**Mẫu Visitor**

1. **Thông tin mẫu**

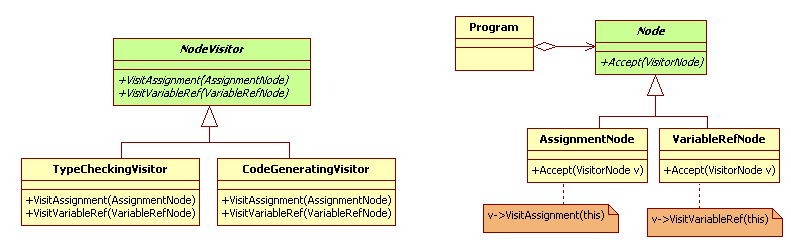
* Tên đầy đủ: Visitor Pattern
* Phân loại: Mẫu hành vi

1. **Mục đích, ý định**

* Biểu diễn một hành động (operation) sẽ được thực hiện trên các phần tử của một cấu trúc đối tượng.
* Cho phép chúng ta định nghĩa một hành động mới mà không làm thay đổi lớp của các phần tử sẽ bị hành động đó tác động.

1. **Động lực sử dụng**

* Giả sử ta có một trình biên dịch sẽ chuyển một chương trình và thể hiện chương trình đã được dịch thành một cấu trúc cây cú pháp trừu tượng (abstract syntax tree – AST). AST có nhiều kiểu node khác nhau, chẳng hạn như các node Assignment, node Variable Reference, và node Arithmetic Expression.
* Các hành động muốn thực hiện trên AST bao gồm:
  + Kiểm tra tất cả các biến đã được định nghĩa chưa.
  + Kiểm tra các biến đã được gán giá trị trước khi được dùng chưa.
  + Kiểm tra type.
  + Sinh code.
  + In/Định dạng code.
* Ta đóng gói một hành động mong muốn vào một đối tượng riêng biêt, được gọi là một visitor.
* Đối tượng Visitor sau đó có thể duyệt qua các phần tử của cây AST.
* Khi một node chấp nhận visitor, nó gọi (invoke) một phương thức của visitor này bao gồm kiểu của của node như là một tham số.
* Visitor sẽ thực thi hành động trên node đó.



1. **Khi nào ứng dụng**

* Một cấu trúc đối tượng chứa nhiều đối tượng với interface khác nhau, và ta muốn thực hiện các hành động trên các đối tượng của nó tùy thuộc vào lớp cụ thể.
* Nhiều hoạt động riêng biệt và không liên quan cần được thực hiện trên các đối tượng của một cấu trúc đối tượng, và ta không muốn làm các lớp của các đối tượng đó trở nên phức tạp hơn với bằn cách định nghĩa các hành động có liên quan bằng trong cùng một lớp.
* Khi đối tượng cấu trúc được chia sẻ trong nhiều ứng dụng, mỗi ứng dụng sẽ có một cách đối xử với đối tượng cấu trúc khác nhau. Sử dụng mẫu Visitor để đưa các hành động vào ứng dụng tương ứng.
* Lớp định nghĩa nên đối tượng cấu trúc ít khi thay đổi, nhưng ta muốn định nghĩa một hành động mới trên cấu trúc đó.
  + Thay đổi các lớp của đối tượng cấu trúc yêu cầu tái định nghĩa lại toàn bộ các interface đối với tất cả các visitor -> tốn chi phí
  + Nếu một đối tượng cấu trúc thường xuyên thay đổi thì tốt hơn hết định nghĩa hành động cần thêm đó vào các lớp.

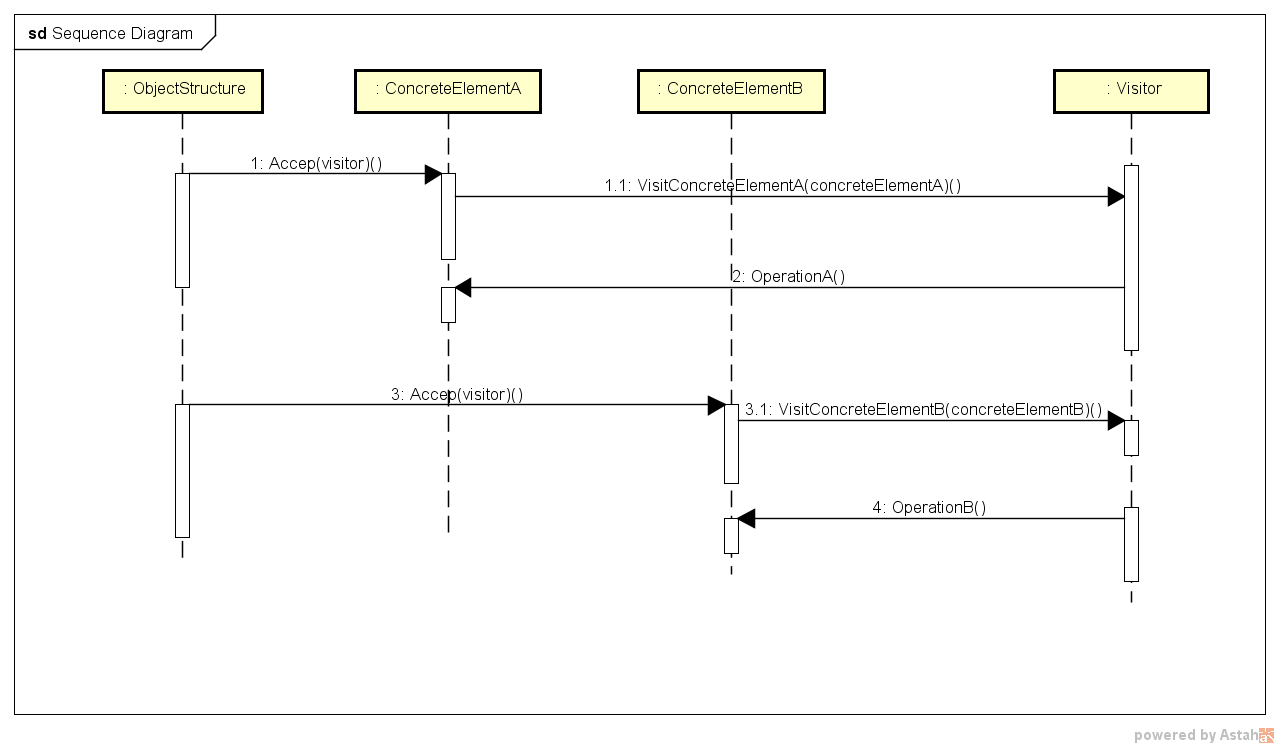
1. **Cấu trúc**



1. **Các thành viên - Mối quan hệ:**

* **Visitor**:
  + Định nghĩa một thao tác Visit cho mỗi lớp ConcreteElement trong cấu trúc đối tượng.
  + Tên và signature của thao tác này sẽ nhận dạng ra lớp
    - Gửi yêu cầu Visit tới visitor,
    - Co phép visitor phát hiện lớp cụ thể của thành phần được visitor ghé thăm.
    - Sau đó visitor có thể truy xuất phần tử trực tiếp thông qua giao diện đặc trưng của nó.
* **ConcreteVisitor**:
  + Thực hiện mỗi thao tác được khai báo bởi Visitor.
  + Mỗi thao tác thực hiện một phần của giải thuật được định nghĩa cho lớp tương ứng của đối tượng trong cấu trúc.
  + Cung cấp ngữ cảnh cho giải thuật và lưu trữ trạng thái cục bộ của nó. Trạng thái này thường dùng để tích lũy các kết quả trong suốt quá trình duyệt qua cấu trúc.
* **Element:**
  + Định nghia một hành động Accep() sẽ nhận một visitor làm tham số.
* **ConcreteElement:**
  + Hiện thực giao diện Element.
* **ObjectStructure:**
  + Có thể liệt kê các phần tử trong cấu trúc.
  + Có thể cung cấp một giao diện cấp cáo cho phép visitor thăm các phần tử của nó.
  + Có thể là một composite hay một collection (list, set,…)

1. **Sự cộng tác**

****

1. **Các hệ quả:**

* Làm việc thêm các operation mới dễ dàng hơn
  + Sử dụng mẫu Visitor giúp chúng ta thêm các hành động mối dựa trên các thành phần của một đối tượng phức tạp dễ dàng hơn. Ta các thể định nghĩa một hành động mới sẽ được dùng trên cấu trúc đối tượng chỉ bằng cách thêm vào một visitor .
  + Tuy nhiên, nếu chúng ta phân tán các chức năng vào quá nhiều lớp khác nhau thì sau đó nếu ta thêm một chức năng mới ta phải thay đổi mỗi lớp.
* Phân nhóm các hoạt động: các hoạt động có liên quan sẽ được tập hợp vào một nhóm và được tách ra khỏi các hoạt động không có liên quan khác.
  + Các hành vi có liên quan sẽ bị phân tán khắp các lớp tham gia xây dựng nên cấu trúc đối tượng. chúng sẽ được tập hợp trong một visitor.
  + Các tập hợp hành vi không có liên quan với nhau sẽ được chứa trong các lớp visitor riêng của chúng.
  + Làm đơn giản hóa các lớp định nghĩa các thành phần (các lớp hiện thực giao diện Element) và các thuật toán được cài trong các Visitor.
  + Các thuật toán đặc trưng cho mỗi cấu trúc có thể được che giấu bên trong Visitor.
* Khó khăn khi cần thêm một lớp ConcreteElement mới:
  + Mỗi lớp ConcreteElement mới được thêm vào thì ta phải thêm một operation ảo trong giao diện Visitor và một thực thi tương ứng trong mỗi lớp con ConcreteVisitor.
  + Đôi khi Visitor sẽ ung cấp một cách hiện thực mặc định, sau đó sẽ được kế thừa lại bởi hầu hết các ConcreteVisitor, nhưng đây lại là một ngoại lệ chứ không phải là một nguyên tắc để tuân theo.
* Tích lũy trạng thái:
  + Khi một Visitor ghé thêm một phần tử của một cấu trúc đối tượng, nó có thể lưu lại trạng thái trong lần ghé thăm đó. Nếu không có visitor, trạng thái này có thể được truyền làm tham số cho các hoạt động thực thi trong quá trình duyệt, hoặc trạng thái này sẽ xuất hiện dưới dạng biến toàn cục.
* Phá vỡ sự đóng gói:
  + Cách tiếp cận của visitor giả định rằng giao diện ConcreteElement đủ mạnh để nó có thể thực hiện được công việc của mình.
  + Do đó, mẫu này có thể yêu cầu chúng ta cung cấp các public operation để truy xuất vào các trạng thái nội tại của phần tử, điều này có thể dẫn đến việc phá vỡ tính đóng gói của phần tử đó.

1. **Lưu ý cài đặt**
2. **Hệ thống sử dụng**
3. **Mẫu liên quan**

* Composite:
  + Visitor có thể được áp dụng trong một hành động duyệt qua cấu trúc đối tượng được định nghĩa bởi mẫu Composite.
* Interpreter:
  + Visitor có thể áp dụng để thực hiện việc thông dịch.

1. **Demo**