

UNIT 1

GUI Programming

(12 Hours) — Advanced Java Programming (CAC354)

Important Notice — Please Read

These notes were prepared from the syllabus and general knowledge of Java. **They may contain errors** (typos, outdated API details, or mistakes in code and explanations), so please take care. Verify important points and test all code yourself, and cross-check with your textbook (Core Java Vol. II / Java: The Complete Reference), the official Java documentation, and your teacher's notes.

1.1 Introducing Swing

Swing is Java's main GUI (Graphical User Interface) toolkit, found in the package `javax.swing`. It is built on top of AWT (Abstract Window Toolkit) but, unlike AWT, most Swing components are **lightweight**: they are drawn by Java itself instead of using native operating-system widgets. This gives the same look on every platform and a much richer set of components.

Feature	AWT	Swing
Package	<code>java.awt</code>	<code>javax.swing</code>
Components	Heavyweight (native peers)	Lightweight (pure Java), except top-level containers
Look and feel	Platform dependent	Pluggable look and feel
Component names	Button, Label, TextField	JButton, JLabel, JTextField
Design pattern	None specific	MVC-based (model separate from view)
Richness	Basic set	Tables, trees, tabbed panes, etc.

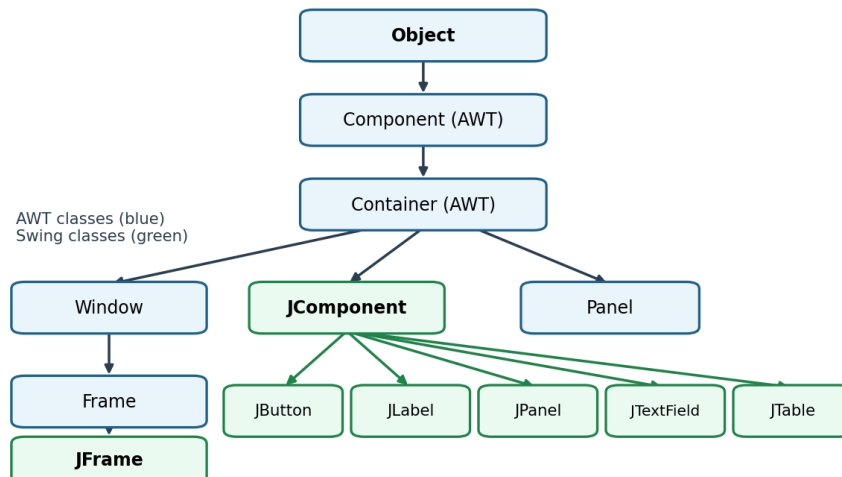


Figure 1.1 — Simplified AWT / Swing class hierarchy

*Note: Swing components should be created and updated on the **Event Dispatch Thread (EDT)**. Use `SwingUtilities.invokeLater(...)` to start a GUI.*

1.2 Creating a Frame

A top-level window is created with `JFrame`. A frame has a title bar, borders and a *content pane* into which components are added.

```

import javax.swing.*;

public class FirstFrame {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("My First Frame");
            frame.setSize(400, 300);           // width, height in pixels
            frame.setLocationRelativeTo(null); // centre on screen
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);            // frames are hidden by default
        });
    }
}
  
```

- `setSize(w,h)` / `setBounds(x,y,w,h)` — size and position of the frame.
- `setDefaultCloseOperation` — what happens on close: `EXIT_ON_CLOSE`, `DISPOSE_ON_CLOSE`, `HIDE_ON_CLOSE`, `DO_NOTHING_ON_CLOSE`.
- `setTitle`, `setResizable`, `setIconImage` — other common frame properties.
- `getContentPane().add(c)` or simply `add(c)` (since Java 5) — adds a component.

1.3 Displaying Information in a Component

To draw text and graphics, extend `JPanel` (or `JComponent`) and override `paintComponent(Graphics g)`. Never call `paintComponent` yourself; call `repaint()` and

the system will call it. Always call `super.paintComponent(g)` first.

```
class MyPanel extends JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello, Swing!", 50, 60);    // (x, y) = baseline start
    }
}
// usage: frame.add(new MyPanel());
```

1.4 Working with 2D Shapes

The `Graphics2D` class (a subclass of `Graphics`) provides the Java 2D API. Cast the `Graphics` object: `Graphics2D g2 = (Graphics2D) g;` Shapes are objects in `java.awt.geom`: `Line2D.Double`, `Rectangle2D.Double`, `Ellipse2D.Double`, `RoundRectangle2D`, `Arc2D`. Shapes are drawn with `g2.draw(shape)` and filled with `g2.fill(shape)`.

```
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    g2.draw(new Rectangle2D.Double(20, 20, 150, 80));
    g2.draw(new Ellipse2D.Double(20, 20, 150, 80));    // ellipse in the same box
    g2.draw(new Line2D.Double(20, 20, 170, 100));
    g2.fill(new Ellipse2D.Double(200, 20, 80, 80));
}
```

Method	Purpose
<code>drawLine(x1,y1,x2,y2)</code>	Draw a line (<code>Graphics</code>)
<code>drawRect</code> / <code>fillRect</code>	Outline / filled rectangle
<code>drawOval</code> / <code>fillOval</code>	Outline / filled oval
<code>drawArc</code> / <code>fillArc</code>	Arc or pie slice
<code>g2.draw(Shape)</code> / <code>g2.fill(Shape)</code>	Draw any Java 2D shape object
<code>g2.setStroke(new BasicStroke(3f))</code>	Change line thickness / dash style

1.5 Using Color

Colors are represented by `java.awt.Color`. You can use predefined constants (`Color.RED`, `Color.BLUE`) or create RGB colors with values 0–255: `new Color(255, 128, 0)`. Use `g2.setColor(c)` to change the drawing colour, `setBackground(c)` and `setForeground(c)` for components. A fourth argument gives **alpha** (transparency): `new Color(0, 0, 255, 100)`.

```

g2.setColor(new Color(31, 97, 141));
g2.fill(new Rectangle2D.Double(10, 10, 100, 50));
g2.setColor(Color.ORANGE);
g2.draw(new Rectangle2D.Double(10, 10, 100, 50));
setBackground(Color.WHITE);           // panel background

```

1.6 Using Special Fonts for Text

Fonts are objects of class `java.awt.Font`: `new Font("Serif", Font.BOLD | Font.ITALIC, 24)`. The logical font names that are always available are *Serif*, *SansSerif*, *Monospaced*, *Dialog*, *DialogInput*. Style constants are `Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`. To find installed fonts use `GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames()`. Text measurement (to centre text) is done with `FontMetrics` or `g2.getFontMetrics()`.

```

Font f = new Font("SansSerif", Font.BOLD, 28);
g2.setFont(f);
FontMetrics fm = g2.getFontMetrics();
int w = fm.stringWidth("Centered");
g2.drawString("Centered", (getWidth() - w) / 2, getHeight() / 2);

```

1.7 Displaying Images

Images are loaded with `ImageIcon` (simple) or `ImageIO.read(File)` (returns a `BufferedImage`). They are drawn with `g.drawImage(image, x, y, observer)`. For a quick image in a label use `new JLabel(new ImageIcon("pic.png"))`.

```

BufferedImage img = ImageIO.read(new File("photo.jpg")); // throws IOException
...
g2.drawImage(img, 0, 0, getWidth(), getHeight(), this); // scaled to panel

```

1.8 Event Handling

1.8.1 Event Handling Basics

Swing uses the **delegation event model**. An **event source** (such as a button) generates an event object when something happens. One or more **event listeners**, registered with the source, receive the event and run code in response.

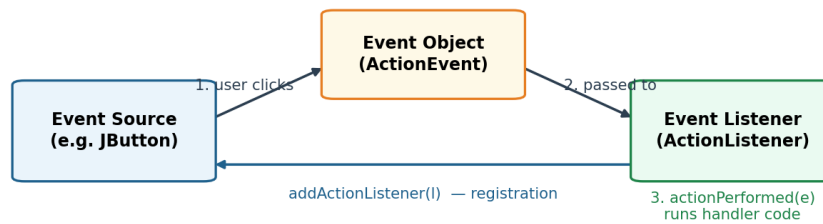


Figure 1.2 — Delegation event model

- **Source:** the object that generates the event (JButton, JTextField, JFrame ...).
- **Event object:** carries the information (getSource(), getActionCommand()).
- **Listener:** implements a listener interface and is registered via addXxxListener(...).

```

JButton b = new JButton("Click me");
b.addActionListener(new ActionListener() { // anonymous inner class
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
});
// lambda form (ActionListener has a single method)
b.addActionListener(e -> System.out.println("Clicked via lambda"));
  
```

1.8.2 Event Classes

All events are subclasses of `java.util.EventObject`. AWT events are in `java.awt.event` and are subclasses of `AWTEvent`.

Event class	Generated by	Listener interface
ActionEvent	Button click, menu item, Enter in text field	ActionListener
MouseEvent	Mouse press, release, click, move, drag	MouseListener, MouseMotionListener
KeyEvent	Keyboard press / release / typed	KeyListener
WindowEvent	Window open, close, minimize	WindowListener
ItemEvent	Checkbox, combo box, radio button selection	ItemListener
TextEvent / DocumentEvent	Text component content change	DocumentListener (Swing)
FocusEvent	Component gains / loses focus	FocusListener
ListSelectionEvent	List / table selection change	ListSelectionListener
AdjustmentEvent	Scrollbar movement	AdjustmentListener

1.8.3 Event Listeners and Adapter Classes

A listener interface may contain many methods. For example `WindowListener` has seven methods (`windowOpened`, `windowClosing`, `windowClosed`, `windowIconified`, `windowDeiconified`, `windowActivated`, `windowDeactivated`). If you only need one, you would still have to write all seven. An **adapter class** provides empty implementations of all methods, so you override only those you need.

Listener interface	Adapter class
<code>WindowListener</code>	<code>WindowAdapter</code>
<code>MouseListener</code>	<code>MouseAdapter</code>
<code>MouseMotionListener</code>	<code>MouseMotionAdapter</code>
<code>KeyListener</code>	<code>KeyAdapter</code>
<code>FocusListener</code>	<code>FocusAdapter</code>
<code>ComponentListener</code>	<code>ComponentAdapter</code>
<code>ActionListener</code> , <code>ItemListener</code> , etc. (one method)	No adapter needed

```
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {        // only this method needed
        System.out.println("Closing...");
        System.exit(0);
    }
});
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Clicked at " + e.getX() + "," + e.getY());
    }
});
```

Example 1.1 — Simple adder with ActionListener

Create a frame with two text fields, a button, and a label that shows the sum of the two numbers.

Solution

Register an `ActionListener` on the button; parse both fields with `Integer.parseInt` and set the label text.

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

public class Adder extends JFrame {
    JTextField t1 = new JTextField(5), t2 = new JTextField(5);
    JLabel result = new JLabel("Sum = ");
    public Adder() {
```

```

super("Adder");
setLayout(new FlowLayout());
JButton add = new JButton("Add");
add(t1); add(t2); add(add); add(result);
add.addActionListener(e -> {
    try {
        int s = Integer.parseInt(t1.getText().trim())
            + Integer.parseInt(t2.getText().trim());
        result.setText("Sum = " + s);
    } catch (NumberFormatException ex) {
        result.setText("Enter valid integers");
    }
});
setSize(350, 120); setDefaultCloseOperation(EXIT_ON_CLOSE);
setLocationRelativeTo(null); setVisible(true);
}
public static void main(String[] a) { SwingUtilities.invokeLater(Adder::new); }
}

```

1.9 Swing and the MVC Design Pattern

Model–View–Controller (MVC) separates a component into three parts:

Part	Role	Swing example
Model	Stores the state / data of the component	ButtonModel, ListModel, TableModel, Document
View	How the component is displayed	The painted appearance (UI delegate)
Controller	Handles user input and updates the model	Event handling code / listeners

Swing actually uses a simplified form called **separable model architecture** (sometimes called *Model–Delegate*), where view and controller are combined in a *UI delegate*. Because of the separation, one model can have several views (for example two tables showing the same TableModel), and the **look and feel** can be changed at run time with `UIManager.setLookAndFeel(...)`.

```

DefaultListModel<String> model = new DefaultListModel<>(); // model
model.addElement("Java"); model.addElement("Swing");
JList<String> list = new JList<>(model); // view
model.addElement("MVC"); // list updates automatically

```

Note: Common Swing models: DefaultButtonModel, DefaultListModel, DefaultTableModel, DefaultTreeModel, PlainDocument, DefaultComboBoxModel.

1.10 Layout Management

A **layout manager** decides how components are sized and positioned inside a container. Set it with `setLayout(new XxxLayout())`. Using layout managers (instead of absolute positions

with `setLayout(null)` keeps the GUI usable when the window is resized or run on another platform.

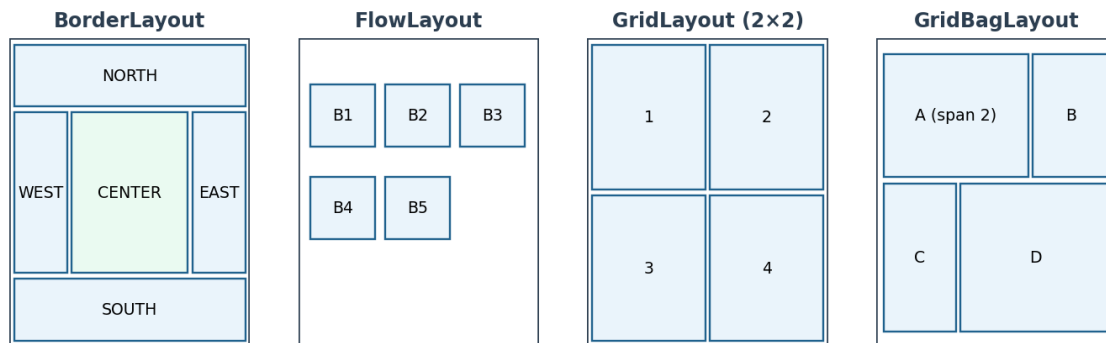


Figure 1.3 — Common layout managers

Layout	Default for	Behaviour
FlowLayout	JPanel	Places components left to right, wraps to next row when full
BorderLayout	JFrame content pane	Five regions: NORTH, SOUTH, EAST, WEST, CENTER (centre takes remaining space)
GridLayout	—	Equal-sized cells in rows × columns grid
GridBagLayout	—	Flexible grid; components can span cells (uses GridBagConstraints)
BoxLayout	Box	Single row or column (X_AXIS / Y_AXIS)
CardLayout	—	Stack of cards; only one visible at a time (wizard-like)
null (absolute)	—	Manual <code>setBounds()</code> ; not recommended

```
JFrame f = new JFrame("Layouts");
f.setLayout(new BorderLayout(5, 5));
f.add(new JButton("North"), BorderLayout.NORTH);
f.add(new JButton("Centre"), BorderLayout.CENTER);
```

```
JPanel grid = new JPanel(new GridLayout(2, 3, 4, 4)); // rows, cols, hgap, vgap
for (int i = 1; i <= 6; i++) grid.add(new JButton("" + i));
f.add(grid, BorderLayout.SOUTH);
```

1.11 Basic Swing Components

Component	Purpose	Key methods / events
JLabel	Display text or icon	<code>setText</code> , <code>setIcon</code>
JButton	Push button	<code>addActionListener</code> , <code>setEnabled</code>
JTextField	Single-line text input	<code>getText</code> , <code>setText</code> ; <code>ActionListener</code>
JPasswordField	Masked text input	<code>getPassword()</code> returns <code>char[]</code>

Component	Purpose	Key methods / events
JTextArea	Multi-line text (usually inside JScrollPane)	append, setLineWrap
JCheckBox	Independent on/off option	isSelected; ItemListener
JRadioButton	One choice from group (use ButtonGroup)	isSelected
JComboBox	Drop-down list	getSelectedItem; ActionListener
JList	List of items (select one or many)	getSelectedValue; ListSelectionListener
JMenuBar / JMenu / JMenuItem	Menus	setJMenuBar, add
JScrollPane	Adds scroll bars to a component	new JScrollPane(component)
JTable	Rows and columns of data	TableModel
JTree	Hierarchical data	TreeModel
JOptionPane	Standard dialog boxes	showMessageDialog, showInputDialog, showConfirmDialog

Example 1.2 — Mini registration form

Build a form with name field, gender radio buttons, a course combo box, and a Submit button that shows the data using JOptionPane.

Solution

Use a GridLayout for the form, group the radio buttons with ButtonGroup, read the selections in the button's listener.

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

public class RegForm extends JFrame {
    public RegForm() {
        super("Registration");
        JTextField name = new JTextField(15);
        JRadioButton m = new JRadioButton("Male", true), f = new JRadioButton("Female");
        ButtonGroup g = new ButtonGroup(); g.add(m); g.add(f);
        JComboBox<String> course = new JComboBox<>(new String[]{"BCA", "BIT", "BBA"});
        JCheckBox agree = new JCheckBox("I agree");
        JButton submit = new JButton("Submit");

        setLayout(new GridLayout(5, 2, 6, 6));
        add(new JLabel("Name:")); add(name);
        add(new JLabel("Gender:")); JPanel gp = new JPanel(); gp.add(m); gp.add(f); add(gp);
        add(new JLabel("Course:")); add(course);
        add(agree); add(new JLabel());
        add(submit);

        submit.addActionListener(e -> {
```

```

        if (!agree.isSelected()) { JOptionPane.showMessageDialog(this, "Please agree"); return; }
        String msg = name.getText() + ", " + (m.isSelected() ? "Male" : "Female")
            + ", " + course.getSelectedItemAt();
        JOptionPane.showMessageDialog(this, msg);
    });
    pack(); setDefaultCloseOperation(EXIT_ON_CLOSE); setVisible(true);
}
public static void main(String[] a) { SwingUtilities.invokeLater(RegForm::new); }
}

```

1.12 Short Questions for Revision

- Differentiate between AWT and Swing.
- What is the Event Dispatch Thread and why is it important?
- Explain the delegation event model with a diagram.
- What is an adapter class? Why is it used?
- Explain MVC in Swing. How is it different from classical MVC?
- List any four layout managers and the default layout of JPanel and JFrame.
- Write a program to draw a filled circle and a rectangle using Graphics2D.

Key Takeaways — Unit 1

- ✓ Swing (javax.swing) provides lightweight, platform-independent components; always start GUIs on the EDT.
- ✓ JFrame is the top-level window; draw custom graphics by overriding paintComponent() and using Graphics2D.
- ✓ Use Color, Font and ImageIO/ImageIcon to enhance the appearance of drawings.
- ✓ Delegation event model: source → event object → registered listener. Adapters avoid writing unused methods.
- ✓ Swing uses a model–delegate (MVC-like) design: models hold data, UI delegates paint and handle input.
- ✓ Layout managers (Flow, Border, Grid, GridBag, Box, Card) control component placement automatically.

— End of Unit 1 —

Reminder: these notes may contain errors — verify against your textbook and run the code yourself.