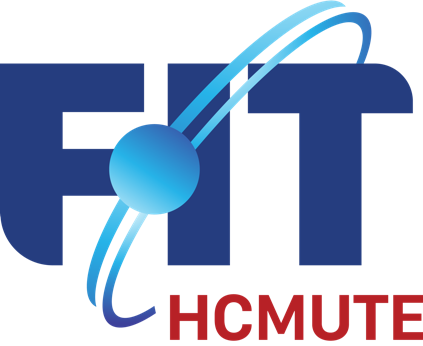
**A white rectangular frame with blue border

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

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

*Đề tài:*

**STRATEGY 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 04 năm 2024*

MỤC LỤC

[**MỞ ĐẦU** 1](#_Toc166107170)

[**1.** **Khái niệm:** 1](#_Toc166107171)

[**2.** **Cấu trúc:** 1](#_Toc166107172)

[**3.** **Trường hợp sử dụng:** 1](#_Toc166107173)

[**4.** **Ưu điểm:** 1](#_Toc166107174)

[**5.** **Nhược điểm:** 1](#_Toc166107175)

[**THIẾT KẾ** 3](#_Toc166107176)

[**1.** **Ý tưởng thiết kế:** 3](#_Toc166107177)

[**2.** **Giải pháp sử dụng Strategy Pattern:** 3](#_Toc166107178)

[**3.** **Thiết kế các lớp:** 4](#_Toc166107179)

[**3.1.** **Lớp giao diện:** 4](#_Toc166107180)

[**3.2.** **Lớp thuật toán cụ thể:** 4](#_Toc166107181)

[**3.3.** **Lớp sử dụng:** 5](#_Toc166107182)

[**4.** **Sử dụng:** 6](#_Toc166107183)

[**5.** **Phân tích:** 7](#_Toc166107184)

[**KẾT LUẬN** 8](#_Toc166107185)

[**TÀI LIỆU THAM KHẢO** 9](#_Toc166107186)

# **MỞ ĐẦU**

1. **Khái niệm:**

Strategy Pattern là một mẫu thiết kế Behavioral, với mục đích xác định một tập hợp các thuật toán, tách biệt việc thực hiện các tuật toán ra khỏi lớp sử dụng chúng. Mẫu này cho phép thay đổi hành vi của một lớp mà không cần làm thay đổi chính lớp đó.

1. **Cấu trúc:**

Một mẫu Strategy Pattern bao gồm:

* Lớp interface (Strategy): định nghĩa chung các thuật toán cụ thể.
* Lớp thuật toán cụ thể (Concrete Strategy): thực thi các thuật toán cụ thể mà lớp interface đã định nghĩa.
* Lớp sử dụng (Context): sử dụng các toán thông qua một lớp interface chung để thực hiện hành vi.

1. **Trường hợp sử dụng:**

* Muốn sử dụng các biến thể khác nhau của một thuật toán trong một đối tượng và có khả năng chuyển đổi từ một thuật toán sang một thuật toán khác trong quá trình chạy.
* Có rất nhiều lớp tương tự chỉ khác nhau trong cách thực hiện một số hành vi.
* Cô lập logic Business của một lớp khỏi các chi tiết cài đặt của các thuật toán có thể không quan trọng trong ngữ cảnh của logic đó.

1. **Ưu điểm:**

* Thay thế các thuật toán một cách linh hoạt.
* Tách biệt phần triển khai và thực thi thuật toán.
* Có thể thay thế việc kế thừa bằng đóng gói thuật toán.
* Tăng tính open-closed: khi thay đổi code không cần thêm mới thuật toán hoặc thay đổi code ở phần context.

1. **Nhược điểm:**

* Người dùng phải biết được sự khác biệt giữa các strategy.
* Số lượng module tăng theo số lượng strategy, nhưng không phải vấn đề lớn.
* Giao tiếp giữa Context và Strategy trở nên phức tạp.

# **THIẾT KẾ**

1. **Ý tưởng thiết kế:**

Trong bài toán này, hệ thống xây dựng cho phép người dùng lựa chọn 1 trong 2 chiến lược để đi đến đích. Người dùng có thể thay đổi con đường lựa chọn nhưng không cần làm thay đổi code của lớp “SelectThePath” hay làm mới người dùng.

Ngôn ngữ lập trình: Java

1. **Giải pháp sử dụng Strategy Pattern:**

A diagram of a computer code

Description automatically generated

* Lớp interface “Path”: thực hiện duy chuyển theo con đường chỉ định.
* Lớp thuật toán:

+ “PathA”: thực hiện duy chuyển theo con đường A được thiết lập.

+ “PathB”: thực hiện duy chuyển theo con đường B được thiết lập.

* Lớp sử dụng “SelectThePath”:

+ Thuộc tính “path”: tham chiếu đến một Path cụ thể.

+ Phương thức “movingToDistance”: thực hiện duy chuyển theo con đường được thiết lập.

1. **Thiết kế các lớp:**
   1. **Lớp giao diện:**

strategies/Path.java:

package org.example.strategies;  
  
public interface Path {  
 void go();  
}

* 1. **Lớp thuật toán cụ thể:**

strategies/impl/PathA.java:

package org.example.strategies.impl;  
  
import org.example.strategies.Path;  
  
public class PathA implements Path {  
 @Override  
 public void go() {  
 System.*out*.print("=============o0o=============\n");  
 System.*out*.print("Chosen Path A: \n");  
 System.*out*.print("Distance: a->b\n");  
 System.*out*.print("Total dístance: 100m\n");  
 System.*out*.print("Distance: b->d\n");  
 System.*out*.print("Total dístance: 500m\n");  
 System.*out*.print("Distance: d->x\n");  
 System.*out*.print("Total dístance: 2000m\n");  
 System.*out*.print("Distance: x->z\n");  
 System.*out*.print("Total dístance: 100m\n");  
 System.*out*.print("=============o0o=============\n");  
 }  
}

strategies/impl/PathB.java:

package org.example.strategies.impl;  
  
import org.example.strategies.Path;  
  
public class PathB implements Path {  
 @Override  
 public void go() {  
 System.out.print("=============o0o=============\n");  
 System.out.print("Chosen Path B: \n");  
 System.out.print("Distance: a->c\n");  
 System.out.print("Total dístance: 350m\n");  
 System.out.print("Distance: c->m\n");  
 System.out.print("Total dístance: 1000m\n");  
 System.out.print("Distance: m->x\n");  
 System.out.print("=============o0o=============\n");  
 }  
}

* 1. **Lớp sử dụng:**

context/SelectThePath.java

package org.example.context;  
  
import org.example.strategies.Path;  
  
public class SelectThePath {  
 private Path path;  
  
 public void setPath(Path path) {  
 this.path = path;  
 }  
  
 public void movingToDistance(){  
 this.path.go();  
 }  
  
}

1. **Sử dụng:**

Main.java

package org.example;  
  
import org.example.context.SelectThePath;  
import org.example.strategies.impl.PathA;  
import org.example.strategies.impl.PathB;  
  
public class Main {  
  
 public static void main(String[] args) {  
 SelectThePath select = new SelectThePath();  
  
 select.setPath(new PathA());  
 select.movingToDistance();  
  
 select.setPath(new PathB());  
 select.movingToDistance();  
 }  
  
}

1. **Phân tích:**

Với ví dụ trên cho thấy được rằng Strategy Pattern đã tách biệt các thuật toán ra khỏi lớp SelectThePath. Nhờ đó mà ta có thể thêm nhiều con đường khác nhau để đi đến nhưng không cần phải thay đổi code trong SelectThePath. Điều này đã làm cho SelectThePath dễ bảo trì trong tương lai mà không phụ thuộc vào chi tiết triển khai của các thuật toán.

# **KẾT LUẬN**

Trong ví dụ về Strategy Pattern, đã cho thấy cách áp dụng nguyên tắc này để tách biệt phần triển khai và thực thi thuật toán đã làm cho hệ thống linh hoạt hơn. Bằng cách sử dụng các strategy khác nhau, chúng ta có thể thay đổi hành vi của đối tượng mà không cần thay đổi mã nguồn hiện tại.

# **TÀI LIỆU THAM KHẢO**

**[1]** Strategy. (không rõ). Refactoring ruru.[**https://refactoring.guru/design-patterns/strategy/java/example**](https://refactoring.guru/design-patterns/strategy/java/example)

**[2]** Hoàng Đình. (21/11/2021). Strategy Design Pattern - Trợ thủ đắc lực của Developers. Vilblo. [**https://viblo.asia/p/strategy-design-pattern-tro-thu-dac-luc-cua-developers-bJzKmdwP59N**](https://viblo.asia/p/strategy-design-pattern-tro-thu-dac-luc-cua-developers-bJzKmdwP59N)