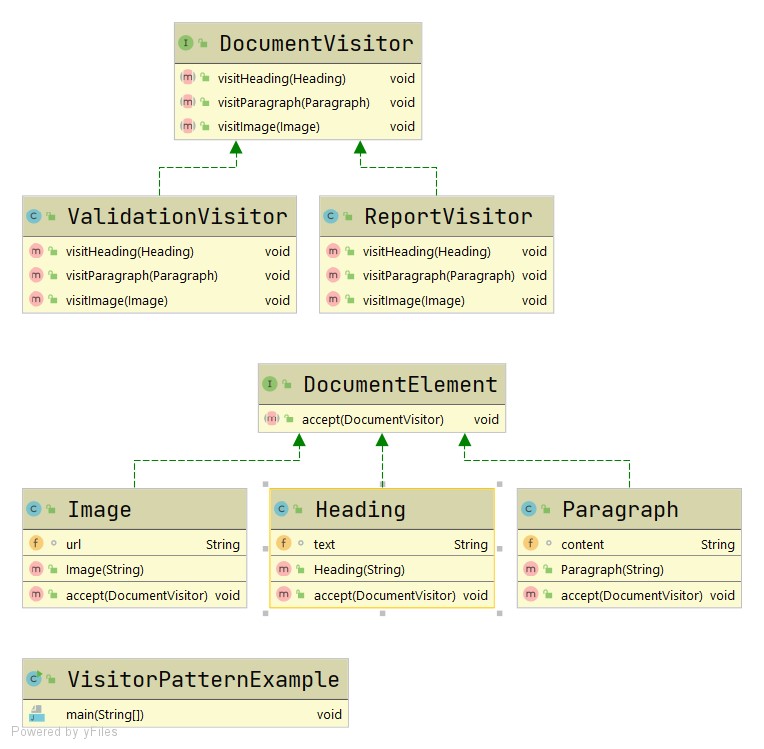
**Mẫu Visitor:**

**Giới thiệu:**

Cho phép thêm hành vi mới vào một cấu trúc đối tượng mà không làm thay đổi các lớp hiện tại. Mẫu này thường được sử dụng để thực hiện các hoạt động trên một tập hợp các đối tượng khác nhau.

**Sau đây là ví dụ về mẫu Visitor:**



Đây là ví dụ về mẫu thiết kế phần mềm Visitor trong bối cảnh phân tích cú pháp biểu thức số học.

**Cấu trúc:**

1. Interface DocumentVisitor:

DocumentVisitor: Giao diện này định nghĩa các chức năng cốt lõi (phương thức) mà các lớp visitor cụ thể phải triển khai. Nó bao gồm ba phương thức:

visitHeading(Heading heading): Phương thức này chịu trách nhiệm thực hiện các thao tác trên đối tượng Heading.

visitParagraph(Paragraph paragraph): Phương thức này chịu trách nhiệm thực hiện các thao tác trên đối tượng Paragraph.

visitImage(Image image): Phương thức này chịu trách nhiệm thực hiện các thao tác trên đối tượng Image.

public interface DocumentVisitor {

void visitHeading(Heading heading);

void visitParagraph(Paragraph paragraph);

void visitImage(Image image);

}

1. Abstract class DocumentElement:

DocumentElement: Lớp trừu tượng này đóng vai trò là lớp cơ sở cho các phần tử tài liệu. Nó áp dụng việc bao gồm một phương thức trừu tượng:

abstract void accept(DocumentVisitor visitor): Phương thức này buộc các lớp con phải triển khai cách chấp nhận một đối tượng DocumentVisitor. Điều này tạo điều kiện cho cơ chế cốt lõi của Mẫu Visitor.

public abstract class DocumentElement {

public abstract void accept(DocumentVisitor visitor);

}

1. Class Heading:

Heading: Lớp này đại diện cho một phần tử tiêu đề trong tài liệu.

public class Heading extends DocumentElement {

private String text;

public Heading(String text) {

this.text = text;

}

*@Override*

public void accept(DocumentVisitor visitor) {

visitor.visitHeading(this);

}

public String getText() {

// **TODO** Auto-generated method stub

return text;

}

}

1. Class Image:

Image: Lớp này đại diện cho một phần tử hình ảnh trong tài liệu.

public class Image extends DocumentElement {

private String url;

public Image(String url) {

this.url = url;

}

*@Override*

public void accept(DocumentVisitor visitor) {

visitor.visitImage(this);

}

public String getUrl() {

// **TODO** Auto-generated method stub

return url;

}

}

1. Class Paragraph:

Paragraph: Lớp này đại diện cho một phần tử đoạn văn trong tài liệu.

public class Paragraph extends DocumentElement {

private String content;

private String text;

public Paragraph(String content, String text) {

this.content = content;

this.text = text;

}

*@Override*

public void accept(DocumentVisitor visitor) {

visitor.visitParagraph(this);

}

public String getContent() {

// **TODO** Auto-generated method stub

return content;

}

}

1. Class ReportVisitor:

ReportVisitor: Lớp này phục vụ như một visitor cụ thể có thể dự định tạo báo cáo dựa trên các phần tử tài liệu.

public class ReportVisitor implements DocumentVisitor {

*@Override*

public void visitHeading(Heading heading) {

// generate report for heading

}

*@Override*

public void visitParagraph(Paragraph paragraph) {

// generate report for paragraph

}

*@Override*

public void visitImage(Image image) {

// generate report for image

}

}

1. Class ValidationVisitor:

ValidationVisitor: Lớp này phục vụ như một visitor cụ thể để xác nhận các phần tử tài liệu.

public class ValidationVisitor implements DocumentVisitor {

*@Override*

public void visitHeading(Heading heading) {

// Kiểm tra tính hợp lệ của tiêu đề

if (heading.getText() == null || heading.getText().isEmpty()) {

System.***out***.println("Error: Invalid heading");

} else {

System.***out***.println("Heading is valid: " + heading.getText());

}

}

*@Override*

public void visitParagraph(Paragraph paragraph) {

// Kiểm tra tính hợp lệ của đoạn văn

if (paragraph.getContent() == null || paragraph.getContent().isEmpty()) {

System.***out***.println("Error: Invalid paragraph");

} else {

System.***out***.println("Paragraph is valid: " + paragraph.getContent());

}

}

*@Override*

public void visitImage(Image image) {

// Kiểm tra tính hợp lệ của hình ảnh

if (image.getUrl() == null || image.getUrl().isEmpty()) {

System.***out***.println("Error: Invalid image URL");

} else {

System.***out***.println("Image is valid: " + image.getUrl());

}

}

}

1. Class VisitorPatternExample:

VisitorPatternExample: Lớp này thể hiện cách sử dụng các lớp DocumentElement và DocumentVisitor để thực hiện các thao tác khác nhau trên tài liệu. Nó tạo các đối tượng Heading, Paragraph, và Image và sau đó sử dụng các đối tượng ValidationVisitor và ReportVisitor để thực hiện các hành động trên chúng.

public class VisitorPatternExample {

public static void main(String[] args) {

Heading heading = new Heading("Proposal");

Paragraph paragraph = new Paragraph("This is the final proposal", null);

Image image = new Image("Information.jpg");

ValidationVisitor validationVisitor = new ValidationVisitor();

ReportVisitor reportVisitor = new ReportVisitor();

heading.accept(validationVisitor);

paragraph.accept(validationVisitor);

image.accept(validationVisitor);

heading.accept(reportVisitor);

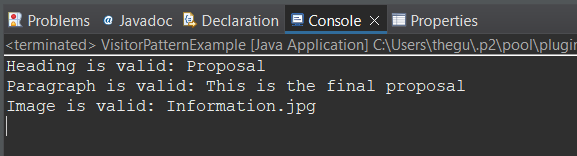
paragraph.accept(reportVisitor);

image.accept(reportVisitor);

}

}

**Kết quả và thử nghiệm:**



Mẫu Visitor cho phép thực hiện nhiều hành vi khác nhau trên các thành phần của một tài liệu mà không thay đổi cấu trúc của các lớp thành phần. Lớp ValidationVisitor kiểm tra tính hợp lệ của các thành phần, trong khi lớp ReportVisitor tạo báo cáo chi tiết về chúng. Mẫu Visitor giúp hệ thống dễ dàng mở rộng bằng cách thêm các Visitor mới cho các hành vi khác nhau.

**Lợi ích:**

Tính linh hoạt: Bạn có thể dễ dàng thêm, xóa hoặc thay đổi các visitor mà không cần sửa đổi mã các phần tử tài liệu.

Khả năng tái sử dụng: Các visitor có thể được sử dụng lại cho nhiều mục đích khác nhau.

Tính tách rời: Mẫu Visitor giúp tách biệt logic truy cập khỏi logic thực thi của các phần tử tài liệu.

Khả năng mở rộng: Bạn có thể dễ dàng mở rộng hệ thống bằng cách thêm các visitor mới.