**I. Tổng quan**

- Tên: Composite

- Phân loại: Structural

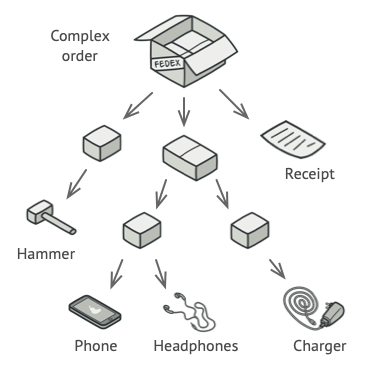
- Mục đích: Tổ chức các đối tượng theo một cấu trúc phân cấp, giúp xử lý các cấu trúc cây một cách linh hoạt và dễ dàng.

**II. Motivation**

**Problem**

Giả sử bạn có hai loại đối tượng: Sản phẩm và Hộp. Một Hộp không chỉ chứa được Sản phẩm mà còn có thể chứa được các Hộp nhỏ. Những Hộp nhỏ này cũng có thể chứa Sản phẩm hoặc thậm chí là các Hộp nhỏ khác.

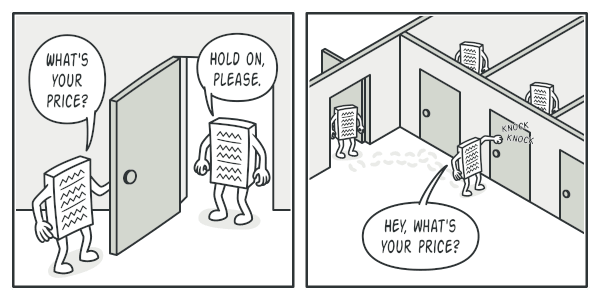
Trường hợp bạn quyết định tạo một hệ thống đặt hàng sử dụng các lớp này. Đơn đặt hàng có thể chỉ chứa một sản phẩm không cần hộp, hoặc có thể là một hộp chứa nhiều sản phẩm và nhiều hộp nhỏ bên trong. Làm sao để xác định giá tiền của một đơn đặt hàng bất kỳ?



**Solution**

Bạn có thể tạo một interface chung cho Sản phẩm và Hộp và khai báo một phương thức để tính giá tiền.

Khi gọi phương thức này trên một Sản phẩm, kết quả trả về sẽ là giá tiền của Sản phẩm đó. Còn nếu gọi trên một Hộp, nó sẽ tính tổng giá tất cả các thành phần chứa trong Hộp đó. Mỗi Hộp còn có thể thêm một vài phụ phí vào giá tiền trong lúc tính toán, chẳng hạn như chi phí đóng gói.



Cách giải quyết này giúp bạn có thể xử lý tất cả các thành phần trong một cây thông qua một interface chung mà không cần biết chính xác class cụ thể của chúng là gì. Khi bạn gọi một phương thức, cây sẽ tự động truyền yêu cầu xuống cho từng phần tử.

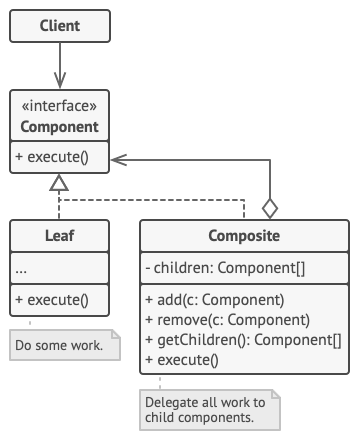
**III. Khả năng ứng dụng**

Khi cần tạo ra các đối tượng trong các cấu trúc cây để biểu diễn hệ thống phân cấp.

Khi muốn client có thể bỏ qua sự khác nhau giữa các thành phần cấu tạo đối tượng và bản thân đối tượng để xử lý chúng dễ dàng hơn.

**IV. Đặc điểm:**

- Cấu trúc mẫu:



- Ý nghĩa của từng thành viên:

1. Component: Interface chứa các phương thức thao tác chung cho các thành phần trong cây.

2. Leaf: Không có con, là thành phần hiện thực các phương thức trong Component

3. Composite: Lưu trữ các thành phần con của chính nó, cài đặt các phương thức được định nghĩa trong bằng cách ủy nhiệm cho các thành phần con xử lý.

**V. Các hệ quả mang lại:**

- Ưu điểm:

1. Làm việc với cấu trúc cây thuận tiện hơn.

2. Open/Closed Principle: có thể thêm kiểu đối tượng mới vào ứng dụng mà không cần phải sửa đổi code hiện có, bằng cách xem nó như một phần tử trong cây.

- Nhược điểm:

1. Đôi khi việc tạo ra một interface chung cho các class khác nhau có thể là một thách thức.

**VI. Cài đặt.**

**VII. Một số so sánh.**

[Skip]

**VIII. Các ví dụ thực tế:**

- Quân đội thường được tổ chức theo hệ thống phân cấp. Một quân đoàn được hợp thành từ các sư đoàn, một sư đoàn bao gồm các lữ đoàn, và lữ đoàn là tập hợp các tiểu đoàn, cứ như vậy ta có thể chia nhỏ xuống cấp trung đội, và tiểu đội. Sau cùng, một tiểu đội là một nhóm những người lính. Mệnh lệnh được đưa ra từ cấp cao nhất và truyền xuống từng cấp cho đến khi mỗi người lính biết được nhiệm vụ của mình.

**IX. Các mẫu liên quan:**

- Builder: Mẫu Builder thường được sử dụng cùng với mẫu Composite để xây dựng cấu trúc cây một cách đệ quy.

- Chain of Responsibility: Chain of Responsibility thường được dùng chung với Composite. Khi có yêu cầu gửi đến một phần tử lá, nó sẽ được ủy nhiệm lên các phần tử cha đến phần tử gốc của cây Composite.

- Iterator: Dùng để duyệt qua các phần tử trong cây Composite.

- Visitor: Dùng để thực thi một công việc trên toàn bộ cây Composite.

- Flyweight: Dùng trong các phần tử lá của cây Composite để tiết kiệm RAM.

- Decorator: Mẫu Decorator cũng liên quan đến việc tổ chức các đối tượng thành cấu trúc cây. Tuy nhiên, trong mẫu Decorator, mỗi đối tượng chỉ có một đối tượng con. Còn mẫu Composite cho phép các đối tượng có thể có nhiều đối tượng con.

- Prototype: Thay vì khởi tạo lại cây Composite từ đầu, có thể sử dụng mẫu Prototype để clone một cây có sẵn và thay đổi các phần tử cần thiết.