Design pattern luôn là một lĩnh vực khá khó nhằn và hơi trừu tượng. Trong bài viết này, mình sẽ giới thiệu về một pattern có tên là Command, với ví dụ trong bài viết hi vọng giúp các bạn dễ hiểu hơn về pattern này.

**Giới thiệu**

**Command** là một trong 23 design pattern Gang of Four nổi tiếng. Command thuộc nhóm các pattern hành vi.

Gang of Four chỉ ra mục đích sử dụng của pattern Command như sau:

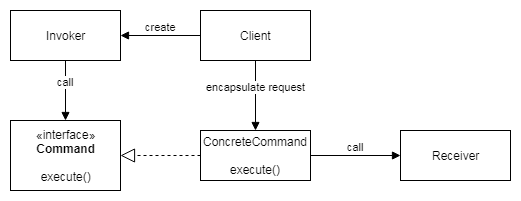
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Đoạn trên xem ra có vẻ hơi trừu tượng. Nhưng chúng ta có thể hiểu đơn giản như thế này: request từ phía client sẽ được đóng gói dưới dạng một đối tượng *command* và đối tượng *command* này sẽ được chuyển cho một thành phần riêng biệt để thực thi (gửi trực tiếp hoặc đẩy vào một queue để lưu vết). Command pattern sẽ phân tách quá trình tiếp nhận request với quá trình thực thi request. Chính vì vậy, công việc xử lý request sẽ trở nên linh hoạt và dễ nâng cấp hơn. Điều này cũng giống như trong thực tế, người phục vụ sẽ chỉ quan tâm đến món ăn mà thực khách yêu cầu, còn việc chế biến món ăn ấy ra sao thì đã có đầu bếp của nhà hàng lo rồi.

Bên cạnh đó, trong Command pattern, chúng ta có thể lưu vết các command đã thực hiện. Nhờ điều này mà Command pattern còn được áp dụng cho tác vụ undo/redo trong các ứng dụng thực tế.

**Cấu trúc**

UML của Command pattern:

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/zxhj96zg4_201901220636.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/zxhj96zg4_201901220636.png)

Command pattern gồm 5 thành phần chính:

* **Command**: đây có thể là một interface hoặc abstract class, chứa một abstract method execute(). Request sẽ được đóng gói dưới dạng Command.
* **Concrete Command**: là các implementation của Command, mỗi một Concrete Command sẽ phục vụ cho một case request riêng.
* **Receiver**: đây mới là thành phần thực sự xử lý business logic cho case request. Trong method execute() của Concrete Command, chúng ta sẽ gọi method thích hợp trong Receiver.
* **Client**: tiếp nhận request từ phía người dùng và đóng gói request thành Concrete Command thích hợp.
* **Invoker**: tiếp nhận Concrete Command từ Client và gọi execute() của Concrete Command để thực thi request. Việc lưu vết command sẽ được thực hiện ở trong Invoker.

**Ví dụ**

Trong ví dụ này, chúng ta sẽ thử áp dụng pattern Command vào bài toán order món ăn tại một nhà hàng. Đầu tiên, khách hàng (customer) sẽ order món ăn. Với từng món ăn (steak hoặc pizza) thì người phục vụ (waiter) sẽ chuyển order cho đầu bếp (chef) chuyên chế biến món đó. Với bài toán này, chúng ta có thể thấy:

* Order sẽ đóng vai trò Command.
* Steak và Pizza chính là các Concrete Command.
* Steak Chef và Pizza Chef đóng vai trò Receiver.
* Customer đóng vai trò Client.
* Waiter sẽ là cầu nối giữa Customer và Chef: Invoker.

Dưới đây là chi tiết cài đặt cho bài toán này:

Order.java

/\*\*

\* Command

\*/

public interface Order {

void execute();

}

Pizza.java

/\*\*

\* Concrete command

\*/

public class Pizza implements Order {

private PizzaChef chef;

public Pizza() {

this.chef = new PizzaChef();

}

@Override

public void execute() {

chef.makePizza();

}

@Override

public String toString() {

return "pizza";

}

}

Steak.java

/\*\*

\* Concrete command

\*/

public class Steak implements Order {

private SteakChef chef;

public Steak() {

this.chef = new SteakChef();

}

@Override

public void execute() {

chef.makeSteak();

}

@Override

public String toString() {

return "steak";

}

}

PizzaChef.java

/\*\*

\* Receiver

\*/

public class PizzaChef {

public void makePizza() {

System.out.println("I'm making pizza");

}

}

SteakChef.java

/\*\*

\* Receiver

\*/

public class SteakChef {

public void makeSteak() {

System.out.println("I'm making steak");

}

}

Waiter.java

/\*\*

\* Invoker

\*/

public class Waiter {

public void takeOrder(Order order) {

System.out.println("Waiter take new order " + order.toString());

order.execute();

}

}

Customer.java

/\*\*

\* Client

\*/

public class Customer {

private Waiter waiter;

public Customer(Waiter waiter) {

this.waiter = waiter;

}

public void request(String request) {

Order order;

if (request.toLowerCase().equals("pizza")) {

order = new Pizza();

} else {

order = new Steak();

}

waiter.takeOrder(order);

}

}

Main.java

public class Main {

public static void main(String[] args) {

Waiter waiter = new Waiter();

Customer customer = new Customer(waiter);

customer.request("pizza");

customer.request("steak");

}

}

Chạy file Main.java, chúng ta sẽ có kết quả như sau: