Java Programming Cheat Sheet

Comprehensive Reference Guide – Classic Video Game Edition

1 Basic Data Types & Control Structures

1.1 Primitive Data Types

Type	Size	Range / Notes
byte	8-bit	−128 to 127
short	16-bit	−32,768 to 32,767
int	32-bit	±2.1 billion
long	64-bit	Use L suffix
float	32-bit	Use f suffix
double	64-bit	Default decimal
char	16-bit	Unicode character
boolean	1-bit	true / false

```
int score = 999999;
double health = 100.0;
char grade = 'A';
boolean gameOver = false;
long highScore = 9999999999L;
```

1.2 Variable Declaration

Syntax	Description
int x = 5;	Explicit type
var x = 5;	Type inference (Java 10+)
final int X = 5;	Constant (immutable)

1.3 Operators

Category	Operators
Arithmetic	+ - * / % ++ --
Relational	== != < > <= >=
Logical	&& || !
Assignment	= += -= *= /= %=
Bitwise	& | ^ ~ << >> >>>
Ternary	condition ? a : b

```
// Pac-Man power pellet logic
String status = isPoweredUp ? "HUNTING" : "FLEEING
    ";
lives += (score % 10000 == 0) ? 1 : 0;
```

1.4 Control Structures

If-Else Statement:

```
if (lives > 0) {
    continueGame();
} else if (hasExtraLife) {
    respawn();
} else {
    gameOver();
}
```

Switch Statement (Traditional):

```
switch (direction) {
    case "UP":    mario.jump(); break;
    case "DOWN":  mario.crouch(); break;
    case "LEFT":  mario.moveLeft(); break;
    case "RIGHT": mario.moveRight(); break;
    default:      mario.idle();
}
```

Switch Expression (Java 14+):

```
String action = switch (buttonPressed) {
    case 'A' -> "JUMP";
    case 'B' -> "ATTACK";
    case 'X', 'Y' -> "SPECIAL";
    default -> "IDLE";
};
```

1.5 Loops

Type	Use Case
for	Known iteration count
for-each	Iterate collections/arrays
while	Unknown count, pre-check
do-while	At least one execution

```
// Spawn 10 Space Invaders
for (int i = 0; i < 10; i++) {
    enemies[i] = new Invader(i * 50, 100);
}

// Process all bullets
for (Bullet b : activeBullets) {
    b.move();
}

// Game loop
while (!gameOver) {
    update();
    render();
}

// Menu selection
do {
    choice = getInput();
} while (!isValidChoice(choice));
```

Loop Control:

break;	Exit loop immediately
continue;	Skip to next iteration
break label;	Exit labeled outer loop

1.6 Arrays

```
// Declaration and initialization
int[] scores = new int[5];
int[] levels = {1, 2, 3, 4, 5};
String[] bosses = {"Bowser", "Ganon", "Dracula"};

// 2D array (game grid)
char[][] tetrisBoard = new char[20][10];

// Array operations
int length = scores.length;
Arrays.sort(scores);
Arrays.fill(scores, 0);
int[] copy = Arrays.copyOf(scores, 10);
```

2 OOP & Inheritance

2.1 Class Structure

```
public class Enemy {
    // Fields (instance variables)
    private String name;
    private int health;
    private static int enemyCount = 0;

    // Constructor
    public Enemy(String name, int health) {
        this.name = name;
        this.health = health;
        enemyCount++;
    }
```

```
    // Overloaded constructor
    public Enemy(String name) {
        this(name, 100); // Constructor chaining
    }

    // Default constructor
    public Enemy() {
        this("Goomba", 50);
    }

    // Instance method
    public void takeDamage(int damage) {
        health -= damage;
        if (health <= 0) die();
    }

    // Static method
    public static int getEnemyCount() {
        return enemyCount;
    }

    // Getters and Setters
    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
}
```

2.2 Access Modifiers

Modifier	Class	Pkg	Sub	World
public	✓	✓	✓	✓
protected	✓	✓	✓	
(default)	✓	✓		
private	✓			

2.3 Inheritance

```
public class Character {
    protected int x, y;
    protected int health;

    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
}

public class Player extends Character {
    private int score;

    public Player(int startX, int startY) {
        super(); // Call parent constructor
        this.x = startX;
        this.y = startY;
        this.health = 100;
    }

    @Override
    public void move(int dx, int dy) {
        super.move(dx, dy); // Call parent method
        checkCollision();
    }
}
```

2.4 Abstract Classes

```
public abstract class PowerUp {
    protected int duration;

    // Abstract method (must be implemented)
    public abstract void activate(Player p);

    // Concrete method
    public boolean isExpired() {
        return duration <= 0;
    }
}

public class StarPower extends PowerUp {
    @Override
    public void activate(Player p) {
        p.setInvincible(true);
        duration = 10;
    }
}
```

2.5 Interfaces

```
public interface Collidable {
    // Constants (public static final)
    int MAX_COLLISION_CHECKS = 100;

    // Abstract methods
    boolean checkCollision(GameObject other);
    void onCollision(GameObject other);

    // Default method (Java 8+)
    default boolean isActive() {
        return true;
    }

    // Static method
    static double distance(Collidable a,
        Collidable b) {
        // Calculate distance...
        return 0.0;
    }
}

public class Bullet implements Collidable,
    Drawable {
    @Override
    public boolean checkCollision(GameObject other
        ) {
        // Implementation
    }

    @Override
    public void onCollision(GameObject other) {
        // Implementation
    }
}
```

2.6 Records (Java 16+)

```
// Immutable data class
public record HighScore(String player, int score,
                        String game) {
    // Compact constructor for validation
    public HighScore {
        if (score < 0)
            throw new IllegalArgumentException();
    }
}

// Usage
var record = new HighScore("AAA", 999999, "Galaga"
    );
String name = record.player(); // Auto-generated
    getter
```

2.7 Enums

```
public enum Direction {
    UP(0, -1), DOWN(0, 1),
    LEFT(-1, 0), RIGHT(1, 0);
```

```
    private final int dx, dy;

    Direction(int dx, int dy) {
        this.dx = dx;
        this.dy = dy;
    }

    public int getDx() { return dx; }
    public int getDy() { return dy; }
}

// Usage
Direction dir = Direction.UP;
player.move(dir.getDx(), dir.getDy());
```

2.8 Key OOP Keywords

Keyword	Purpose
this	Reference to current object
super	Reference to parent class
final	Prevent override/inheritance
static	Class-level member
abstract	No implementation
instanceof	Type checking

```
// instanceof with pattern matching (Java 16+)
if (entity instanceof Enemy e) {
    e.takeDamage(10); // e is auto-cast
}
```

3 Exceptions

3.1 Exception Hierarchy

Type	Description
Throwable	Base class for all
Error	Serious problems (don't catch)
Exception	Recoverable problems
RuntimeException	Unchecked exceptions

3.2 Common Exceptions

Exception	Cause
NullPointerException	Null reference access
ArrayIndexOutOfBounds	Invalid array index
IllegalArgumentException	Bad method argument
NumberFormatException	Invalid number parsing
IOException	I/O operation failure
FileNotFoundException	File doesn't exist
ClassCastException	Invalid type cast

3.3 Try-Catch-Finally

```
public void loadGame(String filename) {
    try {
        SaveFile save = readFile(filename);
        player.loadState(save);
    } catch (FileNotFoundException e) {
        System.out.println("Save not found!");
        startNewGame();
    } catch (IOException e) {
        System.out.println("Error: " + e.
            getMessage());
    } finally {
        // Always executes
        showMainMenu();
    }
}
```

3.4 Try-With-Resources

```
// Auto-closes resources implementing
    AutoCloseable
try (BufferedReader reader =
        new BufferedReader(new FileReader(file));
     BufferedWriter writer =
        new BufferedWriter(new FileWriter(out))) {

    String highScore = reader.readLine();
    writer.write("Score: " + highScore);

} catch (IOException e) {
    e.printStackTrace();
}
```

3.5 Throwing Exceptions

```
public void setLives(int lives) {
    if (lives < 0) {
        throw new IllegalArgumentException(
            "Lives cannot be negative");
    }
    this.lives = lives;
}

// Method that throws checked exception
public void saveGame(String file) throws
    IOException {
    // ... file operations
}
```

3.6 Custom Exceptions

```
public class GameOverException extends Exception {
    private final int finalScore;

    public GameOverException(String msg, int score
        ) {
        super(msg);
        this.finalScore = score;
    }

    public int getFinalScore() {
        return finalScore;
    }
}

// Usage
if (lives <= 0) {
    throw new GameOverException("No lives left!",
        score);
}
```

3.7 Multi-Catch & Rethrow

```
try {
    loadLevel(levelNumber);
} catch (FileNotFoundException | ParseException e)
    {
    // Handle multiple exceptions
    logger.error("Load failed", e);
    throw new GameException("Level load failed", e
        );
}
```

4 Collections Framework

4.1 Collection Interfaces

Interface	Description
`Collection`	Root interface
`List`	Ordered, allows duplicates
`Set`	No duplicates
`Queue`	FIFO ordering
`Deque`	Double-ended queue
`Map`	Key-value pairs

4.2 Common Implementations

Interface	Class	Notes
`List`	`ArrayList`	Fast random access
	`LinkedList`	Fast insert/delete
`Set`	`HashSet`	No order
	`TreeSet`	Sorted
	`LinkedHashSet`	Insertion order
`Map`	`HashMap`	No order
	`TreeMap`	Sorted by key
	`LinkedHashMap`	Insertion order
`Queue`	`PriorityQueue`	Heap-based
	`ArrayDeque`	Stack/queue

4.3 List Operations

```
// Creation
List<Enemy> enemies = new ArrayList<>();
List<String> items = List.of("Sword", "Shield");
    // immutable

// Adding elements
enemies.add(new Goomba());
enemies.add(0, new Koopa()); // Insert at index
enemies.addAll(moreEnemies);

// Accessing elements
Enemy first = enemies.get(0);
int size = enemies.size();
boolean empty = enemies.isEmpty();

// Searching
boolean hasGoomba = enemies.contains(goomba);
int index = enemies.indexOf(goomba);

// Removing
enemies.remove(0);          // By index
enemies.remove(goomba);     // By object
enemies.clear();            // Remove all

// Iterating
for (Enemy e : enemies) { e.update(); }
enemies.forEach(e -> e.update());
```

4.4 Set Operations

```
Set<String> collected = new HashSet<>();
collected.add("Star");
collected.add("Mushroom");
collected.add("Star"); // Ignored (duplicate)

// Set operations
Set<String> bonus = Set.of("Flower", "Star");
collected.retainAll(bonus); // Intersection
collected.addAll(bonus);    // Union
collected.removeAll(bonus); // Difference
```

4.5 Map Operations

```
Map<String, Integer> highScores = new HashMap<>();

// Adding entries
highScores.put("ACE", 999999);
highScores.putIfAbsent("NEW", 0);

// Accessing
int score = highScores.get("ACE");
int safe = highScores.getOrDefault("ZZZ", 0);
boolean hasPlayer = highScores.containsKey("ACE");

// Updating
highScores.replace("ACE", 1000000);
highScores.merge("ACE", 100, Integer::sum);
highScores.compute("ACE", (k, v) -> v + 100);

// Iterating
for (Map.Entry<String, Integer> e :
        highScores.entrySet()) {
    System.out.println(e.getKey() + ": " + e.
        getValue());
}

highScores.forEach((name, sc) ->
    System.out.println(name + ": " + sc));
```

4.6 Queue & Deque Operations

```
Queue<Command> inputQueue = new LinkedList<>();
inputQueue.offer(jumpCommand);  // Add to tail
Command next = inputQueue.poll(); // Remove head
Command peek = inputQueue.peek(); // View head

Deque<State> undoStack = new ArrayDeque<>();
undoStack.push(currentState);   // Stack push
State prev = undoStack.pop();   // Stack pop
```

4.7 Utility Methods (Collections class)

```
Collections.sort(scores);
Collections.reverse(enemies);
Collections.shuffle(deck);
int max = Collections.max(scores);
int min = Collections.min(scores);
Collections.fill(list, defaultValue);
Collections.frequency(list, item);
List<T> sync = Collections.synchronizedList(list);
List<T> immutable = Collections.unmodifiableList(
    list);
```

5 Random & Math

5.1 Math Class Methods

Method	Description
`Math.abs(x)`	Absolute value
`Math.max(a, b)`	Larger of two values
`Math.min(a, b)`	Smaller of two values
`Math.pow(base, exp)`	Exponentiation
`Math.sqrt(x)`	Square root
`Math.cbrt(x)`	Cube root
`Math.round(x)`	Round to nearest
`Math.floor(x)`	Round down
`Math.ceil(x)`	Round up
`Math.log(x)`	Natural logarithm
`Math.log10(x)`	Base-10 logarithm
`Math.sin/cos/tan(x)`	Trig functions (radians)
`Math.toRadians(deg)`	Degrees to radians
`Math.toDegrees(rad)`	Radians to degrees
`Math.random()`	Random [0.0, 1.0)
`Math.PI`	Pi constant
`Math.E`	Euler's number

```
// Calculate distance between game objects
double distance = Math.sqrt(
    Math.pow(enemy.x - player.x, 2) +
    Math.pow(enemy.y - player.y, 2)
);

// Clamp health between 0 and max
health = Math.max(0, Math.min(health, maxHealth));

// Angle to target (for enemy AI)
double angle = Math.atan2(
    player.y - enemy.y,
    player.x - enemy.x
);
```

5.2 Random Class

```
import java.util.Random;

Random rand = new Random();        // Random seed
Random seeded = new Random(42L); // Fixed seed

// Random numbers
int damage = rand.nextInt(10);       // 0-9
int diceRoll = rand.nextInt(6) + 1; // 1-6
int loot = rand.nextInt(50, 101);    // 50-100 (
    Java 17+)
double chance = rand.nextDouble();  // 0.0-1.0
boolean crit = rand.nextBoolean();  // true/false

// Gaussian distribution
double stat = rand.nextGaussian() * 10 + 50; //
    mean=50, sd=10
```

5.3 Common Random Patterns

```
// Random spawn position
int spawnX = rand.nextInt(screenWidth);
int spawnY = rand.nextInt(screenHeight);

// Percentage chance (30% critical hit)
if (rand.nextDouble() < 0.30) {
    damage *= 2;
}

// Random element from array
Enemy[] types = {new Goomba(), new Koopa(), new
    Boo()};
Enemy spawned = types[rand.nextInt(types.length)];

// Shuffle (e.g., deck of cards)
List<Card> deck = new ArrayList<>(allCards);
Collections.shuffle(deck);

// Weighted random selection
int roll = rand.nextInt(100);
if (roll < 60) return "COMMON";
else if (roll < 90) return "RARE";
else return "LEGENDARY";
```

6 Streams & Lambdas

6.1 Lambda Expressions

Lambda Syntax	Equivalent
`x -> x * 2`	Single param, expression
`(x, y) -> x + y`	Multiple params
`() -> 42`	No parameters
`(x) -> { return x * 2; }`	Block with return

6.2 Functional Interfaces

Interface	Method	Signature
`Predicate<T>`	`test`	T → boolean
`Function<T,R>`	`apply`	T → R
`Consumer<T>`	`accept`	T → void
`Supplier<T>`	`get`	() → T
`Comparator<T>`	`compare`	(T,T) → int
`UnaryOperator<T>`	`apply`	T → T
`BiFunction<T,U,R>`	`apply`	(T,U) → R

```
Predicate<Enemy> isAlive = e -> e.getHealth() > 0;
Function<Enemy, Integer> getScore = Enemy::
    getPoints;
Consumer<Enemy> damage = e -> e.takeDamage(10);
Supplier<Enemy> spawner = Goomba::new;
Comparator<Enemy> byHealth =
    Comparator.comparing(Enemy::getHealth);
```

6.3 Method References

Type	Syntax
Static method	`ClassName::staticMethod`
Instance method	`object::instanceMethod`
Arbitrary instance	`ClassName::instanceMethod`
Constructor	`ClassName::new`

```
enemies.forEach(System.out::println);
enemies.sort(Comparator.comparing(Enemy::getHealth
    ));
List<Goomba> goombas =
    positions.stream().map(Goomba::new).toList();
```

6.4 Creating Streams

```
// From collections
Stream<Enemy> stream = enemies.stream();
Stream<Enemy> parallel = enemies.parallelStream();

// From arrays
Stream<String> arr = Arrays.stream(items);

// Factory methods
Stream<String> of = Stream.of("A", "B", "C");
Stream<Integer> range = IntStream.range(1, 11).
    boxed();
```

```
Stream<Double> gen = Stream.generate(Math::random)
    .limit(10);
Stream<Integer> iter = Stream.iterate(0, n -> n +
    1).limit(10);
```

6.5 Intermediate Operations

Operation	Description
`filter(pred)`	Keep matching elements
`map(func)`	Transform elements
`flatMap(func)`	Flatten nested streams
`distinct()`	Remove duplicates
`sorted()`	Natural order sort
`sorted(comp)`	Custom comparator sort
`limit(n)`	First n elements
`skip(n)`	Skip first n elements
`peek(consumer)`	Debug/side effects

6.6 Terminal Operations

Operation	Returns
`forEach(consumer)`	void
`toList()`	List¡T¿
`toArray()`	Object[]
`collect(collector)`	Varies
`count()`	long
`reduce(identity, op)`	T
`min(comp) / max(comp)`	Optional¡T¿
`findFirst() / findAny()`	Optional¡T¿
`anyMatch(pred)`	boolean
`allMatch(pred)`	boolean
`noneMatch(pred)`	boolean

6.7 Stream Examples

```
// Get total score from defeated enemies
int totalScore = enemies.stream()
    .filter(e -> e.getHealth() <= 0)
    .mapToInt(Enemy::getPoints)
    .sum();

// Find strongest enemy
Optional<Enemy> boss = enemies.stream()
    .max(Comparator.comparing(Enemy::getHealth));

// Get names of top 3 players
List<String> top3 = players.stream()
    .sorted(Comparator.comparing(Player::getScore)
        .reversed())
    .limit(3)
    .map(Player::getName)
    .toList();

// Group enemies by type
Map<String, List<Enemy>> byType = enemies.stream()
    .collect(Collectors.groupingBy(Enemy::getType)
        );

// Partition into alive/dead
Map<Boolean, List<Enemy>> partition = enemies.
    stream()
    .collect(Collectors.partitioningBy(
        e -> e.getHealth() > 0));

// Join names with comma
String roster = players.stream()
    .map(Player::getName)
    .collect(Collectors.joining(", "));
```

6.8 Collectors Summary

```
Collectors.toList()
Collectors.toSet()
Collectors.toMap(keyFunc, valueFunc)
Collectors.joining(", ")
Collectors.groupingBy(classifier)
Collectors.partitioningBy(predicate)
Collectors.counting()
Collectors.summingInt(mapper)
Collectors.averagingDouble(mapper)
Collectors.maxBy(comparator)
Collectors.summarizingInt(mapper)
```

7 Basic Swing (GUI)

7.1 Core Components

Component	Purpose
`JFrame`	Main application window
`JPanel`	Container/drawing surface
`JButton`	Clickable button
`JLabel`	Display text/images
`JTextField`	Single-line text input
`JTextArea`	Multi-line text input
`JCheckBox`	Toggle option
`JRadioButton`	Exclusive option
`JComboBox`	Drop-down selection
`JList`	Scrollable list
`JSlider`	Numeric slider
`JMenuBar`	Menu system

7.2 Basic Window Setup

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

public class PacManGame extends JFrame {
    public PacManGame() {
        setTitle("Pac-Man");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.
            EXIT_ON_CLOSE);
        setLocationRelativeTo(null); // Center
        setResizable(false);

        GamePanel panel = new GamePanel();
        add(panel);

        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() ->
            new PacManGame()
        );
    }
}
```

7.3 Layout Managers

Layout	Behavior
`FlowLayout`	Left-to-right, wraps
`BorderLayout`	5 regions (N/S/E/W/Center)
`GridLayout`	Equal-sized grid cells
`BoxLayout`	Single row or column
`GridBagLayout`	Flexible grid (complex)
`null`	Manual positioning

```
JPanel controlPanel = new JPanel(new FlowLayout())
    ;
JPanel mainPanel = new JPanel(new BorderLayout());
JPanel gridPanel = new JPanel(new GridLayout(3, 3)
    );

mainPanel.add(scoreLabel, BorderLayout.NORTH);
mainPanel.add(gameCanvas, BorderLayout.CENTER);
mainPanel.add(controlPanel, BorderLayout.SOUTH);
```

7.4 Event Handling

```
// Button click (lambda)
JButton startBtn = new JButton("START GAME");
startBtn.addActionListener(e -> startGame());

// Keyboard input
addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {
        switch(e.getKeyCode()) {
            case KeyEvent.VK_UP -> player.moveUp()
                ;
            case KeyEvent.VK_DOWN -> player.
                moveDown();
            case KeyEvent.VK_LEFT -> player.
                moveLeft();
            case KeyEvent.VK_RIGHT -> player.
                moveRight();
            case KeyEvent.VK_SPACE -> player.shoot
                ();
        }
    }
});

// Key bindings (better for games)
InputMap im = panel.getInputMap(
    JComponent.WHEN_IN_FOCUSED_WINDOW);
ActionMap am = panel.getActionMap();
im.put(KeyStroke.getKeyStroke("UP"), "moveUp");
am.put("moveUp", new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        player.moveUp();
    }
});

// Mouse input
addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        handleClick(x, y);
    }
});
```

7.5 Custom Painting

```
class GamePanel extends JPanel {
    private Player pacman;
    private List<Ghost> ghosts;
    private int[][] maze;

    public GamePanel() {
        setBackground(Color.BLACK);
        setPreferredSize(new Dimension(800, 600));
        setFocusable(true);

        // Game loop timer
        Timer timer = new Timer(16, e -> {
            update();  // Game logic
            repaint(); // Trigger paint
        });
        timer.start();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;

        // Enable antialiasing
        g2d.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

        drawMaze(g2d);
        drawPacman(g2d);
        drawGhosts(g2d);
        drawScore(g2d);
    }

    private void drawPacman(Graphics2D g) {
        g.setColor(Color.YELLOW);
        g.fillArc(pacman.x, pacman.y, 30, 30,
                  45, 270); // Pac-Man shape
    }

    private void drawScore(Graphics2D g) {
        g.setColor(Color.WHITE);
        g.setFont(new Font("Arial", Font.BOLD, 20)
            );
        g.drawString("Score: " + score, 10, 25);
    }
}
```

7.6 Graphics2D Methods

Method	Description
`setColor(Color)`	Set drawing color
`fillRect(x,y,w,h)`	Filled rectangle
`drawRect(x,y,w,h)`	Rectangle outline
`fillOval(x,y,w,h)`	Filled ellipse
`drawOval(x,y,w,h)`	Ellipse outline
`fillArc(...)`	Filled arc/pie slice
`drawLine(x1,y1,x2,y2)`	Line segment
`drawString(str,x,y)`	Text at position
`drawImage(img,x,y,obs)`	Render image
`setFont(Font)`	Set text font
`setStroke(Stroke)`	Line thickness/style
`rotate/translate/scale`	Transformations

7.7 Loading and Drawing Images

```
// Load image
BufferedImage sprite = ImageIO.read(
    getClass().getResource("/sprites/mario.png"));

// Draw image
g.drawImage(sprite, x, y, null);

// Draw scaled image
g.drawImage(sprite, x, y, width, height, null);

// Draw portion of sprite sheet
g.drawImage(sprite,
    destX, destY, destX + w, destY + h,  // Dest
    srcX, srcY, srcX + w, srcY + h,      //
        Source
    null);
```

7.8 Dialog Boxes

```
// Message dialog
JOptionPane.showMessageDialog(frame,
```

```
    "Game Over! Score: " + score);

// Yes/No dialog
int choice = JOptionPane.showConfirmDialog(frame,
    "Play again?", "Game Over",
    JOptionPane.YES_NO_OPTION);
if (choice == JOptionPane.YES_OPTION) {
    restartGame();
}

// Input dialog
String name = JOptionPane.showInputDialog(frame,
    "Enter your name for high score:");

// Custom option dialog
String[] options = {"Easy", "Medium", "Hard"};
int difficulty = JOptionPane.showOptionDialog(
    frame,
    "Select difficulty:", "New Game",
    JOptionPane.DEFAULT_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null, options, options[1]);
```

7.9 Timer for Animation/Game Loop

```
// Swing Timer (runs on EDT)
Timer gameTimer = new Timer(16, e -> { // ~60 FPS
    updateGame();
    gamePanel.repaint();
});
gameTimer.start();
gameTimer.stop();

// One-shot timer
Timer delay = new Timer(3000, e -> showBoss());
delay.setRepeats(false);
delay.start();
```

Created for RCTC Java Programming Course