CHAPTER NO: 06
AWT and SWING Controls { 12 MARKS}

6.1 Abstract Window toolkit, AWT classes

6.2 Event handling, Delegation event model, Event model, Event classes, source of events, event listener interface,

6.3 windows fundamentals, creating a frame window, working with frame window, working with graphics

6.4 Introduction to AWT controls, inserting user interfaces like buttons, checkbox, list, scrollbar, text field and text areas, layout managers.

6.5 Introduction to SWING GUI Controls

6.1 Abstract Window toolkit, AWT classes

- **AWT (Abstract Window Toolkit)** is a set of APIs provided by **Java** for creating **Graphical User Interfaces (GUIs)**.
- It is part of the **Java Foundation Classes (JFC)** and provides the basic building blocks for GUI development in Java.

What is AWT?

- AWT is a **platform-dependent** GUI toolkit.
- It uses the **native GUI components** of the operating system (called **peers**).
- AWT provides classes for creating:

➢ Windows

➢ Buttons

➢ Text boxes

➢ Event handling

➢ Layout managers

➢ Menus, etc.

Hierarchy of AWT Classes

```
java.lang.Object
    ↳ java.awt.Component
            ↳ java.awt.Container
                    ↳ java.awt.Window
                            ↳ java.awt.Frame
                            ↳ java.awt.Dialog
            ↳ java.awt.Label
            ↳ java.awt.Button
            ↳ java.awt.TextComponent
                    ↳ java.awt.TextField
                    ↳ java.awt.TextArea
```

Important AWT Classes & Their Use:

Class	Description
`Component`	Abstract superclass for all AWT GUI components.
`Container`	A component that can contain other components (e.g., panels, frames).
`Button`	Represents a push button.
`Label`	Displays a short string or text label.
`TextField`	Allows the user to enter a single line of text.
`TextArea`	Allows the user to enter multiple lines of text.
`Checkbox`	A checkbox that can be selected or deselected.
`Choice`	A drop-down list of choices.

Class	Description
`List`	Allows selection from a list of items.
`Frame`	A top-level window with a title.
`Panel`	A generic container for components.
`Canvas`	A blank area for drawing.
`ScrollBar`	A scroll bar component.
`Menu`, `MenuBar`, `MenuItem`	Classes for creating menus.

```java
import java.awt.*;

public class AWTExample {
    public static void main(String[] args) {
        Frame f = new Frame("AWT Example");

        Label label = new Label("Enter your name:");
        label.setBounds(50, 50, 150, 20);

        TextField tf = new TextField();
        tf.setBounds(50, 80, 150, 20);

        Button b = new Button("Submit");
        b.setBounds(50, 110, 60, 30);

        f.add(label);
        f.add(tf);
        f.add(b);

        f.setSize(300, 200);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

This line imports the **Abstract Window Toolkit (AWT)** package, which provides classes for creating GUI (Graphical User Interface) components like windows, buttons, labels, text fields, etc.

This creates a **window frame** with the title "AWT Example".

This creates a **label** component with the text "Enter your name:" to be shown on the frame.

- This sets the **position and size** of the label:
  - `x = 50`, `y = 50` (position from top-left corner)
  - `width = 150`, `height = 20`

Sets the size of the frame to 300 pixels width and 200 pixels height.

- Sets the **size** of the frame to `300` pixels width and `200` pixels height.

- Disables the default layout manager. This allows you to use **absolute positioning** (using `setBounds()`).

Makes the frame **visible on the screen**.

🔷 Pros and Cons of AWT

✅ Advantages:

- Simple and easy to learn.
- Lightweight for small applications.
- Direct access to native OS GUI components.

❌ Disadvantages:

- Platform-dependent (not 100% portable).
- Limited look-and-feel customization.
- Outdated compared to Swing or JavaFX.

- **Heavyweight components** are those that are **tightly bound** to the underlying native system's windowing system.
- They rely on native system calls to render and manage their appearance and behavior.
- This means that they have a **physical representation** in the native windowing system.

Heavyweight Example (AWT): Button in AWT is a heavyweight component because it is implemented as a native button on the OS, so it relies on the OS windowing system.

Lightweight components are **not dependent** on the native windowing system and instead are **rendered using Java's graphics system**. They are managed purely by the Java runtime environment.

Lightweight Example (Swing): Swing components like JButton (from the Swing library, not AWT) are lightweight because they are drawn entirely by Java using its own graphics system, not the OS.

Feature	Heavyweight Components	Lightweight Components
Rendering	Rendered by the native OS windowing system	Rendered by Java's own graphics system
Example (AWT)	`Button`, `TextField`, `Label`	Not applicable in AWT (Swing has lightweight components)
Example (Swing)	Not applicable	`JButton`, `JLabel`, `JTextField`
System Dependence	Platform-dependent (OS-specific behavior)	Platform-independent (same look on all OS)
Performance	Can be slower due to OS dependency	Generally faster as no OS dependency
Look and Feel	Dependent on the OS theme/style	Consistent across platforms (customizable)

Swing vs. AWT:

- **AWT** components are **heavyweight**.
- **Swing** components are mostly **lightweight** and provide more flexibility and better performance.

To summarize:

- **Heavyweight**: Dependent on OS windowing system, slower performance, inconsistent across platforms.
- **Lightweight**: Drawn by Java itself, more flexible, better performance, consistent across platforms.

6.2 Event handling, Delegation event model, Event model, Event classes, source of events, event listener interface,

Event Handling in Java (AWT):

Event handling is a mechanism that controls how an application responds to events like mouse clicks, key presses, button presses, etc.

What is Event Handling?

Event handling in Java allows GUI components (like buttons) to respond to user actions.
For example, when you click a **Submit** button, an **event** is generated, and **event handling code** processes it.

🔷 Key Concepts in Event Handling

1. **Event Source** – The GUI component that generates an event (e.g., a Button).
2. **Event Object** – Contains information about the event (e.g., `ActionEvent`).
3. **Event Listener** – An interface that listens for events and defines how to handle them.

🔄 Delegation Event Model

Java uses the **Delegation Event Model** for event handling. In this model:

- A **source** (e.g., button) generates an **event** (e.g., button clicked).
- The event is sent (delegated) to an **event listener**.
- The listener **handles** the event using a method defined in the listener interface.

📝 Think of it as:

Source → Event → Listener → Handler Method

⚙️ Event Model Components

1. Event Classes (`java.awt.event`)

These classes represent different types of events:

Event Class	Description
`ActionEvent`	Generated when a button is clicked
`KeyEvent`	When a key is pressed or released
`MouseEvent`	Mouse click, press, release, etc.
`WindowEvent`	Window opened, closed, activated, etc.
`ItemEvent`	When an item is selected (like checkbox)

2. Source of Events

Any GUI component (like `Button`, `TextField`, etc.) can be a **source** of events.

Example:

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

When clicked, this `Button` generates an `ActionEvent`.

3. Event Listener Interfaces

These interfaces must be implemented to handle specific event types:

Listener Interface	Method(s)	Used For
`ActionListener`	`actionPerformed(ActionEvent e)`	Buttons, menu items
`KeyListener`	`keyPressed()`, `keyReleased()`, `keyTyped()`	Keyboard input
`MouseListener`	`mouseClicked()`, etc.	Mouse events
`WindowListener`	`windowClosing()`, etc.	Window events
`ItemListener`	`itemStateChanged()`	Checkboxes, lists

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

public class EventExample {
    public static void main(String[] args) {
        Frame f = new Frame("Event Example");

        Button b = new Button("Click Me");
        b.setBounds(100, 100, 80, 30);

        // Registering event listener
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button clicked!");
            }
        });

        f.add(b);
        f.setSize(300, 200);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

- "When someone clicks the button, run this code."
- `addActionListener(...)` → attaches an event handler to the button.
- Inside, we write what should happen: here it **prints "Button clicked!"** in the console.

"When the button is pressed, run this code."

Here, the code simply prints "**Button clicked!**" in the console.

Component	Description
Event Source	Generates event (e.g., `Button`)
Event Object	Carries event data (e.g., `ActionEvent`)
Listener	Handles the event (e.g., `ActionListener`)

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

public class LoginExample {
    public static void main(String[] args) {
        Frame loginFrame = new Frame("Login Window");
        Button loginButton = new Button("Login");

        loginButton.setBounds(100, 100, 80, 30);
        loginFrame.add(loginButton);

        loginFrame.setSize(300, 200);
        loginFrame.setLayout(null);
        loginFrame.setVisible(true);

        // Action listener for login button
        loginButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Hide login window
                loginFrame.setVisible(false);
```

```
            // Show new welcome window
            Frame welcomeFrame = new Frame("Welcome Window");
            Label label = new Label("Login Successful!");
            label.setBounds(80, 100, 150, 30);
            welcomeFrame.add(label);

            welcomeFrame.setSize(300, 200);
            welcomeFrame.setLayout(null);
            welcomeFrame.setVisible(true);
        }
    });
  }
}
```

What happens here:

1. First → **Login Window** opens with a "Login" button.
2. When you click **Login**, the line

   ```java
   loginFrame.setVisible(false);
   ```

   hides the **Login Window**.
3. At the same time, a new **Welcome Window** is shown with a message "Login Successful!".

6.3 windows fundamentals, creating a frame window, working with frame window, working with graphics

Windows Fundamentals

In Java AWT, a **window** is a visual container that displays UI components to the user. The most common window is a **Frame**.

What is a Frame?

- A **Frame** is a top-level window with a title bar, borders, and buttons (minimize, maximize, close).
- It can contain other UI components like buttons, text fields, panels, etc.
- It's used as the main window of a GUI application.

Creating a Frame Window

You create a Frame by instantiating the Frame class.

Example:

```
import java.awt.*;

public class FrameExample {
    public static void main(String[] args) {
        Frame f = new Frame("My First Frame");
        f.setSize(400, 300);            // Set window size (width x height)
        f.setVisible(true);             // Make the frame visible
    }
}
```

Working with Frame Window

You can customize the frame by:

- Setting the layout (f.setLayout(...))
- Adding components (f.add(...))
- Handling window events (like closing)
- Setting background and foreground colors
- Positioning the frame on the screen

Handling Window Close Event:

- By default, closing a Frame window does not stop the program. To close the program on window close:

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

public class FrameCloseExample {
    public static void main(String[] args) {
        Frame f = new Frame("Close Window Example");
        f.setSize(300, 200);
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);   // Exit the program
            }
        });
    }
}
```

Working with Graphics

The Graphics class is used to draw shapes, text, and images inside a GUI component.

Basic concepts:

- You override the `paint(Graphics g)` method of a component (like `Frame` or `Canvas`).
- The `Graphics` object `g` provides drawing methods.

Example: Drawing inside a Frame

```
import java.awt.*;

public class GraphicsExample extends Frame {

    public void paint(Graphics g) {
        g.setColor(Color.RED);
        g.drawString("Hello, Graphics!", 50, 100);

        g.setColor(Color.BLUE);
        g.drawRect(70, 120, 150, 100);  // Draw rectangle

        g.setColor(Color.GREEN);
        g.fillOval(100, 150, 100, 50);  // Draw filled oval
    }

    public static void main(String[] args) {
        GraphicsExample f = new GraphicsExample();
        f.setSize(400, 400);
        f.setVisible(true);
    }
}
```

Graphics methods commonly used:

Method	Description
`drawString(String s, int x, int y)`	Draw text at (x,y)
`drawRect(int x, int y, int w, int h)`	Draw rectangle
`fillRect(int x, int y, int w, int h)`	Draw filled rectangle
`drawOval(int x, int y, int w, int h)`	Draw oval
`fillOval(int x, int y, int w, int h)`	Draw filled oval
`setColor(Color c)`	Set drawing color

Topic	Description
Frame	Top-level window with title and borders
Creating Frame	Using `new Frame()` and setting size, visibility
Window Events	Handle events like window closing
Graphics	Drawing shapes and text inside components by overriding `paint()`

6.4 Introduction to AWT controls, inserting user interfaces like buttons, checkbox, list, scrollbar, text field and text areas, layout managers.

Introduction to AWT Controls

AWT Controls are the user interface components you use to build GUIs in Java. They allow users to interact with your program.

Common AWT Controls:

- Button
- Checkbox
- List
- Scrollbar
- TextField
- TextArea

🔷 1. Button

- A clickable button.
- Used to perform actions when clicked.

```java
Button btn = new Button("Click Me");
```

🔷 2. Checkbox

- A box that can be checked or unchecked.
- Used to select/deselect options.

```java
Checkbox cb = new Checkbox("Accept Terms");
```

🔷 3. List

- Displays a list of items.
- User can select one or multiple items.

```java
List list = new List(4); // List with 4 visible rows
list.add("Apple");
list.add("Banana");
list.add("Cherry");
```

🔷 4. Scrollbar

- A graphical control for scrolling content vertically or horizontally.

```java
Scrollbar sb = new Scrollbar();
```

🔷 5. TextField

- Single line input box for user text.

```java
TextField tf = new TextField(20);  // 20 columns wide
```

🔷 6. TextArea

- Multi-line text input area.

```java
TextArea ta = new TextArea(5, 20); // 5 rows, 20 columns
```

Adding Controls to a Frame or Container

Example of adding a button and checkbox to a frame:

```
Frame f = new Frame("AWT Controls");
Button btn = new Button("Submit");
Checkbox cb = new Checkbox("Subscribe");

f.add(btn);
f.add(cb);

f.setSize(300, 200);
f.setLayout(new FlowLayout());  // Using FlowLayout
f.setVisible(true);
```

Layout Managers

- Layout managers control how components are arranged in a container. They handle resizing and positioning automatically.

Common Layout Managers in AWT:

Layout Manager	Description
`FlowLayout`	Places components in a row, left to right, wrapping as needed.
`BorderLayout`	Divides container into 5 areas: North, South, East, West, Center.
`GridLayout`	Arranges components in a grid (rows x columns).
`CardLayout`	Stacks components like cards, one visible at a time.

Examples:

FlowLayout:

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

BorderLayout:

```java
f.setLayout(new BorderLayout());
f.add(btn, BorderLayout.NORTH);
f.add(cb, BorderLayout.SOUTH);
```

GridLayout:

```java
f.setLayout(new GridLayout(2, 2));  // 2 rows, 2 columns
```

✅ Summary:

Concept	Explanation
AWT Controls	Buttons, checkboxes, lists, scrollbars, text inputs
User Interface	Components allow user input and interaction
Layout Managers	Automatically arrange components in containers

6.5 Introduction to SWING GUI Controls

- What is Swing?
- **Swing** is a part of Java Foundation Classes (JFC) that provides a **more advanced, flexible, and platform-independent** set of GUI components than AWT.
- Unlike AWT, Swing components are **lightweight**, meaning they are written entirely in Java and do **not rely on native OS peers**.
- Swing provides a **rich set of GUI controls** with better look-and-feel and customization.

🔷 Why Use Swing Over AWT?

Feature	AWT	Swing
Components	Heavyweight (depends on OS)	Lightweight (pure Java)
Look and Feel	OS-dependent	Pluggable, customizable
Rich Components	Basic controls	Advanced controls (tables, trees, sliders, etc.)
Customization	Limited	Highly customizable

🔷 Common Swing GUI Controls

Swing Component	Description	AWT Equivalent
`JFrame`	Top-level window	`Frame`
`JButton`	Button	`Button`
`JLabel`	Display text or image	`Label`
`JTextField`	Single line text input	`TextField`
`JTextArea`	Multi-line text input	`TextArea`
`JCheckBox`	Checkbox	`Checkbox`
`JRadioButton`	Radio button	No direct equivalent
`JComboBox`	Drop-down list	`Choice`
`JList`	List with multiple selection	`List`
`JPanel`	Container for grouping components	`Panel`
`JScrollPane`	Provides scrollbars	`Scrollbar` + container

Basic Swing Program Example

```
import javax.swing.*;

public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Example");

        JButton button = new JButton("Click Me");
        frame.add(button);

        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close on exit
        frame.setVisible(true);
    }
}
```

🔷 Key Differences from AWT:

- Swing components start with a `J` prefix (e.g., `JFrame`, `JButton`).
- Swing uses the **MVC (Model-View-Controller)** architecture.
- You need to import `javax.swing.*` for Swing controls.
- Swing components support **pluggable look-and-feel**, meaning you can change the GUI appearance dynamically.

🔷 Summary:

Feature	Swing
GUI Toolkit	Java's advanced, lightweight GUI toolkit
Components	Rich, flexible GUI components
Platform Dependence	Platform independent (pure Java)
Look and Feel	Highly customizable and consistent