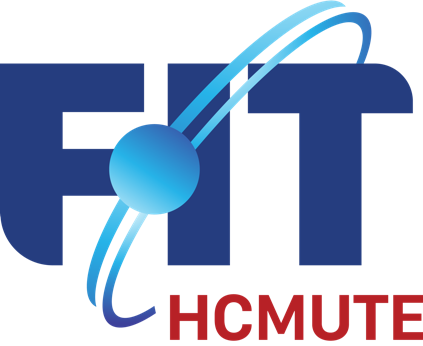
**A white rectangular frame with blue border

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

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

*Đề tài:*

**CHAIN OF REPOSITORY 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](#_Toc166106034)

[**1.** **Khái niệm:** 1](#_Toc166106035)

[**2.** **Cấu trúc:** 1](#_Toc166106036)

[**3.** **Trường hợp sử dụng:** 1](#_Toc166106037)

[**4.** **Ưu điểm:** 1](#_Toc166106038)

[**5.** **Nhược điểm:** 2](#_Toc166106039)

[**THIẾT KẾ** 3](#_Toc166106040)

[**1.** **Ý tưởng thiết kế:** 3](#_Toc166106041)

[**2.** **Giải pháp sử dụng Chain of Repository Pattern:** 3](#_Toc166106042)

[**3.** **Thiết kế các lớp:** 3](#_Toc166106043)

[**3.1.** **Lớp giao diện:** 3](#_Toc166106044)

[**3.2.** **Concrete Handler:** 4](#_Toc166106045)

[**3.3.** **Request Object:** 5](#_Toc166106046)

[**4.** **Sử dụng:** 5](#_Toc166106047)

[**5.** **Phân tích:** 6](#_Toc166106048)

[**KẾT LUẬN** 7](#_Toc166106049)

[**TÀI LIỆU THAM KHẢO** 8](#_Toc166106050)

**MỞ ĐẦU**

1. **Khái niệm:**

Chain of Responsibility là một mẫu thiết kế hành vi cho phép bạn chuyển tiếp các yêu cầu qua một chuỗi các xử lý viên. Khi nhận được một yêu cầu, mỗi xử lý viên quyết định xem liệu có xử lý yêu cầu đó hay chuyển tiếp nó cho xử lý viên kế tiếp trong chuỗi.

1. **Cấu trúc:**

Một mẫu Chain of repository bao gồm:

* Lớp interface/abstract: Handler khai báo giao diện chung cho tất cả các xử lý viên cụ thể. Thông thường, nó chỉ chứa một phương thức duy nhất để xử lý yêu cầu, nhưng đôi khi nó cũng có thể chứa một phương thức khác để thiết lập xử lý viên kế tiếp trong chuỗi.
* Concrete Handlers: lớp cụ thể triển khai interface/abstract class Handler. Mỗi lớp này đảm nhận trách nhiệm xử lý một phần cụ thể của yêu cầu. Nếu không thể xử lý yêu cầu, chúng có thể chuyển tiếp yêu cầu đến handler tiếp theo trong chuỗi.
* Context/Request Object: Đối tượng chứa thông tin cần thiết để xử lý. Trong một số trường hợp, thông tin của yêu cầu có thể được đóng gói vào một đối tượng context hoặc request, và được chuyển qua lại giữa các handler trong chuỗi.

1. **Trường hợp sử dụng:**

* Hệ thống muốn xử lý các loại yêu cầu khác nhau theo các cách khác nhau, nhưng loại cụ thể của yêu cầu và chuỗi của chúng không được biết trước, cho phép liên kết nhiều xử lý viên vào một chuỗi và cho mỗi yêu cầu, hỏi từng xử lý viên xem nó có thể xử lý không, giúp đảm bảo rằng tất cả các xử lý viên có cơ hội xử lý yêu cầu.
* Có yêu cầu thực thi nhiều bộ xử lý theo thứ tự cụ thể.
* Yêu cầu sự linh hoạt của bộ xử lý.

1. **Ưu điểm:**

* Đảm bảo nguyên tắc Single responsibility principle: tách biệt các lớp gọi thực hiện thao tác và các lớp thực hiện thao tác.
* Đảm bảo nguyên tắc Open/Closed Principle: việc thêm các xử lý viên mới vào ứng dụng mà không làm hỏng mã nguồn hiện tại của khách hàng.
* Có thể điều khiển thứ tự xử lý yêu cầu.

1. **Nhược điểm:**

* Một số yêu cầu có thể không được xử lý.

**THIẾT KẾ**

1. **Ý tưởng thiết kế:**

Trong bài toán này, hệ thống xây dựng hệ thống load nhạc tùy thuộc vào trạng thái kết nối mạng.

Ngôn ngữ lập trình: Java

1. **Giải pháp sử dụng Chain of Repository Pattern:**

**A screenshot of a computer

Description automatically generated**

* Lớp interface Handler: khai báo giao diện chung, chứa phương thức handlerFetch để xử lý yêu cầu tải nhạc.
* Concrete Handlers: gồm 2 dạng tải nhạc là FetchingAllSongsHandler và FetchingOfflineSongsHandler - tải toàn bộ nhạc và tải nhạc offline.
* Request Object: Fetching Request – yêu cầu tải nhạc có chứa trạng thái kết nối mạng.

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

Handler.java

package org.example;  
  
public interface Handler {  
 void setNext(Handler next);  
 Handler getNext();  
 public void handlerFetch(FetchingRequest request);  
}

* 1. **Concrete Handler:**

FetchingAllSongsHandler.java

package org.example;  
  
public class FetchingAllSongsHandler implements Handler{  
 private Handler next;  
  
 @Override  
 public void setNext(Handler next) {  
 this.next = next;  
 }  
  
 @Override  
 public Handler getNext() {  
 return this.next;  
 }  
  
 @Override  
 public void handlerFetch(FetchingRequest request) {  
 if (request.getConnectStatus() == FetchingRequest.EConnectStatus.*ONLINE*) {  
 System.*out*.println("Fetching all songs...");  
 } else if (this.next != null) {  
 this.next.handlerFetch(request);  
 } else {  
 System.*out*.println("None of the handlers can fetching song.");  
 }  
 }  
}

FetchingOfflineSongsHandler.java

package org.example;  
  
public class FetchingOfflineSongsHandler implements Handler{  
 private Handler next;  
  
 @Override  
 public void setNext(Handler next) {  
 this.next = next;  
 }  
  
 @Override  
 public Handler getNext() {  
 return this.next;  
 }  
  
 @Override  
 public void handlerFetch(FetchingRequest request) {  
 System.*out*.println("Fetching offline songs...");  
 }  
}

* 1. **Request Object:**

FetchingRequest.java

package org.example;  
  
public class FetchingRequest {  
 private EConnectStatus connectStatus;  
  
 public EConnectStatus getConnectStatus() {  
 return connectStatus;  
 }  
  
 public void setConnectStatus(EConnectStatus connectStatus) {  
 this.connectStatus = connectStatus;  
 }  
  
 public FetchingRequest(EConnectStatus connectStatus) {  
 this.connectStatus = connectStatus;  
 }  
  
 public enum EConnectStatus {  
 *ONLINE*, *OFFLINE* }  
}

}

1. **Sử dụng:**

Main.java

package org.example;  
  
public class Main {  
 public static void main(String[] args) {  
 Handler fetchingAllSongs = new FetchingAllSongsHandler();  
 Handler fetchingOfflineSongs = new FetchingOfflineSongsHandler();  
  
 fetchingAllSongs.setNext(fetchingOfflineSongs);  
  
 FetchingRequest request1 = new FetchingRequest(FetchingRequest.EConnectStatus.*ONLINE*);  
 FetchingRequest request2 = new FetchingRequest(FetchingRequest.EConnectStatus.*OFFLINE*);  
  
 fetchingAllSongs.handlerFetch(request1);  
 fetchingOfflineSongs.handlerFetch(request2);  
 fetchingAllSongs.handlerFetch(request2);  
 fetchingOfflineSongs.handlerFetch(request1);  
 }  
}

1. **Phân tích:**

Kết quả sau khi sử dụng mẫu thiết kế Chain of Responsibility trong ví dụ này là việc tạo ra một hệ thống linh hoạt để xử lý yêu cầu qua một chuỗi các đối tượng xử lý. Interface Handler định nghĩa các phương thức cho việc thiết lập và lấy ra xử lý tiếp theo trong chuỗi, cũng như xử lý yêu cầu. Các lớp cụ thể như FetchingAllSongsHandler và FetchingOfflineSongsHandler cài đặt interface này và xử lý yêu cầu theo cách phù hợp với chúng. Điều này cho phép dễ dàng thêm mới các xử lý và mở rộng chức năng mà không làm thay đổi client gửi yêu cầu.

**KẾT LUẬN**

Trong mẫu thiết kế "Chain of Responsibility", các đối tượng xử lý yêu cầu được tổ chức thành một chuỗi. Mỗi đối tượng trong chuỗi có thể xử lý yêu cầu hoặc chuyển nó cho đối tượng tiếp theo trong chuỗi. Điều này cho phép phân chia trách nhiệm xử lý và mở rộng hệ thống một cách linh hoạt.

**TÀI LIỆU THAM KHẢO**

[1] Refactoring. Chain of Responsibility. Refactoring. https://refactoring.guru/design-patterns/chain-of-responsibility (ngày truy cập 06-05-2024)