1. **Tên và phân loại:**

* Tên: Iterator pattern
* Phân loại: Mẫu hành vi

1. **Mục đích, ý nghĩa:**

* Cung cấp một cách thức truy xuất tuần tự đến các phần tử của một đối tượng tập hợp mà không cần phải phơi bày cấu trúc bên trong của đối tượng này.

1. **Động lực sử dụng:**

- Một đối tượng tập hợp (aggregate object) là một đối tượng thể hiện một nhóm các đối tượng có liên quan với nhau. Ví dụ trong C#, các đối tượng của các lớp Collection như ArrayList, Stack, ... là một đối tượng tập hợp. Một operation - hành vi thường thấy trong các đối tượng như thế này là lặp (iteration) hay xử lý tuần tự (sequential processing) trên mỗi phần tử.

- Một phương pháp cổ điển mà các lớp tập hợp có thể hỗ trợ lặp là định nghĩa các operation như getFirst() và getNext(). Tuy nhiên khi sử dụng các đối tượng từ các lớp này thì client phải hard-code kiểu của lớp đó vào code, gây ra hiện tượng tightly-coupled, mặt khác client code sẽ bị phụ thuộc vào thuật toán lặp của lớp đối tượng, khi thuật toán này thay đổi thì code sử dụng đối tượng đó cũng phải đổi theo.

=> Mẫu Iterator có thể giúp chúng ta xử lý cả hai vấn đề này, áp dụng mẫu này giúp code bên phía client độc lập với lớp tập hợp, đồng thời giúp client sử dụng các đối tượng tập hợp mà không cần quan tâm đến chi tiết cách hiện thực iteration của lớp tập hợp.

1. **Khi nào sử dụng:**

**-** Khi cần truy xuất nội dung của một đối tượng tập hợp mà chúng ta không muốn biết về cấu trúc bên trong của nó.

**-** Để hỗ trợ việc đa dịch chuyển (multiple traversals) trên các đối tượng tập hợp.

- Để cung cấp một giao diện thống nhất cho việc di chuyển trên các đối tượng tập hợp (hay nói cách khác là hỗ trợ tính lặp đa hình - polymorphic iteration).

1. **Cấu trúc**



1. **Các thành viên - Sự cộng tác**

* **Iterator**:
  + Định nghĩa một giao diện cho việc truy xuất và đi qua tập hợp các phần tử.
* **ConcreteIterator**:
  + Hiện thực giao diện **Iterator**, định nghĩa một thuật toán để lặp và di chuyển qua tập hợp.
  + Giữ thông tin về phần tử hiện tại trong quá trình di chuyển qua tập hợp.
* **Aggregate**:
  + Định nghĩa một giao diện cho việc tạo ra đối tượng **Iterator**,
* **ConcreteAggregate**:
  + Các lớp tập hợp hiện thực giao diện Aggregate, khi đó lớp đó sẽ định nghĩa các phương thức cần thiết để trả về một thể hiện của lớp **ConcreteIterator** cho client sử dụng.

1. **Các hệ quả mang lại:**

* Sử dụng mẫu này giúp chúng ta truy xuất tuần tự đến các phần tử của một đối tượng tập hợp mà không quan tâm đến hiện thực của lớp tập hợp tạo ra đối tượng đó, đồng thời độc lập với cách hiện thực thuật toán lặp của nó.
* Client độc lập (loosely-coupling) với kiểu của đối tượng tập hợp và thuật toán lặp sử dụng trong kiểu tập hợp đó. Ta có thể thay đổi mã nguồn của chúng mà không làm ảnh hưởng đến client.
* Hỗ trợ nhiều cách di chuyển trong một tập hợp:
  + Các tập hợp phức tạp có thể được lướt qua bằng nhiều cách, ví dụ như theo thứ tự (inorder) hoặc preorder.
  + Dễ dàng thay đổi thuật toán di chuyển:
    - Chỉ cần thay đổi thể hiện iterator với một thể hiện khác.
    - Hoặc định nghĩa một lớp con của Iterator để hỗ trợ một các di chuyển khác.
* Các Iterator làm đơn giản hóa giao diện Aggregate. Nếu không có các Iterator, nhiều giao diện Aggregate sẽ có phần giao diện cho việc iteration và traversal giống nhau.

1. **Chú ý liên quan đến cài đặt:**

**-** Có hai kiểu iterator: internal iterator và external iterator. Trong C#, ta có thể dùng external iterator một cách dễ dàng.

**-** Các thuật toán di chuyển quả các phần tử trong tập hợp có thể được định nghĩa bên trong lớp tập hợp hoặc bên trong bản thân Iterator, thông thường là trong iterator, khi đó iterator là một lớp inner trong lớp tập hợp.

**-** Robust Iterator:

* Khi sử dụng quá trình lặp bắt đầu,một robust iterator cho phép ta có thể thêm hay xóa các phần tử trong tập hợp mà không tạo ra một một bản copy của tập hợp hiện tại; đồng thời không có phần tử nào bị bỏ qua hay bị lặp lại 2 lần.
* Nếu ta ko cần Robust Iterator thì tức là tập hợp đó không thể chỉnh sửa được và ta cần báo điều này cho client biêt. ta có thể trả về giá trị false hay quăng exception nếu client cố gắng thêm/xóa một phần tử của tập hợp.
* Một giải pháp thay thế là ta định nghĩa thêm các hàm bool remove(); bool insertAfer(); bool insertBefore(); vào bên trong Iterator để ta có thể thêm/xóa các phần tử của tập hợp. Khi đó các hành vi này nên được thêm vào giao diện Iterator.

1. **Hệ thống thực tế:**

* C# hỗ trợ việc lặp và di chuyển qua các phần tử trong một đối tượng tập hợp (điều kiện là lớp của đối tượng đó phải hiện thực giao diện IEnumarable.

1. **Mẫu liên quan**

**-** Composite: Các Iterator thường được áp dụng trong các cấu trúc đệ quy chẳng hạn như Composite.

**-** Factory Method: Các iterator đa hình phụ thuộc vào các factory method để khởi tạo lớp đối tượng Iterator phù hợp.

**-** Memento thường được sử dụng chung với mẫu Iterator. Một iterator có thể sử dụng một memento để chụp (capture) lại trạng thái lặp hiện tại và lưu trữ memento đó một cách cục bộ.