

Họ và tên: Nguyễn Khánh Quy

MSSV: 21110282

Visitor Pattern

Mẫu **Visitor** trong lập trình hướng đối tượng là một mẫu thiết kế hành vi được sử dụng để thực hiện một loạt các thao tác trên các đối tượng khác nhau mà không làm thay đổi cấu trúc của chúng. Điều này cho phép bạn thêm các chức năng mới mà không cần phải thay đổi lớp cơ sở của các đối tượng.

Dưới đây là một ví dụ về mẫu **Visitor** trong ngôn ngữ lập trình Java. Trong ví dụ này, chúng ta có các loại đối tượng **Human** (như Warrior và Wizard) và **Monster** (như CuteWolf và Goblin). Mỗi loại **Human** có khả năng tấn công các loại **Monster**, nhưng cách mà hành động tấn công được thực hiện khác nhau cho mỗi loại Monster.

Thay vì để **Human** trực tiếp thực hiện hành động tấn công trên Monster, chúng ta sử dụng mẫu **Visitor**. Cụ thể, **Human** gọi phương thức **hit(Monster monster)**, sau đó **Monster** sử dụng phương thức **hitBy** tương ứng với loại **Human** để xử lý hành động tấn công. Điều này giúp mỗi loại Monster có thể xử lý hành động tấn công từ các loại **Human** khác nhau một cách độc lập, mà không cần phải biết cụ thể về chúng.



Bước 1: Xác định Interface (Human và Monster):

Tạo Interface **Human** định nghĩa phương thức **hit(Monster monster)** để đại diện cho hành động của con người khi tấn công một sinh vật quái vật.

```
public interface Human {  
  
    void hit(Monster monster);  
  
}
```

Tạo Interface **Monster** định nghĩa hai phương thức **hitBy(Warrior warrior)** và **hitBy(Wizard wizard)** để xử lý hành động bị tấn công từ các loại con người khác nhau.

```
public interface Monster {  
  
    void hitBy(Warrior warrior);  
    void hitBy(Wizard wizard);  
  
}
```

Bước 2: Tạo các lớp (Warrior và Wizard) thực hiện giao diện (Human):

Các lớp này triển khai giao diện **Human** và ghi đè phương thức **hit(Monster monster)** để mô phỏng hành động tấn công.

```
public class Warrior implements Human {  
  
    public void hit(Monster monster) {  
        System.out.println("[Warrior]: Let me introduce you: my sword!!! ->  
[" + monster.getClass().getName() + "]);  
        monster.hitBy(this);  
    }  
  
}
```

```
public class Wizard implements Human {  
  
    public void hit(Monster monster) {  
        System.out.println("[Wizard]: Avada Kedavra! -> [" +  
monster.getClass().getName() + "]);  
    }  
  
}
```

```
        monster.hitBy(this);  
    }  
}
```

Bước 3: Tạo các lớp (CuteWolf và Goblin) thực hiện giao diện (Monster):

Các lớp này triển khai giao diện **Monster** và cung cấp cách xử lý khi bị tấn công bởi **Warrior** và **Wizard**.

```
public class CuteWolf implements Monster {  
  
    public void hitBy(Warrior warrior) {  
        damaged(150);  
    }  
  
    public void hitBy(Wizard wizard) {  
        damaged(110);  
    }  
  
    private void damaged(int hp) {  
        System.out.println("[Cute Wolf]: Woof! I lost " + hp + "hp");  
    }  
}
```

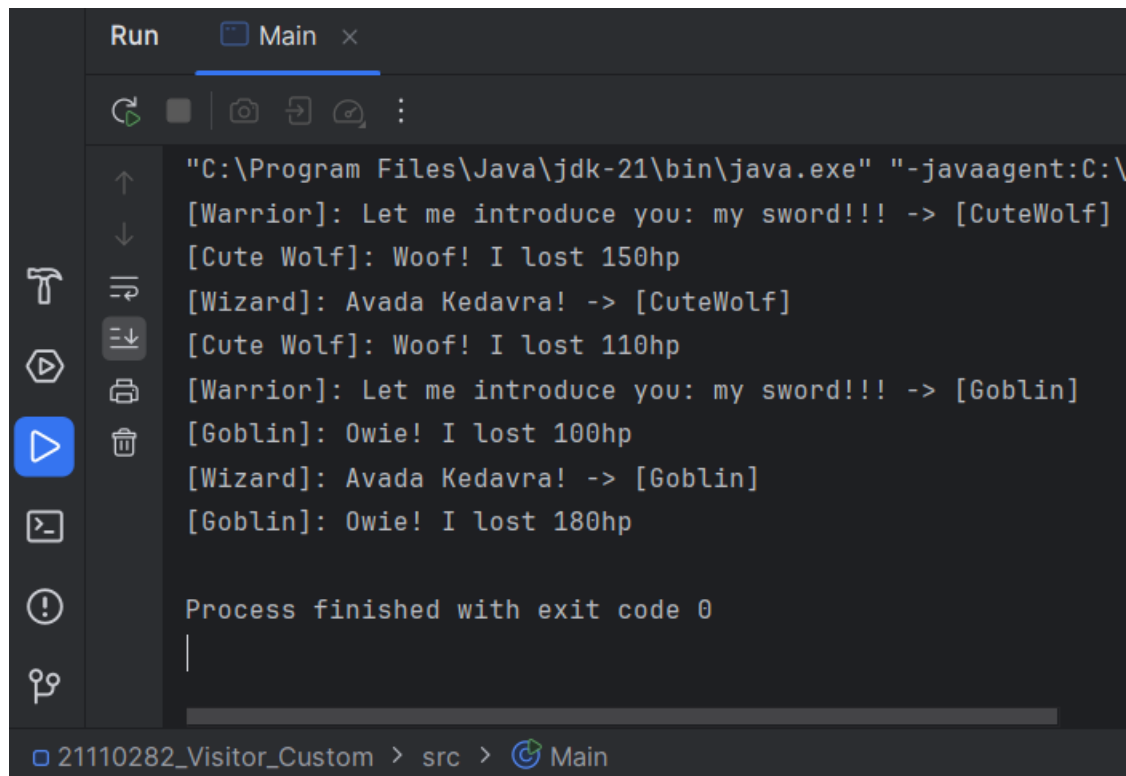
```
public class Goblin implements Monster {  
  
    public void hitBy(Warrior warrior) {  
        damaged(100);  
    }  
  
    public void hitBy(Wizard wizard) {  
        damaged(180);  
    }  
  
    private void damaged(int hp) {  
        System.out.println("[Goblin]: Owie! I lost " + hp + "hp");  
    }  
}
```

Bước 4: Thử nghiệm:

Trong phương thức **main**, chúng ta tạo ra các đối tượng con người (**warrior** và **wizard**) và các đối tượng quái vật (**wolf** và **goblin**), sau đó gọi phương thức **hit(Monster monster)** để mô phỏng hành động tấn công.

```
public class Main {  
  
    public static void main(String[] args) {  
        Human warrior = new Warrior();  
        Human wizard = new Wizard();  
  
        Monster wolf = new CuteWolf();  
        Monster goblin = new Goblin();  
  
        warrior.hit(wolf);  
        wizard.hit(wolf);  
  
        warrior.hit(goblin);  
        wizard.hit(goblin);  
    }  
}
```

Kết quả:



```
Run Main x
[C:\Program Files\Java\jdk-21\bin\java.exe] "-javaagent:C:\
[Warrior]: Let me introduce you: my sword!!! -> [CuteWolf]
[Cute Wolf]: Woof! I lost 150hp
[Wizard]: Avada Kedavra! -> [CuteWolf]
[Cute Wolf]: Woof! I lost 110hp
[Warrior]: Let me introduce you: my sword!!! -> [Goblin]
[Goblin]: Owie! I lost 100hp
[Wizard]: Avada Kedavra! -> [Goblin]
[Goblin]: Owie! I lost 180hp

Process finished with exit code 0
```

Trong ví dụ này, chúng ta áp dụng mẫu thiết kế **Visitor** để thực hiện một tác vụ trên một tập hợp các đối tượng khác nhau mà không cần thay đổi cấu trúc của chúng. Mục tiêu của mẫu này là tách biệt phần xử lý dữ liệu từ cấu trúc dữ liệu của nó, giúp cho việc thêm các phương thức mới vào các lớp đã tồn tại mà không cần phải sửa đổi chúng.