Import java.util.ArrayList;

Import java.util.List;

Interface Observer {

Void update();

}

Abstract class SmartDevice {

Private int id;

Private boolean status;

Private List<Observer> observers = new ArrayList<>();

Public SmartDevice(int id) {

This.id = id;

This.status = false;

}

Public int getId() {

Return id;

}

Public boolean getStatus() {

Return status;

}

Public void setStatus(boolean status) {

This.status = status;

notifyObservers();

}

Public void addObserver(Observer observer) {

Observers.add(observer);

}

Public void removeObserver(Observer observer) {

Observers.remove(observer);

}

Private void notifyObservers() {

For (Observer observer : observers) {

Observer.update();

}

}

Public abstract void turnOn();

Public abstract void turnOff();

}

Class Light extends SmartDevice {

Public Light(int id) {

Super(id);

}

@Override

Public void turnOn() {

setStatus(true);

System.out.println(“Light “ + getId() + “ turned ON.”);

}

@Override

Public void turnOff() {

setStatus(false);

System.out.println(“Light “ + getId() + “ turned OFF.”);

}

}

Class Thermostat extends SmartDevice {

Private int temperature;

Public Thermostat(int id, int temperature) {

Super(id);

This.temperature = temperature;

}

Public int getTemperature() {

Return temperature;

}

Public void setTemperature(int temperature) {

This.temperature = temperature;

notifyObservers();

}

@Override

Public void turnOn() {

setStatus(true);

System.out.println(“Thermostat “ + getId() + “ turned ON.”);

}

@Override

Public void turnOff() {

setStatus(false);

System.out.println(“Thermostat “ + getId() + “ turned OFF.”);

}

}

Class DoorLock extends SmartDevice {

Public DoorLock(int id) {

Super(id);

}

@Override

Public void turnOn() {

setStatus(true);

System.out.println(“Door “ + getId() + “ locked.”);

}

@Override

Public void turnOff() {

setStatus(false);

System.out.println(“Door “ + getId() + “ unlocked.”);

}

}

Class SmartDeviceFactory {

Public static SmartDevice createDevice(String type, int id, int... additionalParams) {

Switch (type.toLowerCase()) {

Case “light”:

Return new Light(id);

Case “thermostat”:

Return new Thermostat(id, additionalParams[0]);

Case “doorlock”:

Return new DoorLock(id);

Default:

Throw new IllegalArgumentException(“Unknown device type: “ + type);

}

}

}

Import java.util.\*;

Class Schedule {

Private SmartDevice device;

Private String time;

Private String command;

Public Schedule(SmartDevice device, String time, String command) {

This.device = device;

This.time = time;

This.command = command;

}

Public SmartDevice getDevice() {

Return device;

}

Public String getTime() {

Return time;

}

Public String getCommand() {

Return command;

}

}

Class Trigger {

Private String condition;

Private String action;

Public Trigger(String condition, String action) {

This.condition = condition;

This.action = action;

}

Public String getCondition() {

Return condition;

}

Public String getAction() {

Return action;

}

}

Class SmartHomeHub implements Observer {

Private List<SmartDevice> devices = new ArrayList<>();

Private List<Schedule> schedules = new ArrayList<>();

Private List<Trigger> triggers = new ArrayList<>();

Public void addDevice(SmartDevice device) {

Devices.add(device);

Device.addObserver(this);

}

Public void removeDevice(SmartDevice device) {

Devices.remove(device);

Device.removeObserver(this);

}

Public void addSchedule(Schedule schedule) {

Schedules.add(schedule);

}

Public void addTrigger(Trigger trigger) {

Triggers.add(trigger);

}

Public void executeCommand(String command) {

String[] parts = command.split(“[()]”);

String action = parts[0];

Int deviceId = Integer.parseInt(parts[1]);

SmartDevice device = devices.stream().filter(d -> d.getId() == deviceId).findFirst().orElse(null);

If (device != null) {

DeviceProxy proxy = new DeviceProxy(device);

Switch (action) {

Case “turnOn”:

Proxy.turnOn();

Break;

Case “turnOff”:

Proxy.turnOff();

Break;

Default:

System.out.println(“Unknown command: “ + action);

}

} else {

System.out.println(“Device not found: “ + deviceId);

}

}

Public void checkSchedules() {

// This method should be called periodically to check schedules

String currentTime = getCurrentTime();

For (Schedule schedule : schedules) {

If (schedule.getTime().equals(currentTime)) {

executeCommand(schedule.getCommand() + “(“ + schedule.getDevice().getId() + “)”);

}

}

}

Private String getCurrentTime() {

// Simplified to return a fixed time for simulation

Return “06:00”;

}

Public void checkTriggers() {

// Simplified trigger checking

For (Trigger trigger : triggers) {

String[] conditionParts = trigger.getCondition().split(“ “);

String deviceType = conditionParts[0];

String operator = conditionParts[1];

Int threshold = Integer.parseInt(conditionParts[2]);

For (SmartDevice device : devices) {

If (device instanceof Thermostat && deviceType.equals(“temperature”)) {

Thermostat thermostat = (Thermostat) device;

If (operator.equals(“>”) && thermostat.getTemperature() > threshold) {

executeCommand(trigger.getAction());

}

}

}

}

}

@Override

Public void update() {

// Handle updates from devices

}

Public void statusReport() {

For (SmartDevice device : devices) {

String status = device.getStatus() ? “On” : “Off”;

If (device instanceof Thermostat) {

Thermostat thermostat = (Thermostat) device;

System.out.println(“Thermostat “ + device.getId() + “ is set to “ + thermostat.getTemperature() + “ degrees.”);

} else if (device instanceof Light) {

System.out.println(“Light “ + device.getId() + “ is “ + status + “.”);

} else if (device instanceof DoorLock) {

Status = device.getStatus() ? “Locked” : “Unlocked”;

System.out.println(“Door “ + device.getId() + “ is “ + status + “.”);

}

}

}

}

Public class SmartHomeApp {

Public static void main(String[] args) {

SmartHomeHub hub = new SmartHomeHub();

// Initialize devices

SmartDevice light1 = SmartDeviceFactory.createDevice(“light”, 1);

SmartDevice thermostat1 = SmartDeviceFactory.createDevice(“thermostat”, 2, 70);

SmartDevice doorLock1 = SmartDeviceFactory.createDevice(“doorlock”, 3);

// Add devices to hub

Hub.addDevice(light1);

Hub.addDevice(thermostat1);

Hub.addDevice(doorLock1);

// Schedule a task

Hub.addSchedule(new Schedule(thermostat1, “06:00”, “turnOn”));

// Add a trigger

Hub.addTrigger(new Trigger(“temperature > 75”, “turnOff(1)”));

// Execute some commands

Hub.executeCommand(“turnOn(1)”);

Hub.executeCommand(“turnOff(3)”);

// Check schedules (simplified simulation)

Hub.checkSchedules();

// Check triggers (simplified simulation)

Hub.checkTriggers();

// Print status report

Hub.statusReport();

}

}