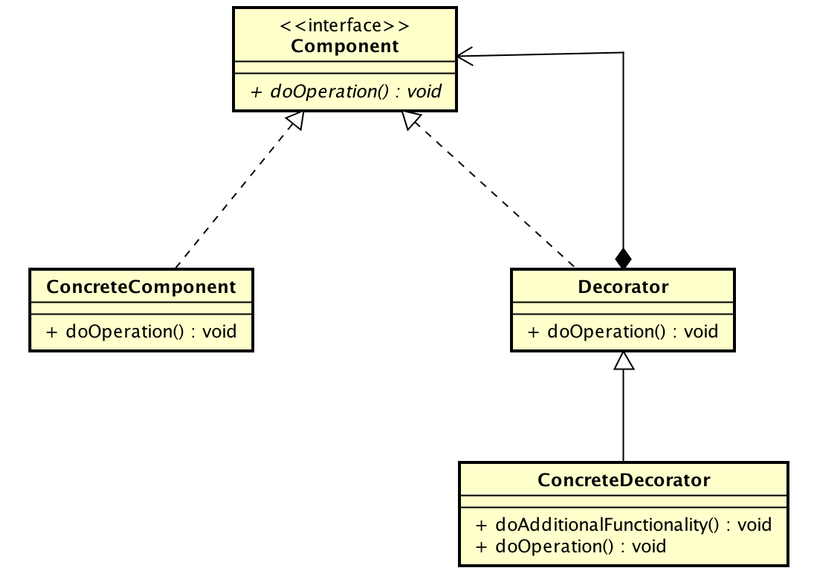
**DECORATOR**

1. **Khái niệm của Decorator Pattern**

* Decorator thuộc nhóm cấu trúc (Structural pattern), là những mẫu thiết kế liên quan đến cách tổ hợp các class và object để tạo thành các cấu trúc lớn hơn.
* Mẫu thiết kế Decorator giúp bổ sung thêm các trạng thái (các thuộc tính) và hoạt động (các phương thức) mới vào một object sẵn có.
* Khác với kế thừa, Decorator cho phép thay đổi những tính chất đã có trong một đối tượng trong khi chương trình đang chạy mà không làm ảnh hưởng đến các tính chất đã tồn tại của các đối tượng khác. Trong khi đó, với kế thừa thì ta sẽ phải triển khai code để mở rộng bằng cách *override*, còn với Decorator sẽ cho phép người dùng thay đổi một đối tượng đã tồn tại nhưng không làm ảnh hưởng đến các đối tượng cùng lớp đó.

1. **Biểu đồ lớp và cách thức hoạt động của Decorator**

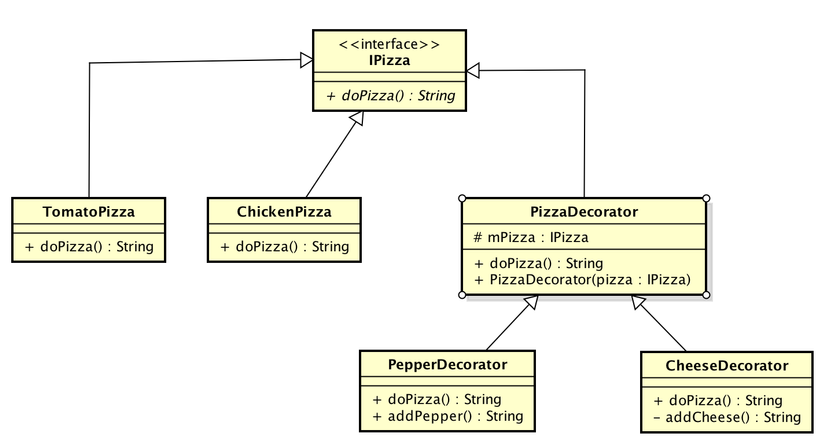


Những thành phần trong mẫu thiết kế của Decorator:

* Component : một interface chung để các đối tượng cần thêm chức năng trong quá trình chạy thì sẽ triển khai interface này.
* ConcreteComponent : một cái đặt giao diện Component mà nó định nghĩa một đối tượng cần thêm chức năng trong quá trình chạy.
* Decorator : một lớp trừu tượng dùng để duy trì một tham chiếu của đối tượng thành phần và đồng thời cài đặt các thành phần của giao diện.
* ConcreteDecorator : một cài đặt của Decorator, nó cài đặt thêm các thành phần vào đầu của các đối tượng thành phần.

1. **Ví dụ**

Ta sẽ lấy ví dụ về món bánh pizza: Giả sử một cửa hàng bánh Pizza, họ có làm bánh pizza cà chua và pizza gà. Tuy nhiên, do khách hang được phép lựa chọn thêm một vài nguyên liệu, giả sử như cà chua và hồ tiêu, khi đó nhà hàng sẽ có khoảng 5 loại pizza khác nhau. Trong trường hợp này, nếu sử dụng mở rộng tĩnh (kế thừa ) thì ta sẽ phải tạo một số lượng lớn các lớp như PepperChickenPizza, TomatoPepperPizza, CheeseChickenPizza, … Mẫu thiết kế Decorator sẽ giúp ta giảm được một lượng lớn các lớp mở rộng. Về cơ bản thì ta sẽ triển khai như sau:



* Ta có interface IPizza với phương thức *doPizza()* được khai báo nguyên mẫu.
* Hai lớp TomatoPizza và ChickenPizza là hai sản phẩm pizza mà nhà hàng đã sẵn có
* PizzaDecorator là một lớp trừu tượng, chứa thuộc tính mPizza là một đối tượng IPizza, được khởi tạo tại hàm tạo của lớp. Giả sử ta muốn thêm một thành phần vào Pizza gà, ta sẽ gọi như sau:

ChickenPizza chickenPizza = new ChickenPizza();

PizzaDecorator pd = new PizzaDecorator(chickenPizza);

* Hai lớp PepperDecorator và CheeseDecorator kế thừa từ lớp PizzaDecorator, cho phép ta cài đặt thêm các tính năng, như ở đây là addPepper() và addCheese()

1. **Lợi ích của decorator pattern**

* Tăng cường khả năng mở rộng của đối tượng, bởi vì những thay đổi được thực hiện bằng cách implement trên các lớp mới.
* Client sẽ không nhận thấy sự khác biệt khi bạn đưa cho nó một wrapper thay vì đối tượng gốc.
* Một đối tượng có thể được bao bọc bởi nhiều wrapper cùng một lúc.
* Cho phép thêm hoặc xóa tính năng của một đối tượng lúc thực thi (run-time).