and updates the View accordingly.

Common Swing Controls (Components)

In Java, Swing is a part of Java Foundation Classes (JFC) used to create Graphical User Interfaces (GUIs). It provides a rich set of lightweight components that work the same across all platforms.

Basic Example Using Swing Controls

```
import javax.swing.*;

public class SwingExample {

    public static void main(String[] args) {

// Create frame

JFrame frame = new JFrame("Swing Example");

frame.setSize(300, 200);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Create components

JLabel label = new JLabel("Enter Name:");

JTextField textField = new JTextField(15);

JButton button = new JButton("Submit");

// Add components to panel
```

```
JPanel panel = new JPanel();

panel.add(label);

panel.add(textField);

panel.add(button);

        // Add panel to frame

        frame.add(panel);

        frame.setVisible(true);

    }

}
```

Components and Containers

In Java, Swing is a part of Java Foundation Classes (JFC) used to create Graphical User Interfaces (GUIs). It provides a rich set of lightweight components that work the same across all platforms.

Common Swing Controls (Components)

Control	Class	Description
Label	JLabel	Displays a short string or an image icon.

Button	JButton	A push button that can trigger an action.
Text Field	JTextField	Allows the user to enter a single line of text.
Password Field	JPasswordField	A text field that hides input (for passwords).
Text Area	JTextArea	Multi-line text input area.
Check Box	JCheckBox	Represents an on/off toggle. Multiple boxes can be selected.
Radio Button	JRadioButton	Allows single selection among grouped options.
Combo Box	JComboBox	A drop-down list of items.
List	JList	Displays a list of items.
Panel	JPanel	A generic container for grouping components.
Frame	JFrame	A top-level window with a title and border.
Dialog	JDialog	A pop-up window for messages or input.
Scroll Pane	JScrollPane	Provides scrollable view for components.
Table	JTable	Displays data in a tabular format.

Tree	JTree	Displays hierarchical data.

Basic Example Using Swing Controls

```
import javax.swing.*;

public class SwingExample {

    public static void main(String[] args) {

        // Create frame

        JFrame frame = new JFrame("Swing Example");

        frame.setSize(300, 200);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create components

        JLabel label = new JLabel("Enter Name:");

        JTextField textField = new JTextField(15);

        JButton button = new JButton("Submit");

        // Add components to panel

        JPanel panel = new JPanel();

        panel.add(label);

        panel.add(textField);

        panel.add(button);
```

```
        // Add panel to frame
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

Notes

- All Swing classes are in the javax.swing package.
- Swing components are lightweight, unlike AWT which uses native code.
- Event handling in Swing is done via Event Listeners (e.g., ActionListener, ItemListener).
- Layout managers (FlowLayout, BorderLayout, GridLayout, etc.) help organize components in a GUI.
- In Java GUI programming (especially using AWT and Swing), Components and Containers are two fundamental building blocks:

---

1. Component

A Component is an object that represents a GUI element. It can be anything that can be seen and interacted with in a window — like buttons, labels, text fields, etc.

- All components inherit from java.awt.Component.
- Examples of components:

- Button
- Label
- TextField
- Checkbox
- List
- Canvas
- In Swing: JButton, JLabel, JTextField, etc.

Examples:

```
Button b = new Button("Click Me");

Label l = new Label("Hello, World!");
```

---

2. Container

A Container is a special type of component that can hold other components (including other containers).

- Containers help in grouping and organizing components.
- Every container is a subclass of java.awt.Container.
- Containers often use layout managers to control how components are arranged.

Types of Containers:

Container	Description
Panel	A simple container for holding components.
Frame	A top-level window with a title bar and border.
Dialog	A pop-up window for messages or input.
Applet	A container used in web-based Java programs.

In Swing: JPanel, JFrame, JDialog, etc.

Example:

```
Frame f = new Frame("Example Frame");

Panel p = new Panel();

Button b = new Button("Click Me");

p.add(b);   // Adding component to container

f.add(p);   // Adding panel to frame

f.setSize(300, 200);

f.setVisible(true);
```

Component vs Container

Feature	Component	Container
Inherits from	java.awt.Component	java.awt.Container
Can hold other components	No	Yes
Examples	Button, Label, TextField	Panel, Frame, Applet

Swing Packages

In Java, Swing is a part of the Java Foundation Classes (JFC) used for building Graphical User Interfaces (GUIs). The Swing packages contain classes for components, containers, event handling, borders, pluggable look-and-feel, and more.**Main Swing Packages in Java**

Package	Description
javax.swing	Core package for all Swing components like JButton, JLabel, JFrame, etc.
javax.swing.event	Contains classes and interfaces for event handling specific to Swing (e.g., ListSelectionEvent, ChangeListener).
javax.swing.border	Provides classes for drawing borders around components (BevelBorder, LineBorder, etc.).

javax.swing.table	Contains classes for handling table components (JTable, TableModel, etc.).
javax.swing.text	Supports customizable text components (JTextComponent, StyledDocument, etc.).
javax.swing.tree	Used for tree components (JTree, TreeModel, etc.).
javax.swing.plaf	Supports the Pluggable Look-and-Feel architecture of Swing.
javax.swing.filechooser	Contains classes related to file choosing dialogs (JFileChooser).
javax.swing.colorchooser	Contains classes for choosing colors (JColorChooser).
javax.swing.text.html	Support for HTML content in text components.
javax.swing.text.html.parser	Parser classes for HTML document structures.

Example Using javax.swing

```
import javax.swing.*;

public class SwingDemo {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Swing Package Example");
```

```
JButton button = new JButton("Click Me");

JLabel label = new JLabel("Hello, Swing!");

JPanel panel = new JPanel();

panel.add(label);

panel.add(button);

frame.add(panel);

frame.setSize(300, 200);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setVisible(true);

    }

}
```

Notes

- Most common Swing components are in javax.swing.
- Swing is platform-independent and lightweight.
- Event handling is usually done using listeners from java.awt.event and javax.swing.event

Event Handling in Swings

In Java Swing, events are handled using an event-driven programming model. Here's how it works:

1. **Event Source:**
    - The component that generates an event, such as a button (JButton), text field (JTextField), or menu (JMenuItem).
2. **Event Listener:**
    - An object that listens for and responds to events. Event listeners are implemented using specific interfaces such as:
        - ActionListener: For handling button clicks and other actions.
        - KeyListener: For handling keyboard events.
        - MouseListener: For handling mouse events.
        - Others like FocusListener, WindowListener, etc.
3. **Event Registration:**
    - The listener is registered with the event source using methods like addActionListener(), addKeyListener(), etc.
4. **Event Object:**
    - When an event occurs, an event object (e.g., ActionEvent, MouseEvent) is created and passed to the listener's method.
5. **Handling Events:**
    - The listener's method executes the required action when the event is triggered.

Example

```
JButton button = new JButton("Click Me");
button.addActionListener(e -> System.out.println("Button clicked!"));
```

Layout Manager-Java Swing

The LayoutManagers are used to arrange components in a particular manner. The Java LayoutManagers facilitate us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Some of the mainly used layouts are as follows:

BorderLayout:

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only.

1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

Constructors of BorderLayout class:

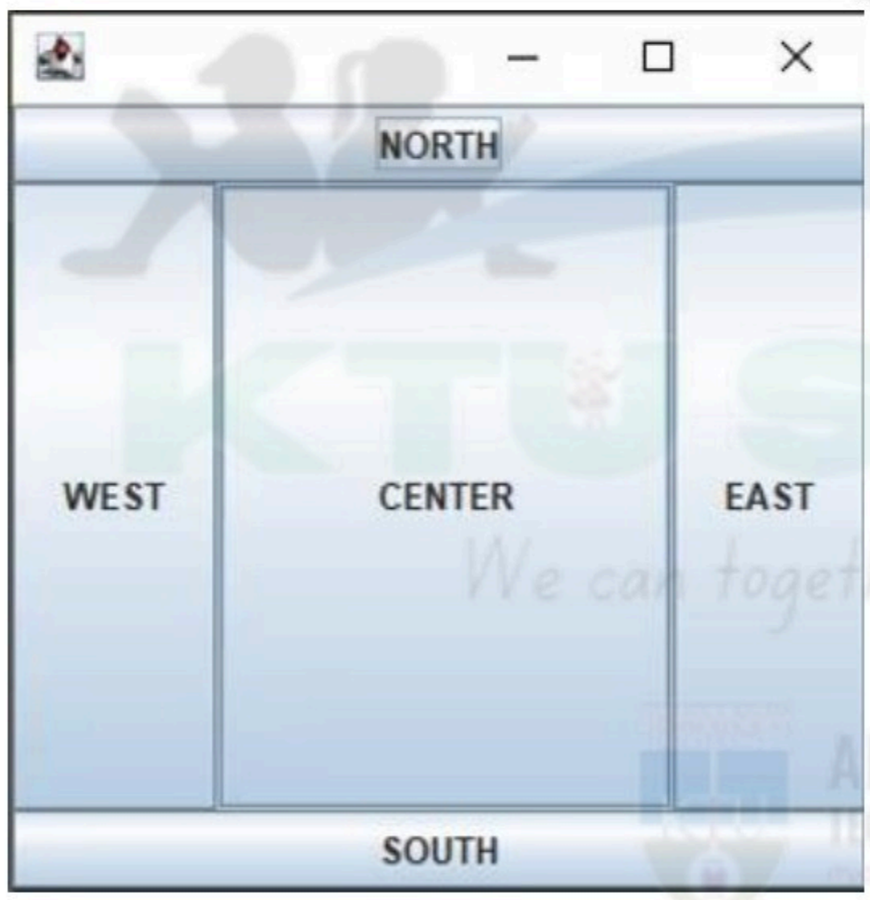
- BorderLayout(): creates a border layout but with no gaps between the components.
- BorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

Example:

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

public class Border
{
    JFrame f;
    Border()
    {
        f = new JFrame();

// creating buttons
JButton b1 = new JButton("NORTH");;
JButton b2 = new JButton("SOUTH");;
JButton b3 = new JButton("EAST");;
JButton b4 = new JButton("WEST");;
JButton b5 = new JButton("CENTER");;
```

```
f.add(b1, BorderLayout.NORTH);
f.add(b2, BorderLayout.SOUTH);
f.add(b3, BorderLayout.EAST);
f.add(b4, BorderLayout.WEST);
f.add(b5, BorderLayout.CENTER);

    f.setSize(300, 300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new Border();
}
 }
```

output



GridLayout:

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

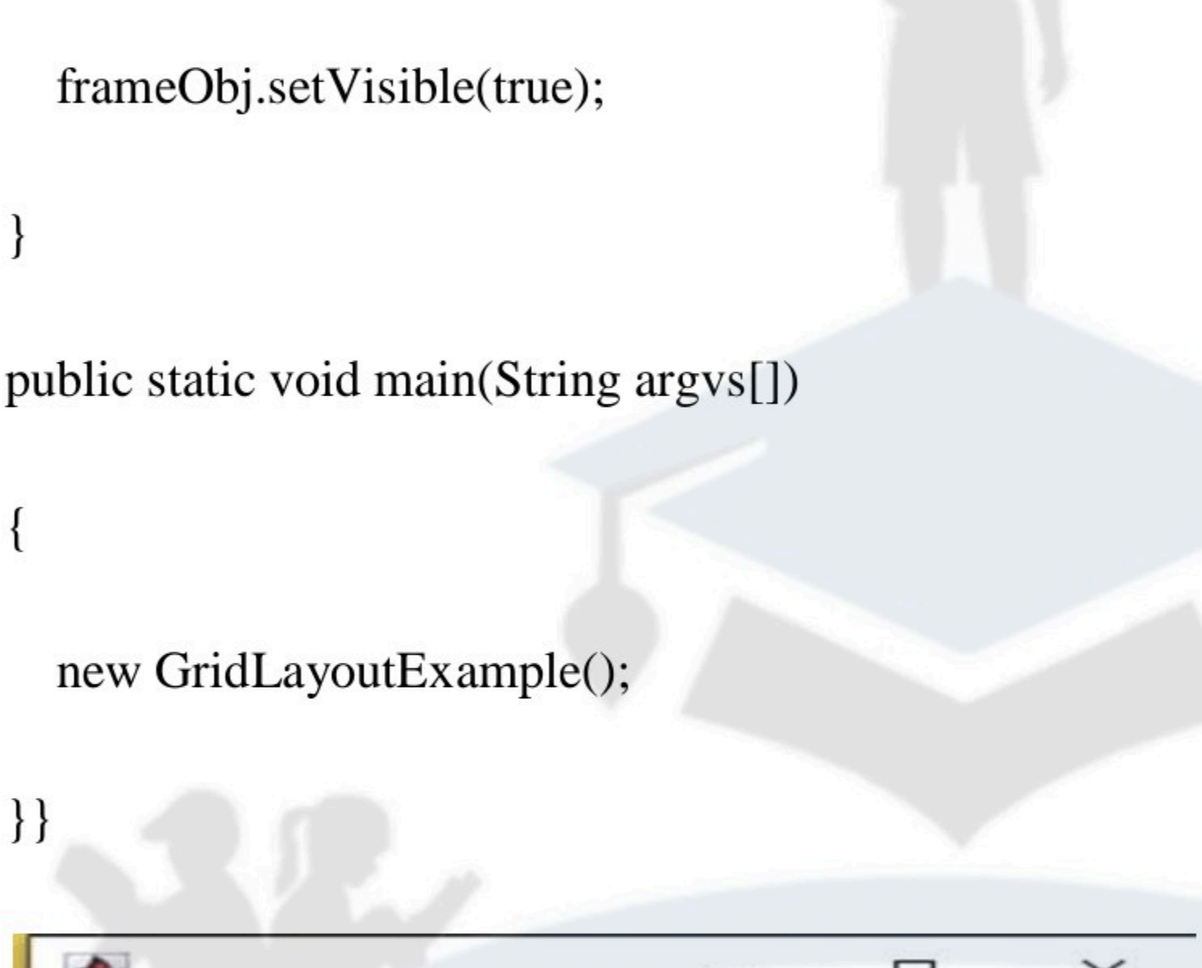
Constructors of GridLayout class

1. GridLayout(): creates a grid layout with one column per component in a row.
2. GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
3. GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example:

```
import java.awt.*;

import javax.swing.*;

public class GridLayoutExample

{

    JFrame frameObj;

GridLayoutExample()

{

    frameObj = new JFrame();

    JButton btn1 = new JButton("1");

    JButton btn2 = new JButton("2");

    JButton btn3 = new JButton("3");

JButton btn4 = new JButton("4");

JButton btn5 = new JButton("5");

JButton btn6 = new JButton("6");

JButton btn7 = new JButton("7");
```

```
JButton btn8 = new JButton("8");

JButton btn9 = new JButton("9");

frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);

frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);

   frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);

   frameObj.setLayout(new GridLayout());

   frameObj.setSize(300, 300);

   frameObj.setVisible(true);

}

public static void main(String argvs[])

{

   new GridLayoutExample();

}}
```



FlowLayout:

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

1. public static final int LEFT
2. public static final int RIGHT
3. public static final int CENTER
4. public static final int LEADING
5. public static final int TRAILING

Constructors of FlowLayout class

1. FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

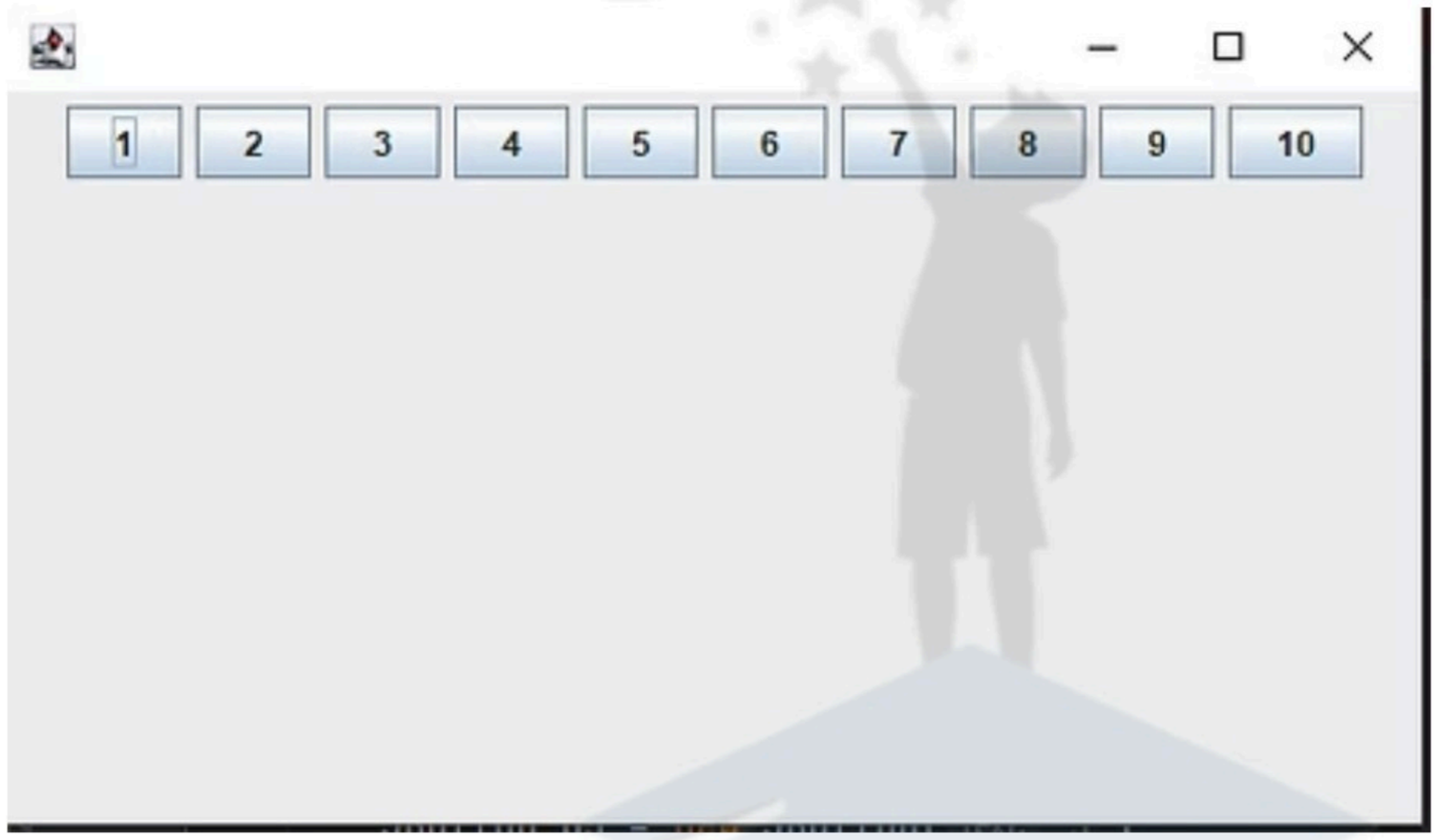
Example:

```
import java.awt.*;

import javax.swing.*;

public class FlowLayoutExample

{

    JFrame frameObj;

    FlowLayoutExample()

{

    frameObj = new JFrame();
```

```
JButton b1 = new JButton("1");

JButton b2 = new JButton("2");

JButton b3 = new JButton("3");

JButton b4 = new JButton("4");

JButton b5 = new JButton("5");

JButton b6 = new JButton("6");

JButton b7 = new JButton("7");

JButton b8 = new JButton("8");

JButton b9 = new JButton("9");

JButton b10 = new JButton("10");

frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);

    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);

    frameObj.add(b9);  frameObj.add(b10);

    frameObj.setLayout(new FlowLayout());

    frameObj.setSize(300, 300);

    frameObj.setVisible(true);

}

public static void main(String argvs[])

{

    new FlowLayoutExample();
```

```
}

}
```

output



BoxLayout:

The Java BoxLayout class is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

Fields of BoxLayout Class

1. public static final int X_AXIS: Alignment of the components are horizontal from left to right.
2. public static final int Y_AXIS: Alignment of the components are vertical from top to bottom.
3. public static final int LINE_AXIS: Alignment of the components is similar to the way words are aligned in a line,
4. public static final int PAGE_AXIS: Alignment of the components is similar to the way text lines are put on a page,

Constructor of BoxLayout class

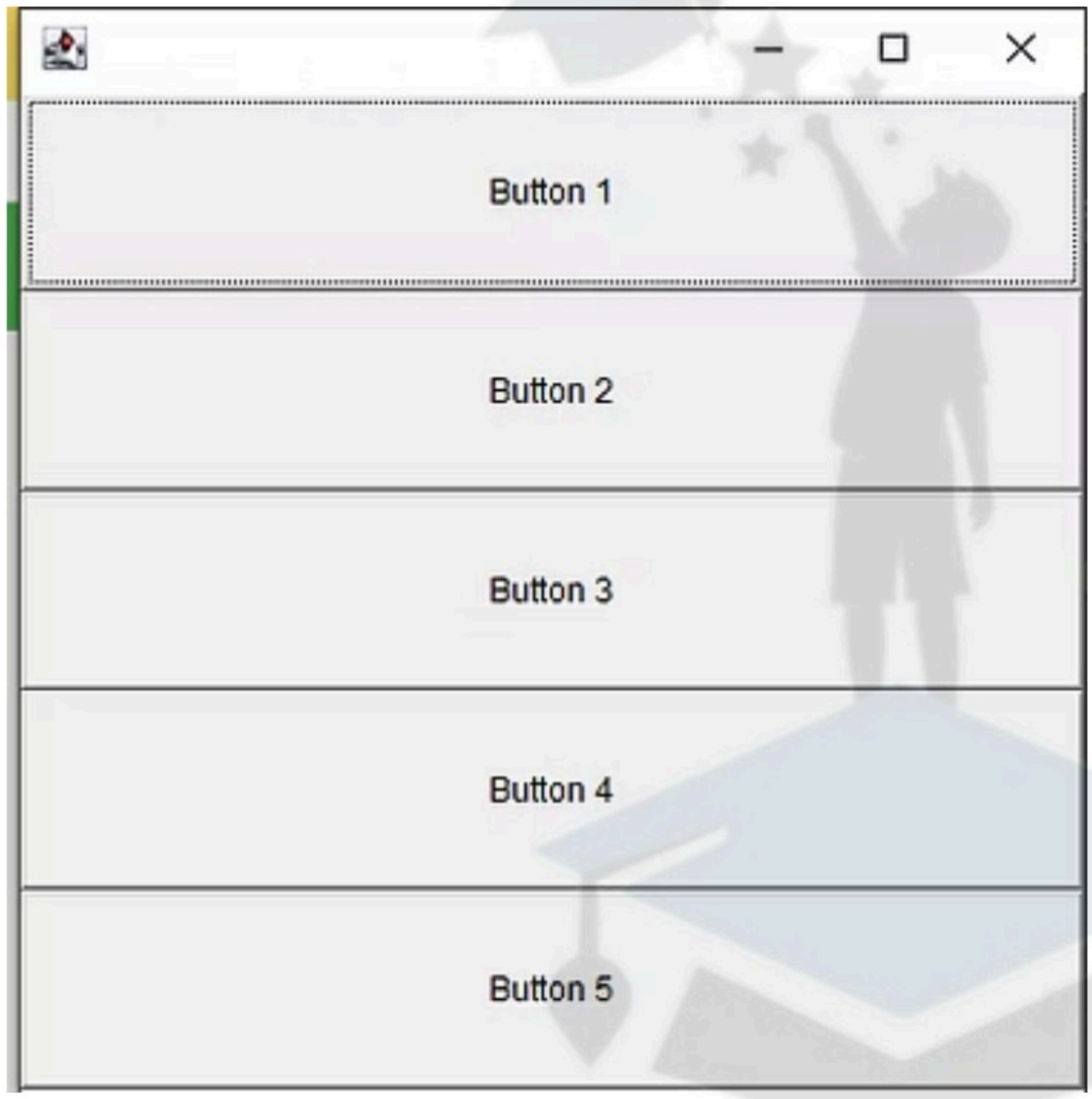
1. BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.

Example:

```
import java.awt.*;

import javax.swing.*;

class BoxLayoutExample1 extends Frame {

    Button buttons[];

    public BoxLayoutExample1 () {

        buttons = new Button [5];

        for (int i = 0;i<5;i++) {

            buttons[i] = new Button ("Button " + (i + 1));

            add (buttons[i]);

        }

    setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));

    setSize(400,400);

    setVisible(true);

    }

public static void main(String args[]){

    BoxLayoutExample1 b=new BoxLayoutExample1();
```

```
    }

}
```

Output:



CardLayout:

The Java CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout Class

1. CardLayout(): creates a card layout with zero horizontal and vertical gap.
2. CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

- public void next(Container parent): is used to flip to the next card of the given container.
- public void previous(Container parent): is used to flip to the previous card of the given container.

- public void first(Container parent): is used to flip to the first card of the given container.
- public void last(Container parent): is used to flip to the last card of the given container.
- public void show(Container parent, String name): is used to flip to the specified card with the given name.

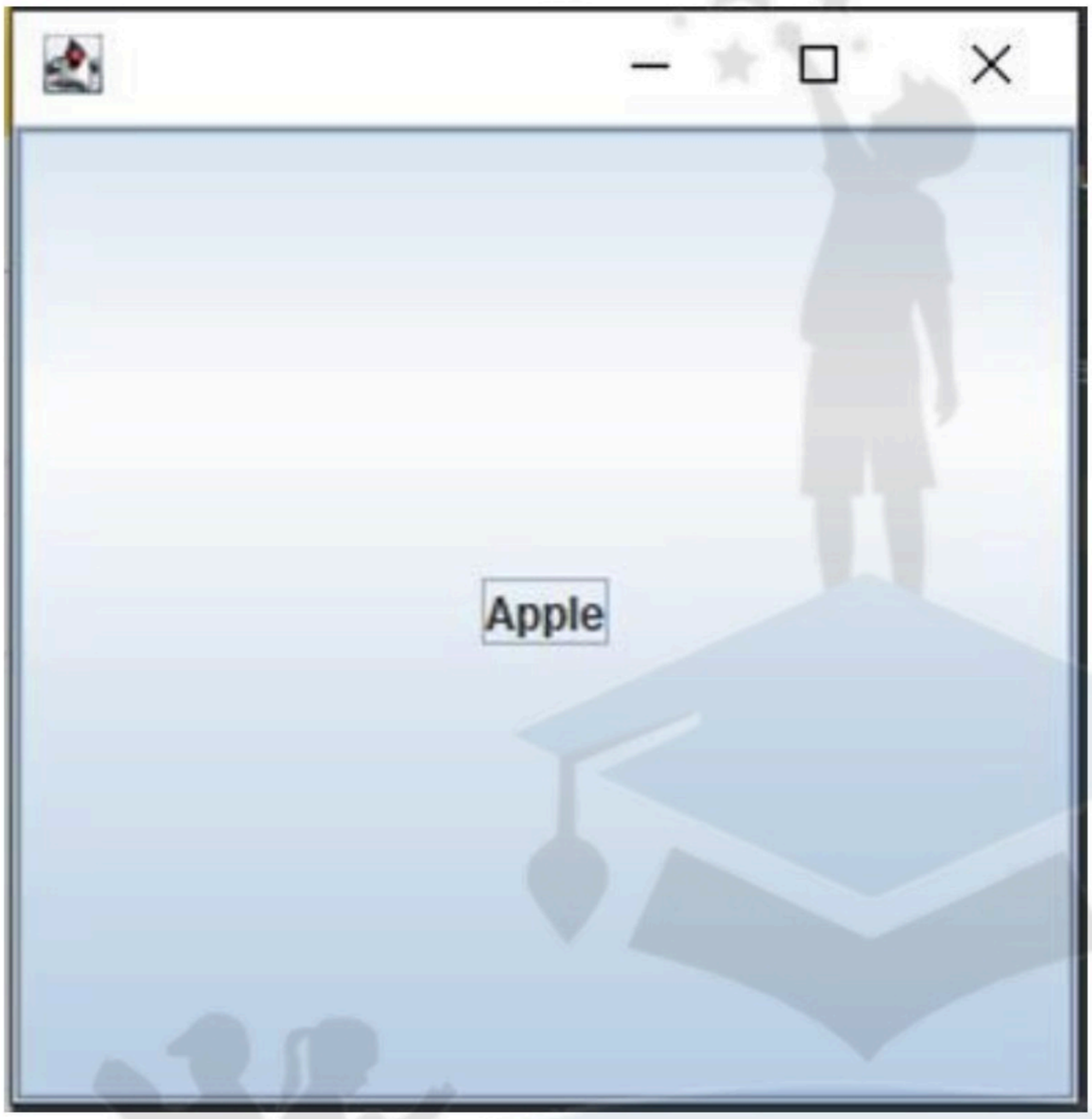
Example:

```
import java.awt.*;

import javax.swing.*;

import java.awt.event.*;

class CardLayoutExample1 extends JFrame implements ActionListener

{

    CardLayout crd;

JButton btn1, btn2, btn3;

Container cPane;

CardLayoutExample1()

{

    cPane = getContentPane();

    crd = new CardLayout();

cPane.setLayout(crd);

btn1 = new JButton("Apple");
```

```
btn2 = new JButton("Boy");

btn3 = new JButton("Cat");

btn1.addActionListener(this);

btn2.addActionListener(this);

    btn3.addActionListener(this);

    cPane.add("a", btn1);

    cPane.add("b", btn2);

    cPane.add("c", btn3);

}

public void actionPerformed(ActionEvent e)

{

   crd.next(cPane);

}

public static void main(String argvs[])

    {

CardLayoutExample1 crdl = new CardLayoutExample1();

        crdl.setSize(300, 300);

        crdl.setVisible(true);
```

```
        crdl.setDefaultCloseOperation(EXIT_ON_CLOSE);

    }

    }
```

Output:



3.Exploring Swings –JFrame, JLabel, The Swing Buttons, JTextField.

Swing Overview

Swing is a part of Java's Java Foundation Classes (JFC) for building graphical user interfaces (GUIs).

Provides lightweight, platform-independent components.

Found in the javax.swing package.

1.JFrame

A top-level container used to create a window.

Represents the main application window.

Methods:

setTitle(String title) - Sets the window title.

setSize(int width, int height) - Sets the size of the window.

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) - Specifies the action on closing the window.

setVisible(boolean) - Makes the window visible.

Example

```
JFrame frame = new JFrame("My Window");

frame.setSize(400, 300);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setVisible(true);
```

2.JLabel

- Used to display a short string or an image icon.
- Cannot be edited by the user.

  Methods:

  - setText(String text) - Sets the text on the label.
  - setIcon(Icon icon) - Sets an image icon.

Example

```
JLabel label = new JLabel("Hello, Swing!");

frame.add(label); // Add label to JFrame
```

Swing Buttons

Buttons allow user interaction via clicks.

Common Swing buttons:

1.JButton: A standard push button.

Example

```
JButton button = new JButton("Click Me");

button.addActionListener(e -> System.out.println("Button Clicked!"));

frame.add(button);
```

2.JCheckBox: A box that can be checked or unchecked.

Example

```
JCheckBox checkBox = new JCheckBox("Check Me");

frame.add(checkBox);
```

3.JRadioButton: A button that is part of a group, allowing single selection.

Example

```
JRadioButton radioButton = new JRadioButton("Option 1");

frame.add(radioButton);
```

JTextField

- A text input field for single-line text.

  Methods:

  - getText() - Retrieves the text entered by the user.
  - setText(String text) - Sets the initial text.

Example:

```
JTextField textField = new JTextField(20); // 20 columns

frame.add(textField);

String input = textField.getText(); // Retrieve input
```

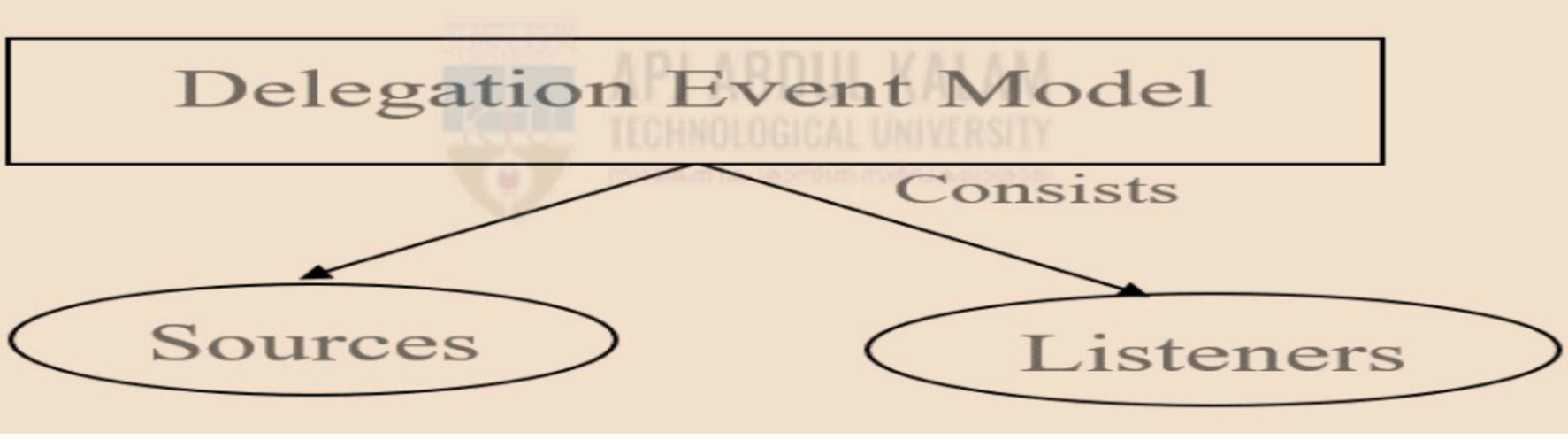
Here's an example combining the components:

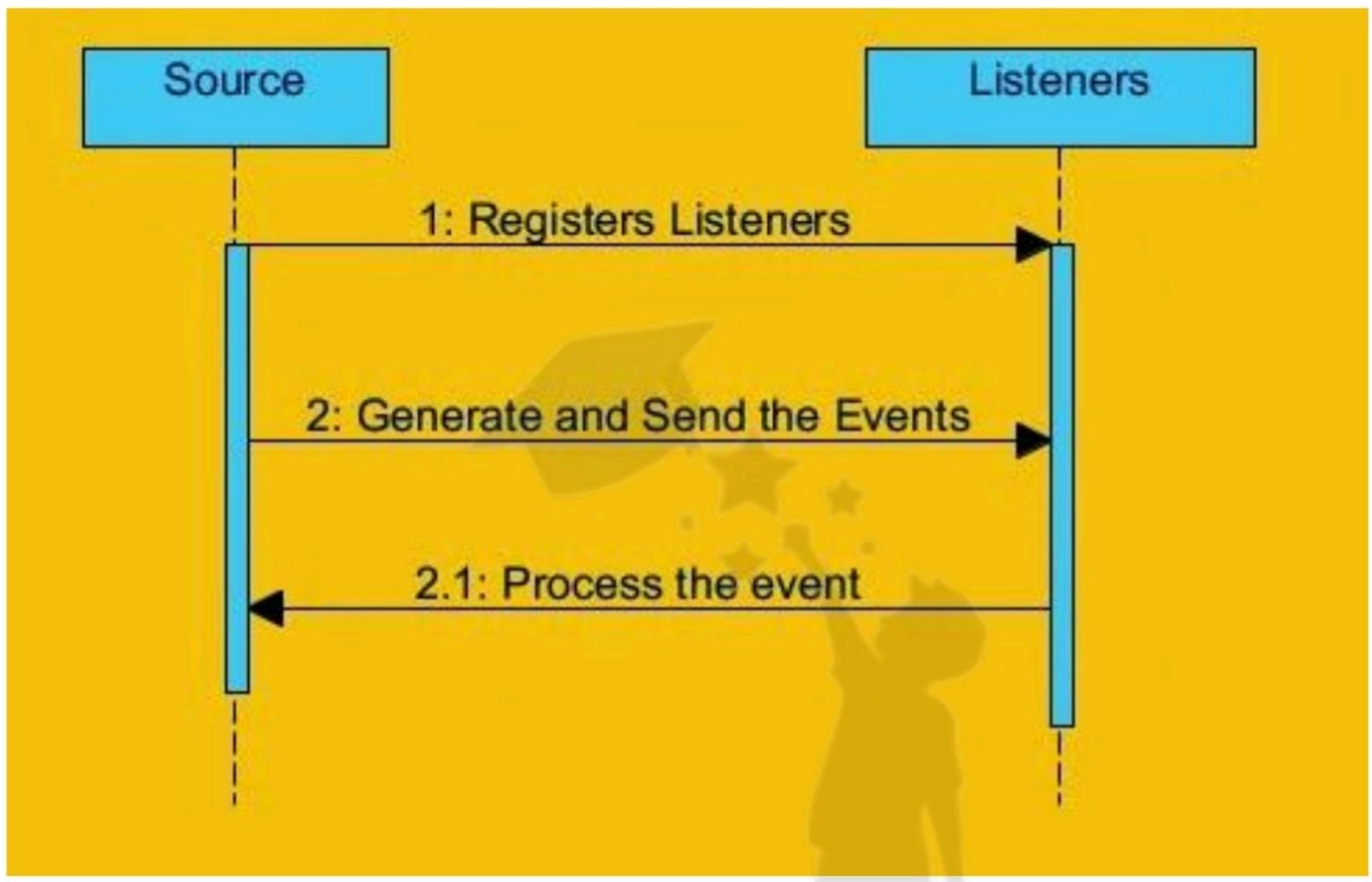
```
import javax.swing.*;
```

```
public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Example");
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         JLabel label = new JLabel("Enter your name:");
        JTextField textField = new JTextField(15);
       JButton button = new JButton("Submit");
button.addActionListener(e -> System.out.println("Hello, " + textField.getText() + "!"));
JPanel panel = new JPanel(); // A container for layout
        panel.add(label);
        panel.add(textField);
        panel.add(button);
      frame.add(panel);
        frame.setVisible(true);
    }
}
```

Event Handling in Java Using the Delegation Event Model

The Delegation Event Model is the standard mechanism in Java for handling events. It is used extensively in Java Swing and AWT for implementing interactive applications. Below is an explanation of how it works:





1. Core Concepts of Delegation Event Model

1. Event Source:
   - The component that generates an event (e.g., a button, text field, or checkbox).
   - When an event occurs (e.g., a button click), the event source notifies the registered listeners.
2. Event Listener:
   - An object that listens for events and handles them.
   - It must implement a specific interface, such as ActionListener, KeyListener, or MouseListener.
   - The event listener must be registered with the event source to receive notifications.
3. Event Object:
   - An instance of a class derived from the java.util.EventObject class.
   - It encapsulates information about the event (e.g., the source of the event and details of the action performed).
   - Examples include:
     - ActionEvent (e.g., button clicks)
     - MouseEvent (e.g., mouse clicks)
     - KeyEvent (e.g., key presses)

2. How the Delegation Event Model Works

1. Event Generation:
    - When the user interacts with a GUI component, an event is generated by the Event Source.
2. Event Propagation:
    - The event object is created and passed to the registered Event Listener(s).
3. Event Handling:
    - The listener responds to the event by executing its defined logic.
    - This separation of event generation and event handling is key to the Delegation Event Model.

3. Key Steps in Handling Events

1. Implement the Listener Interface:
    - Write a class that implements the required event listener interface (e.g., ActionListener).
2. Register the Listener:
    - Attach the listener to the source component using methods like addActionListener() or addMouseListener().
3. Handle the Event:
    - Implement the listener method (e.g., actionPerformed()) to define what happens when the event occurs.

. Common Listener Interfaces

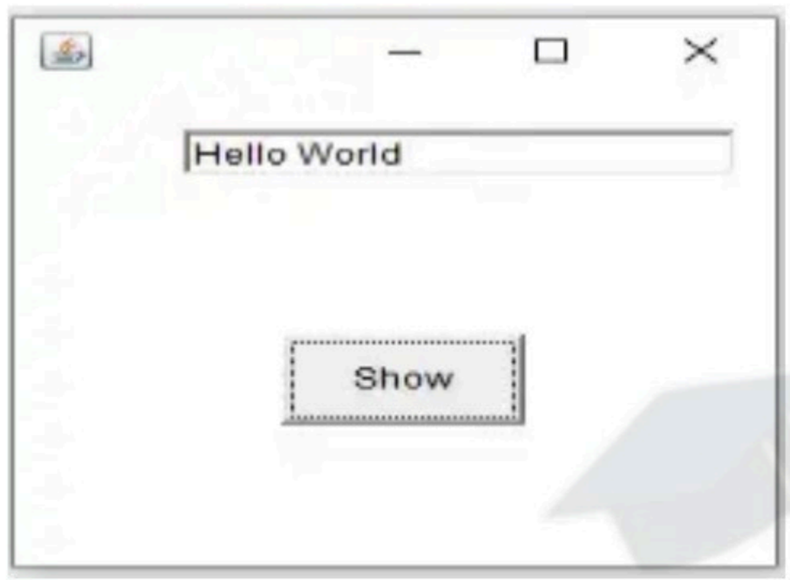
Listener Interface	Description
ActionListener	Handles action events (e.g., button clicks).
MouseListener	Handles mouse events (e.g., clicks, enters, exits).
KeyListener	Handles keyboard events (e.g., key presses).
FocusListener	Handles focus events (e.g., gaining or losing focus).
WindowListener	Handles window events (e.g., opening, closing, minimizing).

EXAMPLE

```
import java.awt.*;

import java.awt.event.*;

class EventHandling extends Frame implements ActionListener
```

```
{
    TextField textField;
    EventHandling ()
    {
        textField = new TextField ();
        textField.setBounds (60, 50, 170, 20);
Button button = new Button ("Show");
button.setBounds (90, 140, 75, 40);
button.addActionListener (this);
add (button);
add (textField);
setSize (250, 250);
setLayout (null);
setVisible (true);
}
public void actionPerformed (ActionEvent e)
{
    textField.setText ("Hello World");
}
public static void main (String args[])
{
    new EventHandling ();
    }
}
```

OUTPUT



JDBC(Java Database Connectivity)

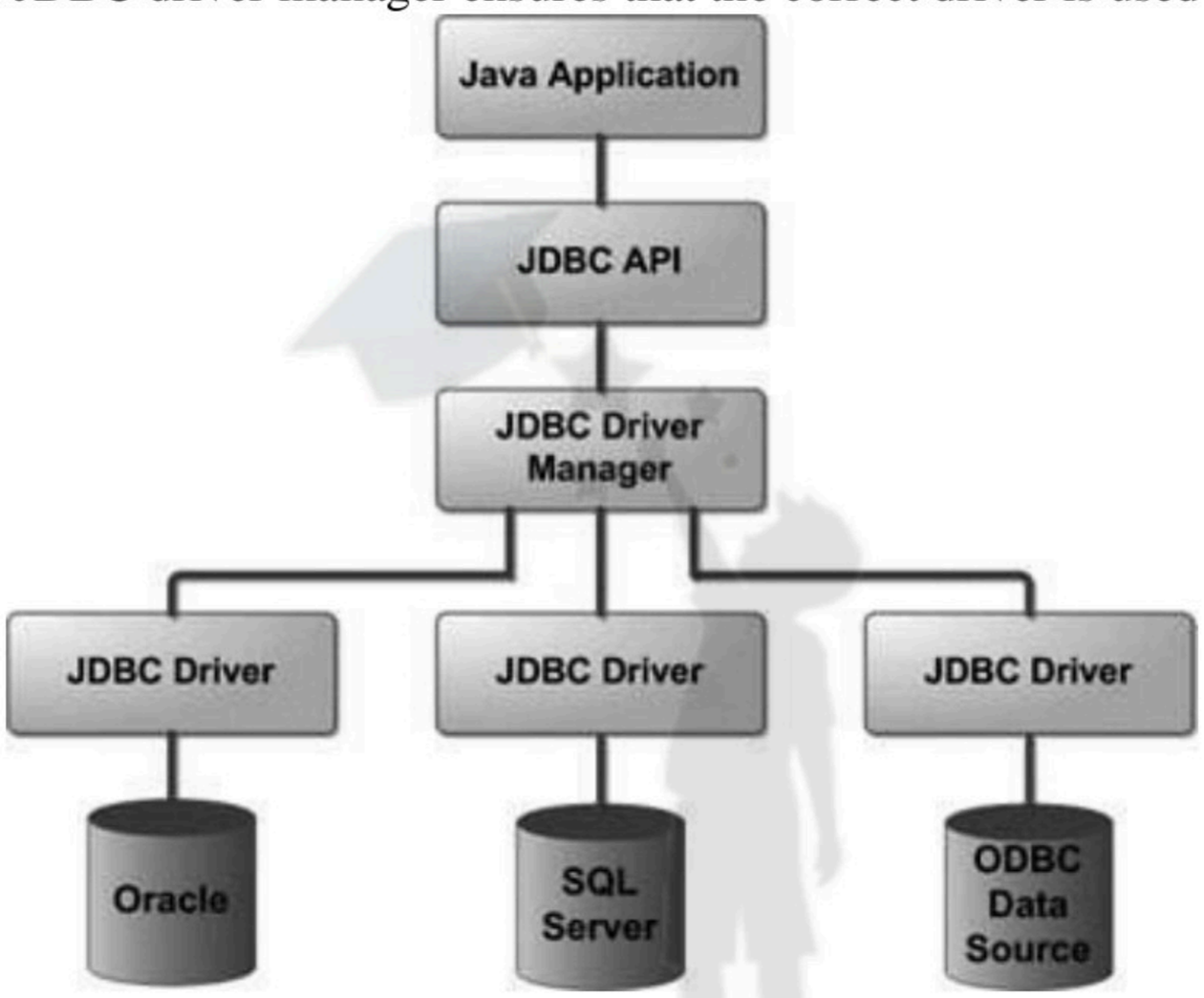
- JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below
  - Making a connection to a database.
  - Creating SQL or MySQL statements.
  - Executing SQL or MySQL queries in the database.
  - Viewing & Modifying the resulting records.

JDBC Architecture

- DBC Architecture consists of two layers −
  - **JDBC API:** This provides the application-to-JDBC Manager connection.
  - **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager and database-specific drivers to provide

transparent connectivity to heterogeneous databases.

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



source.

- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases

- **JDBC API :** The JDBC API provides programmatic access to relational data from the Java programming language.
    - Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source.
    - The JDBC API can also interact with multiple data sources in a distributed, heterogeneous environment.
- **DriverManager:** This class manages a list of database drivers.
- **Driver:** This interface handles the communications with the database server.

Common JDBC Components

The JDBC API provides the following interfaces and classes –

- **DriverManager:** This class manages a list of database drivers.
    - Matches connection requests from the java application with the proper database driver using communication sub protocol.
    - The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the communications with the database server.
    - Use DriverManager objects, which manages objects of this type.
    - It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database.
    - The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute a batch of commands.

- ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- SQLException: This class handles any errors that occur in a database application.

Steps to connect to the database in java

They are as follows:

- Import JDBC Packages
- Register the driver class
- Creating connection
- Creating statement

- Executing queries
- Closing connection

Import JDBC Packages:

- Add import statements to your Java program to import required classes in your Java code.
- Syntax : import java.sql.* ; // for standard JDBC programs

Register the driver class :

Approach 1 :

- The forName() method of "Class" class is used to register the driver class.
- This method is used to dynamically load the driver class.
- Syntax of forName() method :
  - public static void forName(String className)throws ClassNotFoundException
- Eg :- Class.forName("com.mysql.jdbc.Driver");

Approach 2:

- The second approach you can use to register a driver, is to use the static DriverManager.registerDriver() method.
- Syntax :- DriverManager.registerDriver(Driver_object);
- Eg:-

```
Driver myDriver = new com.mysql.jdbc.Driver (); //creating object of mysql
driver DriverManager.registerDriver( myDriver ); //register driver
```

Create the connection object :

- The getConnection() method of the DriverManager class is used to establish connection with the database.
- Syntax of getConnection() method :-

- public static Connection getConnection(String url)throws SQLException
- public static Connection getConnection(String url,String name,String password)throws SQLException

- Eg:-
  Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo","root","mysql");

Create the Statement object :

- Syntax :- public Statement createStatement()throws SQLException
- Eg:- Statement stmt=con.createStatement();

Execute the query :

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

- Syntax of executeQuery() method
  - public ResultSet executeQuery(String sql)throws SQLException
- Example to execute query

```
ResultSet
rs=stmt.executeQuery("select *
from emp"); while(rs.next())
{
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Close the connection object :

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

- Syntax of close() method
  - public void close()throws SQLException
- Example to close connection
  - con.close();

Programs

1.

```
//Java program to display the contents of table emp
import java.sql.*;
class MysqlCon

{
public static void main(String args[])
{
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo","root","mysql");
//here sonoo is database name, root is
username and password Statement
stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from
emp"); while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3)); con.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

2.

```
//Java program to insert the contents into table emp
import java.sql.*;
class MysqlCon
{
public static void main(String args[])
{
try
{
Class.forName("com.mysql.jdbc.Driver");
```

```
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo","root","mysql");
//here sonoo is database name, root is username and password

Statement stmt=con.createStatement();
stmt.executeUpdate("inser into emp
values(3,'test','test')"); con.close();
}
catch(Exception e)
{
System.out.println(e);
} } }
```

Different Database Queries :-

- To create a database:

  stmt.executeUpdate("create database sonoo ");

- To drop Database:

  stmt.executeUpdate("drop database sonoo ");

- To create table:

  stmt.executeUpdate("create table emp(id integer,name varchar(20),designation varchar(20)) ");

- To drop table :

  stmt.executeUpdate("drop table emp ");

- Select specified number of fields:-

  ResultSet rs=stmt.executeQuery("select name from emp"); while(rs.next())
  System.out.println(rs.getString("name")); or
  System.out.println(rs.getString(1));

- Select according to condition:-

  ResultSet rs=stmt.executeQuery("select name from emp where id=1"); while(rs.next())
  System.out.println(rs.getString("name")); or
  System.out.println(rs.getString(1));

- Update query:

  stmt.executeUpdate("update emp set designation=AP where id=3 ");

- Delete Query:

  stmt.executeUpdate("delete from emp where id=1 ");

- Select datas by sorting them in descending according to the total mark rs=stmt.executeQuery("select roll, name from student order by mark1+mark2 desc;"); while(rs.next())
  {
  i++;
  System.out.println(i+"\t"+rs.getInt(1)+"\t"+rs.getString(2));
  }

PreparedStatement interface

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

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

- Syntax:-public PreparedStatement prepareStatement(String

query)throws SQLException

Methods of PreparedStatement interface

Eg:-

//Java program to insert the contents into table emp using prepared statement

```
import java.sql.*;
class InsertPrepared
{
public static void main(String args[])
{
try{
Class.forName("co
m.mysql.jdbc.Driv
er ");
 Connection
  con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo","root","mysql”);
  PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first
   parameter in the query
   stmt.setString(2,"Ratan");
    int
   i=stmt.executeUpd
   ate();
   System.out.println(
   i+" records
   inserted");
   con.close();

catch(Exception e)
   {
      System.out.println(e);
}
 }
}
```

To display contents of a table using prepared statement :-

PreparedStatement

```
stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next())
{
        System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

To delete records of a table using prepared statement :-

```
PreparedStatement stmt=con.prepareStatement("delete from emp where
id=?"); stmt.setInt(1,101);
 int i=stmt.executeUpdate();
System.out.println(i+"
records deleted");
```

Short Answer Questions

1. What does the Liskov Substitution Principle state?
2. How does Swing differ from AWT in terms of component rendering?
3. What is the purpose of JFrame in a Swing application?
4. Mention three commonly used Swing controls and their purposes.
5. Define event source and give two examples from Java Swing.
6. What is DriverManager in JDBC?
7. How does executeQuery() differ from executeUpdate() in JDBC?
8. What is MVC architecture and how is it used in Swing applications?
9. List any three components from the javax.swing package.
10. Write the syntax to register a MouseListener to a component.
11. Mention the steps involved in handling events using the Delegation Event Model.
12. What is the difference between Statement and PreparedStatement in JDBC?

Essay Questions

1. Describe each of the SOLID principles with appropriate Java examples and explain how they improve software quality.
2. Create a Java Swing program for a feedback form using JFrame, JTextArea, JButton, and JLabel. Include event handling to display submitted feedback.

3. Write a Java program to connect to a MySQL database using JDBC. Insert, display, update, and delete student records using PreparedStatement.

4. Explain in detail the Delegation Event Model. Illustrate how event handling works using listeners with a Java example.

5. Design a Java Swing GUI that accepts two numbers and performs basic arithmetic operations (Add, Subtract, Multiply, Divide) using buttons. Implement action listeners for each.

6. Explain the different types of JDBC drivers. Compare their advantages and use cases.

7. Discuss the structure and use of layout managers in Swing. Write a Java program using GridLayout or BorderLayout to arrange components.

8. Write and explain a program that demonstrates the use of MouseListener and KeyListener interfaces in Java Swing.

KTU SPECIAL
CONNECT WITH US

WHATSAPP GROUP

FOLLOW US NOW

TELEGRAM CHANNEL

SUBSCRIBE NOW

www.ktuspecial.in

SUBSCRIBE

FOLLOW US

Visit Website