**Smart Home System**

Smart Home System Project Presentation & OOP Implementation Analysis

**Project Overview**

A Java-based application that simulates controlling smart home devices using Object-Oriented Programming principles.

Table of Contents

1. Project Introduction

2. Class Diagram & Structure

3. OOP Principles Implementation

4. Code Explanation

5. Key Features

6. Output Screenshots

7. Conclusion

---

**Project Introduction**

**Objective**

- Create a smart home system to control various devices

- Demonstrate OOP concepts in real-world scenario

- Implement interface and abstract class usage

**Technologies Used**

- Java Programming Language

- OOP Principles

**Class Diagram & Structure**

┌─────────────────┐

│ Interface │

│ Control │

├─────────────────┤

│ + turnOn() │

│ + turnOff() │

└─────────────────┘

△

│ implements

┌─────────────────┐

│ Abstract Class │

│ Device │

├─────────────────┤

│ - deviceName │

│ - location │

├─────────────────┤

│ + showStatus() │

│ + getters │

└─────────────────┘

△

│ extends

┌─────────────────┐

│ Concrete Class│

│ Light │

├─────────────────┤

│ - isOn │

│ - brightness │

├─────────────────┤

│ + turnOn() │

│ + turnOff() │

│ + showStatus() │

│ + setBrightness()│

└─────────────────┘

**3. OOP Principles Implementation**

**ABSTRACTION**

Purpose: Hides complex implementation details, shows only essential features

// Interface abstraction

interface Control {

void turnOn();

void turnOff();

}

// Abstract class abstraction

abstract class Device {

public abstract void showStatus();

}

```

**ENCAPSULATION**

Purpose: Data hiding and controlled access through methods

private String deviceName;

private String location;

private boolean isOn;

private int brightness;

public String getDeviceName() { return deviceName; }

public String getLocation() { return location; }

**INHERITANCE**

class Light extends Device implements Control

- \*\*extends Device\*\*: Inherits deviceName, location, and methods

- \*\*implements Control\*\*: Implements interface methods

**POLYMORPHISM**

// Method overriding

@Override

public void showStatus() {

// Specific implementation for Light

}

// Runtime polymorphism

Device[] homeDevices = {

new Light("Living Room Light", "Living Room"),

new Light("Kitchen Light", "Kitchen")

};

**Key Features**

**Device Management**

- Add multiple devices to home system

- Track device status and location

**Control Operations**

- Turn devices ON/OFF

- Adjust brightness for lights

- Real-time status monitoring

**Smart Features**

- Brightness control with validation

- Error handling for invalid inputs

- User-friendly menu interface

**Extensibility**

- Easy to add new device types

- Modular design for maintenance

- Scalable architecture

**Menu System Flow**

SMART HOME SYSTEM

↓

========= MENU =========

1. Show Status of Devices

2. Turn On a Device

3. Turn Off a Device

4. Adjust Brightness

5. Exit

↓

User Choice → Corresponding Action

**OOP Benefits in This Project**

**Code Reusability**

- Common functionality in Device class

- Interface ensures consistent behavior

**Maintainability**

- Easy to modify specific device types

- Changes in base class affect all derived classes

**Scalability**

- Add new devices without changing existing code

- Extend functionality through inheritance

**Flexibility**

- Polymorphism allows treating objects generically

- Runtime binding for dynamic behavior

---

**Sample Code**

**import java.util.Scanner;**

**interface Control {**

**void turnOn();**

**void turnOff();**

**}**

**abstract class Device {**

**private String deviceName;**

**private String location;**

**public Device(String name, String location) {**

**this.deviceName = name;**

**this.location = location;**

**}**

**public abstract void showStatus();**

**public String getDeviceName() { return deviceName; }**

**public String getLocation() { return location; }**

**}**

**class Light extends Device implements Control {**

**private boolean isOn;**

**private int brightness;**

**public Light(String name, String location) {**

**super(name, location);**

**this.isOn = false;**

**this.brightness = 50;**

**}**

**@Override**

**public void showStatus() {**

**String status = isOn ? "ON" : "OFF";**

**System.out.println(getDeviceName() + " in " + getLocation() + " is " + status);**

**if (isOn) {**

**System.out.println("Brightness: " + brightness + "%");**

**}**

**}**

**@Override**

**public void turnOn() {**

**isOn = true;**

**System.out.println(getDeviceName() + " turned ON");**

**}**

**@Override**

**public void turnOff() {**

**isOn = false;**

**System.out.println(getDeviceName() + " turned OFF");**

**}**

**public void setBrightness(int level) {**

**if (isOn && level >= 0 && level <= 100) {**

**brightness = level;**

**System.out.println("Brightness set to " + level + "%");**

**} else if (!isOn) {**

**System.out.println("Turn on the light first before adjusting brightness!");**

**} else {**

**System.out.println("Brightness must be between 0 and 100!");**

**}**

**}**

**}**

**public class SmartHome{**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**System.out.println("        SMART HOME SYSTEM       \n");**

**Device[] homeDevices = {**

**new Light("Living Room Light", "Living Room"),**

**new Light("Kitchen Light", "Kitchen")**

**};**

**boolean running = true;**

**while (running) {**

**System.out.println("\n========= MENU =========");**

**System.out.println("1. Show Status of Devices");**

**System.out.println("2. Turn On a Device");**

**System.out.println("3. Turn Off a Device");**

**System.out.println("4. Adjust Brightness of a Light");**

**System.out.println("5. Exit");**

**System.out.print("Choose an option: ");**

**int choice = sc.nextInt();**

**sc.nextLine(); // consume newline**

**switch (choice) {**

**case 1:**

**System.out.println("\nDevice Status:");**

**for (Device device : homeDevices) {**

**device.showStatus();**

**}**

**break;**

**case 2:**

**System.out.println("Which device to turn ON?");**

**for (int i = 0; i < homeDevices.length; i++) {**

**System.out.println((i + 1) + ". " + homeDevices[i].getDeviceName());**

**}**

**int onChoice = sc.nextInt() - 1;**

**if (onChoice >= 0 && onChoice < homeDevices.length) {**

**((Control) homeDevices[onChoice]).turnOn();**

**}**

**break;**

**case 3:**

**System.out.println("Which device to turn OFF?");**

**for (int i = 0; i < homeDevices.length; i++) {**

**System.out.println((i + 1) + ". " + homeDevices[i].getDeviceName());**

**}**

**int offChoice = sc.nextInt() - 1;**

**if (offChoice >= 0 && offChoice < homeDevices.length) {**

**((Control) homeDevices[offChoice]).turnOff();**

**}**

**break;**

**case 4:**

**System.out.println("Which light to adjust brightness?");**

**for (int i = 0; i < homeDevices.length; i++) {**

**if (homeDevices[i] instanceof Light) {**

**System.out.println((i + 1) + ". " + homeDevices[i].getDeviceName());**

**}**

**}**

**int lightChoice = sc.nextInt() - 1;**

**if (lightChoice >= 0 && lightChoice < homeDevices.length && homeDevices[lightChoice] instanceof Light) {**

**System.out.print("Enter brightness (0–100): ");**

**int level = sc.nextInt();**

**((Light) homeDevices[lightChoice]).setBrightness(level);**

**}**

**break;**

**case 5:**

**running = false;**

**System.out.println("Exiting Smart Home System...");**

**break;**

**default:**

**System.out.println("Invalid choice! Please try again.");**

**}**

**}**

**sc.close();**

**}**

**}**

**Sample Output**

SMART HOME SYSTEM

========= MENU =========

1. Show Status of Devices

2. Turn On a Device

3. Turn Off a Device

4. Adjust Brightness of a Light

5. Exit

Choose an option: 1

Device Status:

Living Room Light in Living Room is OFF

Kitchen Light in Kitchen is OFF

---

**Future Enhancements**

**Potential Extensions**

1. Add more device types (Thermostat, SecurityCamera)

2. Implement scheduling features

3. Add energy consumption tracking

4. Create device groups and scenes

5. Implement remote control via network

**Additional Features**

- Database integration for persistence

- GUI interface

- Mobile app connectivity

- Voice control integration

---

**Conclusion**

**Key Takeaways**

- Successfully implemented core OOP principles

- Created extensible smart home system

- Demonstrated real-world application of interfaces and abstract classes

- Built user-friendly interaction system

**Project Value**

- Excellent example of OOP in practice

- Foundation for more complex home automation systems

- Demonstrates good software design principles

- Easily maintainable and extensible codebase

---

**Learning Outcomes**

1. Interface Implementation - Understanding contract-based programming

2. Abstract Classes - Creating base classes with common functionality

3. Inheritance - Building hierarchical relationships

4. Polymorphism - Implementing flexible method behavior

5. Encapsulation - Protecting data with access modifiers

This project serves as a comprehensive demonstration of Object-Oriented Programming concepts applied to a practical, real-world scenario.