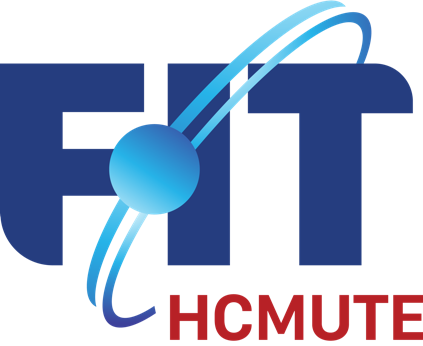
**A white rectangular frame with blue border

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

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

*Đề tài:*

**MEDIATOR 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](#_Toc166106982)

[**1.** **Khái niệm:** 1](#_Toc166106983)

[**2.** **Cấu trúc:** 1](#_Toc166106984)

[**3.** **Trường hợp sử dụng:** 1](#_Toc166106985)

[**4.** **Ưu điểm:** 1](#_Toc166106986)

[**5.** **Nhược điểm:** 2](#_Toc166106987)

[**THIẾT KẾ** 3](#_Toc166106988)

[**1.** **Ý tưởng thiết kế:** 3](#_Toc166106989)

[**2.** **Giải pháp sử dụng Mediator Pattern:** 3](#_Toc166106990)

[**3.** **Thiết kế các lớp:** 4](#_Toc166106991)

[**3.1.** **Lớp giao diện:** 4](#_Toc166106992)

[**3.2.** **Concrete Mediator:** 4](#_Toc166106993)

[**3.3.** **Colleage:** 5](#_Toc166106994)

[**4.** **Sử dụng:** 7](#_Toc166106995)

[**5.** **Phân tích:** 8](#_Toc166106996)

[**KẾT LUẬN** 9](#_Toc166106997)

[**TÀI LIỆU THAM KHẢO** 10](#_Toc166106998)

**MỞ ĐẦU**

1. **Khái niệm:**

Mediator là một mẫu thiết kế hành vi được sử dụng để giảm thiểu sự phụ phức tạp giữa các đối tượng trong một hệ thống phần mềm. Mẫu thiết kế này giúp hạn chế trực tiếp giao tiếp giữa các đối tượng bằng cách buộc chúng phải tương tác thông qua một đối tượng trung gian, làm tăng tính tổ chức và dễ bảo trì của hệ thống.

1. **Cấu trúc:**

Một mẫu Mediator bao gồm:

* Lớp giao diện: Mediator khai báo các phương thức giao tiếp với các thành phần, có thể truyền bất kỳ ngữ cảnh nào như đối số của phương thức này, bao gồm cả các đối tượng của chính mình, nhưng chỉ theo cách sao cho không có sự kết nối xảy ra giữa một thành phần nhận và lớp của người gửi.
* Concrete Mediator: đóng gói các mối quan hệ giữa các thành phần khác nhau, giữ các tham chiếu đến tất cả các thành phần mà chúng quản lý và đôi khi thậm chí quản lý vòng đời của chúng.
* Colleague: Là các đối tượng trong hệ thống mà cần tương tác với nhau thông qua Mediator.

1. **Trường hợp sử dụng:**

* Giảm sự chặt chẽ của mã để hỗ trợ cho việc bảo trì, hỗ trợ trích xuất mọi mối quan hệ giữa các lớp vào một lớp riêng, cô lập bất kỳ thay đổi nào đối với một thành phần cụ thể khỏi phần còn lại của các thành phần.
* Giúp tái sử dụng thành phần của các chương trình trước, các thành phần trở nên không nhận biết về nhau và giao tiếp thông qua một đối tượng trung gian giúp tái sử dụng lớp con đồng thời giảm số lượng lớp con cần tạo ra.

1. **Ưu điểm:**

* Đảm bảo nguyên tắc Single responsibility principle: có thể trích xuất các giao tiếp giữa các thành phần khác nhau vào một nơi duy nhất, làm cho việc hiểu và bảo trì dễ dàng hơn.
* Đảm bảo nguyên tắc Open/Closed Principle: có thể giới thiệu các mediator mới mà không cần thay đổi các thành phần thực tế.
* Có thể giảm sự kết nối giữa các thành phần khác nhau của một chương trình.
* Có thể tái sử dụng các thành phần cá nhân một cách dễ dàng hơn.

1. **Nhược điểm:**

* Theo thời gian, một Mediator có thể phát triển thành một God Object. Điều này xảy ra khi các trách nhiệm và chức năng của Mediator tăng lên quá mức, dẫn đến một thiết kế quá phức tạp và khó hiểu. God Object vi phạm nguyên tắc Single responsibility principle, và nó trở nên khó khăn trong việc bảo trì, hiểu và mở rộng.

**THIẾT KẾ**

1. **Ý tưởng thiết kế:**

Trong bài toán này, hệ thống xây dựng cho phép quản lý các danh sách phát nhạc.

Ngôn ngữ lập trình: Java

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

**A screenshot of a computer program

Description automatically generated**

* Lớp giao diện:PlaylistMediator khai báo các phương thức createPlatList – tạo danh sách phát, addSong – thêm bài hát vào playlist, removeSong – xóa bài hát khỏi playlist.
* Concrete Mediator : MusicPlay triển khai PlaylistMediator, triển khai phương thức createPlatList – tạo danh sách phát, addSong – thêm bài hát vào playlist, removeSong – xóa bài hát khỏi playlist thể hiện sự tương tác giữa các đối tượng tham gia.
* Colleague:

+ Song: chứa thông tin bài hát.

+ Playlist: chứa thông tin danh sách phát.

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

PlaylistMediator.java

package org.example;  
  
public interface PlaylistMediator {  
 public void createPlayList(String playListName);  
 public void addSong(String playlistName, Song song);  
 public void removeSong(String playlistName, Song song);  
}

* 1. **Concrete Mediator:**

MusicPlaylist.java

package org.example;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class MusicPlaylist implements PlaylistMediator{  
 private List<Playlist> playLists;  
  
 public MusicPlaylist(){  
 playLists = new ArrayList<>();  
 }  
  
 @Override  
 public void createPlayList(String playlistName) {  
 if(this.getPlaylist(playlistName) == null){  
 this.playLists.add(new Playlist(playlistName));  
 System.*out*.println("Playlist '" + playlistName + "' created.");  
 } else {  
 System.*out*.println("Playlist '" + playlistName + "' is exist.");  
 }  
 }  
  
 @Override  
 public void addSong(String playlistName, Song song) {  
 Playlist l = getPlaylist(playlistName);  
 if(l == null){  
 l = new Playlist(playlistName);  
 }  
  
 List<Song> songs = l.getSongs();  
  
 songs.add(song);  
 System.*out*.println("Song '" + song.getTitle() + "' added to playlist '" + l.getName() + "'.");  
 l.setSongs(songs);  
  
 }  
  
 @Override  
 public void removeSong(String playlistName, Song song) {  
 Playlist l = getPlaylist(playlistName);  
 if(l == null){  
 System.*out*.println("Playlist '" + l.getName() + "' isn't exist.");  
 return;  
 }  
  
 List<Song> songs = l.getSongs();  
  
 if(songs.remove(song)) {  
 System.*out*.println("Song '" + song.getTitle() + "' removed to playlist '" + l.getName() + "'.");  
 l.setSongs(songs);  
 } else {  
 System.*out*.println("Song '" + song.getTitle() + "' is not in the playlist '" + l.getName() + "'.");  
 }  
 }  
  
 private Playlist getPlaylist(String playListName){  
 for (Playlist l: this.playLists) {  
 if(l.getName().equals(playListName))  
 return l;  
 }  
 return null;  
 }  
}

* 1. **Colleage:**

Song.java

package org.example;  
  
public class Song {  
 private String title;  
 private String singer;  
 private String url;  
  
 public Song(String title, String singer, String url) {  
 this.title = title;  
 this.singer = singer;  
 this.url = url;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public String getSinger() {  
 return singer;  
 }  
  
 public void setSinger(String singer) {  
 this.singer = singer;  
 }  
  
 public String getUrl() {  
 return url;  
 }  
  
 public void setUrl(String url) {  
 this.url = url;  
 }  
}

PlayList.java

package org.example;  
import java.util.ArrayList;  
import java.util.List;  
public class Playlist {  
 private String name;  
 private List<Song> songs;  
  
 public Playlist(String name) {  
 this.name = name;  
 this.songs = new ArrayList<>();  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public List<Song> getSongs() {  
 return songs;  
 }  
  
 public void setSongs(List<Song> songs) {  
 this.songs = songs;  
 }  
}

1. **Sử dụng:**

Giao diện trung gian: MusicApp.java

package org.example;  
  
public class MusicApp {  
 private PlaylistMediator mediator;  
  
 public MusicApp(PlaylistMediator mediator){  
 this.mediator = mediator;  
 }  
  
 public void createPlayList(String playListName){  
 mediator.createPlayList(playListName);  
 }  
 public void addSong(String playlistName, Song song){  
 mediator.addSong(playlistName, song);  
 }  
 public void removeSong(String playlistName, Song song){  
 mediator.removeSong(playlistName, song);  
 }  
}

Main.java

package org.example;  
  
public class Main {  
 public static void main(String[] args) {  
 PlaylistMediator mediator = new MusicPlaylist();  
 MusicApp musicApp = new MusicApp(mediator);  
  
 String playlistName1 = "My Music 1";  
 String playlistName2 = "My Music 2";  
  
 musicApp.createPlayList(playlistName1);  
 musicApp.createPlayList(playlistName2);  
  
 Song song1 = new Song("Mau mat nhung", "Duc Huy", "https://www.youtube.com/watch?v=mtC3ADmaeUU");  
 Song song2 = new Song("Ngay Xa Anh", "My Linh", "https://www.youtube.com/watch?v=ONHK6yic5YA");  
 Song song3 = new Song("Mot Lan Va Mai Mai", "My Tam", "https://www.youtube.com/watch?v=UKSfDqd3t\_o");  
 Song song4 = new Song("Co Doi Khi", "Uyen Linh", "https://www.youtube.com/watch?v=-ghKiWMh89E");  
  
 musicApp.addSong(playlistName1, song1);  
 musicApp.addSong(playlistName1, song2);  
  
 musicApp.addSong(playlistName2, song3);  
 musicApp.addSong(playlistName2, song4);  
  
 musicApp.removeSong(playlistName1, song1);  
 musicApp.removeSong(playlistName2, song2);  
 }  
}

1. **Phân tích:**

Việc tích hợp mẫu thiết kế Mediator vào MusicApp đã mang lại một kiến trúc linh hoạt và dễ bảo trì. Sự phân chia rõ ràng giữa các thành phần giúp giảm thiểu sự phụ thuộc và tạo ra một hệ thống dễ mở rộng. Đồng thời, việc sử dụng mẫu này cũng tăng tính tái sử dụng và giúp tăng cường hiệu suất phát triển trong tương lai. Nhờ vào Mediator, MusicApp có thể duy trì một cấu trúc tổ chức hợp lý và tiện lợi, đồng thời đáp ứng được các yêu cầu ngày càng phức tạp của người dùng một cách hiệu quả.

**KẾT LUẬN**

Trong ví dụ về Mediator Pattern, đã thể hiện rõ sự hiệu quả của việc sử dụng mẫu thiết kế này trong việc quản lý và điều phối tương tác giữa các thành phần của hệ thống. Thay vì cho phép các thành phần tương tác trực tiếp với nhau, Mediator Pattern đã giúp tạo ra một lớp trung gian, giảm sự phụ thuộc và tăng tính linh hoạt của hệ thống. Việc sử dụng Mediator Pattern cũng tăng khả năng tái sử dụng và mở rộng của mã nguồn, làm cho việc thay đổi và mở rộng hệ thống trở nên dễ dàng và ít phức tạp hơn.

**TÀI LIỆU THAM KHẢO**

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