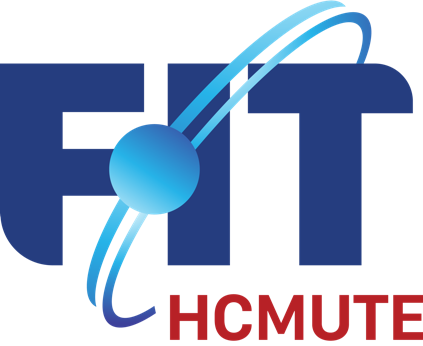
**A white rectangular frame with blue border

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

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

*Đề tài:*

**STATE 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](#_Toc166107010)

[**1.** **Khái niệm:** 1](#_Toc166107011)

[**2.** **Cấu trúc:** 1](#_Toc166107012)

[**3.** **Trường hợp sử dụng:** 1](#_Toc166107013)

[**4.** **Ưu điểm:** 1](#_Toc166107014)

[**5.** **Nhược điểm:** 2](#_Toc166107015)

[**THIẾT KẾ** 3](#_Toc166107016)

[**1.** **Ý tưởng thiết kế:** 3](#_Toc166107017)

[**2.** **Giải pháp sử dụng State Pattern:** 3](#_Toc166107018)

[**3.** **Thiết kế các lớp:** 4](#_Toc166107019)

[**3.1.** **Lớp giao diện:** 4](#_Toc166107020)

[**3.2.** **Concrete States:** 4](#_Toc166107021)

[**3.3.** **Context stores:** 5](#_Toc166107022)

[**4.** **Sử dụng:** 6](#_Toc166107023)

[**5.** **Phân tích:** 6](#_Toc166107024)

[**KẾT LUẬN** 7](#_Toc166107025)

[**TÀI LIỆU THAM KHẢO** 8](#_Toc166107026)

**MỞ ĐẦU**

1. **Khái niệm:**

State là một mẫu thiết kế hành vi cho phép một đối tượng thay đổi hành vi của nó khi trạng thái nội bộ thay đổi. Nó tạo ra ấn tượng như đối tượng đã thay đổi lớp của nó.

1. **Cấu trúc:**

Một mẫu State bao gồm:

* Lớp interface: State khai báo các phương thức cụ thể cho từng trạng thái.
* Concrete States: triển khai lại các phương thức của State triển khai phương thức riêng và cụ thể của từng trạng thái.
* Context stores: tham chiếu đến một trong các đối tượng trạng thái cụ thể và chuyển giao cho nó tất cả công việc cụ thể cho trạng thái đó. Context stores với đối tượng trạng thái thông qua interface status. Context cung cấp một phương thức setter để truyền cho nó một đối tượng trạng thái mới.

1. **Trường hợp sử dụng:**

* Hệ thống tồn tại một đối tượng hoạt động khác nhau tùy thuộc vào trạng thái hiện tại của nó, số lượng trạng thái là rất lớn và mã cụ thể cho từng trạng thái thay đổi thường xuyên.
* Có các điều kiện rẽ nhánh lớn thay đổi cách lớp hoạt động dựa trên giá trị hiện tại của các trường của lớp.
* Nhiều mã trùng lặp trong các trạng thái và chuyển tiếp tương tự của máy trạng thái dựa trên điều kiện.

1. **Ưu điểm:**

* Đảm bảo nguyên tắc Single responsibility principle: tổ chức mã liên quan đến các trạng thái cụ thể vào các lớp riêng biệt. Mỗi lớp trạng thái nên chịu trách nhiệm cho hành vi và hoạt động của nó.
* Đảm bảo nguyên tắc Open/Closed Principle: việc thêm các trạng thái mới mà không làm thay đổi các lớp trạng thái hiện có hoặc Context. Điều này cho phép dễ dàng mở rộng máy trạng thái bằng cách thêm các trạng thái mới như các lớp riêng biệt. Truy cập song song trên cùng một tập hợp vì mỗi đối tượng iterator có chứa trạng thái riêng của nó.
* Để đơn giản hóa mã của đối tượng ngữ cảnh và loại bỏ các điều kiện phức tạp trong máy trạng thái. Thay vì có một khối điều kiện lớn trong đối tượng ngữ cảnh, đối tượng ngữ cảnh chuyển giao công việc cụ thể cho đối tượng trạng thái hiện tại. Điều này giúp tạo ra mã sạch hơn và dễ bảo trì hơn bằng cách đóng gói hành vi cụ thể cho từng trạng thái trong các lớp trạng thái tương ứng.

1. **Nhược điểm:**

* Áp dụng mẫu thiết kế này có thể quá phức tạp nếu máy trạng thái chỉ có một số trạng thái ít hoặc hiếm khi thay đổi.

**THIẾT KẾ**

1. **Ý tưởng thiết kế:**

Trong bài toán này, hệ thống xây dựng hệ thống thể hiện một số trạng thái hoạt động của con người ở mức cơ bản.

Ngôn ngữ lập trình: Java

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

**A screenshot of a computer

Description automatically generated**

* Lớp interface: State khai báo các hành động study – học tập, work – làm việc, entertaiment – giải trí.
* Concrete States: Gồm các trạng thái happy – vui vẻ, normal – bình thường, stress – căng thẳng và triển khai cụ thể cho từng hoạt động tương ứng với trạng thái.
* Context stores: đối tượng lưu trữ trạng thái và tương tác với các hành động.

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

PeopleState.java

package org.example;  
  
public interface PeopleState {  
 public void study(People people);  
 public void entertaiment(People people);  
 public void work(People people);  
}

* 1. **Concrete States:**

HappyState.java

package org.example;  
  
public class HappyState implements PeopleState {  
 @Override  
 public void study(People people) {  
 System.*out*.println("A better study mindset involves quickly and effectively absorbing knowledge.");  
 }  
  
 @Override  
 public void entertaiment(People people) {  
 System.*out*.println("Engaging in recreational activities with enthusiasm and zeal.");  
 }  
  
 @Override  
 public void work(People people) {  
 System.*out*.println("Working more efficiently and productively.");  
 people.setState(new NormalState());  
 }  
}

NormalState.java

package org.example;  
  
public class NormalState implements PeopleState {  
 @Override  
 public void study(People people) {  
 System.*out*.println("Focusing on studying effectively.");  
 }  
  
 @Override  
 public void entertaiment(People people) {  
 System.*out*.println("Participating in recreational activities in moderation.");  
 people.setState(new HappyState());  
 }  
  
 @Override  
 public void work(People people) {  
 System.*out*.println("Completing tasks efficiently.");  
 people.setState(new StressState());  
 }  
}

StressState.java

package org.example;  
  
public class StressState implements PeopleState {  
 @Override  
 public void study(People people) {  
 System.*out*.println("Having difficulty concentrating on studying, experiencing slow and ineffective knowledge absorption.");  
 }  
  
 @Override  
 public void entertaiment(People people) {  
 System.*out*.println("Lacking interest in participating in recreational activities.");  
 people.setState(new NormalState());  
 }  
  
 @Override  
 public void work(People people) {  
 System.*out*.println("Working ineffectively.");  
 }  
}

* 1. **Context stores:**

People.java

package org.example;  
  
public class People {  
 private PeopleState state;  
  
 public PeopleState getState() {  
 return state;  
 }  
  
 public void setState(PeopleState state) {  
 this.state = state;  
 }  
  
 public People(){  
 this.state = new NormalState();  
 }  
  
 public void study(){  
 state.study(this);  
 }  
  
 public void entertaiment(){  
 state.entertaiment(this);  
 }  
  
 public void work(){  
 state.work(this);  
 }  
}

1. **Sử dụng:**

Main.java

package org.example;  
  
public class Main {  
 public static void main(String[] args) {  
 People people = new People();  
 people.work();  
 people.work();  
 people.entertaiment();  
 people.entertaiment();  
 people.study();  
 }  
}

1. **Phân tích:**

Kết quả sau khi sử dụng mẫu thiết kế State trong ví dụ này là việc tạo ra một hệ thống linh hoạt cho quản lý trạng thái của đối tượng People. Bằng cách chuyển đổi giữa các trạng thái như NormalState, HappyState, và StressState, chúng ta có thể điều chỉnh hành vi của người tùy thuộc vào tình huống và hoàn cảnh khác nhau.

Khi sử dụng mẫu thiết kế này, People có khả năng tự động chuyển đổi trạng thái và thích ứng với môi trường xung quanh một cách tự nhiên. Điều này giúp hệ thống linh hoạt và dễ dàng mở rộng, đồng thời giảm thiểu sự phức tạp trong mã nguồn

**KẾT LUẬN**

Trong mẫu thiết kế State Pattern, chúng ta đã thấy sức mạnh của việc tách biệt logic của từng trạng thái vào các lớp riêng biệt, giúp đối tượng có thể thay đổi hành vi của mình một cách linh hoạt và dễ dàng tùy thuộc vào trạng thái hiện tại. Bằng cách sử dụng State Pattern, chúng ta tạo ra các hệ thống linh hoạt, dễ mở rộng và dễ bảo trì, giảm thiểu sự phức tạp trong mã nguồn và tăng tính linh hoạt trong quản lý trạng thái của đối tượng. Điều này giúp cải thiện hiệu suất và sự linh hoạt của ứng dụng, đồng thời tạo ra trải nghiệm người dùng tốt hơn. State Pattern là một công cụ mạnh mẽ trong việc thiết kế và triển khai các hệ thống có trạng thái phức tạp.

**TÀI LIỆU THAM KHẢO**

[1] Refactoring. State. Refactoring. https://refactoring.guru/design-patterns/state (ngày truy cập 06-05-2024)