

Họ và tên: Nguyễn Khánh Quy

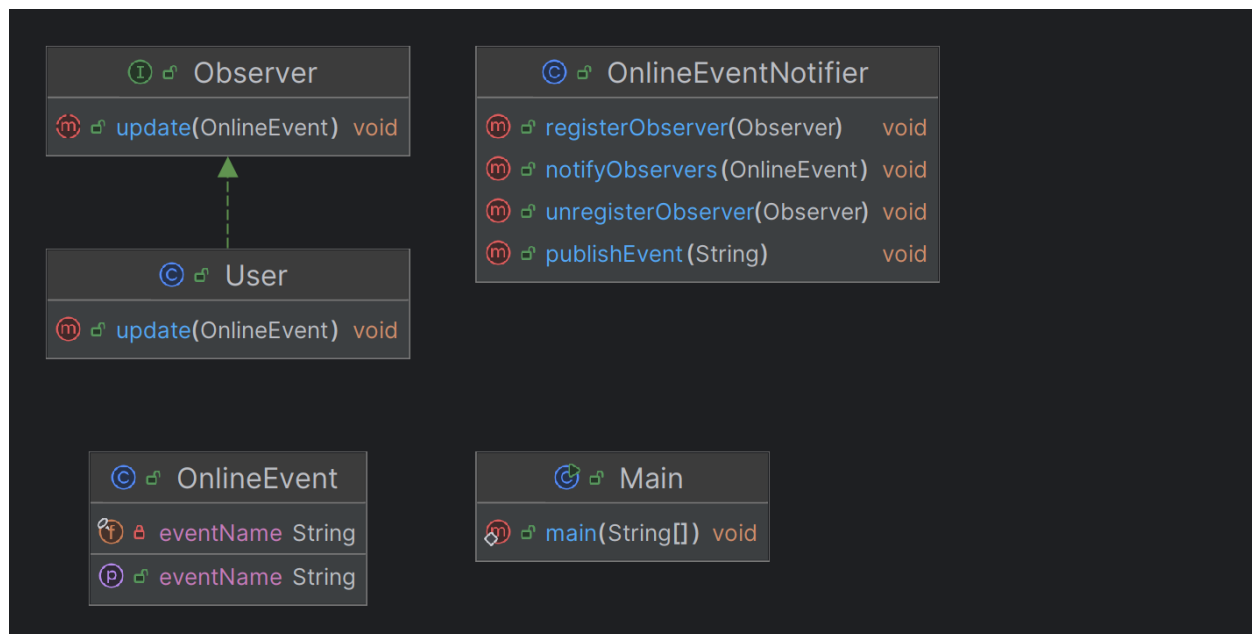
MSSV: 21110282

Observer Pattern

Mẫu **Observer** trong lập trình hướng đối tượng là một mẫu thiết kế hành vi được sử dụng để quản lý sự phụ thuộc giữa các đối tượng sao cho khi một đối tượng thay đổi trạng thái, tất cả các đối tượng quan sát nó sẽ nhận được thông báo và thực hiện các hành động tương ứng. Mẫu này giúp loại bỏ sự ràng buộc cứng nhắc giữa các đối tượng và tạo ra một cách giao tiếp linh hoạt giữa chúng.

Trong ví dụ này, chúng ta đã triển khai mẫu **Observer** bằng ngôn ngữ lập trình Java. Đối tượng **OnlineEventNotifier** đóng vai trò như một chủ đề hoặc đối tượng có thể quan sát, trong khi các đối tượng **User** được đăng ký là các người quan sát. Khi một sự kiện trực tuyến mới được phát sóng, **OnlineEventNotifier** thông báo cho tất cả các **User** đã đăng ký bằng cách gọi phương thức **update()**. Các **User** sau đó cập nhật thông tin tương ứng về sự kiện trực tuyến mà họ nhận được.

Mẫu **Observer** giúp tạo ra một cơ chế linh hoạt và mở rộng để quản lý sự kiện và thông báo trong hệ thống, đồng thời giảm thiểu sự phụ thuộc giữa các đối tượng.



Bước 1: Tạo lớp (OnlineEvent):

Đây là lớp đại diện cho một sự kiện trực tuyến. Lớp này có một trường eventName để lưu trữ tên của sự kiện.

```
public class OnlineEvent {

    private final String eventName;

    public OnlineEvent(String eventName) {
        this.eventName = eventName;
    }

    public String getEventName() {
        return eventName;
    }

}
```

Bước 2: Tạo Interface (Observer):

Đây là giao diện mà các lớp quan sát cần triển khai. Nó có một phương thức **update()** để cập nhật thông tin khi có sự kiện trực tuyến mới.

```
public interface Observer {  
    void update(OnlineEvent event);  
}
```

Bước 3: Tạo lớp (OnlineEventNotifier):

Đây là chủ đề hoặc đối tượng có thể quan sát. Nó duy trì một danh sách các người quan sát (**observers**). Các phương thức bao gồm:

registerObserver(): để đăng ký một người quan sát mới.

unregisterObserver(): để hủy đăng ký một người quan sát.

notifyObservers(): để thông báo cho tất cả người quan sát về sự kiện trực tuyến mới.

publishEvent(): để phát sóng một sự kiện trực tuyến mới bằng cách tạo một đối tượng `OnlineEvent` và gửi nó đến tất cả các người quan sát.

```
import java.util.ArrayList;  
import java.util.List;  
  
public class OnlineEventNotifier {  
    private final List<Observer> observers = new ArrayList<>();  
  
    // Phương thức để đăng ký người quan sát mới  
    public void registerObserver(Observer observer) {  
        observers.add(observer);  
    }  
  
    // Phương thức để hủy đăng ký người quan sát  
    public void unregisterObserver(Observer observer) {  
        observers.remove(observer);  
    }  
  
    // Phương thức để thông báo cho tất cả người quan sát về sự kiện trực  
    // tuyến mới  
    public void notifyObservers(OnlineEvent event) {
```

```

        for (Observer observer : observers) {
            observer.update(event);
        }
    }

    // Phương thức để gửi sự kiện trực tuyến mới cho người dùng
    public void publishEvent(String eventName) {
        OnlineEvent event = new OnlineEvent(eventName);
        notifyObservers(event);
    }
}

```

Bước 4: Tạo lớp (User):

Đây là lớp cụ thể của người quan sát. Nó triển khai phương thức **update()** để cập nhật thông tin khi có sự kiện trực tuyến mới.

```

public class User implements Observer {

    private final String name;

    public User(String name) {
        this.name = name;
    }

    // Phương thức để cập nhật thông tin khi có sự kiện trực tuyến mới
    @Override
    public void update(OnlineEvent event) {
        System.out.println(name + " received online event: " +
            event.getEventName());
    }
}

```

Bước 5: Thử nghiệm:

Trong hàm **main()**, chúng ta thực hiện các bước sau:

Tạo một đối tượng **OnlineEventNotifier**.

Đăng ký các người quan sát (**Observer**).

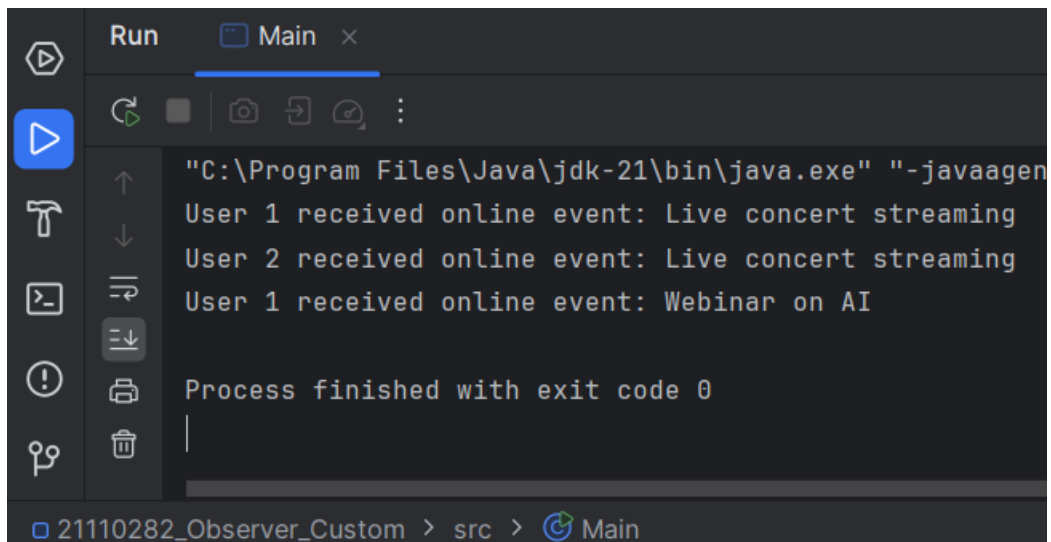
Phát sóng một sự kiện trực tuyến mới.

Hủy đăng ký một số người quan sát.

Phát sóng một sự kiện trực tuyến khác.

```
public class Main {  
  
    public static void main(String[] args) {  
        // Tạo một đối tượng OnlineEventNotifier (Chủ đề)  
        OnlineEventNotifier notifier = new OnlineEventNotifier();  
  
        // Đăng ký các người quan sát (Users)  
        Observer user1 = new User("User 1");  
        Observer user2 = new User("User 2");  
  
        notifier.registerObserver(user1);  
        notifier.registerObserver(user2);  
  
        // Phát sóng một sự kiện trực tuyến mới  
        notifier.publishEvent("Live concert streaming");  
  
        // Hủy đăng ký một số người quan sát  
        notifier.unregisterObserver(user2);  
  
        // Phát sóng một sự kiện trực tuyến khác  
        notifier.publishEvent("Webinar on AI");  
    }  
}
```

Kết quả:



```
Run  Main x  
C:\Program Files\Java\jdk-21\bin\java.exe "-javaagen  
User 1 received online event: Live concert streaming  
User 2 received online event: Live concert streaming  
User 1 received online event: Webinar on AI  
Process finished with exit code 0  
21110282_Observer_Custom > src > Main
```

Trong ví dụ này, chúng ta thực hiện mẫu thiết kế **Observer** để quản lý sự kiện trực tuyến. Đối tượng **OnlineEventNotifier** đóng vai trò là chủ đề, trong khi các đối

tượng **User** đóng vai trò là người quan sát. Khi một sự kiện trực tuyến mới được phát sóng, **OnlineEventNotifier** thông báo cho tất cả các người quan sát bằng cách gọi phương thức **update()** trên mỗi đối tượng người quan sát. Điều này giúp tạo ra một hệ thống linh hoạt, cho phép thêm và xóa người quan sát một cách dễ dàng mà không làm thay đổi cấu trúc của chủ đề hoặc người quan sát.