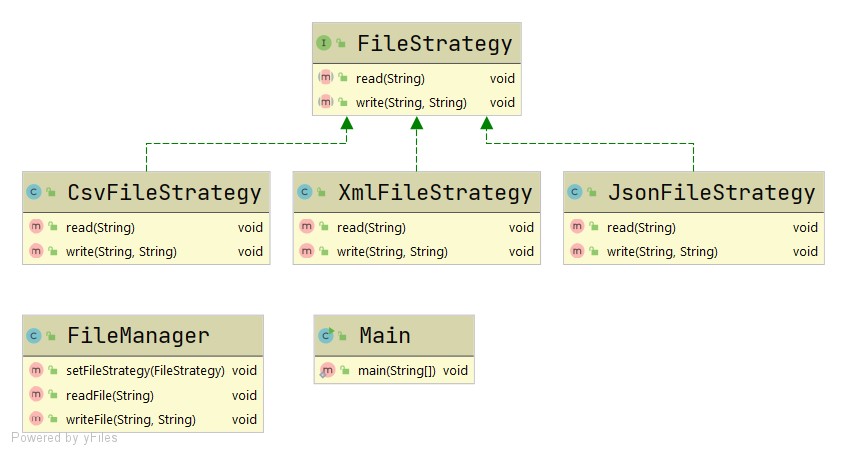
**Mẫu Strategy:**

**Giới thiệu:**

Một tập hợp các thuật toán khác nhau và cho phép lựa chọn thuật toán phù hợp tại thời điểm chạy. Điều này giúp thay đổi hành vi của một đối tượng mà không làm thay đổi cấu trúc của nó.

**Sau đây là ví dụ về mẫu Strategy:**



Trong ví dụ minh họa cho mẫu Strategy trong bối cảnh quản lý tệp tin. Mẫu Strategy cho phép bạn thay đổi hành vi của một thuật toán mà không cần thay đổi cấu trúc của nó.  
**Cấu trúc:**

1. Interface FileStrategy:

FileStrategy là một giao diện định nghĩa các phương thức read() và write().

public interface FileStrategy {

void read(String fileName);

void write(String fileName, String data);

}

1. Class CsvFileStrategy:

Lớp CsvFileStrategy là lớp con của lớp FileStrategy thực hiện các phương thức read() và write() cho các định dạng tệp CSV.

public class CsvFileStrategy implements FileStrategy{

*@Override*

public void read(String fileName) {

// Implement CSV file reading logic here

System.***out***.println("Reading CSV file: " + fileName);

}

*@Override*

public void write(String fileName, String data) {

// Implement CSV file writing logic here

System.***out***.println("Writing CSV data to file: " + fileName);

}

}

1. Class XmlFileStrategy:

Lớp XmlFileStrategy là lớp con của lớp FileStrategy thực hiện các phương thức read() và write() cho các định dạng tệp XML.

public class XmlFileStrategy implements FileStrategy{

*@Override*

public void read(String fileName) {

// Implement XML file reading logic here

System.***out***.println("Reading XML file: " + fileName);

}

*@Override*

public void write(String fileName, String data) {

// Implement XML file writing logic here

System.***out***.println("Writing XML data to file: " + fileName);

}

}

1. Class JsonFileStrategy:

Lớp JsonFileStrategy là lớp con của lớp FileStrategy thực hiện các phương thức read() và write() cho các định dạng tệp JSON.

class JsonFileStrategy implements FileStrategy {

*@Override*

public void read(String fileName) {

// Implement JSON file reading logic here

System.***out***.println("Reading JSON file: " + fileName);

// Parse JSON file content and extract relevant data

}

*@Override*

public void write(String fileName, String data) {

// Implement JSON file writing logic here

System.***out***.println("Writing JSON data to file: " + fileName);

// Convert data into JSON format and write it to the file

}

}

1. Class FileManager:

Lớp FileManager có một thuộc tính fileStrategy là một đối tượng của lớp FileStrategy. Lớp FileManager có các phương thức readFile() và writeFile() sử dụng phương thức read() và write() của đối tượng fileStrategy để đọc và ghi tệp.

public class FileManager {

private FileStrategy fileStrategy;

public FileManager(FileStrategy fileStrategy) {

this.fileStrategy = fileStrategy;

}

public void readFile(String fileName) {

fileStrategy.read(fileName);

}

public void writeFile(String fileName, String data) {

fileStrategy.write(fileName, data);

}

public void setFileStrategy(FileStrategy fileStrategy) {

this.fileStrategy = fileStrategy;

}

}

1. Class Main:

Lớp Main là lớp khởi động chương trình. Lớp Main tạo một đối tượng FileManager và sử dụng các phương thức readFile() và writeFile() của nó để đọc và ghi tệp.

public class Main {

public static void main(String[] args) {

// Create a FileManager with CsvFileStrategy

FileManager fileManager = new FileManager(new CsvFileStrategy());

// Read CSV file

fileManager.readFile("data.csv");

// Write CSV data

fileManager.writeFile("data.csv", "This is some data");

// Change FileStrategy to XmlFileStrategy

fileManager.setFileStrategy(new XmlFileStrategy());

// Read XML file

fileManager.readFile("data.xml");

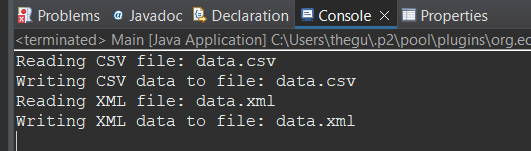
// Write XML data

fileManager.writeFile("data.xml", "<data>This is some data</data>");

}

}

**Kết quả và thử nghiệm:**



Khi tiến hành chạy chương trình, chương trình sẽ đọc và ghi lên các file data.

Trong ví dụ này, FileManager cung cấp phương pháp để đọc và ghi tập tin, nhưng không phụ thuộc vào định dạng tập tin cụ thể nào. Thay vào đó, nó sử dụng một chiến lược định dạng tập tin mà có thể được thay đổi một cách linh hoạt. Điều này cho phép các nhà phát triển dễ dàng mở rộng chương trình bằng cách thêm các định dạng tập tin mới mà không cần sửa đổi mã nguồn của FileManager.

**Lợi ích:**

Mở rộng tính năng dễ dàng: Thêm các chiến lược xử lý tệp tin mới mà không cần thay đổi mã hiện có.

Thay đổi hành vi linh hoạt: Thay đổi chiến lược xử lý tệp tin cho các trường hợp sử dụng khác nhau.

Mã dễ bảo trì: Giữ cho mã tách biệt và dễ hiểu hơn.