ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

Version 1.0

DP\_08

Vũ Hoàng Long – 20194106

Nguyễn Đức Thắng – 20194169

Nguyễn Văn Khoa - 20190166

*Hà Nội,* *7/2023*

Mục lục

Mục lục 1

1 Tổng quan 5

1.1 Mục tiêu 5

1.2 Phạm vi 5

1.2.1 Mô tả khái quát phần mềm 5

1.2.2 Các chức năng chính của phần mềm 5

1.2.3 Cấu trúc mã nguồn 6

1.2.4 Các yêu cầu thêm cần cân nhắc cùng quá trình tái cấu trúc 7

1.2.5 Các hoạt động thực thi trên mã nguồn để đạt được mục tiêu kể trên 7

1.2.6 Kết quả dự kiến 7

1.3 Danh sách thuật ngữ 7

1.4 Danh sách tham khảo 7

2 Đánh giá thiết kế cũ 9

2.1 Nhận xét chung 9

2.2 Đánh giá các mức độ coupling và cohesion 9

2.2.1 Coupling 9

2.2.2 Cohesion 11

2.3 Đánh giá việc tuân theo SOLID 12

2.3.1 SRP 12

2.3.2 OCP 12

2.3.3 DIP 13

2.4 Các vấn đề về Clean Code 14

2.4.1 Clear Name 14

2.4.2 Clean Function/Method 14

2.4.3 Clean Class 14

3 Đề xuất cải tiến 15

3.1 Vấn đề (1) và giải pháp 15

3.2 Vấn đề (2) và giải pháp 16

3.3 Vấn đề (3) và giải pháp 18

3.4 Vấn đề (4) và giải pháp 19

3.5 Vấn đề (5) và giải pháp 19

3.6 Vấn đề (6) và giải pháp 22

4 Tổng kết 26

4.1 Kết quả tổng quan 26

4.2 Các vấn đề tồn đọng 26

**Danh sách các minh họa**

[Hình 1: Sơ đồ usecase 6](#_Toc142219644)

[Hình 2: Cấu trúc mã nguồn 6](#_Toc142219645)

[Hình 3: Sơ đồ lớp minh họa SessionInformation 16](#_Toc142219646)

[Hình 4: Sơ đồ lớp minh họa Popup 17](#_Toc142219647)

[Hình 5: Mã nguồn minh họa lớp PopupScreen 17](#_Toc142219648)

[Hình 6: Mã nguồn minh họa lớp client sử dụng Popup 18](#_Toc142219649)

[Hình 7: Sơ đồ lớp minh họa MediaFactory 18](#_Toc142219650)

[Hình 8: Mã nguồn minh họa MediaFactory 18](#_Toc142219651)

[Hình 9: Mã nguồn minh họa lớp AddToCartButton 19](#_Toc142219652)

[Hình 10: Sơ đồ lớp minh họa CalculatingFee 20](#_Toc142219653)

[Hình 11: Sơ đồ trình tự mô tả quá trình gọi các đối tượng của calculate shipping fee 21](#_Toc142219654)

[Hình 12: Mã nguồn mô tả interface CalculatingDistance 21](#_Toc142219655)

[Hình 13: Mã nguồn mô tả lớp CalculatingDistanceB 21](#_Toc142219656)

[Hình 14: Mã nguồn mô tả lớp CalculatingFee 22](#_Toc142219657)

[Hình 15: Mã nguồn mô tả lớp CalculatingFeeB 22](#_Toc142219658)

[Hình 16: Mã nguồn mô tả lớp client sử dụng phương thức calculatingShippingFee 22](#_Toc142219659)

[Hình 17: Sơ đồ lớp mô tả payment 23](#_Toc142219660)

[Hình 18: Sơ đồ trình tự mô tả các đối tượng được gọi khi có thông điệp extractPaymentTransaction 24](#_Toc142219661)

[Hình 19: Mã nguồn mô tả lớp ExtractingPaymentTransaction 24](#_Toc142219662)

[Hình 20: Mã nguồn mô tả lớp ExtractingCreditCard 24](#_Toc142219663)

[Hình 21: Mã nguồn mô tả lớp InterbankPayloadConverter 25](#_Toc142219664)

**Danh sách các bảng**

[Bảng 1: Các vi phạm coupling 11](#_Toc142219638)

[Bảng 2: Các vi phạm cohesion 12](#_Toc142219639)

[Bảng 3: Các vi phạm nguyên lí SRP 12](#_Toc142219640)

[Bảng 4: Các vi phạm nguyên lí OCP 13](#_Toc142219641)

[Bảng 5: Các vi phạm nguyên lí DIP 14](#_Toc142219642)

[Bảng 6: Các vấn đề clear name 14](#_Toc142219643)

# Tổng quan

## Mục tiêu

Báo cáo mẫu thiết kế phần mềm sẽ đánh giá về các vấn đề của mã nguồn. Sau khi chỉ ra các vấn đề kể trên, báo cáo sẽ đề xuất các giải pháp chỉnh sửa mã nguồn.

Nội dung báo cáo gồm 2 phần chính:

* Đánh giá các vấn đề về mã nguồn: mức độ vi phạm coupling, cohesion; mức độ vi phạm các nguyên lí SOLID, các vấn đề về clean code.
* Đề xuất các giải pháp để chỉnh sửa mã nguồn.

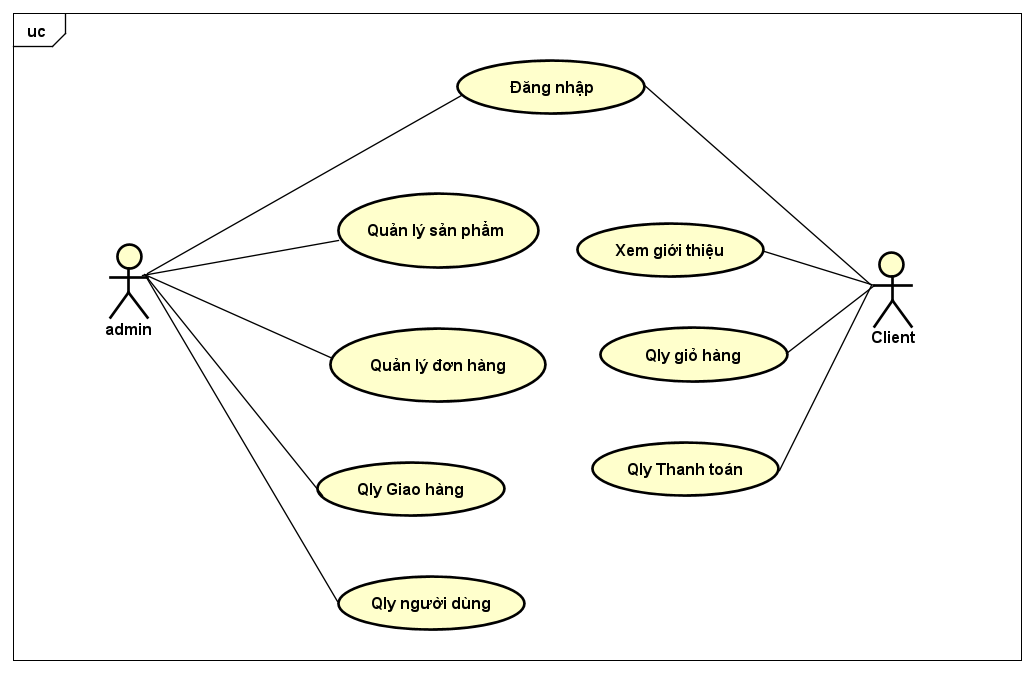
## Phạm vi

### Mô tả khái quát phần mềm

Phần mềm là cửa hàng online giúp cho người dùng có thể đặt hàng qua đó, gồm các sản phẩm đa phương tiện như CD, DVD, Book. Khi người dùng thêm vào giỏ hàng, thì người dùng sẽ thực hiện thanh toán qua Credit Card từ đó người dùng đặt hàng thành công và sản phẩm sẽ được giao đến người dùng.

### Các chức năng chính của phần mềm

Trình bày use case ở đây



Hình : Sơ đồ usecase

### Cấu trúc mã nguồn

Mã nguồn được thiết kế theo mô hình MVC gồm có Model, View và Controller. Thiết kế mã nguồn của hệ thống như hình vẽ sau đây:

A diagram of a computer

Description automatically generated

Hình : Cấu trúc mã nguồn

Trong đó:

* View: Xử lí logic để hiện thị ra giao diện người dùng, theo dõi tương tác của người dùng để gửi cho Controller
* Controller: xử lí dữ liệu được lấy từ database hoặc do người dùng thực hiện tương tác với hệ thống
* Model: Dùng để lưu trữ và truy xuất dữ liệu của hệ thống

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

Trong quá trình tái cấu trúc, cần phải đảm bảo tuân thủ SOLID, cần phân tích hệ thống kỹ càng, thực hiện các thay đổi tránh vi phạm các nguyên tắc SOLID cũng như để cho các lập trình viên dễ đọc, dễ hiểu, dễ dàng làm việc trên mã nguồn khi có những yêu cầu thay đổi có thể trong tương lai

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

Hoạt động để review mã nguồn:

* Xác định các coupling, cohesion trong mã nguồn
* Xác định xem có vi phạm các nguyên tắc SOLID không
* Xác định các vấn đề clean code

Hoạt động để refactor mã nguồn:

* Áp dụng các Design Pattern phù hợp
* Chỉnh sửa những chỗ vi phạm nguyên tắc SOLID

### Kết quả dự kiến

Sau quá trình tái cấu trúc mã nguồn, mã nguồn sẽ đảm bảo không mắc vào các nguyên tắc SOLID, có những chỗ sử dụng Design Pattern đảm bảo cho những yêu cầu có thể có trong tương lai

## Danh sách thuật ngữ

## 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

# Đánh giá thiết kế cũ

## Nhận xét chung

Mã nguồn hiện tại tuy đã đảm bảo được các chức năng của hệ thống tuy nhiên sau khi đọc mã nguồn thấy được mã nguồn có vấn đề khi có yêu cầu được thêm vào hệ thống sẽ làm cho phải thay đổi ở nhiều module do các module phụ thuộc trực tiếp với nhau do đó gây khó khan khi triển khai thay đổi ở hệ thống. Ngoài ra hệ thống có vài chỗ mắc các vấn đề về clean code hay vi phạm nguyên tắc SOLID. Do đó ta cần tái cấu trúc mã nguồn để có thể dễ bảo trì hoặc là thực hiện thay đổi khi có yêu cầu mới.

## Đánh giá các mức độ coupling và cohesion

Một thiết kế mã nguồn tốt là thiết kế có thể dễ dàng phát triển, dễ đọc, dễ hiểu. Thiết kế tốt giúp dễ dàng mở rộng các tính năng mới, có thể sẵn sàng với những yêu cầu thay đổi. Trong đó, cohesion và coupling là các vấn đề cốt lõi cần quan tâm.

Coupling là sự phụ thuộc lẫn nhau giữa các module. Với thiết kế tốt, cần giảm mức độ coupling. Bởi nếu các module quá phụ thuộc lẫn nhau, khi cần thay đổi sẽ phải thay đổi ở nhiều module.

Cohesion là sự liên kết giữa các thành phần trong module. Với thiết kế tốt, cần tăng mức độ cohesion. Với mức độ cohesion cao, các thành phần trong module sẽ liên kết chặt chẽ với nhau để thực hiện một nhiện vụ nhất định, khi cần thay đổi cũng sẽ thay đổi các thành phần trong module, tránh ảnh hưởng đến các module khác.

Sau đây là các vi phạm về coupling và cohesion trong mã nguồn.

### Coupling

Một số vi phạm các mức độ coupling:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Lý do* |
| 1 | Content coupling | Media | Package entity.media | Có chứa thuộc tính protected nhưng module khác có thể truy cập vào |
| DeliveryInfo | Package entity.shipping.DeliveryInfo | Có chứa thuộc tính protected nhưng module khác có thể truy cập vào |
| Order | Package entity.order.Order | Module có chứa thuộc tính protected của module DeliveryInfo |
| 2 | Common coupling | SessionInformation | Lớp thuộc package controller | Các lớp thuộc các package khác nhau tuy nhiên lại cùng đọc và thay đổi dữ liệu của lớp SessionInformation do đó giảm khả năng bảo trì hoặc tái sử dụng (gồm các lớp như sau Authentication, BaseController, Payment Controller, PlaceOrderController,  ViewCartController, Order, MediaHandler, HomeScreenHandler) |
| 3 | Stamp coupling | PlaceOrderControlller | Lớp PlaceOrderController thuộc package controller dùng để xác nhận thông tin giao hang bao gồm phone, name, address | Lớp PlaceOrderController để xác nhận các thuộc tính phone, name, address của DeliveryInfo nên không cần truyền cả object vào |
| Cart | Lớp Cart thuộc package entity.cart có phương thức checkMediaInCart dùng để kiểm tra Media trong giỏ hàng | Lớp Cart có phương thức checkMediaInCart để trả lại CartItem thông qua id của media, do chỉ cần dùng id của media nên không cần phải truyền cả object Media vào |

Bảng : Các vi phạm coupling

### Cohesion

Một số vi phạm về các mức độ cohesion:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Lý do* |
| 1 | Coincidental cohesion | DeliveryInfo | Lớp thuộc gói entity.shiping liên quan đến thông tin vận chuyển đơn hàng của order. | Phương thức calculateShippingFee nên được để trong lớp Order. Chức năng của lớp là lưu thông tin vấn chuyển, không phải là tính phí vận chuyển. |
| 2 | Logical cohesion | PopupScreen | Lớp thuộc package views.screen.popup dùng để hiển thị các popup cho các lớp view như success, error, loading | Lớp PopupScreen đang phải hiển thị nhiều kiểu popup thông qua các phương thức khác nhau: success, error, loading. Các phương thức này tuy thực hiện cùng kiểu chức năng nhưng lại có cách thực hiện khác nhau. |
| 3 | Logical cohesion | ApplicationProgrammingInterface | Lớp thuộc gói utils có hai phương thức để gọi http request với hai phương thức get và post. | Hai phương thức get và post có chức năng giống nhau nhưng lại có cách thức xử lí khác nhau. Nếu có nhiều các phương thức khác như patch, delete, lớp này sẽ thành god class. |
| 4 | Temporal cohesion | HomeScreenHandler và các lớp thuộc gói views | Lớp thuộc package views.screen.home để hiển thị màn hình home | Hai phương thức setupData và setupFunctionality được thực hiện theo một tiến trình nhưng không có liên quan nào khác. |

Bảng : Các vi phạm cohesion

## Đánh giá việc tuân theo SOLID

### SRP

Nhìn chung source code đã đáp ứng khá tốt nguyên lí SRP. Các module chỉ làm một nhiệm vụ liên quan đến một nghiệp vụ nào đó. Tuy nhiên, vẫn còn một số module vi phạm nguyên lí SRP:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | PopupScreen | Lớp thuộc package views.screen.popup dùng để hiển thị các popup cho các lớp view như success, error, loading | Lớp PopupScreen đang phải hiển thị nhiều kiểu popup thông qua các phương thức khác nhau: success, error, loading. Các phương thức này tuy thực hiện cùng kiểu chức năng nhưng lại có cách thực hiện khác nhau. |
| 2 | DeliveryInfo | Lớp thuộc gói entity.shiping liên quan đến thông tin vận chuyển đơn hàng của order. | Phương thức calculateShippingFee nên được để trong lớp Order. Chức năng của lớp là lưu thông tin vấn chuyển, không phải là tính phí vận chuyển. |

Bảng : Các vi phạm nguyên lí SRP

### OCP

Trong thiết kế, có một số lớp vi phạm nguyên lí OCP như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | PopupScreen | Lớp thuộc gói views.screen.popup dùng để hiển thị các popup cho các lớp view như success, error, loading | Trong contructor, nếu có 1 popup mới có liểu style khác với điều kiện khác (thêm tham số mới khác với undecorated) thì sẽ phải thay đổi constructor bằng cách thêm câu lệnh if. |
| 2 | MediaDAO | Lớp thuộc gói dao.media dùng để kết nối với database của các lớp Media | Phương thức getAllMedia mới chỉ lấy thông tin chung của media. Nếu có yêu cầu lấy media với instance là của các lớp con, sẽ cần sử dụng câu lệnh điều kiện switch case xét theo type media để gọi đến phương thức của lớp con tương ứng. |
| 3 | InterbankPayloadConverter | Lớp thuộc gói subsystem.interbank dùng để chuyển đổi các trường cho request, response cho chức năng thanh toán. Thanh toán phụ thuộc vào hình thức thanh toán (card). | Khi có một phương thức thanh toán mới, thông tin thanh toán sẽ thay đổi, request, response với interbank cũng sẽ thay đổi. |

Bảng : Các vi phạm nguyên lí OCP

### DIP

Hầu hết các lớp đều tuân thủ đúng nguyên lí DIP. Tuy nhiên còn một số lớp vi phạm nguyên tắc này:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | DeliveryInfo | Lớp thuộc gói entity.shiping liên quan đến thông tin vận chuyển đơn hàng của order. | Lớp DeliveryInfo sử dụng trực tiếp phương thức calculateDistance của lớp DistanceCalculator. Lớp Order phụ thuộc lớp DeliveryInfo, đồng thời DeliveryInfo cũng phụ thuộc Order. |
| 2 | MediaDAO | Lớp thuộc gói dao.media dùng để kết nối với database của các lớp Media | Phương thức getAllMedia mới chỉ lấy thông tin chung của media. Nếu có yêu cầu lấy media với instance là của các lớp con, sẽ cần sử dụng câu lệnh điều kiện switch case xét theo type media để gọi đến phương thức của lớp con tương ứng. |

Bảng : Các vi phạm nguyên lí DIP

## Các vấn đề về Clean Code

### Clear Name

Cách đặt tên của source code khá tốt. Tuy nhiên còn một số các đoạn code chưa còn gặp vấn đề về clear name.

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Phần vi phạm* | *Sửa đổi* |
| 1 | DeliveryInfo | Chỉ số 1.2 trong phương thức calculateShippingFee khá khó hiểu. | DISTANCE\_RATIO = 1.2 |
| 2 | SuccessPopupScreen | Hằng số “tickgreen.png” | TICKGREEN = "tickgreen.png"; |
| 3 | LoadingPopupScreen | Hằng số “loading.gif” | LOADING = "loading.gif"; |
| 4 | ErrorPopupScreen | Hằng số “tickerror.png” | TICKERROR = "tickerror.png" |
| 5 | CartScreenHandler | Hằng số “Logo.png” | LOGO = "Logo.png" |

Bảng : Các vấn đề clear name

### Clean Function/Method

Mã nguồn đã đáp ứng tốt về clean method, không cần sửa đổi gì thêm.

### Clean Class

Mã nguồn đã đáp ứng tốt về clean class, không cần sửa đổi gì thêm.

# Đề xuất cải tiến

Một số yêu cầu thay đổi có thể dẫn đến vi phạm các mức độ cohesion, coupling, nguyên lí SOLID của mã nguồn được tóm tắt như sau:

(1) Vấn đề về SessionInformation.

(2) Khi thêm một loại popup mới, phải thêm phương thức mới cho popup, lâu dần sẽ khiến popup thành god class.

(3) Khi thêm một loại Media mới dẫn đến thay đổi cách lấy thông tin của Media.

(4) Khi thêm các button mới vào MediaHandler cần sửa thêm lớp này.

(5) Khi thay đổi thư viện tính khoảng cách hoặc thay đổi cách tính giá ship, cần thay đổi nhiều ở lớp Order.

(6) Khi có thêm phương thức thanh toán mới, cần sửa đổi mã nguồn để xử lí được thông tin thanh toán mới.

## Vấn đề (1) và giải pháp

Trong mã nguồn ban đầu, trong lớp SessionInformation, các biến như mainUser và cartInstance là các biến global do đó có thể làm cho các lớp có thể được truy cập và sửa đổi do đó khó để xem xem những lớp nào đã sửa đổi dữ liệu. Điều này vi phạm common coupling.

Giải pháp: Áp dụng mẫu thiết kế Singleton để có thể tạo ra 1 sessionInformation instance và truy cập vào các biến theo phương thức get/set.

Sơ đồ lớp và code minh họa giải pháp:

A close-up of a computer code

Description automatically generated

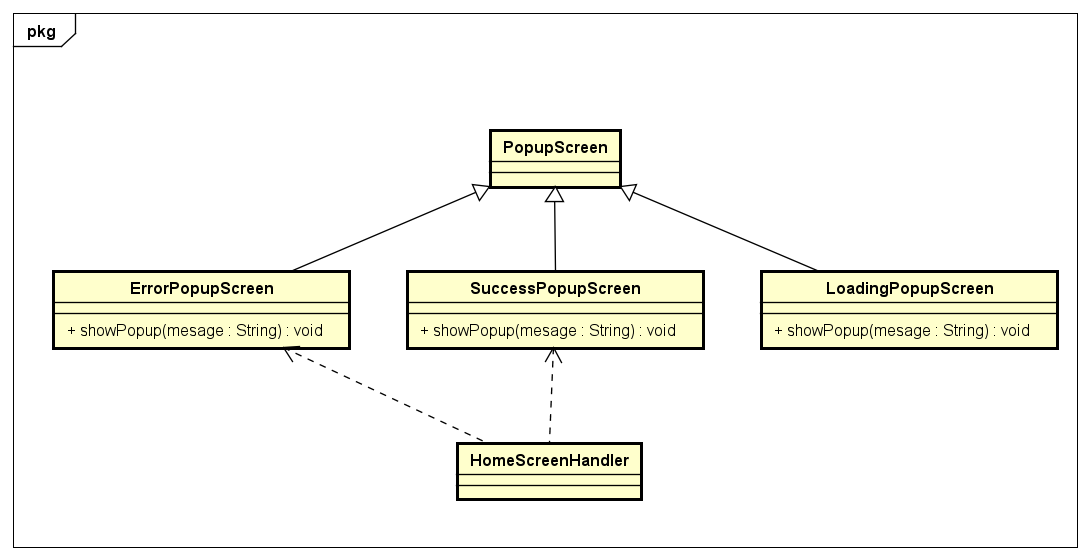
Hình : Sơ đồ lớp minh họa SessionInformation

## Vấn đề (2) và giải pháp

Lớp PopupScreen hiện đang có 3 phương thức success, loading, error. Nếu thêm một loại popup mới, ta cần thêm một phương thức mới. Điều này làm cho PopupScreen có thể trở thành god class. Điều này khiến PopupScreen vi phạm logical coupling và nguyên lí SRP.

Giải pháp: tạo ra các lớp con tương ứng với mỗi loại popup. Nếu Screen nào cần dùng popup sẽ gọi đến từng lớp popup con. Mỗi lớp popup con sẽ được tạo bằng các cách khác nhau, có image khác nhau, thời gian hiện/ẩn khác nhau.

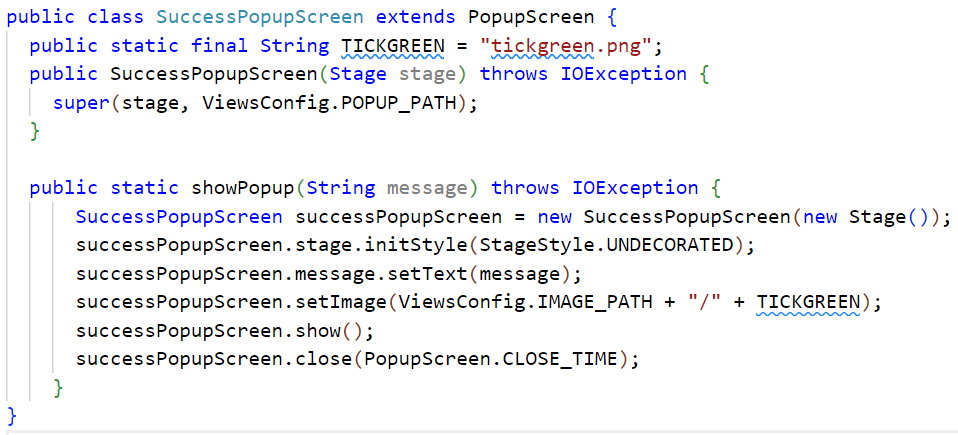
Sơ đồ lớp minh họa:



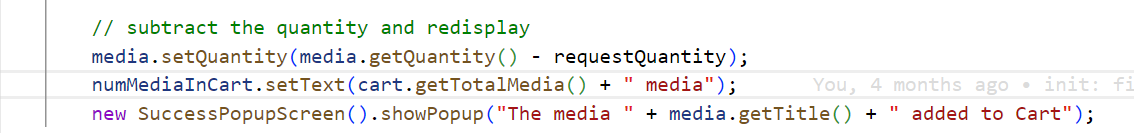
Hình : Sơ đồ lớp minh họa Popup

Ta để PopupScreen làm lớp cha, có phương thức showPopup() để hiển thị popup cùng message được truyền vào. Các lớp con override phương thức showPopup(). Các lớp thuộc package views có sử dụng popup (ở đây lấy một ví dụng minh họa là lớp HomeScreenHandler có cả error popup và success popup) phụ thuộc vào các lớp Popup con này. Khi có một Popup dạng mới, ta chỉ cần tạo một lớp con mới.

Mã nguồn minh họa:



Hình : Mã nguồn minh họa lớp PopupScreen



Hình : Mã nguồn minh họa lớp client sử dụng Popup

## Vấn đề (3) và giải pháp

Lớp MediaDAO đã vi phạm nguyên lí OCP, do phương thức getAllMedia mới chỉ lấy thông tin chung của media. Nếu có yêu cầu lấy media với instance là của các lớp con, sẽ cần sử dụng câu lệnh điều kiện switch case xét theo type media để gọi đến phương thức của lớp con tương ứng.

Giải pháp đề xuất: sử dụng mẫu thiết kế Factory Method để tạo ra từng loại Media riêng, lấy thông tin theo từng Media.

Sơ đồ lớp minh họa giải pháp:

A screenshot of a computer

Description automatically generated

Hình : Sơ đồ lớp minh họa MediaFactory

Mã nguồn mô tả giải pháp:

A screen shot of a computer code

Description automatically generated

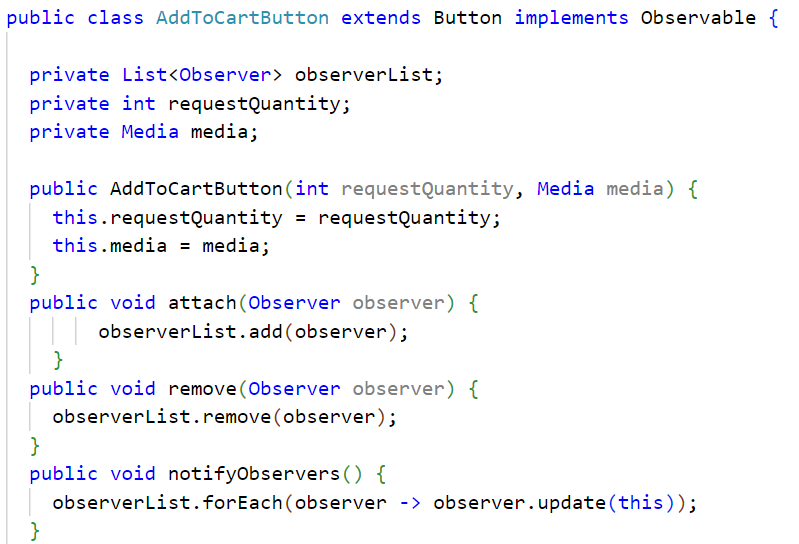
Hình : Mã nguồn minh họa MediaFactory

## Vấn đề (4) và giải pháp

Ở màn hình trang chủ hiển thị danh sách các media (tương ứng với lớp MediaHandler). Mỗi lớp này có một nút để thêm media vào giỏ hàng. Nếu có thêm một màn hình chi tiết media, màn hình này cũng có nút thêm vào giỏ hàng thì cần phải thêm mã nguồn tương tự.

Giải pháp đề xuất: Tạo ra một lớp AddToCartButton để các lớp gioa diện khác sử dụng chung thành phần này.

Mã nguồn minh họa giải pháp:



Hình : Mã nguồn minh họa lớp AddToCartButton

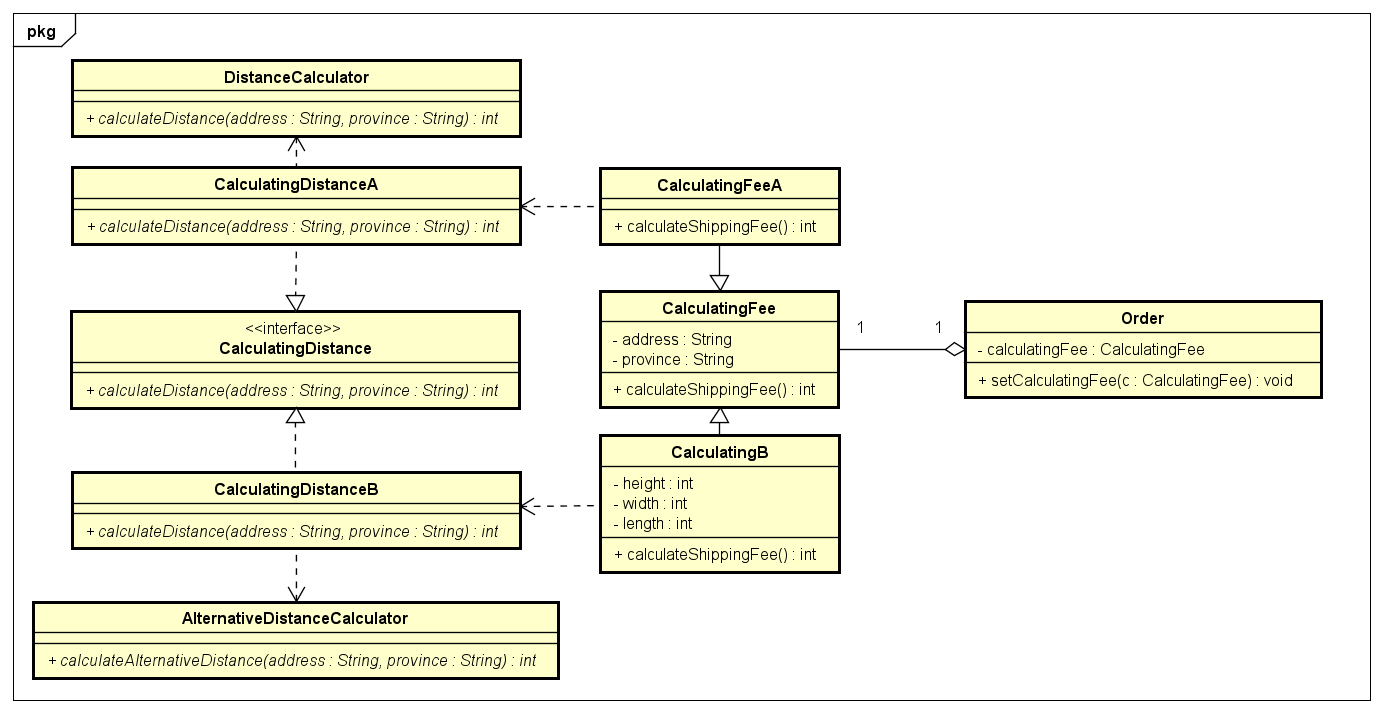
## Vấn đề (5) và giải pháp

Khi có yêu cầu thay đổi thư viện tính khoảng cách hoặc thay đổi công thức tính phí ship, ta thấy lớp DeliveryInfo vi phạm một số vấn đề sau:

* stamp coupling ở phương thức calculateShippingFee(): phương thức truyền vào tham số order nhưng không dùng đến đối tượng.
* Coincidental cohesion: phương thức calculateShippingFee() sử dụng để tính số tiền ship cho đơn hàng. Tuy nhiên, đây là lớp để lưu thông tin vận chuyển nên phương thức này nên được để ở lớp khác.
* Vi phạm nguyên lí DIP: phương thức calculateShippingFee() phụ thuộc vào lớp DistanceCalculator là lớp cụ thể. Điều này khiến phương thức bị thay đổi nhiều nếu phương thức sử dụng cách tính khoảng cách từ một lớp khác có phương thức khác, tham số khác. Đồng thời lớp Order phụ thuộc vào lớp DistanceCalculator, lớp DistanceCalculator cũng phụ thuộc vào lơp Order. Khi thay đổi công thức tính tiền tip, lớp DistanceCalculator sẽ phải thay đổi.

Giải pháp đề xuất: sử dụng mẫu thiết kế Adapter để xử lí vấn đề có các phương thức tính khoảng cách với interface khác nhau. Tạo thêm lớp để tính giá trị tiền tip cho đơn hàng. Với lớp này, sử dụng mẫu thiết kế strategy để tạo các cách tính tiền tip khác nhau.

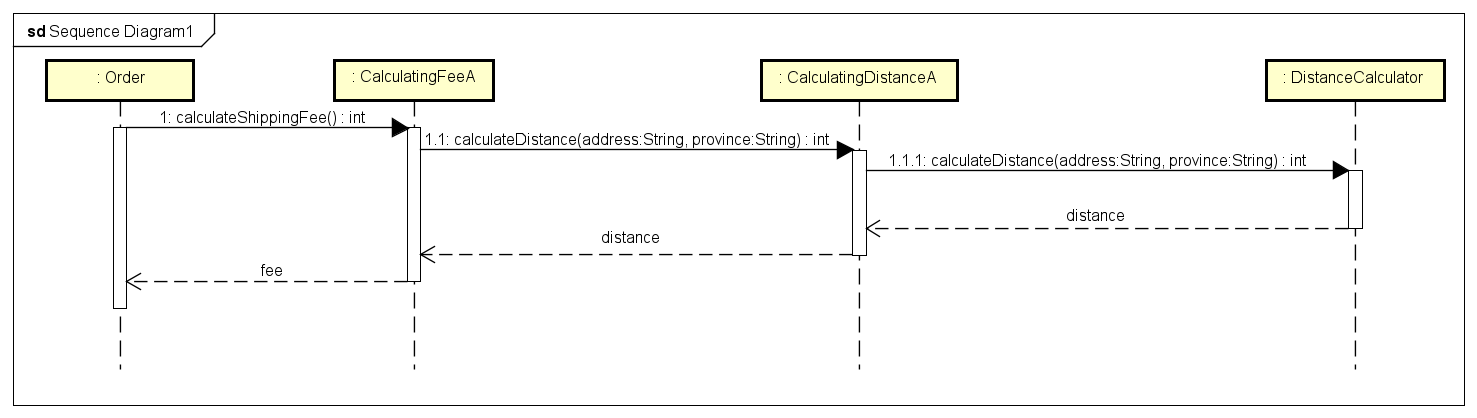
Sơ đồ lớp minh họa giải pháp:



Hình : Sơ đồ lớp minh họa CalculatingFee

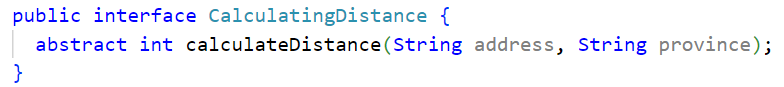
Đối với các lớp tính khoảng cách, tạo một interface CalculatingDistance có phương thức calculateDistance(). Hai lớp CalculatingDistanceA, CalculatingDistanceB implement interface trên và phụ thuộc vào các lớp tính khoảng cách tương ứng là DistanceCalculator và AlternativeDistanceCalculator. Hai lớp CalculatingDistanceA, CalculatingDistanceB đóng vai trò là các Adapter. Đối với việc tính tiền ship, sử dụng mẫu thiết kế strategy tạo một lớp CalculatingFee có phương thức calculateShippingFee(), các lớp con CalculatingFeeA, CalculatingFeeB là các lớp con và override phương thức calculateShippingFee() với các công thức khác nhau. Lớp Order thêm một thuộc tính là calculatingFee để lựa chọn công thức tính tiền ship.

Sơ đồ trình tự mô tả giải pháp:

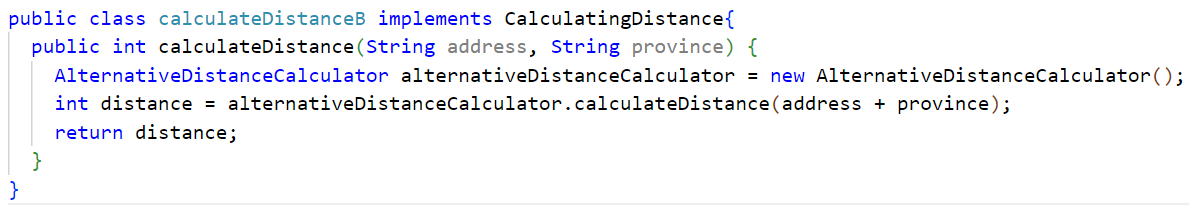


Hình : Sơ đồ trình tự mô tả quá trình gọi các đối tượng của calculate shipping fee

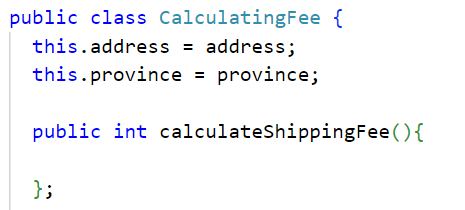
Mã nguồn mô tả giải pháp:



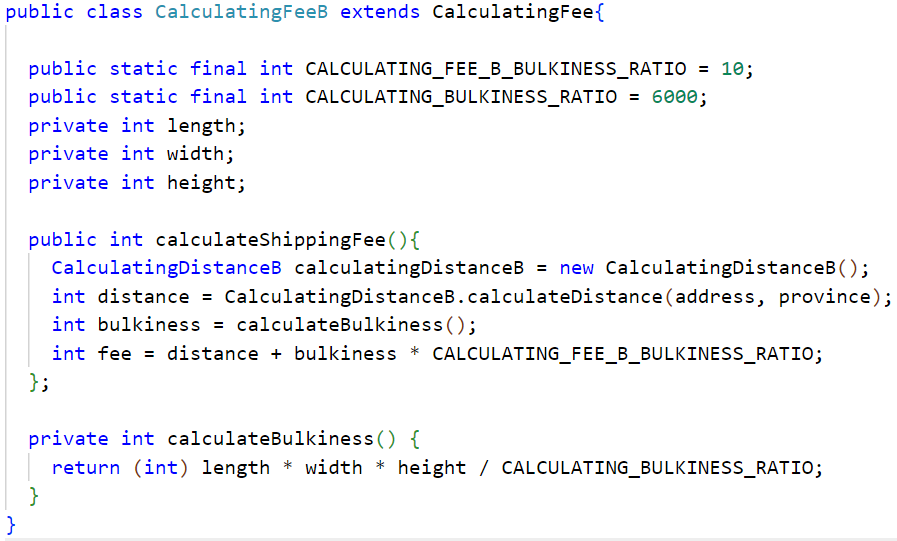
Hình : Mã nguồn mô tả interface CalculatingDistance



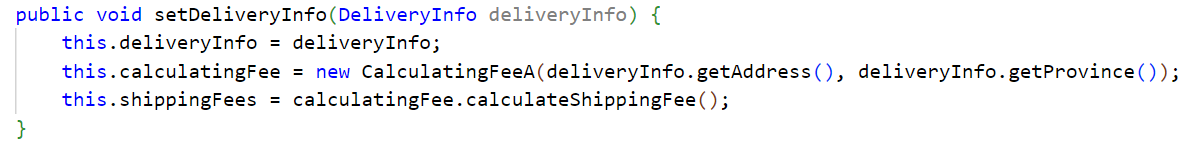
Hình : Mã nguồn mô tả lớp CalculatingDistanceB



Hình : Mã nguồn mô tả lớp CalculatingFee



Hình : Mã nguồn mô tả lớp CalculatingFeeB



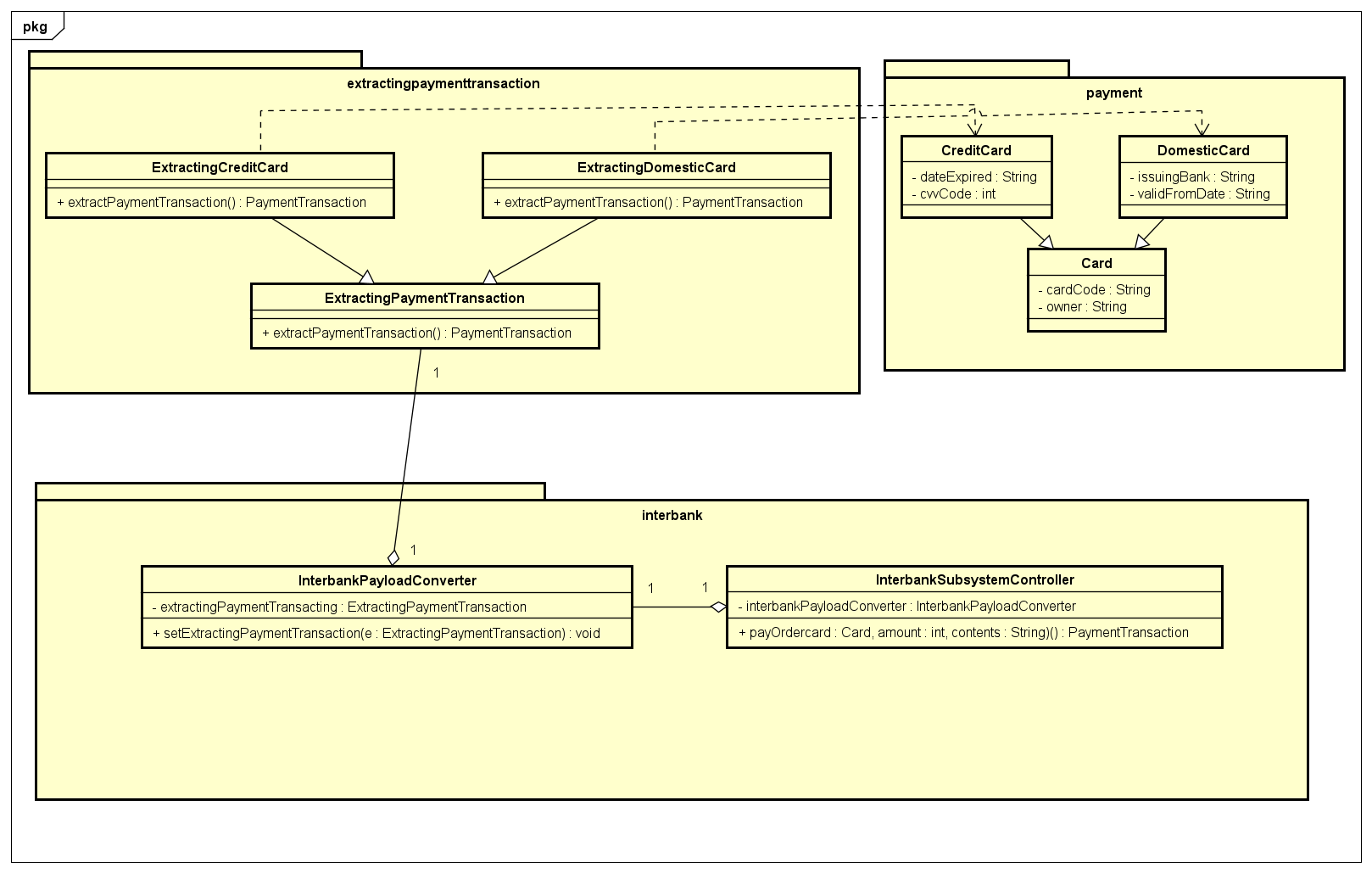
Hình : Mã nguồn mô tả lớp client sử dụng phương thức calculatingShippingFee

## Vấn đề (6) và giải pháp

Lớp InterbankPayloadConverter vi phạm nguyên lí OCP. Khi có một phương thức thanh toán mới (ví dụ DomesticCard) với các thông tin thanh toán khác, lớp InterbankPayloadConverter sẽ phải thay đổi phương thức extractPaymentTransaction để phù hợp với từng loại Card.

Giải pháp: Tạp lớp Card là lớp cha cho các lớp CreditCard, DomesticCard. Sử dụng mẫu thiết kế strategy để tạo ra lớp ExtractingPaymentTransaction và các lớp con ExtractingCreditCard, ExtractingDomesticCard để thực hiện phương thức extractPaymentTransaction theo các cách khác nhau.

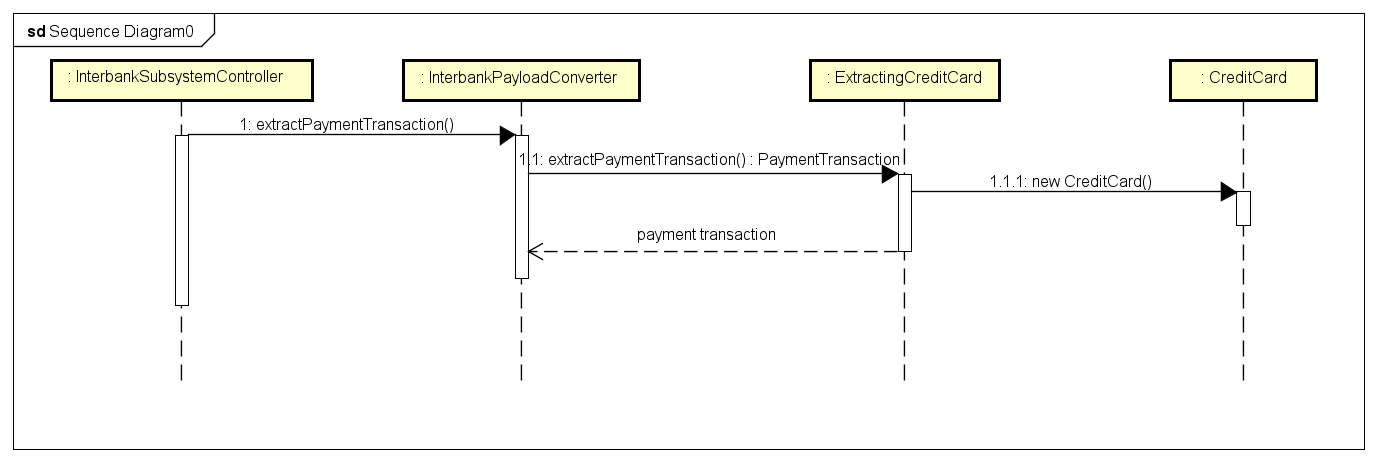
Sơ đồ lớp minh họa giải pháp:



Hình : Sơ đồ lớp mô tả payment

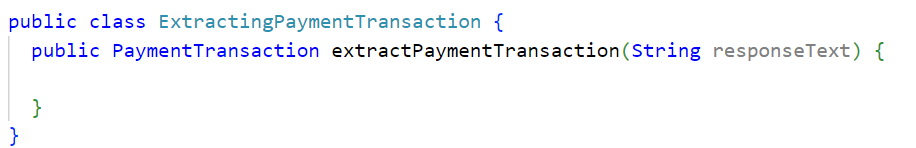
Ta tạo một lớp Card, hai lớp CreditCard và DomesticCard kế thừa từ Card. Tạo thêm package extractingpaymenttransaction để tạo ra các lớp kế thừa từ ExtractingPaymentTransaction với các kiểu extractPaymentTransaction khác nhau. Lớp ExtractingCreditCard lấy thông tin của CreditCard, còn ExtractingDomesticCard lấy thông tin của DomesticCard. Lớp InterbankPayloadConverter thêm thuộc tính extractingPaymentTransaction để sử dụng các kiểu extracting.

Sơ đồ trình tự mô tả giả pháp:

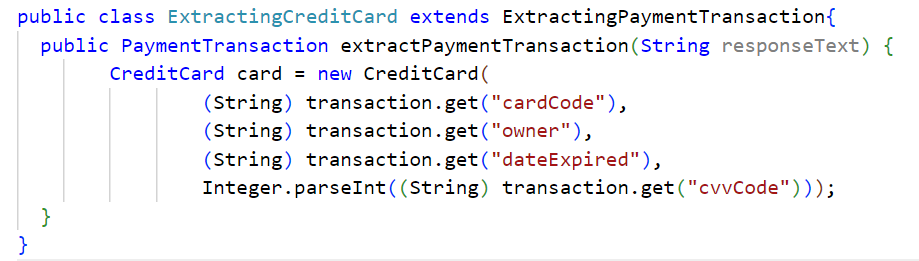


Hình : Sơ đồ trình tự mô tả các đối tượng được gọi khi có thông điệp extractPaymentTransaction

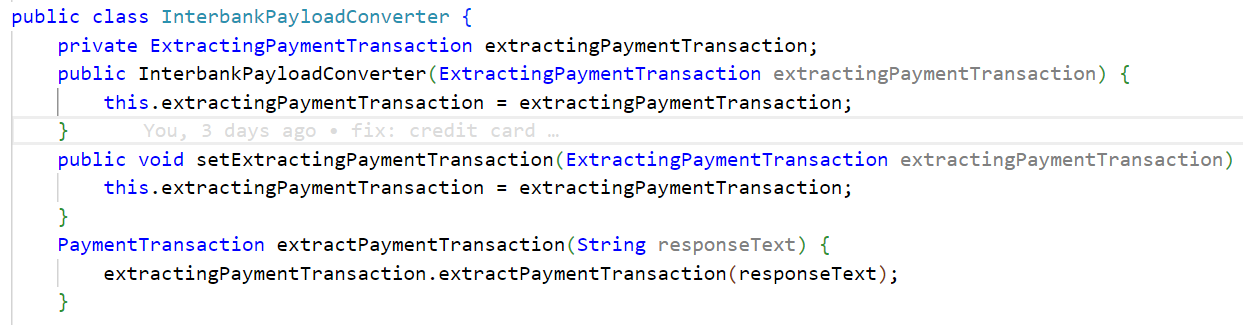
Mã nguồn mô tả giải pháp:



Hình : Mã nguồn mô tả lớp ExtractingPaymentTransaction



Hình : Mã nguồn mô tả lớp ExtractingCreditCard



Hình : Mã nguồn mô tả lớp InterbankPayloadConverter

# Tổng kết

## Kết quả tổng quan

So với thiết kế ban đầu, nhóm đã sửa đổi để cải thiện một số vấn đề như sau:

* Các lớp Card, Media đã được thiết kế lại để phù hợp hơn với việc mở rộng thêm các lớp con mới.
* SessionInfomation được tối ưu, chỉ còn duy nhất một instance trong hệ thống để dễ dàng quản lí.
* Xử lí được các vấn đề về thay đổi công thức tính của khoảng cách giao hàng và tính phí giao hàng.

## Các vấn đề tồn đọng

Tuy nhiên, một số vấn đề còn tồn đọng trong source code như sau:

* Các lớp thuộc gói views còn vi phạm temporal coupling nhưng chưa được sửa.
* Lớp ApplicationProgrammingInterface còn có hai phương thức vi phạm logical coupling nhưng hiện tại chưa được sửa.