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

Stéphane Derrode

CM5

Graphics 2D & Events

- Custom painting — paintComponent() & Graphics2D
- Drawing primitives, colors, fonts & strokes
- Images — BufferedImage & ImageIO
- Mouse events — MouseListener & MouseMotionListener
- Keyboard events — KeyListener & focus management
- Animation — javax.swing.Timer

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

CM5 — Agenda

01 Custom painting — paintComponent()

02 Graphics2D — shapes, colors, fonts

03 Images — BufferedImage & ImageIO

04 Mouse events

05 Keyboard events & focus

06 Animation — Timer

Custom Painting — paintComponent()

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

public class DrawPanel extends JPanel {

    // Override paintComponent — NEVER call it directly
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);   // ← ALWAYS call this
first!

        // Cast to Graphics2D for more features
        Graphics2D g2 = (Graphics2D) g;

        // Enable antialiasing for smooth edges
        g2.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

        // Draw a filled blue rectangle
        g2.setColor(Color.BLUE);
        g2.fillRect(20, 20, 100, 60);

        // Draw a red circle outline
        g2.setColor(Color.RED);
        g2.drawOval(150, 20, 80, 80);

        // Draw text
```

The golden rules

1. Always call super.paintComponent(g) first.
2. Never call paintComponent() directly — call repaint().
3. Never store the Graphics object — valid only during the call.

Graphics vs Graphics2D

Graphics is the original AWT class. Graphics2D extends it with strokes, transforms, better rendering. Always cast:
Graphics2D g2 = (Graphics2D) g;

RenderingHints

KEY_ANTIALIASING + VALUE_ANTIALIAS_ON — smooth edges.
KEY_TEXT_ANTIALIASING + VALUE_TEXT_ANTIALIAS_ON — smooth text.

The repaint() call chain

repaint() → update() → paint(g)
paint() calls in order:
paintComponent() ← your code here
paintBorder()
paintChildren()
Never call paintComponent() directly.

Graphics2D — Shapes, Colors & Strokes

```
// ── Shapes ──────────────────────────────────────
g2.drawRect(x, y, w, h);          // rectangle outline
g2.fillRect(x, y, w, h);          // filled rectangle
g2.drawRoundRect(x, y, w, h, arcW, arcH); // rounded corners
g2.drawOval(x, y, w, h);          // ellipse/circle outline
g2.fillOval(x, y, w, h);          // filled ellipse/circle
g2.drawLine(x1, y1, x2, y2);      // line segment

// Polygon (e.g. triangle)
int[] xs = {100, 50, 150};
int[] ys = {50, 150, 150};
g2.fillPolygon(xs, ys, 3);

// ── Colors ──────────────────────────────────────
g2.setColor(Color.RED);
g2.setColor(new Color(255, 128, 0));    // RGB
g2.setColor(new Color(0, 0, 255, 128)); // RGBA (semi-
transparent)

// ── Strokes ─────────────────────────────────────
g2.setStroke(new BasicStroke(3.0f));    // 3px thick
g2.setStroke(new BasicStroke(
    2.0f,                               // width
    BasicStroke.CAP_ROUND,              // end cap
    BasicStroke.JOIN_ROUND,             // join style
    10.0f,                              // miter limit
    new float[]{6, 4},                  // dash: 6 on, 4 off
    0.0f));                             // dash phase
```

draw vs fill

draw*() draws the outline only. fill*() fills the interior. Use draw for empty shapes, fill for solid. Both respect the current color.

Color constants

Color.RED BLUE GREEN BLACK WHITE GRAY YELLOW ORANGE CYAN MAGENTA PINK

Or use new Color(r, g, b) for any RGB value.

Font styles

Font.PLAIN Font.BOLD Font.ITALIC Font.BOLD|Font.ITALIC

Font name: "Arial", "Times New Roman", "Monospaced"

For cross-platform: use Font.DIALOG,

Useful for Spider Game

Grid lines: drawLine()

Cells background: fillRect()

Pieces: fillOval() for circles

Player names: drawString()

Stroke width for selected piece: BasicStroke(3f)

Images — BufferedImage & ImageIO

```
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;

public class ImagePanel extends JPanel {
    private BufferedImage img;

    public ImagePanel() {
        try {
            // Load from file
            img = ImageIO.read(
                new File("resources/background.png"));

            // Or load from classpath (recommended for Maven)
            img = ImageIO.read(
                getClass().getResource(
                    "/images/board.png"));
        } catch (IOException e) {
            System.err.println("Image not found: " + e);
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
```

Maven resource folder

Place images in src/main/resources/images/. Maven copies them to the classpath. Access with getClass().getResource("/images/piece.png").

Supported formats

PNG, JPEG, GIF, BMP — all handled by ImageIO. PNG recommended for game graphics (lossless + transparency).

Creating image in code

```
BufferedImage img = new BufferedImage(w, h,
TYPE_INT_ARGB);
Graphics2D g = img.createGraphics();
// draw on g...
g.dispose();
```

ImageIcon for buttons

```
ImageIcon icon = new ImageIcon("piece_red.png");
new JButton(icon);
new JLabel(icon);
```

— Quick way to add images to components without custom painting.

Mouse Events — MouseListener & MouseMotionListener

```
// MouseAdapter: override only what you need
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        int x = e.getX();         // click position
        int y = e.getY();
        int btn = e.getButton();  // BUTTON1=left,
BUTTON3=right
        int clicks = e.getClickCount(); // 1 or 2 (double-
click)
        System.out.println("Clicked at (" + x + "," + y +
")");
    }

    @Override
    public void mousePressed(MouseEvent e) { /* button down */
}

    @Override
    public void mouseReleased(MouseEvent e) { /* button up
*/ }

    @Override
    public void mouseEntered(MouseEvent e) { /* cursor enters
*/ }

    @Override
    public void mouseExited(MouseEvent e)  { /* cursor leaves
```

MouseListener vs MouseAdapter

MouseListener (interface) has 5 methods — you must implement all. MouseAdapter (abstract class) provides empty implementations — override only what you need.

Useful for Spider Game

mouseClicked: detect which cell was clicked
Convert pixel (x,y) → grid (row,col):

```
int col = e.getX() / cellSize;
int row = e.getY() / cellSize;
```

Popup trigger

e.isPopupTrigger() returns true for right-click (platform-dependent). Check in BOTH mousePressed and mouseReleased for cross-platform compatibility.

mouseClicked vs mousePressed

mouseClicked fires only if press and release occur at the same position. For games, use mousePressed — it's more responsive. mouseReleased is used for drag-and-drop end detection.

Keyboard Events & Focus Management

```
// Add keyboard listener to your panel
panel.addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {
        int code = e.getKeyCode();

        switch (code) {
            case KeyEvent.VK_LEFT  -> moveLeft();
            case KeyEvent.VK_RIGHT -> moveRight();
            case KeyEvent.VK_UP    -> moveUp();
            case KeyEvent.VK_DOWN  -> moveDown();
            case KeyEvent.VK_SPACE -> shoot();
            case KeyEvent.VK_ESCAPE -> pauseGame();
        }
        repaint();
    }

    @Override
    public void keyTyped(KeyEvent e) {
        // fires for printable characters ('a', '1', '@')
        char c = e.getKeyChar();
        inputBuffer += c;
    }

    @Override
    public void keyReleased(KeyEvent e) { /* key up */ }
});
```

keyPressed vs keyTyped

keyPressed — any key, including arrows, F-keys, Ctrl. keyTyped — printable chars only, with keyboard layout applied. Use keyPressed for game controls.

Common key codes

VK_LEFT VK_RIGHT VK_UP VK_DOWN VK_SPACE VK_ENTER VK_ESCAPE VK_A…VK_Z VK_0…VK_9 VK_F1…VK_F12

Focus is required!

A component only receives key events when it has keyboard focus. Always call setFocusable(true) and requestFocusInWindow() on your drawing panel.

Key Bindings (alternative)

getInputMap() / getActionMap() is more robust than KeyListener — it works even when the component doesn't have focus, and survives focus changes. Recommended for complex apps.

Animation — javax.swing.Timer

```
import javax.swing.Timer;

public class AnimationPanel extends JPanel {
    private int ballX = 0, ballY = 100;
    private int dx = 3, dy = 2;  // speed
    private Timer timer;

    public AnimationPanel() {
        // Fire every 16ms ≈ 60 fps
        timer = new Timer(16, e -> {
            update();   // move objects
            repaint();  // trigger paintComponent
        });
        timer.start();
    }

    private void update() {
        ballX += dx;
        ballY += dy;
        // Bounce off edges
        if (ballX < 0 || ballX > getWidth()-20)  dx = -dx;
        if (ballY < 0 || ballY > getHeight()-20) dy = -dy;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
```

javax.swing.Timer vs java.util.Timer

Always use javax.swing.Timer for UI animations — it fires on the EDT automatically. java.util.Timer fires on a background thread — UI updates from it require invokeLater().

Frame rate guide

16ms ≈ 60fps — smooth animation.
33ms ≈ 30fps — acceptable for simple games.
100ms — slow, noticeable lag.
For Spider Game: no animation needed unless you add move highlights.

Game loop pattern

Timer fires → update() moves objects → repaint() schedules paintComponent(). Keep update() and paintComponent() independent — no logic in paint, no drawing in update.

Stop/start the timer

timer.stop() — stops the loop (pause).
timer.start() — resumes.
timer.isRunning() — check state.
Always stop the timer when the window closes:
frame.addWindowListener(new WindowAdapter() { void windowClosing(...) { timer.stop(); } })

Putting It Together — The 3×3 Grid Component

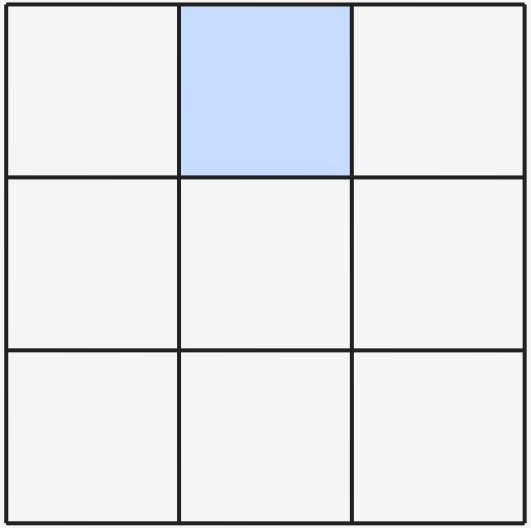
```
public class GridPanel extends JPanel {
    private static final int CELL = 120; // pixels per cell
    private int selectedRow = -1, selectedCol = -1;

    public GridPanel() {
        setPreferredSize(new Dimension(CELL*3, CELL*3));
        setBackground(Color.WHITE);

        addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                // Convert pixels → grid coordinates
                selectedCol = e.getX() / CELL;
                selectedRow = e.getY() / CELL;
                System.out.println(
                    "Cell (" + selectedRow
                    + "," + selectedCol + ")");
                repaint();
            }
        });
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
```

This is the foundation of the Spider Game project



Grid with pieces

Extension for Spider Game
Replace the simple click handler with the full game logic: check whose turn it is, validate the placement, detect a win condition, then call repaint().

Graphics2D Quick Reference

Task	Method / code
Set color	`g2.setColor(Color.RED)  or  g2.setColor(new Color(r,g,b))`
Draw outline	`g2.drawRect(x,y,w,h)  drawOval(x,y,w,h)  drawLine(x1,y1,x2,y2)`
Fill shape	`g2.fillRect(x,y,w,h)  fillOval(x,y,w,h)  fillPolygon(xs,ys,n)`
Draw text	`g2.setFont(new Font("Arial",Font.BOLD,14))  g2.drawString(s,x,y)`
Center text	`FontMetrics fm=g2.getFontMetrics();  int cx=(w-fm.stringWidth(s))/2;`
Line thickness	`g2.setStroke(new BasicStroke(3f))`
Antialiasing	`g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, VALUE_ANTIALIAS_ON)`
Load image	`BufferedImage img = ImageIO.read(getClass().getResource("/img.png"))`
Draw image	`g2.drawImage(img, x, y, width, height, this)`
Mouse click pos	`int x=e.getX();  int y=e.getY();  – in MouseAdapter.mouseClicked()`
Pixel → cell	`int col=e.getX()/CELL_SIZE;  int row=e.getY()/CELL_SIZE;`
Animation timer	`new Timer(16, e -> { update(); repaint(); }).start();`

Part 2 Recap — Building a GUI Application

CM4 + CM5: everything needed for the Spider Game project

CM4 — Structure	CM5 — Custom drawing	Spider Game needs
JFrame window setup	paintComponent() + repaint()	GridPanel extends JPanel
JPanel + layout nesting	Graphics2D shapes & colors	Board state → paintComponent
JButton, JLabel, JTextField	Mouse click → cell (row, col)	Click → place/move piece
ActionListener (lambda)	MouseAdapter selection highlight	Win detection → JOptionPane
JMenuBar, JOptionPane	repaint() on state change	New game → reset + repaint

Key design principle: Model ↔ View separation

Keep game logic (Board, Cell, Piece, Player) completely free of Swing code. The GamePanel reads the model state and renders it. User actions (mouse clicks) update the model, then call repaint(). This separation makes the code testable and maintainable.

CM5 — Key Takeaways

paintComponent	Override in JPanel; call super() first; trigger with repaint(), never directly
Graphics2D	Cast from Graphics; drawRect/fillOval/drawLine/drawString; set color & stroke before drawing
Antialiasing	setRenderingHint(KEY_ANTIALIASING, VALUE_ANTIALIAS_ON) — always add at start of paintComponent
Mouse events	MouseAdapter.mouseClicked(); e.getX()/getY(); pixel÷CELL_SIZE → grid row/col
Keyboard	setFocusable(true) + requestFocusInWindow(); KeyAdapter.keyPressed(); VK_* constants
Timer	new Timer(16, e -> { update(); repaint(); }) — fires on EDT; stop() / start()

Up next → TD5: drawing canvas + 3×3 grid, then Spider Game project