Public class OfficeConfiguration {

Private static OfficeConfiguration instance;

Private int numberOfMeetingRooms;

Private OfficeConfiguration() {}

Public static synchronized OfficeConfiguration getInstance() {

If (instance == null) {

Instance = new OfficeConfiguration();

}

Return instance;

}

Public void setNumberOfMeetingRooms(int numberOfMeetingRooms) {

This.numberOfMeetingRooms = numberOfMeetingRooms;

}

Public int getNumberOfMeetingRooms() {

Return numberOfMeetingRooms;

}

}

Import java.util.ArrayList;

Import java.util.List;

Interface Observer {

Void update(boolean isOccupied);

}

Class Room {

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

Private boolean isOccupied = false;

Public void addObserver(Observer observer) {

Observers.add(observer);

}

Public void removeObserver(Observer observer) {

Observers.remove(observer);

}

Public void setOccupied(boolean occupied) {

isOccupied = occupied;

notifyObservers();

}

Public boolean isOccupied() {

Return isOccupied;

}

Private void notifyObservers() {

For (Observer observer : observers) {

Observer.update(isOccupied);

}

}

}

Class LightSystem implements Observer {

@Override

Public void update(boolean isOccupied) {

If (isOccupied) {

System.out.println(“Lights turned ON.”);

} else {

System.out.println(“Lights turned OFF.”);

}

}

}

Class AirConditioningSystem 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.”);

}

}

}

Interface Command {

Void execute();

}

Class BookRoomCommand implements Command {

Private Room room;

Public BookRoomCommand(Room room) {

This.room = room;

}

@Override

Public void execute() {

If (!room.isOccupied()) {

Room.setOccupied(true);

System.out.println(“Room booked.”);

} else {

System.out.println(“Room is already occupied.”);

}

}

}

Class CancelBookingCommand implements Command {

Private Room room;

Public CancelBookingCommand(Room room) {

This.room = room;

}

@Override

Public void execute() {

If (room.isOccupied()) {

Room.setOccupied(false);

System.out.println(“Booking cancelled.”);

} else {

System.out.println(“Room is not occupied.”);

}

}

}

Class CheckRoomStatusCommand implements Command {

Private Room room;

Public CheckRoomStatusCommand(Room room) {

This.room = room;

}

@Override

Public void execute() {

If (room.isOccupied()) {

System.out.println(“Room is occupied.”);

} else {

System.out.println(“Room is not occupied.”);

}

}

}

Public class SmartOfficeApp {

Public static void main(String[] args) {

// Office Configuration

OfficeConfiguration officeConfig = OfficeConfiguration.getInstance();

officeConfig.setNumberOfMeetingRooms(5);

// Create rooms

Room[] rooms = new Room[officeConfig.getNumberOfMeetingRooms()];

For (int i = 0; i < rooms.length; i++) {

Rooms[i] = new Room();

Rooms[i].addObserver(new LightSystem());

Rooms[i].addObserver(new AirConditioningSystem());

}

// Create commands

Command bookRoomCommand = new BookRoomCommand(rooms[0]);

Command cancelBookingCommand = new CancelBookingCommand(rooms[0]);

Command checkRoomStatusCommand = new CheckRoomStatusCommand(rooms[0]);

// Execute commands

bookRoomCommand.execute(); // Book the room

checkRoomStatusCommand.execute(); // Check room status

cancelBookingCommand.execute(); // Cancel the booking

checkRoomStatusCommand.execute(); // Check room status again

}

}