EXERCISE:2

SMART HOME SYSTEMS

```
import java.util.*;

// BEHAVIORAL DESIGN PATTERN
interface Person {
    void update(String eventType, int deviceId);
}

// Device base class
abstract class Device {
    protected int id;
    protected String type;

    public Device(int id, String type) {
        this.id = id;
        this.type = type;
    }

    public abstract void turnOn();
    public abstract void turnOff();
    public abstract String getStatus();
}

// Concrete Device classes
class Light extends Device {
    private boolean isOn = false;

    public Light(int id) {
        super(id, "light");
    }

    public void turnOn() {
        isOn = true;
    }

    public void turnOff() {
        isOn = false;
    }

    public String getStatus() {
        return "Light " + id + " is " + (isOn ? "On" : "Off");
    }
}
```

```
class Thermostat extends Device {
    private int temp;

    public Thermostat(int id, int temp) {
        super(id, "thermostat");
        this.temp = temp;
    }

    public void turnOn() {
        // Do nothing
    }

    public void turnOff() {
        // Do nothing
    }

    public void set_Temp(int temp) {
        this.temp = temp;
    }

    public int get_Temp() {
        return temp;
    }

    public String getStatus() {
        return "Thermostat is set to " + temp+ " degrees";
    }
}

class DoorLock extends Device {
    private boolean isLocked = true;

    public DoorLock(int id) {
        super(id, "door");
    }

    public void turnOn() {
        isLocked = false;
    }

    public void turnOff() {
        isLocked = true;
    }

    public String getStatus() {
```

```
        return "Door is " + (isLocked ? "Locked" : "Unlocked");
    }
}
// STRUCTURAL AND CREATIVE DESIGN PATTERN
// Factory Method
class DeviceFactory {
    public static Device create(int id, String type) {
        switch (type) {
            case "light":
                return new Light(id);
            case "thermostat":
                return new Thermostat(id, 70);
            case "door":
                return new DoorLock(id);
            default:
                throw new IllegalArgumentException("Unknown device type");
        }
    }
}

// Proxy Pattern
class DeviceProxy {
    private Device device;

    public DeviceProxy(Device device) {
        this.device = device;
    }

    public void turnOn() {
        device.turnOn();
    }

    public void turnOff() {
        device.turnOff();
    }

    public String getStatus() {
        return device.getStatus();
    }
}

// Smart Home System
class SmartHomeSystem {
    private Map<Integer, DeviceProxy> devices = new HashMap<>();
    private List<Person> persons = new ArrayList<>();
    private List<String> s_Tasks = new ArrayList<>();
```

```
    private List<String> auto_Triggers = new ArrayList<>();

    public void addDevice(int id, String type) {
        Device device = DeviceFactory.create(id, type);
        devices.put(id, new DeviceProxy(device));
        notifyPerson("added", id);
    }

    public void removeDevice(int id) {
        devices.remove(id);
        notifyPerson("removed", id);
    }

    public void turnOn(int deviceId) {
        DeviceProxy device = devices.get(deviceId);
        if (device != null) {
            device.turnOn();
            notifyPerson("turned on", deviceId);
        } else {
            throw new IllegalArgumentException("Device with ID " + deviceId + " not found.");
        }
    }

    public void turnOff(int deviceId) {
        DeviceProxy device = devices.get(deviceId);
        if (device != null) {
            device.turnOff();
            notifyPerson("turned off", deviceId);
        } else {
            throw new IllegalArgumentException("Device with ID " + deviceId + " not found.");
        }
    }

    public String getStatus() {
        StringBuilder statusReport = new StringBuilder();
        for (DeviceProxy device : devices.values()) {
            statusReport.append(device.getStatus()).append(". ");
        }
        return statusReport.toString();
    }

    public void setSchedule(int deviceId, String time, String command) {
        s_Tasks.add("{device: " + deviceId + ", time: \"" + time + "\", command: \"" + command
+ "\"}");
    }
```

```
    public void addTrigger(String condition, String operator, int value, String action) {
        auto_Triggers.add("{condition: \"" + condition + " " + operator + " " + value + "\",
action: \"" + action + "\"}");
    }

    public String getScheduledTasks() {
        return s_Tasks.toString();
    }

    public String getAutomatedTriggers() {
        return auto_Triggers.toString();
    }

    public void addPerson(Person person) {
        persons.add(person);
    }

    public void notifyPerson(String eventType, int deviceId) {
        for (Person person : persons) {
            person.update(eventType, deviceId);
        }
    }
}

// Example Observer
class DevicePerson implements Person {
    public void update(String eventType, int deviceId) {
        System.out.println("Device " + deviceId + " has been " + eventType);
    }
}

// Main class to demonstrate functionality
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        SmartHomeSystem system = new SmartHomeSystem();
        DevicePerson person = new DevicePerson();
        system.addPerson(person);

        System.out.println("*** HELLO!!! ***");
        while (true) {
            System.out.println("Select an option:");
            System.out.println("1. Add Device");
            System.out.println("2. Remove Device");
            System.out.println("3. Turn On Device");
            System.out.println("4. Turn Off Device");
```

```
System.out.println("5. Set Schedule");
System.out.println("6. Add Trigger");
System.out.println("7. Get Status");
System.out.println("8. Get Scheduled Tasks");
System.out.println("9. Get Automated Triggers");
System.out.println("10. Exit");
System.out.print("Enter choice: ");
int choice = scanner.nextInt();
scanner.nextLine(); // consume the newline character

try {
   switch (choice) {
      case 1:
         System.out.print("Enter device ID: ");
         int addId = scanner.nextInt();
         scanner.nextLine(); // consume the newline
         System.out.print("Enter device type (light, thermostat, door): ");
         String type = scanner.nextLine();
         system.addDevice(addId, type);
         break;
      case 2:
         System.out.print("Enter device ID to remove: ");
         int removeId = scanner.nextInt();
         scanner.nextLine(); // consume the newline
         system.removeDevice(removeId);
         break;
      case 3:
         System.out.print("Enter device ID to turn on: ");
         int onId = scanner.nextInt();
         scanner.nextLine(); // consume the newline
         system.turnOn(onId);
         break;
      case 4:
         System.out.print("Enter device ID to turn off: ");
         int offId = scanner.nextInt();
         scanner.nextLine(); // consume the newline
         system.turnOff(offId);
         break;
      case 5:
         System.out.print("Enter device ID: ");
         int scheduleId = scanner.nextInt();
         scanner.nextLine(); // consume the newline
         System.out.print("Enter time (HH:MM): ");
         String time = scanner.nextLine();
         System.out.print("Enter command (TurnOn/TurnOff): ");
         String command = scanner.nextLine();
```

```
                    system.setSchedule(scheduleId, time, command);
                    break;
                case 6:
                    System.out.print("Enter condition (e.g., temperature): ");
                    String condition = scanner.nextLine();
                    System.out.print("Enter operator (e.g., >): ");
                    String operator = scanner.nextLine();
                    System.out.print("Enter value (e.g., 75): ");
                    int value = scanner.nextInt();
                    scanner.nextLine(); // consume the newline
                    System.out.print("Enter action (e.g., turnOff(1)): ");
                    String action = scanner.nextLine();
                    system.addTrigger(condition, operator, value, action);
                    break;
                case 7:
                    System.out.println(system.getStatus());
                    break;
                case 8:
                    System.out.println(system.getScheduledTasks());
                    break;
                case 9:
                    System.out.println(system.getAutomatedTriggers());
                    break;
                case 10:
                    System.out.println("Bye...!");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```