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:
Nguyễn Văn Mạnh - 20204668
Lang Thành Long - 20194098

Hà Nội, tháng 6 năm 2024

MỤC LỤC

Danh mục hình ảnh

Danh mục bảng

1. Tổng quan

1.1. Mục tiêu

Tài liệu này được soạn thảo phục vụ cho 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 bao gồm lập trình viên, những người có nền tảng về lập trình hướng đối tượng, các nhà thiết kế và phát triển hệ thống phần mềm, cũng như những ai muốn tìm hiểu để viết hướng dẫn nghiệp vụ. Đây là thành quả của nhóm tác giả, nhằm tổng hợp quá trình thực hiện và kết quả của việc tái cấu trúc mã nguồn dựa trên một hệ thống phần mềm có sẵn do giảng viên cung cấp. Nhóm đã áp dụng các kiến thức về Cohesion, Coupling, nguyên tắc SOLID và các Design Pattern phổ biến được giảng dạy trong khóa học để phân tích và cải tiến thiết kế hệ thống hiện tại.

1.2. Phạm vi

1.2.1. Mô tả khái quát phần mềm

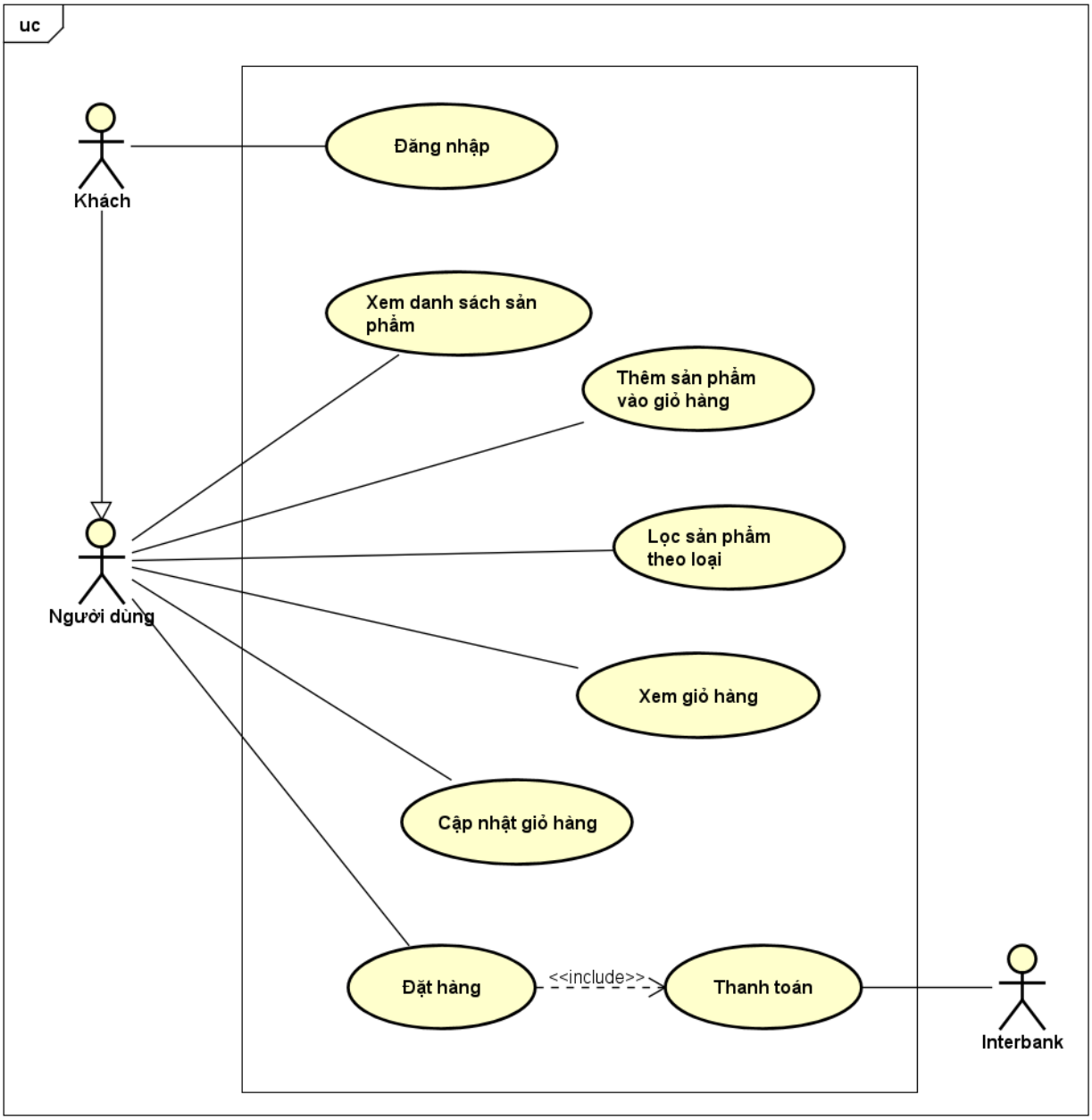
Phần mềm này là một cửa hàng trực tuyến cho phép người dùng đặt mua các sản phẩm đa phương tiện như sách, đĩa CD, và đĩa DVD từ cửa hàng. Sau khi đặt hàng, người dùng sẽ thanh toán trực tuyến qua thẻ tín dụng, và sản phẩm sẽ được giao đến địa chỉ do khách hàng cung cấp.

Mặc dù phần mềm hiện tại đã đáp ứng được các chức năng cơ bản, nhưng nó chưa tuân thủ đầy đủ các nguyên tắc SOLID. Do đó, phần mềm sẽ gặp khó khăn trong việc đáp ứng các yêu cầu thay đổi trong tương lai. Vì vậy, việc tái cấu trúc mã nguồn là cần thiết để cải thiện tính linh hoạt và khả năng bảo trì của phần mềm.

1.2.2. Các chức năng chính của phần mềm

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

- Xem danh sách các sản phẩm hiện có trong cửa hàng
- Thêm các sản phẩm vào giỏ hàng
- Điền thông tin thanh toán đơn hàng, xác định địa chỉ giao hàng và hướng dẫn giao hàng, thanh toán trực tuyến



Hình 1: Biểu đồ usecase tổng quan

Phần mềm có các tác nhân: Khách, Người dùng và InterBank. Khách, Người dùng đều có chung các hành vi nghiệp vụ khi tương tác với phần mềm, cụ thể là Khách dù không đăng nhập vẫn có thể thực hiện việc xem hàng, thêm vào giỏ và đặt hàng.

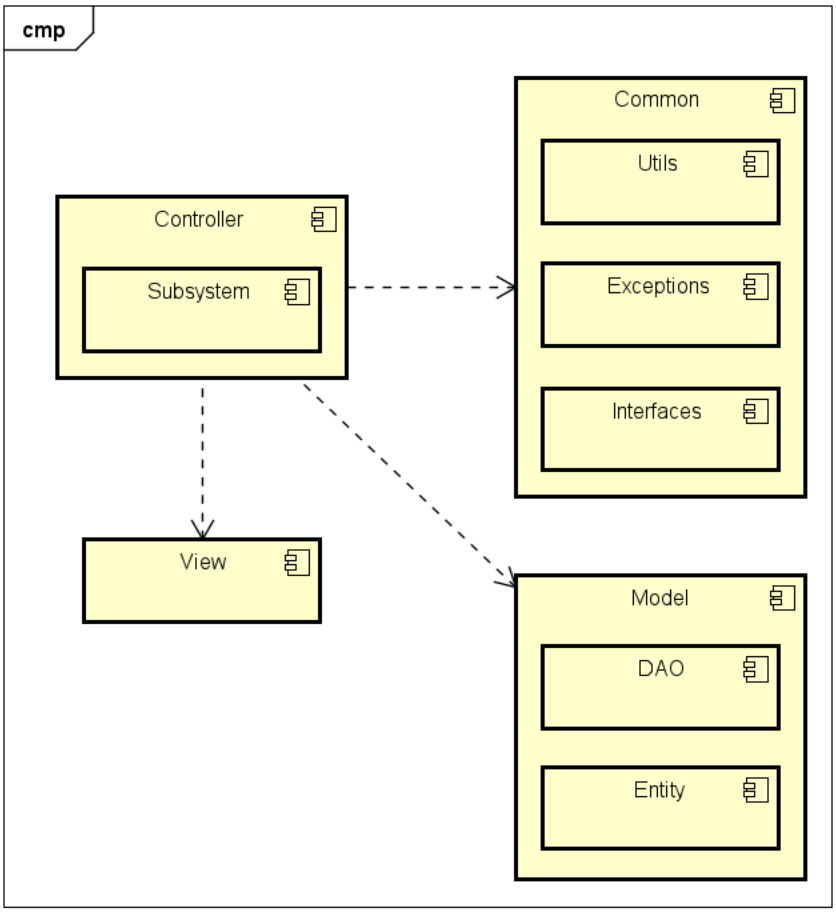
Người dùng, Khách có thể thực hiện việc Xem danh sách sản phẩm, Thêm sản phẩm vào giỏ hàng, Cập nhật giỏ hàng, Đặt hàng. Trong khi thực hiện đặt hàng, sẽ có nghiệp vụ Thanh toán. Nghiệp

vụ này có sự tham gia của tác nhân InterBank, là một tác nhân hệ thống, đại diện cho hệ thống thanh toán trực tuyến bằng thẻ.

1.2.3. Cấu trúc mã nguồn

Mã nguồn của phần mềm được thiết kế theo mô hình MVC rõ ràng, bao gồm ba tầng: Model, View và Controller. Trong mô hình MVC, tầng View chịu trách nhiệm về logic giao diện người dùng, bao gồm việc hiển thị các màn hình, thu thập dữ liệu từ người dùng, chuyển dữ liệu cho tầng Controller và điều hướng giữa các màn hình. Tầng Controller xử lý logic luồng dữ liệu, bao gồm xác thực thông tin người dùng và tương tác với các hệ thống con để thực hiện thanh toán đơn hàng, bao gồm cả các hệ thống con nội bộ. Tầng Model chịu trách nhiệm lưu trữ dữ liệu hệ thống và bao gồm hai thành phần chính: Data Access Object và Entity. Ngoài ra, hệ thống còn có một thành phần chung gọi là Common, chứa các Interfaces, Exceptions, và Utils, được sử dụng trên 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

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

Việc tái cấu trúc mã nguồn phần mềm nhằm mục đích giúp phần mềm có khả năng thích ứng với các thay đổi trong tương lai và duy trì mã nguồn "Sạch" để lập trình viên dễ dàng làm việc. Quá trình này tuân thủ các nguyên tắc SOLID, đặc biệt là Nguyên lý Đóng-mở (OCP) và Nguyên lý Đảo ngược Phụ thuộc (DIP). Tái cấu trúc phần mềm để tuân thủ SOLID có thể bao gồm việc phân chia, tái cấu trúc và tái thiết kế các module, lớp và giao diện. Điều này đòi hỏi một quá trình phân tích kỹ lưỡng, đánh giá và thực hiện các thay đổi một cách cẩn thận để đảm bảo tính chính xác và hiệu quả của hệ thống. Trong quá trình tái cấu trúc, cần đảm bảo không thay đổi các chức năng và phi chức năng của hệ thống ban đầu.

1.2.5. 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.

1.2.6. 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.

1.3. 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

Bảng 1: Danh sách thuật ngữ

1.4. Danh sách tài liệu tham khảo

- 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 trực tuyến khác trên internet.

2. Đánh giá thiết kế cũ

2.1. Nhận xét chung

Hệ thống hiện tại đã đảm bảo được các chức năng cơ bản. Tuy nhiên, khi xem xét cách tổ chức hệ thống, có thể nhận thấy nhiều vấn đề tiềm ẩn khi có yêu cầu mới phát sinh, khiến hệ thống khó thích ứng. Các module phụ thuộc trực tiếp vào nhau thay vì phụ thuộc vào các mức trừu

tượng, dẫn đến việc khi thay đổi một module, các module liên quan cũng cần được thay đổi. Ngoài ra, còn tồn tại các vấn đề khác như dư thừa dữ liệu và thiếu sự tuân thủ các nguyên tắc clean code. Do đó, việc tái cấu trúc và sửa đổi là cần thiết để:

- Loại bỏ sự phụ thuộc trực tiếp, thay vào đó sử dụng sự phụ thuộc trừu tượng.
- Tuân thủ các nguyên tắc clean code.
- Loại bỏ các dữ liệu không cần thiết.

Những cải tiến này sẽ giúp hệ thống dễ dàng bảo trì và nâng cấp trong tương lai.

2.2. 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 mức độ phụ thuộc giữa hai hay nhiều lớp (classes) hoặc module. Thiết kế có coupling thấp (low coupling) khi một lớp hoặc module có thể thay đổi mà không gây ảnh hưởng lớn đến các lớp hoặc module khác mà nó liên kết. Ngược lại, thiết kế có coupling cao (high coupling) khi các lớp hoặc module bị ràng buộc chặt chẽ với nhau, dẫn đến việc một thay đổi nhỏ trong một thành phần có thể yêu cầu thay đổi ở nhiều thành phần khác, làm cho hệ thống khó thay đổi và bảo trì.

Khác với coupling, mức độ cohesion được xác định bởi mức độ liên kết chặt chẽ giữa các thành phần bên trong một lớp, module hoặc package. Thiết kế có cohesion thấp (low cohesion) khi một module thực hiện quá nhiều công việc không liên quan chặt chẽ với nhau, chẳng hạn như các package Utils thường chứa nhiều hàm tiện ích ít liên quan đến nhau. Ngược lại, thiết kế có cohesion cao (high cohesion) khi một lớp, module hoặc package chỉ tập trung vào một nhiệm vụ hoặc mục tiêu cụ thể và duy nhất.

Tóm lại, một thiết kế phần mềm tốt nên có mức độ cohesion cao và coupling thấp, nghĩa là high cohesion và low coupling.

2.2.1. Mức độ coupling

#	Các mức độ về Coupling	Module	Mô tả	Lý do
		entity\media\Media.java	Các thuộc tính có phạm vi truy cập là protected	Khi để phạm vi truy cập của thuộc tính là protected, các lớp con và các lớp cùng pakage có thể truy xuất trực

1	Content coupling			tiếp và sửa đổi giá trị thuộc tính
		entity\shipping\DeliveryInfo.java	Các thuộc tính có phạm vi truy cập là protected	Khi để phạm vi truy cập của thuộc tính là protected, các lớp cùng pakage có thể truy xuất trực tiếp và sửa đổi giá trị thuộc tính
2	Common coupling	views\screen\ViewsConfig.java	Các biến PERCENT_VAT và REGULAR_FONT là các biến static dùng chung cho nhiều thành phần 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 với cú pháp final để tránh việc bị thay đổi
		controller/SessionInformation	Lớp này được sử dụng và sửa đổi bởi nhiều lớp khác	Do có nhiều lớp tham gia sử dụng và sửa đổi nên khó quản lý dữ liệu
3	Control coupling	Không có	Không có	Không có
4	Stamp coupling	entity\cart\Cart.java	Tham số của phương thức checkMediaInCart đang có kiểu dữ liệu là Media	Trong hàm không cần sử dụng hết các giá trị của Media mà chỉ cần dùng id, dẫn đến dư thừa dữ liệu truyền vào

		controller\PlaceOrderController.java	Tham số của hàm ValidateDeliveryInfo đang truyền vào cả HashMap	Truyền cả giá trị HashMap vào trong khi hàm chỉ sử dụng giá trị phone, name, address dẫn đến dư thừa dữ liệu
		dao\media\MediaDAO.java	Tham số truyền vào phương thức updateMediaFieldById có tbname dạng String	Truyền giá trị tbname vào nhưng không sử dụng dễn đến dư thừa
		entity\cart\CartItem.java	Hàm khởi tạo CartItem có tham số truyền vào là Cart	Truyền giá trị Cart vào nhưng không sử dụng dễn đến dư thừa
		entity\shipping\DeliveryInfo.java	Hàm CalculateShippingFee có tham số truyền vào là Order	Truyền cả giá trị order nhưng bên trong không sử dụng hết dẫn đến dư thừa

Bảng 2: Mức độ coupling

2.2.2. Mức độ cohesion

#	Các mức độ về Cohesion	Module	Mô tả	Lý do
1	Coincidental cohesion	controller\AuthenticationController.java	Hàm md5 không phục vụ mục đích của lớp là xác thực người dùng	Hàm md5 giúp mã hóa chuỗi String, không liên quan đến xác thực, nên được tách ra lớp riêng
		controller\PaymentController.java	Phương thức getExpirationDate không phục vụ mục đích của lớp đó là xử lý thanh toán trực tuyến	Hàm getExpirationDate giúp kiểm tra ngày hết hạn thẻ có đúng format không và trả về format theo yêu cầu, không liên quan đến nhiệm vụ của lớp, nên được tách ra lớp riêng
		subsystem\interbank\InterbankPayloadConverter.java	Phương thức getToday không phục vụ mục đích của lớp	Hàm getToday có chức năng lấy ra thời gian hiện tại không liên quan đến nhiệm vụ của lớp, nên được tách ra lớp riêng
2	Logical cohesion	utils\ApplicationProgrammingInterface.java	Các phương thức post và get liên quan logic với nhau	Hai phương thức get và post có chức năng thuộc cùng loại nhưng lại có cách

				thức xử lí khác nhau, nên chia ra các lớp riêng
3	Temporal cohesion	Các lớp ScreenHandler trong \views\screen	Có các phương thức setUpData và setUpFunctional được gọi trong phương thức khởi tạo	Các phương thức setUpData và setUpFunctional cùng thực hiện nhiệm vụ tại thời điểm khởi tạo các màn ScreenHandler
4	Procedure cohesion	Không có	Không có	Không có
5	Communicational cohesion	entity\cart\Cart.java	Có các phương thức của lớp thực hiện trên cùng thành phần dữ liệu	Các phương thức getTotalMedia, calSubtotal, checkAvailabilityOfProduct, checkMediaInCart đều thực hiện cùng trên một thành phần dữ liệu là lstCartItem

Bảng 3: Mức độ cohesion

2.3. Đánh giá việc tuân theo SOLID

2.3.1. Single Responsibility Principle (SRP)

#	Module	Mô tả	Lý do

1	entity.shipping. DeliveryInfo.java	Class DeliveryInfo chứa một method tính toán chi phí vân chuye	Class DeliveryInfo là một entity chỉ chứa các thông tin về Delivery. Giải pháp đưa phương thức tinh toán chi phí vận chuyển ở môt
2	controller. AuthenticationController.java	Class này có hàm băm md5 có vai trò khác với method login và logout8u	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.java	Class này có các method placeOrder(), createOrder(), createInvoice(), các method validate dữ liệu	Class này có nhiều hơn một lý do đe thay đôi mã nguồn. Giải pháp đó là chia các method không liên quan sang một class khác.

Bảng 4: Vi phạm SRP

2.3.2. Open Closed Principle (OCP)

#	Module	Mô tả	Lý do
1	- subsystem.InterbankSubsystem	Các class bên đang có một phụ thuộc trực tiếp vào CreditCard là một phương thức thanh toán	Các phương thức payOrder trong các class trên chỉ dành cho CreditCard. Nếu có thêm một

	- subsystem.InterbankInterface.java - controller. PaymentController.java		phương thức thanh toán khác thì phải thay đổi trực tiếp mã nguồn.
2	controller. PlaceOrderController.java	Class phụ thuộc trực tiếp vào class DistanceCalculator	Nếu như có một cách tính khoảng cách khác thì sẽ phải thay đổi mã nguồn do PlaceOrderController phụ thuộc vào DistanceCalculator
3	entity.shipping.DeliveryInfo.java	Là một class entity nhưng lại chứa phương thức tính phí ship calculateShippingFee()	Method calculateShippingFee chỉ tính phí ship theo khoảng cách, nếu như có thêm một cách tính phí khác dựa trên cân nặng của sản phẩm thì phải thay đổi mã nguồn.

Bảng 5: Vi phạm OCP

2.3.3. Liskov Substitution Principle (LSP)

2.3.4. Interface Segregation Principle (ISP)

2.3.5. Dependency Inversion Principle (DIP)

#	Module	Mô tả	Lý do
1	- controller. PaymentController.java - subsystem	Các class và interface phụ thuộc vào chi tiết đó là CreditCard trong thanh toán.	Hệ thống có thể mở rộng ra và có thêm nhiều phương thức thanh toán khác ngoài CreditCard. Nên có abstract class hoặc interface để

	- entity.payment. PaymentTransaction.java		những chi tiết này phụ thuộc vào tránh thay đổi mã nguồn.

Bảng 6: Vi phạm DIP

2.4. Các vấn đề về Clean Code

2.4.1. 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.java	Method getExpirationDate có tên biến strs	Sửa strs thành dateStrs để thể hiện danh sách các ngày hết hạn
2	controller.PlaceOrderController.java	Method validatePhoneNumber có tên hằng số 10 là số chữ số điện thoại	Chuyển hàng số thành biến final có ý nghĩa lengthPhoneNumber
3	view.screen..home.HomeScreenHandler.java	Method setupData()	biến m nên đổi thành mediaHandler
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

5	views.screen.payment.PaymentScreenHandler.java	Method confirmToPayOrder()	biến ctrl nên đổi thành paymentController
6	views.screen.FXML.ScreenHandler.java	Method setImage()	Tham số là imv đổi th
7	utils.ApplicationProgrammingInterface.java	Các method	Các biến in, conn chưa rõ ràng, n thành input, connection

Bảng 7: Vi phạm clean name

2.4.2. Clean function/method

#	Module	Vị trí có tên chưa clear	Đề xuất sửa đổi
1	controller.AuthenticationController.java	Phương thức md5	Thay đổi tên phương thức md5 thành genDigestByMd5 để thể hiện nhiệm vụ của hàm này là sinh một message-digest và mã hoá bằng cách dùng thuật toán md5.
2	dao.media.MediaDAO.java	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.java	Phương thức update có hai vị trí log error dùng hard-code	Tạo các biến dùng chung addMediaFail, loginFail

4	controller.ViewCartController.java	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.java	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

Bảng 8: Vi phạm clean function/method

2.4.3. 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.java entity.media.Book.java entity.media.DVD.java
2	Có các attribute được khai báo nhưng không được sử dụng	utils.Utils.java utils.ApplicationProgrammingInterface.java

3	Có các method được khai báo nhưng không được sử dụng	entity.invoice.Invoice.java dao.media.MediaDAO.java views.screen.popup.PopupScreen.java views.screen.shipping.ShippingScreenHandler.java
4	Các lớp không được sử dụng	dao.order.OrderDAO.java dao.order.OrderItemDAO.java dao.invoice.InvoiceDAO.java
5	Class có các đoạn log có cùng giá trị nhưng đang hard-code thay vì tạo biến để dùng chung	utils.MyMap.java

Bảng 9: Vi phạm clean class

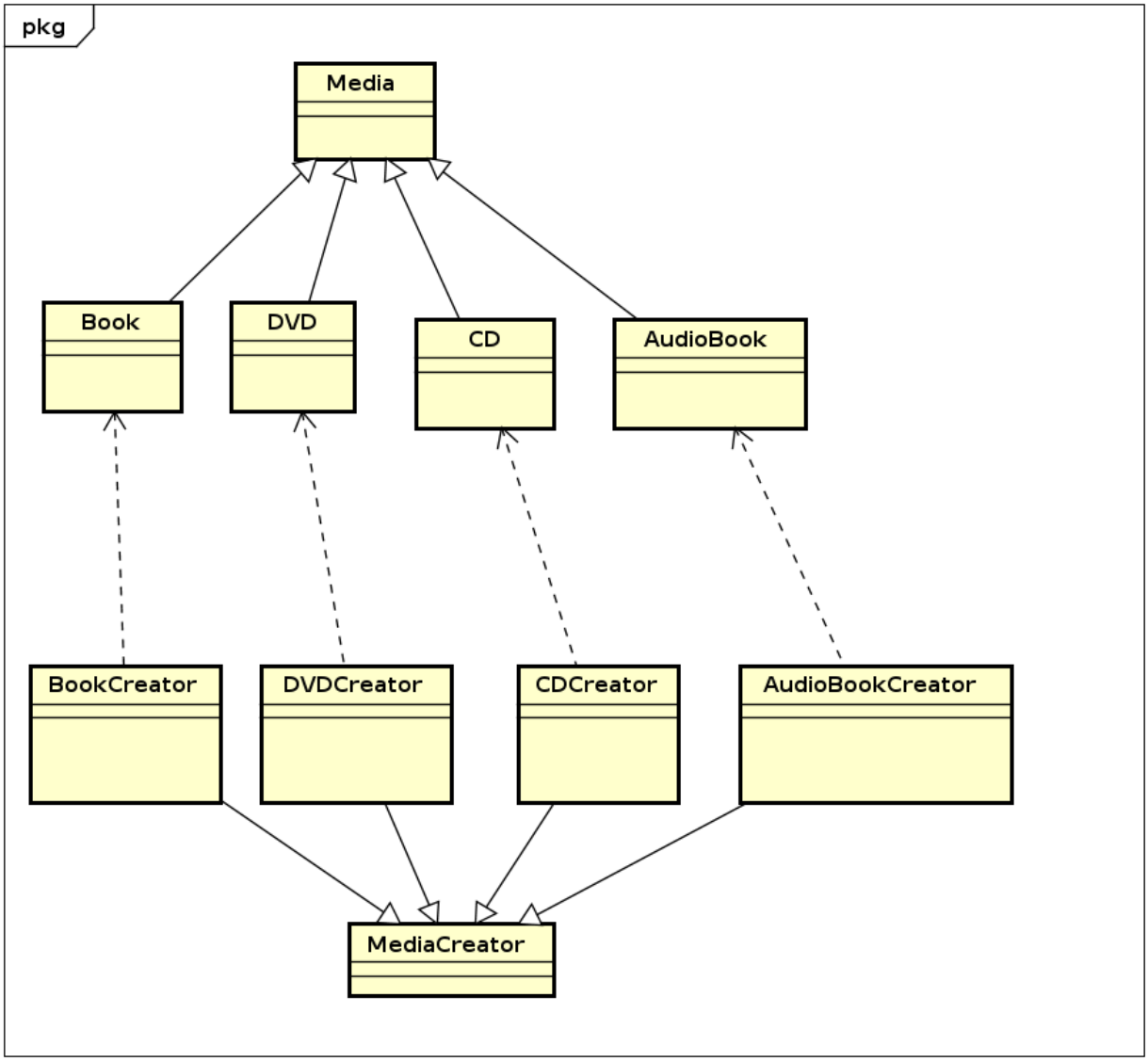
3. Đề xuất cải tiến

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

Ta sử dụng Factory Method cho việc lấy thông tin của media theo từng loai.

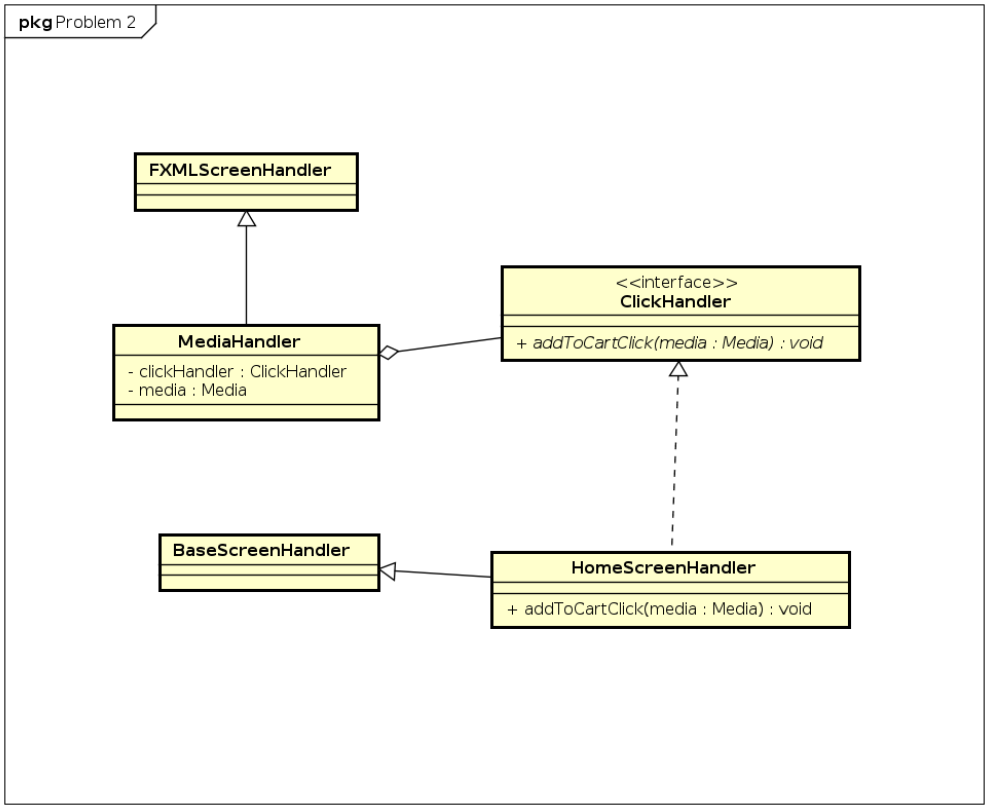
Có một class MediaCreator và các lớp BookCreator, DVDCreator, CDCreator và AudioBookCreator kế thừa lớp này.

Khi thêm một loại mặt hàng Media mới, cần thêm một lớp kế thừa lớp Media và một lớp Creator kế thừa MediaCreator.



Hình 3: Thiết kế lớp cho vấn đề thêm loại Media

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



Hình 4: Thiết kế lớp cho vấn đề xem chi tiết sản phẩm

Sử dụng một interface dùng để giao tiếp giữa HomeScreenHandler và MediaHandler. HomeScreenHandler sẽ implement interface này. Khi có sự kiện nào đó xảy ra ở MediaHandler thì cần gọi phương thức từ ClickHandler dể HomeScreenHandler xử lý.

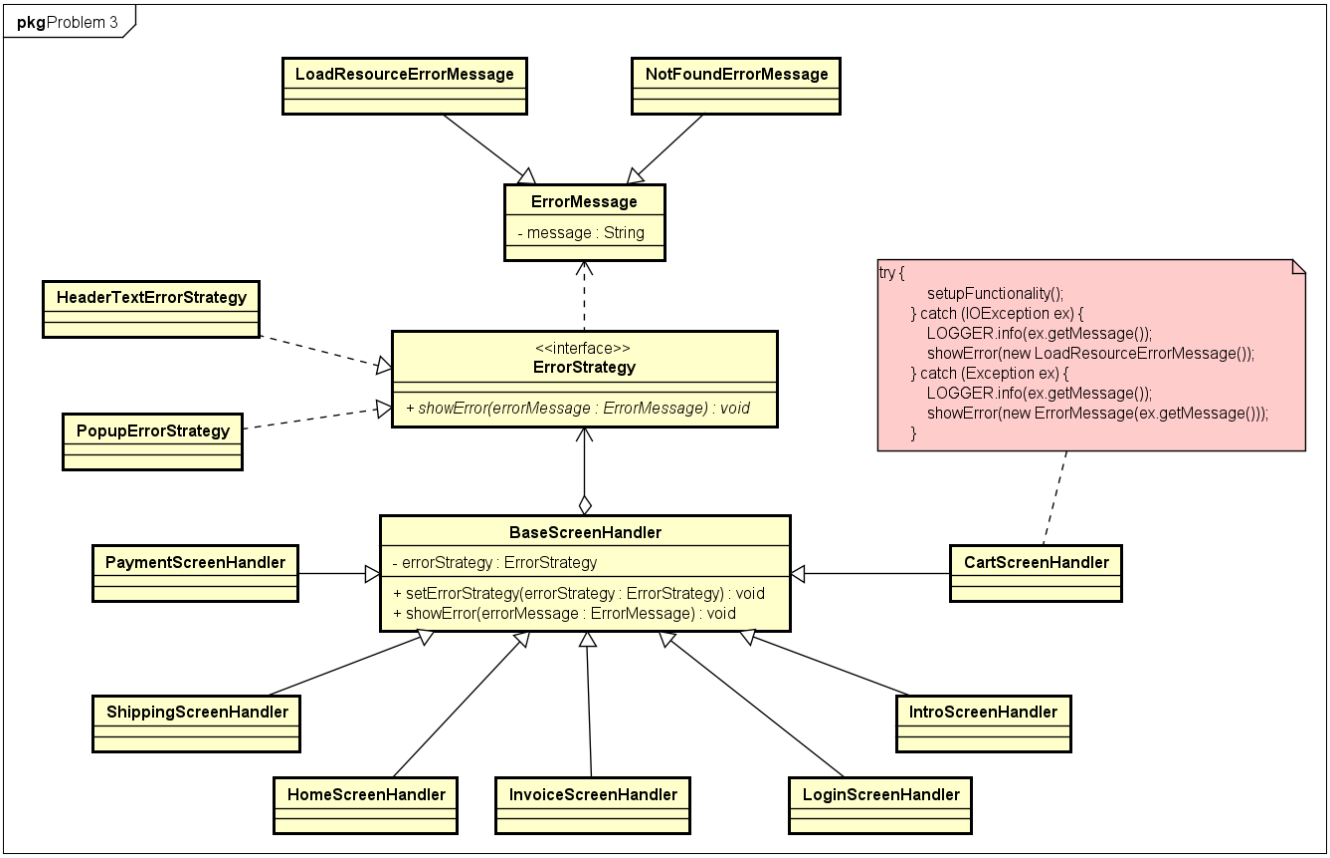
3.3. Đề xuất khi thay đổi yêu cầu khi load giao diện

Khi mỗi lần xảy ra lỗi IOException, các phương thức sẽ gọi tới một hàm static PopupScreen.error(“message”), với thiết kế như vậy thì sẽ tồn tại 2 vấn đề.

- Nếu sau này muốn thay đổi cách hiển thị thông báo lỗi là hiển thị trực tiếp lỗi màu đỏ trên phía trên cùng của các trang thì cần phải đi sửa tất cả những nơi gọi PopupScreen.
- Nếu hiển thị thông báo lỗi theo kiểu truyền String thì sẽ khó quản lý những kiểu lỗi xảy ra, ngoài ra con gây tình trạng trùng lặp kiểu lỗi.

Nhóm đề xuất sử dụng Strategy Pattern để tái cấu trúc mã nguồn. Trong lớp BaseScreenHandler thêm vào ErrorStrategy để đại diện cho hành vi thông báo lỗi, có phương thức showError (ErrorMessage). Trong đó, ErrorMessage là lớp cha đại diện cho các kiểu lỗi, từ đó có các lớp con kế thừa như LoadResourseErrorMessage, NotFoundErrorMessage. Như vậy, khi các lớp ScreenHandler cần thông báo lỗi chỉ cần gọi đến phương thức showError (ErrorMessage), nếu cần thay đổi cách thức thông báo thì gọi setErrorStrategy (ErrorStrategy).

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



Hình 5: Thiết kế lớp giải quyết vấn đề xử lý IOException khi load màn hình

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

Thay đổi phương thức tính khoảng cách sử dụng thư viện mới: Class DeliveryInfo hiện tại đang có một thuộc tính có kiểu dữ liệu là DistanceCalculator (thuộc thư viện Distance-API). Vấn

đề đặt ra là nếu như muốn thay đổi chuyển sang sử dụng thư viện tính khoảng cách mới (Alt-Distance-API) thì có thể làm theo các cách sau:

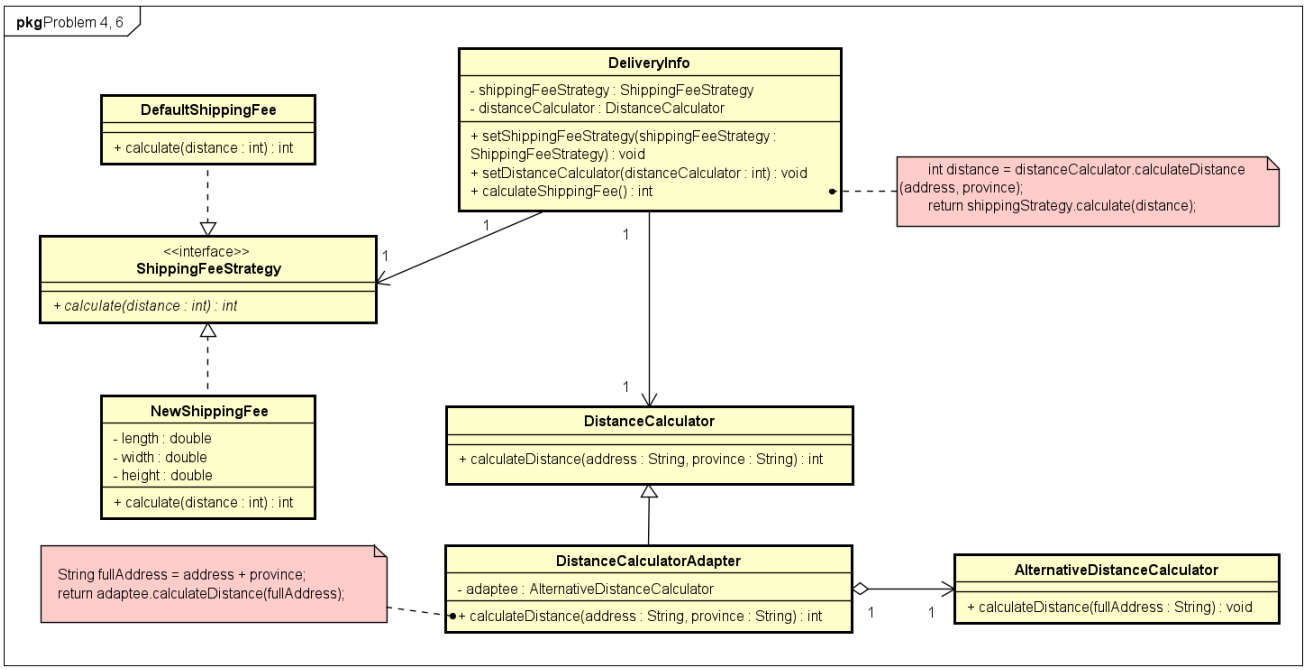
- Sửa đổi thuộc tính DistanceCalculator: Đối với giải pháp này thì sẽ ảnh hưởng đến rất nhiều nơi đang sử dụng nó
- Sửa đổi thư viện DistanceCalculator: Không thể sửa đổi thư viện

Do vậy, nhóm đề xuất sử dụng Adapter Pattern để giải quyết vấn đề này như sau: tạo DistanceCalculatorAdapter kế thừa từ DistanceCalculator và ghi đè lại phương thức calculateDistance, trong phương thức có gọi đến adaptee.calculateDistance (với adaptee là một đối tượng của lớp AlternativeDistanceCalculator).

Thay đổi công thức tính phí vận chuyển: Class DeliveryInfo hiện đang tính phí vận chuyển qua phương thức calculateShippingFee. Nếu như muốn chuyển đổi sang sử dụng một cách tính phí vận chuyển khác thì có thể có nhiều cách, nhưng cần đưa ra một giải pháp làm sao cho có khả năng chuyển đổi linh hoạt trong việc sử dụng, mở rộng trong trường hợp nếu phát sinh thêm nhiều cách tính phí khác.

Để có một thiết kế tốt đảm bảo các yêu cầu trên thì nhóm đề xuất sử dụng Strategy Pattern. Tạo interface ShippingFeeStrategy đại diện cho cách tính phí, có phương thức là calculate và cho các lớp DefaultShippingFee và NewShippingFee cài đặt nó. Bên trong DeliveryInfo thêm thuộc tính mới là shippingFeeStrategy, thuộc tính này được gọi bên trong phương thức calculateShipping để tính toán phí vận chuyển.

Biểu đồ thiết kế sau khi bổ sung như sau:



Hình 6: Giải quyết vấn đề thay đổi cách tính chi phí và thư viện tính khoảng cách

3.5. 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. 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 thì không thể thêm được. Nhóm đề xuất tạo lớp cha chung cho tất cả loại thẻ là lớp Card, sau đó thay thế toàn bộ các phương thức, thuộc tính phụ thuộc vào CreditCard bằng Card.

Ngoài ra, còn một vấn đề khác khi thay đổi loại thẻ, đó là trong phương thức extractPaymentTransaction của lớp InterbankPayloadConverter có nhiệm vụ nhận vào responseText và từ đó tạo mới PaymentTransaction, phương thức này đang bị phụ thuộc vào CreditCard như sau:

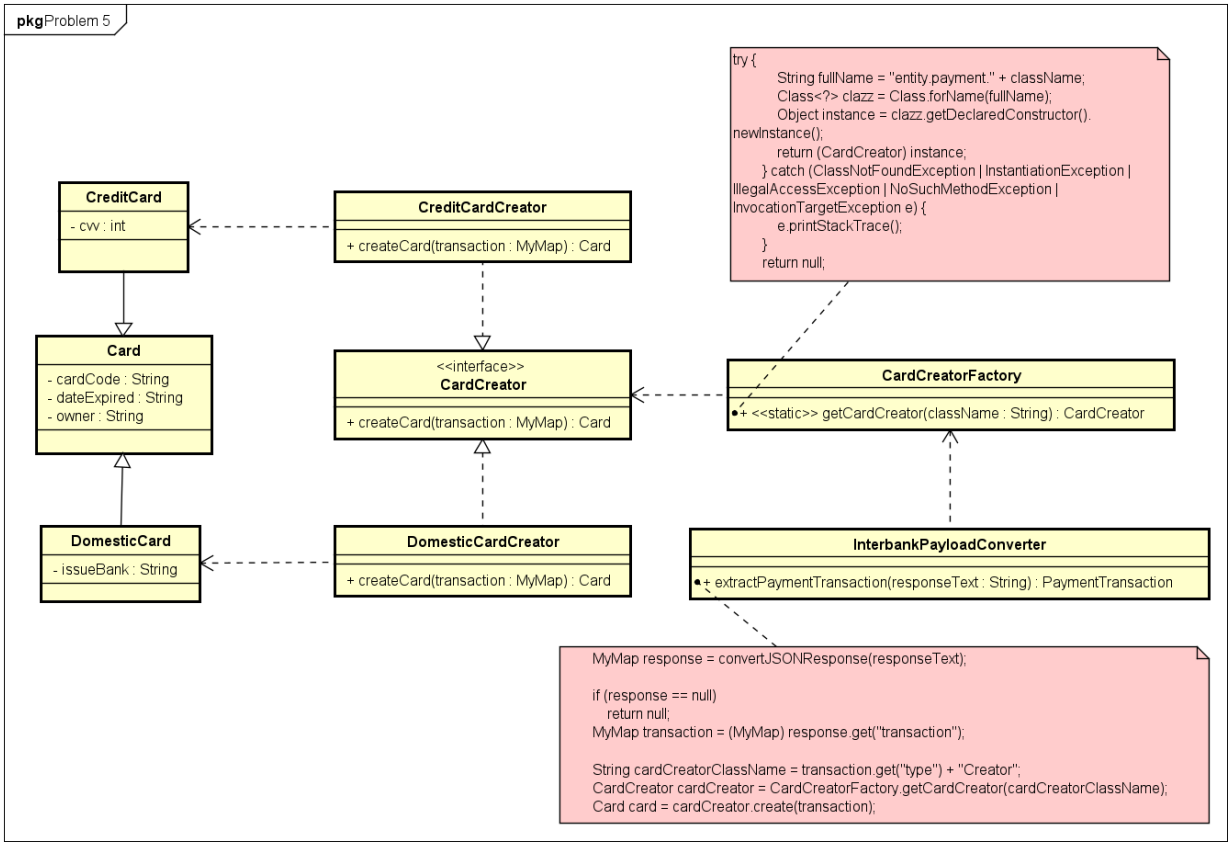
```
PaymentTransaction extractPaymentTransaction(String responseText) {
    MyMap response = convertJSONResponse(responseText);

    if (response == null)
        return null;
    MyMap transaction = (MyMap) response.get("transaction");
    CreditCard card = new CreditCard(
            (String) transaction.get("cardCode"),
            (String) transaction.get("owner"),
            (String) transaction.get("dateExpired"),
            Integer.parseInt((String) transaction.get("cvvCode")));

    PaymentTransaction trans = new PaymentTransaction(
            (String) response.get("errorCode"),
            card,
            (String) transaction.get("transactionId"),
            (String) transaction.get("transactionContent"),
            Integer.parseInt((String) transaction.get("amount")),
            (String) transaction.get("createdAt"));
```

Hình 7: Vấn đề khi thêm thẻ nội địa (DomesticCard)

Nhóm đề xuất sử dụng Factory Pattern, kết hợp với thư viện reflection trong Java để tránh việc phải sử dụng các câu lệnh điều kiện if else dẫn đến vi phạm nguyên lý thiết kế OCP. Để áp dụng Factory Pattern, cần tạo interface CardCreator có phương thức create nhận vào tham số là kiểu MyMap, các lớp con CreditCardCreator, DomesticCardCreator cài đặt interface CardCreator.
Để có thể khởi tạo các lớp một cách linh hoạt, cần tạo lớp CardCreatorFactory có nhiệm vụ nhận vào tham số là tên lớp dạng chuỗi và trả về đối tượng CardCreator tương ứng.
Biểu đồ thiết kế sau khi bổ sung như sau:



Hình 8: Thiết kế lớp giải quyết vấn đề phát sinh phương thức thanh toán mới

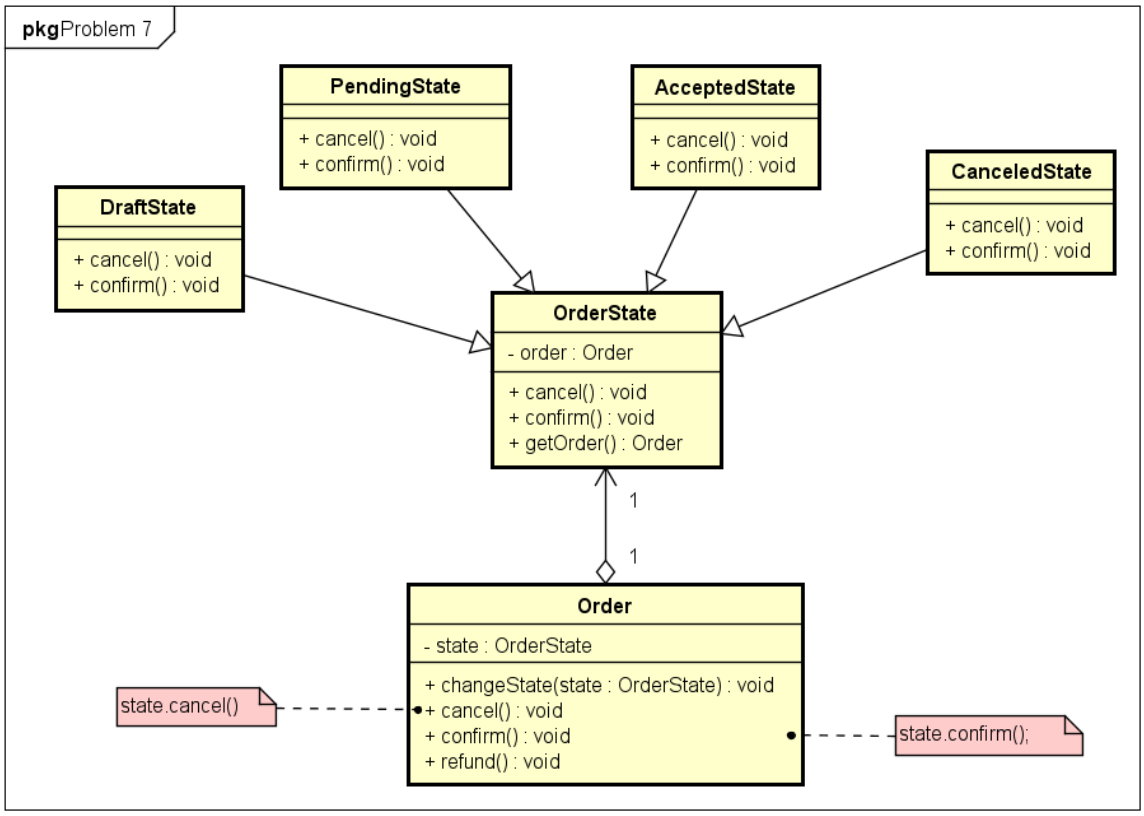
3.6. Cập nhật lại chức năng hủy đơn hàng

Trong thiết kế cũ, khách hàng có thể hủy đơn hàng và được hoàn tiền bất kỳ lúc nào, lớp Order hoàn toàn không nhận biết được trạng thái đơn hàng. Vấn đề là đối với lần cập nhật tiếp theo chức năng hủy sẽ bị vô hiệu hóa nếu như quản trị viên đã phê duyệt, nếu như lớp Order không biết về trạng thái thì có thể đơn giản là trong phương thức cancel có thể sử dụng if else dựa trên tham số đầu vào, hoặc các cách thức khác tương tự. Như vậy đều dẫn đến vi phạm nguyên lý OCP, khó mở rộng và phát triển, do đó nhóm để xuất sử dụng State Pattern, bởi vì lớp Order có nhiều trạng thái có thể chuyển đổi lẫn nhau, và tương ứng với các trạng thái lại có các hành vi khác nhau với cùng phương thức.

Việc cài đặt State Pattern được thực hiện như sau, trước tiên tạo lớp abtract OrderState chứa thuộc tính Order và các phương thức chung là cancel() và confirm(), sau đó tạo các lớp con kế thừa lại

OrderState: DraftState, PendingState, AcceptedState, CanceledState. Sau đó, trong lớp Order cho chứa thuộc tính orderState đại diện cho trạng thái của Order, chứa các phương thức cancel(), confirm() phía trong thân phương thức có gọi đến orderState và hàm tương ứng của nó.

Biểu đồ thiết kế sau khi bổ sung như sau:



Hình 9: Thiết kế lớp giải quyết vấn đề cập nhật chức năng hủy đơn hàng

4. Tổng kết

4.1. 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 đã giảm đi đáng kể.

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.

4.2. 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 còn tồn đọng về tính high coupling và low cohesion của các lớp vẫn chưa được giải quyết triệt để.