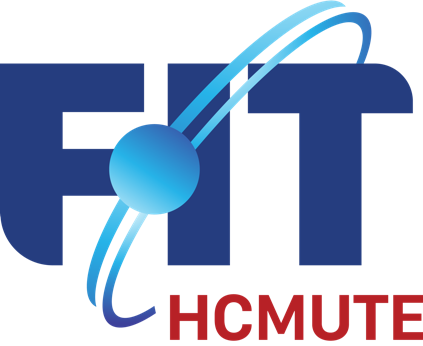
**A white rectangular frame with blue border

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

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

*Đề tài:*

**COMMAND 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](#_Toc166106217)

[**1.** **Khái niệm:** 1](#_Toc166106218)

[**2.** **Cấu trúc:** 1](#_Toc166106219)

[**3.** **Trường hợp sử dụng:** 1](#_Toc166106220)

[**4.** **Ưu điểm:** 1](#_Toc166106221)

[**5.** **Nhược điểm:** 2](#_Toc166106222)

[THIẾT KẾ 3](#_Toc166106223)

[**1.** **Ý tưởng thiết kế:** 3](#_Toc166106224)

[**2.** **Giải pháp sử dụng mẫu Command:** 3](#_Toc166106225)

[**3.** **Thiết kế các lớp:** 4](#_Toc166106226)

[**3.1.** **Lớp interface Command:** 4](#_Toc166106227)

[**3.2.** **Concrete Command:** 4](#_Toc166106228)

[**3.3.** **Receiver:** 5](#_Toc166106229)

[**3.4.** **Invoker:** 6](#_Toc166106230)

[**4.** **Sử dụng - Client:** 6](#_Toc166106231)

[**5.** **Phân tích:** 7](#_Toc166106232)

[KẾT LUẬN 8](#_Toc166106233)

[TÀI LIỆU THAM KHẢO 9](#_Toc166106234)

**MỞ ĐẦU**

1. **Khái niệm:**

Command là một mẫu thiết kế hành vi, chuyển đổi một yêu cầu thành một đối tượng độc lập chứa tất cả thông tin về yêu cầu. Quá trình chuyển đổi này cho phép bạn truyền các yêu cầu như là đối số của một phương thức, trì hoãn hoặc xếp hàng thực thi yêu cầu, và hỗ trợ các hoạt động có thể được hoàn tác.

1. **Cấu trúc:**

Một mẫu Command bao gồm:

* Lớp interface/abstract Command: khai báo giao diện chung cho hành động cụ thể, có chứa phương thức execute() để thực thi hành động đó.
* Concrete Command: lớp cụ thể triển khai lớp interface/abstract Command. Mỗi lớp này triển khai các hàm một các cụ thể cho phương thức execute. Mỗi lệnh liên kết với một hoặc nhiều hành động cụ thể.
* Receiver: thực hiện các hành động liên quan đến yêu cầu.
* Invoker: đối tượng yêu cầu thực thi lệnh, gọi phương thức execute().
* Client: đối tượng tạo và sử dụng command để gán cho Invoker.

1. **Trường hợp sử dụng:**

* Tham số hóa các đối tượng với các hoạt động, biến chúng thành các đối tượng độc lập có thể được truyền, lưu trữ và thực thi linh hoạt.
* Khi bạn muốn xếp thứ tự các hoạt động, lên lịch thực thi chúng hoặc thực thi chúng từ xa, cho phép tuần tự hóa các lệnh, giúp trì hoãn và lập lịch thực thi chúng.
* Thực hiện các hoạt động có thể hoàn tác, bằng cách lưu trữ lịch sử của các hoạt động đã thực hiện, nó cho phép quản lý và thực thi các thao tác ngược lại.

1. **Ưu điểm:**

* Đảm bảo nguyên tắc Single responsibility principle: tách rời các lớp gọi thực thi các hoạt động với các lớp thực hiện những hoạt động đó.
* Đảm bảo nguyên tắc Open/Closed Principle: việc thêm các lệnh mới vào ứng dụng mà không làm hỏng mã nguồn hiện tại của khách hàng.
* Thực hiện chức năng hoàn tác/làm lại (undo/redo).
* Thực hiện việc thực thi trì hoãn của các hoạt động (deferred execution).
* Tổ hợp một tập hợp các lệnh đơn giản thành một lệnh phức tạp.

1. **Nhược điểm:**

* Mã có thể trở nên phức tạp hơn vì bạn đang giới thiệu một tầng hoàn toàn mới giữa người gửi và người nhận.

**THIẾT KẾ**

1. **Ý tưởng thiết kế:**

Trong bài toán này, xây dựng một hệ thống phát nhạc, nơi có thể chơi, tạm dừng.

Ngôn ngữ lập trình: Java

1. **Giải pháp sử dụng mẫu Command:**

A screenshot of a computer program

Description automatically generated

* Lớp interface Command: khai báo giao diện chung, chứa phương thức như execute() và undo() để thực thi và hoàn tác.
* Concrete Commands: gồm 2 lớp là PauseCommand và PlayCommand – điều kiển việc dừng và phát nhạc.
* Receiver: Music Player – chứa các phương thức play(), pause(), stop() trong chức năng dừng và phát nhạc.
* Invoker: RemoteControl – chứa hai phương thức để thực hiện command và hoàn tác command.
* Client: Client – sử dụng RemoteControl để điều khiển trình phát nhạc.

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

Command.java

package org.example;  
  
public interface Command {  
 public void undo();  
 public void execute();  
}

* 1. **Concrete Command:**

PauseCommand.java

package org.example;  
  
public class PauseCommand implements Command{  
  
 private MusicPlayer musicPlayer;  
  
 public PauseCommand(MusicPlayer musicPlayer){  
 this.musicPlayer = musicPlayer;  
 }  
  
 @Override  
 public void undo() {  
 musicPlayer.play();  
 }  
  
 @Override  
 public void execute() {  
 musicPlayer.pause();  
 }  
}

PlayCommand.java

package org.example;  
  
public class PlayCommand implements Command{  
 private MusicPlayer musicPlayer;  
 public PlayCommand(MusicPlayer musicPlayer){  
 this.musicPlayer = musicPlayer;  
 }  
 @Override  
 public void undo() {  
 musicPlayer.stop();  
 }  
 @Override  
 public void execute() {  
 musicPlayer.play();  
 }  
}

* 1. **Receiver:**

MusicPlayer.java

package org.example;  
  
public class MusicPlayer {  
 public MusicPlayer(){}  
  
 public void play(){  
 System.*out*.println("Playing");  
 }  
 public void pause(){  
 System.*out*.println("Pausing");  
 }  
 public void stop(){  
 System.*out*.println("Stopping");  
 }  
}

* 1. **Invoker:**

RemoteControl.java

package org.example;  
  
import java.util.Stack;  
  
public class RemoteControl {  
 private Stack<Command> commands = new Stack<>();  
 public RemoteControl(){}  
 public void executeCommand(Command command){  
 commands.push(command);  
 command.execute();  
 }  
 public void undoCommand(){  
 if(!commands.isEmpty()){  
 commands.pop().undo();  
 } else {  
 System.*out*.println("No command");  
 }  
 }  
}

1. **Sử dụng - Client:**

Client.java

package org.example;  
  
public class Client {  
  
 public static void main(String[] args) {  
 MusicPlayer musicPlayer = new MusicPlayer();  
 RemoteControl remoteControl = new RemoteControl();  
  
 Command play = new PlayCommand(musicPlayer);  
 Command pause = new PauseCommand(musicPlayer);  
  
 remoteControl.executeCommand(play);  
 remoteControl.executeCommand(pause);  
 remoteControl.undoCommand();  
 }  
}

1. **Phân tích:**

Kết quả sau khi sử dụng mẫu Command là khả năng điều khiển các hành động trên đối tượng MusicPlayer một cách linh hoạt và dễ dàng mở rộng. Việc này được thực hiện thông qua việc sử dụng các lớp PlayCommand và PauseCommand, mỗi lớp đại diện cho một hành động cụ thể và triển khai interface Command. Bằng cách này, RemoteControl có thể thực thi các lệnh một cách độc lập và linh hoạt, không phụ thuộc vào cách thức xử lý cụ thể của từng hành động trên MusicPlayer. Điều này giúp tăng tính tái sử dụng của mã nguồn và dễ dàng mở rộng hệ thống trong tương lai.

**KẾT LUẬN**

Mẫu Command là một trong những mẫu thiết kế quan trọng, giúp tách biệt người gửi yêu cầu và người thực hiện hành động, tạo ra sự linh hoạt và tái sử dụng trong mã nguồn. Bằng cách biểu diễn mỗi yêu cầu dưới dạng một đối tượng, mẫu Command cho phép thực thi các hành động, hoàn lại và làm lại chúng một cách dễ dàng, giảm sự phụ thuộc giữa các thành phần và tạo ra cấu trúc linh hoạt. Nó còn cung cấp khả năng thực thi các hành động theo thứ tự và xếp lệnh vào hàng đợi, giúp quản lý và điều khiển hành vi của hệ thống một cách dễ dàng.

**TÀI LIỆU THAM KHẢO**

[1] Refactoring. Command. Refactoring. https://refactoring.guru/design-patterns/command (ngày truy cập 09-05-2024)