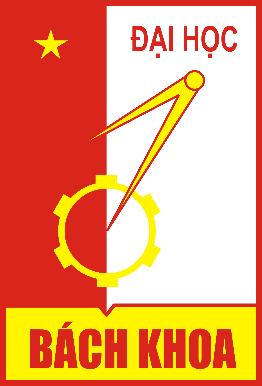
**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

----- □ 🕮 □ -----



**ĐỒ ÁN MÔN HỌC**

**MẪU THIẾT KẾ PHẦN MỀM**

**Giảng viên:** TS. Nguyễn Thị Thu Trang  
 TS. Bùi Thị Mai Anh

**Nhóm sinh viên thực hiện:**

Lương Thái Nam 20194126

Lê Minh Vũ 20194212

Nguyễn Xuân Sơn 20194160

Trần Văn Quang 20194147

**Hà Nội, tháng 7 năm 2023**

**MỤC LỤC**

[**1. Tổng quan 4**](#_heading=h.gjdgxs)

[1.1. Mục tiêu 4](#_heading=h.30j0zll)

[1.2. Phạm vi 4](#_heading=h.1fob9te)

[1.2.1. Mô tả khái quát phần mềm 4](#_heading=)

[1.2.2. Các chức năng chính của phần mềm 4](#_heading=)

[1.2.3. Cấu trúc mã nguồn 6](#_heading=)

[1.2.4. Các yêu cầu cần cân nhắc thêm trong quá trình tái cấu trúc 7](#_heading=)

[1.2.5. Các hoạt động review, refactor 7](#_heading=)

[1.2.6. Kết quả dự kiến 7](#_heading=)

[1.3. Danh sách thuật ngữ 7](#_heading=h.tyjcwt)

[1.4. Danh sách tài liệu tham khảo 9](#_heading=h.3dy6vkm)

[**2. Đánh giá thiết kế cũ 9**](#_heading=h.1t3h5sf)

[2.1. Nhận xét chung 9](#_heading=h.4d34og8)

[2.2. Đánh giá các mức độ coupling và cohesion 9](#_heading=h.2s8eyo1)

[2.2.1. Mức độ coupling 10](#_heading=)

[2.2.2. Mức độ cohesion 12](#_heading=)

[2.3. Đánh giá việc tuân theo SOLID 15](#_heading=h.17dp8vu)

[2.3.1. Single Responsibility Principle (SRP) 15](#_heading=)

[2.3.2. Open Closed Principle (OCP) 16](#_heading=)

[2.3.3. Liskov Substitution Principle (LSP) 16](#_heading=)

[2.3.4. Interface Segregation Principle (ISP) 16](#_heading=)

[2.3.5. Dependency Inversion Principle (DIP) 17](#_heading=)

[2.4. Các vấn đề về Clean Code 17](#_heading=h.3rdcrjn)

[2.4.1. Clean name 17](#_heading=)

[2.4.2. Clean function/method 18](#_heading=)

[2.4.3. Clean class 19](#_heading=)

[**3. Đề xuất cải tiến 20**](#_heading=h.26in1rg)

[3.1. Đề xuất khi phát sinh thêm một loại Media mới: AudioBook. 20](#_heading=h.lnxbz9)

[3.2. Đề xuất khi thêm màn hình: Xem chi tiết sản phẩm 22](#_heading=h.2jxsxqh)

[3.3. Đề xuất khi thay đổi yêu cầu khi load giao diện 23](#_heading=h.3j2qqm3)

[3.4. Cải tiến vấn đề thay đổi phương thức tính khoảng cách sử dụng thư viện mới và thay đổi công thức tính phí vận chuyển 24](#_heading=h.4i7ojhp)

[3.5. Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card) 26](#_heading=h.3whwml4)

[**4. Tổng kết 28**](#_heading=h.ihv636)

[4.1. Kết quả tổng quan 28](#_heading=h.32hioqz)

[4.2. Các vấn đề còn tồn đọng 29](#_heading=h.1hmsyys)

## Tổng quan

### Mục tiêu

Tài liệu này được sử dụng làm báo cáo môn học Mẫu thiết kế phần mềm - IT4536 của Trường Công nghệ thông tin và truyền thông – Đại học Bách Khoa Hà Nội.

Đối tượng người đọc hướng đến những lập trình viên, những người đã có kiến thức cơ sở về lập trình hướng đối tượng, người thiết kế và xây dựng hệ thống phần mềm hoặc những người muốn tham khảo tài liệu để viết hướng dẫn nghiệp vụ.

Tài liệu này là kết quả làm việc của nhóm tác giả, nhằm tổng kết quá trình thực hiện và kết quả việc tái cấu trúc lại mã nguồn dựa trên một hệ thống phần mềm có sẵn được cung cấp bởi giảng viên giảng dạy môn học. Nhóm tác giả đã sử dụng các kiến thức về Cohesion, Coupling, SOLID, và các Design Pattern phổ biến được giảng dạy trong môn học để áp dụng vào phân tích và cải tiến thiết kế hiện tại.

### Phạm vi

#### Mô tả khái quát phần mềm

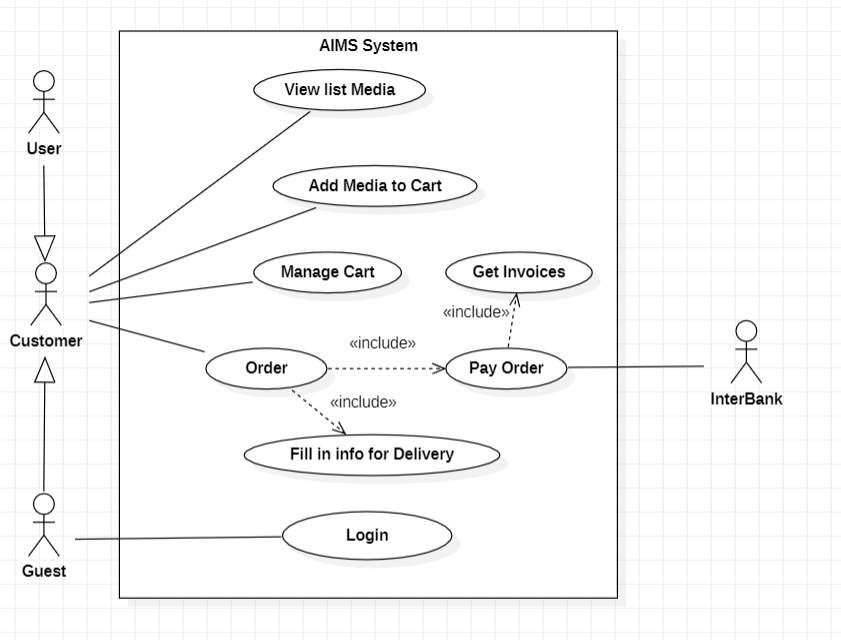
Phần mềm là một cửa hàng online giúp người dùng đặt hàng các sản phẩm đa phương tiện như sách, đĩa CD, đĩa DVD từ cửa hàng. Sau khi thực hiện đặt hàng, đơn hàng sẽ được thanh toán online qua Credit Card (Thẻ tín dụng) của người dùng và sau đó sản phẩm sẽ được vận chuyển tới địa chỉ mà khách hàng yêu cầu.

Về đánh giá sơ bộ, phần mềm đã đáp ứng được các chức năng cơ bản tại thời điểm hiện tại, có thể đưa vào sử dụng và vận hàng. Tuy nhiên do chưa đảm bảo việc tuân thủ các nguyên lý SOLID nên phần mềm sẽ không có khả năng đáp ứng được một số yêu cầu thay đổi có thể phát sinh trong tương lai. Do vậy việc tái cấu trúc lại code phần mềm là việc cần thiết.

#### Các chức năng chính của phần mềm

Phần mềm AIMS có một số tính năng chính như sau:

* Xem danh sách các sản phẩm đang có trong cửa hàng
* Thêm các sản phẩm trong danh mục vào giỏ hàng
* Điền thông tin thanh toán đơn hàng, xác định địa chỉ giao hàng và hướng dẫn giao hàng.



**Hình 1:** Use case Diagram tổng quan

Phần mềm có các tác nhân: Guest, User, Customer và InterBank. Guest, User đều có chung một số hành vi nghiệp vụ khi tương tác với phần mềm, cụ thể là Guest dù không đăng nhập vẫn có thể thực hiện việc đặt hàng. Do đó Guest và User đều generalization Customer (tác nhân trừu tượng) và đều có thể đem lại doanh thu thông qua việc đặt mua, thanh toán đơn hàng.

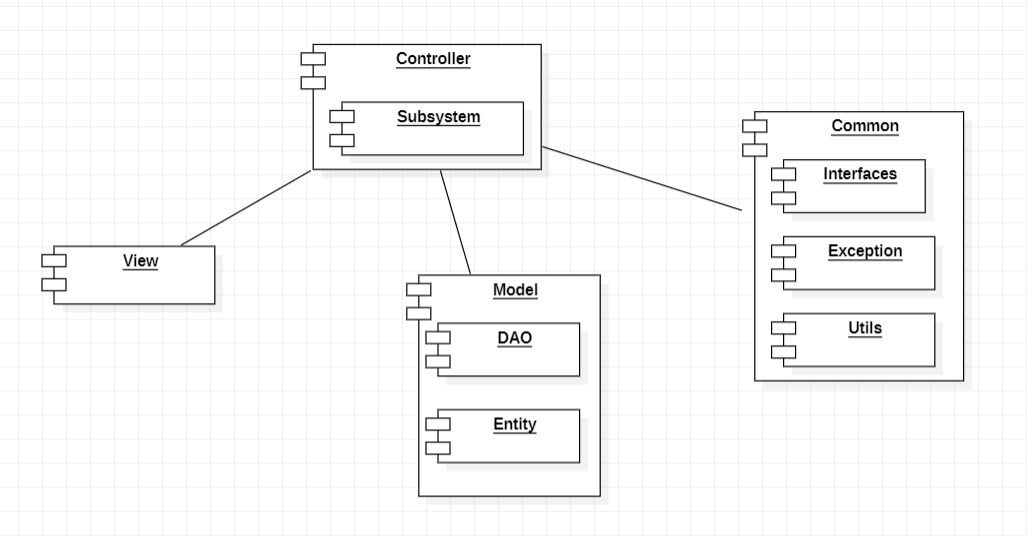
Customer có thể thực hiện việc View List Media (Xem danh sách sản phẩm), Add Media to Cart (Thêm sản phẩm vào giỏ hàng), Manage Cart (Xem, sửa giỏ hàng), Order (Đặt hàng). Trong khi thực hiện đặt hàng, sẽ có nghiệp vụ Fill in Info for Delivery (Điền thông tin giao hàng) và Pay Order (Thanh toán đơn hàng), ngoài ra có thể Get Invoices (Nhận hóa đơn) sau đi đặt hàng thành công. Nghiệp vụ này có sự tham gia của tác nhân InterBank, là một tác nhân hệ thống, đại diện cho hệ thống thanh toán Online.

#### Cấu trúc mã nguồn

Mã nguồn của phần mềm được thiết kế rõ ràng theo mô hình MVC, gồm ba tầng là Model, View và Controller.

Trong mô hình MVC, tầng View đảm nhận vai trò xử lý logic liên quan đến giao diện người dùng, bao gồm việc hiển thị các màn hình, lấy dữ liệu từ người dùng và chuyển tiếp cho tầng Controller, cũng như điều hướng giữa các màn hình. Tầng Controller xử lý logic liên quan đến luồng dữ liệu, bao gồm xác thực dữ liệu người dùng và tương tác với các hệ thống con để thực hiện thanh toán đơn hàng của người dùng, ở tầng này bao gồm cả Subsystem bên trong. Tầng Model chịu trách nhiệm lưu trữ dữ liệu của hệ thống và bao gồm hai thành phần chính là Data Access Object và Entity. Ngoài ra, hệ thống còn có một thành phần chung gọi là Common, bao gồm các Interfaces, Exceptions và Utils, được sử dụng chung trong toàn bộ hệ thống.

Tổng quan thiết kế ban đầu của hệ thống phần mềm được module hóa và miêu tả như biểu đồ sau:



**Hình 2** Cấu trúc của hệ thống ban đầu

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

Việc tái cấu trúc mã nguồn của phần mềm với mục tiêu giúp phần mềm có khả năng đáp ứng với các thay đổi trong tương lai và giúp cho code “Clean” giúp cho các lập trình dễ dàng làm việc trên mã nguồn.

Việc tái cấu trúc đảm bảo tuân thủ SOLID, đặc biệt là OCP (Open-closed Principle) và DIP (Dependency Inversion Principle). Quá trình tái cấu trúc phần mềm để đáp ứng SOLID có thể bao gồm việc chia nhỏ, tái cấu trúc và tái thiết kế các module, lớp, và giao diện. Nó yêu cầu một quá trình phân tích kỹ lưỡng, đánh giá và thực hiện các thay đổi một cách cẩn thận để đảm bảo tính đúng đắn và hiệu quả của hệ thống.

Trong quá trình tái cấu trúc, yêu cầu không được thay đổi về mặc chức năng và phi chức năng của hệ thống ban đầu.

#### Các hoạt động review, refactor

Việc review với mã nguồn được thực hiện như sau:

* Xác định các mức độ coupling và cohesion
* Xác định các vi phạm nguyên lý SOLID
* Xác định các vấn đề về clean code

Việc refactor mã nguồn được thực hiện như sau:

* Tái cấu trúc các variable, method và class để đảm bảo clean code
* Tái cấu trúc những đoạn vi phạm nguyên lý SOLID
* Áp dụng các Design Pattern đã học vào hệ thống một cách có chọn lọc và phù hợp.

#### Kết quả dự kiến

Sau khi tái cấu trúc mã nguồn, codebase sẽ đảm bảo được các nguyên lý về SOLID, được áp dụng các Design Pattern phù hợp và đảm bảo clean code để dễ đọc dễ hiểu cho các lập trình viên sau này.

Từ đó, phần mềm sẽ có khả năng nâng cấp, mở rộng tốt hơn trong tương lai.

### Danh sách thuật ngữ

|  |  |  |
| --- | --- | --- |
| STT | Thuật ngữ | Giải thích |
| 1 | coupling | Mức độ liên kết giữa các class, các thành phần trong hệ thống |
| 2 | cohesion | Mức độ liên kết chi tiết trong module, class |
| 3 | Method | Phương thức của một lớp |
| 4 | SOLID | Nguyên tắc thiết kế nhằm làm cho các thiết kế hướng đối tượng trở nên dễ hiểu, linh hoạt và dễ bảo trì hơn |
| 5 | Refactor | Tái cấu trúc mã nguồn |
| 6 | Data Access Object (DAO) | Cung cấp giao diện trừu tượng cho một số loại cơ sở dữ liệu hoặc cơ chế lưu giữ khác. Bằng cách ánh xạ các cuộc gọi ứng dụng tới lớp kiên trì, DAO cung cấp các hoạt động dữ liệu mà không để lộ chi tiết cơ sở dữ liệu.. |
| 7 | Entity | Thực thể lưu trữ dữ liệu của hệ thống |
| 8 | Design Pattern | Mẫu thiết kế phần mềm - Những mô hình hoặc mẫu quy ước được phát triển để giải quyết các vấn đề chung trong thiết kế phần mềm. |
| 9 | Module | Các thành phần trong hệ thống phục vụ một mục đích chung |

### Danh sách tài liệu tham khảo

* Sách Head First Design Patterns: A Brain-Friendly Guide
* Sách Clean Code: A Handbook of Agile Software Craftsmanship
* Tài liệu, vở ghi trên lớp môn học Mẫu thiết kế phần mềm IT4536 - Trường Công nghệ Thông tin và Truyền thông - Đại học Bách Khoa Hà Nội.
* Các tài liệu online khác

## Đánh giá thiết kế cũ

### Nhận xét chung

Hệ thống hiện tại đã đảm bảo được luồng hoạt động chính. Tuy nhiên khi nhìn vào cách tổ chức hệ thống, ta sẽ có thể phát hiện ra nhiều vấn đề mà khi có các yêu cầu phát sinh hệ thống sẽ khó thích ứng: Các modules phụ thuộc trực tiếp vào nhau thay vì phụ thuộc ở mức trừu tượng. Điều này sẽ khiến khi thay đổi một module, các module có liên quan cũng phải được thay đổi theo. Thêm vào đó là các vấn đề khác như dư thừa dữ liệu, vấn đề clean code. Do đó việc tái cấu trúc và sửa đổi là cần thiết: Loại bỏ sự phụ thuộc trực tiếp mà thay vào đó là phụ thuộc trừu tượng, clean code, loại bỏ dữ liệu không cần thiết, … Từ đó hệ thống sẽ dễ bảo trì vào nâng cấp hơn

### Đánh giá các mức độ coupling và cohesion

Trong lĩnh vực thiết kế hệ thống phần mềm, mức độ coupling của một thiết kế được xác định bằng sự phụ thuộc, gắn kết giữa hai hay nhiều classes, modules. Một thiết kế được coi là low coupling khi một class, module này bị thay đổi, sẽ không gây nhiều ảnh hưởng tới các class, module khác mà nó liên kết tới. Ngược lại, một thiết kế được coi là high coupling khi các class, module bị kết dính quá chặt chẽ với nhau, một thay đổi của thành phần này gây ra nhiều sự thay đổi của các thành phần liên quan tới nó, dẫn tới việc hệ thống rất khó thay đổi và bảo trì.

Không giống với cohesion, mức độ cohesion của một class, module, package được xác định bằng độ liên kết chặt chẽ của các thành phần bên trong một class, module, package đó. Một thiết kế được coi là low cohesion khi module đó thực hiện quá nhiều công việc, mà không tập trung vào một công việc cụ thể nào, ví dụ như các package Utils trong thực tế thường được sử dụng để chứa nhiều hàm tiện ích của hệ thống mà có ít liên quan với nhau. Và ngược lại, một thiết kế được coi là high cohesion khi class, module, package chỉ tập trung vào việc thực hiện một nhiệm vụ, mục tiêu cụ thể và duy nhất.

Tổng kết lại, một thiết kế phần mềm tốt là một thiết kế có mức độ cohesion cao và coupling thấp (high cohesion và low coupling).

#### Mức độ coupling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Các mức độ về Coupling | Module | Mô tả | Lý do |
| 1 | Content coupling | entity\shipping\DeliveryInfo.java | access modifier là protected | Khi để access modifier là protected, các lớp con cùng package và lớp con có thể truy xuất data trực tiếp, vừa vi phạm content coupling, vừa vi phạm nguyên lý encapsulation. |
| 2 | Common coupling | views\screen\ViewsConfig.java | Các biến PERCENT\_VAT và REGULAR\_FONT là các biến dùng chúng cho hệ thống nhưng không được khai báo final | Khi khai báo biến static để dùng chung cho cả hệ thống, biến nên được khai báo là hằng số final để tránh việc bị thay đổi khi hệ thống vận hành |
| SessionInformation | Nhiều lớp có tham gia vào việc dùng và sửa đổi dữ liệu này | Việc nhiều lớp có thể thay đổi dữ liệu này khiến khó để quản lý trạng thái dữ liệu |
| 3 | Control coupling | Không có | Không có | Không có |
| 4 | Stamp coupling | entity\cart\Cart.java | Tham số của method checkMediaInCart đang có kiểu dữ liệu là Media | implement của hàm chỉ cần duy nhất trường id của Media, dẫn đến dư thừa dữ liệu |
| views\screen\  intro\  IntroScreenHandler.java | Tham số của method setupData đang có kiểu dữ liệu là Object | Hàm chưa được implement dẫn đến việc truyền dữ liệu vào hàm là dư thừa |
| views\screen\  home\  LoginScreenHandler.java | method backToHomeScreen đang có kiểu dữ liệu là MouseEvent | implement của hàm không cần tới tham số này -> dư thừa |
| views\screen\popup\PopupScreen.java | Tham số truyền vào là một Object | Hàm chưa được implement dẫn đến việc truyền dữ liệu vào hàm là dư thừa |
| controller\PlaceOrderController.java | tham số truyền vào dạng HashMap | Hàm có thể không dùng hết các trường có trong HashMap |
| dao\media\MediaDAO.java | Tham số truyền vào có tbname dạng String | Trong implement của hàm không dùng tới trường này dẫn đến dư thừa dữ liệu |
| entity\cart\CartItem.java | CartItem có tham số truyền vào là Cart | implement của hàm không cần tới tham số này -> dư thừa |
| entity\shipping\DeliveryInfo.java | CalculateShippingFee có tham số truyền vào là Order | implement của hàm không cần tới tham số này -> dư thừa |
| 5 | Data coupling | Không có | Không có | Không có |

#### Mức độ cohesion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Các mức độ về Cohesion | Module | Mô tả | Lý do |
| 1 | Co-incidental cohesion | utils\Utils.java | Biến DATE\_FORMATTER và LOGGER không được sử dụng trong class | 2 biến này được khởi tạo nhưng không được sử dụng trong class |
| views\screen\BaseScreenHandler.java | Biến LOGGER được khởi tạo nhưng không được sử dụng.  Phương thức forward() được khởi tạo nhưng không được sử dụng | Tương tự như trên |
| views\screen\home\HomeScreenHandler.java | Biến LOGGER được khởi tạo nhưng không được sử dụng. | Tương tự như trên |
| views\screen\shipping | Method notifyError của class được khai báo nhưng không được sử dụng và cũng không có phần cài đặt. | Tương tự như trên |
| package dao | Các lớp InvoiceDAO , OrderDAO và OrderItemDAO được khai báo nhưng không được implement |  |
| controller\AuthenticationController.java | md5() không phục vụ mục đích của lớp | hàm md5 là hàm sinh ra digest, không phục vụ mục đích của lớp là xác thực người dùng, nên tách thành 1 hàm helper riêng |
| controller\PaymentController.java | getExpirationDate không phục vụ mục đích của class | Nên tách thành 1 hàm helper riềng |
| subsystem\interbank\InterbankPayloadConverter.java | tương tự trên, getToday() không cùng mục đích với class cha | Tách thành 1 helper method riêng |
| 2 | Logical cohesion | utils\ApplicationProgrammingInterface.java | các methods post và get liên quan logic với nhau | các methods post và get trong module chỉ liên kết với nhau về logic gọi API ngoài hệ thống chứ không liên kết về chức năng |
| 3 | Temporal cohesion | Không có | Không có | Không có |
| 4 | Procedure cohesion | controller\PlaceOrderController.java | Các methods validateDeliveryInfo, validatePhoneNumber, validateName và validateAddress đang có mức độ procedure cohesion | Các methods có cùng liên kết về chưacs năng validate dữ liệu, nhưng đây không phải mục đích của class cha. Nên tách thành hàm helper riêng |
| 5 | Communicational cohesion | Không có | Không có | Không có |
| 6 | Sequential cohesion | Không có | Không có | Không có |
| 7 | Functional cohesion | Không có | Không có | Không có |

### Đánh giá việc tuân theo SOLID

#### Single Responsibility Principle (SRP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | entity.shipping.  DeliveryInfo | Class DeliveryInfo đang có chứa một method dùng để tính toán chi phí vận chuyển. | Class DeliveryInfo là một entity vậy nên việc tính toán chi phí vận chuyển không phải là nhiệm vụ của nó. Hãy tách hàm tính toán chi phí vận chuyển ra một class riêng và truyền vào DeliveryInfo. |
| 2 | controller.  AuthenticationController | Class AuthenticationController vi phạm SRP do có nhiều hơn một vai trò (Cụ thể là hàm sinh mã băm MD5) | Class AuthenticationController có nhiệm vụ xử lý logic các yêu cầu authenticate của người dùng, việc sinh mã băm không phải là nhiệm vụ của nó. Hãy tách hàm sinh mã băm ra một class riêng và truyền vào AuthenticationController. |
| 3 | controller.  PlaceOrderController | Class PlaceOrderController vi phạm SRP do có nhiều hơn một vai trò (Cụ thể là vừa validate trường, vừa đặt hàng, tạo hóa đơn). | Class PlaceOrderController có nhiệm vụ xử lý yêu cầu đặt hàng, tạo hóa đơn của người dùng, việc validate không phải nhiệm vụ của nó.  Hãy tách hàm validate ra một class riêng và truyền vào PlaceOrderController. |

#### Open Closed Principle (OCP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | * entity.payment.   PaymentTransaction   * entity.payment.   CreditCard   * controller.   PaymentController | Các class bên đang có một phụ thuộc trực tiếp vào CreditCard | Có thể trong tương lai có những yêu mới phát sinh về payment mà CreditCard hiện tại không đáp ứng được, dẫn tới việc sẽ phải sửa đổi CreditCard, việc này chứng tỏ thiết kế hiện tại đang vi phạm Open Closed Principle. |
| 2 | controller.  PlaceOrderController | Class này đang phụ thuộc trực tiếp vào class DistanceCalculator | Do class đang phụ thuộc vào một thư viện tính toán khoảng cách bên ngoài DistanceCalculator. Việc này có nguy cơ nếu muốn thêm các cách tính khoảng cách khác thì bắt buộc phải thay đổi code dẫn đến vi phạm OCP. |

#### Liskov Substitution Principle (LSP)

Hệ thống hiện tại không vi phạm Liskov Substitution Principle (LSP).

#### Interface Segregation Principle (ISP)

Hệ thống hiện tại không vi phạm Interface Segregation Principle (ISP).

#### Dependency Inversion Principle (DIP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | * controller.   PaymentController   * subsystem * entity.payment.   PaymentTransaction | Các class và interface trong PaymentController, PaymentTransaction và trong gần như toàn bộ của subsystem hiện tại đang phụ thuộc trực tiếp vào class CreditCard. | Trong tương lai, hệ thống khả năng cao sẽ mở rộng các phương thức giao dịch, thanh toán mới (Payment). Ví dụ như Cash, EWallet, Bitcoin,... Do vậy việc sử dụng và phụ thuộc vào CreditCard khiến cho việc mở rộng trở nên rất khó khăn. Chúng ta nên tạo ra các abstraction (interface hoặc abstract class) để định nghĩa giao diện và chức năng chung của các thành phần. |

### Các vấn đề về Clean Code

#### Clean name

Với mã nguồn ban đầu các vấn đề clean name hầu như code đều đã sạch, tuy nhiên vẫn có một vài điểm cần thay đổi tên cho phù hợp với ngữ cảnh và chức năng mà hàm/biến đó thể hiện. Dưới đây là các vấn đề về clean code mà nhóm tìm ra:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Vị trí có tên chưa clear | Đề xuất sửa đổi |
| 1 | controller.PaymentController | Method getExpirationDate có tên biến strs | Sửa tên biến strs thành dateStrs để thể hiện danh sách date |
| 2 | controller.PlaceOrderController | Method validatePhoneNumber  có tên hàng số 10 | Chuyển hàng số thành biến final có ý nghĩa lengthPhoneNumber |
| 3 | utils.ApplicationProgrammingInterface | Method get có tên biến in | tên "bufferedReader" cho biết rõ rằng biến đó là một thể hiện của lớp BufferedReader và được sử dụng để đọc dữ liệu từ một luồng đầu vào. |
| 4 | views.screen.home.HomeScreenHandler | Method setupData có tên biến medium thể hiện chưa rõ ý nghĩa mà nó thực hiện | Đổi tên biến medium thành mediaList thể hiện danh sách các đối tượng media |

#### Clean function/method

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Vị trí có tên chưa clear | Đề xuất sửa đổi |
| 1 | controller.AuthenticationController | Phương thức md5 | Thay đổi tên phương thức md5 thành genDigestByMd5 để thể hiện nhiệm vụ của hàm này là sinh một message-digest bằng cách dùng thuật toán md5. |
| 2 | dao.media.MediaDao | Tham số id của phương thức getCurrentQuantity được truyền vào nhưng không sử dụng | Sửa đổi câu truy vấn cơ sở dữ liệu trở thành “SELECT \* FROM MEDIA WHERE ID = ?” và truyền tham số id của method vào. |
| 3 | views.screen.home.HomeScreenHandler | Phương thức update có hai vị trí log error dùng hard-code | Tạo các biến dùng chung addMediaFail, loginFail |
| 4 | controller.ViewCartController | Phương thức getCartSubtotal | Bỏ việc đặt tên cho biến thay vào đó return luôn giá tra do hàm chỉ có 1 phép tính toán, chuyển về dạng inline-function |
| 5 | views.screen.intro.IntroScreenHandler | Phương thức setupFunctionality dùng hard-code cho đường dẫn dẫn logo đến khó thay đổi sau này | Chuyển giá trị đường dẫn tới logo của ứng dụng vào class ViewsConfig đặt tên là LOGO\_PATH và gán vào phương thức setupFunctionality |

#### Clean class

Với mã nguồn ban đầu các vấn đề về clean class đáp ứng đầy đủ yêu cầu ở hầu hết các lớp. Các vấn đề còn tồn đọng mà nhóm tìm ra được như: Biến khai báo nhưng không được sử dụng, hàm không dùng hay class chỉ được viết ra nhưng không hề có xử lý logic. Cụ thể như sau

|  |  |  |
| --- | --- | --- |
| # | Vấn đề tồn tại | Danh sách class |
| 1 | Có các attribute cần được đóng gói | entity.media.CD  entity.media.Book  entity.media.DVD |
| 2 | Có các attribute được khai báo nhưng không được sử dụng | utils.Utils  utils.ApplicationProgrammingInterface |
| 3 | Có các method được khai báo nhưng không được sử dụng | entity.invoice.Invoice  dao.media.MediaDAO  views.screen.popup.PopupScreen  views.screen.shipping.ShippingScreenHandler |
| 4 | Các lớp không được sử dụng | dao.order.OrderDAO  dao.order.OrderItemDAO  dao.invoice.InvoiceDAO |
| 5 | Class có các đoạn log có cùng giá trị nhưng đang hard-code thay vì tạo biến để dùng chung | utils.MyMap |

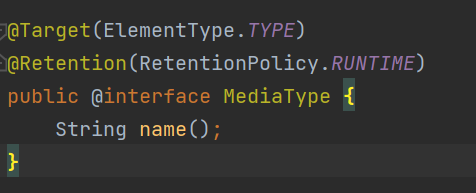
## Đề xuất cải tiến

### Đề xuất khi phát sinh thêm một loại Media mới: AudioBook.

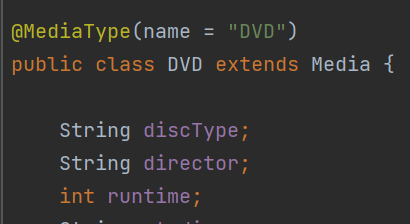
Như đã đánh giá ở trong phần SOLID, việc chỉnh sửa, thêm bớt các loại Media hiện tại đang vi phạm nguyên lý Open Closed Principle (OCP). Do vậy để đảm bảo tuân thủ SOLID, phục vụ cho việc mở rộng trong tương lai. Chúng ta cần cải tiến hệ thống.

Trong trường hợp này, chúng ta có thể sử dụng Reflections để lấy các class hiện đang có trong “entity.media” và tiến hành lấy tên của từng class để đưa vào HomeScreenHandler tiến hành xử lý ra giao diện.

Tuy nhiên không phải lúc nào tên class cũng thỏa mãn yêu cầu khi hiển thị ra giao diện, vậy nên chúng ta cần xử lý việc thay đổi tên này. Nhờ vào Java Annotation, chúng ta có thể tạo ra một annotation riêng của mình để phục vụ cho việc đánh dấu từng loại Media. Ở đây, ta đặt tên cho annotation mới là MediaType.

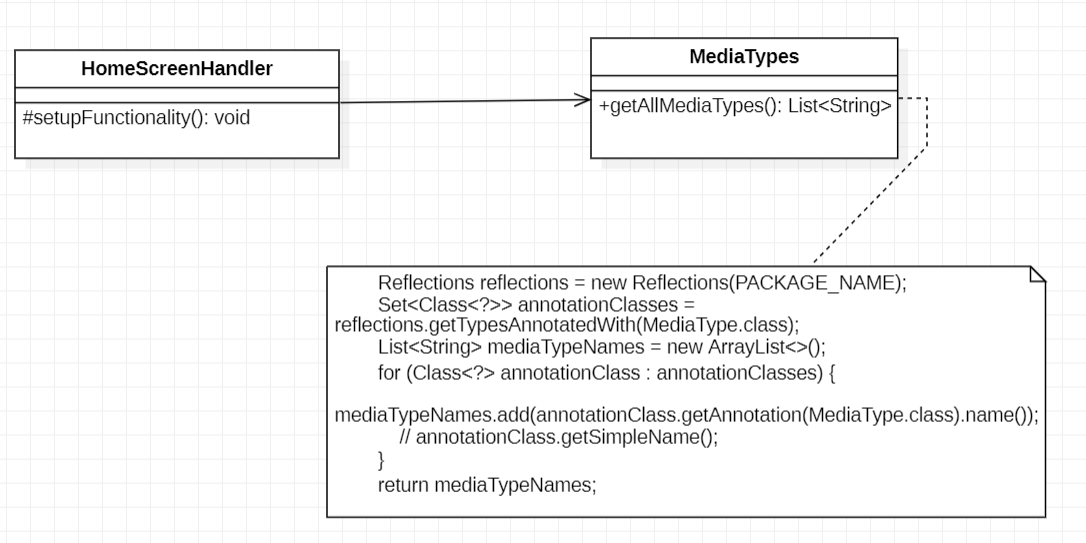


**Hình 3:** Cấu trúc annotation MediaType

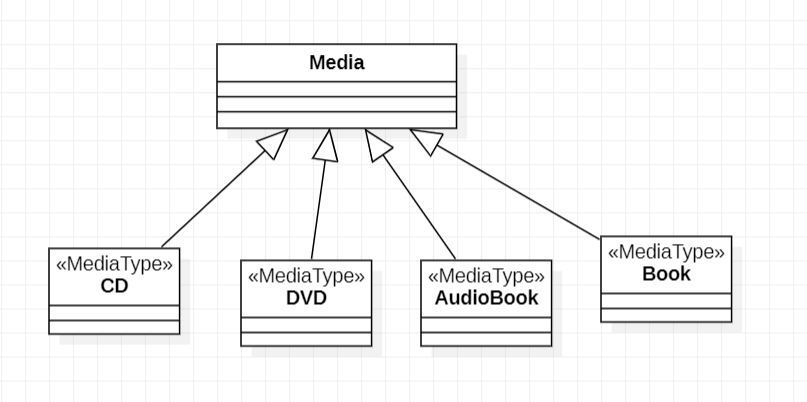
****

**Hình 4:** Sử dụng annotation vào các loại Media

Ở đây, ElementType.TYPE đảm bảo rằng Annotation của bạn chỉ có thể được áp dụng cho lớp, giao diện, enum hoặc annotation. Trong tương lai, nếu dự định sử dụng cho các phương thức trong class, có thể thêm ElementType.FIELD vào.



**Hình 5:** Sử dụng Java Reflections trong MediaTypes để xử lý HomeScreenHandler

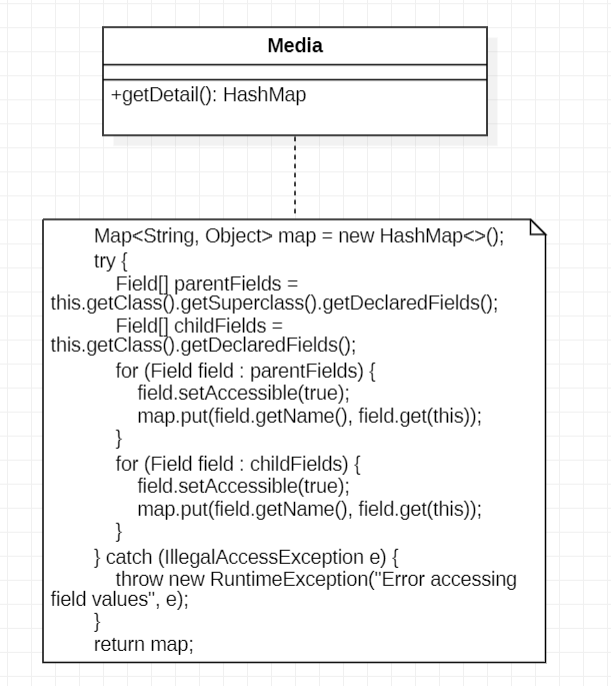


**Hình 6:** Tạo Annotation MediaType phục vụ xử lý trong MediaTypes

Sau khi cải tiến code. Ta có thể dễ dàng thêm các loại Media mới. Ví dụ như AudioBook theo yêu cầu cải tiến. Bên cạnh đó cần tạo thêm AudioBookDAO kế thừa MediaDAO để tiến hành truy xuất dữ liệu từ database tương ứng.

### Đề xuất khi thêm màn hình: Xem chi tiết sản phẩm

Với yêu cầu thêm việc xem chi tiết sản phẩm. Việc chúng ta cần làm đơn giản là thêm method để lấy thông tin chi tiết ở trong Media. Từ đó các lớp con cũng được kế thừa lại và sử dụng method đó. Trong getDetails, sử dụng Reflection Field cho việc lấy các trường được khai báo trong class. Việc còn lại là thực hiện thay đổi ở giao diện, thêm button “Xem chi tiết” và thay đổi ở ScreenHandler để có thể xử lý logic việc bấm vào nút đó.



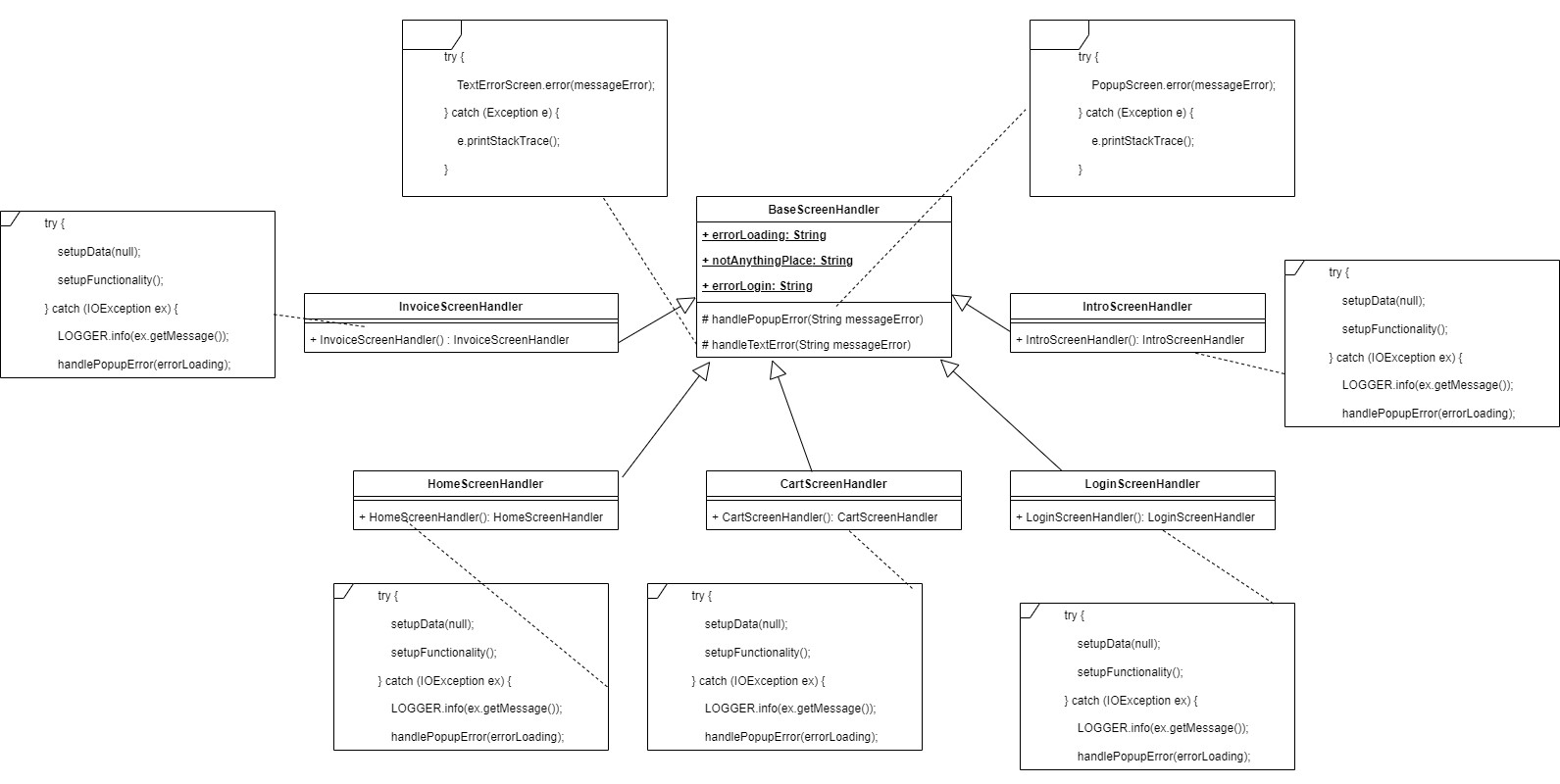
**Hình 7:** Method getDetails trong Media để giúp lấy thông tin chi tiết sản phẩm

Ngoài ra, tương tự như ở giải pháp trên, nếu muốn hiển thị tên các thuộc tính tốt hơn, có thể tạo annotation mới “AttributeName” và gán cho từng thuộc tính.

### Đề xuất khi thay đổi yêu cầu khi load giao diện

Khi mỗi lần xảy ra lỗi IOException, các method sẽ gọi tới một hàm static PopupScreen.error(“Error when loading resources.”), việc thiết kế như vậy đang sử dụng hard-code điều này dẫn đến khi cần thay đổi phải trỏ thẳng vào class cần thay đổi. Hơn nữa mỗi class lại có thể sẽ có một lỗi khác nhau. Việc kiểm soát và bảo trì trở nên khó khăn. Nhóm đề xuất thay vì viết hàm báo lỗi ở các lớp con, ta chuyển việc code này lên lớp cấp cao hơn đó là lớp BaseScreenHandler. Ở lớp này ta thêm phương thức handlePopupError cho thông báo lỗi bằng cách popup ra bên màn hình, và phương thức handleTextError cho thông báo lỗi ở phía trên cùng góc bên phải. Việc thay đổi lỗi ta xây dựng các biến static final dùng chung các hàm chỉ cần truyền biến là có thể thay đổi kiểu báo lỗi.

Biểu đồ lớp thiết kế sau khi bổ sung như sau:



**Hình 8** Giải quyết vấn đề xử lý IOException khi load màn hình

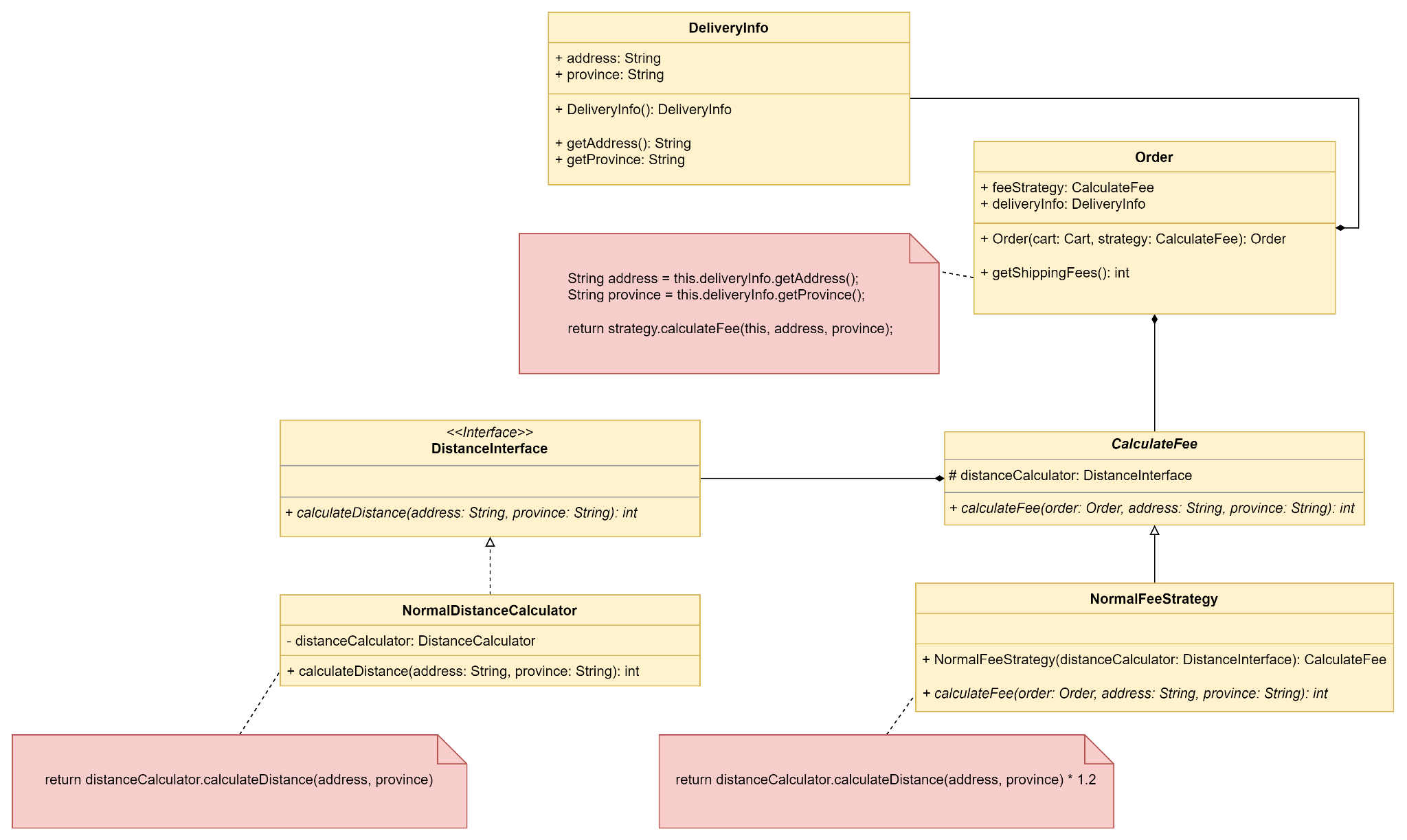
### Cải tiến vấn đề thay đổi phương thức tính khoảng cách sử dụng thư viện mới và thay đổi công thức tính phí vận chuyển

Class DeliveryInfo hiện tại đang có một thuộc tính có kiểu dữ liệu là DistanceCalculator (thuộc thư viện Distance-API). Nếu như ta muốn thay đổi thư viện tính khoảng cách mới (Alt-Distance-API), ta phải sửa class DeliveryInfo.

Việc tính phí và tính khoảng cách đang được thực hiện ở class DeliveryInfo. Khi có yêu cầu mới là tính phí dựa trên khối lượng, độ cồng kềnh và khoảng cách sẽ có vấn đề phát sinh, vì các thông tin này nằm ở trong class Order. Ta nên chuyển việc tính phí và khoảng cách cho class Order sẽ dễ dàng hơn

Từ đó ta sẽ thiết kế lại, bằng cách sử dụng Adapter Pattern, tạo một interface cho phương thức tính khoảng cách, và class nào muốn sử dụng sẽ thực hiện việc implement interface này. Việc tách thành interface riêng sẽ giúp chúng ta khi có yêu cầu mới sẽ chỉ cần tạo một class implement interface này, các class đã implement từ trước sẽ không bị ảnh hưởng

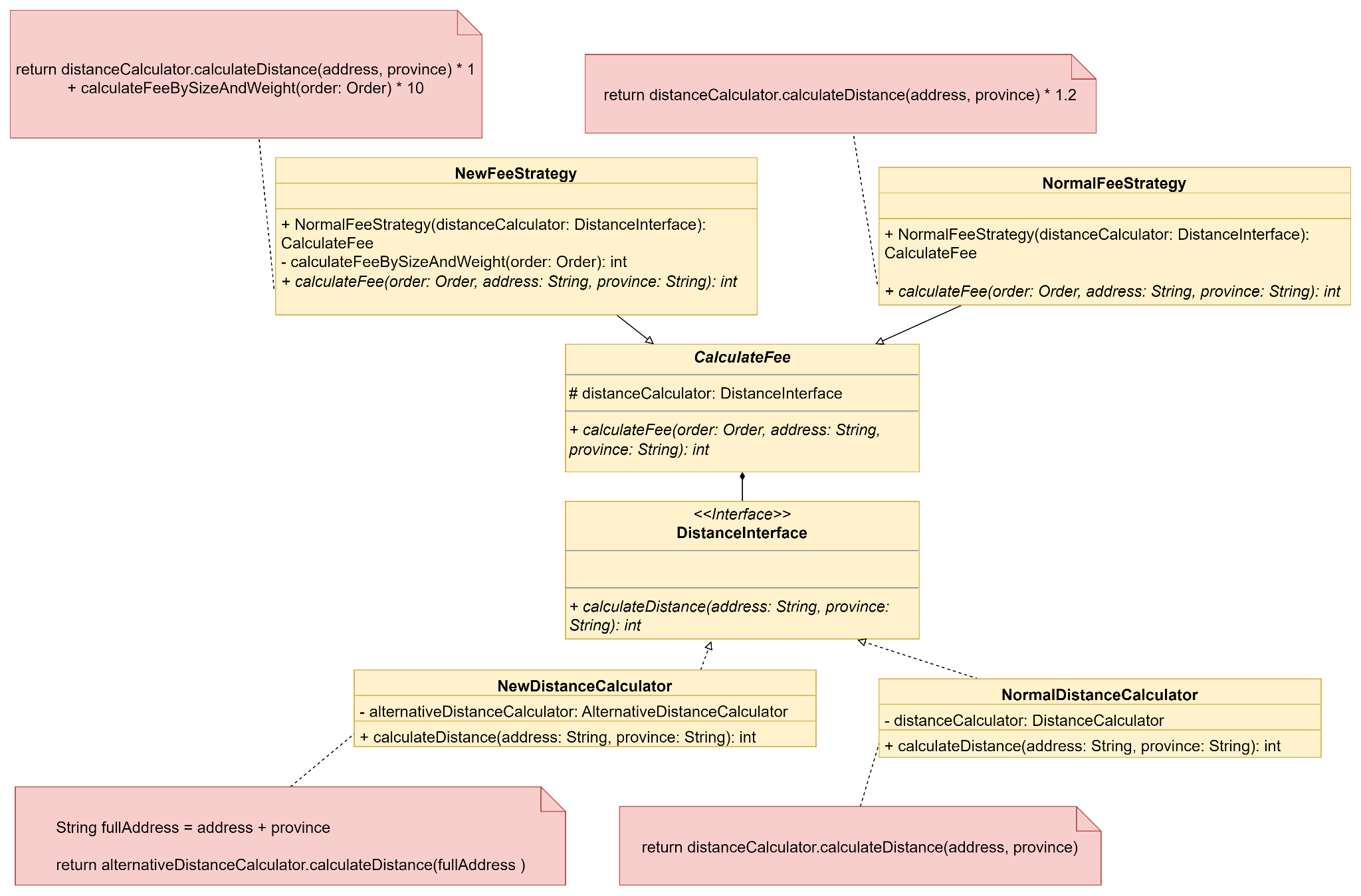
Vấn đề có yêu cầu thay đổi công thức tính khoảng cách, trước tiên sẽ chuyển việc tính phí ở DeliveryInfo sang class Order, sau đó áp dụng Strategy Pattern, tạo một abstract class ***CalculateFee***, khi cần thay đổi ta sẽ tạo một class extends class abstract này



**Hình 9** Giải quyết vấn đề thay đổi cách tính chi phí và thư viện tính khoảng cách

Hiện tại, để tính chi phí chỉ cần biết tọa độ điểm nhận, nhưng tham số của hàm calculate trong interface ShippingFeeCalculatorStrategy được thiết kế với tham số là Order. Chúng ta chấp nhận mức vi phạm Stamp Coupling ở hệ thống hiện tại sau khi cải tiến, do trong công thức tính chi phí mới phải cần các thông tin khác trong Order để tính toán, nếu chỉ sử dụng hai tham số là điểm giao và nhận thì sẽ không tái sử dụng được interface này trong tương lai.

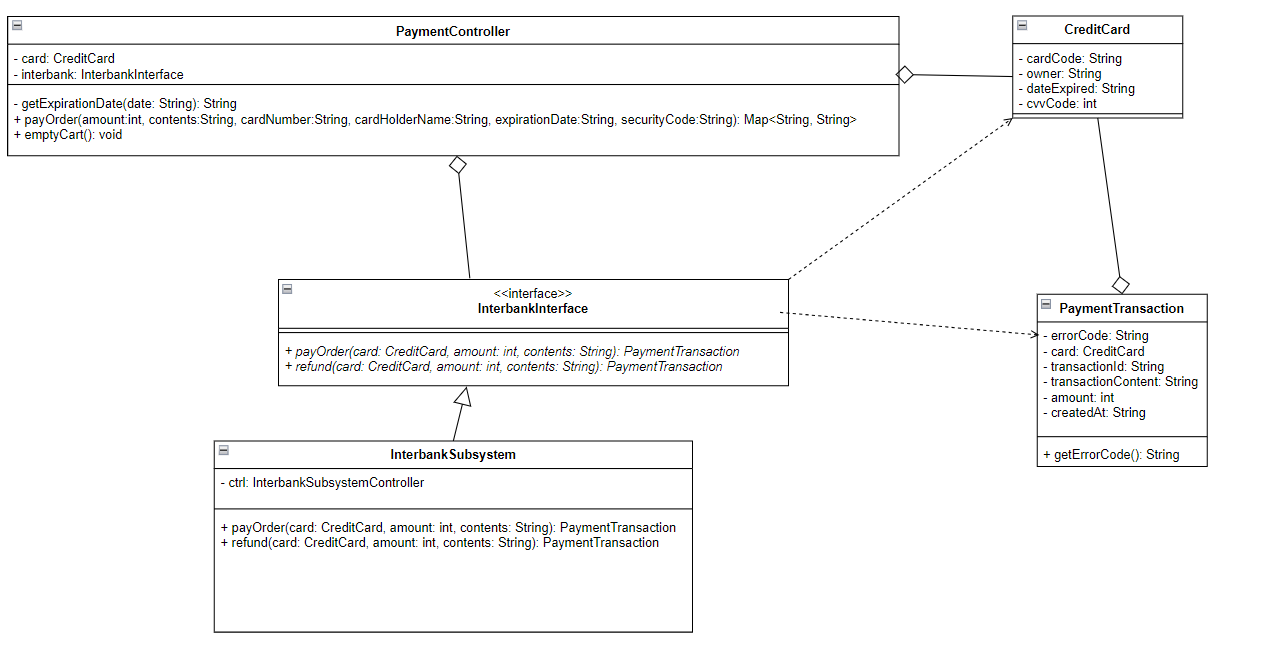
Khi yêu cầu phát sinh xảy ra, chúng ta chỉ cần cài đặt thêm class OrderDistanceBaseStrategy và class AltDistanceCbalculator tương ứng với thay đổi về việc thay đổi công thức tính toán và thay đổi thư viện tính khoảng cách. Thiết kế khi xảy ra cả hai thay đổi sẽ như sau:



**Hình 10** Thiết kế khi xảy ra thay đổi thư viện tính khoảng cách và công thức tính phí

### Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)

Trong thiết kế cũ, lớp PaymentController có nhiệm vụ thực hiện việc xử lý thanh toán, và nó phụ thuộc trực tiếp vào thuộc tính card của lớp CreditCart cũng như ta đang cố định một phương thức thanh toán trong này. Thêm vào đó các lớp khác như ở trong pakage subsystem, class PaymentTransaction cũng phụ thuộc trực tiếp vào CreditCard. Như vậy rõ ràng trong tương lai nếu có thêm một loại thẻ mới, một phương thức thanh toán mới thì ta sẽ sửa trực tiếp code trong những class trên. Như vậy đang có hai vấn đề liên quan tới loại card và phương thức thanh toán. Dưới đây là thiết kế cũ:

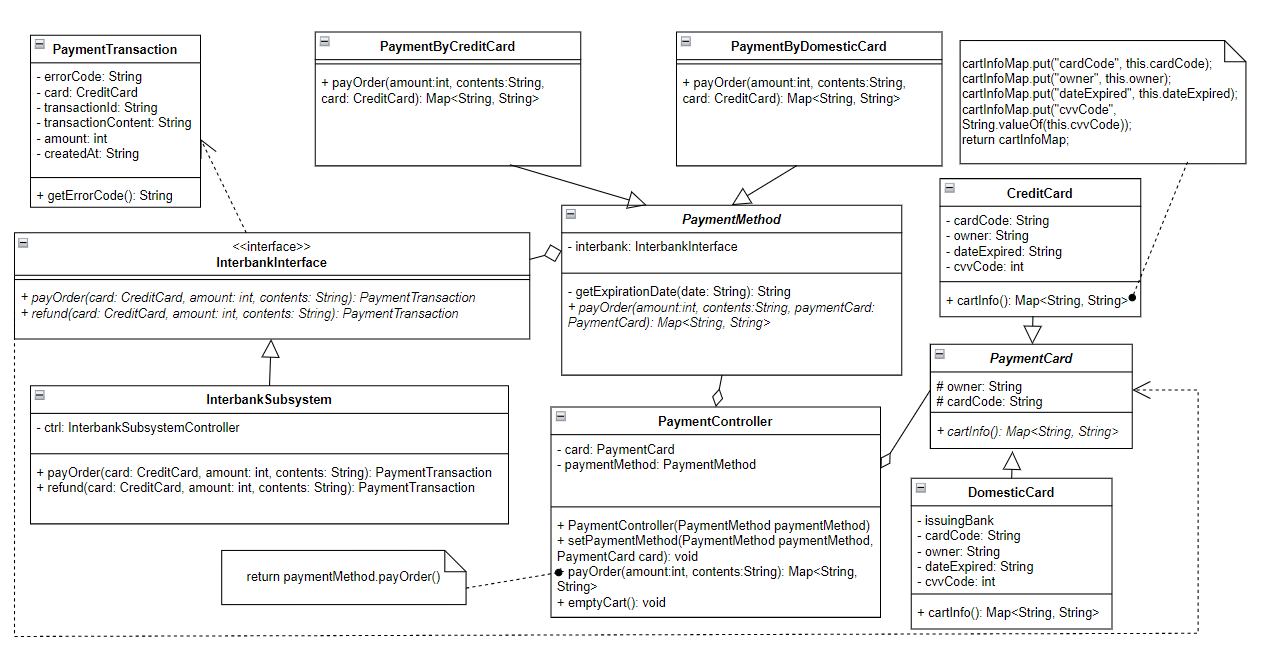


**Hình 11** Mã nguồn phải sửa code khi thay đổi phương thức thanh toán và loại thẻ thanh toán

Để khắc phục điều này ta làm như sau:

+ Đầu tiên với việc thêm các loại thẻ mới ta sử dụng strategy pattern. Ta sẽ tạo một lớp abstract PaymentCard chứa thông tin chung của tất các loại thẻ. Thẻ hiện tại và thẻ trong tươi lai ta chỉ việc extends lớp abstract này là xong. Các lớp trước đó có sử dụng trực tiếp CreditCard thì ta chỉ việc sửa đổi nó thành PaymentCard.

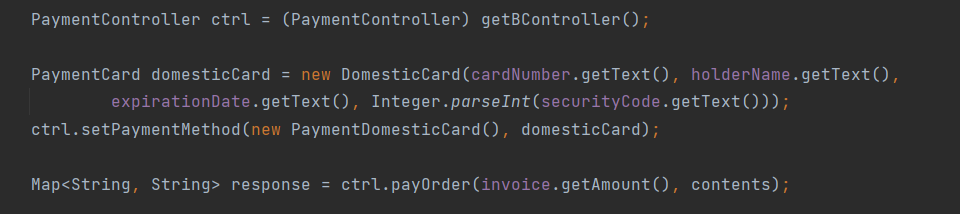
+ Đối với việc thêm phương thức thanh toán mới, ta cũng sử dụng strategy pattern. Ta sẽ chuyển toàn bộ việc thanh toán sang lớp abstract PaymentMethod. Ta sẽ chuyển interface InterbankInterface và phương thức getExpirationDate của class PaymentController sang PaymentMethod. Từ đây những class phương thức mới ta chỉ việc extends class này.



**Hình 12** Giải quyết vấn đề phát sinh phương thức thanh toán mới

=> từ đây ta thầy do ban đầu việc sử dụng thư viện ngoài là interbank cũng dùng Strategy nên việc chuyển đổi API thanh toán cũng trở nên dễ dàng.

Khi đó class PaymentScreenHandler sẽ sử dụng lớp PaymentController như sau:



**Hình 13** Sử dụng mã nguồn mới cho việc thay đổi phương thức thanh toán trong lớp ScreenHandler

## Tổng kết

### Kết quả tổng quan

Sau khi hoàn thành bài tập lớn lần này, dự án cửa hàng online AIMS đã có thiết kế tốt hơn. Mã nguồn về cơ bản đã đáp ứng được các nguyên lý SOLID, từ đây trong tương lai nếu có những yêu cầu mới hay cần sửa đổi mã nguồn thì những việc phát sinh đó sẽ được xử lý dễ dàng hơn trước. Thêm vào đó các vấn đề clean code như clean name, clean method, clean class cũng đã được khắc phục. Do đã sửa đổi theo nguyên lý SOLID mà các vấn đề coupling cohesion trước đó cũng không còn nữa.

Qua việc làm bài tập lớn lần này đã giúp các thành viên trong nhóm cải thiện kiến thức một cách rõ rệt, mỗi thành viên đều đã biết vận dụng kiến thức trong việc tái cấu trúc mã nguồn, thiết kế mã nguồn làm sao cho hợp lý và hiệu quả nhất.

### Các vấn đề còn tồn đọng

Tuy đã có sửa đổi và bổ sung thêm các tính năng, nhưng việc thiếu sót vẫn có thể xảy ra. Bên cạnh các sai sót chưa nhìn nhận được, nhóm cũng đã nhận thấy các vấn đề như: Khi tái cấu trúc để đáp ứng yêu cầu đổi tình phí vận chuyển, nhóm đã xây dựng hàm tính phí theo khối lượng có tham số đầu vào là order, nhưng trong thân hàm không sử dụng tới. Việc này dẫn tới vấn đề bị Stamp Coupling, do một Order có các thuộc tính không cần thiết trong việc tính toán chi phí vận chuyển, ví dụ như thuế (tax), phụ phí (subtotal). Tuy nhiên ở đây nhóm vẫn chấp nhận bị coupling vì các dữ kiện như khối lượng, độ cồng kềnh là các thuộc tính của order