S7 Inf A3 — Java for Graphical and Mobile Programming · Centrale Lyon

Stéphane Derrode

CM4

Swing: Components, Layouts & Events

- JFrame, JPanel and the Swing component hierarchy
- Layout managers — FlowLayout, BorderLayout, GridLayout
- Basic components — JButton, JLabel, JTextField, JComboBox
- Event-driven programming — ActionListener, lambda syntax
- Menus and dialogs — JMenuBar, JOptionPane

Module Overview

Part 1 — Java Fundamentals	Part 2 — GUI with Swing	Part 3 — Android
CM1 From Python to Java	**CM4** Swing: Components & Layouts	**CM6** Android Architecture
CM2 Collections, Exceptions & I/O	**CM5** Graphics 2D & Events	**CM7** Layouts, Navigation & Intents
CM3 Inheritance, Interfaces		**CM8** Persistence & Threads

Projects: ★ Spider Game (Part 2) ★ Calculator / Currency Converter (Part 3)

Exam 50% + BEs 50%

Stéphane Derrode — Centrale Lyon · S7 Inf A3

CM4 — Agenda

01 Swing architecture & JFrame

02 Layout managers

03 Basic components

04 Event-driven programming

05 Menus & dialogs

06 Full example — Voiture UI

Swing Architecture — Component Hierarchy

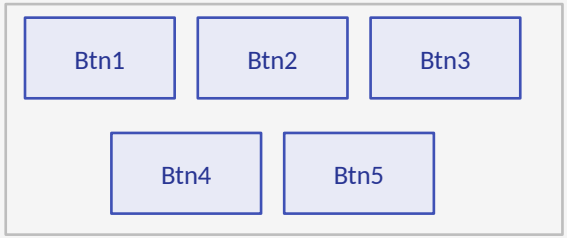
JFrame — the top-level window

JFrame extends Frame (AWT). It is NOT a JComponent — it is a window that contains a content pane (itself a JPanel) where you add all your components.

```
JFrame frame = new JFrame("Voiture Control");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
;
frame.setSize(400, 300);
frame.setVisible(true);
```

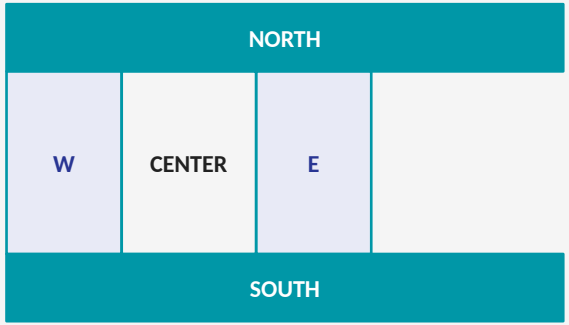
Layout Managers

FlowLayout
left→right, wraps




```
// FlowLayout (default for
JPanel)
JPanel p = new JPanel();
p.setLayout(new FlowLayout());
p.add(new JButton("Btn1"));
p.add(new JButton("Btn2"));
```

BorderLayout
5 zones




```
// BorderLayout
JPanel p = new JPanel(
    new BorderLayout());
p.add(new JLabel("Title"),
    BorderLayout.NORTH);
p.add(new JButton("OK"),
    BorderLayout.SOUTH);
```

GridLayout(2,3)
equal cells


```
// GridLayout
JPanel p = new JPanel(
    new GridLayout(2, 3));
for(int i=1;i<=6;i++)
    p.add(new JButton(""+i));
```

BoxLayout — single row or column
new BoxLayout(p, BoxLayout.Y_AXIS) — stacks vertically.
BoxLayout.X_AXIS — lines up horizontally.
Useful for toolbars, side panels.

CardLayout — switch between views
Shows one panel at a time: card.show(parent, "name").
Ideal for: placement phase → movement phase, login screen → game screen.

Basic Components

```
// JLabel — static text or icon
JLabel label = new JLabel("Speed: 0 km/h");
label.setFont(new Font("Arial", Font.BOLD, 14));
label.setForeground(Color.BLUE);

// JButton — clickable
JButton startBtn = new JButton("Start");
startBtn.setEnabled(false);          // disabled

// JTextField — single-line input
JTextField powerField = new JTextField("4", 5); // 5 cols wide
String text = powerField.getText();
powerField.setText("6");

// JPasswordField — hidden input
JPasswordField pwd = new JPasswordField(10);

// JCheckBox
JCheckBox turbo = new JCheckBox("Turbo mode", false);
boolean checked = turbo.isSelected();

// JComboBox — drop-down
String[] gears = {"1", "2", "3", "4", "5"};
JComboBox<String> gearBox = new JComboBox<>(gears);
gearBox.setSelectedIndex(0);
String selected = (String) gearBox.getSelectedItem();

// JTextArea — multi-line
```

Useful methods — all components

setEnabled(boolean) setVisible(boolean) setPreferredSize(Dimension) setToolTipText(String) setBorder(BorderFactory.createX())

Colors & fonts

setForeground(Color.RED) setBackground(Color.WHITE) setFont(new Font("Arial", Font.BOLD, 14)) — Color constants: RED, BLUE, GREEN, BLACK, WHITE, GRAY

JScrollPane wrapper

Always wrap JTextArea in a JScrollPane — JTextArea doesn't scroll by itself. Add the JScrollPane to the panel, not the JTextArea directly.

Tip: setPreferredSize()

Layout managers respect preferred sizes. new Dimension(200, 30) gives a 200px wide, 30px tall component. Not always honoured — depends on the layout.

Event-Driven Programming — the Listener Model

The event model

1 — Anonymous inner class

```
startBtn.addActionListener(
    new ActionListener() {
        @Override
        public void
actionPerformed(
                ActionEvent e)
{
            voiture.demarre();
            updateDisplay();
        }
    }
);
```

2 — Lambda (Java 8+, recommended)

```
startBtn.addActionListener(e ->
{
    voiture.demarre();
    updateDisplay();
});

// Even shorter for one line:
stopBtn.addActionListener(
    e -> voiture.arrete());
```

3 — implements ActionListener

```
public class VoitureUI
        implements
ActionListener {

    @Override
    public void
actionPerformed(
            ActionEvent e) {
        if (e.getSource() ==
startBtn)
            voiture.demarre();
        else
if(e.getSource()==stopBtn)
```

Recommendation

Use lambdas (option 2) for simple handlers. Use option 3 (implements) only when you need to distinguish which component fired the event via e.getSource().

The 4 Ways to Write a Listener — BipBip Progression

From Derrode (2024) — same button, 4 progressively cleaner solutions

① Separate listener class

verbose

```
class Ecoute
    implements
ActionListener {
  JLabel label;
  public Ecoute(JLabel l)
{label=l;}
  @Override
  public void
actionPerformed(
      ActionEvent e) {

label.setText("Done!");
  }
}
// Usage:
Ecoute el = new
Ecoute(label);
btn.addActionListener(el)
;
```

+ flexible
– lots of code

② implements ActionListener

single handler

```
public class BipBip
    implements
ActionListener {
  JLabel label;
  JButton btn;
  //
btn.addActionListener(thi
s);

  @Override
  public void
actionPerformed(
      ActionEvent e) {
    // e.getSource()
needed if
    // multiple buttons!

label.setText("Done!");
  }
```

+ simple
– one handler only

③ Inner named class

best of both

```
public class BipBip {
  JLabel label;
  public BipBip() {
    // ...

btn.addActionListener(
      new
DoItListener());
  }
  class DoItListener
      implements
ActionListener {
    public void
actionPerformed(
        ActionEvent e) {
      // has access to
label!

label.setText("Done!");
```

+ flexible + concise
– extra class names

④ Lambda (recommended)

★ shortest

```
public class BipBip {
  JLabel label;
  public BipBip() {
    // ...


btn.addActionListener(
      e ->
label.setText("Done!"));


    // access to label ✓
    // concise ✓
    // multiple buttons:
just
    // add more lambdas ✓
  }
}
```

+ flexible + concise
– none (Java 8+)

Event Dispatch Thread & Other Listeners

```
// ── Starting Swing correctly ──────────────────
public static void main(String[] args) {
    // ALWAYS start Swing on the EDT
    SwingUtilities.invokeLater(() -> {
        VoitureUI ui = new VoitureUI();
        ui.setVisible(true);
    });
}

// ── Other common listeners ────────────────────

// ItemListener: JCheckBox, JComboBox state changes
turboBox.addItemListener(e -> {
    boolean on = (e.getStateChange() == ItemEvent.SELECTED);
    System.out.println("Turbo: " + on);
});

// ChangeListener: JSlider value changes
JSlider speedSlider = new JSlider(0, 200, 0);
speedSlider.addChangeListener(e -> {
    int val = speedSlider.getValue();
    speedLabel.setText("Speed: " + val + " km/h");
});

// WindowListener: intercept window close
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
```

The EDT rule

All UI updates MUST happen on the Event Dispatch Thread. Creating components off-EDT causes race conditions.

If you update the UI from a background thread: SwingUtilities.invokeLater(() ->

Key listener types

ActionListener — button, menu, Enter key
ItemListener — checkbox, combo state
ChangeListener — slider value
DocumentListener — text field content
WindowListener — window lifecycle

WindowAdapter

WindowListener has 7 methods. WindowAdapter provides empty implementations so you only override what you need — typical Adapter pattern.

Note on WindowAdapter

This is the Adapter design pattern — you'll see it again in MouseAdapter (CM5). It avoids implementing all 7 empty methods of MouseListener.

Menus & Dialogs

```
// ── JMenuBar ──────────────────────────────────────
JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");
JMenuItem saveItem = new JMenuItem("Save");
JMenuItem exitItem = new JMenuItem("Exit");

saveItem.addActionListener(e -> saveData());
exitItem.addActionListener(e -> System.exit(0));

fileMenu.add(saveItem);
fileMenu.addSeparator();
fileMenu.add(exitItem);

JMenu helpMenu = new JMenu("Help");
helpMenu.add(new JMenuItem("About"));

menuBar.add(fileMenu);
menuBar.add(helpMenu);
frame.setJMenuBar(menuBar);

// ── JOptionPane — simple dialogs ──────────────────
// Information
JOptionPane.showMessageDialog(frame,
    "Game over!", "Result",
    JOptionPane.INFORMATION_MESSAGE);

// Confirmation
```

JMenuBar structure

JMenuBar (the bar)
└ JMenu ("File", "Edit"...)
└ JMenuItem ("Save", "Exit")
└ JCheckBoxMenuItem
└ JRadioButtonMenuItem

JOptionPane dialog types

showMessageDialog — information, warning, error
showConfirmDialog — YES/NO/CANCEL → returns int
showInputDialog — text entry → returns String

Icon constants

INFORMATION_MESSAGE WARNING_MESSAGE
ERROR_MESSAGE QUESTION_MESSAGE
PLAIN_MESSAGE
Passed as the 4th argument to showMessageDialog

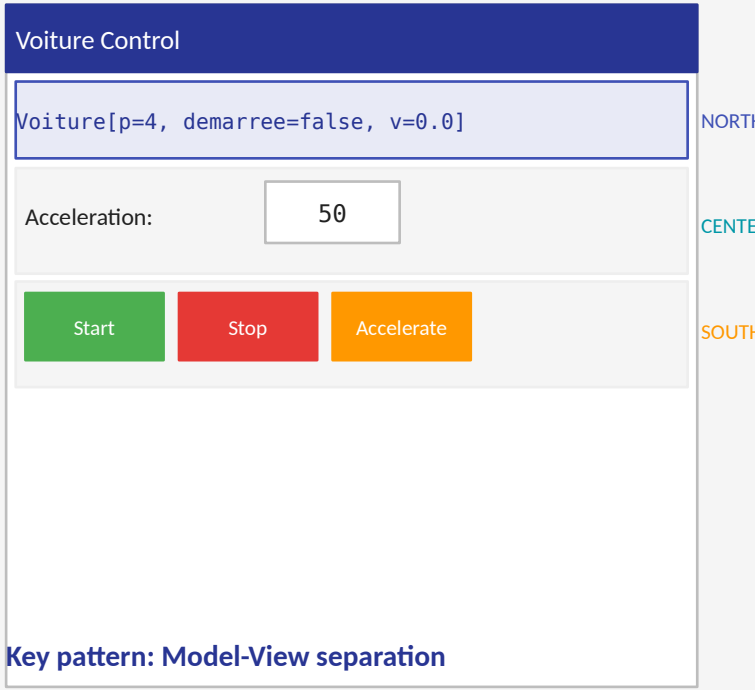
Context menu (JPopupMenu)

```
JPopupMenu popup = new JPopupMenu();
popup.add(new JMenuItem("Reset"));
component.addMouseListener(new MouseAdapter()
{
  public void mousePressed(MouseEvent e) {
    if(e.isPopupTrigger()) popup.show(...);
  }
});
```

Full Example — Voiture Control Panel

```
public class VoitureUI extends JFrame {
    private Voiture voiture = new Voiture(4);
    private JLabel statusLabel;
    private JTextField accelField;

    public VoitureUI() {
        setTitle("Voiture Control");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // NORTH: status display
        statusLabel = new JLabel(voiture.toString());
        statusLabel.setBorder(
            BorderFactory.createEmptyBorder(8,8,8,8));
        add(statusLabel, BorderLayout.NORTH);

        // CENTER: acceleration input
        JPanel center = new JPanel(new FlowLayout());
        center.add(new JLabel("Acceleration:"));
        accelField = new JTextField("50", 5);
        center.add(accelField);
        add(center, BorderLayout.CENTER);

        // SOUTH: control buttons
        JPanel south = new JPanel(new FlowLayout());
        JButton startBtn = new JButton("Start");
        JButton stopBtn  = new JButton("Stop");
        JButton accelBtn = new JButton("Accelerate");
```

UI layout



Key pattern: Model-View separation

The Voiture (model) has no Swing code.
The VoitureUI (view) delegates all logic to voiture.*
refresh() keeps the display in sync.

Swing Quick Reference

Component	Key constructor / methods	Listener
`JFrame`	setTitle setSize pack setVisible setDefaultCloseOperation	`WindowListener`
`JPanel`	setLayout add setBorder setBackground	—
`JButton`	new JButton(text) setText setEnabled setIcon	`ActionListener`
`JLabel`	new JLabel(text) setText setIcon setHorizontalAlignment	—
`JTextField`	new JTextField(cols) getText setText setEditable	`ActionListener` `DocumentListener`
`JCheckBox`	new JCheckBox(text, selected) isSelected setSelected	`ItemListener`
`JComboBox`	new JComboBox<>(array) getSelectedItem addItem	`ActionListener` `ItemListener`
`JSlider`	new JSlider(min,max,val) getValue setPaintTicks	`ChangeListener`

Essential imports

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

Swing in Maven — No Extra Dependency

Good news: Swing is part of the JDK

No dependency to add to pom.xml — javax.swing is bundled with Java SE 17. Just add the three imports and you're ready to go.

```
<!-- pom.xml — no extra dependency needed for Swing
-->
<project>
  ...
  <properties>

<maven.compiler.source>17</maven.compiler.source>

<maven.compiler.target>17</maven.compiler.target>
  </properties>

  <!-- To run with mvn exec:java -->
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>3.1.0</version>
        <configuration>

<mainClass>com.s7infa3.VoitureUI</mainClass>
        </configuration>
      </plugin>
    </plugins>
```

```
// Running from VSCodium:
// Option 1: click ▶ Run above main()
//           → works, but no native look-and-
feel

// Option 2: mvn exec:java
//           → uses the configured mainClass

// Option 3: terminal
mvn compile exec:java   -
Dexec.mainClass="com.s7infa3.VoitureUI"

// Setting native look-and-feel:
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(

UIManager.getSystemLookAndFeelClassName());
    } catch (Exception ignored) {}
    SwingUtilities.invokeLater(() -> {
        new VoitureUI().setVisible(true);
    });
}
```

CM4 — Key Takeaways

JFrame	Top-level window; contains a JPanel content pane; always call pack() or setSize()
Layouts	FlowLayout (default), BorderLayout (5 zones), GridLayout (equal cells); nest JPanels
Components	JButton JLabel JTextField JCheckBox JComboBox JSlider JTextArea+JScrollPane
Events	addActionListener(e -> ...); always use SwingUtilities.invokeLater() for startup
Menus	JMenuBar > JMenu > JMenuItem; setJMenuBar() on JFrame
Dialogs	JOptionPane.showMessageDialog / showConfirmDialog / showInputDialog

Up next → TD4: Voiture Control Panel & UI practice