OOPS PROJECT, GROUP 53

Smart Home Automation System

Project Documentation

1. Introduction

A *Smart Home Automation System* is designed to enhance comfort, energy efficiency, and security by allowing users to monitor and control home devices (lights, fans, air conditioners, security systems, etc.) remotely or automatically. This project implements a user-friendly, role-based smart home system with automation, scheduling, and device customization features that are accessible via a graphical interface.

2. Objectives

- Remote and local control: Enable users to control and monitor home devices from anywhere via a desktop application.
- Role-based access: Implement authentication and authorization for Admin and Regular Users, restricting sensitive operations to admins.
- Automation: Support device automation (e.g., motion-activated lights, auto temperature adjustment in ACs).
- Scheduling: Allow users to schedule device operations (turn on/off, set temperature, etc.).

- Customization: Support device-specific features (e.g., light color selection, AC modes).
- Security: Integrate a security system with an alarm and motion detection.
- Extensibility: Design the system to easily add new devices and features.

3. System Architecture

3.1 Components

- User Interface (SmartHomeGUI): Java Swing-based GUI for user interaction.
- Core Logic (SmartHomeSystem): Handles authentication, device management, scheduling, and automation.
- Devices: Abstract base class with concrete implementations for Light, Fan, AirConditioner, and SecuritySystem.
- Users: Admin and Regular User classes with role-based permissions.
- Exceptions: For handling authentication and device errors.
- Interfaces: For device capabilities (Switchable, Dimmable).
- Scheduling: Allows users to automate device actions at specific times/days.

3.2 Device Features

Device	**Features**
Light	On/Off, Brightness control, Motion activation, Color selection (White, Warm, Blue, Red)

Fan	On/Off, Speed control (no oscillation/auto-temp adjust)
Air Conditioner	On/Off, Temperature (16–30°C), Modes, Energy Saving, Auto Temperature Adjust
SecuritySystem	On/Off, Alarm, Mode (Home/Away/Disarmed), Motion detection, Event logs

4. User Roles and Permissions

- Admin
    - Add/remove devices and users
    - View system logs
    - Full device control and scheduling
- Regular User
    - Control devices and schedule tasks
    - Cannot add/remove devices or users
    - Cannot view logs

Authentication is required for all users. Permissions are enforced throughout the application.

5. Automation & Scheduling

- Motion-activated lights: Lights in a room turn on automatically when motion is detected.
- AC Auto Temperature Adjust: The AC can automatically set the temperature based on the time of day for comfort and efficiency.
- Scheduling: Users can schedule device actions (e.g., turn on the light at 7 pm every day).

6. User Interface

- Login Screen: Secure login for all users.
- Main Dashboard: Device list, system controls, user info.
- Device Control Panel: Device-specific controls (brightness, color, speed, temperature, etc.).
- Device Scheduling: Add/view/remove scheduled tasks.
- Admin Features: Add/remove users/devices, view logs.

All controls are permission-aware and visually intuitive.

7. Key Classes and Structure

```
text

src/
└─ com/
  └─ smarthome/
    ├─ SmartHomeApp.java
    ├─ gui/
    │   └─ SmartHomeGUI.java
    ├─ system/
    │   └─ SmartHomeSystem.java
```

```
├─ models/
│   ├─ Device.java
│   ├─ Light.java
│   ├─ Fan.java
│   ├─ AirConditioner.java
│   ├─ SecuritySystem.java
│   ├─ ScheduledTask.java
│   ├─ User.java
│   └─ AdminUser.java
├─ interfaces/
│   ├─ Switchable.java
│   └─ Dimmable.java
└─ exceptions/
   ├─ AuthenticationException.java
   └─ DeviceNotFoundException.java
```

8. Example Use Cases

- Login: User enters credentials; system grants access based on role.
- Device Control: User selects a device, adjusts settings (e.g., light brightness, AC temperature).
- Automation: Light turns on automatically when user enters a room.
- Scheduling: User sets AC to turn on at 6pm on weekdays.
- Admin Management: Admin adds a new user or device.

9. Extensibility

The system is designed for easy expansion:

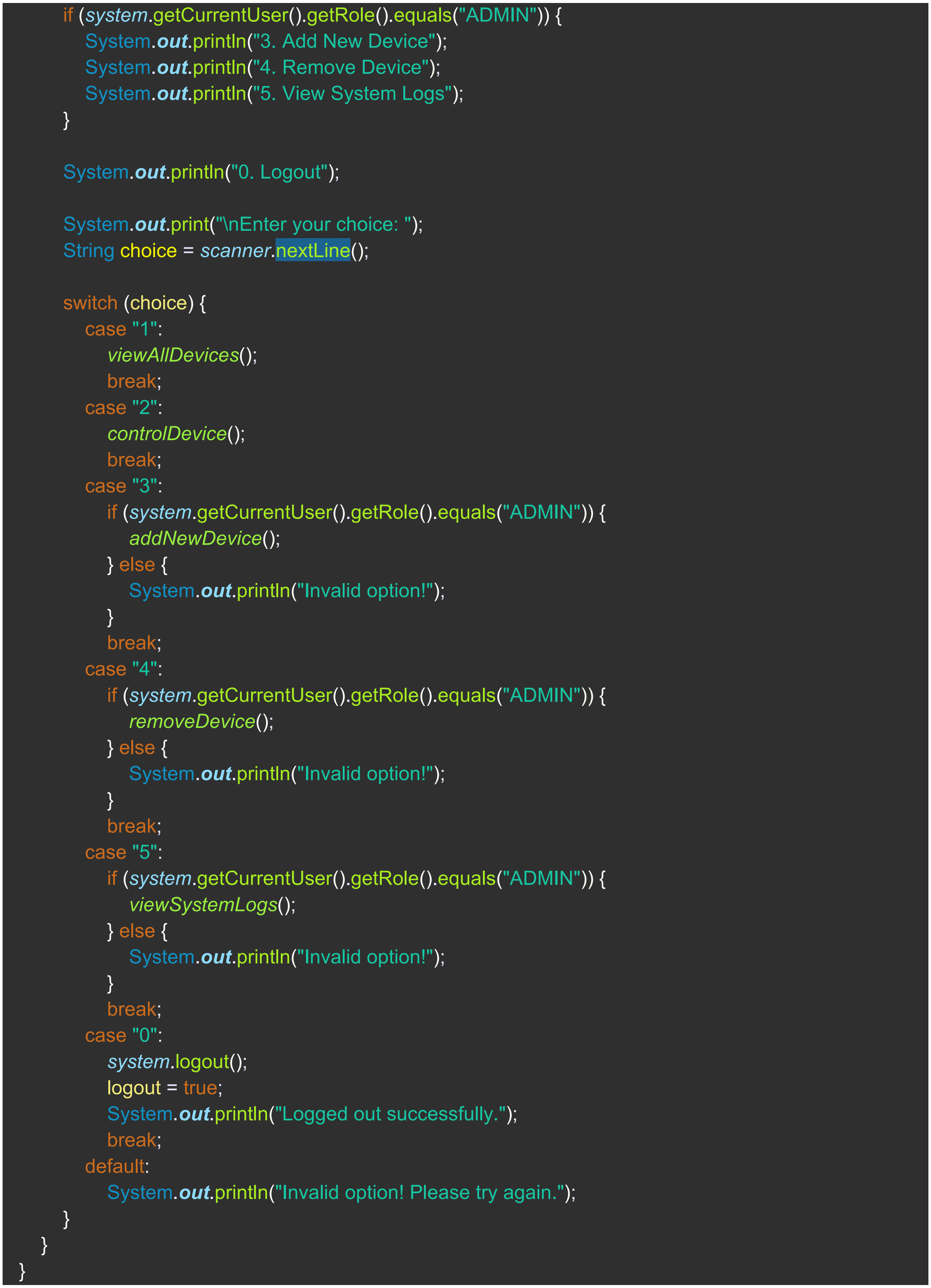
- New device types can be added by extending the Device class.
- New automation rules can be implemented in SmartHomeSystem.
- GUI updates automatically reflect new device features.

10. Conclusion

This Smart Home Automation System demonstrates a robust, extensible, and user–friendly approach to home automation. Combining role–based security, automation, scheduling, and device customization provides a comprehensive solution for modern smart homes. The modular design ensures future enhancements can be integrated with minimal effort.

THE CODE:-
SMARTPHONEAPP CLASS

```
package smarthome;
import smarthome.system.SmartHomeSystem;
import smarthome.models.*;
import smarthome.exceptions.*;
import smarthome.SmartHomeGUI;
import java.util.Scanner;
public class SmartHomeApp {
  private static SmartHomeSystem system;
  private static Scanner scanner;

  public static void main(String[] args) {
     // Initialize the system
     system = SmartHomeSystem.getInstance();
     scanner = new Scanner(System.in);

     // For console testing before GUI launch
     boolean useConsole = false;

     if (useConsole) {
        runConsoleInterface();
     } else {
        // Initialize default devices
        initializeDefaultDevices();
        // Launch the GUI
        SmartHomeGUI gui = new SmartHomeGUI(system);
        gui.launch();
    }
}

private static void runConsoleInterface() {
    System.out.println("Welcome to Smart Home System");
    System.out.println("----------------------------");

    boolean running = true;
    while (running) {
        System.out.println("\nPlease login to continue:");
        System.out.print("Username: ");
        String username = scanner.nextLine();
        System.out.print("Password: ");
        String password = scanner.nextLine();

        try {
            system.login(username, password);
            System.out.println("Login successful!");

            // Main menu after login
            showMainMenu();

        } catch (AuthenticationException e) {
            System.out.println("Login failed: " + e.getMessage());
        }

        System.out.print("\nDo you want to exit? (y/n): ");
        String exit = scanner.nextLine();
        if (exit.equalsIgnoreCase("y")) {
            running = false;
        }
    }

    System.out.println("Thank you for using Smart Home System. Goodbye!");
    scanner.close();
}

private static void showMainMenu() {
    boolean logout = false;

    while (!logout) {
        System.out.println("\n===== Main Menu =====");
        System.out.println("1. View All Devices");
        System.out.println("2. Control a Device");

        // Admin-only options
```



```java
        if (system.getCurrentUser().getRole().equals("ADMIN")) {
            System.out.println("3. Add New Device");
            System.out.println("4. Remove Device");
            System.out.println("5. View System Logs");
        }

        System.out.println("0. Logout");

        System.out.print("\nEnter your choice: ");
        String choice = scanner.nextLine();

        switch (choice) {
            case "1":
                viewAllDevices();
                break;
            case "2":
                controlDevice();
                break;
            case "3":
                if (system.getCurrentUser().getRole().equals("ADMIN")) {
                    addNewDevice();
                } else {
                    System.out.println("Invalid option!");
                }
                break;
            case "4":
                if (system.getCurrentUser().getRole().equals("ADMIN")) {
                    removeDevice();
                } else {
                    System.out.println("Invalid option!");
                }
                break;
            case "5":
                if (system.getCurrentUser().getRole().equals("ADMIN")) {
                    viewSystemLogs();
                } else {
                    System.out.println("Invalid option!");
                }
                break;
            case "0":
                system.logout();
                logout = true;
                System.out.println("Logged out successfully.");
                break;
            default:
                System.out.println("Invalid option! Please try again.");
        }
    }
}
```

```
private static void viewAllDevices() {
    System.out.println("\n===== All Devices =====");

    for (Device device : system.getAllDevices()) {
        System.out.println(device);
    }

    if (system.getAllDevices().isEmpty()) {
        System.out.println("No devices found.");
    }
}

private static void controlDevice() {
    // Implementation omitted for brevity
}

private static void addNewDevice() {
    System.out.println("\n===== Add New Device =====");
    System.out.println("Select device type:");
    System.out.println("1. Light");
    System.out.println("2. Fan");
    System.out.println("3. Air Conditioner");
    System.out.println("4. Security System");
    System.out.println("0. Cancel");

    System.out.print("\nEnter your choice: ");
    String choice = scanner.nextLine();

    if (choice.equals("0")) {
        return;
    }

    System.out.print("Enter device name: ");
    String name = scanner.nextLine();

    System.out.print("Enter device location: ");
    String location = scanner.nextLine();

    try {
        Device newDevice = null;

        switch (choice) {
            case "1":
                newDevice = new Light(name, location, system.getCurrentUser().getUsername());
                break;
            case "2":
                newDevice = new Fan(name, location, system.getCurrentUser().getUsername());
                break;
                case "3":
                    newDevice = new AirConditioner(name, location, system.getCurrentUser().getUsername());
                    break;
                case "4":
                    newDevice = new SecuritySystem(name, location, system.getCurrentUser().getUsername());
                    break;
                default:
                    System.out.println("Invalid device type!");
                    return;
            }

            system.addDevice(newDevice);
            System.out.println("Device added successfully: " + newDevice.getName());

        } catch (AuthenticationException e) {
            System.out.println("Error adding device: " + e.getMessage());
        }
    }

    private static void removeDevice() {
        // Implementation omitted for brevity
    }

    private static void viewSystemLogs() {
        // Implementation omitted for brevity
    }

    private static void initializeDefaultDevices() {
        try {
            // Login as admin to add devices
            system.login("admin", "admin123");

            // Add some default devices
            system.addDevice(new Light("Living Room Light", "Living Room", "admin"));
            system.addDevice(new Fan("Bedroom Fan", "Bedroom", "admin"));
            system.addDevice(new AirConditioner("Living Room AC", "Living Room", "admin"));
            system.addDevice(new SecuritySystem("Main Security System", "Entrance", "admin"));

            // Logout
            system.logout();
        } catch (AuthenticationException e) {
            System.err.println("Error during initialization: " + e.getMessage());
        }
    }
}
```

SMARTHOMEGUI:-

```
package smarthome;
import smarthome.system.SmartHomeSystem;
import smarthome.models.*;
import smarthome.exceptions.*;
import smarthome.interfaces.Switchable;
import smarthome.interfaces.Dimmable;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Date;
public class SmartHomeGUI {
  private SmartHomeSystem system;
  private JFrame mainFrame;
  private JPanel mainPanel;
  private JPanel devicePanel;
  private JPanel controlPanel;
  private Timer scheduleTimer;

  public SmartHomeGUI(SmartHomeSystem system) {
    this.system = system;
  }

  public void launch() {
    // Create the main frame
    mainFrame = new JFrame("Smart Home System");
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainFrame.setSize(1000, 600);
    mainFrame.setLayout(new BorderLayout());

    // Create the main panel
    mainPanel = new JPanel(new BorderLayout());

    // Create the login panel
    createLoginPanel();

    // Start a timer to check scheduled tasks every minute
    scheduleTimer = new Timer(60000, e -> system.executeScheduledTasks());
    scheduleTimer.start();

    // Show the frame
    mainFrame.setVisible(true);
  }

  private void createLoginPanel() {
    JPanel loginPanel = new JPanel();
    loginPanel.setLayout(new GridBagLayout());

    GridBagConstraints gbc = new GridBagConstraints();
```

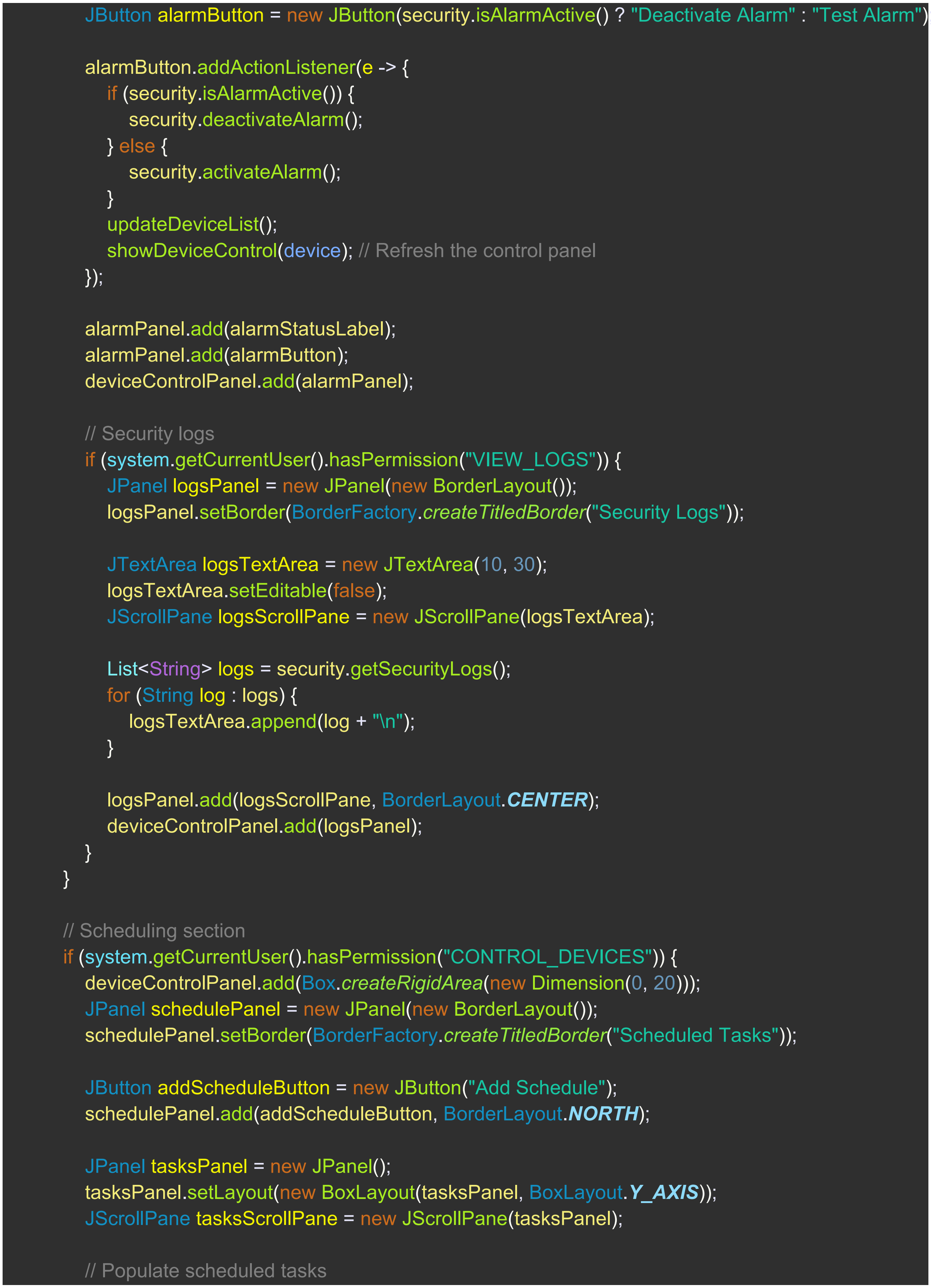
```java
gbc.insets = new Insets(5, 5, 5, 5);

JLabel titleLabel = new JLabel("Smart Home System");
titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.anchor = GridBagConstraints.CENTER;
loginPanel.add(titleLabel, gbc);

JLabel usernameLabel = new JLabel("Username:");
gbc.gridx = 0;
gbc.gridy = 1;
gbc.gridwidth = 1;
gbc.anchor = GridBagConstraints.EAST;
loginPanel.add(usernameLabel, gbc);

JTextField usernameField = new JTextField(15);
gbc.gridx = 1;
gbc.gridy = 1;
gbc.anchor = GridBagConstraints.WEST;
loginPanel.add(usernameField, gbc);

JLabel passwordLabel = new JLabel("Password:");
gbc.gridx = 0;
gbc.gridy = 2;
gbc.anchor = GridBagConstraints.EAST;
loginPanel.add(passwordLabel, gbc);

JPasswordField passwordField = new JPasswordField(15);
gbc.gridx = 1;
gbc.gridy = 2;
gbc.anchor = GridBagConstraints.WEST;
loginPanel.add(passwordField, gbc);

JButton loginButton = new JButton("Login");
gbc.gridx = 0;
gbc.gridy = 3;
gbc.gridwidth = 2;
gbc.anchor = GridBagConstraints.CENTER;
loginPanel.add(loginButton, gbc);

JLabel statusLabel = new JLabel(" ");
gbc.gridx = 0;
gbc.gridy = 4;
gbc.gridwidth = 2;
loginPanel.add(statusLabel, gbc);

loginButton.addActionListener(new ActionListener() {
```

```
            @Override
            public void actionPerformed(ActionEvent e) {
                String username = usernameField.getText();
                String password = new String(passwordField.getPassword());

                try {
                    system.login(username, password);
                    mainFrame.getContentPane().removeAll();
                    createMainPanel();
                    mainFrame.getContentPane().add(mainPanel);
                    mainFrame.revalidate();
                    mainFrame.repaint();
                } catch (AuthenticationException ex) {
                    statusLabel.setText("Login failed: " + ex.getMessage());
                    statusLabel.setForeground(Color.RED);
                }
            }
        });

        mainFrame.getContentPane().add(loginPanel);
    }

    private void createMainPanel() {
        mainPanel.removeAll();

        // Create the top panel with system controls
        JPanel topPanel = new JPanel(new BorderLayout());

        // User info and logout
        JPanel userPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        JLabel userLabel = new JLabel("Logged in as: " + system.getCurrentUser().getName() +
                                    " (" + system.getCurrentUser().getRole() + ")");
        JButton logoutButton = new JButton("Logout");
        userPanel.add(userLabel);
        userPanel.add(logoutButton);

        // System on/off toggle
        JPanel systemControlPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JToggleButton systemToggle = new JToggleButton("System OFF");
        systemToggle.setSelected(system.isSystemOn());
        if (system.isSystemOn()) {
            systemToggle.setText("System ON");
        }

        systemControlPanel.add(systemToggle);

        // Admin-only buttons
        if (system.getCurrentUser().hasPermission("MANAGE_USERS")) {
            JButton addUserButton = new JButton("Add User");
            JButton viewLogsButton = new JButton("View Logs");

            systemControlPanel.add(addUserButton);
            systemControlPanel.add(viewLogsButton);

            addUserButton.addActionListener(e -> showAddUserDialog());
            viewLogsButton.addActionListener(e -> showSystemLogs());
        }

        topPanel.add(userPanel, BorderLayout.EAST);
        topPanel.add(systemControlPanel, BorderLayout.WEST);

        // Create the device list panel
        devicePanel = new JPanel();
        devicePanel.setLayout(new BoxLayout(devicePanel, BoxLayout.Y_AXIS));
        JScrollPane deviceScrollPane = new JScrollPane(devicePanel);
        deviceScrollPane.setBorder(BorderFactory.createTitledBorder("Devices"));

        // Create the control panel (right side)
        controlPanel = new JPanel();
        controlPanel.setLayout(new BorderLayout());
        controlPanel.setBorder(BorderFactory.createTitledBorder("Control Panel"));

        // Create a default message for the control panel
        JLabel defaultControlLabel = new JLabel("Select a device to control");
        defaultControlLabel.setHorizontalAlignment(JLabel.CENTER);
        controlPanel.add(defaultControlLabel, BorderLayout.CENTER);

        // Create a split pane
        JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, deviceScrollPane,
controlPanel);
        splitPane.setDividerLocation(300);

        // Add "Add Device" button if user has permission
        if (system.getCurrentUser().hasPermission("ADD_DEVICE")) {
            JButton addDeviceButton = new JButton("Add New Device");
            addDeviceButton.addActionListener(e -> showAddDeviceDialog());

            JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
            buttonPanel.add(addDeviceButton);

            deviceScrollPane.setColumnHeaderView(buttonPanel);
        }

        // Add components to the main panel
        mainPanel.add(topPanel, BorderLayout.NORTH);
        mainPanel.add(splitPane, BorderLayout.CENTER);

        // Add action listeners
```

```
    logoutButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            system.logout();
            mainFrame.getContentPane().removeAll();
            createLoginPanel();
            mainFrame.revalidate();
            mainFrame.repaint();
        }
    });

    systemToggle.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (systemToggle.isSelected()) {
                system.turnSystemOn();
                systemToggle.setText("System ON");
            } else {
                system.turnSystemOff();
                systemToggle.setText("System OFF");
            }
            updateDeviceList();
        }
    });

    // Populate the device list
    updateDeviceList();
  }

  private void updateDeviceList() {
    devicePanel.removeAll();

    List<Device> devices = system.getAllDevices();

    for (Device device : devices) {
        JPanel deviceItemPanel = new JPanel(new BorderLayout());
        deviceItemPanel.setBorder(BorderFactory.createEtchedBorder());

        String statusText = device instanceof Switchable ?
                    (((Switchable) device).isOn() ? "ON" : "OFF") : "N/A";

        JLabel deviceLabel = new JLabel(device.getName() + " (" + device.getLocation() + ") - " +
statusText);
        deviceItemPanel.add(deviceLabel, BorderLayout.CENTER);

        // All users should be able to select devices
        deviceItemPanel.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                showDeviceControl(device);
            }
        });

        // Only add remove button if user has permission
        if (system.getCurrentUser().hasPermission("REMOVE_DEVICE")) {
            JButton removeButton = new JButton("Remove");
            removeButton.addActionListener(e -> {
                try {
                    system.removeDevice(device.getId());
                    updateDeviceList();
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(mainFrame,
                                                "Error removing device: " + ex.getMessage(),
                                                "Error", JOptionPane.ERROR_MESSAGE);
                }
            });
            deviceItemPanel.add(removeButton, BorderLayout.EAST);
        }

        devicePanel.add(deviceItemPanel);
    }

    devicePanel.revalidate();
    devicePanel.repaint();
}

private void showDeviceControl(Device device) {
    controlPanel.removeAll();

    JPanel deviceControlPanel = new JPanel();
    deviceControlPanel.setLayout(new BoxLayout(deviceControlPanel, BoxLayout.Y_AXIS));

    // Device title
    JLabel titleLabel = new JLabel(device.getName());
    titleLabel.setFont(new Font("Arial", Font.BOLD, 18));
    titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
    deviceControlPanel.add(titleLabel);

    // Location
    JLabel locationLabel = new JLabel("Location: " + device.getLocation());
    locationLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
    deviceControlPanel.add(locationLabel);

    deviceControlPanel.add(Box.createRigidArea(new Dimension(0, 20)));

    // Check if user has control permission before adding controls
    if (system.getCurrentUser().hasPermission("CONTROL_DEVICES")) {
        // On/Off control if device is Switchable
        if (device instanceof Switchable) {
            Switchable switchableDevice = (Switchable) device;
            JPanel switchPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JToggleButton onOffToggle = new JToggleButton(switchableDevice.isOn() ? "ON" : "OFF");
            onOffToggle.setSelected(switchableDevice.isOn());

            onOffToggle.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    if (onOffToggle.isSelected()) {
                        switchableDevice.turnOn();
                        onOffToggle.setText("ON");
                    } else {
                        switchableDevice.turnOff();
                        onOffToggle.setText("OFF");
                    }
                    updateDeviceList();
                }
            });

            switchPanel.add(new JLabel("Power:"));
            switchPanel.add(onOffToggle);
            deviceControlPanel.add(switchPanel);
        }

        // Device-specific controls
        if (device instanceof Light) {
            Light light = (Light) device;

            // Brightness control
            JPanel brightnessPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JSlider brightnessSlider = new JSlider(0, 100, light.getBrightness());
            JLabel brightnessValueLabel = new JLabel(light.getBrightness() + "%");

            brightnessSlider.addChangeListener(e -> {
                int value = brightnessSlider.getValue();
                light.setBrightness(value);
                brightnessValueLabel.setText(value + "%");
                if (!brightnessSlider.getValueIsAdjusting()) {
                    updateDeviceList();
                }
            });

            brightnessPanel.add(new JLabel("Brightness:"));
            brightnessPanel.add(brightnessSlider);
            brightnessPanel.add(brightnessValueLabel);
            deviceControlPanel.add(brightnessPanel);

            // Color selection - NEW FEATURE
```

```
JPanel colorPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
colorPanel.add(new JLabel("Color:"));

String[] colors = {
    Light.COLOR_WHITE,
    Light.COLOR_WARM,
    Light.COLOR_BLUE,
    Light.COLOR_RED
};

JComboBox<String> colorComboBox = new JComboBox<>(colors);
colorComboBox.setSelectedItem(light.getColor());

// Create color buttons with actual colors
JPanel colorButtonsPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

JButton whiteButton = new JButton("   ");
whiteButton.setBackground(Color.WHITE);
whiteButton.setToolTipText(Light.COLOR_WHITE);
whiteButton.addActionListener(e -> {
    light.setColor(Light.COLOR_WHITE);
    colorComboBox.setSelectedItem(Light.COLOR_WHITE);
    updateDeviceList();
});

JButton warmButton = new JButton("   ");
warmButton.setBackground(new Color(255, 244, 229)); // Warm white color
warmButton.setToolTipText(Light.COLOR_WARM);
warmButton.addActionListener(e -> {
    light.setColor(Light.COLOR_WARM);
    colorComboBox.setSelectedItem(Light.COLOR_WARM);
    updateDeviceList();
});

JButton blueButton = new JButton("   ");
blueButton.setBackground(Color.BLUE);
blueButton.setToolTipText(Light.COLOR_BLUE);
blueButton.addActionListener(e -> {
    light.setColor(Light.COLOR_BLUE);
    colorComboBox.setSelectedItem(Light.COLOR_BLUE);
    updateDeviceList();
});

JButton redButton = new JButton("   ");
redButton.setBackground(Color.RED);
redButton.setToolTipText(Light.COLOR_RED);
redButton.addActionListener(e -> {
    light.setColor(Light.COLOR_RED);
    colorComboBox.setSelectedItem(Light.COLOR_RED);
```

```
                updateDeviceList();
            });

            colorButtonsPanel.add(whiteButton);
            colorButtonsPanel.add(warmButton);
            colorButtonsPanel.add(blueButton);
            colorButtonsPanel.add(redButton);

            colorComboBox.addActionListener(e -> {
                String selectedColor = (String) colorComboBox.getSelectedItem();
                light.setColor(selectedColor);
                updateDeviceList();
            });

            colorPanel.add(colorComboBox);
            deviceControlPanel.add(colorPanel);
            deviceControlPanel.add(colorButtonsPanel);

            // Motion activation control
            JPanel motionPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JCheckBox motionCheckbox = new JCheckBox("Motion Activated", light.isMotionActivated());

            motionCheckbox.addActionListener(e -> {
                light.setMotionActivated(motionCheckbox.isSelected());
            });

            motionPanel.add(motionCheckbox);
            deviceControlPanel.add(motionPanel);

        } else if (device instanceof Fan) {
            Fan fan = (Fan) device;

            // Speed control
            JPanel speedPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JSlider speedSlider = new JSlider(1, 5, fan.getSpeed());
            speedSlider.setMajorTickSpacing(1);
            speedSlider.setPaintTicks(true);
            speedSlider.setPaintLabels(true);
            JLabel speedValueLabel = new JLabel("Speed: " + fan.getSpeed());

            speedSlider.addChangeListener(e -> {
                int value = speedSlider.getValue();
                fan.setSpeed(value);
                speedValueLabel.setText("Speed: " + value);
                if (!speedSlider.getValueIsAdjusting()) {
                    updateDeviceList();
                }
            });
            speedPanel.add(speedValueLabel);
            speedPanel.add(speedSlider);
            deviceControlPanel.add(speedPanel);

            // Oscillation and Auto-adjust options removed as requested

        } else if (device instanceof AirConditioner) {
            AirConditioner ac = (AirConditioner) device;

            // Temperature control
            JPanel tempPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JSlider tempSlider = new JSlider(16, 30, ac.getTemperature());
            tempSlider.setMajorTickSpacing(2);
            tempSlider.setPaintTicks(true);
            tempSlider.setPaintLabels(true);
            JLabel tempValueLabel = new JLabel(ac.getTemperature() + "°C");

            tempSlider.addChangeListener(e -> {
                int value = tempSlider.getValue();
                ac.setTemperature(value);
                tempValueLabel.setText(value + "°C");
                if (!tempSlider.getValueIsAdjusting()) {
                    updateDeviceList();
                }
            });

            tempPanel.add(new JLabel("Temperature:"));
            tempPanel.add(tempSlider);
            tempPanel.add(tempValueLabel);
            deviceControlPanel.add(tempPanel);

            // Mode control
            JPanel modePanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            String[] modes = {"COOL", "HEAT", "FAN", "DRY", "AUTO"};
            JComboBox<String> modeComboBox = new JComboBox<>(modes);
            modeComboBox.setSelectedItem(ac.getMode());

            modeComboBox.addActionListener(e -> {
                String selectedMode = (String) modeComboBox.getSelectedItem();
                ac.setMode(selectedMode);
                updateDeviceList();
            });

            modePanel.add(new JLabel("Mode:"));
            modePanel.add(modeComboBox);
            deviceControlPanel.add(modePanel);

            // Energy saving mode
            JPanel energyPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JCheckBox energyCheckbox = new JCheckBox("Energy Saving Mode",
ac.isEnergySavingMode());

            energyCheckbox.addActionListener(e -> {
                ac.setEnergySavingMode(energyCheckbox.isSelected());
                updateDeviceList();
            });

            energyPanel.add(energyCheckbox);
            deviceControlPanel.add(energyPanel);

            // Auto temperature adjust - NEW FEATURE
            JPanel autoTempPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JCheckBox autoTempCheckbox = new JCheckBox("Auto Temperature Adjust",
ac.isAutoTempAdjust());

            autoTempCheckbox.addActionListener(e -> {
                ac.setAutoTempAdjust(autoTempCheckbox.isSelected());
                updateDeviceList();
            });

            autoTempPanel.add(autoTempCheckbox);
            deviceControlPanel.add(autoTempPanel);

        } else if (device instanceof SecuritySystem) {
            SecuritySystem security = (SecuritySystem) device;

            // Security mode control
            JPanel modePanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            String[] modes = {"DISARMED", "HOME", "AWAY"};
            JComboBox<String> modeComboBox = new JComboBox<>(modes);
            modeComboBox.setSelectedItem(security.getSecurityMode());

            modeComboBox.addActionListener(e -> {
                String selectedMode = (String) modeComboBox.getSelectedItem();
                security.setSecurityMode(selectedMode);
                updateDeviceList();
            });

            modePanel.add(new JLabel("Security Mode:"));
            modePanel.add(modeComboBox);
            deviceControlPanel.add(modePanel);

            // Alarm status
            JPanel alarmPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
            JLabel alarmStatusLabel = new JLabel("Alarm Status: " +
                                    (security.isAlarmActive() ? "ACTIVE" : "Inactive"));
            alarmStatusLabel.setForeground(security.isAlarmActive() ? Color.RED : Color.BLACK);
```



```
        JButton alarmButton = new JButton(security.isAlarmActive() ? "Deactivate Alarm" : "Test Alarm")

        alarmButton.addActionListener(e -> {
            if (security.isAlarmActive()) {
                security.deactivateAlarm();
            } else {
                security.activateAlarm();
            }
            updateDeviceList();
            showDeviceControl(device); // Refresh the control panel
        });

        alarmPanel.add(alarmStatusLabel);
        alarmPanel.add(alarmButton);
        deviceControlPanel.add(alarmPanel);

        // Security logs
        if (system.getCurrentUser().hasPermission("VIEW_LOGS")) {
            JPanel logsPanel = new JPanel(new BorderLayout());
            logsPanel.setBorder(BorderFactory.createTitledBorder("Security Logs"));

            JTextArea logsTextArea = new JTextArea(10, 30);
            logsTextArea.setEditable(false);
            JScrollPane logsScrollPane = new JScrollPane(logsTextArea);

            List<String> logs = security.getSecurityLogs();
            for (String log : logs) {
                logsTextArea.append(log + "\n");
            }

            logsPanel.add(logsScrollPane, BorderLayout.CENTER);
            deviceControlPanel.add(logsPanel);
        }
    }

    // Scheduling section
    if (system.getCurrentUser().hasPermission("CONTROL_DEVICES")) {
        deviceControlPanel.add(Box.createRigidArea(new Dimension(0, 20)));
        JPanel schedulePanel = new JPanel(new BorderLayout());
        schedulePanel.setBorder(BorderFactory.createTitledBorder("Scheduled Tasks"));

        JButton addScheduleButton = new JButton("Add Schedule");
        schedulePanel.add(addScheduleButton, BorderLayout.NORTH);

        JPanel tasksPanel = new JPanel();
        tasksPanel.setLayout(new BoxLayout(tasksPanel, BoxLayout.Y_AXIS));
        JScrollPane tasksScrollPane = new JScrollPane(tasksPanel);

        // Populate scheduled tasks
```

```
            List<ScheduledTask> tasks = device.getScheduledTasks();
            for (ScheduledTask task : tasks) {
                JPanel taskItemPanel = new JPanel(new BorderLayout());
                taskItemPanel.setBorder(BorderFactory.createEtchedBorder());

                JLabel taskLabel = new JLabel(task.toString());
                JButton removeButton = new JButton("Remove");

                removeButton.addActionListener(e -> {
                    device.removeScheduledTask(task);
                    showDeviceControl(device); // Refresh the control panel
                });

                taskItemPanel.add(taskLabel, BorderLayout.CENTER);
                taskItemPanel.add(removeButton, BorderLayout.EAST);

                tasksPanel.add(taskItemPanel);
            }

            schedulePanel.add(tasksScrollPane, BorderLayout.CENTER);
            deviceControlPanel.add(schedulePanel);

            // Add schedule button action
            addScheduleButton.addActionListener(e -> {
                showAddScheduleDialog(device);
            });
        }
    } else {
        // If user doesn't have control permission, just show device info
        JLabel infoLabel = new JLabel("You don't have permission to control this device");
        infoLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        deviceControlPanel.add(infoLabel);
    }

    // Add the device control panel to the main control panel
    JScrollPane controlScrollPane = new JScrollPane(deviceControlPanel);
    controlPanel.add(controlScrollPane, BorderLayout.CENTER);

    controlPanel.revalidate();
    controlPanel.repaint();
}

private void showAddDeviceDialog() {
    JDialog dialog = new JDialog(mainFrame, "Add New Device", true);
    dialog.setSize(400, 300);
    dialog.setLayout(new BorderLayout());

    JPanel formPanel = new JPanel();
    formPanel.setLayout(new BoxLayout(formPanel, BoxLayout.Y_AXIS));
formPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Device type selection
JPanel typePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
typePanel.add(new JLabel("Device Type:"));
String[] deviceTypes = {"Light", "Fan", "Air Conditioner", "Security System"};
JComboBox<String> typeComboBox = new JComboBox<>(deviceTypes);
typePanel.add(typeComboBox);
formPanel.add(typePanel);

// Device name
JPanel namePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
namePanel.add(new JLabel("Device Name:"));
JTextField nameField = new JTextField(20);
namePanel.add(nameField);
formPanel.add(namePanel);

// Device location
JPanel locationPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
locationPanel.add(new JLabel("Location:"));
JTextField locationField = new JTextField(20);
locationPanel.add(locationField);
formPanel.add(locationPanel);

// Buttons
JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
JButton cancelButton = new JButton("Cancel");
JButton addButton = new JButton("Add Device");

cancelButton.addActionListener(e -> dialog.dispose());

addButton.addActionListener(e -> {
    String name = nameField.getText();
    String location = locationField.getText();
    String type = (String) typeComboBox.getSelectedItem();

    if (name.isEmpty() || location.isEmpty()) {
        JOptionPane.showMessageDialog(dialog,
                                    "Name and location cannot be empty",
                                    "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    try {
        Device newDevice = null;
        String createdBy = system.getCurrentUser().getUsername();

        switch (type) {
            case "Light":
                    newDevice = new Light(name, location, createdBy);
                    break;
                case "Fan":
                    newDevice = new Fan(name, location, createdBy);
                    break;
                case "Air Conditioner":
                    newDevice = new AirConditioner(name, location, createdBy);
                    break;
                case "Security System":
                    newDevice = new SecuritySystem(name, location, createdBy);
                    break;
            }

            if (newDevice != null) {
                system.addDevice(newDevice);
                updateDeviceList();
                dialog.dispose();
            }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(dialog,
                                        "Error adding device: " + ex.getMessage(),
                                        "Error", JOptionPane.ERROR_MESSAGE);
        }
    });

    buttonPanel.add(cancelButton);
    buttonPanel.add(addButton);

    dialog.add(formPanel, BorderLayout.CENTER);
    dialog.add(buttonPanel, BorderLayout.SOUTH);

    dialog.setLocationRelativeTo(mainFrame);
    dialog.setVisible(true);
}

private void showAddUserDialog() {
    JDialog dialog = new JDialog(mainFrame, "Add New User", true);
    dialog.setSize(400, 300);
    dialog.setLayout(new BorderLayout());

    JPanel formPanel = new JPanel();
    formPanel.setLayout(new BoxLayout(formPanel, BoxLayout.Y_AXIS));
    formPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    // Username
    JPanel usernamePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
    usernamePanel.add(new JLabel("Username:"));
    JTextField usernameField = new JTextField(20);
    usernamePanel.add(usernameField);
```

```
formPanel.add(usernamePanel);

// Password
JPanel passwordPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
passwordPanel.add(new JLabel("Password:"));
JPasswordField passwordField = new JPasswordField(20);
passwordPanel.add(passwordField);
formPanel.add(passwordPanel);

// Name
JPanel namePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
namePanel.add(new JLabel("Display Name:"));
JTextField nameField = new JTextField(20);
namePanel.add(nameField);
formPanel.add(namePanel);

// Role
JPanel rolePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
rolePanel.add(new JLabel("Role:"));
String[] roles = {"User", "Admin"};
JComboBox<String> roleComboBox = new JComboBox<>(roles);
rolePanel.add(roleComboBox);
formPanel.add(rolePanel);

// Buttons
JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
JButton cancelButton = new JButton("Cancel");
JButton addButton = new JButton("Add User");

cancelButton.addActionListener(e -> dialog.dispose());

addButton.addActionListener(e -> {
    String username = usernameField.getText();
    String password = new String(passwordField.getPassword());
    String name = nameField.getText();
    String role = (String) roleComboBox.getSelectedItem();

    if (username.isEmpty() || password.isEmpty()) {
        JOptionPane.showMessageDialog(dialog,
                                    "Username and password cannot be empty",
                                    "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    try {
        User newUser;
        if (role.equals("Admin")) {
            newUser = new AdminUser(username, password, name);
        } else {
                newUser = new User(username, password, name);
            }

            system.addUser(newUser);
            dialog.dispose();

            JOptionPane.showMessageDialog(mainFrame,
                                        "User added successfully",
                                        "Success", JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(dialog,
                                        "Error adding user: " + ex.getMessage(),
                                        "Error", JOptionPane.ERROR_MESSAGE);
        }
    });

    buttonPanel.add(cancelButton);
    buttonPanel.add(addButton);

    dialog.add(formPanel, BorderLayout.CENTER);
    dialog.add(buttonPanel, BorderLayout.SOUTH);

    dialog.setLocationRelativeTo(mainFrame);
    dialog.setVisible(true);
}

private void showSystemLogs() {
    JDialog logsDialog = new JDialog(mainFrame, "System Logs", true);
    logsDialog.setSize(700, 500);
    logsDialog.setLayout(new BorderLayout());

    JTextArea logsArea = new JTextArea();
    logsArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(logsArea);

    List<String> logs = system.getSystemLogs();
    for (String log : logs) {
        logsArea.append(log + "\n");
    }

    JButton closeButton = new JButton("Close");
    closeButton.addActionListener(e -> logsDialog.dispose());

    JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    buttonPanel.add(closeButton);

    logsDialog.add(scrollPane, BorderLayout.CENTER);
    logsDialog.add(buttonPanel, BorderLayout.SOUTH);
```

```
        logsDialog.setLocationRelativeTo(mainFrame);
        logsDialog.setVisible(true);
    }

    private void showAddScheduleDialog(Device device) {
        JDialog scheduleDialog = new JDialog(mainFrame, "Add Schedule", true);
        scheduleDialog.setSize(400, 400);
        scheduleDialog.setLayout(new BorderLayout());

        JPanel formPanel = new JPanel();
        formPanel.setLayout(new BoxLayout(formPanel, BoxLayout.Y_AXIS));
        formPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        // Task name
        JPanel namePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        namePanel.add(new JLabel("Task Name:"));
        JTextField nameField = new JTextField(20);
        namePanel.add(nameField);
        formPanel.add(namePanel);

        // Action selection
        JPanel actionPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        actionPanel.add(new JLabel("Action:"));

        String[] actions;
        if (device instanceof Light) {
            actions = new String[]{"ON", "OFF", "SET_BRIGHTNESS"};
        } else if (device instanceof Fan) {
            actions = new String[]{"ON", "OFF", "SET_SPEED"};
        } else if (device instanceof AirConditioner) {
            actions = new String[]{"ON", "OFF", "SET_TEMPERATURE"};
        } else if (device instanceof SecuritySystem) {
            actions = new String[]{"ON", "OFF", "SET_SECURITY_MODE"};
        } else {
            actions = new String[]{"ON", "OFF"};
        }

        JComboBox<String> actionComboBox = new JComboBox<>(actions);
        actionPanel.add(actionComboBox);
        formPanel.add(actionPanel);

        // Parameters panel (changes based on action)
        JPanel parametersPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel paramLabel = new JLabel("Value:");
        JTextField paramField = new JTextField(10);
        parametersPanel.add(paramLabel);
        parametersPanel.add(paramField);
        formPanel.add(parametersPanel);
```

```java
// Initially hide parameters if not needed
parametersPanel.setVisible(actionComboBox.getSelectedItem().toString().startsWith("SET_"));

actionComboBox.addActionListener(e -> {
    String selectedAction = actionComboBox.getSelectedItem().toString();
    parametersPanel.setVisible(selectedAction.startsWith("SET_"));

    if (selectedAction.equals("SET_BRIGHTNESS")) {
        paramLabel.setText("Brightness (0-100):");
    } else if (selectedAction.equals("SET_SPEED")) {
        paramLabel.setText("Speed (1-5):");
    } else if (selectedAction.equals("SET_TEMPERATURE")) {
        paramLabel.setText("Temperature (16-30):");
    } else if (selectedAction.equals("SET_SECURITY_MODE")) {
        paramLabel.setText("Mode (DISARMED/HOME/AWAY):");
    }
});

// Time selection
JPanel timePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
timePanel.add(new JLabel("Time:"));
JSpinner timeSpinner = new JSpinner(new SpinnerDateModel());
JSpinner.DateEditor timeEditor = new JSpinner.DateEditor(timeSpinner, "HH:mm");
timeSpinner.setEditor(timeEditor);
timePanel.add(timeSpinner);
formPanel.add(timePanel);

// Days of week
JPanel daysPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
daysPanel.add(new JLabel("Days:"));
JCheckBox[] dayCheckboxes = new JCheckBox[7];
String[] dayNames = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};

for (int i = 0; i < 7; i++) {
    dayCheckboxes[i] = new JCheckBox(dayNames[i]);
    daysPanel.add(dayCheckboxes[i]);
}
formPanel.add(daysPanel);

// Buttons
JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
JButton cancelButton = new JButton("Cancel");
JButton saveButton = new JButton("Save");

cancelButton.addActionListener(e -> scheduleDialog.dispose());

saveButton.addActionListener(e -> {
    String taskName = nameField.getText();
    String action = actionComboBox.getSelectedItem().toString();
```

```
      String[] parameters = null;
      if (action.startsWith("SET_")) {
        parameters = new String[]{paramField.getText()};
      } else {
        parameters = new String[0];
      }

      // Get time
      Date date = (Date) timeSpinner.getValue();
      LocalTime time = LocalTime.of(date.~~getHours~~(), date.~~getMinutes~~());

      // Get days
      boolean[] days = new boolean[7];
      for (int i = 0; i < 7; i++) {
        days[i] = dayCheckboxes[i].isSelected();
      }

      // Create and add the scheduled task
      ScheduledTask task = new ScheduledTask(taskName, device, action, parameters, time, days);
      device.addScheduledTask(task);

      scheduleDialog.dispose();
      showDeviceControl(device); // Refresh the control panel
    });

    buttonPanel.add(cancelButton);
    buttonPanel.add(saveButton);

    scheduleDialog.add(formPanel, BorderLayout.CENTER);
    scheduleDialog.add(buttonPanel, BorderLayout.SOUTH);

    scheduleDialog.setLocationRelativeTo(mainFrame);
    scheduleDialog.setVisible(true);
  }
}
```

AuthenticationException CLASS

```
package smarthome.exceptions;
public class AuthenticationException extends Exception {

  public AuthenticationException(String message) {
    super(message);
  }



}package smarthome.exceptions;
```

```
public class DeviceNotFoundException extends Exception {

  public DeviceNotFoundException(String message) {
    super(message);
  }
}
package smarthome.exceptions;
public class DeviceNotFoundException extends Exception {

  public DeviceNotFoundException(String message) {
    super(message);
  }
}
package smarthome.interfaces;
public interface Switchable {
  void turnOn();
  void turnOff();
  boolean isOn();
}
package smarthome.models;
import java.util.ArrayList;
public class AdminUser extends User {

  public AdminUser(String username, String password) {
    super(username, password);
    setupAdminPermissions();
  }

  public AdminUser(String username, String password, String name) {
    super(username, password, name);
    setupAdminPermissions();
  }

  private void setupAdminPermissions() {
    // Admin has additional permissions
    addPermission("ADD_DEVICE");
    addPermission("REMOVE_DEVICE");
    addPermission("MANAGE_USERS");
    addPermission("VIEW_LOGS");
    addPermission("SYSTEM_SETTINGS");
  }

  @Override
  public String getRole() {
    return "ADMIN";
  }

  @Override
  public String toString() {
```

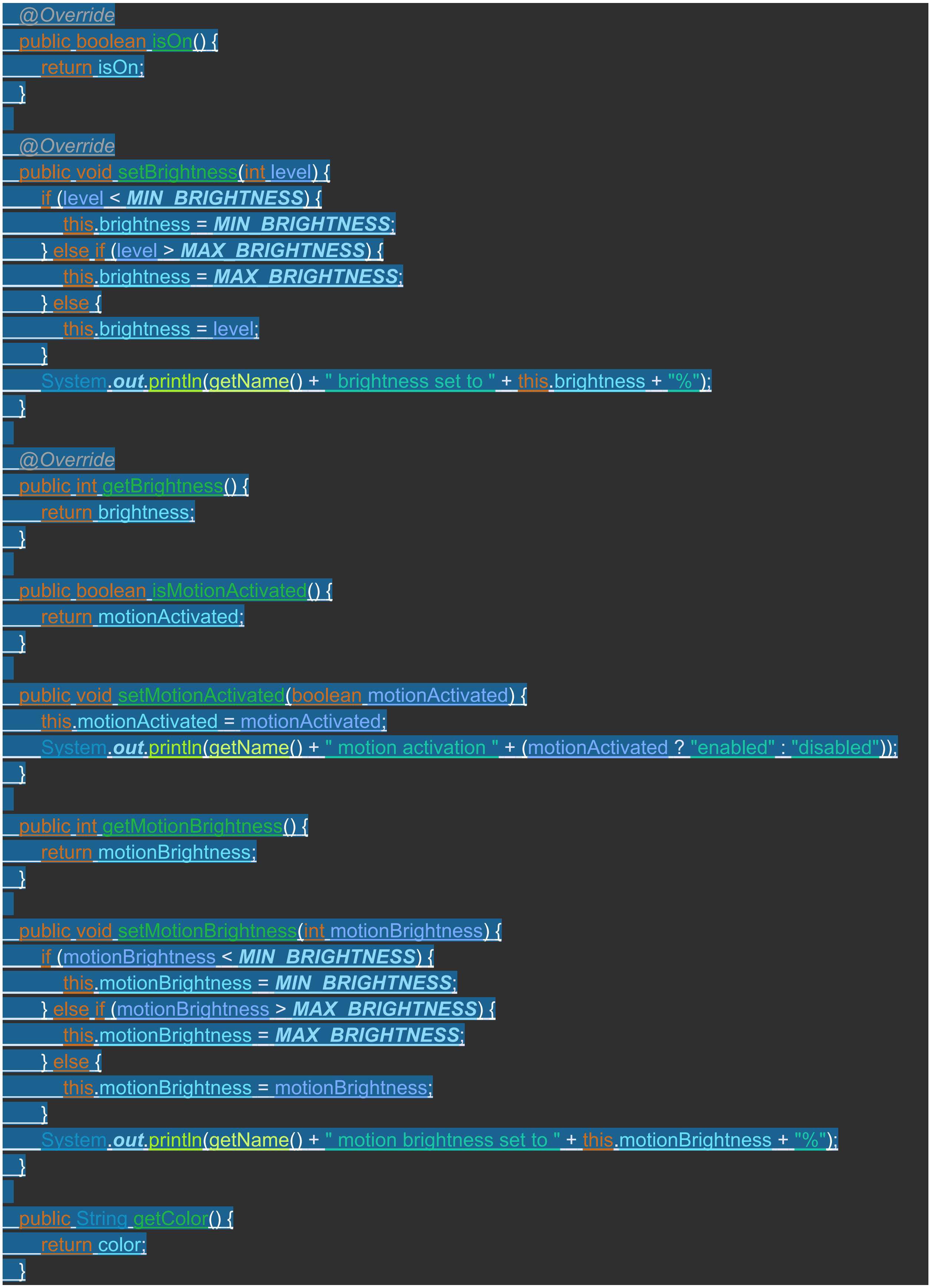
```java
        return super.toString() + " [ADMIN]";
    }
}
package smarthome.models;
import smarthome.interfaces.Switchable;
import java.util.Random;
import java.time.LocalTime;
public class AirConditioner extends Device implements Switchable {
    private boolean isOn;
    private int temperature;
    private String mode; // COOL, HEAT, FAN, DRY, AUTO
    private static final int DEFAULT_TEMPERATURE = 24;
    private static final int MAX_TEMPERATURE = 30;
    private static final int MIN_TEMPERATURE = 16;
    private static final String DEFAULT_MODE = "COOL";
    private boolean energySavingMode;
    private boolean autoTempAdjust; // Added auto temp adjust
    private LocalTime quietHoursStart;
    private LocalTime quietHoursEnd;
    
    public AirConditioner(String name, String location, String createdBy) {
        super(name, location, createdBy);
        this.isOn = false;
        this.temperature = DEFAULT_TEMPERATURE;
        this.mode = DEFAULT_MODE;
        this.energySavingMode = false;
        this.autoTempAdjust = false; // Default is off
        this.quietHoursStart = LocalTime.of(22, 0); // 10 PM
        this.quietHoursEnd = LocalTime.of(7, 0);    // 7 AM
    }
    
    @Override
    protected String generateDeviceId() {
        Random r = new Random();
        return "AC-" + (10000 + r.nextInt(90000));
    }
    
    @Override
    public void turnOn() {
        if (!isOn) {
            isOn = true;
            updateLastStateChange();
            System.out.println(getName() + " turned ON");
        }
    }
    
    @Override
    public void turnOff() {
        if (isOn) {
            isOn = false;
            updateLastStateChange();
            System.out.println(getName() + " turned OFF");
        }
    }

    @Override
    public boolean isOn() {
        return isOn;
    }

    public void setTemperature(int temperature) {
        // Enforce temperature limits
        if (temperature < MIN_TEMPERATURE) {
            this.temperature = MIN_TEMPERATURE;
        } else if (temperature > MAX_TEMPERATURE) {
            this.temperature = MAX_TEMPERATURE;
        } else {
            this.temperature = temperature;
        }
        System.out.println(getName() + " temperature set to " + this.temperature + "°C");
    }

    public int getTemperature() {
        return temperature;
    }

    public void setMode(String mode) {
        // Validate mode
        if (mode.equals("COOL") || mode.equals("HEAT") || mode.equals("FAN") ||
            mode.equals("DRY") || mode.equals("AUTO")) {
            this.mode = mode;
        } else {
            // Invalid mode, use default
            this.mode = DEFAULT_MODE;
        }
        System.out.println(getName() + " mode set to " + this.mode);
    }

    public String getMode() {
        return mode;
    }

    public boolean isEnergySavingMode() {
        return energySavingMode;
    }

    public void setEnergySavingMode(boolean energySavingMode) {
        this.energySavingMode = energySavingMode;
```

```java

        // If energy saving mode is enabled, adjust settings
        if (energySavingMode && isOn) {
            // In cooling mode, increase temperature to save energy
            if (mode.equals("COOL") && temperature < 24) {
                setTemperature(24);
            }
            // In heating mode, decrease temperature to save energy
            else if (mode.equals("HEAT") && temperature > 20) {
                setTemperature(20);
            }
        }
        System.out.println(getName() + " energy saving mode " + (energySavingMode ? "enabled" : "disabled"));
    }

    public boolean isAutoTempAdjust() {
        return autoTempAdjust;
    }

    public void setAutoTempAdjust(boolean autoTempAdjust) {
        this.autoTempAdjust = autoTempAdjust;
        System.out.println(getName() + " auto temperature adjustment " + (autoTempAdjust ? "enabled" : "disabled"));

        // If auto temp adjust is enabled, adjust temperature based on time of day
        if (autoTempAdjust && isOn) {
            adjustTemperatureAuto();
        }
    }

    // Auto adjust temperature based on time of day
    public void adjustTemperatureAuto() {
        if (!autoTempAdjust || !isOn) {
            return;
        }

        LocalTime now = LocalTime.now();

        // Early morning (5-8 AM): Comfortable waking temperature
        if (now.isAfter(LocalTime.of(5, 0)) && now.isBefore(LocalTime.of(8, 0))) {
            if (mode.equals("COOL")) {
                setTemperature(23); // Slightly cooler in the morning
            } else if (mode.equals("HEAT")) {
                setTemperature(22); // Warmer in the morning
            }
        }
        // Daytime (8 AM-5 PM): Energy efficient
        else if (now.isAfter(LocalTime.of(8, 0)) && now.isBefore(LocalTime.of(17, 0))) {
            if (mode.equals("COOL")) {
                setTemperature(25); // Higher during day when people may be out
            } else if (mode.equals("HEAT")) {
                setTemperature(20); // Lower during day
            }
        }
        // Evening (5-10 PM): Comfortable evening temperature
        else if (now.isAfter(LocalTime.of(17, 0)) && now.isBefore(LocalTime.of(22, 0))) {
            if (mode.equals("COOL")) {
                setTemperature(24); // Comfortable evening temperature
            } else if (mode.equals("HEAT")) {
                setTemperature(22); // Comfortable evening temperature
            }
        }
        // Night (10 PM-5 AM): Sleep temperature
        else {
            if (mode.equals("COOL")) {
                setTemperature(26); // Higher at night for sleep
            } else if (mode.equals("HEAT")) {
                setTemperature(19); // Lower at night for sleep
            }
        }
    }

    public void setQuietHours(LocalTime start, LocalTime end) {
        this.quietHoursStart = start;
        this.quietHoursEnd = end;
    }

    // Check if current time is during quiet hours
    public boolean isQuietHours() {
        LocalTime now = LocalTime.now();

        if (quietHoursStart.isBefore(quietHoursEnd)) {
            // Simple case: start time is before end time (e.g., 22:00 to 07:00)
            return now.isAfter(quietHoursStart) && now.isBefore(quietHoursEnd);
        } else {
            // Complex case: start time is after end time (spans midnight)
            return now.isAfter(quietHoursStart) || now.isBefore(quietHoursEnd);
        }
    }

    // Adjust settings for quiet hours
    public void adjustForQuietHours() {
        if (isOn && isQuietHours()) {
            // During quiet hours, use more moderate settings
            if (mode.equals("COOL") && temperature < 24) {
                setTemperature(24);
            } else if (mode.equals("HEAT") && temperature > 22) {
```

```java
                setTemperature(22);
            }
        }
    }

    @Override
    public void setToDefaultSettings() {
        this.temperature = DEFAULT_TEMPERATURE;
        this.mode = DEFAULT_MODE;
        System.out.println(getName() + " set to default temperature: " + DEFAULT_TEMPERATURE +
                          "°C, mode: " + DEFAULT_MODE);
    }

    @Override
    public String toString() {
        return super.toString() + " - Status: " + (isOn ? "ON" : "OFF") +
               ", Temperature: " + temperature + "°C, Mode: " + mode +
               ", Energy Saving: " + (energySavingMode ? "ON" : "OFF") +
               ", Auto Temp Adjust: " + (autoTempAdjust ? "ON" : "OFF");
    }
}
package smarthome.models;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
public abstract class Device {
    private String id;
    private String name;
    private String location;
    private LocalDateTime lastStateChange;
    private List<ScheduledTask> scheduledTasks;
    private String createdBy; // Track who created this device

    // Constructor with random ID generation
    public Device(String name, String location, String createdBy) {
        // Generate a somewhat random ID with a prefix based on device type
        this.id = generateDeviceId();
        this.name = name;
        this.location = location;
        this.lastStateChange = LocalDateTime.now();
        this.scheduledTasks = new ArrayList<>();
        this.createdBy = createdBy;
    }

    // Constructor with specific ID
    public Device(String id, String name, String location, String createdBy) {
        this.id = id;
        this.name = name;
        this.location = location;
        this.lastStateChange = LocalDateTime.now();
        this.scheduledTasks = new ArrayList<>();
        this.createdBy = createdBy;
    }

    // Generate a random device ID
    protected String generateDeviceId() {
        // Each implementation can override this to create type-specific IDs
        Random rand = new Random();
        return "DEV-" + rand.nextInt(10000);
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public LocalDateTime getLastStateChange() {
        return lastStateChange;
    }

    protected void updateLastStateChange() {
        this.lastStateChange = LocalDateTime.now();
    }

    public String getCreatedBy() {
        return createdBy;
    }

    public void addScheduledTask(ScheduledTask task) {
        scheduledTasks.add(task);
    }

```

```java
    public void removeScheduledTask(ScheduledTask task) {
        scheduledTasks.remove(task);
    }

    public List<ScheduledTask> getScheduledTasks() {
        return new ArrayList<>(scheduledTasks);
    }

    public abstract void turnOn();

    public abstract void turnOff();

    public abstract boolean isOn();

    public abstract void setToDefaultSettings();

    @Override
    public String toString() {
        return name + " (" + location + ")";
    }
}
package smarthome.models;
import smarthome.interfaces.Switchable;
import java.util.Random;
public class Fan extends Device implements Switchable {
    private boolean isOn;
    private int speed;
    private static final int DEFAULT_SPEED = 2;
    private static final int MAX_SPEED = 5;
    private static final int MIN_SPEED = 1;

    public Fan(String name, String location, String createdBy) {
        super(name, location, createdBy);
        this.isOn = false;
        this.speed = DEFAULT_SPEED;
    }

    @Override
    protected String generateDeviceId() {
        Random r = new Random();
        return "FAN-" + (10000 + r.nextInt(90000));
    }

    @Override
    public void turnOn() {
        if (!isOn) {
            isOn = true;
            updateLastStateChange();
            System.out.println(getName() + " turned ON");
        }
    }

    @Override
    public void turnOff() {
        if (isOn) {
            isOn = false;
            updateLastStateChange();
            System.out.println(getName() + " turned OFF");
        }
    }

    @Override
    public boolean isOn() {
        return isOn;
    }

    public void setSpeed(int speed) {
        // Validate speed is within range
        if (speed < MIN_SPEED) {
            this.speed = MIN_SPEED;
        } else if (speed > MAX_SPEED) {
            this.speed = MAX_SPEED;
        } else {
            this.speed = speed;
        }
        System.out.println(getName() + " speed set to " + this.speed);
    }

    public int getSpeed() {
        return speed;
    }

    @Override
    public void setToDefaultSettings() {
        this.speed = DEFAULT_SPEED;
        System.out.println(getName() + " set to default speed: " + DEFAULT_SPEED);
    }

    @Override
    public String toString() {
        return super.toString() + " - Status: " + (isOn ? "ON" : "OFF") +
               ", Speed: " + speed;
    }
}
package smarthome.models;
import smarthome.interfaces.Switchable;
import smarthome.interfaces.Dimmable;
import java.util.Random;
```

```java
public class Light extends Device implements Switchable, Dimmable {
    private boolean isOn;
    private int brightness;
    private boolean motionActivated;
    private int motionBrightness;
    private String color; // Added color property
    private static final int DEFAULT_BRIGHTNESS = 50;
    private static final int MAX_BRIGHTNESS = 100;
    private static final int MIN_BRIGHTNESS = 0;

    // Available colors
    public static final String COLOR_WHITE = "White";
    public static final String COLOR_WARM = "Warm White";
    public static final String COLOR_BLUE = "Blue";
    public static final String COLOR_RED = "Red";

    public Light(String name, String location, String createdBy) {
        super(name, location, createdBy);
        this.isOn = false;
        this.brightness = DEFAULT_BRIGHTNESS;
        this.motionActivated = true;
        this.motionBrightness = 70;
        this.color = COLOR_WHITE; // Default color
    }

    @Override
    protected String generateDeviceId() {
        Random r = new Random();
        return "LIGHT-" + (10000 + r.nextInt(90000));
    }

    @Override
    public void turnOn() {
        if (!isOn) {
            isOn = true;
            updateLastStateChange();
            System.out.println(getName() + " turned ON");
        }
    }

    @Override
    public void turnOff() {
        if (isOn) {
            isOn = false;
            updateLastStateChange();
            System.out.println(getName() + " turned OFF");
        }
    }

```



```
    @Override
    public boolean isOn() {
        return isOn;
    }

    @Override
    public void setBrightness(int level) {
        if (level < MIN_BRIGHTNESS) {
            this.brightness = MIN_BRIGHTNESS;
        } else if (level > MAX_BRIGHTNESS) {
            this.brightness = MAX_BRIGHTNESS;
        } else {
            this.brightness = level;
        }
        System.out.println(getName() + " brightness set to " + this.brightness + "%");
    }

    @Override
    public int getBrightness() {
        return brightness;
    }

    public boolean isMotionActivated() {
        return motionActivated;
    }

    public void setMotionActivated(boolean motionActivated) {
        this.motionActivated = motionActivated;
        System.out.println(getName() + " motion activation " + (motionActivated ? "enabled" : "disabled"));
    }

    public int getMotionBrightness() {
        return motionBrightness;
    }

    public void setMotionBrightness(int motionBrightness) {
        if (motionBrightness < MIN_BRIGHTNESS) {
            this.motionBrightness = MIN_BRIGHTNESS;
        } else if (motionBrightness > MAX_BRIGHTNESS) {
            this.motionBrightness = MAX_BRIGHTNESS;
        } else {
            this.motionBrightness = motionBrightness;
        }
        System.out.println(getName() + " motion brightness set to " + this.motionBrightness + "%");
    }

    public String getColor() {
        return color;
    }
```

```java

    public void setColor(String color) {
        this.color = color;
        System.out.println(getName() + " color set to " + this.color);
    }

    public void activateByMotion() {
        if (motionActivated) {
            turnOn();
            int prevBrightness = brightness;
            setBrightness(motionBrightness);
            System.out.println(getName() + " activated by motion detection");
        }
    }

    @Override
    public void setToDefaultSettings() {
        this.brightness = DEFAULT_BRIGHTNESS;
        this.color = COLOR_WHITE;
        System.out.println(getName() + " set to default brightness: " + DEFAULT_BRIGHTNESS + "%, color: " + COLOR_WHITE);
    }

    @Override
    public String toString() {
        return super.toString() + " - Status: " + (isOn ? "ON" : "OFF") +
               ", Brightness: " + brightness + "%, Color: " + color +
               ", Motion Activated: " + (motionActivated ? "Yes" : "No");
    }
}
package smarthome.models;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.UUID;
import smarthome.interfaces.Switchable;
public class ScheduledTask {
    private String id;
    private String name;
    private Device device;
    private String action; // ON, OFF, SET_TEMPERATURE, etc.
    private String[] parameters; // Additional parameters for the action
    private LocalTime time; // Time to execute the task
    private boolean[] daysOfWeek; // Sunday to Saturday
    private boolean isEnabled;

    public ScheduledTask(String name, Device device, String action, String[] parameters,
                         LocalTime time, boolean[] daysOfWeek) {
        this.id = UUID.randomUUID().toString();
        this.name = name;
        this.device = device;
        this.action = action;
        this.parameters = parameters;
        this.time = time;
        this.daysOfWeek = daysOfWeek;
        this.isEnabled = true;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Device getDevice() {
        return device;
    }

    public String getAction() {
        return action;
    }

    public String[] getParameters() {
        return parameters;
    }

    public LocalTime getTime() {
        return time;
    }

    public void setTime(LocalTime time) {
        this.time = time;
    }

    public boolean[] getDaysOfWeek() {
        return daysOfWeek;
    }

    public void setDaysOfWeek(boolean[] daysOfWeek) {
        this.daysOfWeek = daysOfWeek;
    }

    public boolean isEnabled() {
        return isEnabled;
    }

    public void setEnabled(boolean enabled) {
        this.isEnabled = enabled;
    }

    public void execute() {
        if (!isEnabled) return;

        System.out.println("Executing scheduled task: " + name);

        switch (action) {
            case "ON":
                if (device instanceof Switchable) {
                    ((Switchable) device).turnOn();
                }
                break;
            case "OFF":
                if (device instanceof Switchable) {
                    ((Switchable) device).turnOff();
                }
                break;
            case "SET_TEMPERATURE":
                if (device instanceof AirConditioner && parameters.length > 0) {
                    try {
                        int temp = Integer.parseInt(parameters[0]);
                        ((AirConditioner) device).setTemperature(temp);
                    } catch (NumberFormatException e) {
                        System.err.println("Invalid temperature parameter: " + parameters[0]);
                    }
                }
                break;
            case "SET_BRIGHTNESS":
                if (device instanceof Light && parameters.length > 0) {
                    try {
                        int brightness = Integer.parseInt(parameters[0]);
                        ((Light) device).setBrightness(brightness);
                    } catch (NumberFormatException e) {
                        System.err.println("Invalid brightness parameter: " + parameters[0]);
                    }
                }
                break;
            case "SET_SPEED":
                if (device instanceof Fan && parameters.length > 0) {
                    try {
                        int speed = Integer.parseInt(parameters[0]);
                        ((Fan) device).setSpeed(speed);
                    } catch (NumberFormatException e) {
```

```java
                        System.err.println("Invalid speed parameter: " + parameters[0]);
                    }
                }
                break;
            case "SET_SECURITY_MODE":
                if (device instanceof SecuritySystem && parameters.length > 0) {
                    ((SecuritySystem) device).setSecurityMode(parameters[0]);
                }
                break;
            default:
                System.out.println("Unknown action: " + action);
        }
    }

    @Override
    public String toString() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");
        String timeStr = time.format(formatter);

        String daysStr = "";
        String[] dayNames = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
        for (int i = 0; i < daysOfWeek.length; i++) {
            if (daysOfWeek[i]) {
                daysStr += dayNames[i] + " ";
            }
        }

        return name + " - " + device.getName() + " - " + action + " at " + timeStr + " on " + daysStr.trim();
    }
}
package smarthome.models;
import smarthome.interfaces.Switchable;
import java.util.ArrayList;
import java.util.List;
public class SecuritySystem extends Device implements Switchable {
    private boolean isOn;
    private boolean alarmActive;
    private List<String> securityLogs;
    private String securityMode; // AWAY, HOME, DISARMED
    private static final String DEFAULT_MODE = "DISARMED";

    public SecuritySystem(String id, String name, String location) {
        super(id, name, location);
        this.isOn = false;
        this.alarmActive = false;
        this.securityLogs = new ArrayList<>();
        this.securityMode = DEFAULT_MODE;
    }

    @Override
    public void turnOn() {
        if (!isOn) {
            isOn = true;
            updateLastStateChange();
            System.out.println(getName() + " turned ON");
            addSecurityLog("System armed");
        }
    }

    @Override
    public void turnOff() {
        if (isOn) {
            isOn = false;
            updateLastStateChange();
            System.out.println(getName() + " turned OFF");
            addSecurityLog("System disarmed");
            deactivateAlarm(); // Ensure alarm is off when system is off
        }
    }

    @Override
    public boolean isOn() {
        return isOn;
    }

    public void setSecurityMode(String mode) {
        if (mode.equals("AWAY") || mode.equals("HOME") || mode.equals("DISARMED")) {
            this.securityMode = mode;
            addSecurityLog("Security mode changed to " + mode);
            System.out.println(getName() + " security mode set to " + mode);
        } else {
            System.out.println("Invalid security mode. Using default: " + DEFAULT_MODE);
            this.securityMode = DEFAULT_MODE;
        }
    }

    public String getSecurityMode() {
        return securityMode;
    }

    public void activateAlarm() {
        if (isOn && !alarmActive) {
            alarmActive = true;
            addSecurityLog("ALARM ACTIVATED!");
            System.out.println("ALARM ACTIVATED on " + getName() + "!");
        }
    }
    public void deactivateAlarm() {
        if (alarmActive) {
            alarmActive = false;
            addSecurityLog("Alarm deactivated");
            System.out.println("Alarm deactivated on " + getName());
        }
    }

    public boolean isAlarmActive() {
        return alarmActive;
    }

    public void detectMotion(String location) {
        if (isOn) {
            addSecurityLog("Motion detected in " + location);

            if (securityMode.equals("AWAY")) {
                // In AWAY mode, any motion triggers the alarm
                activateAlarm();
            } else if (securityMode.equals("HOME")) {
                // In HOME mode, only motion in certain areas triggers the alarm
                if (location.equals("Entrance") || location.equals("Window")) {
                    activateAlarm();
                }
            }
            // In DISARMED mode, just log the motion but don't trigger alarm
        }
    }

    private void addSecurityLog(String event) {
        String logEntry = java.time.LocalDateTime.now() + ": " + event;
        securityLogs.add(logEntry);
    }

    public List<String> getSecurityLogs() {
        return new ArrayList<>(securityLogs);
    }

    @Override
    public void setToDefaultSettings() {
        this.securityMode = DEFAULT_MODE;
        this.alarmActive = false;
        System.out.println(getName() + " set to default mode: " + DEFAULT_MODE);
    }

    @Override
    public String toString() {
        return super.toString() + " - Status: " + (isOn ? "ON" : "OFF") +
               ", Mode: " + securityMode + ", Alarm: " + (alarmActive ? "ACTIVE" : "Inactive");
    }
}
package smarthome.models;
import java.util.ArrayList;
import java.util.List;
public class User {
    private String username;
    private String password;
    private String name;
    private String role;
    private List<String> permissions;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
        this.name = username;
        this.role = "USER";
        this.permissions = new ArrayList<>();
        setupDefaultPermissions();
    }

    public User(String username, String password, String name) {
        this.username = username;
        this.password = password;
        this.name = name;
        this.role = "USER";
        this.permissions = new ArrayList<>();
        setupDefaultPermissions();
    }

    private void setupDefaultPermissions() {
        // Regular users can view devices and control them
        this.permissions.add("VIEW_DEVICES");
        this.permissions.add("CONTROL_DEVICES");
    }

    public String getUsername() {
        return username;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getRole() {
        return role;
    }

    public boolean hasPermission(String permission) {
        return permissions.contains(permission);
    }

    public void addPermission(String permission) {
        if (!permissions.contains(permission)) {
            permissions.add(permission);
        }
    }

    public void removePermission(String permission) {
        permissions.remove(permission);
    }

    public List<String> getPermissions() {
        return new ArrayList<>(permissions);
    }

    public boolean authenticate(String password) {
        return this.password.equals(password);
    }

    public void changePassword(String oldPassword, String newPassword) {
        if (authenticate(oldPassword)) {
            this.password = newPassword;
            System.out.println("Password changed successfully.");
        } else {
            System.out.println("Incorrect old password.");
        }
    }

    @Override
    public String toString() {
        return name + " (" + username + ")";
    }
}
package smarthome.system;
import smarthome.models.*;
import smarthome.exceptions.*;
import smarthome.interfaces.Switchable;
import java.util.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class SmartHomeSystem {
    private static SmartHomeSystem instance;
    private Map<String, Device> devices;
    private Map<String, User> users;
    private User currentUser;
    private boolean systemOn;
    private List<String> systemLogs;

    private SmartHomeSystem() {
        devices = new HashMap<>();
        users = new HashMap<>();
        systemOn = false;
        systemLogs = new ArrayList<>();

        // Add admin user by default
        users.put("admin", new AdminUser("admin", "admin123"));

        // Add a regular user for testing
        users.put("user", new User("user", "user123"));

        logSystemEvent("System initialized on " + LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));
    }

    public static SmartHomeSystem getInstance() {
        if (instance == null) {
            instance = new SmartHomeSystem();
        }
        return instance;
    }

    public void turnSystemOn() {
        this.systemOn = true;

        // Start all devices with default settings
        for (Device device : devices.values()) {
            if (device instanceof Switchable) {
                ((Switchable) device).turnOn();
                device.setToDefaultSettings();
            }
        }

        logSystemEvent("System turned ON by " + (currentUser != null ? currentUser.getUsername() : "SYSTEM"));
    }

    public void turnSystemOff() {
        this.systemOn = false;

        // Turn off all devices
        for (Device device : devices.values()) {
            if (device instanceof Switchable) {
                ((Switchable) device).turnOff();
            }
        }
        
        logSystemEvent("System turned OFF by " + (currentUser != null ? currentUser.getUsername() : 
"SYSTEM"));
    }
    
    public boolean isSystemOn() {
        return systemOn;
    }
    
    public void addDevice(Device device) throws AuthenticationException {
        // Check if user has permission to add devices
        if (currentUser == null) {
            throw new AuthenticationException("User not authenticated");
        }
        
        if (currentUser.hasPermission("ADD_DEVICE")) {
            devices.put(device.getId(), device);
            logSystemEvent("Device added: " + device.getName() + " by " + currentUser.getUsername());
        } else {
            throw new AuthenticationException("User does not have permission to add devices");
        }
    }
    
    public void removeDevice(String deviceId) throws DeviceNotFoundException, AuthenticationException {
        // Check if user has permission to remove devices
        if (currentUser == null) {
            throw new AuthenticationException("User not authenticated");
        }
        
        if (currentUser.hasPermission("REMOVE_DEVICE")) {
            if (devices.containsKey(deviceId)) {
                Device removed = devices.remove(deviceId);
                logSystemEvent("Device removed: " + removed.getName() + " by " + 
currentUser.getUsername());
            } else {
                throw new DeviceNotFoundException("Device with ID " + deviceId + " not found.");
            }
        } else {
            throw new AuthenticationException("User does not have permission to remove devices");
        }
    }
    
    public Device getDevice(String deviceId) throws DeviceNotFoundException {
        if (devices.containsKey(deviceId)) {
            return devices.get(deviceId);
        } else {
            throw new DeviceNotFoundException("Device with ID " + deviceId + " not found.");
        }
    }

    public List<Device> getAllDevices() {
        return new ArrayList<>(devices.values());
    }

    public void login(String username, String password) throws AuthenticationException {
        if (users.containsKey(username)) {
            User user = users.get(username);
            if (user.authenticate(password)) {
                currentUser = user;
                logSystemEvent("User logged in: " + username);
            } else {
                logSystemEvent("Failed login attempt for user: " + username);
                throw new AuthenticationException("Invalid password.");
            }
        } else {
            logSystemEvent("Failed login attempt for unknown user: " + username);
            throw new AuthenticationException("User not found.");
        }
    }

    public void logout() {
        if (currentUser != null) {
            logSystemEvent("User logged out: " + currentUser.getUsername());
            currentUser = null;
        }
    }

    public User getCurrentUser() {
        return currentUser;
    }

    public void addUser(User user) throws AuthenticationException {
        if (currentUser == null || !currentUser.hasPermission("MANAGE_USERS")) {
            throw new AuthenticationException("Only admin users can add new users.");
        }

        users.put(user.getUsername(), user);
        logSystemEvent("New user added: " + user.getUsername() + " by " + currentUser.getUsername());
    }

    public void handleMotionDetected(String locationName) {
        if (systemOn) {
            logSystemEvent("Motion detected in " + locationName);

            // Turn on lights in the location where motion is detected
```

```
      for (Device device : devices.values()) {
        if (device instanceof Light && device.getLocation().equals(locationName)) {
          Light light = (Light) device;
          if (light.isMotionActivated() && !light.isOn()) {
            light.activateByMotion();
            logSystemEvent("Turned on " + device.getName() + " due to motion detection");
          }
        }
        
        // If there's a security system, notify it about motion
        if (device instanceof SecuritySystem) {
          ((SecuritySystem)device).detectMotion(locationName);
        }
      }
    }
  }
 
  public void executeScheduledTasks() {
    if (!systemOn) return;
    
    LocalDateTime now = LocalDateTime.now();
    int dayOfWeek = now.getDayOfWeek().getValue() % 7; // 0 = Sunday, 6 = Saturday
    
    for (Device device : devices.values()) {
      for (ScheduledTask task : device.getScheduledTasks()) {
        if (task.isEnabled() && task.getDaysOfWeek()[dayOfWeek]) {
          // Check if it's time to execute the task
          if (task.getTime().getHour() == now.getHour() &&
            task.getTime().getMinute() == now.getMinute()) {
            task.execute();
            logSystemEvent("Executed scheduled task: " + task.getName() + " for " + device.getName());
          }
        }
      }
    }
  }
 
  private void logSystemEvent(String event) {
    String timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
    String logEntry = timestamp + " - " + event;
    systemLogs.add(logEntry);
    
    // Print to console for debugging
    System.out.println(logEntry);
  }
 
  public List<String> getSystemLogs() {
    // Only admin can access logs
    if (currentUser != null && currentUser.hasPermission("VIEW_LOGS")) {
      return new ArrayList<>(systemLogs);
    }
    return new ArrayList<>();
  }
}
```