public interface Observer {

void update(boolean isOccupied);

}

import java.util.ArrayList;

import java.util.List;

public abstract class Subject {

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

public void attach(Observer observer) {

observers.add(observer);

}

public void detach(Observer observer) {

observers.remove(observer);

}

public void notifyObservers(boolean isOccupied) {

for (Observer observer : observers) {

observer.update(isOccupied);

}

}

}

public class OccupancySensor extends Subject {

private boolean isOccupied;

public void setOccupancy(boolean isOccupied) {

this.isOccupied = isOccupied;

notifyObservers(isOccupied);

}

public boolean isOccupied() {

return isOccupied;

}

}

public class AirConditioning implements Observer {

@Override

public void update(boolean isOccupied) {

if (isOccupied) {

System.out.println("Air Conditioning turned ON.");

} else {

System.out.println("Air Conditioning turned OFF.");

}

}

public class Lighting implements Observer {

@Override

public void update(boolean isOccupied) {

if (isOccupied) {

System.out.println("Lighting turned ON.");

} else {

System.out.println("Lighting turned OFF.");

}

}

}

public interface Command {

void execute();

}

public class BookRoomCommand implements Command {

private ConferenceRoomBookingSystem bookingSystem;

private String roomName;

public BookRoomCommand(ConferenceRoomBookingSystem bookingSystem, String roomName) {

this.bookingSystem = bookingSystem;

this.roomName = roomName;

}

@Override

public void execute() {

bookingSystem.bookRoom(roomName);

}

}

public class ChangeOccupancyCommand implements Command {

private OccupancySensor occupancySensor;

private boolean isOccupied;

public ChangeOccupancyCommand(OccupancySensor occupancySensor, boolean isOccupied) {

this.occupancySensor = occupancySensor;

this.isOccupied = isOccupied;

}

@Override

public void execute() {

occupancySensor.setOccupancy(isOccupied);

}

}

import java.util.HashMap;

import java.util.Map;

public class ConferenceRoomBookingSystem {

private Map<String, Boolean> rooms;

public ConferenceRoomBookingSystem() {

rooms = new HashMap<>();

rooms.put("Room1", false);

rooms.put("Room2", false);

}

public void bookRoom(String roomName) {

if (rooms.containsKey(roomName)) {

if (!rooms.get(roomName)) {

rooms.put(roomName, true);

System.out.println(roomName + " has been booked.");

} else {

System.out.println(roomName + " is already booked.");

}

} else {

System.out.println(roomName + " does not exist.");

}

}

public void releaseRoom(String roomName) {

if (rooms.containsKey(roomName)) {

if (rooms.get(roomName)) {

rooms.put(roomName, false);

System.out.println(roomName + " has been released.");

} else {

System.out.println(roomName + " is already free.");

}

} else {

System.out.println(roomName + " does not exist.");

}

}

}

import java.util.Scanner;

public class SmartOfficeApp {

public static void main(String[] args) {

ConferenceRoomBookingSystem bookingSystem = new ConferenceRoomBookingSystem();

OccupancySensor occupancySensor = new OccupancySensor();

AirConditioning airConditioning = new AirConditioning();

Lighting lighting = new Lighting();

occupancySensor.attach(airConditioning);

occupancySensor.attach(lighting);

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("1. Book Room");

System.out.println("2. Release Room");

System.out.println("3. Change Occupancy");

System.out.println("4. Exit");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.println("Enter room name to book:");

String roomToBook = scanner.nextLine();

Command bookRoomCommand = new BookRoomCommand(bookingSystem, roomToBook);

bookRoomCommand.execute();

break;

case 2:

System.out.println("Enter room name to release:");

String roomToRelease = scanner.nextLine();

bookingSystem.releaseRoom(roomToRelease);

break;

case 3:

System.out.println("Enter occupancy state (true/false):");

boolean isOccupied = scanner.nextBoolean();

Command changeOccupancyCommand = new ChangeOccupancyCommand(occupancySensor, isOccupied);

changeOccupancyCommand.execute();

break;

case 4:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Try again.");

}

}

}

}