جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Intelligent Medical Systems Department

Subject: Graphical User Interface (GUI) in Java

Class:3rd

Lecturer: Asst.Lect Mustafa Ameer Awadh

Lecture: (4)

Introduction

A **Graphical User Interface (GUI)** allows users to interact with software through visual elements like buttons, windows, and text fields, making applications more user-friendly and intuitive. Unlike command-line interfaces, GUIs provide a visual approach to perform actions. In Java, GUIs are built using libraries such as **Swing** and **AWT (Abstract Window Toolkit)**, which offer various components and tools to create windows, dialogs, and interactive elements. This lecture will introduce the concepts of preparing containers and components, managing component layouts, and handling events to create fully interactive and responsive Java applications.

1.Preparing Containers and Components

1.1 Containers:

A **container** is a component in Java that holds other GUI components. It serves as the basic structure or window where all visual elements (buttons, text fields, etc.) are placed.

Types of Containers:

- **JFrame**: A top-level window that represents the main application window.
- **JPanel**: A generic container that holds and organizes components, often used inside a JFrame.
- **JDialog**: A small pop-up window that appears temporarily to gather user input or display messages.

Example: Creating a Simple JFrame:

```
import javax.swing.*;

public class SimpleFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My First GUI");  // Create a JFrame window
        frame.setSize(400, 300);                    // Set the window size
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  // Close app on e

        JButton button = new JButton("Click Me!");  // Create a button
        frame.add(button);  // Add button to the frame

        frame.setVisible(true);  // Make the frame visible
    }
}
```

1.2 Components:

Components are the individual elements that form the interface within a container. These include buttons, labels, text fields, checkboxes, etc.

Common Components:

- **JButton**: A clickable button.

- **JLabel**: Used to display a line of text or an image.
- **JTextField**: Allows the user to enter a single line of text.
- **JCheckBox**: A box that can be checked or unchecked.
- **JComboBox**: A drop-down list from which users can choose an option.

2. Managing Component Layout

In Java, **layout managers** are responsible for organizing components inside containers, determining where each component is placed and how much space it occupies. Java provides different layout managers for managing the arrangement of components within a container.

2.1 Common Layout Managers:

1. **FlowLayout**: Arranges components in a row, one after another. If there's no space left, components wrap to the next line (like words in a paragraph).

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

2. **BorderLayout**: Divides the container into five regions:

NORTH, SOUTH, EAST, WEST, and CENTER. Each region can hold one component.

```
frame.setLayout(new BorderLayout());
frame.add(button, BorderLayout.NORTH);  // Adds button to the top
```

3. **GridLayout**: Places components in a grid of rows and columns. All components get equal space.

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

4. **BoxLayout**: Aligns components either vertically (BoxLayout.Y_AXIS) or horizontally (BoxLayout.X_AXIS).

```
frame.setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
```

2.2 Nested Layouts:

You can use different layouts inside different containers to create complex UI designs. For instance, a JPanel can have its own layout, nested inside a JFrame that uses a different layout.

Example of Using Multiple Layouts:

```
JFrame frame = new JFrame();
frame.setLayout(new BorderLayout());

JPanel panel = new JPanel(new FlowLayout());  // Inner panel with FlowLayout
panel.add(new JButton("Button 1"));
panel.add(new JButton("Button 2"));

frame.add(panel, BorderLayout.SOUTH);  // Adds the panel to the south region
```

3. Event Handling

Event handling is essential for making the GUI interactive. Events occur when users interact with components, such as clicking a button, typing in a text field, or selecting an item from a combo box. In Java, **event listeners** are used to respond to these events.

3.1 Common Event Listeners:

- **ActionListener**: Listens for actions like button clicks or pressing the "Enter" key.

- **MouseListener**: Detects mouse events, such as clicks and hovering.
- **KeyListener**: Listens for keyboard input.
- **WindowListener**: Responds to window events like opening, closing, or minimizing.

3.2 Handling Button Clicks:

To handle button click events, you need to attach an ActionListener to the button. This interface requires implementing the actionPerformed() method, which defines what happens when the button is clicked.

Example: Button Click Event:

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

public class ButtonClickExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Handling Example");
        JButton button = new JButton("Click Me");

        // Add ActionListener to the button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button clicked!");
            }
        });

        frame.add(button);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

3.3 Using Lambda Expressions (Java 8 and later):

Java 8 introduced **lambda expressions**, which provide a simpler and more concise way to handle events.

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

3.4 Handling Other Events:

You can also handle other events, like mouse or key events, using their respective listeners.

MouseListener Example:

```
button.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        System.out.println("Mouse clicked!");
    }
});
```

KeyListener Example:

```
frame.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent e) {
        System.out.println("Key pressed: " + e.getKeyChar());
    }
});
```

Homework Assignment

1. Create a simple GUI application using `JFrame` and `JPanel`. Use `GridLayout` to arrange four buttons (labeled 1, 2, 3, and 4).
2. Add `ActionListener` to each button so that when clicked, a message is printed in the console with the button number that was clicked.
3. Experiment with different layouts (like `BorderLayout` and `BoxLayout`) to see how the arrangement of components changes.