**Mẫu Memento**

1. **Thông tin mẫu:**

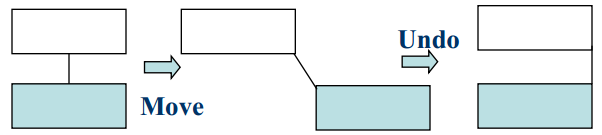
* Tên: Memento Pattern
* Phân loại: Mẫu hành vi

1. **Mục đích, ý định**

* Nắm bắt và biểu lộ trạng thái nội tại của một đối tượng mà không vi phạm tính đóng gói của nó, nhờ vậy ta có thể khôi phục lại trạng thái đó sau này.

1. **Động lực sử dụng**

* Trong quá trình sử dụng đối tượng, trạng thái của nó có thể bị thay đổi liên tục, ta có nhu cầu khôi phục lại trạng thái của nó tại một thời điểm nào đó.
* Ví dụ:
  + Một chương trình đồ họa hiển thị các đối tượng đồ họa trên màn hình, ta thực hiện một thao tác làm thay đổi vị trí của chúng, nếu cảm thấy không phù hợp ta có thể chọn “Undo” để hủy thao tác vừa rồi.



1. **Khi nào ứng dụng**

* Chúng ta cần một snapshot lưu trữ trạng thái của đối tượng để có thể khôi phục lại sau này

1. **Cấu trúc**



1. **Mối quan hệ - Các thành viên:**

* **Originator:** 
  + Tạo một memento lưu trữ một snapshot về trạng thái hiện tại của nó.
  + Sử dụng memento để khôi phục trạng thái.
* **Memento:**
  + Lưu trữ trạng thái nội tại của đối tượng Originator.
  + Ngăn các đối tượng khác với Originator truy cập vào trạng thái này,
* **Caretaker:**
  + Chịu trách nhiệm cho việc đảm bảo tính an toàn của memento.
  + Không bao giờ thực hiện bất cứ hành vi nào hay đọc nội dung của memento.

1. **Sự cộng tác**

* Client yêu cầu một Memento để lưu trạng thái hiện tại của đối tượng Originator.
* Đối tượng Originator khởi tạo một đối tượng Memento với trạng thái hiện tại của nó.
* Client là “Caretaker” của đối tượng Memento, chỉ có đối tượng Originator mới có quyền lưu trữ và lấy thông tin từ đối tượng Memento.
* Nếu Client muốn rollback trạng thái của đối tượng Originator, nó đưa Memento trở lại cho đối tượng Originator để phục hồi.
* Khả năng “undo” và “redo” cho đối tượng Originator có thể được thực hiện bằng một stack các đối tượng Command và một stack các đối tượng Memento.

1. **Các hệ quả**

* Đảm bảo các ranh giới đóng gói được an toàn
  + Các thông tin mà Originator lưu trong Memento được bảo đảm an toàn để các lớp không có liên quan không thể nào biết được, đảm bảo tính đóng gói của đối tượng Originator.
* Làm đơn giản hóa Originator.
  + Client có thể yêu cầu Originator lưu thông tin về các trạng thái của nó. Khi đó Originator sẽ cần phải có các đoạn mã thực hiện các công việc lưu trữ và khôi phục trạng thái của chính mình, điều này làm code của Originator trở nên phức tạp. Sử dụng mẫu Memento giúp giải quyết vấn đề này.
* Sử dụng các Memento có thể rất tốn kém
  + Nếu các thông tin được lưu trữ trong Memento có dung lượng lớn, việc lưu trữ nhiều trạng thái khác nhau của Originator có thể trở thành vấn đề.
* Định nghĩa các giao diện narrow và wide:
  + Tùy thuộc vào ngôn ngữ áp dụng mẫu này, có thể ta không đảm bảo được ràng chỉ có Originator mới được truy cập vào trạng thái của Memento.
* Chi phí ẩn cho việc “chăm sóc” các memento.
  + Caretaker chịu trách nhiệm xóa các memento hết hạn, tuy nhiên caretaker lại không biết có bao nhiêu trạng thái có thể được lưu trong memento. Vì thế một caretake lightweight có thể phải gánh chịu chi phí lưu trữ to lớn khi nó lưu thông tin các memento.

1. **Lưu ý cài đặt**

* Lưu ý vai trò của “caretaker” và “originator”.
* Khi tạo lớp Memento cần khai báo lớp Originator là lớp bạn của nó.
* Originator tạo đối tượng Memento và sao chép trạng thái của nó váo Memento đó.
* Caretaker phải biết khi nào phải lưu trữ một bản sao và khi nào khôi phục lại Originator.
* Originator phải khôi phục trạng thái của nó từ trạng thái lưu trong Memento.

1. **Hệ thống sử dụng**

* Các chương trình cần sử dụng khả năng undo và redo như các chương trình xử lý văn bản, ….

1. **Mẫu liên quan**

* **Command:** sử dụng các Memento để lưu trữ các trạng thái cho các hành vi có thể khôi phục được.
* **Iterator:** Memento có thể được dùng trong quá trình lặp để lưu trữ trạng thái của các phần tử trong đối tượng tập hợp.

1. **Demo**