**Mẫu Strategy**

1. **Thông tin mẫu:**

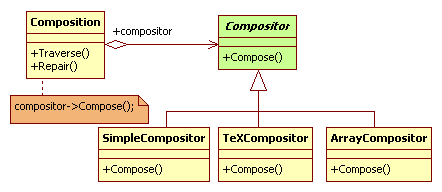
* Tên đầy đủ: Strategy Pattern
* Phân loại: Mẫu Hành vi

1. **Mục đích, ý định**

* Định nghĩa tập hợp các thuật toán khác nhau, đóng gói chúng và làm cho chúng có thể thay thế nhau.
* Mẫu này làm cho các thuật toán thay đổi một cách độc lập và tách biệt khỏi chương trình sử dụng chúng.

1. **Động lực sử dụng**

* Giả sử ta có tình huống sau:
* Rất nhiều các thuật toán khác nhau được sử dụng để đọc dữ liệu từ một luồng dữ liệu từ một file text, sau đó ứng với mỗi loại dữ liệu có được (dạng số, dạng chuỗi,…) , sau đó lưu các thông tin đọc được thành một file text.
* Vấn đề đặt ra là làm thế nào chúng ta có thể cấu hình ứng dụng để tự động lựa chọn thuật toán phù hợp trong quá trình đọc?



1. **Khi nào ứng dụng**

* Nhiều lớp có liên quan với nhau chỉ khác biệt ở hành vi của chúng
  + Mẫu Strategy cung cấp một phương pháp để cấu hình một lớp với một hoặc nhiều hành vi khác nhau.
* Chúng ta cần nhiều thể loại thuật toán khác nhau:
  + Các Strategy có thể được sử dụng khi các biến thể thuật toán này được hiện thực theo kiểu cây kế thừa của các thuật toán.
* Một lớp định nghĩa nhiều hành vi, và xuất hiện dưới dạng các câu lệnh điều kiện trong các operation của nó.
  + Thay vì dùng các câu lệnh điều kiện, chúng ta di chuyển các nhánh điều kiện có liên quan vào trong một lớp Strategy.

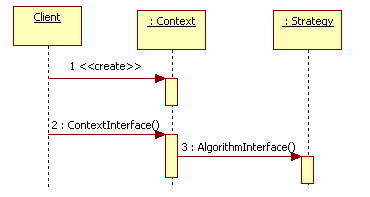
1. **Cấu trúc**



1. **Mối quan hệ - Các thành viên:**

* **Strategy:**
  + Định nghĩa một giao diện chung cho tất cả các thuật toán được hỗ trợ.
  + **Context** sử dụng giao diện này để gọi các thuật toán được định nghĩa trong các **ConcreteStrategy**.
* **ConcreteStrategy:**
  + Định nghĩa các thuật toán, hiện thực từ giao diện **Strategy.**
* **Context:**
  + Được cấu hình với một đối tượng **ConcreteStrategy.**
  + Giữ một tham chiếu đến đối tượng có kiểu **Strategy.**
  + Nên định nghĩa một giao diện cho phép đối tượng **Strategy** đó truy xuất dữ liệu của **Context**.

1. **Sự cộng tác**

****

1. **Các hệ quả:**

* Các cây phân cấp của các thuật toán có liên quan:
  + Cây phân cấp của các lớp Strategy định nghĩa một gia đình của các thuật toán hay các thành viên để Context có thể tái sử dụng.
  + Kế thừa có thể giúp tìm ra các điểm chung trong các thuật toán.
* Một giải pháp thay thế cho việc tạo lớp con:
  + Kế thừa cung cấp một cách khác để hỗ trợ một loạt các thuật toán hoặc hành vi. Chúng ta có thể tạo lớp con từ một lớp Context một cách trực tiếp để cho nó có các hành vi khác nhau. Tuy nhiên cách này sẽ làm các hành vi bị gắn cứng vào lớp Context. Nó pha trộn việc thực hiện thuật toán vào việc hiện thực lớp Context, làm cho Context khó hiểu, khó bảo trì và mở rộng. Mặt khác ta không thể thay đổi thuật toán một cách tự động. Nếu muốn sử dụng một thuật toán khác, bạn phải hiện thực thêm nhiều lớp con khác chỉ để hiện thực các thuật toán khác mà bạn cần.
  + Đóng gói thuật toán vào các lớp riêng biệt giúp cho chúng ta thay đổi các thuật toán một cách độc lập với ngữ cảnh sử dụng chúng, làm cho việc chuyển đổi thuật toán trở nên dễ dàng, dễ hiểu và dễ mở rộng.
* Loại bỏ các câu lệnh điều kiện:
  + Cung cấp một giải pháp thay thế cho các câu lệnh điều kiện trong việc lựa chọn các hành vi mong muốn. Khi các hành vi khác nhau được gộp chung vào trong cùng một lớp, rất khó để tránh việc sử dụng các câu lệnh điều kiện để lựa chọn hành vi đúng. Đóng gói các hành vi vào các lớp riêng biệt, các lớp Strategy giúp loại bỏ các câu điều kiện không cần thiết đó.
* Có thêm nhiều lựa chọn để triển khai:
  + Có thể cung cấp nhiều cách triển khác nhau cho cùng một hành vi. Các client có thể lựa chọn giữa các chiến lược khác nhau.

1. **Mẫu liên quan**

* Flyweight: các đối tượng Strategy thường là các đối tượng flyweight.

1. **Lưu ý cài đặt**
2. **Hệ thống sử dụng**
3. **Demo**