Iterator Pattern là một mẫu thiết kế (design pattern) trong lập trình hướng đối tượng, được sử dụng để cung cấp một cách để truy cập các phần tử của một tập hợp mà không tiết lộ cách thức bên trong của tập hợp đó. Iterator Pattern tách biệt việc lặp lại các phần tử từ cách thức truy cập chúng, cho phép các phần tử có thể được lặp lại mà không ảnh hưởng đến cách thức truy cập của chúng.

Một ưu điểm của Iterator Pattern là nó cung cấp tính linh hoạt trong việc duyệt các phần tử của một tập hợp mà không cần biết cách thức bên trong của tập hợp đó. Nó cũng cho phép các phần tử được lặp lại một cách dễ dàng, bất kể cách thức truy cập của chúng. Tuy nhiên, một nhược điểm của Iterator Pattern là nó có thể làm tăng độ phức tạp của mã, đặc biệt là khi số lượng các phần tử trong tập hợp lớn.

Một ví dụ về việc sử dụng Iterator Pattern trong một mô hình bệnh viện :

Chúng ta có một tập hợp các bệnh nhân trong bệnh viện. Mỗi bệnh nhân được đại diện bởi một đối tượng **Patient** có các thuộc tính như tên, tuổi, mã số bệnh nhân, v.v. Chúng ta muốn duyệt các bệnh nhân trong tập hợp này một cách dễ dàng và không cần biết cách thức bên trong của tập hợp đó.

Đầu tiên, ta tạo một interface **Iterator** định nghĩa các phương thức cần thiết để truy cập các phần tử của tập hợp:

public interface Iterator {  
 boolean hasNext();  
 Object next();  
}

Sau đó, ta tạo một lớp **PatientIterator** để duyệt các phần tử của tập hợp **Patient**:

public class PatientIterator implements Iterator {  
 private Patient[] patients;  
 private int position = 0;  
  
 public PatientIterator(Patient[] patients) {  
 this.patients = patients;  
 }  
  
 public boolean hasNext() {  
 if (position >= patients.length || patients[position] == null) {  
 return false;  
 } else {  
 return true;  
 }  
 }  
  
 public Object next() {  
 Patient patient = patients[position];  
 position++;  
 return patient;  
 }  
}

Lớp **PatientIterator** có một thuộc tính là **patients** để lưu trữ tập hợp các bệnh nhân và một thuộc tính là **position** để lưu trữ vị trí hiện tại của iterator. Trong phương thức **hasNext()**, ta kiểm tra xem vị trí hiện tại có lớn hơn hoặc bằng số lượng các phần tử trong mảng **patients** hay không. Nếu có, hoặc nếu phần tử ở vị trí hiện tại là null, ta trả về false. Ngược lại, ta trả về true. Trong phương thức **next()**, ta trả về phần tử ở vị trí hiện tại của iterator và tăng vị trí lên 1.

Tiếp theo, ta tạo một lớp **PatientCollection** để đại diện cho tập hợp các bệnh nhân và cung cấp phương thức để trả về một iterator cho tập hợp đó:

public class PatientCollection {  
 private Patient[] patients;  
  
 public PatientCollection(Patient[] patients) {  
 this.patients = patients;  
 }  
  
 public Iterator createIterator() {  
 return new PatientIterator(patients);  
 }  
}

Lớp **PatientCollection** có một thuộc tính là **patients** để lưu trữ tập hợp các bệnh nhân và một phương thức **createIterator()** để trả về một đối tượng iterator cho tập hợp đó.

Sử dụng Iterator Pattern, ta có thể duyệt các bệnh nhân trong tập hợp một cách dễ dàng mà không cần biết cách thức bên trong của tập hợp đó:

public class Main {  
 public static void main(String[] args) {  
 Patient[] patients = new Patient[3];  
 patients[0] = new Patient("John", 25, "P001");  
 patients[1] = new Patient("Jane", 30, "P002");  
 patients[2] = new Patient("Bob", 40, "P003");  
  
 PatientCollection collection = new PatientCollection(patients);  
 Iterator iterator = collection.createIterator();  
 while (iterator.hasNext()) {  
 Patient patient = (Patient)iterator.next();  
 System.*out*.println(patient.getName());  
 }  
 }  
}

Kết quả :

