6.23 Mẫu thiết kế ITERATOR

6.23.1 Mở đầu

Các đối tượng hợp lại như danh sách, cần cho bạn cách truy cập đến các phần tử của nó mà không cần mở cấu trúc bên trong. Hơn nữa, bạn có thể muốn duyệt danh sách theo các cách khác nhau, phụ thuộc dựa trên cái gì bạn muốn thực hiện. Nhưng bạn có thể không muốn làm phồng lên giao diện *List* với thao tác cho các bộ duyệt khác nhau, ngay cả nếu bạn có thể thấy trước cái bạn cần. Bạn có thể cũng có nhiều hơn một bộ duyệt chưa quyết định trên cùng một danh sách.

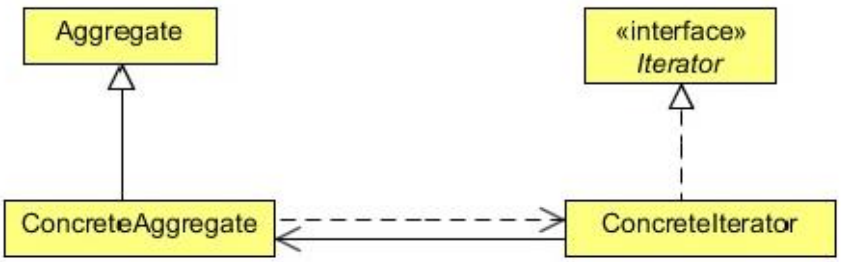
Mẫu Iterator cho phép bạn làm mọi việc đó. Ý tưởng chính trong mẫu này là nhận trách nhiệm để truy cập và duyệt một đối tượng như danh sách *list* và đặt chúng vào trong một đối tượng *Iterator*. Lớp *Iterator* định nghĩa một giao diện để truy cập các phần tử của danh sách. Một đối tượng *Iterator* là có trách nhiệm giữ theo dõi phần tử hiện thời; như vậy, nó biết các phần tử nào đã được viếng thăm.

6.23.2 Mẫu Iterator là gì

Mục đích của mẫu thiết kế Iterator là cung cấp cách truy cập các phần tử của một đối tượng hợp lại một cách tuần tự mà không mở biểu diễn bên trong của nó.

Mẫu *Iterator* cho phép đối tượng client truy cập đến nội dung của vật chứa theo cách tuần tự, mà không biết gì về thể hiện bên trong của nội dung của nó. Thuật ngữ vật chứa *container*, được sử dụng ở trên, có thể đơn giản được định nghĩa như một họ các dữ liệu hoặc các đối tượng. Các đối tượng bên trong vật chứa đến lượt nó có thể là họ các họ.

Mâu *Iterator* cho phép đối tượng client duyệt qua họ này của các đối tượng (hay vật chứa) mà không mở vật chứa xem dữ liệu bên trong được lưu như thế nào. Để làm được việc đó, mẫu Iterator đề xuất đối tượng *Container* cần được thiết kế để cung cấp giao diện *public* ở dạng một đối tượng *Iterator* cho các đối tượng client khác nhau truy cập đến nội dung của nó. Một đối tượng *Iterator* chứa các phương thức *public* cho phép một đối tượng client định hướng qua danh sách các đối tượng bên trong *Container*.



Iterator

- Định nghĩa giao diện cho việc truy vấn thăm viếng các phần tử.

ConcreteIterator

- Cài đặt giao diện *Iterator*.
- Theo dõi vị trị hiện thời trong viếng thăm hợp phần tử

Aggregate

- Định nghĩa giao diện tạo đối tượng *Iterator*

ConcreteAggregate

- Cài đặt giao diện tạo Iterator để trả về khởi tạo của *ConcreteIterator*

6.23.3 Cài đặt mẫu thiết kế Iterator

Chúng ta sẽ cài đặt mẫu thiết kế *Iterator* sử dụng lớp *Shape*. Chúng ta sẽ lưu giữ và duyệt các đối tượng *Shape* sử dụng *Iterator*.

```
package com.javacodegeeks.patterns.iteratorpattern;

public class Shape {

        private int id;
        private String name;

        public Shape(int id, String name){
                this.id = id;
                this.name = name;
        }

        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }

        @Override
        public String toString(){
                return "ID: "+id+" Shape: "+name;
        }

}
```

Lớp *Shape* đơn giản có *Id* và *Name* như các thuộc tính của nó

```
package com.javacodegeeks.patterns.iteratorpattern;

public class ShapeStorage {

        private Shape []shapes = new Shape[5];
```

```
        private int index;

        public void addShape(String name){
                int i = index++;
                shapes[i] = new Shape(i,name);
        }

        public Shape[] getShapes(){
                return shapes;
        }
}
```

Lớp trên được sử dụng để lưu giữ các đối tượng *Shape*. Lớp này chứa mảng kiểu *Shape*, để đơn giản chúng ta khởi tạo mảng gồm 5 phần tử. Phương thức *addShape* được sử dụng để bổ sung đối tượng *Shape* vào mảng và tăng chỉ số lên một. Phương thứ *getShapes* trả về mảng kiểu *Shape*.

```
package com.javacodegeeks.patterns.iteratorpattern;

import java.util.Iterator;

public class ShapeIterator implements Iterator<Shape>{

        private Shape [] shapes;
        int pos;

        public ShapeIterator(Shape []shapes){
                this.shapes = shapes;
        }
        @Override
        public boolean hasNext() {
                if(pos >= shapes.length || shapes[pos] == null)
                        return false;
                return true;
        }

        @Override
        public Shape next() {
                return shapes[pos++];
        }

        @Override
        public void remove() {
                if(pos <=0 )
                        throw new IllegalStateException("Illegal position");
                if(shapes[pos-1] !=null){
                        for (int i= pos-1; i<(shapes.length-1);i++){
                                shapes[i] = shapes[i+1];
                        }
                        shapes[shapes.length-1] = null;
                }
        }
}
```

Lớp trên là *Iterator* cho lớp *Shape*. Lớp này cài đặt giao diện *Iterator* và định nghĩa mọi phương thức của giao diện *Iterator*.

Phương thức *hasNext* trả về giá trị Bool, còn hay không phần tử còn lại. Phương thức next trả về phần tử tiếp theo từ trong họ đó và phương thức *remove* xóa bỏ phần tử hiện thời khỏi họ.

```
package com.javacodegeeks.patterns.iteratorpattern;

public class TestIteratorPattern {

	public static void main(String[] args) {
		ShapeStorage storage = new ShapeStorage();
		storage.addShape("Polygon");
		storage.addShape("Hexagon");
		storage.addShape("Circle");
		storage.addShape("Rectangle");
		storage.addShape("Square");

		ShapeIterator iterator = new ShapeIterator(storage.getShapes());
		while(iterator.hasNext()){
			System.out.println(iterator.next());
		}
		System.out.println("Apply removing while iterating...");
		iterator = new ShapeIterator(storage.getShapes());
		while(iterator.hasNext()){
			System.out.println(iterator.next());
			iterator.remove();
		}
	}

}
```

Code trên sẽ cho kết quả đầu ra như sau:

```
ID: 0 Shape: Polygon
ID: 1 Shape: Hexagon
ID: 2 Shape: Circle
ID: 3 Shape: Rectangle
ID: 4 Shape: Square
Apply removing while iterating...
ID: 0 Shape: Polygon
ID: 2 Shape: Circle
ID: 4 Shape: Square
```

Trong lớp trên, chúng ta đã tạo đối tượng *ShapeStorage* và lưu giữ các đối tượng *Shapes* ở đó. Tiếp theo, chúng ta tạo đối tượng *ShapeIterator* và gán nó cho các *shape*. Chúng ta lặp hai lần, lần đầu không gọi phương thức *remove* và sau đó có gọi *remove*.

Đầu ra chỉ ra cho bạn tác động của *remove*. Tại vòng lặp đầu tiên, *iterator* in ra mọi *shape* nhưng khi phương thức *remove* được gọi , trước hết nó in ra *shape* hiện thời và sau đó *remove shape* tiếp theo. Sau đó, chúng ta gọi phương thức *remove* trên nó mà loại bỏ phần tử hiện thời và tiếp theo *iterato*r trỏ đến đối tượng *shape* tiếp theo đang sẵn sàng.

Đó là tại sao, đầu ra thay đổi thành "*Áp dụng removing trong khi iterating*…" chỉ đưa ra các đối tượng 0, 2 và 4.

6.23.4 Iterator bên trong và bên ngoài

Một *Iterator* có thể được thiết kế là *iterator* bên trong hoặc *iterator* bên ngoài.

1. Iterator bên trong

- Một họ bản thân nó đề xuất các phương thức mà cho phép client viếng thăm các đối tượng khác nhau bên trong họ đó. Chẳng hạn, lớp *Java.util.ResultSet* chứa dữ liệu và cũng đề xuất các phương thức như *next* để điều hướng dọc theo danh sách các phần tử.
- Có thể chỉ có một *iterator* trên một họ tại mỗi thời điểm.
- Một họ cần phải duy trì hoặc lưu trạng thái của *iterator.*

2. Iterator bên ngoài

- Chức năng lặp bên ngoài là tách bạch khỏi họ các phần tử và giữ bên trong một đối tượng khác được tham chiếu là *iterator*. Thông thường, bản thân họ trả về đối tượng *iterator* phù hợp cho client phụ thuộc vào đầu vào của client. Chẳng hạn, lớp *Java.util.Vector* có *iterator* được định nghĩa ở dạng một đối tượng riêng kiểu *Enumeration*. Đối tượng này được trả về cho đối tượng client để phản hồi cho lời gọi của phương thức *element ().*
- Có thể có nhiều bộ duyệt *iterator* trên cùng một họ tại một thời điểm.
- Chi phí cho việc lưu giữ trạng thái của *iterator* là không liên quan đến họ đó. Nó đi cùng đối tượng *Iterator* dành riêng.

6.23.5 Khi nào dùng mẫu thiết kế Iterator

Sử dụng mẫu *Iterator*:

- Để truy cập đến nội dung của đối tượng hợp thành mà không mở biểu diễn bên trong của nó.
- Để hỗ trợ duyệt đa chiều các đối tượng hợp thành.
- Cung cấp giao diện đồng nhất để duyệt các cấu trúc hợp thành khác nhau (như là, hỗ trợ lặp đa hình).