TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**LÝ TUẤN AN – 52000620  
NGUYỄN THÀNH ĐẠT - 52100781**

**YAME STORE POS**

**BÁO CÁO CUỐI KỲ**

**MẪU THIẾT KẾ**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**LÝ TUẤN AN – 52000620**

**NGUYỄN THÀNH ĐẠT - 52100781**

**YAME STORE POS**

**BÁO CÁO CUỐI KỲ**

**MẪU THIẾT KẾ**

Người hướng dẫn

**TS. Vũ Đình Hồng**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

**LỜI CẢM ƠN**

Đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến TS. Vũ Đình Hồng – Giảng viên khoa Công nghệ thông tin – Trường đại học Tôn Đức Thắng, đã hỗ trợ và giúp đỡ nhiệt tình trong quá trình thực hiện Dự án này.

Chúng em trân trọng cảm ơn Thầy Cô giảng viên Trường đại học Tôn Đức Thắng nói chung cũng như Thầy Cô giảng viên khoa Công nghệ thông tin nói riêng đã giảng dạy và truyền đạt nhiều kinh nghiệm quý trong suốt quá trình học tập tại trường.

Cuối cùng, xin cám ơn gia đình, bạn bè đã luôn động viên và đồng hành trong quá trình học tập cũng như quá trình thực hiện Dự án này.

Mặc dù rất cẩn thận trong quá trình thực hiện đồ án cũng như viết báo cáo nhưng cũng không tránh khỏi những thiếu sót. Chúng em rất mong nhận đựợc sự gópý từ các Thầy/Cô để đồ án được hoàn thiện hơn.

Xin chân thành cám ơn!

*TP. Hồ Chí Minh, ngày 19 tháng 4 năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*A signature of a person

Description automatically generated*

*****Lý Tuấn An*

*Nguyễn Thành Đạt*

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của TS. Vũ Đình Hồng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 19 tháng 4 năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*A signature of a person

Description automatically generated*

*****Lý Tuấn An*

*Nguyễn Thành Đạt*

**MỤC LỤC**

[CHƯƠNG 1. GIỚI THIỆU VỀ DESIGN PATTERN 5](#_Toc165413955)

[1.1 Design Pattern là gì 5](#_Toc165413956)

[1.2 Ý nghĩa của Design Pattern 5](#_Toc165413957)

[1.3 Các loại Design Pattern 5](#_Toc165413958)

[1.4 Tại sao chúng quan trọng trong phát triển phần mềm? 5](#_Toc165413959)

[1.5 Ứng dụng thực tiễn của Design Pattern 5](#_Toc165413960)

[CHƯƠNG 2. CÁC DESIGN PATTERN ĐƯỢC SỬ DỤNG TRONG ỨNG DỤNG 6](#_Toc165413961)

[2.1 Singleton Pattern 6](#_Toc165413962)

[2.1.1 Định nghĩa 6](#_Toc165413963)

[2.1.2 Cấu trúc Singleton Pattern 6](#_Toc165413964)

[2.1.3 Sơ đồ lớp của Singleton Pattern 6](#_Toc165413965)

[2.1.4 Áp dụng Singleton Pattern vào đề tài 6](#_Toc165413966)

[2.2 Factory Pattern 9](#_Toc165413967)

[2.2.1 Định nghĩa 9](#_Toc165413968)

[2.2.2 Các loại Factory Pattern 9](#_Toc165413969)

[2.2.3 Cấu trúc của Factory Pattern 9](#_Toc165413970)

[2.2.4 Áp dụng Factory Pattern vào đề tài 9](#_Toc165413971)

[2.3 Strategy Pattern 13](#_Toc165413972)

[2.3.1 Định nghĩa 13](#_Toc165413973)

[2.3.2 Cấu trúc của Strategy Pattern 13](#_Toc165413974)

[2.3.3 Áp dụng Strategy Pattern và đề tài 13](#_Toc165413975)

[2.4 Template Pattern 17](#_Toc165413976)

[2.4.1 Định nghĩa 17](#_Toc165413977)

[2.4.2 Mục đích và lợi ích của việc sử dụng Template Pattern 17](#_Toc165413978)

[2.4.3 Các thành phần của Template Pattern 18](#_Toc165413979)

[2.4.4 Áp dụng Template Pattern và đề tài 18](#_Toc165413980)

[2.5 Observer Pattern 21](#_Toc165413981)

[2.5.1 Định nghĩa 21](#_Toc165413982)

[2.5.2 Mục đích và lợi ích của việc sử dụng Observer Pattern 21](#_Toc165413983)

[2.5.3 Các thành phần của Observer Pattern 21](#_Toc165413984)

[2.5.4 Áp dụng Observer Pattern và chủ đề 22](#_Toc165413985)

[2.6 Chain of Reponsibility Pattern 25](#_Toc165413986)

[2.6.1 Chain of Responsibility Pattern là gì? 25](#_Toc165413987)

[2.6.2 Các thành phần của Chain of Responsibility Pattern 25](#_Toc165413988)

[2.6.3 Áp dụng Chain of Responsibility và đề tài 25](#_Toc165413989)

[2.7 Decorator Pattern 30](#_Toc165413990)

[2.7.1 Định nghĩa 30](#_Toc165413991)

[2.7.2 Mục đích sử dụng 30](#_Toc165413992)

[2.7.3 Các thành phần chính của Decorator Pattern 30](#_Toc165413993)

[2.7.4 Áp dụng Decorator Pattern vào đề tài 30](#_Toc165413994)

[2.8 Command Pattern 34](#_Toc165413995)

[2.8.1 Command Pattern là gì? 34](#_Toc165413996)

[2.8.2 Các thành phần của Comamnd Pattern 34](#_Toc165413997)

[2.8.3 Áp dụng Command Pattern và đề tài 34](#_Toc165413998)

# GIỚI THIỆU VỀ DESIGN PATTERN

## Design Pattern là gì

Design Pattern là những giải pháp có thể tái sử dụng cho các vấn đề tái diễn trong thiết kế phần mềm. Chúng gói gọn các phương pháp hay nhất và cung cấp mẫu để giải quyết các thách thức thiết kế cụ thể.

## Ý nghĩa của Design Pattern

Design Pattern không chỉ là một mẫu thiết kế cụ thể, mà còn là một cách tiếp cận chung trong việc giải quyết vấn đề trong phát triển phần mềm. Bằng cách cung cấp các mô hình kiến trúc được kiểm chứng, Design Pattern giúp tăng tính tái sử dụng, tính linh hoạt và dễ bảo trì của các hệ thống phần mềm.

## Các loại Design Pattern

Design Pattern được chia thành ba loại chính: Creational, Structural, và Behavioral. Mỗi loại có các mẫu thiết kế đặc trưng, được sử dụng để giải quyết các vấn đề cụ thể trong quá trình phát triển phần mềm.

## Tại sao chúng quan trọng trong phát triển phần mềm?

Design Pattern thúc đẩy khả năng sử dụng lại, khả năng bảo trì và khả năng mở rộng của mã. Chúng cho phép các nhà phát triển giải quyết hiệu quả các vấn đề thiết kế phổ biến và nâng cao chất lượng tổng thể của hệ thống phần mềm.

## Ứng dụng thực tiễn của Design Pattern

Trong phần này, chúng ta sẽ xem xét một số ví dụ về cách sử dụng các Design Patterns. Những ví dụ này sẽ giúp chúng ta hiểu rõ hơn về cách áp dụng các mẫu thiết kế vào ứng dụng thực tế và lợi ích mà chúng mang lại.

CÁC DESIGN PATTERN ĐƯỢC SỬ DỤNG TRONG ỨNG DỤNG

## Singleton Pattern

### Định nghĩa

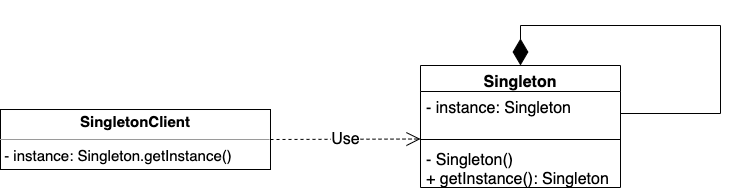
Singleton Pattern là creational design pattern nhằm hạn chế việc khởi tạo một lớp cho một đối tượng duy nhất. Mục tiêu chính của nó là kiểm soát việc tạo đối tượng bằng cách đảm bảo rằng chỉ có một phiên bản của lớp tồn tại trong suốt vòng đời của ứng dụng.

### Cấu trúc Singleton Pattern

Cấu trúc của Singleton Pattern thường bao gồm ba thành phần chính:

* Singleton Class: Đây là lớp chỉ được phép tồn tại một phiên bản. Nó thường có một hàm tạo riêng để ngăn chặn việc khởi tạo trực tiếp từ bên ngoài lớp.
* Static Instance Variable: Singleton Class chứa một biến tĩnh chứa tham chiếu đến một thể hiện duy nhất của lớp. Biến này thường được đặt tên là "instance" và được khởi tạo thành null.
* Static Access Method: Để truy cập phiên bản singleton, một phương thức tĩnh được cung cấp trong lớp Singleton. Phương thức này kiểm tra xem một thể hiện của lớp đã tồn tại chưa. Nếu không, nó sẽ tạo một thể hiện mới và trả về nó. Nếu một phiên bản đã tồn tại, nó chỉ trả về phiên bản hiện có.

### Sơ đồ lớp của Singleton Pattern



*Hình 3.1 Sơ đồ lớp của Singleton Pattern.*

Giải thích từng thành phần trong lớp:

* Singleton Class: Đây là lớp triển khai Singleton Pattern. Nó thường có một hàm tạo riêng để ngăn chặn việc khởi tạo bên ngoài và một phương thức tĩnh để cung cấp quyền truy cập vào một phiên bản duy nhất.
* Static Instance Variable: Đây là một biến tĩnh được khai báo trong lớp Singleton để giữ tham chiếu đến một thể hiện duy nhất của lớp. Nó thường được khai báo là riêng tư để ngăn chặn truy cập trực tiếp từ bên ngoài lớp.
* Static Access Method: Phương thức truy cập tĩnh, thường được đặt tên là getInstance(), được sử dụng để truy xuất cá thể singleton. Nó kiểm tra xem một thể hiện đã tồn tại chưa. Nếu có, nó sẽ trả về phiên bản hiện có. Nếu không, nó sẽ tạo một thể hiện mới và trả về nó. Phương thức này đảm bảo rằng chỉ có một phiên bản của lớp được tạo và cung cấp một điểm truy cập toàn cục vào nó.

### Áp dụng Singleton Pattern vào đề tài

Sơ đồ lớp:



Lý do áp dụng:

Đảm bảo duy nhất: Singleton Pattern đảm bảo rằng chỉ có một thể hiện của lớp được tạo ra trong suốt vòng đời của ứng dụng. Trong trường hợp của UserSession, chỉ có một phiên đăng nhập duy nhất được duy trì trong ứng dụng.

Dễ dàng truy cập: Bằng cách sử dụng Singleton, các phần khác của ứng dụng có thể dễ dàng truy cập vào phiên đăng nhập hiện tại thông qua thuộc tính Instance mà không cần tạo một thể hiện mới. Như lưu lại thông tin của người thanh toán vào hóa đơn,...

Kiểm soát truy cập: Singleton Pattern cho phép kiểm soát truy cập vào đối tượng duy nhất từ mọi nơi trong ứng dụng. Điều này giúp đảm bảo tính nhất quán và an toàn của trạng thái và hành vi của UserSession.

Ngoài ra nhóm em còn áp dụng singleton kết hợp với abstract factory để quản lý sử dụng cơ sở dữ liệu. Phần trình bày sẽ ở phần tiếp theo

Code:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

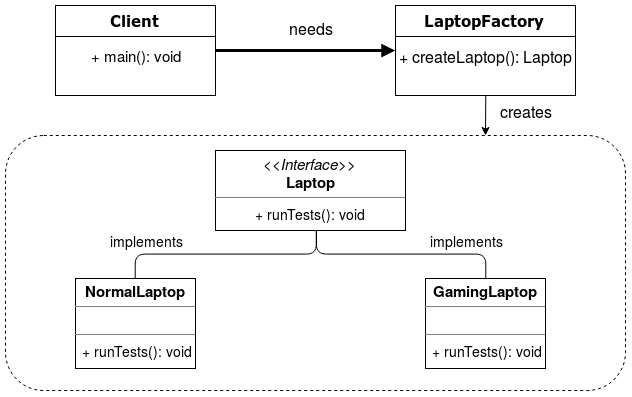
## Factory Pattern

### Định nghĩa

Factory Pattern là một creational design pattern cung cấp giao diện để tạo các đối tượng trong superclass nhưng cho phép các lớp con thay đổi loại đối tượng sẽ được tạo. Nói một cách đơn giản hơn, nó đóng gói logic tạo đối tượng và ủy quyền cho các lớp con, do đó thúc đẩy sự ghép nối lỏng lẻo và tăng cường tính linh hoạt trong mã.

### Các loại Factory Pattern

* Simple Factory Pattern: Bao gồm single factory class chịu trách nhiệm tạo các đối tượng dựa trên đầu vào được cung cấp. Nó gói gọn quá trình tạo đối tượng và bảo vệ client khỏi sự phức tạp của việc khởi tạo đối tượng. Tuy nhiên, nó thiếu tính linh hoạt được cung cấp bởi các mô hình nhà máy khác



*Hình 3.5 Simple Factory Pattern*

* Factory Method Pattern: Định nghĩa một interface để tạo các đối tượng nhưng cho phép các lớp con thay đổi quá trình khởi tạo. Mỗi lớp con có thể cung cấp cách triển khai phương thức xuất xưởng, do đó cho phép tạo ra các loại đối tượng khác nhau mà không cần sửa đổi mã máy khách.
* Abstract Factory Pattern: tiến thêm một bước nữa bằng cách cung cấp interface để tạo nhóm các đối tượng liên quan hoặc phụ thuộc. Nó gói gọn một nhóm các phương thức xuất xưởng, mỗi phương thức chịu trách nhiệm tạo ra một loại đối tượng cụ thể. Mẫu này thúc đẩy việc tạo ra các họ đối tượng gắn kết và đảm bảo tính tương thích của chúng.

### Cấu trúc của Factory Pattern

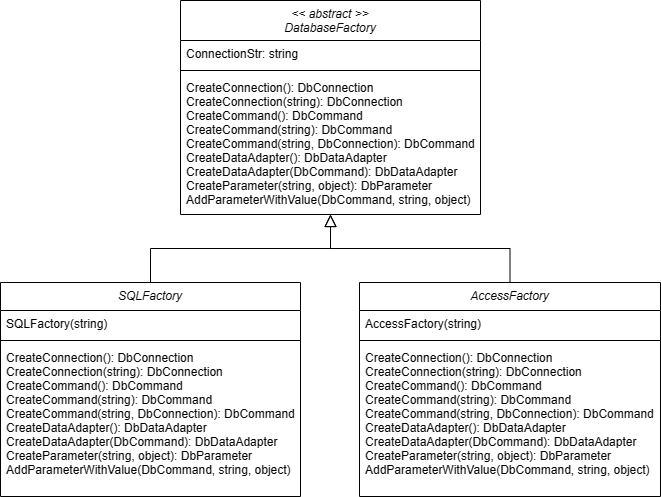
* Product: Đây là interface hay abstract class định nghĩa phương thức và thuộc tính chung cho một nhóm class con cụ thể. Mọi class con (Concrete Product) phải implement hoặc kế thừa từ interface hoặc abstract class này.
* Concrete Product: Đây là class cụ thể kế thừa hoặc implement từ Product. Concrete Product định nghĩa đối tượng cụ thể sẽ được tạo ra. Trong Factory pattern, thường có nhiều class con của Product, và mỗi một Concrete Product tương ứng với một biến thể của đối tượng cần tạo.
* Creator: Là class có chứa phương thức để tạo đối tượng (Factory Method). Phương thức này thường là abstract hoặc có một implementation mặc định để trả về một instance của một Concrete Product. Tuy nhiên, nó có thể được ghi đè (overridden) để trả về một instance của một Concrete Product khác.
* Concrete Creator: Đây là class cụ thể kế thừa từ Creator. Nó ghi đè phương thức tạo đối tượng để trả về đối tượng của loại Concrete Product phù hợp.



*Hình 3.6 Factory Pattern*

### Áp dụng Factory Pattern vào đề tài

Sơ đồ lớp:



Lý do áp dụng:

Tính linh hoạt và mở rộng: Abstract Factory cho phép tạo ra các nhóm đối tượng liên quan mà không cần chỉ định lớp cụ thể. Trong trường hợp này, ta có thể tạo ra các nhóm đối tượng cho các loại cơ sở dữ liệu khác nhau mà không cần thay đổi mã của các lớp sử dụng chúng. Điều này làm cho việc mở rộng hệ thống với các loại cơ sở dữ liệu mới trở nên dễ dàng hơn.

Tách biệt giao diện và triển khai: Abstract Factory Pattern tách biệt giao diện (interface) của một nhóm đối tượng từ việc triển khai cụ thể của chúng. Điều này cho phép tạo ra các nhóm đối tượng mà không cần biết chi tiết về cách triển khai cụ thể của chúng. Trong trường hợp này, giao diện DatabaseFactory định nghĩa phương thức tạo các đối tượng cơ sở dữ liệu, trong khi các lớp cụ thể như SQLFactory và AccessFactory cung cấp triển khai cụ thể cho các đối tượng này.

Tính chuyển đổi cao: Abstract Factory Pattern cho phép thay đổi cụ thể nhóm đối tượng được tạo mà không ảnh hưởng đến các lớp sử dụng chúng. Ví dụ, nếu khi cửa hàng muốn chuyển từ sử dụng SQL Server sang sử dụng Acess, ta chỉ cần thay đổi triển khai của DatabaseFactory từ SQLFactory sang AcessFactory, trong khi các lớp sử dụng DatabaseFactory vẫn giữ nguyên.

Tính di động cao: Abstract Factory Pattern làm cho việc thay đổi loại cơ sở dữ liệu trở nên dễ dàng hơn mà không cần thay đổi nhiều mã. Điều này giúp giảm thiểu sự phụ thuộc giữa các lớp và tăng tính di động của hệ thống.

Code:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Kết hợp singleton để quản lý

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

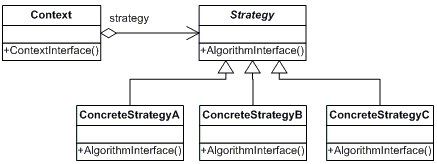
## Strategy Pattern

### Định nghĩa

Về cốt lõi, Strategy Pattern là một mẫu thiết kế hành vi (behavioral design pattern) cho phép định nghĩa một nhóm thuật toán, đóng gói từng thuật toán và làm cho chúng có thể hoán đổi cho nhau. Điều này cho phép khách hàng thay đổi các thuật toán một cách độc lập với bối cảnh sử dụng chúng.

### Cấu trúc của Strategy Pattern

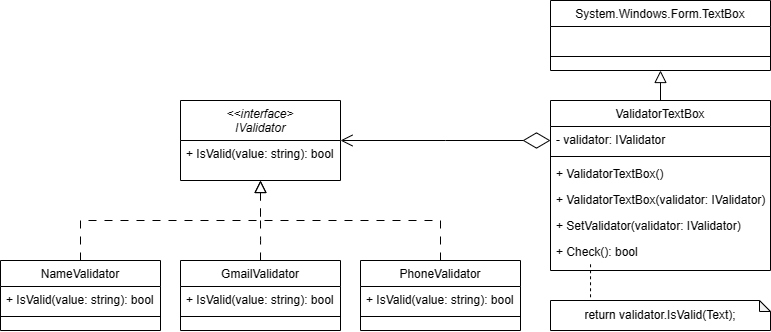
* Context: Là lớp chứa đối tượng chiến lược và sử dụng chúng để thực hiện một nhiệm vụ cụ thể. Bối cảnh có thể nhận và chuyển đổi các chiến lược khác nhau tùy thuộc vào yêu cầu cụ thể.
* Strategy: Đây là giao diện hoặc lớp trừu tượng mô tả một tập hợp các thuật toán có thể thay đổi. Mỗi chiến lược đại diện cho một phương pháp cụ thể để thực hiện một tác vụ.
* Concrete Strategies: Các lớp cụ thể triển khai các thuật toán cụ thể được định nghĩa trong giao diện chiến lược. Mỗi chiến lược cụ thể cung cấp một triển khai cụ thể cho một thuật toán.



*Hình 3.7 Strategy Pattern*

### Áp dụng Strategy Pattern và đề tài

Sơ đồ lớp:



Lý do áp dụng:

Tính linh hoạt: Strategy Pattern cho phép chọn một chiến lược validate phù hợp với yêu cầu cụ thể của ứng dụng. Bằng cách này, ta có thể dễ dàng thay đổi hoặc mở rộng quy tắc validate mà không cần thay đổi cấu trúc của lớp TextBox. Ví dụ, nếu ta muốn thêm một quy tắc validate mới hoặc thay đổi quy tắc validate hiện có, ta chỉ cần tạo một lớp mới thực hiện giao diện IValidator mà không ảnh hưởng đến TextBox.

Sử dụng trong các form dùng để tạo tài khoản cho nhân viên hoặc khách hàng, khi đó cần các thông tin như tên, gmail, số điện thoại,... sẽ dễ dàng thực hiện việc validate các input đó.

Code:

A computer screen shot of text

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Ví dụ form đăng ký tài khoản cho nhân viên

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

## Template Pattern

### Định nghĩa

Template Pattern là một mẫu thiết kế hành vi xác định khung của thuật toán trong superclass nhưng cho phép các lớp con ghi đè các bước cụ thể của thuật toán mà không thay đổi cấu trúc của nó.

### Mục đích và lợi ích của việc sử dụng Template Pattern

Template Pattern cho phép các nhà phát triển xác định cấu trúc tổng thể của thuật toán đồng thời cho phép linh hoạt trong việc triển khai các bước riêng lẻ. Điều này thúc đẩy việc tái sử dụng mã và đơn giản hóa việc bảo trì vì các thay đổi đối với thuật toán có thể được thực hiện ở một vị trí trung tâm.

### Các thành phần của Template Pattern

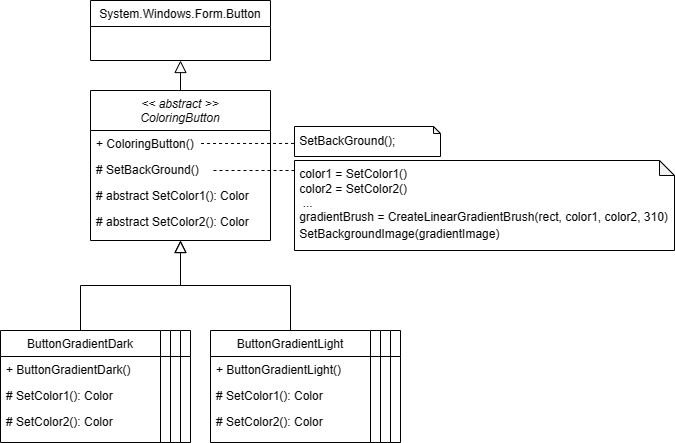
* Abstract class: Cốt lõi của Template Pattern là một lớp trừu tượng xác định template method, phác thảo cấu trúc của thuật toán. Lớp này cũng có thể bao gồm các phương thức trừu tượng mà các lớp con phải thực hiện.
* Concrete classes: Các lớp cụ thể mở rộng lớp trừu tượng và cung cấp các triển khai cho các phương thức trừu tượng. Các lớp này tùy chỉnh hành vi của thuật toán bằng cách ghi đè các bước cụ thể nếu cần.
* Template method: Điều phối thuật toán bằng cách gọi trình tự các bước được xác định trong lớp trừu tượng. Các lớp con có thể ghi đè các bước nhất định để điều chỉnh thuật toán theo yêu cầu cụ thể của chúng.



*Hình 3.8 Template Pattern*

### Áp dụng Template Pattern và đề tài

Sơ đồ lớp:



Lý do áp dụng:

Tính linh hoạt và tái sử dụng: Template Method Pattern cho phép định nghĩa khung cho một thuật toán trong một lớp cha và để các lớp con triển khai các bước cụ thể của thuật toán đó.

Trong trường hợp này, lớp cha định nghĩa cấu trúc chung cho việc tạo gradient, trong khi các lớp con cung cấp các màu sắc cụ thể để tạo ra gradient mong muốn. Điều này giúp tạo ra một hệ thống linh hoạt và dễ mở rộng, nơi ta có thể dễ dàng thêm các button mới với các màu sắc gradient khác nhau mà không cần viết lại mã.

Từ đó tạo được các component custom với nhiều màu sắc như giao diện web để tăng tính thẩm mỹ cho ứng dụng.

Code:

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Có thể thấy ở giao diện POS, sử dụng các button có màu sắc gradient. Và các component này có thể kéo thả như ta thấy ở thanh Toolbox

## Observer Pattern

### Định nghĩa

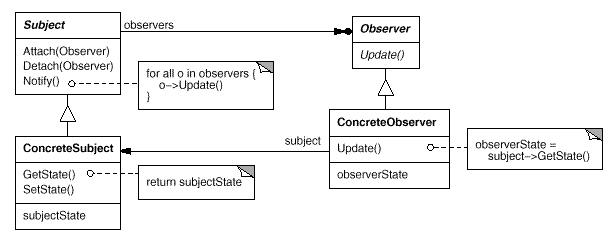
Observer Pattern là một mẫu thiết kế hành vi mà xác định cấu trúc cơ bản của một chương trình và tự động thay đổi trong trạng thái của đối tượng, thông báo cho tất cả các đối tượng đã đăng ký khi trạng thái của đối tượng thay đổi.

### Mục đích và lợi ích của việc sử dụng Observer Pattern

Observer Patterncho phép các đối tượng theo dõi và phản ứng khi trạng thái của một đối tượng khác thay đổi. Điều này giúp tạo ra một hệ thống linh hoạt và phản ứng, nơi mà các thành phần có thể tương tác mà không cần sự can thiệp trực tiếp từ phía người lập trình.

### Các thành phần của Observer Pattern

* Subject: Đóng vai trò là tâm điểm của mẫu người quan sát, chứa dữ liệu quan trọng và thông tin trạng thái. Nó cho phép người quan sát đăng ký hoặc hủy đăng ký dựa trên sở thích nhận thông tin cập nhật của họ.
* Observer: Là những đối tượng quan tâm đến trạng thái của subject. Họ tự đăng ký với chủ thể và nhận thông báo bất cứ khi nào trạng thái của chủ thể có thay đổi.
* Concrete Subject: là các implement của subject interface. Họ duy trì một danh sách những Observer và thông báo cho họ về bất kỳ thay đổi trạng thái nào.
* Concrete Observer: là các implement của observer interface, xác định các hành động cụ thể sẽ được thực hiện khi nhận được thông báo từ chủ thể.



*Hình 3.9 Observer Pattern*

### Áp dụng Observer Pattern và chủ đề

Sơ đồ lớp:

Lý do áp dụng:

Tính linh hoạt: Observer Pattern cho phép các đối tượng theo dõi một đối tượng chủ đề (subject) mà không cần biết về sự tồn tại của nhau. Trong trường hợp này, các đối tượng cảnh báo về số lượng tồn kho có thể theo dõi sự kiện khi một thanh toán được thực hiện mà không làm ảnh hưởng đến hệ thống khác.

Tách rời logic: Việc sử dụng Observer Pattern giúp tách rời logic cảnh báo số lượng tồn kho từ logic của việc thực hiện thanh toán. Điều này giúp dễ dàng bảo trì và mở rộng các chức năng một cách độc lập.

Code:

A computer screen shot of a program

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

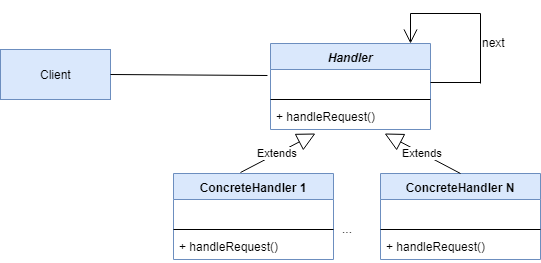
## Chain of Reponsibility Pattern

### Chain of Responsibility Pattern là gì?

Chain of Responsibility là một behavioral pattern cho phép bạn chuyển các yêu cầu dọc theo một chuỗi trình xử lý. Khi nhận được yêu cầu, mỗi trình xử lý sẽ quyết định xử lý yêu cầu hoặc chuyển nó cho trình xử lý tiếp theo trong chuỗi.

### Các thành phần của Chain of Responsibility Pattern

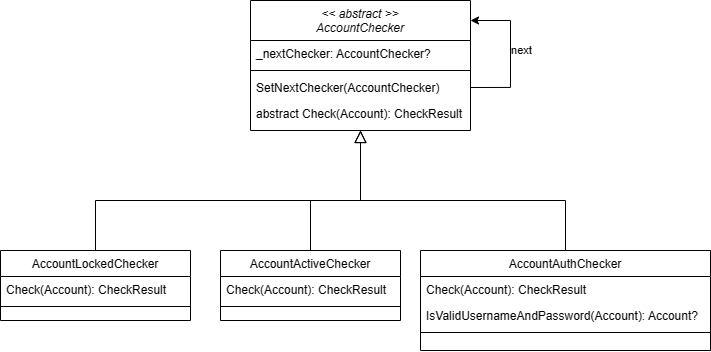
* Handler interface: khai báo giao diện, chung cho tất cả các trình xử lý cụ thể. Nó thường chỉ chứa một phương thức duy nhất để xử lý các yêu cầu, nhưng đôi khi nó cũng có thể có một phương thức khác để thiết lập trình xử lý tiếp theo trong chuỗi.
* Concrete handlers: Là các lớp cụ thể thực hiện Handler interface và chứa logic để xử lý các loại yêu cầu cụ thể. Chúng quyết định liệu họ có xử lý yêu cầu hay chuyển tiếp nó cho đối tượng xử lý tiếp theo.
* Client: Là đối tượng khởi tạo yêu cầu và chuyển nó cho đối tượng xử lý đầu tiên trong chuỗi. Khách hàng không cần biết về cấu trúc của chuỗi và các đối tượng xử lý cụ thể.



*Hình 3.10 Chain of Responsibility Pattern*

### Áp dụng Chain of Responsibility và đề tài

Sơ đồ lớp:



Lý do áp dụng:

Chain of Responsibility Pattern cho phép ta xác định một chuỗi các đối tượng xử lý yêu cầu theo một thứ tự nhất định.

Trong trường hợp này, mỗi AccountChecker là một đối tượng xử lý có thể quyết định xem một tài khoản đăng nhập có được chấp nhận hay không dựa trên các tiêu chí khác nhau như xác thực, tài khoản bị khóa, và tài khoản có được kích hoạt hay không.

Bằng cách này, ta có thể dễ dàng thêm, loại bỏ hoặc thay đổi thứ tự của các bước kiểm tra mà không cần thay đổi nhiều mã. Ví dụ như thêm các lớp kiểm tra SQL Injection, hay tự động đính kèm hậu tố “@gmail.com” trước khi xác thực tài khoản khi đăng nhập,...

Code:

A computer screen shot of a code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

A computer screen shot of a program

Description automatically generated

Gọi và kiểm tra ở lớp Controller

A computer screen with many colorful text

Description automatically generated

## Decorator Pattern

### Định nghĩa

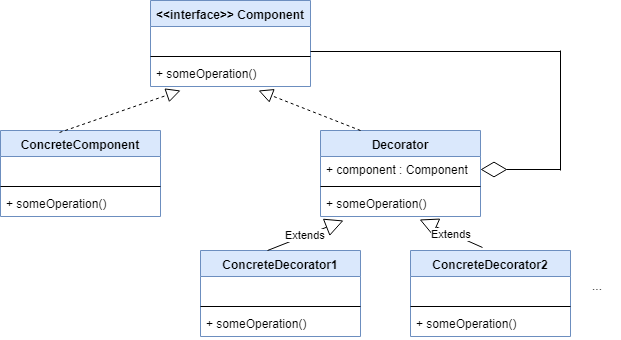
Decorator Pattern thuộc danh mục mẫu thiết kế cấu trúc. Nó cho phép thêm hành vi vào từng đối tượng một cách linh hoạt mà không ảnh hưởng đến hành vi của các đối tượng khác trong cùng một lớp. Mẫu này đặc biệt hữu ích khi nhu cầu về chức năng bổ sung thay đổi và không được xác định trước.

### Mục đích sử dụng

Decorator Pattern gắn các trách nhiệm bổ sung cho các đối tượng một cách linh hoạt. Nó cung cấp một giải pháp thay thế linh hoạt cho việc phân lớp để mở rộng chức năng. Bằng cách cho phép các đối tượng được tạo theo cách đệ quy, nó thúc đẩy khả năng sử dụng lại mã và khả năng mở rộng.

### Các thành phần chính của Decorator Pattern

* Component Interface: Định nghĩa interface cho các đối tượng có thể được thêm trách nhiệm một cách linh hoạt.
* Concrete Component: Thành phần cụ thể đại diện cho đối tượng cơ bản mà có thể thêm chức năng bổ sung vào.
* Decorator: Là một lớp trừu tượng implement Component Interface và duy trì tham chiếu đến Component object. Nó cũng cung cấp một giao diện để thêm hành vi bổ sung.



*Hình 3.11 Decorator Pattern*

### Áp dụng Decorator Pattern vào đề tài

Sơ đồ lớp:

Lý do áp dụng:

Mở rộng chức năng: Decorator Pattern cho phép ta mở rộng chức năng của một đối tượng mà không cần sửa đổi code hiện có. Bằng cách này, ta có thể áp dụng các loại giảm giá mới một cách linh hoạt và dễ dàng.

Tách biệt trách nhiệm: Mỗi decorator chịu trách nhiệm cho một loại giảm giá cụ thể, giữ cho code dễ hiểu và dễ bảo trì.

Dễ dàng kết hợp: Ta có thể kết hợp nhiều decorator với nhau để áp dụng nhiều loại giảm giá hoặc mở rộng bổ sung tính phí ship cho hóa đơn một cách linh hoạt.

Code:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen with text on it

Description automatically generated

Code áp dụng khi có kích hoạt từ giao diện, người dùng áp dụng thẻ thành viên lúc thanh toán, hoặc sử dụng voucher

A screen shot of a computer program

Description automatically generated

## Command Pattern

### Command Pattern là gì?

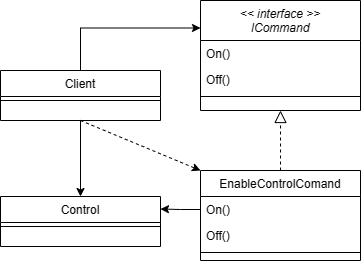
Comamnd Pattern là một cách đóng gói yêu cầu dưới dạng đối tượng, cho phép tham số hóa máy khách bằng hàng đợi, yêu cầu và thao tác. Mục đích chính của nó là tách rời người gửi yêu cầu khỏi người nhận, mang lại sự linh hoạt và khả năng mở rộng trong thiết kế phần mềm.

### Các thành phần của Comamnd Pattern

* Command: Đại diện cho một hoạt động sẽ được thực hiện. Nó gói gọn tất cả các thông tin cần thiết để thực hiện một hành động cụ thể. Sự trừu tượng hóa này cho phép lệnh được truyền đi khắp nơi dưới dạng đối tượng hạng nhất, giúp bạn có thể tạo hàng chờ, ghi nhật ký hoặc hoàn tác các hoạt động.
* Invoker: Còn được gọi là "sender" hoặc "client", nó yêu cầu lệnh để thực hiện một thao tác. Người gọi không biết lệnh sẽ được thực thi như thế nào; nó chỉ biết cách kích hoạt nó. Sự tách biệt các mối quan tâm này đảm bảo sự liên kết lỏng lẻo giữa người gọi và hoạt động thực tế.
* Receiver: Xác định interface để thực hiện các hoạt động liên quan đến một lệnh. Nó biết cách thực hiện thao tác và là mục tiêu của yêu cầu. Bộ thu có thể là bất kỳ đối tượng nào có khả năng thực hiện hành động cụ thể, cho phép thiết kế hệ thống linh hoạt.
* Client: Tạo một lệnh và thiết lập bộ thu của nó. Máy khách chịu trách nhiệm tạo các phiên bản lệnh cụ thể và liên kết chúng với các bộ thu thích hợp. Nó đóng vai trò là người điều phối, xác định lệnh nào sẽ được thực thi và khi nào.

### Áp dụng Command Pattern và đề tài

Sơ đồ lớp:



Lý do áp dụng:

Phân tách yêu cầu và thực hiện: Command Pattern tách biệt yêu cầu từ thực thi của chúng. Điều này giúp loại bỏ sự phụ thuộc giữa người gửi yêu cầu và người thực hiện, cung cấp sự linh hoạt và dễ bảo trì hơn.

Không còn dồn quá nhiều code vào một hàm khi muốn bật hoặc tắt component.

Code:

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Giao diện khi chuyển đổi radio button

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated