**COMPOSITE**

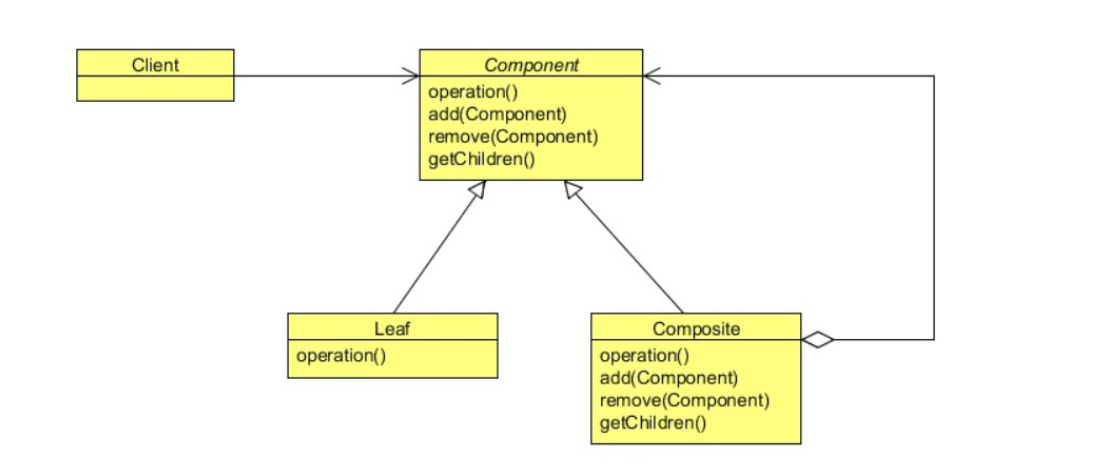
1. **Giới thiệu**

* Composite là một mẫu thiết kế thuộc nhóm cấu trúc (Structural Pattern). Composite Pattern là một sự tổng hợp những thành phần có quan hệ với nhau để tạo ra thành phần lớn hơn. Nó cho phép thực hiện các tương tác với tất cả đối tượng trong mẫu tương tự nhau.
* Tần suất sử dụng: khá cao

1. **Mục đích ra đời**

* Composite Pattern được sử dụng khi chúng ta cần xử lý một nhóm đối tượng tương tự theo cách xử lý 1 object. Composite pattern sắp xếp các object theo cấu trúc cây để diễn giải 1 phần cũng như toàn bộ hệ thống phân cấp. Pattern này tạo một lớp chứa nhóm đối tượng của riêng nó. Lớp này cung cấp các cách để sửa đổi nhóm của cùng 1 object. Pattern này cho phép Client có thể viết code giống nhau để tương tác với composite object này, bất kể đó là một đối tượng riêng lẻ hay tập hợp các đối tượng.
* Ví dụ: bạn có hai loại đối tượng: Sản phẩm và Hộp. Một Hộp có thể chứa nhiều Sản phẩm cũng như một số Hộp nhỏ hơn. Những chiếc Hộp nhỏ này có thể cũng giữ một số Sản phẩm hoặc thậm chí các Hộp nhỏ hơn, v.v.
* Giả sử bạn quyết định tạo một hệ thống đặt hàng. Làm thế nào bạn sẽ xác định tổng giá của một đơn đặt hàng như vậy? Bạn có thể mở tất cả các hộp, xem qua tất cả các sản phẩm và sau đó tính tổng. Nhưng cách tiếp cận này khi thực thi chương trình đòi hỏi mức độ lồng ghép và cấu trúc phức tạp.
* Giải pháp: Mẫu Composite cho phép làm việc với Sản phẩm và Hộp thông qua một giao diện chung khai báo một phương pháp tính tổng giá. Đối với một hộp, nó sẽ đi qua từng mục trong hộp chứa, hỏi giá của nó và sau đó trả lại tổng giá cho hộp này. Lợi ích lớn nhất của phương pháp này là không cần phải quan tâm đến các lớp cụ thể của các đối tượng tạo ra và xử lý chúng với cùng một phương thức.

1. **Sơ đồ Composite UML**



Trong biểu đồ trên thì:

* **Component** (thành phần):
  + Khai báo interface hoặc abtract chung cho các thành phần đối tượng.
  + Chứa các method thao tác chung của các thành phần đối tượng.
* **Leaf** (Lá):
  + Biểu diễn các đối tượng lá trong thành phần đối tượng.
* **Composite** (hỗn hợp):
  + Định nghĩa một thao tác cho các thành phần có thành phần con.
  + Lưu trữ các thành con.
  + Thực thi sự quản lý các thành phần con của Component.
* **Client**:
  + Điều khiển các đối tượng trong Composite thông qua các method trong Component.

1. **Ưu & Nhược điểm**
2. **Ưu điểm**

* Bạn có thể làm việc với các cấu trúc cây phức tạp thuận tiện hơn.
* Nguyên tắc mở/ đóng: có thể khởi tạo các loại phần tử mới vào ứng dụng mà không phá vỡ code hiện có đang hoạt động với đối tượng cây.

1. **Nhược điểm**

* Code có thể trở nên phức tạp hơn mức bình thường, vì có rất nhiều interfaces và classes được khởi tạo cùng với mẫu.

1. **Khi nào sử dụng**

* Khi bạn muốn tạo ra các đối tượng trong các cấu trúc cây để biểu diễn hệ thống phân lớp.
* Có thể khó cung cấp một interface chung cho các lớp có chức năng khác nhau quá nhiều. Trong một số trường hợp nhất định, bạn cần tổng quát hóa quá mức interface thành phần khiến nó khó hiểu hơn.

1. **Khi nào không sử dụng**

* **Composite Design Pattern** làm cho việc hạn chế loại thành phần của hỗn hợp khó hơn. Vì vậy, nó không nên được sử dụng khi bạn không muốn biểu diễn một hệ thống phân cấp đầy đủ hoặc một phần của các đối tượng.
* **Composite Design Pattern** có thể làm cho thiết kế trở nên quá chung chung. Việc hạn chế các thành phần của composite sẽ khó hơn. Đôi khi bạn muốn một tổng hợp chỉ có một số thành phần nhất định. Với Composite, bạn không thể dựa vào type system để thực thi những ràng buộc đó cho bạn. Thay vào đó, bạn sẽ phải sử dụng kiểm tra thời gian chạy.