JAVA UNIT 6

1. Explain Swing along with features of Swing

- **Swing** is a part of Java used to create **window-based GUI applications**. It is built on top of AWT and is written completely in Java, which makes it platform independent.
- Swing provides a **rich set of GUI components** like JButton, JLabel, JTextField, JTable, JList, JMenu, etc., which help in building professional user interfaces.

Features of Swing:

- Swing components are **lightweight**, meaning they do not depend on native operating system components and are more flexible.
- It supports **pluggable look and feel**, so the appearance of the application can be changed without changing code.
- Swing follows **MVC architecture**, which separates data, user interface, and control logic.
- It supports **event-driven programming**, making applications interactive and responsive.
- Swing uses **double buffering**, which reduces screen flickering and gives smooth display.

2. Differentiate between AWT and Swing

- **AWT components are platform dependent**, meaning they rely on the operating system for display. Swing components are **platform independent** because they are written entirely in Java.
- AWT components are **heavyweight**, as they use native OS resources. Swing components are **lightweight** and do not depend on OS components.
- AWT does **not support pluggable look and feel**, so its appearance cannot be changed easily. Swing **supports pluggable look and feel**, allowing flexible UI design.
- AWT provides **limited GUI components**, mainly basic controls. Swing provides **advanced components** like JTable, JTree, JScrollPane, and JTabbedPane.
- AWT **does not follow MVC architecture**, while Swing **follows MVC**, which improves design and maintainability.

3. What is JFrame? Explain its constructors and methods with example and its role as top-level container

- **JFrame** is a Swing class used to create the **main window** of an application. It acts as a container where all other Swing components are added.
- JFrame is a **top-level container**, meaning it is not placed inside another container and holds components like buttons, labels, and text fields.

Constructors:

- JFrame() creates a frame with no title and is invisible by default.
- JFrame(String title) creates a frame with a specified title.

Common Methods:

- setTitle() sets the title of the frame.
- setSize() sets the width and height.
- setDefaultCloseOperation() defines action on closing the window.
- setVisible(true) makes the frame visible.

Example: (write inside class and main method)

```
JFrame f = new JFrame("My Frame");

f.setSize(300,200);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

f.setVisible(true);
```

- Here, JFrame works as a **top-level container** holding all GUI components.

4. Explain Swing component hierarchy along with classes of Swing

- Swing follows a **class hierarchy** that defines how components are organized and inherited.
- At the top of the hierarchy is the **Object class**, followed by **Component**, which provides basic GUI features.
- The **Container** class extends Component and is used to hold other components.
- **JComponent** is the base class for most Swing components like JButton, JLabel, JTextField, and JCheckBox.
- **Top-level containers** such as JFrame and JDialog do not inherit JComponent and act as main windows.
- Common Swing classes include JButton, JLabel, JTextArea, JTextField, JCheckBox, JRadioButton, JComboBox, JTable, and JMenu.
- This hierarchy helps in **reusability, consistency, and organized GUI design**.

5. Explain JLabel and JTextField with example program

- **JLabel** is a Swing component used to display **read-only text** or images. It does not accept user input and is mainly used for descriptions or messages.
- JLabel is a **passive component**, meaning the user cannot interact with it directly.
- **JTextField** is used to **accept single-line input** from the user. It allows editing and generates an event when the Enter key is pressed.
- JTextField is commonly used for **input and output of text**.

Example Program: (write inside class and main method)

```
JFrame f = new JFrame("Label and TextField");

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

JTextField t = new JTextField(15);


f.setLayout(new FlowLayout());

f.add(l);

f.add(t);

f.setSize(300,200);

f.setVisible(true);
```

- This program shows how JLabel displays text and JTextField accepts user input in a Swing application.

6. Explain JButton with ActionListener with the help of an example program

- **JButton** is a Swing component used to create a **push button** in a GUI application.
- When the user clicks a JButton, it generates an **ActionEvent**, which is used to perform some action.
- To handle this event, we use **ActionListener**, which is an interface present in the java.awt.event package.
- ActionListener has only **one method**, actionPerformed(ActionEvent e), which is executed when the button is clicked.
- The button must be **registered** with the listener using the addActionListener() method.

Example Program: (write inside class and main method)

```
JFrame f = new JFrame("Button Example");

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


b.addActionListener(e -> {

   System.out.println("Button is clicked");

});


f.add(b);

f.setSize(200,200);

f.setVisible(true);
```

- In this example, when the button is clicked, the message is printed.
- JButton makes the application **interactive** by responding to user actions.

7. Explain JCheckBox, JRadioButton, and JComboBox with example program

- **JCheckBox** is used to select or unselect an option. It has two states: **selected** or **unselected**.
- Multiple JCheckBoxes can be selected at the same time. It is commonly used when multiple choices are allowed.
- **JRadioButton** is used to select **only one option** from multiple choices.
- JRadioButtons are grouped using the **ButtonGroup** class so that only one can be selected.
- **JComboBox** is a drop-down list that allows the user to select **one item** from a list. It can be editable or read-only.

Example Program: (write inside class and main method)

```
JFrame f = new JFrame("Components");

JCheckBox c1 = new JCheckBox("Java");

JRadioButton r1 = new JRadioButton("Male");

JRadioButton r2 = new JRadioButton("Female");

ButtonGroup bg = new ButtonGroup();

bg.add(r1); bg.add(r2);

JComboBox cb = new JComboBox(new String[]{"A","B","C"});
```

```
f.setLayout(new FlowLayout());

f.add(c1); f.add(r1); f.add(r2); f.add(cb);

f.setSize(300,200);

f.setVisible(true);
```

8. What is the purpose of Layout Manager? Explain FlowLayout with example

- A layout manager is used in Java Swing to control how components are arranged inside a container such as JFrame or JPanel. It decides the position and size of components automatically.
- The main purpose of a layout manager is to make GUI design flexible and responsive, so components adjust properly when the window size changes.
- Layout managers reduce manual work because the programmer does not need to calculate exact coordinates for each component.
- FlowLayout is the simplest layout manager in Swing. It arranges components from left to right in a row, similar to words in a paragraph.
- When there is no space left in the current row, FlowLayout automatically moves the components to the next row.
- By default, components are center aligned and have a horizontal and vertical gap of 5 pixels between them.
- FlowLayout also supports alignment options like LEFT, RIGHT, and CENTER.

Example:

```
JFrame f = new JFrame("FlowLayout Example");

f.setLayout(new FlowLayout());

f.add(new JButton("Button 1"));

f.add(new JButton("Button 2"));

f.add(new JButton("Button 3"));

f.setSize(300,200);

f.setVisible(true);
```

9. Explain GridLayout with the help of an example program

- **GridLayout** is a layout manager in Java Swing that arranges components in the form of **rows and columns**. Each component is placed inside a rectangular cell, and all cells are of **equal size**.
- The main purpose of GridLayout is to provide a **structured and uniform layout**, where every component gets the same amount of space.
- Components are added to the container **row by row from left to right**. Once a row is filled, the layout manager moves to the next row automatically.
- GridLayout is very useful in applications like **calculators, keypads, and forms**, where buttons or fields need to be arranged evenly.
- The constructor GridLayout(int rows, int columns) is used to specify the number of rows and columns. Another constructor allows setting **horizontal and vertical gaps** between components.

Example Program:

```
JFrame f = new JFrame("GridLayout Example");

f.setLayout(new GridLayout(2, 2));


f.add(new JButton("Button 1"));

f.add(new JButton("Button 2"));

f.add(new JButton("Button 3"));

f.add(new JButton("Button 4"));


f.setSize(300, 200);

f.setVisible(true);
```

In this example, the JFrame uses a **2×2 GridLayout**, and all four buttons are displayed with equal size.

GridLayout automatically adjusts component sizes when the window is resized, making the layout simple and effective.

10. Differentiate between FlowLayout and GridLayout

- FlowLayout arranges components from left to right in a row, similar to words in a sentence. GridLayout arranges components in a fixed number of rows and columns.
- In FlowLayout, components keep their original (preferred) size, whereas in GridLayout, all components are resized equally to fit the grid cells.
- When there is no space in a row, FlowLayout automatically moves components to the next row. GridLayout strictly fills the grid row by row.
- FlowLayout allows alignment options such as LEFT, RIGHT, and CENTER. GridLayout does not provide alignment options.
- FlowLayout is commonly used for simple interfaces like button panels. GridLayout is used where uniform and structured layout is required.
- FlowLayout is more flexible, while GridLayout is more organized and systematic in appearance.
- FlowLayout is suitable when the number of components may change, while GridLayout is better when the number of components is fixed.
- FlowLayout adjusts components based on available space, whereas GridLayout divides the container into equal-sized cells.

11. Explain Event Delegation Model

- The Event Delegation Model is a Java mechanism used to handle events generated by GUI components. An event occurs when there is a change in the state of a component.
- It consists of two main parts: Source and Listener. The source generates the event, and the listener handles it.
- The source must be registered with the listener using methods like addActionListener(). Without registration, the event will not be handled.
- When an event occurs, the source sends a notification to the registered listener.
- The listener contains code that is executed in response to the event, such as clicking a button.
- This model helps in separating GUI design from event-handling logic, making programs easier to understand and maintain.
- It improves code readability, as event-handling code is written separately from UI code.
- The Event Delegation Model supports multiple listeners for a single source, making event handling more flexible.

12. Explain ActionEvent and ActionListener with the help of an example program

- **ActionEvent** is an event that occurs when a user performs an action on a component such as **clicking a button**, pressing the Enter key in a text field, or selecting a menu item.
- In Swing, components like **JButton** and **JTextField** generate ActionEvent when an action is performed by the user.
- ActionEvent contains information about the event, such as the **source of the event** and the **action command**.
- **ActionListener** is an interface used to handle ActionEvent. It is present in the java.awt.event package.
- ActionListener has **only one method**, actionPerformed(ActionEvent e), which is executed automatically when the event occurs.
- To handle an event, the component must be **registered** with the ActionListener using the addActionListener() method.

Example Program:

```
import javax.swing.*;

import java.awt.event.*;

public class ActionEventExample {

   public static void main(String[] args) {

      JFrame f = new JFrame("ActionEvent Example");

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

      b.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {

            System.out.println("Button Clicked");

         }

});

      f.add(b);

      // Set frame properties

      f.setSize(300, 200);

      f.setLayout(new java.awt.FlowLayout());

      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      f.setVisible(true);

   } }
```

13. Explain Menu in Swing. WAP to implement Menu in Java Swing

Menu in Swing (Explanation)

- A **menu** in Swing is used to provide a list of commands or options to the user in an organized way.
- Swing provides three main classes to create menus: **JMenuBar**, **JMenu**, and **JMenuItem**.
- **JMenuBar** is displayed at the top of a JFrame and holds one or more menus.
- **JMenu** represents menu headings such as *File*, *Edit*, or *Help*.
- **JMenuItem** represents individual options inside a menu, like *Open*, *Save*, or *Exit*.
- Menus improve **usability** by grouping related actions and making applications user-friendly.
- Menu items can also generate **ActionEvent**, which can be handled using ActionListener.

Program: Menu using Swing

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample {
   public static void main(String[] args) {

      JFrame f = new JFrame("Menu Example");
      JMenuBar mb = new JMenuBar();
      JMenu menu = new JMenu("File");
      JMenuItem i1 = new JMenuItem("New");
      JMenuItem i2 = new JMenuItem("Exit");
      menu.add(i1);
      menu.add(i2);
      mb.add(menu);
      f.setJMenuBar(mb);
      f.setSize(300, 200);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
   }
}
```

14. WAP to create a Login Form using Swing (with ActionListener)

Login Form using Swing – Explanation

- A **login form** is used to accept **username and password** from the user.
- Swing components such as **JLabel, JTextField, JPasswordField, and JButton** are used to design the form.
- JLabel displays text, while JTextField and JPasswordField are used for user input.
- JButton is used to **submit login details**.
- **ActionListener** is used to handle the button click event.
- When the Login button is clicked, an **ActionEvent** is generated and processed using actionPerformed().

Program: Login Form using Swing with ActionListener

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

public class LoginForm {
   public static void main(String[] args) {

      JFrame f = new JFrame("Login Form");
      JLabel l1 = new JLabel("Username:");
      JLabel l2 = new JLabel("Password:");
      JTextField t1 = new JTextField(15);
      JPasswordField t2 = new JPasswordField(15);
      JButton b = new JButton("Login");

      // Add ActionListener to button

      b.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String user = t1.getText();
            String pass = new String(t2.getPassword());
```

```
        if (user.equals("admin") && pass.equals("1234")) {
          JOptionPane.showMessageDialog(f, "Login Successful");
        } else {
          JOptionPane.showMessageDialog(f, "Invalid Login");
        }
      }
    });
    f.setLayout(new FlowLayout());
    f.add(l1);
    f.add(t1);
    f.add(l2);
    f.add(t2);
    f.add(b);

    f.setSize(300, 200);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
  }}
```

15. Create a Simple Calculator using Swing (Addition and Subtraction)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SimpleCalculator {
  public static void main(String[] args) {
    JFrame f = new JFrame("Simple Calculator");
    JLabel l1 = new JLabel("Number 1:");
    JLabel l2 = new JLabel("Number 2:");
    JLabel l3 = new JLabel("Result:");
```

```
  JTextField t1 = new JTextField(10);
  JTextField t2 = new JTextField(10);
  JTextField t3 = new JTextField(10);
  t3.setEditable(false);
  JButton b1 = new JButton("Add");
  JButton b2 = new JButton("Subtract");

  b1.addActionListener(new ActionListener() {       // ActionListener for Add button
    public void actionPerformed(ActionEvent e) {
      int a = Integer.parseInt(t1.getText());
      int b = Integer.parseInt(t2.getText());
      t3.setText(String.valueOf(a + b));
    }
  });
  b2.addActionListener(new ActionListener() {            // ActionListener for Subtract button
    public void actionPerformed(ActionEvent e) {
      int a = Integer.parseInt(t1.getText());
      int b = Integer.parseInt(t2.getText());
      t3.setText(String.valueOf(a - b));
    }
  });
  f.setLayout(new FlowLayout());
  f.add(l1); f.add(t1);
  f.add(l2); f.add(t2);
  f.add(b1); f.add(b2);
  f.add(l3); f.add(t3);

  f.setSize(300, 250);
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  f.setVisible(true);
}}
```

16. Explain ActionEvent and ActionListener with the help of an example program

Explanation

- **ActionEvent** is an event that occurs when a user performs an action such as **clicking a button**, pressing the Enter key in a text field, or selecting a menu item.
- Components like **JButton, JTextField, and JMenuItem** generate ActionEvent.
- ActionEvent carries information such as the **source of the event** and the **action command**.
- **ActionListener** is an interface used to handle ActionEvent.
- It is present in the java.awt.event package and contains only **one method**, actionPerformed(ActionEvent e).
- The component must be **registered with the listener** using addActionListener() method.
- When the event occurs, the actionPerformed() method is automatically executed.

Example Program

```
import javax.swing.*;

import java.awt.event.*;

public class ActionEventDemo {

  public static void main(String[] args) {

    JFrame f = new JFrame("ActionEvent Example");

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

    b.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {

        System.out.println("Button Clicked");

      }

    });


    f.add(b);

    f.setSize(300,200);

    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    f.setVisible(true);

  }

}
```

17. Explain JLabel, JTextField, JButton and their methods. WAP to demonstrate them

Explanation

JLabel

- JLabel is used to display **read-only text** in a Swing application.
- It is a **passive component**, meaning the user cannot modify it.
- Common methods:
    - setText(String s) – sets label text
    - getText() – returns label text

JTextField

- JTextField is used to accept **single-line input** from the user.
- It can be used for both **input and output**.
- Common methods:
    - setText(String s) – sets text
    - getText() – gets user input
    - addActionListener() – handles Enter key event

JButton

- JButton is used to create a **clickable button**.
- When clicked, it generates an **ActionEvent**.
- Common methods:
    - setText(String s)
    - getText()
    - addActionListener()

Program: Demonstration of JLabel, JTextField, JButton

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

```
public class ComponentDemo {
   public static void main(String[] args) {

      JFrame f = new JFrame("Swing Components");

      JLabel l = new JLabel("Enter Name:");
      JTextField t = new JTextField(15);
      JButton b = new JButton("Submit");

      b.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String name = t.getText();
            JOptionPane.showMessageDialog(f, "Hello " + name);
         }
      });

      f.setLayout(new FlowLayout());
      f.add(l);
      f.add(t);
      f.add(b);

      f.setSize(300,200);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
   }
}
```

18. Explain Combo Box, Check Box, Radio Button and their methods. WAP to demonstrate them

Explanation

JCheckBox

- JCheckBox is used to **select or unselect an option**.
- It has two states: **selected (true)** and **unselected (false)**.
- Multiple checkboxes can be selected at the same time.
- Commonly used when **multiple choices are allowed**.

Important Methods:

- setText(String s) – sets checkbox text
- isSelected() – checks whether it is selected
- addItemListener() – handles state change

JRadioButton

- JRadioButton is used to select **only one option** from multiple choices.
- Radio buttons are grouped using **ButtonGroup**.
- Widely used in **quiz and gender selection forms**.

Important Methods:

- setText(String s)
- isSelected()
- addActionListener()

JComboBox

- JComboBox is a **drop-down list** used to select one item at a time.
- It can be **editable or non-editable**.
- Saves space compared to radio buttons.

Important Methods:

- addItem()
- getSelectedItem()
- removeItem()

Program: JCheckBox, JRadioButton, JComboBox

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

public class ComboCheckRadioDemo {
   public static void main(String[] args) {

      JFrame f = new JFrame("Swing Components");

      JCheckBox c1 = new JCheckBox("Java");
      JCheckBox c2 = new JCheckBox("Python");

      JRadioButton r1 = new JRadioButton("Male");
      JRadioButton r2 = new JRadioButton("Female");

      ButtonGroup bg = new ButtonGroup();
      bg.add(r1);
      bg.add(r2);

      JComboBox cb = new JComboBox(new String[]{"FY", "SY", "TY"});
      f.setLayout(new FlowLayout());
      f.add(c1); f.add(c2);
      f.add(r1); f.add(r2);
      f.add(cb);

      f.setSize(300, 250);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
   }
}
```

19. Explain Menu creation in Swing. WAP to implement menu with ActionListener

Explanation

- A **menu** in Swing provides a list of commands in an organized way.
- Menu system is created using **three main classes**:
    - **JMenuBar** – holds menus
    - **JMenu** – menu heading (File, Edit)
    - **JMenuItem** – selectable menu option
- Menus improve **usability and navigation** of applications.
- Menu items generate **ActionEvent** and can be handled using **ActionListener**.

Program: Menu using Swing with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class MenuDemo {
  public static void main(String[] args) {

    JFrame f = new JFrame("Menu Example");

    JMenuBar mb = new JMenuBar();
    JMenu menu = new JMenu("File");

    JMenuItem i1 = new JMenuItem("New");
    JMenuItem i2 = new JMenuItem("Exit");

    i1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(f, "New File Created");
      }
    });
```

```
        i2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });

        menu.add(i1);
        menu.add(i2);
        mb.add(menu);

        f.setJMenuBar(mb);
        f.setSize(300, 200);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```