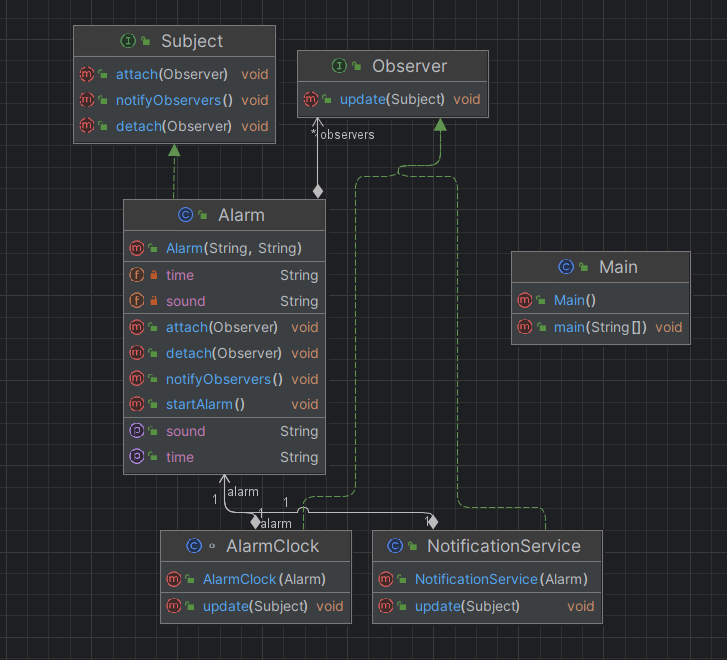
**Mẫu Observer:**

**Giới thiệu:**

Một đối tượng (Subject) thông báo cho các đối tượng khác (Observers) khi có sự thay đổi. Điều này giúp tách biệt giữa Subject và Observers, cho phép chúng hoạt động độc lập.

**Sau đây là ví dụ về mẫu Observer:**



Trong ví dụ này, đối tượng Alarm đóng vai trò là chủ đề, còn các đối tượng AlarmClock và NotificationService là những người quan sát.

**Cấu trúc:**

1. Interface Observer:

Giao diện này định nghĩa hợp đồng cho các đối tượng quan sát (observer).

Nó khai báo một phương thức update(Subject subject)

public interface Observer {

void update(Subject subject);

}

class AlarmClock implements Observer {

private Alarm alarm;

public AlarmClock(Alarm alarm) {

this.alarm = alarm;

alarm.attach(this);

}

*@Override*

public void update(Subject subject) {

Alarm alarm = (Alarm) subject;

System.***out***.println("Đồng hồ báo thức hiển thị: " + alarm.getTime() + " - " + alarm.getSound());

}

}

1. Interface Subject:

Giao diện này định nghĩa hợp đồng cho các đối tượng chủ đề (subject).

Nó khai báo ba phương thức.

public interface Subject {

void attach(Observer observer);

void detach(Observer observer);

void notifyObservers();

}

1. Class Alarm:

Lớp này đại diện cho đối tượng báo thức.

Nó lưu trữ thời gian và âm thanh của báo thức.

Nó duy trì một danh sách các đối tượng Observer đang quan sát nó.

Nó triển khai các phương thức của interface Subject.

public class Alarm implements Subject {

private String time;

private String sound;

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

public Alarm(String time, String sound) {

this.time = time;

this.sound = sound;

}

public String getTime() {

return time;

}

public String getSound() {

return sound;

}

*@Override*

public void attach(Observer observer) {

observers.add(observer);

}

*@Override*

public void detach(Observer observer) {

observers.remove(observer);

}

*@Override*

public void notifyObservers() {

for (Observer observer : observers) {

observer.update(this);

}

}

public void startAlarm() {

System.***out***.println("Báo thức " + time + " vang lên với âm thanh " + sound);

notifyObservers();

}

}

1. Class Main:

Lớp này là lớp chính nhưng không liên quan trực tiếp đến mẫu Observer.

public class Main {

public static void main(String[] args) {

System.***out***.println("Báo thức!");

}

}

1. Class NotificationService:

ớp này đại diện cho một dịch vụ nhận thông báo từ đối tượng Alarm.

Nó là một đối tượng Observer quan sát sự thay đổi trạng thái của đối tượng Alarm.

Nó triển khai phương thức update(Subject subject).

public class NotificationService implements Observer {

private Alarm alarm;

public NotificationService(Alarm alarm) {

this.alarm = alarm;

alarm.attach(this);

}

*@Override*

public void update(Subject subject) {

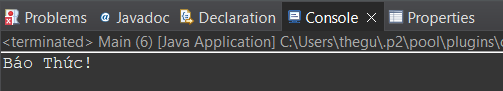
Alarm alarm = (Alarm) subject;

System.***out***.println("Gửi thông báo: Báo thức " + alarm.getTime() + " vang lên!");

}

}

**Kết quả và thử nghiệm:**

****

Khi báo thức reo (sự kiện thay đổi trạng thái), nó sẽ thông báo cho cả đồng hồ báo thức và dịch vụ thông báo, giúp chúng cập nhật hiển thị hoặc gửi thông báo đến người dùng.

**Lợi ích:**

Giảm thiểu sự phụ thuộc, khả năng tái sử dụng, tăng tính linh hoạt.

Mẫu Observer phù hợp cho các tình huống mà một đối tượng cần truyền tải thay đổi cho nhiều đối tượng khác.

Mẫu Observer thường được gọi là mô hình "đăng ký - xuất bản" hoặc "sự kiện".