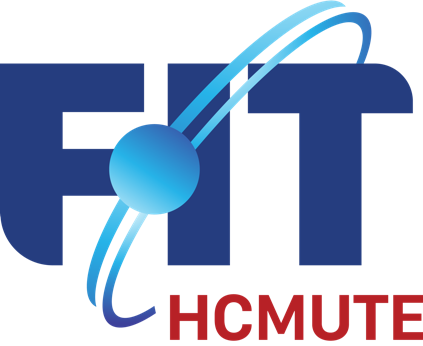
**A white rectangular frame with blue border

Description automatically generatedTRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

*Đề tài:*

**INTERPRETER PATTERN**

**Môn:** Mẫu Thiết Kế Phần Mềm

**Mã lớp:** DEPA330879\_03CLC

**GVHD:** ThS. Nguyễn Minh Đạo

**SVTH:** Nguyễn Hà Quỳnh Giao

**MSSV:** 21110171

*TP. Hồ Chí Minh, tháng 05 năm 2024*

MỤC LỤC

[MỞ ĐẦU 1](#_Toc166110054)

[**1.** **Khái niệm:** 1](#_Toc166110055)

[**2.** **Cấu trúc:** 1](#_Toc166110056)

[**3.** **Trường hợp sử dụng:** 1](#_Toc166110057)

[**4.** **Ưu điểm:** 1](#_Toc166110058)

[**5.** **Nhược điểm:** 2](#_Toc166110059)

[THIẾT KẾ 3](#_Toc166110060)

[**1.** **Ý tưởng thiết kế:** 3](#_Toc166110061)

[**2.** **Giải pháp sử dụng mẫu Interpreter:** 3](#_Toc166110062)

[**3.** **Thiết kế các lớp:** 3](#_Toc166110063)

[**3.1.** **Lớp interface Expression:** 3](#_Toc166110064)

[**3.2.** **Concrete Expression:** 4](#_Toc166110065)

[**3.3.** **Context:** 5](#_Toc166110066)

[**4.** **Sử dụng - Client:** 5](#_Toc166110067)

[**5.** **Phân tích:** 6](#_Toc166110068)

[KẾT LUẬN 7](#_Toc166110069)

[TÀI LIỆU THAM KHẢO 8](#_Toc166110070)

**MỞ ĐẦU**

1. **Khái niệm:**

Mẫu thiết kế Interpreter cung cấp phương tiện để đánh giá cú pháp của ngôn ngữ hoặc biểu thức. Đây là một mẫu thiết kế thuộc loại hành vi (behavioral pattern), được áp dụng để xử lý và diễn giải các biểu thức trong một ngữ cảnh cụ thể. Mẫu này bao gồm triển khai một giao diện biểu thức (expression interface) để cho phép diễn giải một ngữ cảnh cụ thể. Interpreter thường được sử dụng trong việc phân tích SQL, các công cụ xử lý ký hiệu (symbol processing engine), và các ứng dụng tương tự mà yêu cầu đánh giá các biểu thức dựa trên ngôn ngữ.

1. **Cấu trúc:**

Một mẫu Interpreter bao gồm:

* Lớp interface/abstract Expression: chứa phương thức interpret() để diễn giải biểu thức.
* Terminal Expression: triển khai interface/abstract Expression mô tả cụ thể các thành phần cơ bản trong biểu thức.
* Non - Terminal Expression: triển khai interface/abstract Expression, mô tả các biểu thức chứa biểu thức con để diễn giải một hành động phức tạp hơn.
* Context: chứa thông tin diễn giải biểu thức (các biến, giá trị,..)
* Client: sử dụng mẫu diễn giải và thực thi.

1. **Trường hợp sử dụng:**

* Bộ ngữ pháp đơn giản.
* Không quan tâm nhiều về hiệu suất.
* Muốn định nghĩa và diễn giải một ngôn ngữ hoặc ngữ cảnh để thực hiện các hành động hoặc tính toán dựa trên các biểu thức đầu vào.
* Sử dụng trong trình biên dịch (compiler), định nghĩa các bộ ngữ pháp, rule, trình phân tích SQL, XML,

1. **Ưu điểm:**

* Giảm sự phụ thuộc giữa abstraction và implementation: giúp tách biệt giữa phần diễn giải của biểu thức và chi tiết triển khai của từng phần tử trong biểu thức, tạo ra một mức độ kết nối lỏng lẻo giữa các thành phần của hệ thống.
* Giảm số lượng lớp con không cần thiết: tạo ra một cấu trúc phân cấp cho việc diễn giải biểu thức.
* Code gọn gàng và kích thước ứng dụng nhỏ hơn: giảm độ phức tạp của mã nguồn và kích thước của ứng dụng.
* Dễ bảo trì và mở rộng nhờ vào việc cấu trúc rõ ràng và tách biệt giữa các thành phần.
* Cho phép ẩn các chi tiết implement từ client giúp tăng tính trừu tượng và giảm sự phức tạp khi sử dụng.

1. **Nhược điểm:**

* Ngôn ngữ đặc tả phải có cấu trúc ngữ pháp đơn giản.
* Hiệu suất không đảm bảo do yêu cầu thời gian và tài nguyên tính toán.

**THIẾT KẾ**

1. **Ý tưởng thiết kế:**

Trong bài toán này, xây dựng một hệ thống tính toán.

Ngôn ngữ lập trình: Java

1. **Giải pháp sử dụng mẫu Interpreter:**

A diagram of a function

Description automatically generated with medium confidence

* Lớp interface Expression: định nghĩa phương thức interpret() diễn giải biểu thức.
* Concrete Expression: gồm 4 lớp là Number – số nguyên cố định, Variable – đại diện cho một context, Plus và Minus – phép tính.
* Context: chứa các biến và cung cấp phương thức để thiết lập truy xuất giá trị của biến.
* Client: Main – khai báo và sử dụng.

1. **Thiết kế các lớp:**
   1. **Lớp interface Expression:**

Expression.java

package org.example;  
  
public interface Expression {  
 public int interpret(Context context);  
}

* 1. **Concrete Expression:**

Number.java

package org.example;  
  
public class Number implements Expression {  
  
 private int number;  
  
 public Number (int number){  
 this.number = number;  
 }  
  
 @Override  
 public int interpret(Context context) {  
 return number;  
 }  
}

}

Variable.java

package org.example;  
  
public class Variable implements Expression{  
  
 private String name;  
  
 public Variable(String name){  
 this.name = name;  
 }  
  
 @Override  
 public int interpret(Context context) {  
 return context.getValue(name);  
 }  
}

Plus.java

package org.example;  
  
public class Plus implements Expression{  
 private Expression leftOperand;  
 private Expression rightOperand;  
  
 public Plus(Expression leftOperand, Expression rightOperand){  
 this.leftOperand = leftOperand;  
 this.rightOperand = rightOperand;  
 }  
  
 @Override  
 public int interpret(Context context) {  
 return this.leftOperand.interpret(context) + rightOperand.interpret(context);  
 }  
}

Minus.java

package org.example;  
  
public class Minus implements Expression {  
 private Expression leftOperand;  
 private Expression rightOperand;  
  
 public Minus(Expression leftOperand, Expression rightOperand){  
 this.leftOperand = leftOperand;  
 this.rightOperand = rightOperand;  
 }  
  
 @Override  
 public int interpret(Context context) {  
 return this.leftOperand.interpret(context) - rightOperand.interpret(context);  
 }  
}

* 1. **Context:**

Context.java

package org.example;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class Context {  
 private Map<String, Integer> variables;  
  
 public Context(){  
 variables = new HashMap<>();  
 }  
  
 public int getValue(String key){  
 return variables.get(key);  
 }  
  
 public void setValue(String key, int value){  
 variables.put(key, value);  
 }  
}

1. **Sử dụng - Client:**

Client.java

package org.example;  
  
public class Main {  
 public static void main(String[] args) {  
 Context context = new Context();  
 context.setValue("x", 10);  
 context.setValue("y", 5);  
  
 Expression expression1 = new Plus(new Number(20), new Variable("x"));  
 Expression expression2 = new Minus(new Variable("y"), new Number(2));  
  
 int result1 = expression1.interpret(context);  
 int result2 = expression2.interpret(context);  
  
 System.*out*.println("Result of expression1: " + result1);  
 System.*out*.println("Result of expression2: " + result2);  
 }  
}

1. **Phân tích:**

Kết quả sau khi sử dụng mẫu Interpreter là việc xây dựng một hệ thống linh hoạt và dễ mở rộng để diễn giải các biểu thức số học trong một ngữ cảnh cụ thể. Bằng cách ánh xạ mỗi thành phần của biểu thức thành các đối tượng tương ứng, mẫu này cho phép chúng ta diễn giải biểu thức một cách dễ dàng và linh hoạt, đồng thời tách biệt logic diễn giải từ ngữ cảnh. Điều này giúp giảm độ phức tạp của mã nguồn, tăng tính tái sử dụng và dễ bảo trì trong quá trình phát triển và duy trì hệ thống.

.

**KẾT LUẬN**

Mẫu Interpreter là nó cung cấp một cách linh hoạt và mạnh mẽ để diễn giải và xử lý các biểu thức trong một ngữ cảnh cụ thể. Bằng cách tạo ra một cấu trúc phân cấp của các biểu thức và triển khai các phương thức diễn giải tương ứng, mẫu này giúp chúng ta tách biệt logic diễn giải từ ngữ cảnh và tạo ra một hệ thống dễ bảo trì và mở rộng. Việc sử dụng mẫu Interpreter giúp giảm độ phức tạp của mã nguồn, tăng tính tái sử dụng và giúp hệ thống dễ dàng thích ứng với các yêu cầu mới trong quá trình phát triển và duy trì. Tuy nhiên, việc thiết kế và triển khai các biểu thức và ngữ cảnh phù hợp đòi hỏi sự hiểu biết sâu sắc về cấu trúc và logic của hệ thống, và việc này có thể gây ra sự phức tạp trong một số trường hợp.

**TÀI LIỆU THAM KHẢO**

[1] Tutorialspoint. Design Patterns - Interpreter Pattern. Tutorialspoint. https://www.tutorialspoint.com/design\_pattern/interpreter\_pattern.htm (ngày truy cập 09-05-2024)

[2] Hoàng Đinh. (2-1-2022). Interpreter Design Pattern - Trợ thủ đắc lực của Developers. Viblo. <https://viblo.asia/p/interpreter-design-pattern-tro-thu-dac-luc-cua-developers-djeZ1d43KWz> (ngày truy cập 09-05-2024)

[3] Pankaj. (17-6-2023). Interpreter Design Pattern – Building a Language Interpreter. Neatcode. <https://www.neatcode.org/interpreter-pattern> (ngày truy cập 09-05-2024)