ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

GVHD: TS. Nguyễn Thị Thu Trang

và TS. Bùi Thị Mai Anh

Nhóm 05

Danh sách thành viên:

Nguyễn Trường Giang	20173083
Đỗ Quang Hiếu	20173108
Nguyễn Văn Trung Hiếu	20173107
Lê Đức Hải	20173094

Hà Nội, tháng 6, 2021

Mục lục

1 Tổng quan

1.1 Mục tiêu

Báo cáo này được viết với mục đích nêu lên các vấn đề trong code base mà vi phạm các nguyên lý thiết kế khi xây dựng một phần mềm, sau đó đưa ra các giải pháp để cải thiện, tối ưu mã nguồn.

Báo cáo này hướng đến các đối tượng là những người phát triển phần mềm, có quan tâm đến các vấn đề, nguyên lý thiết kế trong quá trình xây dựng dựng phát triển phần mềm và việc bảo trì, phát triển phầm mềm trong tương lai.

Báo cáo sẽ gồm 4 phần chính:

1. Tổng quan: Giới thiệu và đưa ra các mục tiêu, phạm vi báo cáo hướng đến

2. Đánh giá thiết kế cũ: Nêu ra các vấn đề code base cũ còn tồn động như các vi phạm về coupling, cohesion, SOLID, clean code,...

3. Đề xuất cải tiến: Sử dụng các kiến thức đã học, đặc biệt là các nguyên lý trong design pattern để giải quyết và cải tiến các vấn đề đã nêu ra trong mục 2

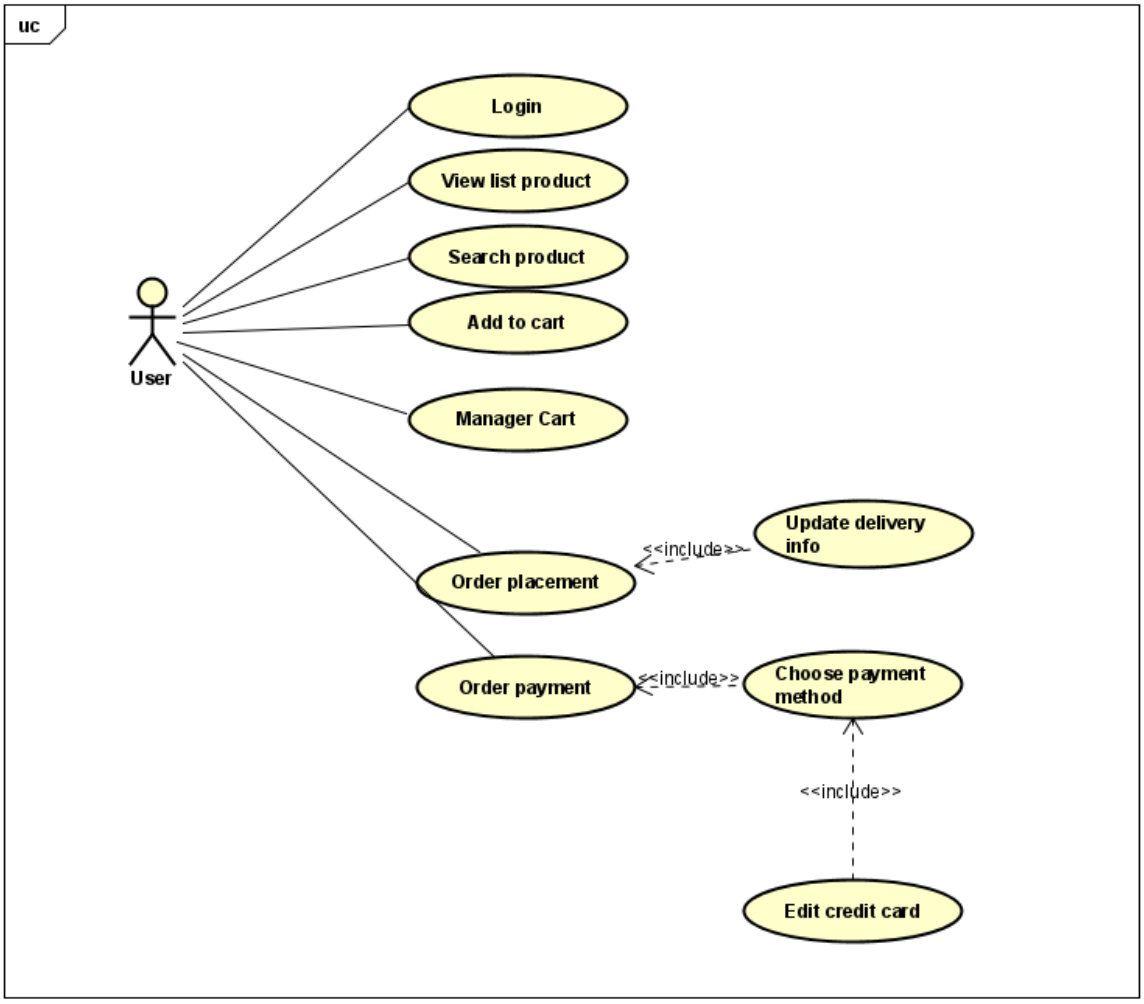
4. Tổng kết: Đúc kết lại hiệu năng của thiết kế sau khi tái cấu trúc. Mô tả các vấn đề còn sót lại hoặc chưa kịp thực hiện

1.2 Phạm vi

1.2.1 Mô tả khái quát phần mềm

AIMS (An Internet Media Store) là một hệ thống thương mại điện tử chuyên về mua bán sản phẩm phương tiện truyền thông (Sách, đĩa CD, đĩa DVD,...). Phầm mềm cung cấp các tính năng chính như xác thực tài khoản, quản trị hệ thống sản phẩm, người dùng, các thao tác mua hàng như xem sản phẩm, quản lý giỏ hàng, đặt hàng và thanh toán.

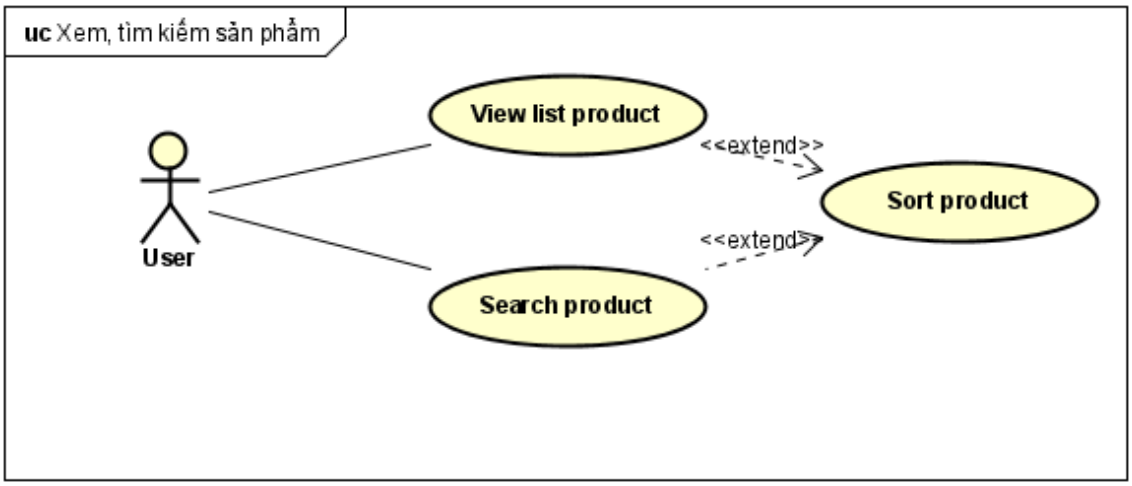
1.2.2 Các chức năng chính của phần mềm



Biểu đồ use case tổng quan

Một vài Use case phân rã chính:

- **Use case Xem, tìm kiêm sản phẩm**

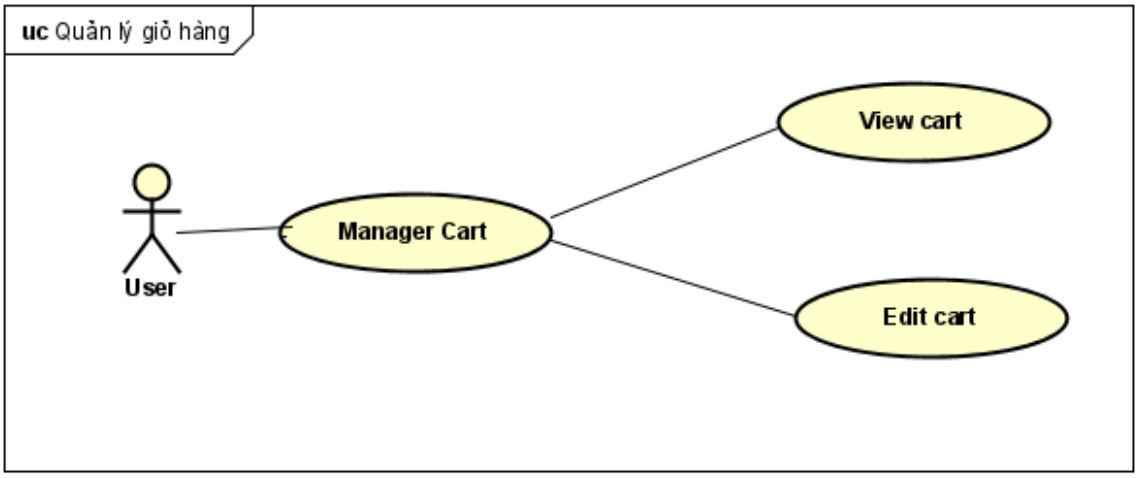


Use case Xem, tìm kiêm sản phẩm

Đặc tả use case

Tác nhân	User
Mô tả	Cho phép người dùng xem, tìm kiếm sản phẩm
Luồng sự kiện chính	- Người dùng truy cập vào hệ thống, hệ thống sẽ hiển thị danh sách các sản phẩm nổi bật hoặc các sản phẩm người dùng đã tìm kiếm - Người dùng có thể sắp xếp danh sách sản phẩm theo giá

- **Use case Quản lý giỏ hàng**

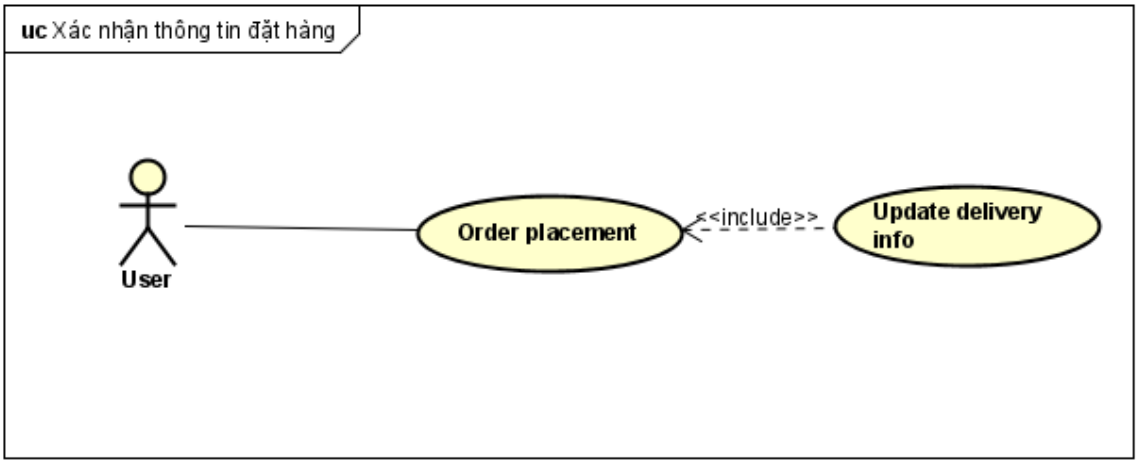


Use case Quản lý giỏ hàng

Đặc tả use case

Tác nhân	User
Mô tả	Cho phép người dùng thực hiện các thao tác với giỏ hàng
Tiền điều kiện	Người dùng đã đăng nhập
Luồng sự kiện chính	- Khi xem giỏ hàng, hệ thống sẽ hiển thị các thông tin chi tiết như tổng giá cả, danh sách sản phẩm... - Khách hàng có thể chỉnh sửa số lượng sản phẩm trong giỏ và bỏ sản phẩm khỏi giỏ

- **Use case Xác nhận thông tin giao hàng**

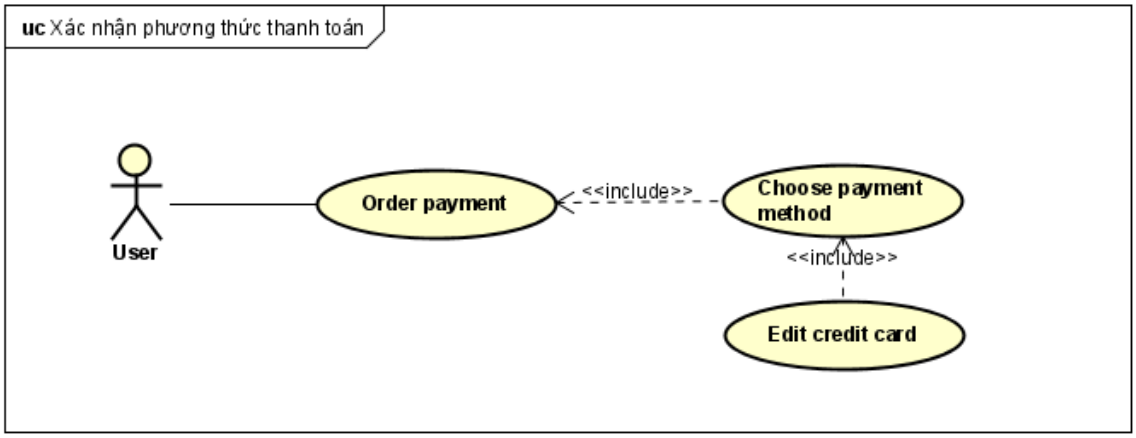


Use case Xác nhận thông tin giao hàng

Đặc tả use case:

Tác nhân	User
Mô tả	Người dùng xác nhận các thông tin cần thiết để đặt hàng
Tiền điều kiện	Người dùng đã đăng nhập
Luồng sự kiện chính	- Khách hàng cập nhật các thông tin giao hàng. - Sau khi hệ thống xác nhận các thông tin sẽ hiển thị các thông tin đơn hàng và phí giao hàng
Luồng sự kiện rẽ nhánh	- Hệ thống đưa ra thông báo thông tin giao hàng bị thiếu hoặc không hợp lệ - Khách hàng cập nhật lại các thông tin cho đến khi hợp lệ

- **Use case Xác nhận phương thức thanh toán**



Use case xác nhận phương thức thanh toán

Đặc tả use case:

Tác nhân	User
Mô tả	Người dùng xác nhận phương thức thanh toán cho đơn hàng
Tiền điều kiện	Người dùng đã đăng nhập, xác nhận các thông tin đặt hàng hệ thống yêu cầu
Luồng sự kiện chính	- Khách hàng chọn phương thức thanh toán - Nếu khách hàng thanh toán bằng thẻ tín dụng thì cần cung cấp các thông tin giao dịch gồm thông tin thẻ và nội dung giao dịch
Luồng sự kiện rẽ nhánh	- Khách hàng nhập sai thông tin thẻ, hệ thống sẽ yêu cầu nhập lại - Khách hàng cần nhập lại cho đến khi hợp lệ hoặc lựa chọn phương thức thanh toán khác

1.2.3 Cấu trúc mã nguồn

Mã nguồn được xây dựng theo mô hình MVC, các package chính và chức năng được mô tả dưới đây:

- common: chứa các exception trong quá trình thực thi code
- controller: xử lý các vấn đề về logic trong mã nguồn , trả dữ liệu cho phần view

- dao: trực tiếp tương tác, đọc ghi dữ liệu trong cơ sở dữ liệu và trả về cho controller
- entity: chứa các đối tượng trong mã nguồn
- subsystem: chứa code xây dựng phần giả lập hệ thống thanh toán
- utils: cung cấp một vài chức năng hữu ích hỗ trợ cho toàn bộ mã nguồn
- views: xây dựng các màn hình giao diện của phần mềm

1.2.4 Các yêu cầu thêm cần cân nhắc cùng quá trình tái cấu trúc

Trong quá trình tái cấu trúc cần đảm bảo mã nguồn sau khi hoàn thành không bị vi phạm các nguyên lý SOLID hay coupling, cohesion khi trong tương lai hệ thống bổ sung các tính năng mới như :

- Thêm loại mặt hàng mới
- Thêm màn hình Xem chi tiết sản phẩm
- Thay đổi yêu cầu khi load giao diện
- Thay đổi cách tính khoảng cách, sử dụng thư viện mới
- Thêm phương thức thanh toán mới: Thẻ nội địa

1.2.5 Các hoạt động thực thi trên mã nguồn để đạt được mục tiêu trên

- Chỉnh sửa mã nguồn để đảm bảo hạn chế vi phạm các nguyên lý SOLID, đạt được low coupling và high cohesion
- Xây dựng mã nguồn Clear name, Clear method/function, Clear Class
- Áp dụng các nguyên lý design pattern để khắc phục các vấn đề và cải tiến mã nguồn

1.2.6 Kết quả dự kiến

Sau khi tái cấu trúc mã nguồn sẽ đảm bảo đạt được các mức low coupling và high cohesion, hạn chế sự vi phạm các nguyên lý SOLID, clean code, áp dụng được các design pattern để giải quyết các vấn đề ban đầu và dễ dàng mở rộng, nâng cấp các tính năng có thể có trong tương lai.

1.3 Phân công nhiệm vụ

Khi phân chia nhiệm vụ, do các module nếu có dấu hiệu code smell, thì thường cũng vi phạm SOLID hoặc không đảm bảo high cohesion, low coupling. Do đó phân công trong nhóm dựa trên các cách clean code và các mẫu design pattern có thể tìm được và áp dụng:

Thành viên	MSSV	Nhiệm vụ
Lê Đức Hải	20173094	Các vấn đề liên quan đến State pattern và Adapter pattern
Đỗ Quang Hiếu	20173108	Các vấn đề liên quan đến Clean function/method, Template method và Factory method
Nguyễn Văn Trung Hiếu	20173107	Các vấn đề liên quan đến Cleanclass, cleanName và Observable pettern
Nguyễn Trường Giang	20173083	Các vấn đề liên quan đến Singleton pattern và Strategy pattern

Khi làm các vấn đề về clean code, tìm những module có thể áp dụng design pattern, các thành viên nhóm cũng đồng thời tìm các vấn đề vi phạm SOLID hay chưa đảm bảo high cohesion, low coupling.

1.4 Danh sách tham khảo

1. Centers for Medicare & Medicaid Services. (n.d.). System Design Document Template. Retrieved from Centers for Medicare & Medicaid Services: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SystemDesignDocument.docx

2. Cornell University How We Refactor and How We Document it? On the Use of Supervised Machine Learning Algorithms to Classify Refactoring Documentation

Retrived from www.elsevier.com/locate/eswa

3. Slide bài giảng môn Mẫu thiết kế phần mềm

2 Đánh giá thiết kế cũ

2.1 Nhận xét chung

Nhìn chung, các tổ chức class và source code của case study đã đảm bảo được hầu hết high cohesion và low coupling. Tuy nhiên vẫn còn tồn đọng một vài class / package chưa đáp ứng được hai điều kiện này, nhất là các vấn đề về clear name, clean function, gây ra những vi phạm các tính chất của OOP, các nguyên tắc SOLID dẫn tới khó khăn trong việc mở rộng, bảo trì, và tính dễ đọc hiểu của mã nguồn.

2.2 Đánh giá các mức độ coupling và cohesion

2.2.1 Coupling

#	*Các mức độ về Coupling*	*Module*	*Mô tả*	*Lý do*
1	common coupling	controller.AuthenticationController	Lớp AuthenticationController truy cập và thay đổi tham số của lớp khác trực tiếp	Lớp AuthenticationController truy cập và thay đổi tham số của lớp khác trực tiếp. Phương thức getMainUser() , login(), logOut() sử dụng tham số mainUser của lớp SessionInformation, và sửa đổi tham số này.
2	common coupling	controller.BaseController	Trong phương thức này đã sử dụng biến toàn cục cartInstance	Trong phương thức này đã sử dụng biến toàn cục cartInstance

3	common coupling	controller.PaceOrderController	Phương thức placeOrder và createOrder	Trong các phương thức này đã sử dụng biến toàn cục cartInstance
4	common coupling	controller.ViewCartController	Phương thức checkAvailabilityOfProduct và getCartSubtotal	Hai phương thức này đều sử dụng biến toàn cục cartInstance
5	Control Coupling	subsystem.interbank.InterbankPayloadConverter	Phương thức extractPaymentTransaction	Trong phương thức này có truyền tham số responseText, từ tham số này có thể lấy được error code dùng để điều khiển và lựa chọn các exception
6	Stamp coupling	entity.shipping.DeliveryInfo	Hàm calculateShippingFee dùng không hết các tham số truyền vào	Tham số truyền vào của hàm là 1 Object nhưng trong Hàm không sử dụng đến các thuộc tính của Object đó
7	Stamp coupling	controller.BaseController	Phương thức checkMediaInCart	Tham số truyền vào của hàm là một object Media
8	Stamp coupling	controller.PaceOrderController	-Phương thức createInvoice -phương thức processDeliveryInfo và validateDeliveryInfo	-Sử dụng tham số Order là một object -Tham số truyền vào hai phương thức này là kiểu cấu trúc HashMap

2.2.2 Cohesion

#	*Các mức độ về Cohesion*	*Module*	*Mô tả*	*Lý do*
1	Coincidental	subsystem.interbank.InterbankPayloadConverter	Phương thức getToday không liên quan đến class này	Class này có nhiệm vụ convert các thông tin về thanh toán, thẻ thanh toán, còn phương thức getToday trả về thông tin về ngày giờ hiện tại
2	Coincidental	controller.PaceOrderController	Các phương thức về validate không không có nhiệm vụ ở lớp này	Lớp này tạo các thông tin về mua hàng, hoá đơn, các vấn đề xác thực thông tin chuyển phát nên đề ở class riêng
3	Temporal	App	Đoạn code ở hàm start nhằm hiển thị màn hình splash, sau đó vào màn hình main.	Những đoạn code fadeIn, fadeout được sử dụng tuần tự nhau.
4	Communicational	Entity.Order	Hàm setDeliveryInfo	Hàm setDeliveryInfo gọi đến hàm calculateShippingFee là không cần thiết do 2 hàm khác về chức năng mà chỉ dùng chung 1 dữ liệu

2.3 Đánh giá việc tuân theo SOLID

Đối với thiết kế hiện tại, việc đảm bảo các quy tắc SOLID khá chặt chẽ, tuy nhiên còn một số module có dấu hiệu vi phạm. Dưới đây là những module vi phạm SOLID mà nhóm tìm thấy

2.3.1 SRP

#	*Module*	*Mô tả*	*Lý do*
1	PaymentController	Đảm nhận nhiều nhiệm vụ cùng lúc	Trong lớp này có phương thức getExpirationDate dùng để kiểm tra ngày tháng có phù hợp hay không. Chức năng này nằm ngoài phạm vi của lớp (chỉ thanh toán)
2	AuthenticateController	Vi phạm SRP trong tương lai	Trong lớp này có phương thức md5() dùng để mã password, nếu sau này đổi cách mã hóa password phải sửa code.
3	PlaceOrderController	Lớp này đang chứa các phương thức không nằm trong phạm vi	Các phương thức createOrder và createInvoice không nên để cùng một chỗ. Thực tế lớp này đang đảm nhận nhiệm vụ làm controller của 2 màn hình (CartScreenhandler, ShippingScreenHandler). Ngoài ra các phương thức validate cũng nên để ở chỗ khác.

2.3.2 OCP

#	*Module*	*Mô tả*	*Lý do*

1	view.screen	Thay đổi cách xử lý lỗi sẽ phải sửa code hiện tại	Tại contructor của các lớp trong module này, khi gọi các phương thức setupFunctionality, setupData đều có bắt sự kiện Exception, và xử lý exception theo cách cố định, việc thay đổi cách xử lý lỗi sẽ khó khăn. Đồng thời vi phạm ISP khi sử dụng trực tiếp module PopupScreen
2	PlaceOrderController	Hàm validateDeliveryInfo vi phạm OCP trong tương lai	Lớp PlaceOrderController đang phụ thuộc trực tiếp vào số info có kiểu Map. Sau này khi tham số info có đổi tên key, thì phải sửa lại ở lớp này.
3	Entity.payment	Mở rộng có thể vi phạm OCP	Lớp này chỉ chứa đối tượng Card, các lớp khác sử dụng đối tượng Card trực tiếp. Việc mở rộng sau này có thể vi phạm OCP

2.3.3 LSP

#	*Module*	*Mô tả*	*Lý do*
1	BaseController	Lớp này chứa các phương thức không tổng quát	Các lớp khác đang kế thừa lớp BaseController nhưng không sử dụng đến các method/attribute của BaseController: checkMediaInCart, getListCartMedia

2.3.4 ISP

#	*Module*	*Mô tả*	*Lý do*
1	entity.shipping.DeliveryInfo	Phụ thuộc trực tiếp vào đối tượng	đang phụ thuộc trực tiếp vào thư viện DistanceCalculator

2.3.5 DIP

#	*Module*	*Mô tả*	*Lý do*
1	BaseScreenHandler	Module cấp cao phụ thuộc vào module cấp thấp hơn	lớp này phụ thuộc (association) vào lớp HomeScreenHandler là một lớp kế thừa từ BaseScreenHandler

2.4 Các vấn đề về Clean Code

2.4.1 Clear Name

Mã nguồn codebase đáp ứng khá tốt Clear Name, tuy nhiên vẫn còn một vài biến trong một số class được đặt tên chưa thể hiện rõ chức năng của biến đó.

2.4.1.1 Use Intention revealing Names

Tên của một biến, hàm / phương thức, lớp phải nêu rõ mục đích

- Class PaymentController: biến strs không có ý nghĩa

```
private String getExpirationDate(String date) throws
InvalidCardException {
	String[] strs = date.split("/");
	if (strs.length != 2) {
		throw new InvalidCardException();
	}

	……….
}
```

- Trong class InterbankSubsystem: biến ctrl chưa thể hiện được mục đích

```
public class InterbankSubsystem implements InterbankInterface {

    /**
     * Represent the controller of the subsystem
     */
    private InterbankSubsystemController ctrl;

      ……….
}
```

2.4.1.2 Avoid Disinformation

Không đặt tên sai ý nghĩa, không đặt tên giống nhau cho các biến khác nhau

- Có hai class MediaHandler trong package views.screen.cart và views.screen.home

2.4.1.3 . Method names

Tên của phương thức nên là động từ hoặc cụm động từ

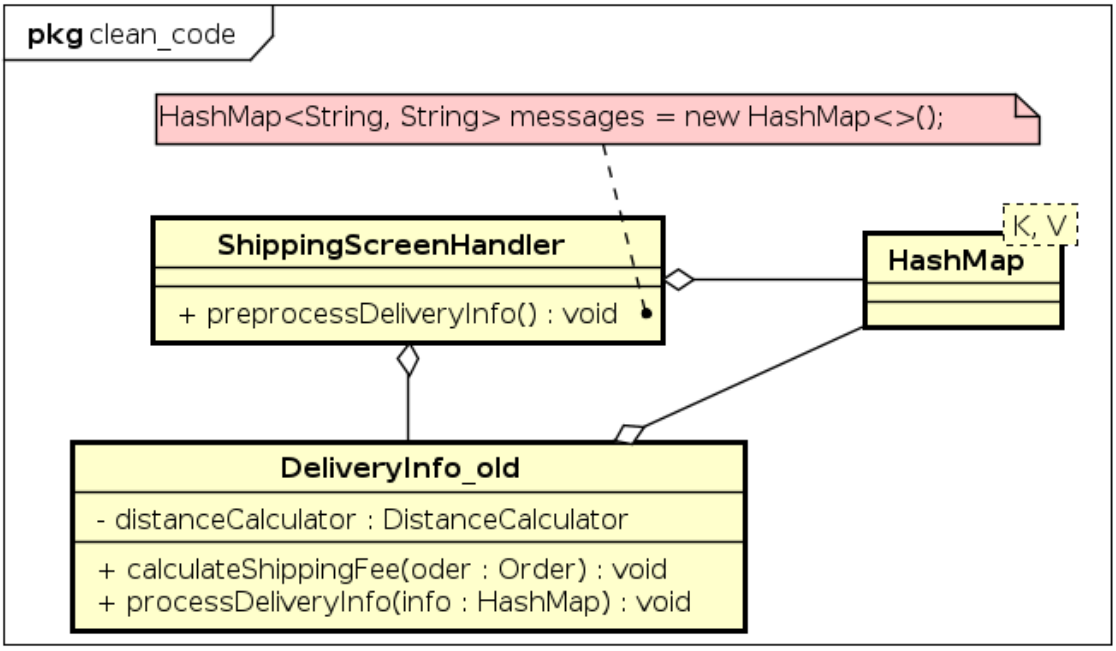
- Phương thức md5() trong lớp AuthenticationController không chỉ rõ mục đích

2.4.2 Clean Function/Method

Mã nguồn hiện tại còn nhiều chỗ chưa đáp ứng được clean funcion/method, cụ thể:

1. Phương thức checkAvailabilityOfProduct của class entity.cart.Cart sử dụng biến allAvailable để tạo ra lỗi, mặc dù có thể return ngay khi có thể trong vòng lặp for.
2. Trong lớp PlaceOrderController, có các method validatePhoneNumber, validateName, validateAddress không phải chức năng chính của lớp này. Nếu các lớp các cũng cần validate thì hoặc lớp đó cần chứa đối tượng PlaceOrderController (tăng coupling), hoặc tạo lại các phương thức validate mới (lặp code). Hoặc khi các phương thức validate cần thay đổi thì phải sửa cả lớp PlaceOrderController (vi phạm OCP)
3. Ở các lớp PaymentController phương thức payOrder, ResultScreenHandler phương thức setupData có sử dụng kiểu Map để trao đổi dữ liệu message; như vậy có thể dẫn tới việc khó hiểu do kiểu

Map có thể lưu trữ nhiều loại dữ liệu "thừa" không dùng tới trong trường hợp này. (code smell - Data level)

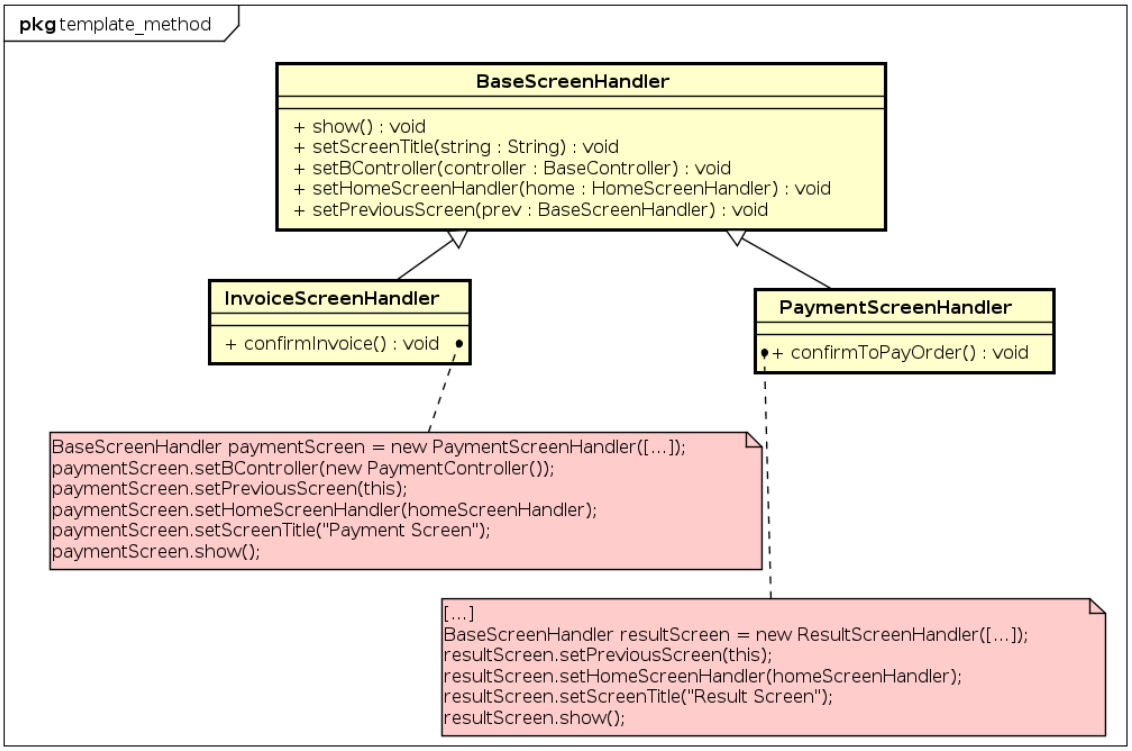


Hình 2.4.1: Code smell trong ShippingScreenhandler

4. Ở hàm preprocessDeliveryInfo trong lớp views.screen.shipping. ShippingScreenHandler sử dụng biến message kiểu Map để truyền dữ liệu. Việc này gây ra code-smell (data-level).
5. Tại lớp App, có những đoạn code dài, được comment lại thành các khối. Đây là dấu hiệu code smell (statement level).
6. Tại phương thức setMediaInfo của lớp views.screen.invoice. MediaInvoiceScreenHandler, phương thức setMediaInfo của lớp views.screen.home. MediaHandler, phương thức này đang làm nhiều nhiệm vụ cùng lúc là tạo đối tượng file để cài đặt hình ảnh hiện thị và các thông tin khác. Việc này dẫn tới code smell (statement level).
7. Tại phương thức addMenuItem lớp views.screen.home. HomeScreenHandler, requestToPlaceOrder lớp CartScreenHandler đang xử lý quá nhiều việc , các công việc được chia thành các khối lệnh và sử dụng comment để chia tách.
8. Ở các phương thức login (AuthenticationController), post (ApplicationProgrammingInterface), md5(AuthenticationController),

calculateShippingFee (DeliveryInfo), setupFunctionality (HomeScreenHandler) đang sử dụng những "magic number"

9. Tại những lớp kế thừa BaseScreeenHandler trong view.screen, nhiều màn hình trước khi gọi sang màn hình tiếp theo đều dùng các phương thức chung như setPreviousScree, setHomeScreenHandler, setBController, show trước khi chuyển sang màn hình mới.



Hình 2.4.2 : Ví dụ về lặp code ở views.screen

10. Tại những lớp kế thừa BaseScreeenHandler trong view.screen, khi khởi tạo đối tượng đều áp dùng một form chung

```
        super(stage, screenPath);
        try {
            setupData(null);
            setupFunctionality();
        } catch (IOException ex) {
            LOGGER.info(ex.getMessage());
            PopupScreen.error("Error when loading resources.");
        } catch (Exception ex) {
            LOGGER.info(ex.getMessage());
            PopupScreen.error(ex.getMessage());
        }
```

Các dòng code này có tác dụng tương tự nhau ở nhiều lớp khác nhau.

11. Phương thức checkAvailabilityOfProduct của lớp entity.cart.Cart đang sử dụng kiểu so sánh trực tiếp, dẫn tới khó mở rộng sau này.
12. Phương thức getExpirationDate trong lớp PaymentController sử dụng các kiểu if-else

```
if (month < 1 || month > 12 || year <
Calendar.getInstance().get(Calendar.YEAR) % 100 || year > 100)
{
                throw new InvalidCardException();
            }
```

Sử dụng như vậy dẫn tới code smell (datalevel, statement-level) do sử dụng các magic number, nhiều điều kiện so sánh trong một câu so sánh

2.4.3 Clean Class

Trong codebase vẫn tồn tại một số class chưa tuân thủ Clean Class, ta có thể sử dụng các DesignPattern để xử lí

2.4.3.1 Large Class

- Class AuthenticationController vừa làm nhiệm vụ authentication vừa làm nhiệm vụ mã hoá md5

- Class PlaceOrderController: có thêm các phương thức validate thông tin chuyển phát

- Class InterbankPayloadConverter: có phương thức getToday không liên quan đến nhiệm vụ của class này.

2.4.3.2 Divergent change

- PaymentController, CreditCart: Khi thay đổi phương thức thanh toán cần thay đổi các class này

2.4.3.3 Shortgun surgery

- Các lớp Media, MediaHandler, MediaInvoiceScreenHandler: khi thay đổi các thuộc tính trong lớp Media thì cũng phải thay đổi thuộc tính trong hai lớp còn lại

2.5 Các vấn đề khác

#	*Module*	*Mô tả*	*Lý do*

1	Entity.Order	Áp dụng State DP vào lớp Order để lưu trữ trạng thái đơn hàng	Trong tương lai Các Đơn hàng sẽ có nhiều trạng thái (VD như: Created, Submitted, Accepted) vậy nên để cải tiến cho thiết kế ban đầu ta có thể áp dụng State DP vào lớp Order để lưu trữ trạng thái đơn hàng

3 Đề xuất cải tiến

3.1 Vấn đề Clean code và giải pháp

3.1.1 Giải quyết Clear Name

Đặt lại tên cho các biến sao cho phù hợp với mục đích, cách sử dụng của biến đó:

```
private String getExpirationDate(String date) throws
InvalidCardException {
      String[] strs = date.split("/");
      if (strs.length != 2) {
            throw new InvalidCardException();
      }

      ……….
}
```

➔ String[] dateSplitArray = date.split(“”/”);

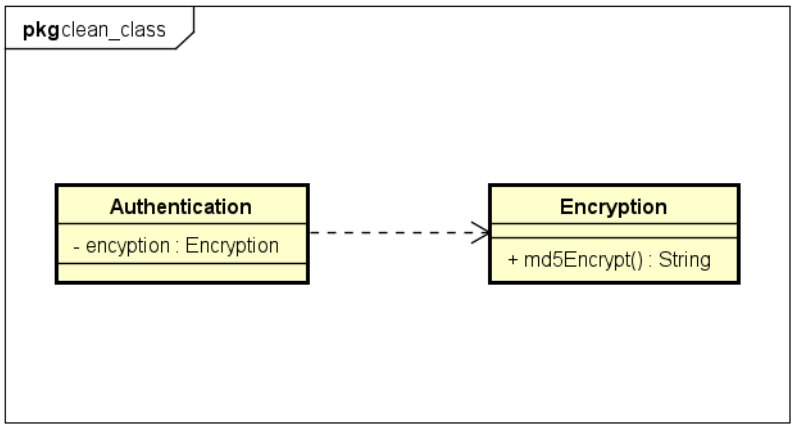
```
public class InterbankSubsystem implements InterbankInterface {

    /**
     * Represent the controller of the subsystem
     */
    private InterbankSubsystemController ctrl;

      ……….
}
```

➔ private InterbankSubsystemController interbankSubsystemController;

3.1.2 Giải quyết Clean Class

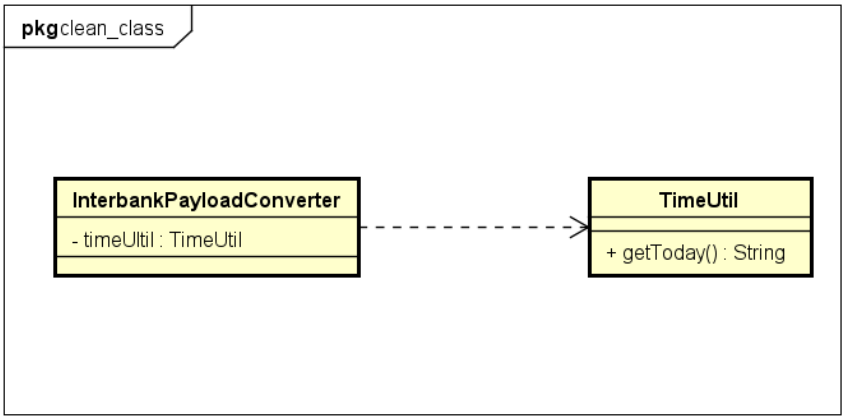
Đối với vấn đề Clean Class, cách giải quyết chủ yếu là phân tách lớp thành các lớp nhỏ hơn hoặc tách các phương thức thành nhiều phương thức con, mục đích của phương pháp này là giảm sự trùng lặp code, giúp cho việc bảo trì hoặc nâng cấp, sửa đổi code trở nên đơn giản, dễ dàng hơn

- Lớp AuthenticationController tách phương thức md5() ra thành một lớp riêng



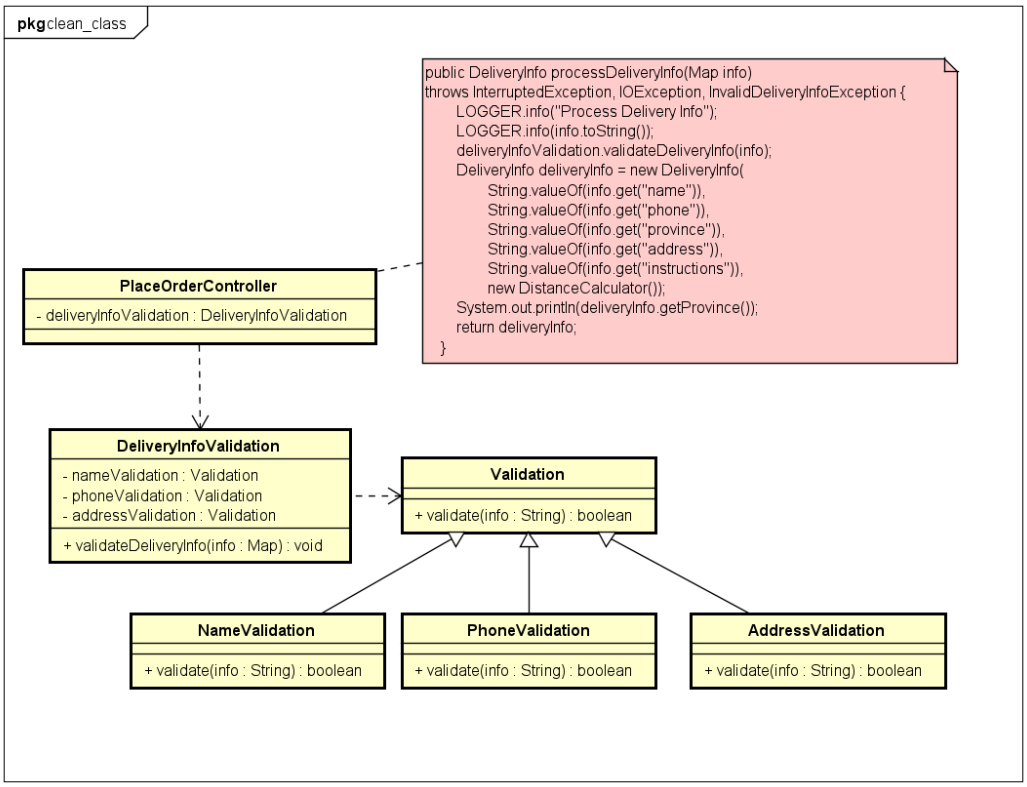
Hình 3.1.1: Clean class trong AuthenticationController

- Lớp InterbankPayloadConverter tách phương thức getToday() ra một lớp riêng



Hình 3.1.2: Clean class trong InterbankPayloadConverter

- Lớp PlaceOrderController nên tách phần validate ra thành các lớp riêng để tránh vi phạm nguyên lí SRP



Hình 3.1.3: Clean class trong PlaceOrderController

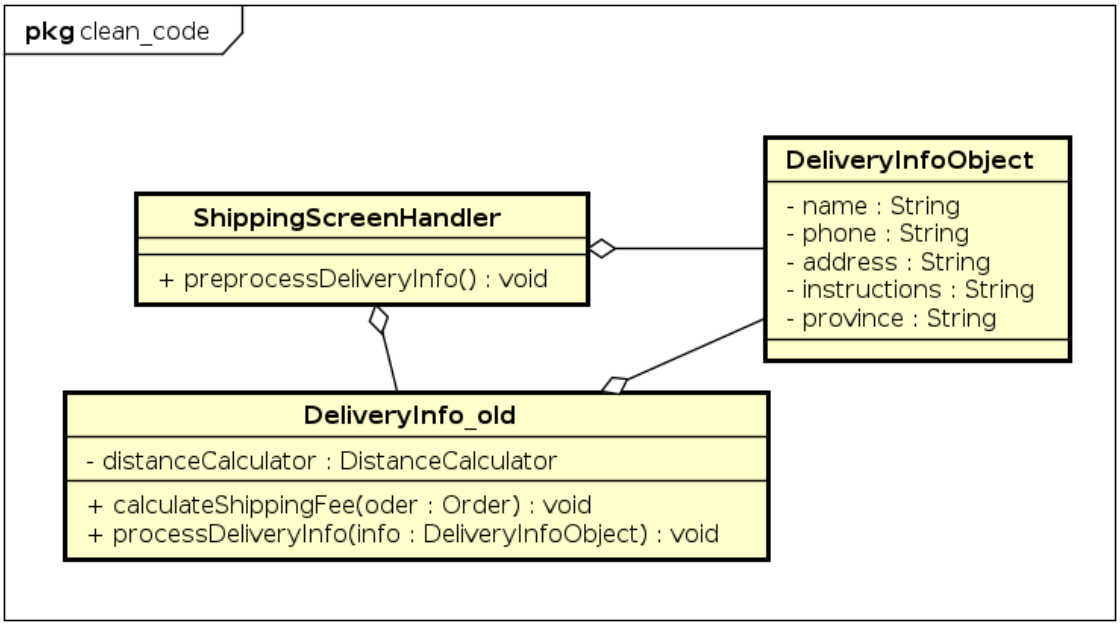
3.1.3 Giải quyết Clean Function/Method

- Vấn đề 2.4.2.1:
  Lớp entity.cart.Cart:
  Phương thức checkAvailabilityOfProduct: return ngay khi có thể thay vì return sau khi duyệt vòng lặp:

```
   public void checkAvailabilityOfProduct() throws SQLException {
        boolean allAvailable = true;
        for (CartItem cartItem : lstCartItem) {
            if (!cartItem.availableQuantity())throw new
MediaNotAvailableException("Some media not available");
        }
}
```

  Việc return ngay khi có thể cũng được dùng tại phương thức md5 trong AuthenticationController, phương thức payOrder trong PaymentController
- Vấn đề 2.4.2.3 và 2.4.2.4: Sử dụng kiểu Map trong PaymentController:

Giải pháp: Tạo ra một đối tượng DeleveriInfoObj để giao tiếp giữa các lớp thay vì sử dụng kiểu Map



Hinh 3.1.4: Clean function trong ShippingScreenHandler

- Vấn đề 2.4.2.5, 2.4.2.6 và 2.4.2.7: Các vấn đề về lagre function:
  - Tại lớp App, hàm start đang bị large function, nên tách thành các đoạn code nhỏ để thực hiện:

```
public class App extends Application {
      void initIntroScreen(){…}
      void initFaceIn(){…}
      void initFadeOut(){…}
      void start(){
            initIntroScreen(primaryStage);
            initFaceIn();
            initFadeOut(primaryStage);
            fadeIn.play();
      }
      ...
}
```

  - Tại lớp view.screen.invoice.MediaInvoiceScreenHandler, phương thức setMediaInfo cũng đang bị large function
    Giải quyết: Tách thành các function nhỏ

```
public class MediaInvoiceScreenHandler extends
FXMLScreenHandler {
    setTitle() { … };
    setPrice(){ … };
```

```
    setNumOfProd() { … };
  void setCoverImage() { …  }
  public void setMediaInfo() throws SQLException {
    setTitle();
    setPrice();
    setNumOfProd();
    setCoverImage();
  }
}
```

- Large code ở phương thức addMenuItem trong lớp views.screen.home.HomeScreenHandler:

```
public class HomeScreenHandler{
    void getLabelMenuItem(String text, MenuButton
menuButton) { … };
    void emptyHomeMedia() { … };
    void filterHomeMediaItems() { … };
    void addMediaHome() { … };

    private void addMenuItem(int position, String
text, MenuButton menuButton) {
        MenuItem menuItem = new MenuItem();
        Label label = getLabelMenuItem(text,
menuButton);
        menuItem.setGraphic(label);
        menuItem.setOnAction(e -> {
            emptyHomeMedia();
            List filteredItems =
filterHomeMediaItems(text);
        (filteredItems);
        });
        menuButton.getItems().add(position, menuItem);
    }
}
```

Ngoài ra còn có large function ở các method requestToPlaceOrder lớp CartScreenHandler, phương thức query trong InterbanlBoundary, setMediaInfo trong MediaInvoiceScreenHandler, setupData trong home/MediaHandler

- Việc truyền thông báo giữa các màn hình:

  Các lớp PlaymentController (phương thức payOrder), PaymentScreenHandler (phương thức confirmToPayOrder), ResultScreenHandler (phương thức setupData) đang sử dụng kiểu HashMap để truyền dữ liệu.

Giải pháp: Tạo một đối tượng ResponseMessage để truyền dữ liệu

```
public class ResponseMessage {
  private String result;
  private String message;

  public ResponseMessage(String result, String message) {
    this.message = message;
    this.result = result;
  }
      get, set…
}
```

- Vấn đề 2.4.2.8: Các vấn đề về magic number: tại các phương thức
    - login (AuthenticationController),
    - post (ApplicationProgrammingInterface),
    - md5(AuthenticationController),
    - calculateShippingFee (DeliveryInfo),
    - setupFunctionality (HomeScreenHandler),
    - …

=> Giải pháp: sử dụng các static const

Do phần này có nhiều nơi vi phạm, và cách giải quyết thì giống nhau, nên trong báo cáo này chỉ ví dụ một nơi để giải quy

Ví dụ phương thức setupFunctionality (HomeScreenHandler

```
addMenuItem(BOOK_POSITION, "Book", splitMenuBtnSearch);
addMenuItem(DVD_POSITION, "DVD", splitMenuBtnSearch);
addMenuItem(CD_POSITION, "CD", splitMenuBtnSearch);
```

```
Public class HomeScreenHandler{
    final int BOOK_POSITION = 0;
    final int DVD_POSITION = 1;
    final int CD_POSITION = 2;
…
}
```

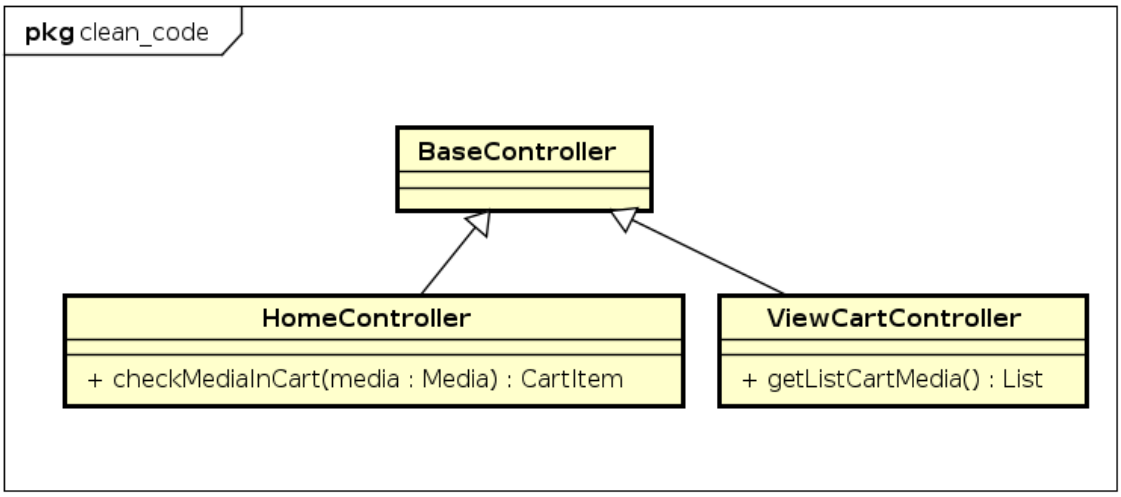
- Vấn đề về sử dụng nhiều điều kiện tại một dòng:
  Ở các phương thức getExpirationDate trong lớp PaymentController đang có code smell; giải quyết: Tách khối điều kiện ra thành phương thức riêng, tạo ra một đối tượng DateTimeUtils chứa phương thức validate

```
private String getExpirationDate(String date) throws
InvalidCardException {
    try {
      String[] strs =
DateTimeUtils.getMonthYearFromString(date);
      return DateTimeUtils.getExpiredDate(strs[0], strs[1]);
    } catch (Exception exc) {
      throw new InvalidCardException();
    }
  }
```

```
public class DateTimeUtils {
    static public String[] getMonthYearFromString(String
date) throws Exception { … }
    static public String getExpiredDate(String strMonth,
String strYear) throws Exception { … }
}
```

3.2 Vấn đề 2.3.3.1 vi phạm LSP trong BaseController

Do lớp Các phương thức kiểm tra cart trong BaseController chỉ dùng cho các màn hình sử dụng cart. Nên chuyển các method này xuống các lớp con.

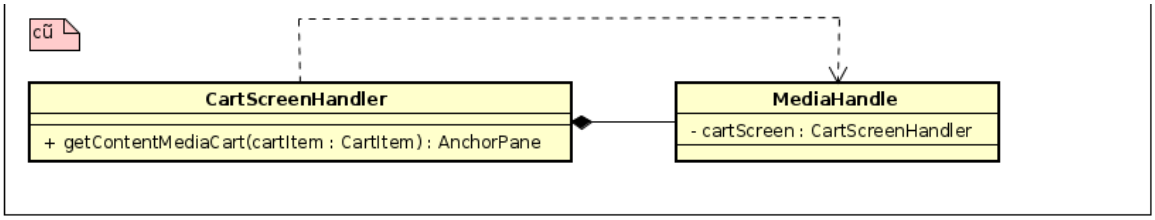


Hình 3.2.1: Giải quyết LSP trong BaseController

3.3 Vấn đề về Observable trong views.screen.Cart

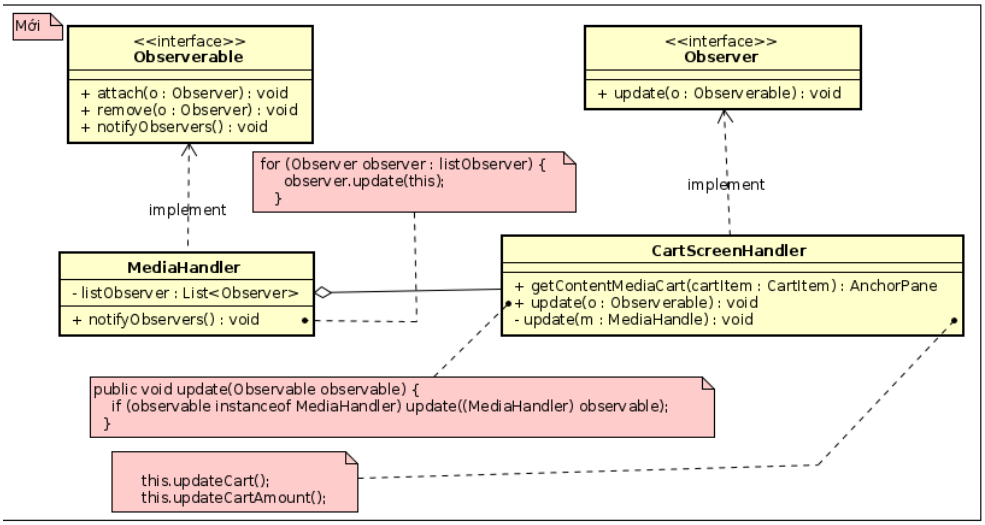
Trong lớp view.cart.CartScreenHandler có mỗi quan hệ Composition với lớp view.cart.MediaHandler. Lớp view.cart.MediaHandler lại chứa 1 parameter kiểu view.cart.CartScreenHandler. Do đó 2 lớp này đã bị high coupling.

Việc coupling này để cập nhật view.cart.CartScreenHandle khi lớp view.cart.MediaHandler thay đổi



Hình 3.3.1: CartScreenHandler và MediaHandler cũ

Giải pháp: Áp dụng Observable



Hình 3.3.2: Ứng dụng Observable trong CartScreenHandler và MediaHandler

```
views.screen.cart.MediaHandler
  public void attach(Observer observer) {
    if (observer != null)
      listObserver.add(observer);
  }
 public void notifyObservers() {
    for (Observer observer : listObserver) {
```

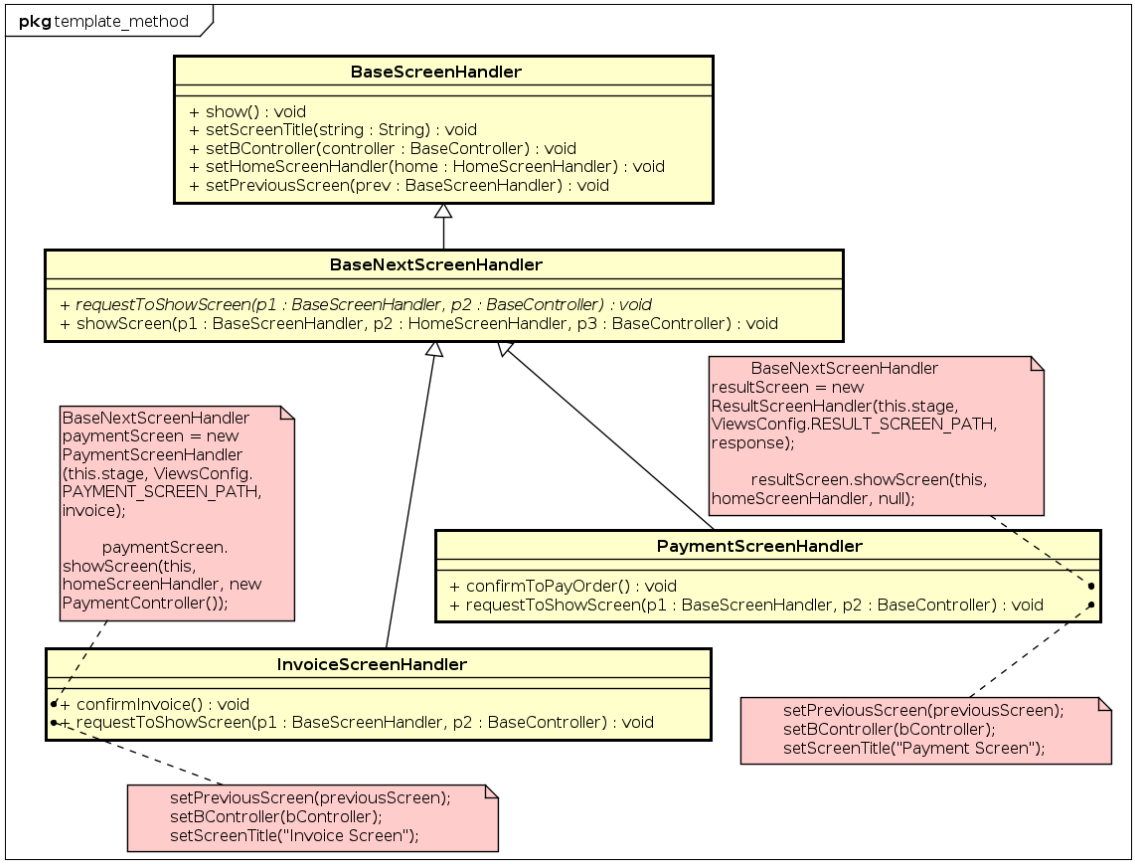
```
      observer.update(this);
    }
  }
```

```
views.screen.cart.CartScreenHandler

  AnchorPane getContentMediaCart(CartItem cartItem) throws IOException {
    MediaHandler mediaCartScreen = new
MediaHandler(ViewsConfig.CART_MEDIA_PATH);
    mediaCartScreen.setCartItem(cartItem);

    // design pattern: Observable
    mediaCartScreen.attach(this);

    return mediaCartScreen.getContent();
  }

 public void update(Observable observable) {
    if (observable instanceof MediaHandler) update((MediaHandler)
observable);
  }

  private void update(MediaHandler mediaHandler) {
    try {
      this.updateCart();
      this.updateCartAmount();
    } catch (SQLException exp) {
      exp.printStackTrace();
      throw new ViewCartException();
    }
  }
```

3.4 Vấn đề 2.4.2.9 về hiển thị trong module view.screen và giải pháp

Các phương thức khởi tạo trong các lớp ở module view.screen đều sử dụng những kiểu giống nhau. Chỉ có một số lớp có sự thay đổi nhỏ.

Giải pháp: Áp dụng template method:

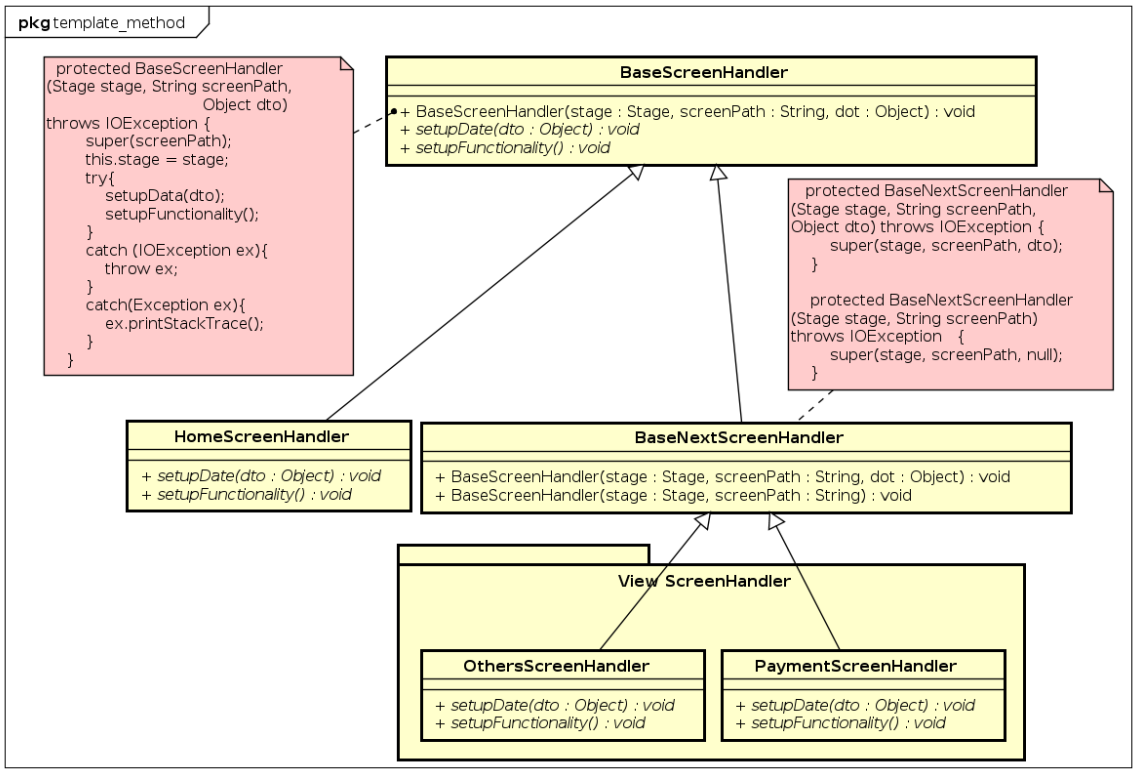


Hình 3.4.1: Giải pháp template method cho các lớp views.screen

Tạo một abstract BaseNextScreenHandler, dùng để quy định các lớp có thể chuyển tiếp màn hình. Trong class này có function showScreen quy định để hiển thị; và abstract function requestToShowScreen để các lớp kế thừa nó override lại.

3.5 Vấn đề 2.4.2.10 về phương thức khởi tạo trong module views.screen và cách giải quyết

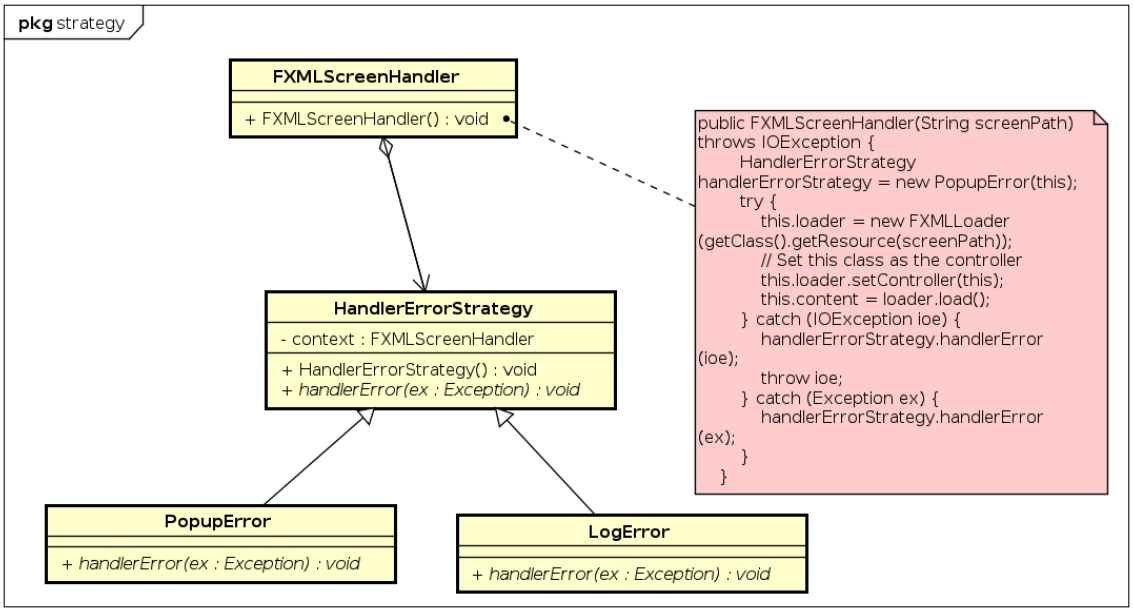
Các lớp con của BaseScreenHandler và BaseNextScreenHandler đều có cách khởi tạo tương tự nhau, nên tách về lớp cha



Hình 3.6.1: Áp dụng template method trong constructor của views.screen

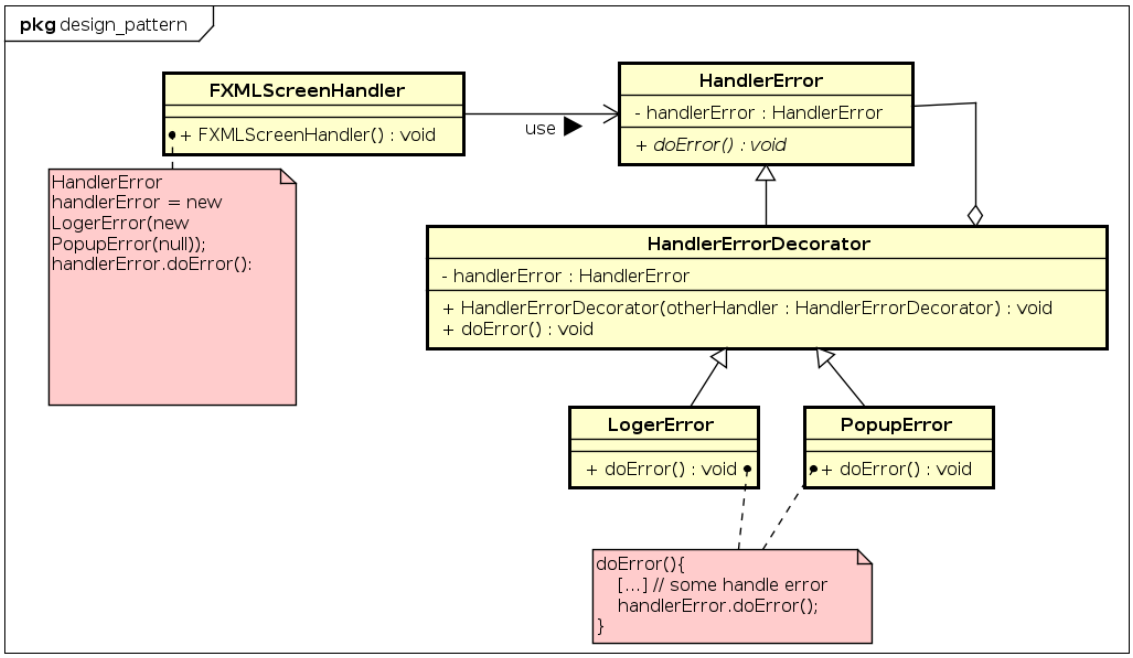
3.6 Vấn đề 2.3.2.1 về xử lý lỗi khi khởi tạo các screen trong views.screen

Khi áp dụng template method cho các constructor của views.screen, phần xử lý lỗi sẽ được đẩy lên lớp cao nhất (lớp FXMLScreenHandler). Tại lớp này, có nhiều cách để thông báo lỗi. Do đó có thể áp dụng Strategy hoặc Decorator.



Hình 3.6.1: Xử lý bằng strategy

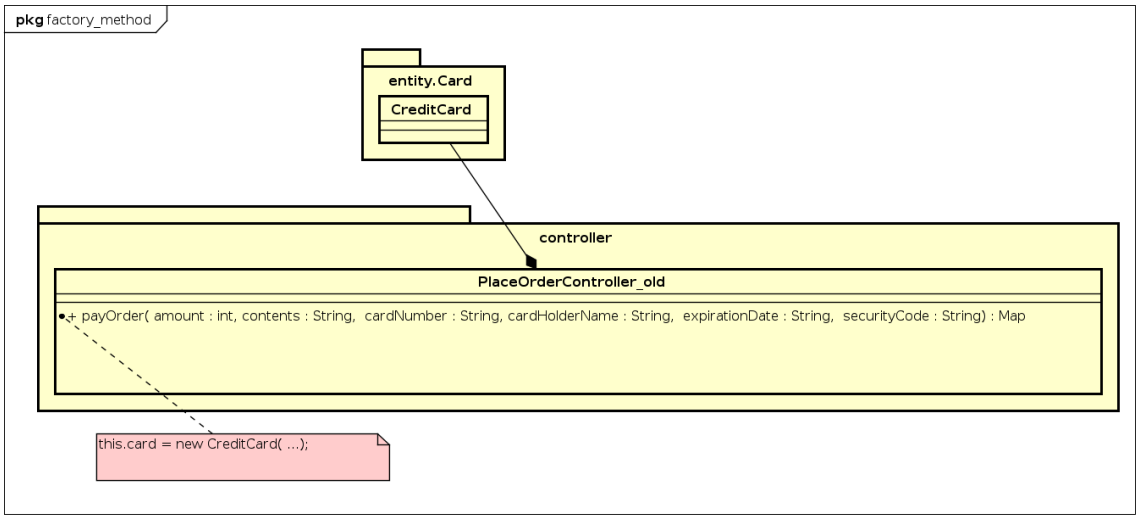
Xử lý bằng Decorator: Do có thể có nhiều cách thông báo lỗi, các cách thông báo này có thể diễn ra đồng thời, nên áp dụng Decorator để giải quyết.



Hình 3.6.2: Xử lý bằng Decorator

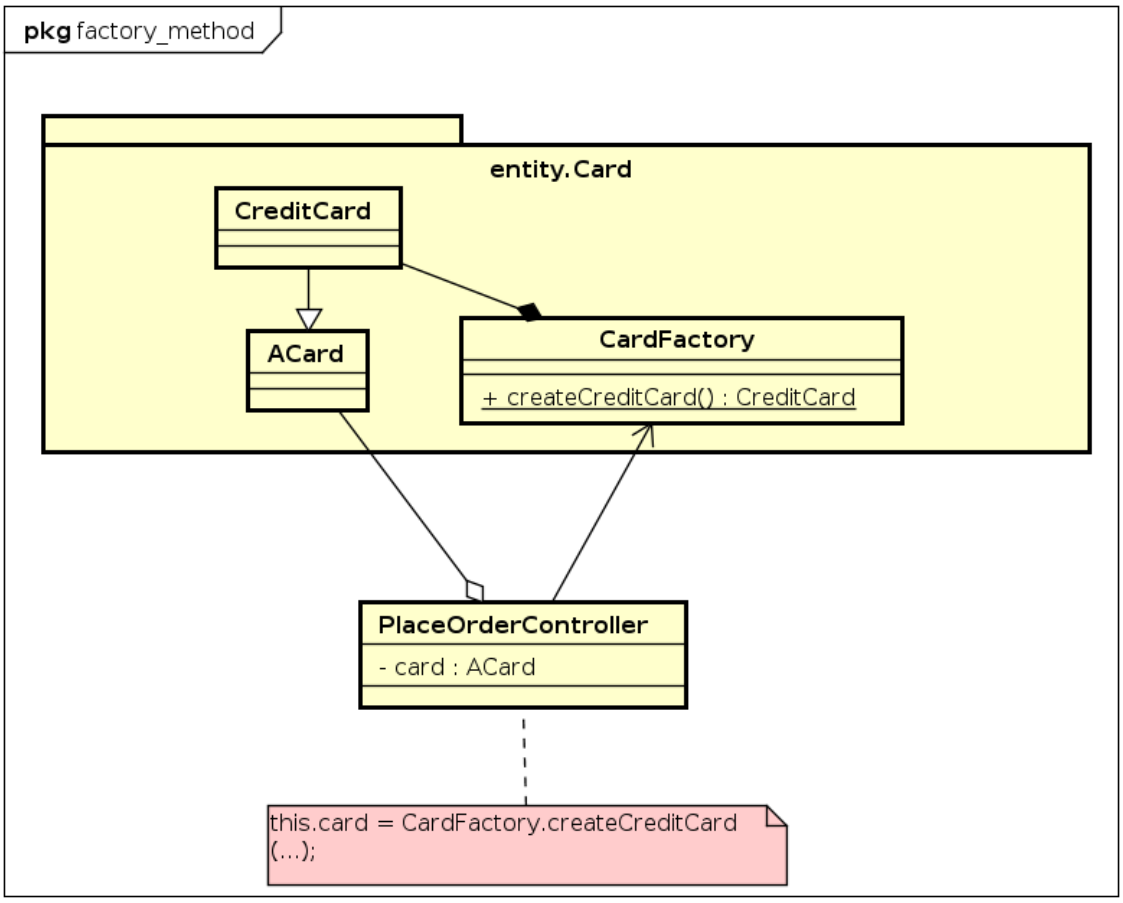
3.7 Vấn đề 2.3.2.3 về kiểu thanh toán mới và giải pháp

Trong module entity.payment đang vi phạm OCP do dối tượng CreditCard được sử dụng trực tiếp ở các lớp khác, việc mở rộng các đối tượng card sau này dẫn tới vi phạm OCP



Hình 3.7.1: Vấn đề entity.card trong codebase

Giải pháp: Sử dụng factory method



Hình 3.7.2: Áp dụng factory method

Sau này, khi mở rộng các kiểu card khác, chỉ cần thêm phương thức khởi tạo trong đối tượng CardFactory. Nếu có nhiều đối tượng card khác loại, có thể áp dụng Abstract factory để giải quyết.

3.8 Các vấn đề về singleton và giải pháp

Trong quá trình cải tiến mã nguồn, chúng ta mong muốn có những đối tượng cần tồn tại duy nhất và có thể truy xuất mọi lúc mọi nơi. Chúng ta có thể sử dụng một biến toàn cục (global variable : public static final). Tuy nhiên, việc sử dụng biến toàn cục nó phá vỡ quy tắc của OOP (encapsulation). Để giải bài toán trên, nhóm em đã sử dụng một giải pháp là Singleton pattern.

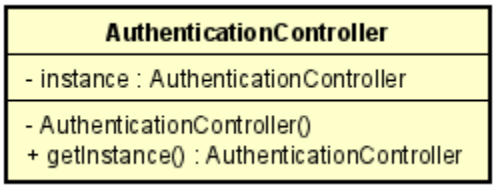
#	*Module*	*Mô tả*	*Lý do*

1	controller/AuthenticationController.java	Các lớp này chỉ cần một thể hiện duy nhất do yêu cầu nghiệp vụ	Do đăng nhập, đăng kí chỉ cần thực hiện tạo đối tượng 1 lần, không cần tạo mới nhiều lần vì các function được sử dụng trong class này không phụ thuộc vào các attribute của class
2	controller/SessionInformation.java		SessionInformation là nơi chứa dữ liệu của một phiên giao dịch trong ứng dụng. Do đó lớp này cần đảm bảo tính duy nhất.
3	entity/cart/Cart.java		Lớp này dùng để lưu trữ những mặt hàng đã được thêm vào cart. Do đó với 1 phiên làm việc chỉ cần 1 lớp Cart tạo ra

Giải pháp :

- controller/AuthenticationController.java:

Class Diagram:



Hình 3.8.1: Áp dụng Singleton vào AuthenticationController

Tạo instance và phương thức truy xuất trong đối tượng:

```
public class AuthenticationController extends BaseController {
      private static AuthenticationController instance;
      public static AuthenticationController getInstance() {
            if (instance == null) {
                        instance = new AuthenticationController();
            }
            return instance;
      }
      private AuthenticationController () {}
}
```

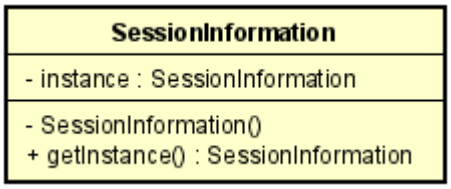
Các class HomeScreenHandler, LoginTest có gọi đến AuthenticationController thay vì sử dụng code cũ

```
new AuthenticationController()
```

thì sẽ gọi bằng cách sau : AuthenticationController.getInstance( )

controller/SessionInformation.java

Class Diagram:



Hình 3.8.1: Áp dụng Singleton vào SessionInfomation

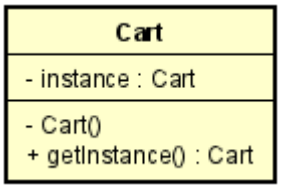
Tạo instance và phương thức truy xuất trong đối tượng:

```
public class SessionInformation extends BaseController {
      private static SessionInformation instance;
      public static SessionInformation getInstance() {
            if (instance == null) {
                        instance = new SessionInformation();
            }
            return instance;
      }
      private SessionInformation () {}
}
```

Các class AuthenticationController, BaseController, PaymentController, ViewCartController, Order, HomeScreenHandler có gọi đến SessionInformation thay vì sử dụng code cũ : new SessionInformation( ) thì sẽ gọi bằng cách sau : SessionInformation.getInstance( )

entity/cart/Cart.java

Class Diagram:



Hình 3.8.3: Áp dụng Singleton vào Cart

Tạo instance và phương thức truy xuất trong đối tượng:

```
public class Cart {
	private static Cart instance;
	public static Cart getInstance() {
		if (instance == null) {
				instance = new Cart();
		}
		return instance;
	}
	private Cart () {}
}
```

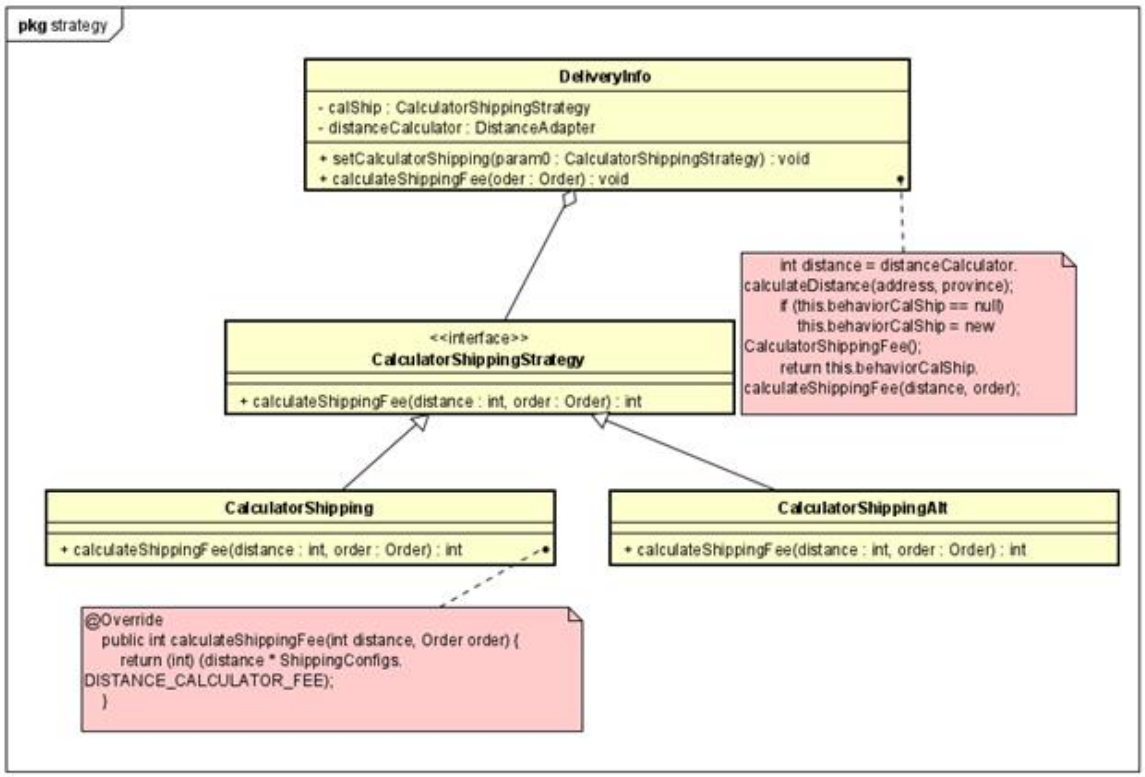
Class SessionInformation có gọi đến Cart thay vì sử dụng code cũ :

new Cart() thì sẽ gọi bằng cách sau : Cart.getInstance( )

3.9 Vấn đề cải tiến phần thanh toán để có thể áp dụng nhiều cách tính phí vận chuyển mới và giải pháp

#	*Module*	*Mô tả*	*Lý do*
1	entity.shipping/	Áp dụng strategy pattern vào cách tính phí vận chuyển	Ở class DeliveryInfo, phương thức calculateShippingFee dùng để tính phí vận chuyển. Trong tương lai, cách tính phí vận chuyển có thể thay đổi (thay đổi công thức tính, thay đổi thư viện tính khoảng cách,…). Do có nhiều cách để tính phí vận chuyển, và những cách này chỉ khác nhau ở cách tính phí, không nên áp dụng Factory (tạo hẳn một đối tượng mới có những thuộc tính và hành động khác), có thể áp dụng Strategy ở phần này

Class Diagram:



Hình 3.9.1: Áp dụng Strategy vào cách tính phí vận chuyển

- Tạo interface CalculatorShippingStrategy chứa phương thức calculateShippingFee:

```
public interface CalculatorShippingStrategy {
    int calculateShippingFee(int distance, Order order);
}
```

- Tạo class CalculatorShippingFee implements interface trên:

```
public class CalculatorShippingFee implements
CalculatorShippingStrategy {
    @Override
    public int calculateShippingFee(int distance, Order order) {
        return (int) (distance *
                    ShippingConfigs.DISTANCE_CALCULATOR_FEE);
    }
}
```

- Trong class DeliveryInfo, sử dụng strategy cho phương thức calculateShippingFee:

```
public int calculateShippingFee(Order order) {
        int distance=distanceCalculator.calculateDistance(address,
                                                    province);
        if (this.behaviorCalShip == null)
```

```
            this.behaviorCalShip = new CalculatorShippingFee();
        return this.behaviorCalShip.calculateShippingFee(distance,
                                                        order);
}
```

- Ở lớp PlaceOrderController, phương thức processDeliveryInfo, sau khi tạo một đối tượng DeliveryInfo sẽ tiếp tục gọi phương thức setCalShip để cài đặt cách tính khoảng cách. Nếu sau này có thay đổi chỉ cần thay đổi class tính khoảng cách là được:

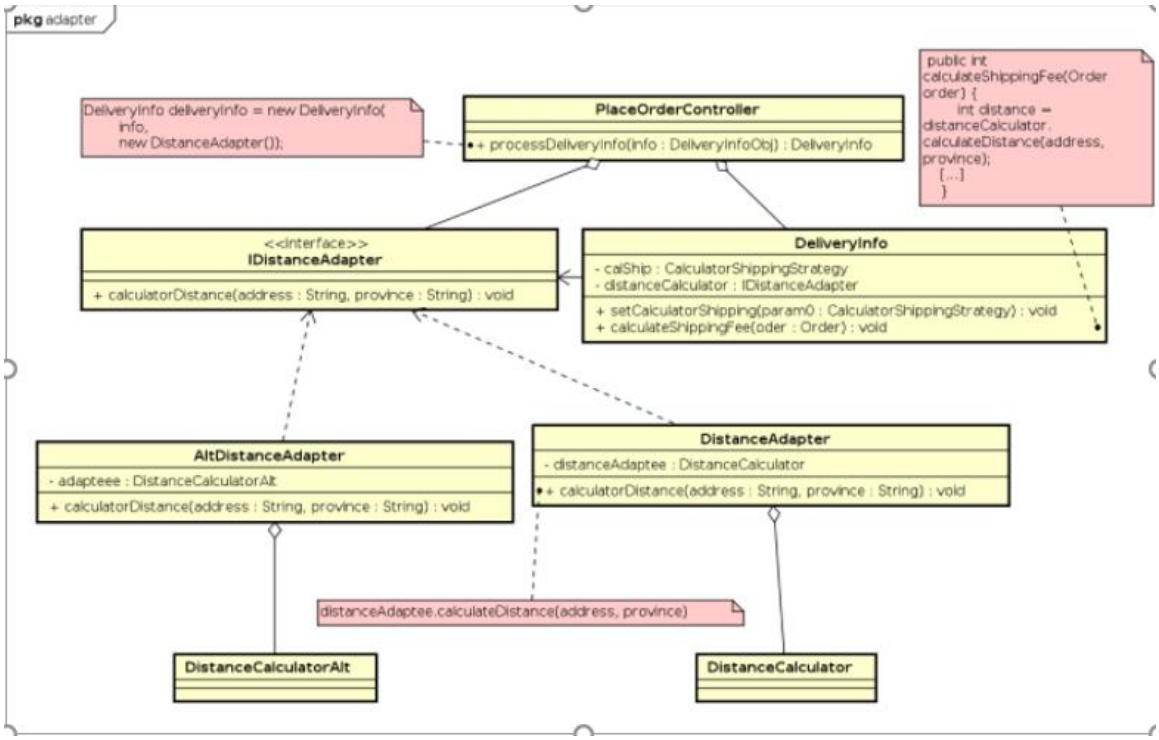
```
public DeliveryInfo processDeliveryInfo(DeliveryInfoObj info) throws
            InterruptedException, IOException,
InvalidDeliveryInfoException {
    LOGGER.info("Process Delivery Info");
    LOGGER.info(info.toString());
    validateDeliveryInfo(info);
    DeliveryInfo deliveryInfo = new DeliveryInfo(
        info,
        new DistanceAdapter());
      //  design pattern: strategy
      deliveryInfo.setCalShip(new CalculatorShippingFee());
    System.out.println(deliveryInfo.getProvince());
    return deliveryInfo;
  }
```

3.10 Các vấn đề xử lí phần thanh toán khi có nhiều cách tính khoảng cách mới và giải pháp

#	*Module*	*Mô tả*	*Lý do*
1	Entity.shipping.DeliveryInfo	Áp dụng Adapter pattern vào cách tính phí vận chuyển	Module entity.Shipping.Deliveryinfo đang phụ thuộc trực tiếp vào thư viện DistanceCalculator nên đang vi phạm nguyên lí thiết kế ISP (Interface Segregation Principle) Và trong tương lai có thể có nhiều cách tính khoảng cách mới. Do đó ta có thể áp dụng Adapter design pattern để giải quyết vấn đề trên

Giải pháp:

- Class diagram:



Hình 3.10.1: Áp dụng Adapter vào cách tính khoảng cách

- Tạo lớp interface IDistanceAdapter:

```
package entity.shipping;

public interface IDistanceAdapter {
    public int calculateDistance(String address, String province);
}
```

- Tạo lớp DistanceAdapter implement interface 'IDistanceAdapter':

```java
public class DistanceAdapter implements IDistanceAdapter{
    DistanceCalculator adaptee;

    public DistanceAdapter(){
        adaptee = new DistanceCalculator();
    }

    public int calculateDistance(String address, String province) {
        return adaptee.calculateDistance(address, province);
    }
}
```

Khi thay đổi thư viện DistanceCalculator thì adaptee gọi đến 1 instance của DistanceCaculator mới đó. Hàm calculateDistance sẽ tính theo adaptee mới.

4 Tổng kết

4.1 Kết quả tổng quan

Sau quá trình giải quyết code smell, clean code, giải quyết các vấn đề vi phạm SOLID, không đảm bảo high cohesion và low coupling, nhìn chung code mới của nhóm có tính dễ đọc, dễ mở rộng hơn.

Cụ thể, khi áp dụng các yêu cầu mới, hay thay đổi các module tích hợp vào mã nguồn đều có thể dễ dàng thực hiện, không phải sửa đổi tới những thành phần đã hoạt động.

Mặc dù vậy, vẫn phải có những sự đánh đổi về cohesion và coupling do có một số nơi vi phạm không thể đảm bảo cả hai điều kiện. (Ví dụ lớp App, việc tách các khối lệnh trong phương thức run sẽ làm code dễ đọc, tuy nhiên sẽ tăng coupling ở mức chấp nhận được – Data coupling).

4.2 Các vấn đề tồn đọng

- Ở phần xem chi tiết sản phẩm (yêu cầu thêm), có thể mở rộng bằng cách áp dụng Decorator.
- Vấn đề tạo ra sản phẩm media mới kế thức lớp Media: Do sản phẩm mới có các thông số quá khác biệt với lớp Media sẵn có, nên chưa có cách giải quyết triệt để
- Khởi tạo các đối tượng trong package entity.media cần truyền rất nhiều tham số, tuy nhiên các tham số này là bắt buộc và chưa có cách giải quyết triệt để.