Lab 08 Concepts

This review sheet covers key topics related to graphical applications and game development in Java. The topics include rendering graphics, handling animations and input, organizing code, and performing network operations. Whether you're developing a game or a rich interactive application, these concepts form the foundation of your work in Java.

---

1. Painting to the Screen

Key Concepts

- **Painting & Repainting:** In Java GUI applications (especially with Swing), painting is usually done by overriding the `paintComponent(Graphics g)` method of a `JPanel` or similar component.
- **Double-buffering:** Swing automatically provides double-buffering to help reduce flickering during animations.
- **Graphics Context:** The `Graphics` object provides methods to draw shapes, text, and images.

Sample Code

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

public class MyPanel extends JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g); // Clears the panel
        // Custom painting code:
        g.setColor(Color.BLUE);
        g.fillRect(50, 50, 200, 100); // Draw a blue rectangle
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Painting Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new MyPanel());
        frame.setSize(300, 250);
        frame.setVisible(true);
    }
}
```

---

2. Using Timers

Key Concepts

- **Swing Timer:** `javax.swing.Timer` is ideal for GUI applications as it ensures that action events are dispatched on the Event Dispatch Thread.
- **Scheduling:** Timers are used to perform actions at regular intervals—essential for animations or periodic updates.
- **Alternative:** For non-GUI tasks, consider using `java.util.Timer` or scheduled executor services.

Sample Code

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

public class TimerExample {
    public static void main(String[] args) {
        int delay = 1000; // milliseconds (1 second)
        ActionListener taskPerformer = new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                // Code to execute periodically
                System.out.println("Timer ticked!");
            }
        };
        new Timer(delay, taskPerformer).start();

        // Keep application running (e.g., show a GUI)
        JOptionPane.showMessageDialog(null, "Close to stop timer.");
    }
}
```

---

3. Animating Translational Motion

Key Concepts

- **Updating Position:** Change the x and y coordinates of an object over time to create the illusion of movement.
- **Repainting:** After updating positions, call `repaint()` to refresh the display.
- **Frame Rate:** Use a timer to control how often the position is updated (e.g., 60 updates per second for smooth motion).

Sample Code

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

public class AnimationPanel extends JPanel {
    private int x = 0, y = 50; // Starting position

    public AnimationPanel() {
        Timer timer = new Timer(16, new ActionListener() { // Roughly 60fps
            public void actionPerformed(ActionEvent e) {
                x += 2; // Move object horizontally
                if (x > getWidth()) {
                    x = 0; // Reset position when reaching the edge
                }
                repaint();
            }
        });
        timer.start();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillOval(x, y, 30, 30); // Draw a moving circle
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Translational Motion Animation");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new AnimationPanel());
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```

---

4. Keyboard Input Processing

Key Concepts

- **KeyListener Interface:** Implement methods such as `keyPressed`, `keyReleased`, and `keyTyped` to handle keyboard events.
- **Focus:** The component receiving the events must be focusable (using

`setFocusable(true)`) and have focus.

- **Key Bindings:** As an alternative to `KeyListener`, Swing's key bindings offer a more flexible and focus-independent approach.

Sample Code (KeyListener)

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

public class KeyInputExample extends JPanel implements KeyListener {
    public KeyInputExample() {
        setFocusable(true);
        addKeyListener(this);
    }

    @Override
    public void keyPressed(KeyEvent e) {
        System.out.println("Key Pressed: " + e.getKeyCode());
    }

    @Override
    public void keyReleased(KeyEvent e) {
        System.out.println("Key Released: " + e.getKeyCode());
    }

    @Override
    public void keyTyped(KeyEvent e) {
        // Not used for movement typically
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Keyboard Input Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new KeyInputExample());
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

---

5. Breaking Large Codebases into Separate Files

Key Concepts

- **Modularity:** Organize your code by separating classes and interfaces into different files.

- **Packages:** Group related classes in packages to manage namespace and enhance code readability.
- **Separation of Concerns:** Divide your application logic, UI code, data models, and utilities into different files.

Best Practices

- **Naming Conventions:** Use clear and descriptive names for classes and packages.
- **Directory Structure:** Follow a directory structure that mirrors your package structure (e.g., `com.example.game`).
- **Use IDEs:** Leverage IDE features (like refactoring and project organization) to manage large codebases.

---

6. Making HTTP Requests

Key Concepts

- **HttpURLConnection:** A standard way to make HTTP requests in Java.
- **Java 11 HttpClient:** Offers a modern API for handling synchronous and asynchronous HTTP calls.
- **Error Handling:** Always check response codes and manage exceptions.

Sample Code (Using HttpURLConnection)

```
import java.net.*;
import java.io.*;

public class HttpRequestExample {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://api.example.com/data");
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");

            int responseCode = connection.getResponseCode();
            System.out.println("Response Code : " + responseCode);

            BufferedReader in = new BufferedReader(
                                    new InputStreamReader(connection.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
```

```
            in.close();

            System.out.println("Response: " + response.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Sample Code (Using Java 11 HttpClient)

```
import java.net.http.*;
import java.net.URI;
import java.io.IOException;

public class HttpClientExample {
    public static void main(String[] args) throws IOException, InterruptedException {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("https://api.example.com/data"))
            .build();

        HttpResponse<String> response =
            client.send(request, HttpResponse.BodyHandlers.ofString());

        System.out.println("Status Code: " + response.statusCode());
        System.out.println("Body: " + response.body());
    }
}
```

---

7. Managing Event Loops in Games

Key Concepts

- **Game Loop Fundamentals:** A loop that continually updates game state, processes input, and renders graphics.
- **Fixed vs. Variable Time Steps:** Fixed time steps can simplify physics calculations; variable time steps adapt to performance variations.
- **Timers vs. Threads:** Swing's Timer is suitable for simple games, while more complex games might use a dedicated game loop thread.

Sample Pseudocode (Timer-based Game Loop)

```
// Inside your game panel or main loop class:
Timer gameTimer = new Timer(16, new ActionListener() { // Approximately 60fps
```

```
    public void actionPerformed(ActionEvent e) {
        updateGameState(); // Process input, update positions, check collisions, etc.
        repaint();         // Render the updated state
    }
});
gameTimer.start();
```

---

8. Loading Sprites

Key Concepts

- **ImageIO:** Use `ImageIO.read(File file)` or `ImageIO.read(InputStream)` to load images.
- **BufferedImage:** Represents an image with an accessible buffer of image data.
- **Resource Management:** Load images once (preferably during initialization) and reuse them to improve performance.

Sample Code

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

public class SpriteLoader {
    public BufferedImage loadSprite(String path) {
        try {
            return ImageIO.read(new File(path));
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    // Usage:
    public static void main(String[] args) {
        SpriteLoader loader = new SpriteLoader();
        BufferedImage sprite = loader.loadSprite("resources/sprite.png");
        if (sprite != null) {
            System.out.println("Sprite loaded successfully!");
        }
    }
}
```

---

9. Animating Sprites

Key Concepts

- **Sprite Sheets:** A single image containing multiple frames of animation.
- **Frame Cycling:** Change the sub-image (frame) drawn on each update to create an animation.
- **Timing:** Use a timer or game loop to control the frame rate of the animation.

Sample Code (Animating a Sprite from a Sprite Sheet)

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

public class SpriteAnimationPanel extends JPanel {
    private BufferedImage spriteSheet;
    private int frameWidth = 64, frameHeight = 64;
    private int currentFrame = 0;
    private int totalFrames = 4; // Assuming 4 frames in the sprite sheet

    public SpriteAnimationPanel() {
        // Load your sprite sheet (error handling omitted for brevity)
        try {
            spriteSheet = ImageIO.read(new File("resources/spriteSheet.png"));
        } catch (Exception e) {
            e.printStackTrace();
        }

        Timer animationTimer = new Timer(100, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                currentFrame = (currentFrame + 1) % totalFrames;
                repaint();
            }
        });
        animationTimer.start();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        // Calculate source rectangle for current frame
        int sx = currentFrame * frameWidth;
        int sy = 0;
        g.drawImage(spriteSheet,
```

```
                    50, 50, 50 + frameWidth, 50 + frameHeight, // Destination rectangle
                    sx, sy, sx + frameWidth, sy + frameHeight, // Source rectangle
                    null);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Sprite Animation");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new SpriteAnimationPanel());
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Conclusion

- **Render graphics** using Swing's painting methods.
- **Control timing** with Swing timers for animations and updates.
- **Animate translational motion** by updating object positions.
- **Process keyboard inputs** via KeyListeners or key bindings.
- **Organize large codebases** with modular design and proper package structure.
- **Perform network operations** using HTTP requests.
- **Manage game loops** to handle events and render frames.
- **Load and animate sprites** from image files and sprite sheets.