TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CUỐI KỲ MÔN MẪU THIẾT KẾ**

**Áp dụng mẫu thiết kế vào ứng dụng**

**quản lý cửa hàng mỹ phẩm**

*Người hướng dẫn*: **ThS VŨ ĐÌNH HỒNG**

*Người thực hiện*: **NGUYỄN TRƯỜNG AN - 51800002**

**LÊ THỊ MAI - 51800078**

**TRẦN PHƯƠNG NGỌC ANH - 51702061**

**HUỲNH ANH TÀI - 51702171**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CUỐI KỲ MÔN MẪU THIẾT KẾ**

**Áp dụng mẫu thiết kế vào ứng dụng**

**quản lý cửa hàng mỹ phẩm**

*Người hướng dẫn*: **ThS VŨ ĐÌNH HỒNG**

*Người thực hiện*: **NGUYỄN TRƯỜNG AN**

**LÊ THỊ MAI**

**TRẦN PHƯƠNG NGỌC ANH**

**HUỲNH ANH TÀI**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021**

LỜI CẢM ƠN

Để hoàn thành được đồ án này, chúng em xin chân thành cảm ơn sự truyền đạt và hướng dẫn tận tình của giáo viên hướng dẫn ThS Vũ Đình Hồng, cũng đã cho chúng em cơ hội tìm hiểu và nghiên cứu về môn học, giúp chúng em có thêm nhiều kiến thức hơn về các mẫu thiết kế qua đó vận dụng hiểu biết của mình vào bài toán thực tiễn.

Tuy nhiên, do trong quá trình tìm hiểu còn nhiều hạn chế không tránh khỏi, nếu có sai sót mong thầy xem xét và đóng góp ý kiến để chúng em ngày một hoàn thiện và hoàn thành tốt hơn.

Chúng em xin chân thành cảm ơn thầy.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Chúng tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của ThS 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 đồ 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 đồ 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 03 tháng 04 năm 2021*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Trường An Lê Thị Mai*

*Trần Phương Ngọc Anh Huỳnh Anh Tài*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Design Pattern là các phương pháp tối ưu có khả năng giải quyết các vấn đề mang tính chất tương tự nhau, vận dụng vào việc tái sử dụng code, hạn chế được những lỗi tiềm ẩn, giúp lập trình viên dễ dàng mở rộng và kiểm tra, đáp ứng được các yêu cầu thay đổi và những tính năng mới của dự án. Chính vì những lợi ích mà nó mang lại, design pattern trở thành một công cụ đắc lực cho các nhà phát triển phần mềm nói chung.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc68386836)

[TÓM TẮT iv](#_Toc68386837)

[MỤC LỤC 1](#_Toc68386838)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 3](#_Toc68386839)

[CHƯƠNG 1 – TỔNG QUAN 5](#_Toc68386840)

[CHƯƠNG 2 – NGHIÊN CỨU THỰC NGHIỆM 8](#_Toc68386841)

[2.1 Bài toán áp dụng 8](#_Toc68386842)

[2.2 Áp dụng mô hình design pattern 8](#_Toc68386843)

[2.2.1 Template Method 8](#_Toc68386844)

[2.2.1.1 Lý do sử dụng 8](#_Toc68386845)

[2.2.1.2 Cách áp dụng 9](#_Toc68386846)

[2.2.1.3 Sơ đồ lớp của Template Method 10](#_Toc68386847)

[2.2.2 Factory Method 10](#_Toc68386848)

[2.2.2.1 Lý do sử dụng 10](#_Toc68386849)

[2.2.2.2 Cách áp dụng 10](#_Toc68386850)

[2.2.2.3 Sơ đồ lớp của Factory Method 12](#_Toc68386851)

[2.2.3 Singleton Pattern 12](#_Toc68386852)

[2.2.3.1 Lý do sử dụng 12](#_Toc68386853)

[2.2.3.2 Cách áp dụng 12](#_Toc68386854)

[2.2.3.3 Sơ đồ lớp của Singleton Pattern 13](#_Toc68386855)

[2.2.4 Abstract Factory 13](#_Toc68386856)

[2.2.4.1 Lý do sử dụng 13](#_Toc68386857)

[2.2.4.2 Cách áp dụng 14](#_Toc68386858)

[2.2.4.3 Sơ đồ lớp của Abstract Factory 16](#_Toc68386859)

[2.2.5 Command Pattern 16](#_Toc68386860)

[2.2.5.1 Lý do sử dụng 16](#_Toc68386861)

[2.2.5.2 Cách áp dụng 17](#_Toc68386862)

[2.2.5.3 Sơ đồ lớp của Command Pattern 19](#_Toc68386863)

[2.2.6 Mediator Pattern 19](#_Toc68386864)

[2.2.6.1 Lý do sử dụng 19](#_Toc68386865)

[2.2.6.2 Cách áp dụng 19](#_Toc68386866)

[2.2.6.3 Sơ đồ lớp của Mediator Pattern 23](#_Toc68386867)

[2.2.7 Memento Pattern 23](#_Toc68386868)

[2.2.7.1 Lý do sử dụng 23](#_Toc68386869)

[2.2.7.2 Cách áp dụng 24](#_Toc68386870)

[2.2.7.3 Sơ đồ lớp của Memento Pattern 28](#_Toc68386871)

[CHƯƠNG 3 – TỔNG KẾT 29](#_Toc68386872)

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 1.1: Nhóm Creational Pattern 6](#_Toc68383741)

[Hình 1.2: Nhóm Structural Pattern 6](#_Toc68383742)

[Hình 1.3: Nhóm Behavioral Pattern 7](#_Toc68383743)

[Hình 2.1: Lớp ControlScreen 9](#_Toc68383744)

[Hình 2.2: Hai bộ khung của hàm screenClick và firstActive 9](#_Toc68383745)

[Hình 2.3: Sơ đồ lớp của Template Method 10](#_Toc68383746)

[Hình 2.4: Form Factory 11](#_Toc68383747)

[Hình 2.5: Dispatcher được sử dụng 11](#_Toc68383748)

[Hình 2.6: Sơ đồ lớp của Factory Method 12](#_Toc68383749)

[Hình 2.7: Singleton ở lớp AccountDAO 13](#_Toc68383750)

[Hình 2.8: Sơ đồ lớp của Singleton Pattern 13](#_Toc68383751)

[Hình 2.9: Lớp DatabaseFactory 15](#_Toc68383752)

[Hình 2.10: Lớp DatabaseMySql 15](#_Toc68383753)

[Hình 2.11: Lớp DatabaseMySql 16](#_Toc68383754)

[Hình 2.12: Nơi sử dụng Abstract Factory 16](#_Toc68383755)

[Hình 2.13: Sơ đồ lớp của Abstract Factory 16](#_Toc68383756)

[Hình 2.14: Lớp CommandButtonManagement 17](#_Toc68383757)

[Hình 2.15: Lớp CommandButtonImp 18](#_Toc68383758)

[Hình 2.16: Sử dụng Command Pattern ở lớp QuanLyNhanVienControl 18](#_Toc68383759)

[Hình 2.17: Sơ đồ lớp của Command Pattern 19](#_Toc68383760)

[Hình 2.18: Lớp CenterMediatorControlScreen 20](#_Toc68383761)

[Hình 2.19: Lớp CenterMediatorImpl 21](#_Toc68383762)

[Hình 2.20: Các hàm xử lý trong lớp CenterMediatorImpl 21](#_Toc68383763)

[Hình 2.21: Kiểu dữ liệu TypeDataChange 22](#_Toc68383764)

[Hình 2.22: Nơi sử dụng Mediator Pattern 22](#_Toc68383765)

[Hình 2.23: Hàm notifyOtherControlDataChange 22](#_Toc68383766)

[Hình 2.24: Thông báo khi type product có sự thay đổi 23](#_Toc68383767)

[Hình 2.25: Sơ đồ lớp của Mediator Pattern 23](#_Toc68383768)

[Hình 2.26: Lớp CardModel 25](#_Toc68383769)

[Hình 2.27: Lớp MementoCard 25](#_Toc68383770)

[Hình 2.28: Lớp CardCareTaker 26](#_Toc68383771)

[Hình 2.29: Nơi sử dụng Memento Pattern 26](#_Toc68383772)

[Hình 2.30: Xóa mọi trạng thái giỏ hàng khi bấm nút refresh 27](#_Toc68383773)

[Hình 2.31: Nơi dùng CardCareTaker 27](#_Toc68383774)

[Hình 2.32: Hoàn tác lại trạng thái cho CardModel khi bấm nút hoàn tác 27](#_Toc68383775)

[Hình 2.33: Sơ đồ lớp của Memento Pattern 28](#_Toc68383776)

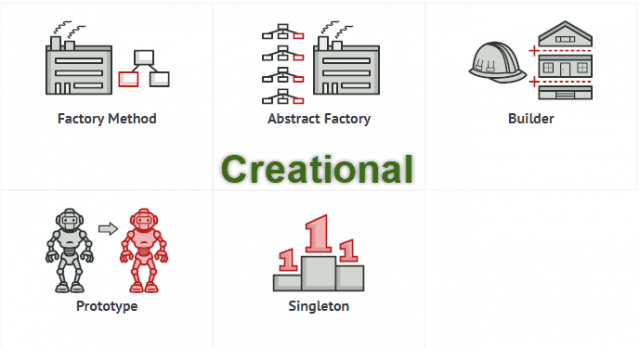
CHƯƠNG 1 – TỔNG QUAN

Mẫu thiết kế được biết đến như một sản phẩm được phát triển hoàn chỉnh để có thể tái sử dụng về sau. Trong công nghệ thông tin, mẫu thiết kế, hay còn gọi là design pattern, là mẫu chuẩn tối ưu nhất, là giải pháp chung để giải quyết các vấn đề phổ biến có tính chất tương tự nhau, lặp đi lặp lại nhiều lần khi thiết kế phần mềm.

Mẫu thiết kế tập hợp các phương pháp tổng thể nhằm tối ưu hóa, tái sử dụng và giải quyết cho các vấn đề chung trong thiết kế phần mềm, giúp lập trình viên linh hoạt hơn trong việc nâng cấp và bảo trì code, cải thiện đáng kể những rủi ro, các lỗi tiềm ẩn mà người dùng có thể gặp phải.

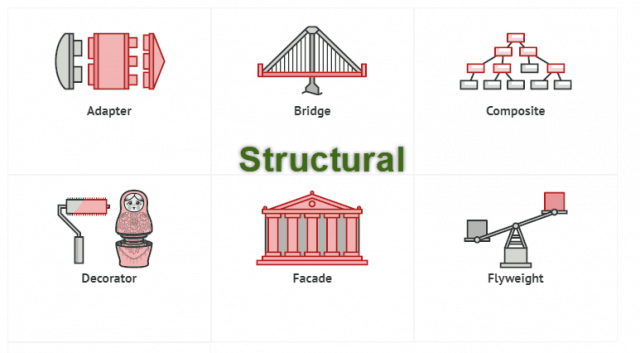
Hệ thống design pattern gồm 23 mẫu được chia thành 3 nhóm bao gồm: nhóm creational, nhóm structural và nhóm behavioral.

Nhóm creational (nhóm khởi tạo) sẽ hữu ích trong việc khởi tạo đối tượng; nhóm structural (nhóm cấu trúc) sẽ thiết lập, định nghĩa mối quan hệ giữa các đối tượng có liên quan và cuối cùng, nhóm behavioral (nhóm hành vi) sẽ tập trung thực hiện hành vi của các đối tượng, xác định các mẫu giao tiếp chung. Tùy theo nhu cầu và mục đích sử dụng mà mỗi nhóm sẽ có các mẫu thiết kế tương ứng.



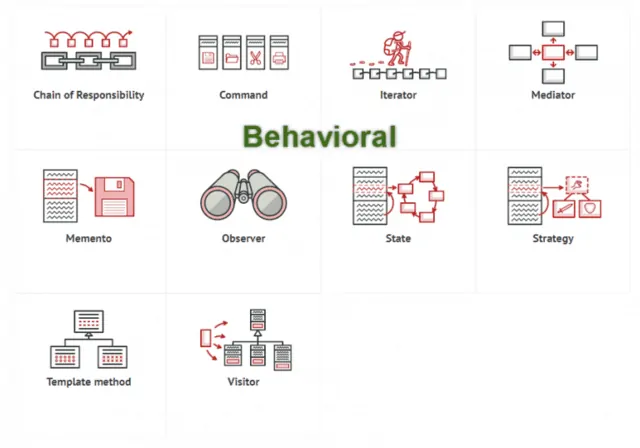
Hình 1.1: Nhóm Creational Pattern

(Nguồn: https://itviec.com/blog/design-pattern/)



Hình 1.2: Nhóm Structural Pattern

(Nguồn: https://itviec.com/blog/design-pattern/)



Hình 1.3: Nhóm Behavioral Pattern

(Nguồn: https://itviec.com/blog/design-pattern/)

**CHƯƠNG 2 –** NGHIÊN CỨU THỰC NGHIỆM

1. Bài toán áp dụng

Bài toán quản lý cửa hàng mỹ phẩm được chọn làm đề tài cơ sở để áp dụng các mô hình design pattern. Ứng dụng được xây dựng nhằm giải quyết vấn đề quản lý và thống kê các hàng hóa của doanh nghiệp với một số lượng lớn, tránh gây ra sai sót và những tổn thất không đáng có. Do tính chất công việc đòi hỏi nhiều công đoạn phức tạp và khó khăn, cũng như nhiều vấn đề cần được giải quyết cho các câu hỏi xoay quanh việc phân phối, xuất nhập sản phẩm, quản lý sao cho hiệu quả nhất, lưu trữ khối lượng lớn dữ liệu, khả năng chịu tải của hệ thống quản lý, làm sao để tăng năng suất làm việc.

Mặt khác, không chỉ tập trung phát triển các chức năng chính, với đa dạng hóa các công việc cần được kiểm soát thì hiệu suất của hệ thống cũng rất quan trọng. Chính vì vậy, áp dụng các phương pháp design pattern sẽ giúp bài toán trở nên đơn giản và tối ưu hơn, giảm thiểu tối đa những rủi ro có thể xảy ra trong quá trình xây dựng ứng dụng và trong cả việc quản lý bán hàng.

1. Áp dụng mô hình design pattern

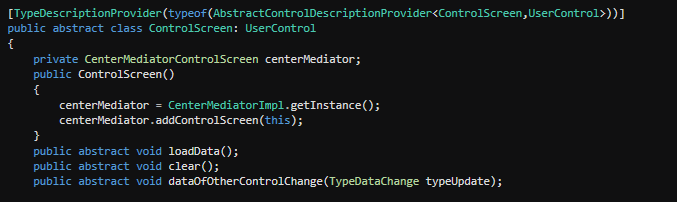
Sau đây là các mô hình design pattern được áp dụng vào bài toán.

1. Template Method
2. Lý do sử dụng

Template Pattern giúp ta tạo nên bộ khung của một chức năng và cho phép lớp con định nghĩa lại một số hành vi trong bộ khung. Mô hình này áp dụng khi thay đổi các UserControl, nó sẽ thực hiện các hành vi trong một bộ khung đã được định nghĩa trước đó, tránh việc lặp đi lặp lại code ở nhiều nơi, đồng thời đối với việc các lớp con có hành vi khác nhau, nó giúp định nghĩa lại những hành vi riêng biệt đó, hạn chế được những ảnh hưởng đến các hành vi có cấu trúc giống nhau giữa lớp con, cũng như việc khởi tạo riêng cho các phương thức gây nên tình trạng dư thừa không đáng có. Về lâu về dài, điều này sẽ hạn chế việc thay đổi code ở nhiều nơi mà chỉ cần chỉnh sửa ở một nơi nhất định.

1. Cách áp dụng

Template Method được sử dụng trong ứng dụng thông qua lớp ControlScreen. Lớp ControlScreen sẽ chứa hai bộ khung code template có sẵn trong hai hàm: screenClick và firstActive. Hàm screenClick được kích hoạt khi các lớp con của ControlScreen được bấm để chuyển đổi với nhau. Và khi hàm screenClick được kích hoạt thì nó thực hiện một bộ khung có 4 hoạt động: đem lớp con của ControlScreen được bấm hiện lên, focus vào lớp đó và cuối cùng là tùy thuộc vào lớp con là lớp gì sẽ thực hiện hai method là loadData và clear. Chính vì 2 hoạt động loadData và clear được thực hiện tùy vào lớp con của ControlScreen nên nó sẽ được định nghĩa ở lớp con. Tương tự hàm firstActive được dành cho việc hiển thị các lớp con ngay từ đầu. Và cũng như hàm screenClick, khi được kích hoạt, nó cũng sẽ thực hiện theo một bộ khung có sẵn.

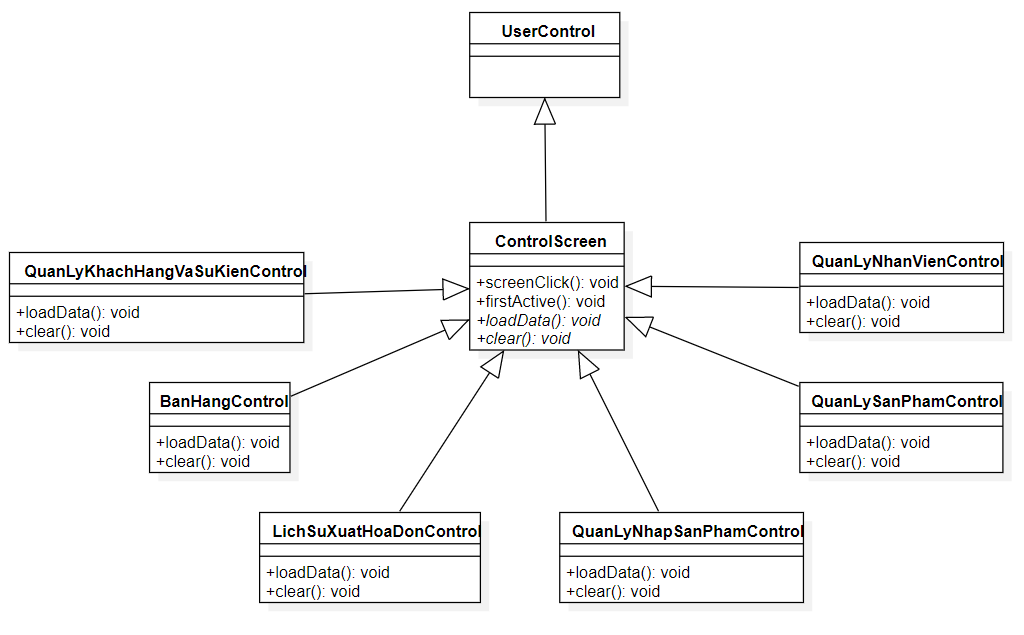


Hình 2.1: Lớp ControlScreen



Hình 2.2: Hai bộ khung của hàm screenClick và firstActive

1. Sơ đồ lớp của Template Method



Hình 2.3: Sơ đồ lớp của Template Method

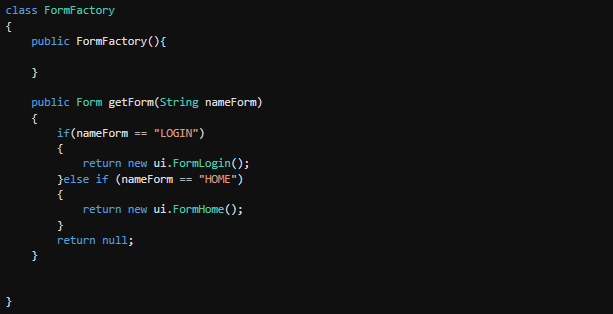
1. Factory Method
2. Lý do sử dụng

Factory Pattern là mô hình giúp tạo ra các đối tượng theo yêu cầu, đối với bài toán, nó được áp dụng để tạo ra các đối tượng Form khác nhau phụ thuộc vào Form muốn khởi tạo, đồng thời mẫu thiết kế này cũng giúp ta có một cấu trúc code dễ nhìn, dễ quản lý hơn rất nhiều khi muốn tạo ra một đối tượng.

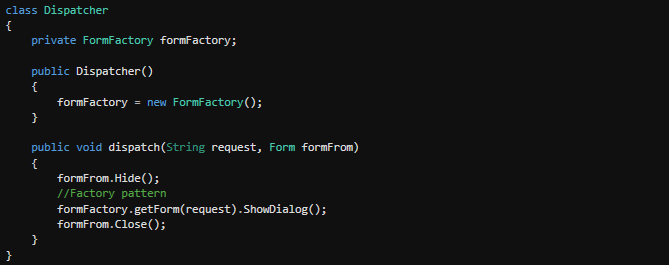
1. Cách áp dụng

Factory Pattern được sử dụng như sau:

Lớp Dispatcher là lớp được dùng để chuyển đổi giữa các Form với nhau thông qua hàm dispatch. Ở hàm này ta cần tạo ra một lớp con của lớp Form tùy thuộc vào biến request, và công việc tạo các lớp con của lớp Form sẽ được thực hiện bởi lớp FormFactory. Do đó khi hàm dispatch được kích hoạt, nó sẽ gọi hàm getForm ở lớp FormFactory và truyền biến request qua để lớp này có thể tạo ra các lớp con của Form tùy thuộc vào biến đó và trả về kết quả về hàm dispatch ở lớp Dispatcher để nó thực hiện các hoạt động dựa trên lớp con của Form đã được tạo ra.

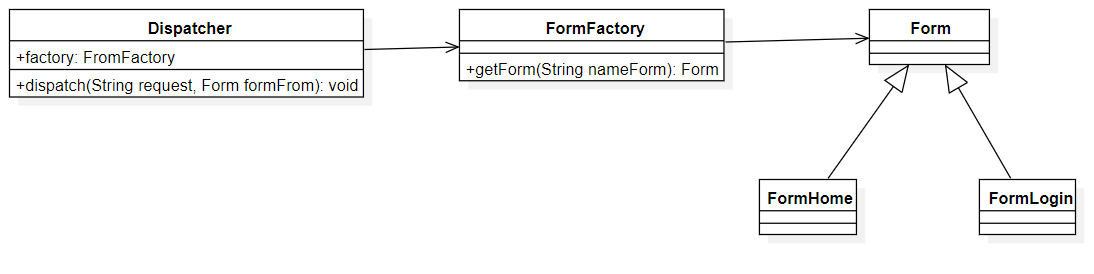


Hình 2.4: Form Factory



Hình 2.5: Dispatcher được sử dụng

1. Sơ đồ lớp của Factory Method



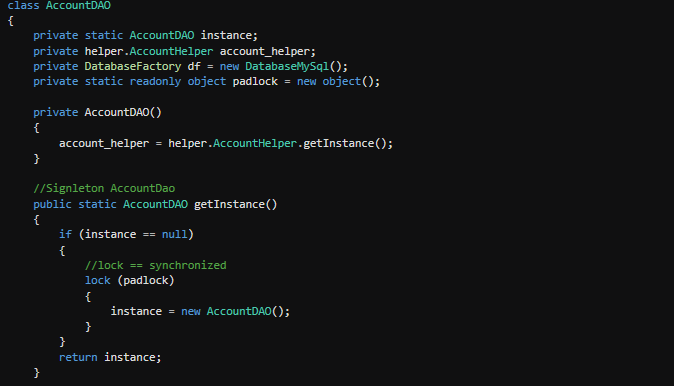
Hình 2.6: Sơ đồ lớp của Factory Method

1. Singleton Pattern
2. Lý do sử dụng

Singleton Pattern là một trong những mô hình được sử dụng phổ biến nhất, nó giúp cho việc tạo đối tượng chỉ diễn ra đúng một lần từ khi khởi tạo ứng dụng. Mô hình này giúp giữ các instance của những lớp chỉ cần khởi tạo một lần như: lớp Dao, Helper,… điều đó giúp cho việc gọi đến những lớp này dễ dàng và quan trọng hơn là dữ liệu trong những lớp đó sẽ không bị thay đổi khi gọi lại, làm cho quá trình xây dựng ứng dụng được liền mạch và xuyên suốt.

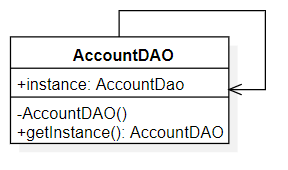
1. Cách áp dụng

Lấy ví dụ ở lớp AccountDAO khi khởi tạo, ta sẽ không thể gọi hàm khởi tạo của lớp AccountDAO như cách thông thường vì khi sử dụng Singleton Pattern, hàm khởi tạo được đặt ở chế độ private. Chính vì vậy khi muốn gọi đến lớp AccountDAO thì ta sẽ gọi qua hàm static getInstance và kết quả trả về là một instance AccountDAO. Hàm này có nhiệm vụ kiểm tra biến static với tên instance và type là AccountDAO là có khác null không. Nếu bằng null thì hàm getInstance sẽ gọi hàm khởi tạo ở lớp AccountDAO và gán cho biến instance và trả về biến instance. Ngược lại thì nếu biến instance khác null thì hàm getInstance sẽ trả thẳng ra instance mà không gọi hàm khởi tạo của AccountDAO. Từ những điều trên thì nó sẽ giúp project của ta luôn luôn chỉ có một instance duy nhất của lớp AccountDAO. Tương tự như lớp AccountDAO thì các lớp ở thư mục dao, helper, lớp DatabaseMySql, FrontController, CenterMeidatorImpl đều hoạt động như vậy.



Hình 2.7: Singleton ở lớp AccountDAO

1. Sơ đồ lớp của Singleton Pattern



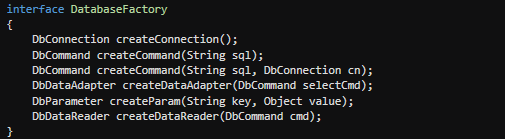
Hình 2.8: Sơ đồ lớp của Singleton Pattern

1. Abstract Factory
2. Lý do sử dụng

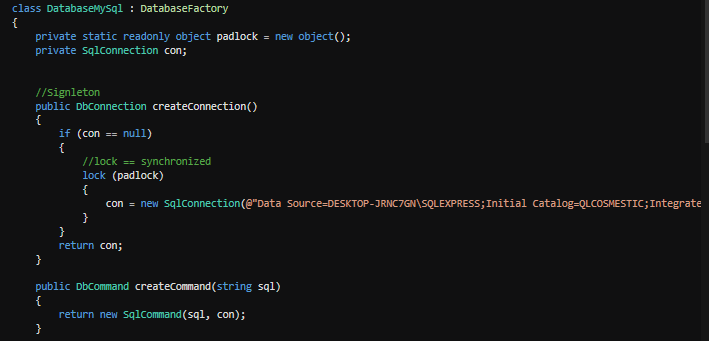
Abstract Factory Pattern là mẫu thiết kế giúp quản lý việc trả về các đối tượng liên quan với nhau. Do đó mẫu thiết kế này tạo ra một nơi thực hiện một việc duy nhất là quản lý, tạo và trả về các đối tượng kết nối database, command, param,… Hiện tại ứng dụng chỉ kết nối với một loại cơ sở dữ liệu chính là SQL nhưng sau này nếu kết nối với một loại cơ sở dữ liệu khác thì mô hình vẫn hỗ trợ để có thể gọi các method trả về đối tượng trước đó đã sử dụng bằng cách thêm một lớp con thừa kế lớp cơ sở dữ liệu của lớp cha mà đã định nghĩa các hàm khởi tạo và điều chỉnh các đối tượng được tạo ra tùy thuộc vào cơ sở dữ liệu đó mà không ảnh hưởng đến lớp cơ sở dữ liệu cũ, cũng như chương trình đã được thực hiện trước đó. Mặt khác, ta cũng giảm thiểu đi việc sử dụng if else quá nhiều để tạo các đối tượng ở cơ sở dữ liệu mà ta muốn.

1. Cách áp dụng

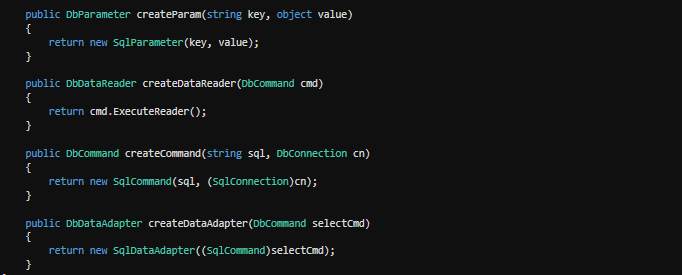
Ta có một interface DatabaseFactory, lớp này sẽ thực hiện việc định nghĩa các hành vi tạo các đối tượng Db sẽ có ở các lớp con. Và các lớp con sẽ thừa kế lớp DatabaseFactory định nghĩa lại các hành vi đó và tùy thuộc vào lớp con đó muốn trả về giá trị gì thì ở lớp con sẽ thực hiện trả về các giá trị đó. Giống như hiện tại lớp con DatabaseMySql sẽ thực hiện việc thừa kế lớp DatabaseFactory và định nghĩa lại các hành vi và trả về các đối tượng có dạng là Sql. Khi sử dụng ở lớp AccountHelper thì ta chỉ việc gọi lại class DatabaseFactory và phụ thuộc mình muốn khởi tạo các đối tượng loại Database nào thì mình sẽ khởi tạo Database đó ví dụ ta muốn đối tượng là là dạng sql thì ta khởi tạo DatabaseMySql. Sau đó ta chỉ việc gọi các hành vi tạo các đối tượng sử dụng ở các hàm checkAccountExistDb, selectCurrentUser,… Sau này ta có muốn thêm một lớp database nào khác thì ta chỉ cần tạo lớp database đó, thừa kế lại lớp DatabaseFactory, và định nghĩa lại các hành vi tạo đối tượng dưới dạng Db mà lớp đó muốn trả về và chỗ nào muốn sử dụng thì ta chỉ việc khởi tạo lớp Database đó dưới dạng lớp DatabaseFactory là được. Tương tự như các lớp ở trong thực mục dao, helper cũng thực hiện việc khởi tạo các object database như ở lớp AccountHelper.



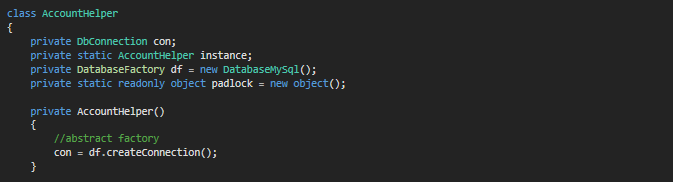
Hình 2.9: Lớp DatabaseFactory



Hình 2.10: Lớp DatabaseMySql

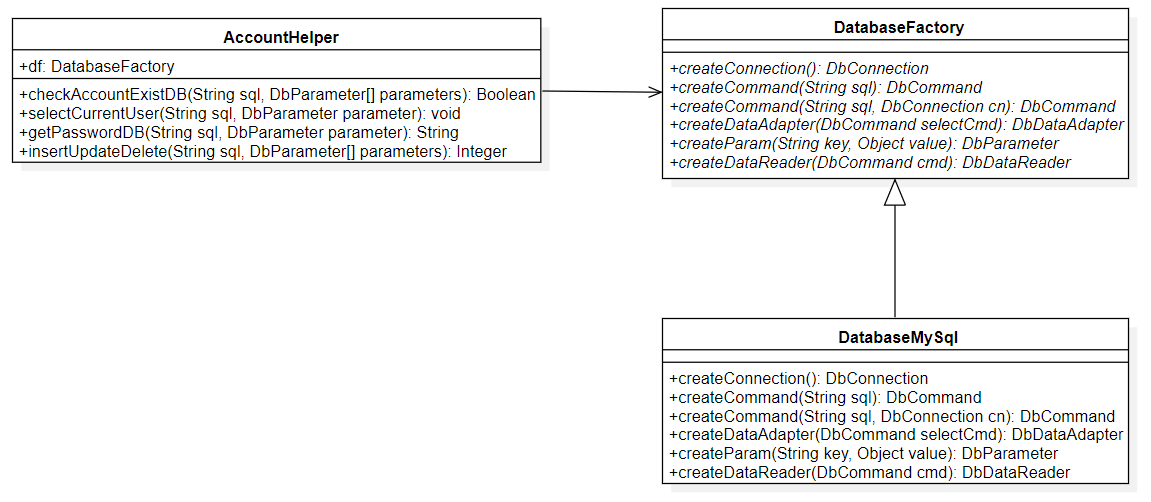


Hình 2.11: Lớp DatabaseMySql



Hình 2.12: Nơi sử dụng Abstract Factory

1. Sơ đồ lớp của Abstract Factory



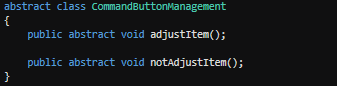
Hình 2.13: Sơ đồ lớp của Abstract Factory

1. Command Pattern
2. Lý do sử dụng

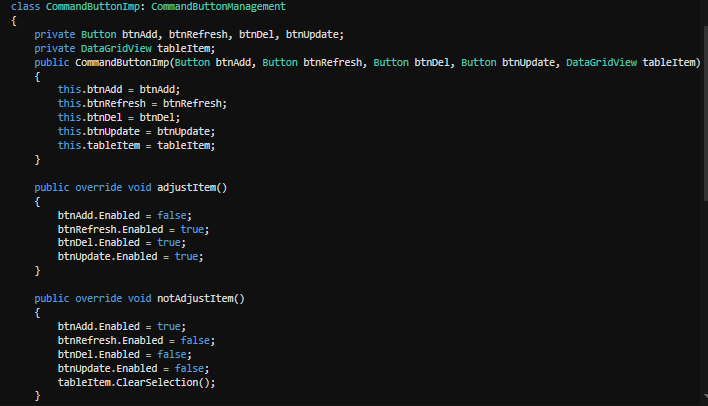
Command Pattern là mẫu thiết kế giúp đóng gói các thông tin cần thiết vào một đối tượng để thực hiện một hành động hay sự kiện nào đó. Mẫu thiết kế này giúp thực hiện các hoạt động ẩn hoặc hiện các nút thêm, xóa, sửa, refresh, bảng dữ liệu ở mỗi màn hình quản lý. Vì vậy, nếu sau này muốn thay đổi code thực hiện các hoạt động ẩn hiện các nút cũng không ảnh hưởng đến lớp sử dụng nó, và đồng thời vì có nhiều màn hình cùng sử dụng chức năng này nên dùng Command Pattern sẽ đảm bảo việc hạn chế chỉnh sửa nhiều nơi.

1. Cách áp dụng

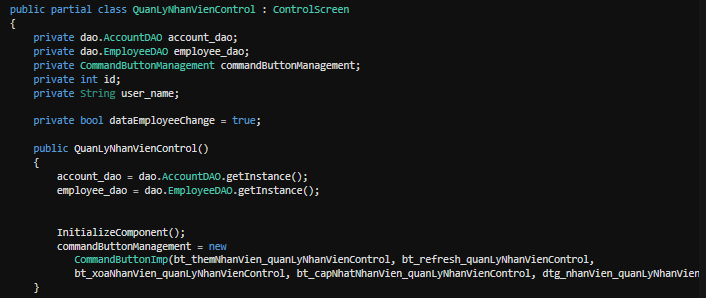
Ta có một lớp abstract CommandButtonManagement, lớp này sẽ định nghĩa các hoạt động sẽ xảy ra ở lớp con và ở đây ta có hai hoạt động là adjustItem và notAdjustItem. Đồng thời có 1 lớp con là CommandButtonImp, lớp này sẽ thừa kế lại lớp CommandButtonManagement để định nghĩa lại các hành vi ở hai method trên và lớp này sẽ bao gồm các property có thuộc tính Button và 1 property có thuộc tính là DataGridView. Ý nghĩa của các hành động đó như sau: ở các lớp QuanLyKhachHangVaSuKienControl, QuanLyNhanVienControl, QuanLyNhapSanPhamControl và QuanLySanPhamControl thì khi ta không chọn một item nào trên 1 DataGrideView, ta sẽ gọi lệnh notAdjustItem ở lớp CommandButtonImp dựa trên lớp CommandButtonManagement để thực thi việc tắt các nút refresh, delete, update, bật nút add và xóa sự lựa chọn trên bảng DataGridView ở các lớp đó. Ngược lại nếu có 1 item nào được chọn để edit, xóa thì ta sẽ thực hiện lệnh adjustItem để bật các nút refresh, delete, update, tắt nút add ở các lớp trên.



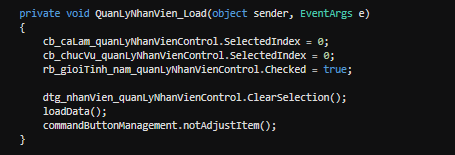
Hình 2.14: Lớp CommandButtonManagement



Hình 2.15: Lớp CommandButtonImp

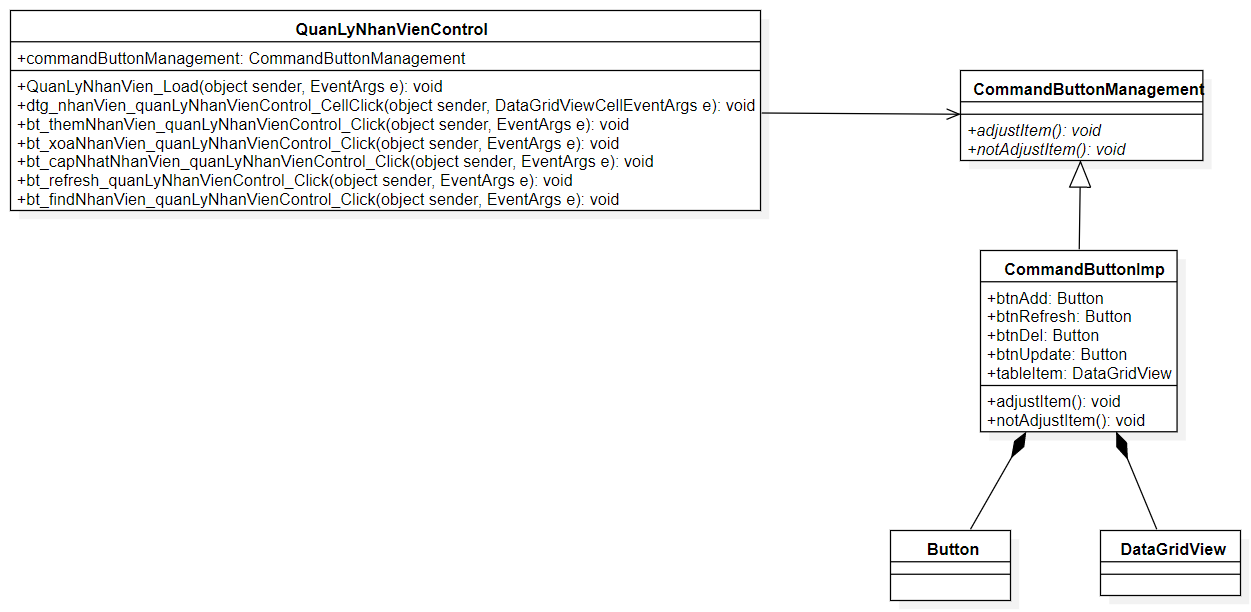


Hình 2.16: Sử dụng Command Pattern ở lớp QuanLyNhanVienControl



Ví dụ minh họa sài Command Pattern ở lớp QuanLyNhanVienControl

1. Sơ đồ lớp của Command Pattern



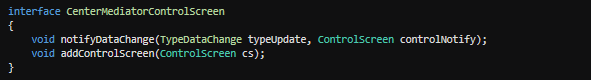
Hình 2.17: Sơ đồ lớp của Command Pattern

1. Mediator Pattern
2. Lý do sử dụng

Mediator Pattern là mẫu thiết kế giúp các class tương tác, giao tiếp với nhau, giúp cho khi có một màn hình thay đổi data thì có thể thông báo đến những màn hình khác rằng data đã được thay đổi, từ đó các màn hình sẽ nhận biết được có nên load lại data không tránh load lại data ở tất cả màn hình dù không có thay đổi.

1. Cách áp dụng

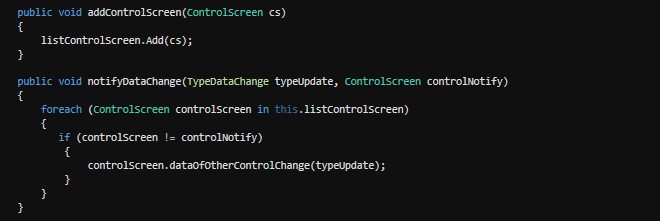
Ta có một lớp interface CenterMediatorControlScreen định nghĩa các hoạt động sẽ có trong lớp con của nó. Cụ thể ở đây gồm 2 hoạt động là addControlScreen với mục đích là thêm các lớp của ControlScreen vào một mảng dùng để thông báo đến các ControlScreen đó khi cần và hoạt động thứ hai là notifyDataChange với mục đích là khi có một ControlScreen có data thay đổi thì có thể dùng hàm này để thông báo đến các ControlScreen khác thông qua biến typeUpdate. Ví dụ nếu data product thay đổi thì typeUpdate sẽ là PRODUCT và nếu ControlScreen nào liên quan đến typeUpdate đó thì khi chuyển qua ControlScreen đó sẽ load lại data, còn lại thì không cần load. Ở màn hình QuanLySanPhamControl nếu có thay đổi về product thì sẽ gọi đến hàm notifyOtherControlDataChange ở lớp cha ControlScreen, lớp này sẽ gọi đến class CenterMediatorImpl để thông báo đến các ControlScreen khác là data product có sự thay đổi, trong đó có ControlScreen BanHangControl nên khi quay lại màn hình BanHangControl thì nó sẽ tự load lại data product.



Hình 2.18: Lớp CenterMediatorControlScreen



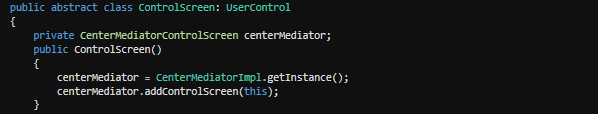
Hình 2.19: Lớp CenterMediatorImpl



Hình 2.20: Các hàm xử lý trong lớp CenterMediatorImpl



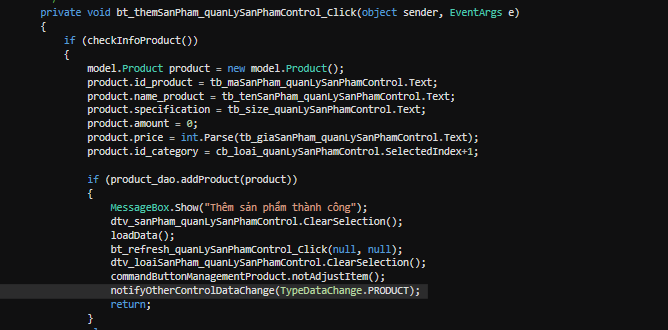
Hình 2.21: Kiểu dữ liệu TypeDataChange



Hình 2.22: Nơi sử dụng Mediator Pattern

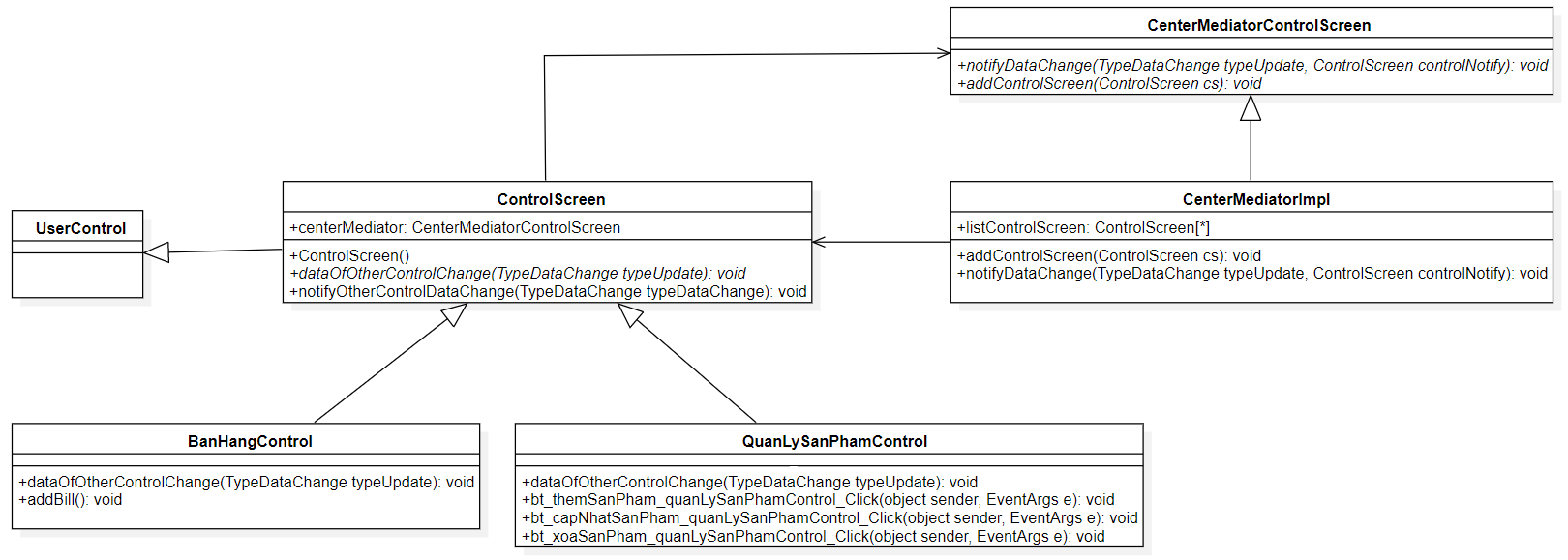


Hình 2.23: Hàm notifyOtherControlDataChange



Hình 2.24: Thông báo khi type product có sự thay đổi

1. Sơ đồ lớp của Mediator Pattern



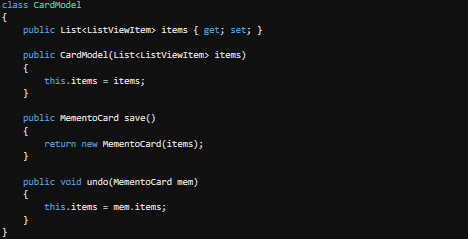
Hình 2.25: Sơ đồ lớp của Mediator Pattern

1. Memento Pattern
2. Lý do sử dụng

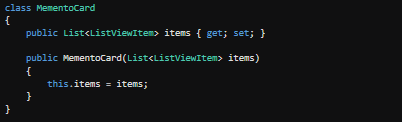
Memento Pattern là pattern giúp lưu lại trạng thái của một đối tượng để sau này có thể khôi phục lại trạng thái của đối tượng đó. Do đó pattern này được ứng dụng vào để lưu lại trạng thái giỏ hàng sau mỗi hoạt động để có thể thực hiện chức năng hoàn tác giỏ hàng trên ứng dụng và pattern này giúp tối giản code hơn trong việc sao lưu trạng thái của một đối tượng trong ứng dụng.

1. Cách áp dụng

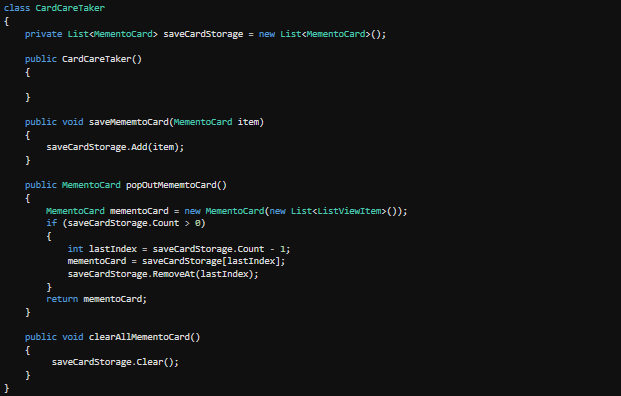
Ta có 3 lớp chính: với CardModel là model được dùng để lưu các item (sản phẩm) có trong giỏ hàng, lớp MementoCard là lớp được dùng để thể hiện trạng thái của CardModel và cuối cùng là lớp CardCareTaker được dùng để lưu trữ các trạng thái của CardModel và cũng là lớp giúp khôi phục lại trạng thái (MementoCard ) của CardModel. Cách hoạt động của Memento Pattern được ứng dụng như sau: khi trong lớp BanHangControl có sự thay đổi trạng thái của giỏ hàng (CardModel), ta dùng lớp CardCareTaker lưu lại trạng thái cũ thông qua method saveMememtoCard rồi mới update giỏ hàng (CardModel). Trong BanHangControl có một nút hoàn tác, khi ta bấm nút này, BanHangControl sẽ gọi đến method popOutMememtoCard để trả về trạng thái gần nhất và ta gắn trạng thái (MementoCard) cho giỏ hàng hiện tại. Nếu thanh toán xong hoặc bấm nút refresh thì ta sẽ xóa mọi trạng thái của giỏ hàng thông qua hàm clearAllMementoCard trong lớp CardCareTaker.



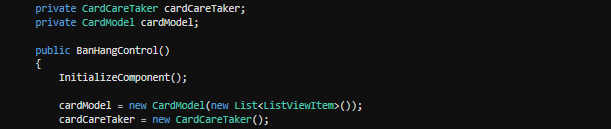
Hình 2.26: Lớp CardModel



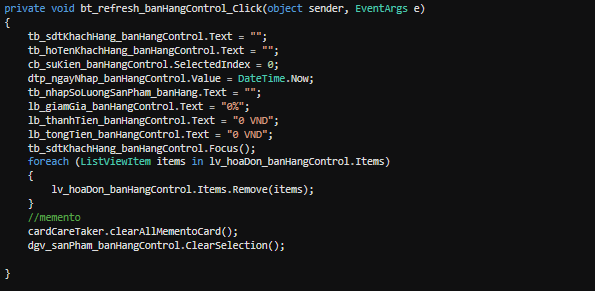
Hình 2.27: Lớp MementoCard



Hình 2.28: Lớp CardCareTaker



Hình 2.29: Nơi sử dụng Memento Pattern

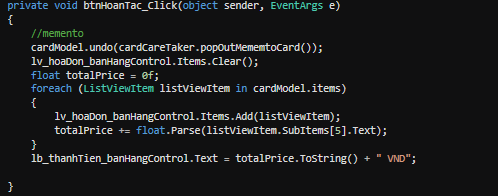


Hình 2.30: Xóa mọi trạng thái giỏ hàng khi bấm nút refresh

Dùng CardCareTaker để save lại trạng thái cũ của CardModel và update trạng thái mới cho CardModel.

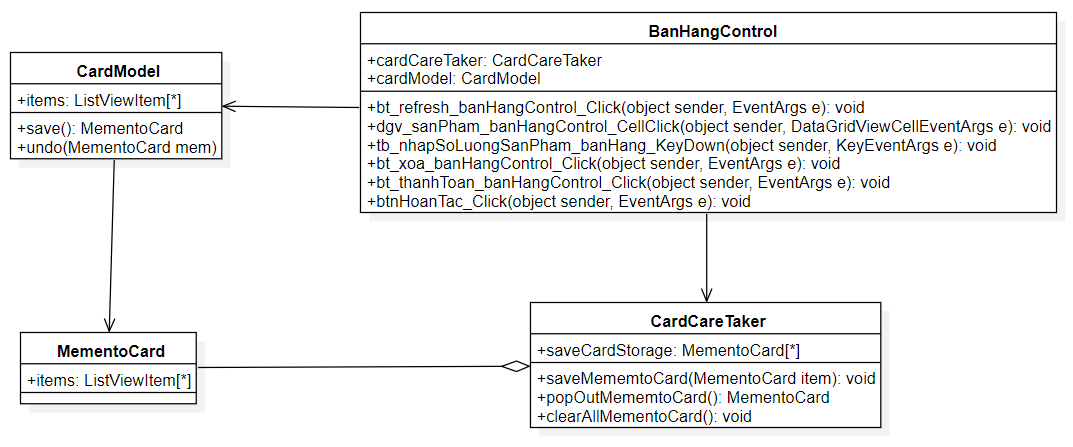


Hình 2.31: Nơi dùng CardCareTaker



Hình 2.32: Hoàn tác lại trạng thái cho CardModel khi bấm nút hoàn tác

1. Sơ đồ lớp của Memento Pattern



Hình 2.33: Sơ đồ lớp của Memento Pattern

CHƯƠNG 3 – TỔNG KẾT

Trên thực tế, ngành công nghiệp phần mền ứng dụng design pattern rất nhiều. Nó là phương pháp vô cùng tiện ích cho giải pháp của một vấn đề và được áp dụng ngày càng rộng rãi, phổ biến, hỗ trợ hầu hết các bài toán mà lập trình viên thường gặp phải. Những mẫu thiết kế này đã được chứng minh và đúc kết từ kinh nghiệm từ nhiều nhà nghiên cứu, do đó người dùng có thể dễ dàng và vận dụng và kết hợp chúng cho dự án của mình mà không phải mất thời gian thiết kế lại.

Tuy nhiên, số lượng trong hệ thống các mẫu thiết kế là khá nhiều nhưng chỉ một vài trong số đó thông dụng với người dùng. Vì vậy, ta chỉ cần nắm bắt ý tưởng của chúng và nghiên cứu sâu hơn khi cần thiết.

Ngoài các mẫu thiết kế chính thức được giới thiệu, theo thời gian, một số mẫu thiết kế mới khác cũng được cho ra đời, tạo ra sự nhầm lần và không rõ ràng cho người mới bắt đầu, mặt khác một số vẫn chưa hoàn toàn được công nhận do vậy, thông thường, ta chỉ nên tập trung vào 23 mẫu thiết kế đã được chấp nhận rộng rãi.