|  |  |
| --- | --- |
|  | **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC GIA ĐỊNH**  **KHOA CÔNG NGHỆ THÔNG TIN** |

**TIỂU LUẬN**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG NÂNG CAO**

**TÊN ĐỀ TÀI**

**COMMAND PATTERN**

**Sinh viên thực hiện: NHÓM 7**

**Giảng viên hướng dẫn: ThS LÊ HUỲNH PHƯỚC**

2008110143 – Vũ Trường Giang

2004110046 – Nguyễn Thành Đạt

1731103049 – Nguyễn Thiện Quang

**Lớp: K14DCPM01**

**Khóa: 14**

1. **Command Pattern là gì?**

**1.Định nghĩa**

– Command Pattern là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern). Nó cho phép chuyển yêu cầu thành đối tượng độc lập, có thể được sử dụng để tham số hóa các đối tượng với các yêu cầu khác nhau như log, queue (undo/redo), transtraction.

– Nói cho dễ hiểu, Command Pattern cho phép tất cả những Request gửi đến object được lưu trữ trong chính object đó dưới dạng một object Command. Khái niệm Command Object giống như một class trung gian được tạo ra để lưu trữ các câu lệnh và trạng thái của object tại một thời điểm nào đó.

– Command dịch ra nghĩa là ra lệnh. Commander nghĩa là chỉ huy, người này không làm mà chỉ ra lệnh cho người khác làm. Như vậy, phải có người nhận lệnh và thi hành lệnh. Người ra lệnh cần cung cấp một class đóng gói những mệnh lệnh. Người nhận mệnh lệnh cần phân biệt những interface nào để thực hiện đúng mệnh lệnh.

– Command Pattern còn được biết đến như là Action hoặc Transaction.

**2.Vị trí của Command Pattern**

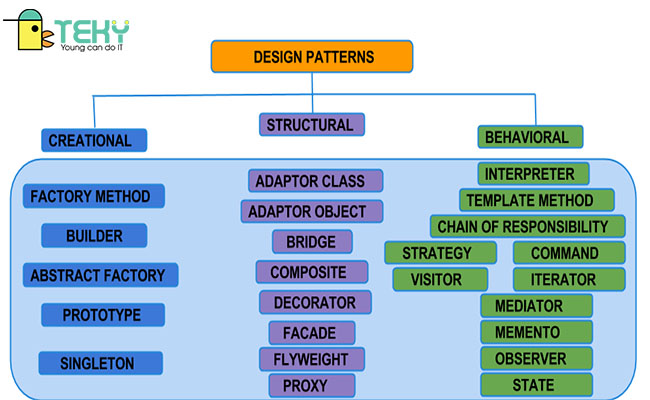
- Có 3 nhóm Design Pattern phổ biến là:

**+** Creational Patterns (nhóm khởi tạo): Nó cung cấp một giải pháp để tạo ra các object và che giấu được logic của việc tạo ra nó, thay vì tạo ra object một cách trực tiếp.

+ Structural Patterns (nhóm cấu trúc): Nó dùng để thiết lập, định nghĩa quan hệ giữa các đối tượng.

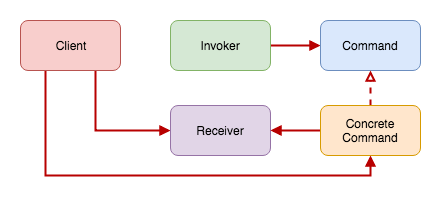
+Behavioral Patterns (nhóm hành vi): Nhóm này dùng trong thực hiện các hành vi của đối tượng.

-Command Pattern là một trong 23 design pattern Gang of Four nổi tiếng. Command pattern thuộc nhóm các pattern hành vi: Đóng gói tất cả thông tin cần thiết vào 1 đối tượng để thực hiện hành động hay kích hoạt một sự kiện thực hiện sau đó. Các thông tin có thể bao gồm tên phương thức, các biến và giá trị cần thiết...hay đơn giản hơn đó là nó cho phép chuyển yêu cầu thành đối tượng độc lập, có thể được sử dụng để tham số hóa các đối tượng với các yêu cầu khác nhau như log, queue (undo/redo), transtraction.



1. **các thành phần cần có của một command pattern.**

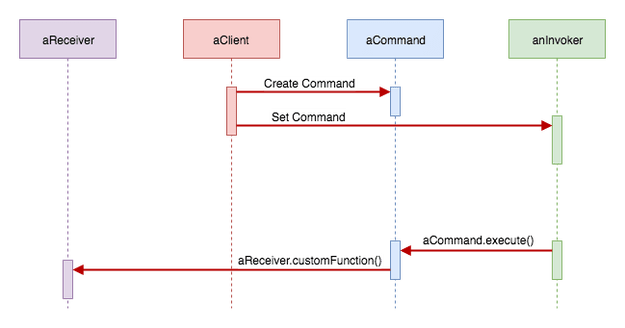
**1.Thành phần**



Trong đó:

* **Command** : là một interface hoặc abstract class, chứa một phương thức trừu tượng thực thi (execute) một hành động (operation). Request sẽ được đóng gói dưới dạng Command.
* **ConcreteCommand** : là các implementation của Command. Định nghĩa một sự gắn kết giữa một đối tượng Receiver và một hành động. Thực thi execute() bằng việc gọi operation đang hoãn trên Receiver. Mỗi một ConcreteCommand sẽ phục vụ cho một case request riêng.
* **Client** : tiếp nhận request từ phía người dùng, đóng gói request thành ConcreteCommand thích hợp và thiết lập receiver của nó.
* **Invoker** : tiếp nhận ConcreteCommand từ Client và gọi execute() của ConcreteCommand để thực thi request.
* **Receiver** : đây là thành phần thực sự xử lý business logic cho case request. Trong phương thức execute() của ConcreteCommand chúng ta sẽ gọi method thích hợp trong Receiver.

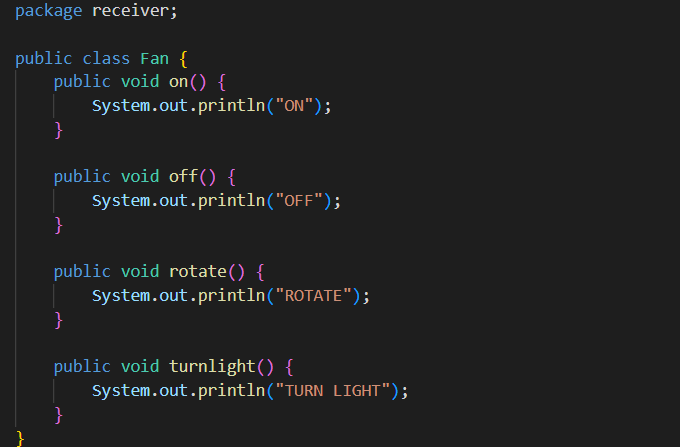
đây là sequence diagram của command pattern.



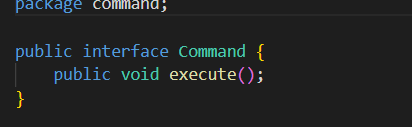
Client và Invoker sẽ thực hiện việc tiếp nhận request. Còn việc thực thi request sẽ do Command, ConcreteCommand và Receiver đảm nhận.

**2.Cách sử dụng**

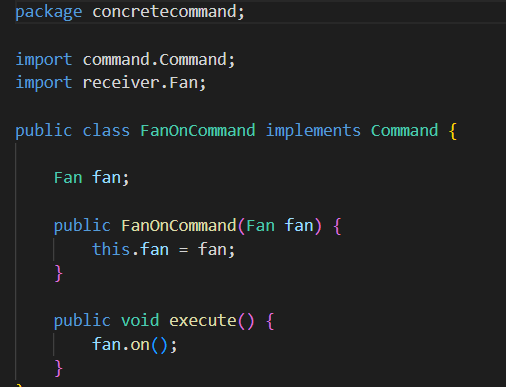
Để hiểu cách sử dụng nó chúng ta sẽ thông qua 1 ví dụ như sau: Cùng khởi tạo playground để demo cách sử dụng: Chúng ta có 1 đối tượng Fan có 4 phương thức là switchOn và switchOff, RotateCommand, TurnlightCommand: Đây đóng vai trò là 1 **class request**

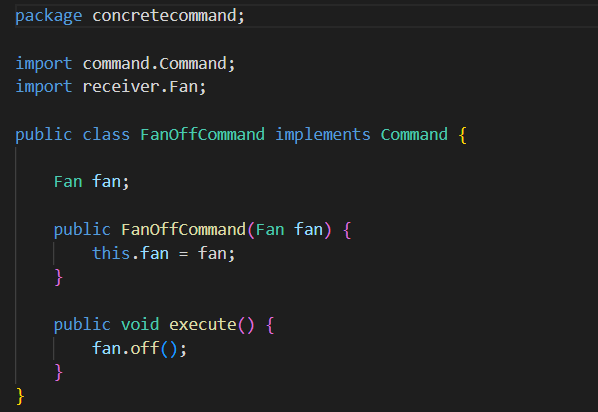


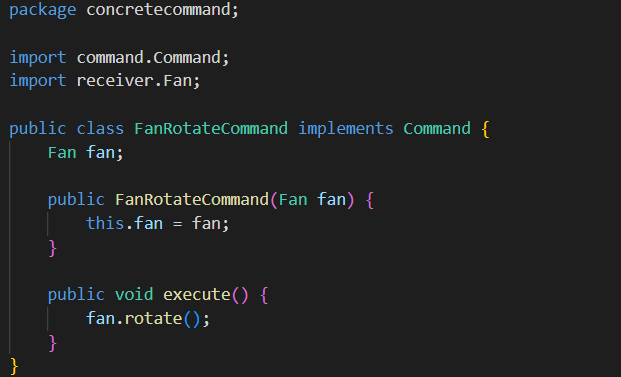
Như đã mô tả ở phần trên thì Command pattern cần có 5 thành phần vì vậy chúng ta sẽ từng bước xây dựng nó. Chúng ta xây dựng 1 interface tên là **Command** không trực tiếp tắt bật quạt mà chỉ ra lệnh cho fan on or off.

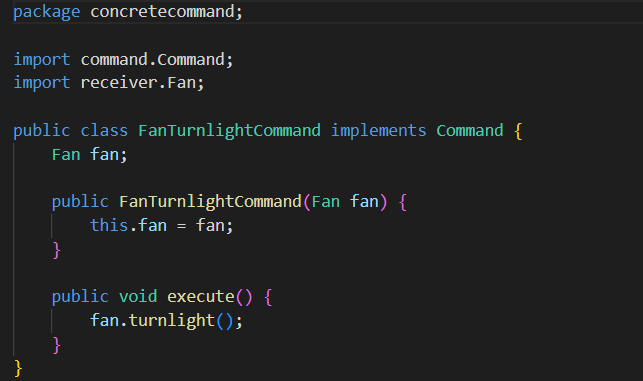


sau đó chúng ta sẽ khởi tạo 4 class FanCommandOff và FanCommandOn implement Command và Fan RotateCommand, FanTurnlightCommand implement Command: 4 class này chính là **ConcreteCommand**

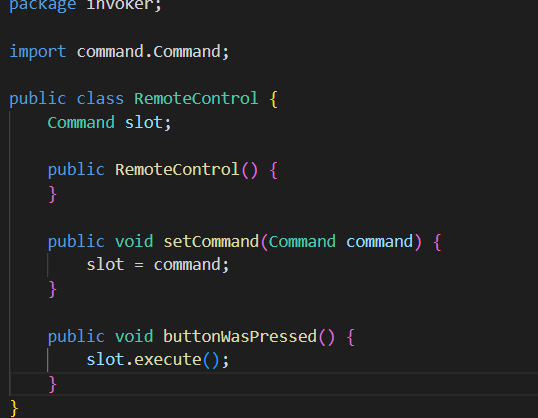




****

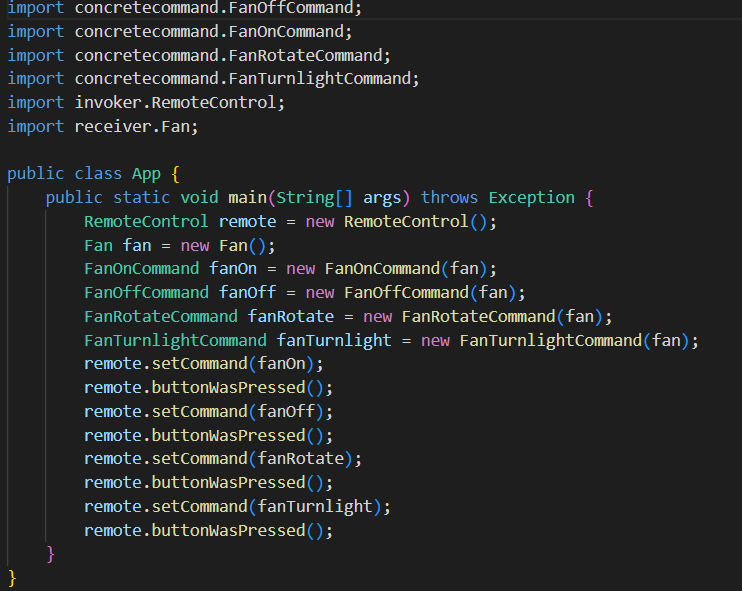
****

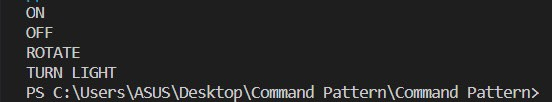
Tiếp theo tiến hành đóng gói các command này vào trong 1 bộ điều khiển gọi là Remote Control: class này đóng vai trò là **invoker**

****

Vậy thì với RemoteControll chúng ta có thể truyền các command khác nhau vào để thực hiện các chức năng riêng biệt khác nhau. Re quest được đóng giói dưới dạng object đúng như trong tư tưởng của pattern.

Chúng ta tạo 1 **client** để thực thi và xem kết quả:

****



1. **Mục đích ra đời**

Trong thiết kế hướng đối tượng – OOP, đôi khi chúng ta cần gửi các requests cho các Objects mà không biết bất cứ điều gì về hoạt động được yêu cầu hoặc người nhận yêu cầu. Chẳng hạn chúng có một ứng dụng văn bản, khi click lên button undo/ redo, save, … yêu cầu sẽ được chuyển đến hệ thống xử lý, chúng ta sẽ không thể biết được object nào sẽ nhận xử lý, cách nó thực hiện như thế nào. Command Pattern là một pattern được thiết kế cho những ứng dụng như vậy, giúp chúng ta:

* Tránh các hard-wired(kết nối cứng). Việc triển khai hard-wired vào 1 lớp là không linh hoạt.
* Không nên để lớp đối tượng phụ thuộc cụ thể vào một yêu cầu nào đó
* Cần đưa ra yêu cầu cho đối tượng mà không cần biết bất cứ gì về hoạt động được yêu cầu cũng như cụ thể nơi nhận yêu cầu.

1. **Ưu & nhược điểm**

**Ưu điểm**

* Đảm bảo nguyên tắc Single Responsibility
* Đảm bảo nguyên tắc Open/Closed
* Có thể thực hiện hoàn tác
* Giảm kết nối phụ thuộc giữa Invoker và Receiver
* Cho phép đóng gói yêu cầu thành đối tượng, dễ dàng chuyển dữ liệu giữa các thành phần hệ thống

**Nhược điểm**

* Khiến code trở nên phức tạp hơn, sinh ra các lớp mới gây phức tạp cho mã nguồn.