Interpreter Pattern là một trong những Design Pattern thuộc nhóm hành vi (Behavioral Pattern) của Gang of Four (GoF). Nó cung cấp một cách để giải quyết các vấn đề liên quan đến việc diễn giải và đánh giá các biểu thức (expressions) hoặc cấu trúc ngữ pháp (grammar) của một ngôn ngữ.

Interpreter Pattern bao gồm 4 thành phần chính:

* AbstractExpression: Đây là lớp trừu tượng, định nghĩa một phương thức interpret() để diễn giải và đánh giá biểu thức hoặc cấu trúc ngữ pháp.
* TerminalExpression: Đây là lớp kế thừa từ AbstractExpression, biểu diễn một phần tử cuối cùng trong biểu thức hoặc cấu trúc ngữ pháp, không có các phần tử con. Ví dụ: một số, một biến, một toán tử đơn giản.
* NonterminalExpression: Đây là lớp kế thừa từ AbstractExpression, biểu diễn một phần tử trong biểu thức hoặc cấu trúc ngữ pháp, có các phần tử con. Ví dụ: một phép tính, một biểu thức điều kiện.
* Context: Đây là lớp chứa các thông tin cần thiết để diễn giải và đánh giá biểu thức hoặc cấu trúc ngữ pháp.

Ví dụ sử dụng Interpreter Pattern trong mô hình bệnh viện. Giả sử bạn đang thiết kế một ứng dụng quản lý bệnh viện bằng ngôn ngữ Java. Bạn muốn thiết kế một chức năng cho phép tìm kiếm thông tin bệnh nhân theo các tiêu chí như tuổi, bệnh lý, địa chỉ, tình trạng bệnh, v.v. Bạn muốn sử dụng Interpreter Pattern để xử lý các yêu cầu tìm kiếm này.

* AbstractExpression: Trong trường hợp này, lớp AbstractExpression sẽ được đại diện bởi interface Expression, có một phương thức interpret() để diễn giải và đánh giá các biểu thức.

public interface Expression {  
 boolean interpret(Patient patient);  
}

* TerminalExpression: Lớp TerminalExpression được đại diện bởi các lớp tương ứng với các tiêu chí tìm kiếm. Ví dụ: AgeExpression, DiseaseExpression, AddressExpression, StatusExpression.

public class AgeExpression implements Expression {  
 private int minAge;  
  
 public AgeExpression(int minAge) {  
 this.minAge = minAge;  
 }  
  
 @Override  
 public boolean interpret(Patient patient) {  
 return patient.getAge() >= minAge;  
 }  
}

* NonterminalExpression: Lớp NonterminalExpression được đại diện bởi các lớp tương ứng với các mệnh đề logic (AND, OR, NOT) được sử dụng để kết hợp các tiêu chí tìm kiếm. Ví dụ: AndExpression, OrExpression, NotExpression.

public class AndExpression implements Expression {  
 private Expression expr1;  
 private Expression expr2;  
   
 public AndExpression(Expression expr1, Expression expr2) {  
 this.expr1 = expr1;  
 this.expr2 = expr2;  
 }  
  
 @Override  
 public boolean interpret(Patient patient) {  
 return expr1.interpret(patient) && expr2.interpret(patient);  
 }  
}

* Context: Trong trường hợp này, lớp Context được đại diện bởi danh sách bệnh nhân và các tiêu chí tìm kiếm.

public class PatientSearchContext {  
 private List<Patient> patients;  
  
 public PatientSearchContext(List<Patient> patients) {  
 this.patients = patients;  
 }  
  
 public List<Patient> search(Expression expression) {  
 List<Patient> result = new ArrayList<>();  
 for (Patient patient : patients) {  
 if (expression.interpret(patient)) {  
 result.add(patient);  
 }  
 }  
 return result;  
 }  
}

Sau khi đã thiết kế các lớp trên, bạn có thể sử dụng chúng để thực hiện tìm kiếm bệnh nhân theo các tiêu chí khác nhau. Ví dụ:

public class Main {  
 public static void main(String[] args) {  
 List<Patient> patients = new ArrayList<Patient>();  
 PatientSearchContext context = new PatientSearchContext(patients);  
  
 Expression ageExpr = new AgeExpression(30);  
 Expression diseaseExpr = new DiseaseExpression("cancer");  
  
 Expression andExpr = new AndExpression(ageExpr, diseaseExpr);  
 List<Patient> result = context.search(andExpr);  
 }  
}

Ưu điểm:

- Interpreter Pattern giúp giải quyết các vấn đề liên quan đến việc diễn giải và đánh giá biểu thức hoặc cấu trúc ngữ pháp của một ngôn ngữ.

- Nó cho phép thêm các biểu diễn mới một cách đơn giản bằng cách kế thừa các lớp AbstractExpression, TerminalExpression và NonterminalExpression.

- Nó giúp tách biệt quá trình diễn giải và đánh giá khỏi phần còn lại của ứng dụng.

Nhược điểm:

- Interpreter Pattern có thể dẫn đến việc tạo ra quá nhiều đối tượng vì mỗi biểu diễn hoặc phần tử trong cấu trúc ngữ pháp sẽ có một đối tượng tương ứng.

- Nó có thể dẫn đến việc tăng độ phức tạp của mã và giảm hiệu suất của ứng dụng nếu biểu diễn hoặc cấu trúc ngữ pháp rất phức tạp và có nhiều cấp độ.