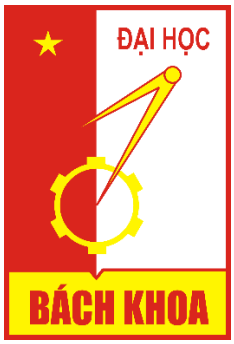
ĐẠI HỌC BÁCH KHOA HÀ NỘI

Trường CNTT & TT

----- ☙ 🕮 ❧ -----



Báo cáo học phần
Mẫu thiết kế phần mềm

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

Sinh viên thực hiện: Nhóm 06

Họ và tên	Mã số sinh viên
Chu Mạnh Tiến	20194182
Nguyễn Văn Biển	20193993
Trần Xuân Đồng	20194020
Nguyễn Việt Tùng	20194208

Hà Nội, tháng 7 năm 2023

Phân công công việc

Các vấn đề Coupling, Cohesion, SOLID: Nhóm thực hiện theo subteam (2 người) và trao đổi chéo với nhau. Trong đó:

- Subteam 1:
    - Nguyễn Việt Tùng - 20194208
    - Trần Xuân Đồng - 20194020
- Subteam 2:
    - Nguyễn Văn Biển – 20193993
    - Chu Mạnh Tiến - 20194182

Các công việc còn lại, nhóm thực hiện phân công như sau:

Họ tên	**MSSV**	**Công việc**
Nguyễn Văn Biển	20193993	Yêu cầu thêm số 2, 5 (Thêm mặt hàng Media mới, Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)); Giải quyết vấn đề vi phạm nguyên lý Single Responsibility Principle trong class AuthenticationController); làm báo cáo, slide
Trần Xuân Đồng	20194020	Yêu cầu thêm số 3 (Vấn đề Thay đổi yêu cầu khi load giao diện); Clean code (clean name, clean function - method, clear class); làm báo cáo, slide
Chu Mạnh Tiến	20194182	Yêu cầu thêm số 4, 6 (Thay đổi cách tính khoảng cách, sử dụng thư viện mới; thay đổi cách tính phí vận chuyển mới); Singleton pattern; Clean code – clean name; làm báo cáo, slide
Nguyễn Việt Tùng	20194208	Yêu cầu thêm số 2 (Vấn đề Thêm màn hình: Xem chi tiết sản phẩm và giải pháp); Biểu đồ usecase tổng quan; làm báo cáo, slide

Danh sách các hình ảnh

Danh sách các bảng

Mục lục

1. Tổng quan

Nội dung bản báo cáo trình bày gồm có:

- Đánh giá thiết kế cũ theo các nguyên tắc Coupling, Cohesion, nguyên lý SOLID, các vấn đề về Clean Code
- Các đề xuất cải tiến cho các vấn đề nêu trên dựa theo các Design Pattern, được mô tả thông qua biểu đồ lớp, biểu đồ tương tác và minh họa mã nguồn

1.1. 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 là những người đã có kiến thức cơ sở về lập trình hướng đối tượng sử dụng ngôn ngữ Java, người thiết kế hệ thống phần mềm, người có kiến thức cơ bản về công nghệ thông tin. Báo cáo là kết quả làm việc của nhóm tác giả, nhằm tổng kết kết quả, cách thức thực hiện việc tái cấu trúc, refactor mã nguồn dựa trên một hệ thống phần mềm có sẵn được cung cấp bởi giảng viên giảng dạy môn học, sử dụng các mẫu thiết kế phần mềm trong môn học để áp dụng vào phân tích và cải tiến thiết kế hiện tại.

1.2. Phạm vi

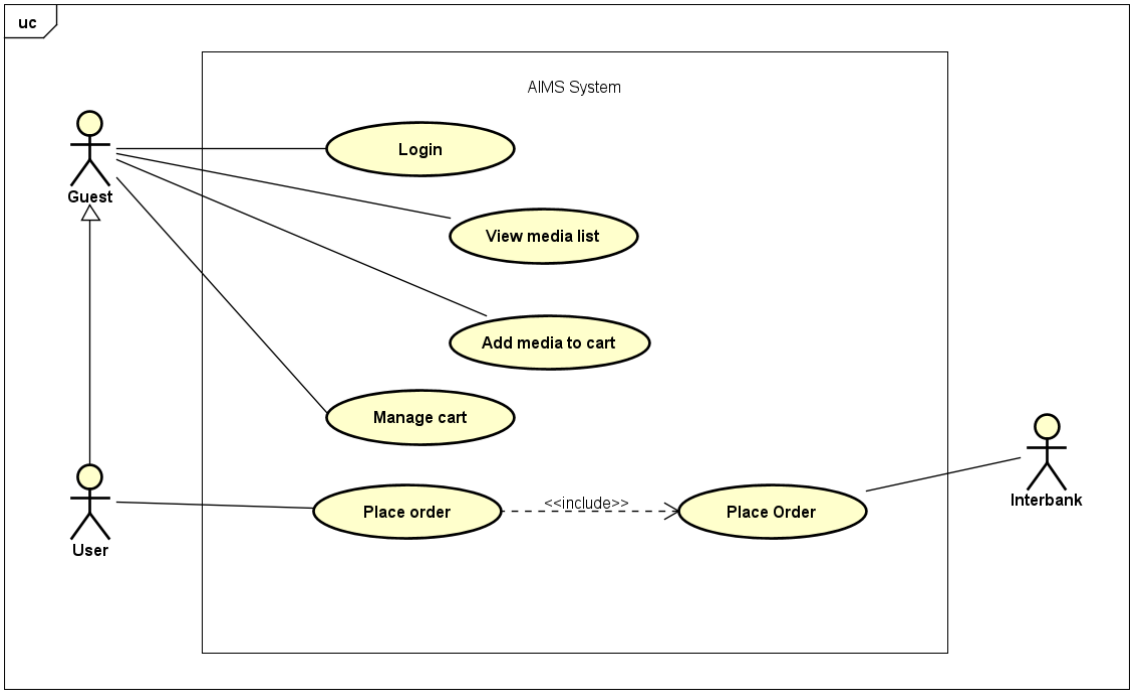
1.2.1. Mô tả khái quát phần mềm

Phần mềm AIMS là một cửa hàng online, có chức năng đặt hàng các loại media (sách, đĩa CD, đĩa DVD) từ cửa hàng. Sau khi đặt hàng, đơn hàng sẽ được thanh toán online bởi thẻ tín dụng của người dùng và sau đó sản phẩm sẽ được vận chuyển tới địa chỉ mà khách hàng yêu cầu. Phần mềm hiện tại đã đáp ứng được các chức năng chính tại thời điểm hiện tại, nhưng chưa đảm bảo việc tuân thủ các nguyên lý SOLID và không có khả năng đáp ứng được một số yêu cầu thay đổi có thể phát sinh trong tương lai.

1.2.2. Các chức năng chính của phần mềm

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

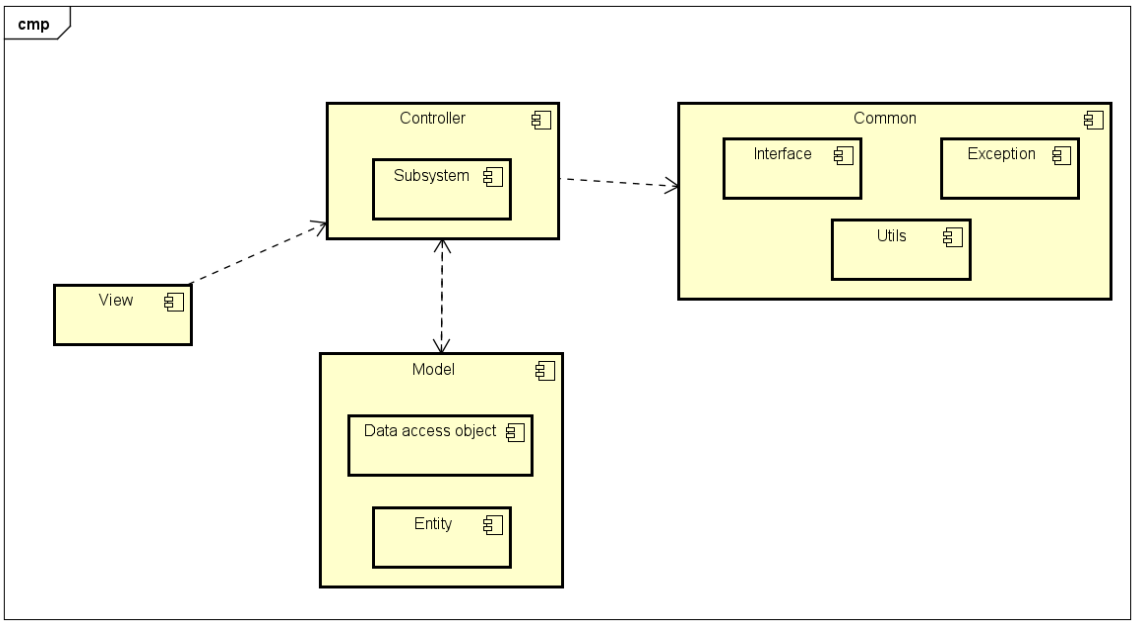
- Xem, tìm kiếm danh sách các sản phẩm đang có trong cửa hàng
- Thanh toán đơn hàng, xác định địa chỉ giao hàng và hướng dẫn giao hàng



Hình 1. Biểu đồ use case tổng quan

Hệ thống AIMS có các tác nhân: Guest, User và Interbank. Guest, User đều có chung một số hành vi nghiệp vụ khi tương tác với hệ thống AIMS.Guest là vai trò của người sử dụng chưa đăng nhập vào hệ thống, do đó Guest có thể thực hiện Login (đăng nhập), View Media List (xem danh sách sản phẩm), Add media to cart (thêm sản phẩm vào giỏ hàng), Manage cart (quản lý giỏ hàng, bảo gồm xem, sửa giỏ hàng). User là vai trò của người sử dụng đã đăng nhập vào hệ thống, có thể sử dụng chức năng Place order (đặt hàng). Trong khi thực hiện đặt hàng, sẽ có nghiệp vụ Pay order (thanh toán đơn hàng). Nghiệp vụ này có sự tham gia tương tác của tác nhân Interbank, là một tác nhân hệ thống, đại diện cho hệ thống thanh toán online.

1.1.3. Cấu trúc mã nguồn



Hình 2. Thiết kế tổng quan của hệ thống ban đầu

Mã nguồn của phần mềm hiện tại đang được thiết kế theo mô hình MVC, gồm ba tầng chính là Model, View và Controller. Trong đó, tầng View bao gồm các xử lý logic ở các màn hình, lấy dữ liệu từ người người dùng và chuyển cho tầng Controller và thực hiện điều hướng giữa các màn hình với nhau; tầng Controller bao gồm việc xử lý logic của luồng dữ liệu, xác thực dữ liệu người dùng và tương tác với hệ thống subsystem để thực hiện thanh toán tiền cho đơn hàng của người dùng; tầng Model sẽ thực hiện lưu dữ liệu của hệ thống và bao gồm hai modules chính là Data Access Object và Entity. Ngoài ra hệ thống còn có một các thành phần dùng chung, gọi chung là Common, bao gồm các Interfaces, các Exceptions và Utils.

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

Việc tiến hành tái cấu trúc mã nguồn của hệ thống hiện tại chỉ nhằm mục đích giúp hệ thống có khả năng đáp ứng với các thay đổi trong tương lai một các tốt hơn, đảm bảo các nguyên lý trong SOLID. Việc tái cấu trúc này phải đảm bảo các tính năng của hệ thống hiện tại không bị thay đổi cả về mặt chức năng và phi chức năng.

1.1.5. Các hoạt động review, refactor thực thi trên mã nguồn để đạt được mục tiêu

Các hoạt động review mã nguồn hiện tại bao gồm:

- Xác định các mức độ coupling và cohesion của mã nguồn
- Xác định các vi phạm nguyên lý SOLID

Các hoạt động refactor mã nguồn bao gồm:

- Refactor methods, class để đảm bảo các tiêu chí về clean code
- Refactor những vi phạm nguyên lý SOLID đã phát hiện được trong bước review
- Áp dụng các mẫu thiết kế phần mềm đã học vào hệ thống một cách phù hợp

1.1.6. Kết quả dự kiến

Kết quả dự kiến sau khi refactor mã nguồn là hệ thống đáp ứng được tiêu chí High Cohesion và Loose Coupling, loại bỏ những vi phạm các nguyên lý trong SOLID, giúp hệ thống có khả năng thích ứng với các yêu cầu trong tương lai tốt hơn, có khả năng mở rộng tốt hơn so với mã nguồn ban đầu bằng cách sử dụng các mẫu thiết kế, các kiến thức về clean code trên lớp.

1.2. 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

2. Đánh giá thiết kế cũ

2.1. Nhận xét chung

Về mặt chức năng hiện tại của hệ thống, mã nguồn cũ đã đáp ứng được tương đối đầy đủ các yêu cầu. Tuy nhiên với các yêu cầu có thể phát sinh trong tương lai, hệ thống hiện tại lại tỏ ra chưa có khả năng thích ứng tốt. Mã nguồn hiện tại vẫn còn các modules phụ thuộc vào nhau ở mức thấp thay vì chỉ phụ thuộc vào nhau ở mức trừu tượng. Khi phụ thuộc vào nhau ở level thấp, nếu một module thay đổi thì sẽ gây ra nhiều thay đổi ở các module mà phụ thuộc vào nó hay ngược lại. Còn khi thiết kế chỉ thuộc vào nhau ở mức trừu tượng, nếu một module thay đổi thì các module còn lại do chỉ biết về class đó một cách trừu tượng nên sẽ ít bị ảnh hưởng hơn. Hơn nữa, như đã phân tích ở trên, thiết kế cũ vẫn còn tồn đọng nhiều vấn đề về clean code, cần phải tiến hành tái cấu trúc và sửa đổi để đảm bảo mã nguồn dễ bảo trì và mở rộng hơn.

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 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 coupling, 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).

2.2.1. Coupling

#	*Các mức độ về Coupling*	*Module*	*Mô tả*	*Lý do*
1	Content Coupling	Lớp \entity\car\Cart.java	Tại phương thức getListMediaCart(), phương thức trả về một List các đối tượng CartItem	Vi phạm việc truy cập trái phép tới các đối tượng CartItem, vì nó trả về tham chiếu các đối tượng lớp CartItem nên ta hoàn toàn có thể sử dụng và sửa đổi trực tiếp dữ liệu của các đối tượng lớp CartItem
2	Content Coupling	Lớp \entity\order\Order.java	Tại phương thức getListOrderMedia(), phương thức trả về một List các đối tượng OrderItem	Vi phạm việc truy cập trái phép tới các đối tượng Order, vì nó trả về tham chiếu các đối tượng lớp OrderItem nên ta hoàn toàn có thể sử dụng và sửa đổi trực tiếp dữ liệu của các đối tượng lớp OrderItem
3	Content Coupling	\utils\ApplicationProgrammingInterface.java	Trong phương thức allowMethods() sử dụng hàm setAccessible() để thay đổi phạm vi truy cập của đối tương Field	Thay đổi phạm vi truy cập của đối tượng lớp Field sẽ làm vi phạm tính đóng gói
4	Content Coupling	\utils\MyMap.java	Trong phương thức toMyMap() sử dụng hàm setAccessible()	Thay đổi phạm vi truy cập của đối tượng lớp Field sẽ làm vi phạm tính đóng gói

			để thay đổi phạm vi truy cập của đối tương Field	
5	Common coupling	\views\screen\ViewsConfig.java	Các biến PERCENT_VAT và REGULAR_FONT của lớp Views là các biến dùng chung cho cả hệ thống. Cụ thể PERCENT_VAT có lớp Order sử dụng. REGULAR_FONT có lớp MediaHandler sử dụng	Các biến trên không được khai báo là final và được sử dụng chung giữa các lớp, các lớp có thể truy cập và sửa đổi giá trị ⇒ dẫn tới Common coupling
6	Stamp Coupling	\controller\PlaceOrderController.java	Phương thức validateDeliveryInfo (HashMap<String, String> info) có tham số info	info có nhiều trường thuộc tính khác không dùng như address, instructions nhưng vẫn truyền cả cấu trúc vào
7	Stamp Coupling	entity\shipping\DeliveryInfo.java	Phương thức calculateShippingFee(Order order)	Mặc dù truyền tham số order vào nhưng hiện tại phương thức này không sử dụng đến
8	Stamp Coupling	\subsystem\interbank\InterbankSubsystemController.java	Phương thức refund(CreditCard card, int amount, String contents)	Mặc dù truyền các tham số như mô tả nhưng không dùng, chỉ trả về giá trị null
9	Stamp Coupling	\views\screen\invoice\InvoiceScreenHandler.java	Phương thức setupData(invoice) được gọi trong InvoiceScreenHandler(Stage stage, String screenPath, Invoice invoice)	Phương thức setupData truyền vào invoice nhưng chỉ dùng invoice.order, không dùng invoice.amount

10	Data Coupling	Thường xảy ra giữa các Lớp UI và các lớp Controller	Thường nằm trong phương thức khởi tạo của các Lớp UI (Screen Handler), các lớp Controller sẽ được dùng làm tham số truyền vào trong phương thức của các lớp UI ScreenHandler, đặc biệt là phương thức khởi tạo. VD: Trong phương thức của lớp BaseScreenHandler là setBController(BaseController bController) ⇒ Data Coupling với lớp BaseController Ngoài ra: trong lớp Order: Có phương thức khởi tạo truyền đối tượng lớp Cart vào, phương thức setDeliveryInfo(DeliveryInfo deliveryInfo) ⇒ Order sẽ data coupling với lớp DeliveryInfo và lớp Cart	Tất cả dữ liệu truyền vào phương thức đều sử dụng hết, không thừa, là tham số dữ liệu, không phải tham số điều khiển

Bảng 1. Coupling

2.2.2. Cohesion

#	*Các mức độ về Cohesion*	*Module*	*Mô tả*	*Lý do*
1	Coincidental Cohesion	\utils\Utils.java	Gồm 2 biến DATE_FORMATTER và Logger	DATE_FORMATTER và Logger không hề liên quan gì đến nhau
2	Coincidental Cohesion	\subsystem\interbank\InterbankPayloadConverter.java	Phương thức getToday() được đặt trong lớp đó	Phương thức getToday() để lấy ngày hiện tại, không hề liên quan đến nhiệm vụ chính của nó là xử lý việc thanh toán qua ngân hàng
3	Logical Cohesion	\views\screen\popup\PopupScreen.java	Gồm các phương thức popup(), success(), error(), loading()	Các methods không hề có sự liên quan đến nhau về logic xử lý, chúng chỉ cùng mục đích là hiển thị thông tin lên 1 pop up nào đó
4	Temporal Cohesion	Các lớp handler trong \views\screen	Có các phương thức setUpData() và setUpFunctional() được gọi trong phương thức khởi tạo	Các hàm setUpData() setUpFunctional() cùng thực hiện nhiệm vụ tại thời điểm khởi tạo các màn ScreenHandler
5	Procedural Cohesion	\controller\PlaceOrderController.java	Có các method placeOrder(), createOrder(), processDeliveryInfo() và các	Các methods trên thực hiện theo quy trình xử lý khi đặt hàng

			method validate, createInvoice()	
6	Communicational Cohesion	\controller\AuthenticationController.java	Các phương thức getMainUser(), login(), logout() đều làm việc trên dữ liệu người dùng được lưu ở lớp SesssionInformation	Các phương thức trên được gom vào 1 lớp vì lý do cùng làm việc trên dữ liệu về thông tin mainUser, cartInstance, expiredTime của lớp SessionInformaiton, do đó vi phạm Communicational Cohesion
7	Communicational Cohesion	entity\cart\Cart.java	Các phương thức của lớp đều thực hiện cùng trên một thành phần dữ liệu là lstCartItem	Các phương thứcđược gom vào 1 lớp vì đều thực hiện cùng trên một thành phần dữ liệu là lstCartItem, do đó vi phạm Communicational Cohesion.
8	Functional Cohesion			

Bảng 2. Cohesion

2.3. Đánh giá việc tuân theo SOLID

Nhìn chung, mã nguồn thực hiện khá tốt các yêu cầu về mặt chức năng, tuy nhiên mã nguồn cũ vi phạm khá nhiều các nguyên lý thiết kế SOLID. Về nguyên lý Single Responsibility Principle, hiện vẫn còn một số lớp thực hiện nhiều hơn một nhiệm vụ, và có nhiều hơn một lý do để thay đổi. Tiêu biểu như lớp PlaceOrderController thực hiện những công việc liên quan đến đặt hàng vừa thực hiện những việc liên quan đến xác thực dữ liệu. Về nguyên lý Open Closed Principle, mã nguồn vẫn còn một số lớp vi phạm, nguyên nhân ở đây là do khi ta thay đổi yêu cầu bằng việc thêm một số hình thức thanh toán mới, hay thêm cách tính phí vận chuyển mới, thì mã nguồn hiện tại không đáp ứng được với các thay đổi đó, mà phải trực tiếp thay đổi mã nguồn nếu có thêm các yêu cầu thay đổi. Về nguyên lý Liskov Substitution Principle, có những lớp mặc dù có quan hệ cha con nhưng mà lớp con không hề có

khả năng tái sử dụng một số phương thức của lớp cha. Tiêu biểu là AuthenticationController, HomeController, PaymentController. Về nguyên lý Interface Segregation Principle, thật tốt khi trong mã nguồn không có dấu hiệu vi phạm nguyên lý này, một phần cũng là do khá ít interface trong mã nguồn. Cuối cùng, nguyên lý Dependency Inversion Principle, khá nhiều các lớp phụ thuộc trực tiếp vào lớp cụ thể, thay vì các thành phần mức trừu tượng cao. Điều này là không tốt cho thiết kế vì nếu như có sự thay đổi, hay thêm mới các hình thức khác thì ta phải thay đổi trực tiếp mã nguồn.

Với yêu cầu phát sinh là thay đổi các tính chi phí vận chuyển, thay vì chỉ tính dựa trên khoảng cách như hiện tại, cách tính mới yêu cầu việc tính chi phí dựa vào các yếu tố bổ sung là khối lượng thực tế của hàng hóa và độ cồng kềnh của kiện hàng, thiết kế hiện tại của hệ thống chưa tuân theo nguyên lý thiết kế Single Responsibility Principle, Open Closed Principle và Dependency Inversion Principle.

Với yêu cầu phát sinh là thay đổi cách tính khoảng cách, sử dụng thư viện mới, thiết kế hiện tại đang vi phạm nguyên lý Open Closed Principle và Dependency Inversion Principle. Do class DeliveryInfo đang có một attribute là DistanceCalculator là một class của thư viện bên ngoài nên khi cần thay đổi thư viện mới, chúng ta cần phải sửa attribute này thành class của thư viện mới. Việc sử dụng một thư viện mới có interface khác với interface của thư viện hiện tại cũng là dấu hiệu của việc phải sử dụng mẫu thiết kế Adapter Design Pattern để có thể refactor code của hệ thống hiện tại.

Với yêu cầu phát sinh là thêm phương thức thanh toán mới là thẻ nội địa (DomesticCard), thiết kế hiện tại cũng đang vi phạm nguyên lý Open Closed Principle và Dependency Inversion Principle do thiết kế đang sử dụng class CreditCard không phải abstract class, có các attributes mà DomesticCard không có, dẫn tới việc khó mở rộng, phát triển thêm khi có yêu cầu thêm loại thẻ DomesticCard.

Với yêu cầu phát sinh là thêm màn hình xem chi tiết sản phẩm, tại màn hình này, tất cả thông tin của sản phẩm được hiển thị đầy đủ để khách hàng xem xét. Do hệ thống có nhiều loại sản phẩm, mỗi loại lại có những thuộc tính khác nhau, nên để xử lý với yêu cầu phát sinh này, việc bổ sung một method giúp hiển thị thông tin sản phẩm ra màn hình trong class Media là cần thiết. Với yêu cầu phát sinh này, thiết kế hiện tại của hệ thống không bị vi phạm nguyên lý nào trong SOLID.

Với yêu cầu phát sinh là thêm mặt hàng Media mới là AudioBook, hiện tại trên màn hình chính của ứng dụng có liệt kê các loại sản phẩm (bao gồm Book, DVD, CD) để người dùng có thể lọc và tìm kiếm, và phần hiển thị tên các loại sản phẩm này đang bị hardcode trong codebase. Khi thêm loại Media mới sẽ phải tiến hành sửa đoạn code này (method setupFunctionality trong class HomeScreenHandler), điều này đang vi phạm nguyên lý Open Closed Principle.

2.3.1. SRP

#	*Module*	*Mô tả*	*Lý do*
1	\controller\PlaceOrderController.java	Có các method placeOrder(), createOrder(), createInvoice(), processDeliveryInfo() và các method validate, createInvoice()	Vừa xử lý, xác thực cho nhiệm vụ đặt hàng, vừa xử lý và xác thực cho delivery info => có nhiều hơn 1 lý do để thay đổi mã nguồn. Giải pháp là tách các nhiệm vụ liên quan đến xác thực qua một lớp khác.
2	\entity\shipping\DeliveryInfo.java	Ngoài các phương thức liên quan đến thông tin vận chuyển, chứa 1 phương thức để tính toán chi phí vận chuyển là calculateShippingFee()	Class DeleveyInfo là 1 entity nên việc tinh toán chi phí vận chuyển không phải là nhiệm vụ của nó. Giải pháp là ta có thể tách việc tính toán phí vận chuyển thông qua một lớp khác liên quan đến phương thức tính phí vận chuyển. Sau đó việc tính toán chi phí vận chuyển sẽ thực hiện trực tiếp trên lớp đó.
3	\controller\AuthenticationController.java	Có nhiều hơn 1 vai trò (cụ thể là hàm băm md5)	Class có nhiệm vụ xử lý logic các yêu cầu authenticate của người dùng, việc sinh mã băm không phải nhiệm vụ của nó. Giả sử sau này muốn thay đổi việc mã hóa password hoặc thay đổi cách xác

			thực người dùng ⇒ có nhiều hơn 2 lý do để thay đổi.

Bảng 3. Single Reponsibility Principle

2.3.2. OCP

#	*Module*	*Mô tả*	*Lý do*
1	controller\PaymentController.java; \subsystem\InterbankInterface.java \subsystem\InterbankSubsystem.java subsystem\interbank\InterbankSubsystemController.java	Phương thức payOrder() ở trong các module trên	Phương thức payOrder() chỉ dành cho Credit Card, sau này có thêm các hình thức thanh toán khác nữa thì phải thay đổi trực tiếp mã nguồn
2	controller\AuthenticationController.java	Lớp chứa các phương thức xác thực người dùng, chứa phương thức login()	Phương thức login(): ví dụ sau này muốn thay đổi cách xác thực (có thể qua vân tay hoặc giọng nói thì cần thay đổi trực tiếp mã nguồn)
3	\entity\shipping\DeliveryInfo.java	Chứa thông tin của delivery và phương thức tính phí ship calculateShippingFee()	Hiện tại calculateShippingFee() chỉ tính phí ship theo khoảng cách, sau này nếu thêm hình thức tính phí ship như cân nặng thì phải modify mã nguồn
4	\subsystem\interbank\InterbankPayloadConverter.java	Chứa các phương thức liên quan đến việc trích xuất thanh toán	Phương thức convertToRequestPayload() chỉ dành cho Credit Card, sau này có

		và convert dữ liệu thanh toán	thêm các hình thức thanh toán khác nữa thì phải thay đổi trực tiếp mã nguồn
5	\views\screen\popup\PopupScreen.java	Chứa các phương thức liên quan đến việc cài đặt và hiển thị Popup	Trong tương lai có thể có thêm nhiều loại Pop up nữa (VD: Warning), nên phải thay đổi mã nguồn trực tiếp (vi phạm close for modify)

Bảng 4. Open Closed Principle

2.3.3. LSP

#	*Module*	*Mô tả*	*Lý do*
1	\controller\BaseController.java	Các lớp controller AuthenticationController, HomeController, PaymentController kế thừa BaseController có các phương thức checkMediaInCart(), getListCartMedia()	Các lớp trên kế thừa BaseController là không hợp lý vì nó không sử dụng hay liên quan tới việc check và lấy dữ liệu từ giỏ hàng
2	\views\screen\popup\PopupScreen.java	Kế thừa BaseScreenHandler	Không thể sử dụng các phương thức của BaseScreenHandler như getPreviousScreen(), getBController()...

Bảng 5. Liskov Substitution Principle

2.3.4. ISP

2.3.5. DIP

#	*Module*	*Mô tả*	*Lý do*
1	\controller\AuthenticationController.java	Có phương thức login gọi đối tượng của UserDAO để thực hiện authenticate thông tin email và password	Phụ thuộc vào phương thức của lớp cụ thể, khi muốn login theo thông tin khác sẽ dẫn đến thay đổi mã nguồn => nên phụ thuộc vào 1 lớp abstract
2	\controller\PaymentController.java \subsystem\InterbankInterface.java \entity\order\Order.java	Phương thức payOrder chỉ nhận các thông tin của CreditCard	Phụ thuộc vào thông tin của 1 lớp cụ thể là CreditCard, khi muốn thanh toán theo phương thức khác sẽ phải modify mã nguồn => nên phụ thuộc vào thông tin của 1 lớp abstract có thể là cha của CreditCard

	\subsystem\InterbankSubsystem.java \subsystem\interbank\InterbankSubsystemController.java		
3	\subsystem\interbank\InterbankPayloadConverter.java	Chứa các phương thức liên quan đến việc trích xuất thanh toán và convert dữ liệu thanh toán	Lớp InterbankPayloadConverter phụ thuộc vào lớp cụ thể là CreditCard (cụ thể trong phương thức convertToRequestPayload(), sau này khi thêm phương thức tính phí mới (hoặc loại Card mới) sẽ phải chỉnh sửa trực tiếp mã nguồn. Giải pháp là sẽ nên phụ thuộc vào một lớp Card abstract thay vì lớp CreditCard
4	\entity\order\Order.java	Phương thức setDeliveryInfo() đang tính phí ship phụ thuộc vào phương thức cụ thể calculateShippingFee của deliveryInfo	Phụ thuộc vào phương thức tính phí ship của 1 lớp cụ thể là tính theo khoảng cách, khi thêm phương thức tính phí mới sẽ phải modify mã nguồn => nên phụ thuộc vào 1 phương thức abstract có nhiệm vụ tính phí ship

Bảng 6. Dependency Inversion Principle

2.4. Các vấn đề về Clean Code

2.4.1. Clean Name

Về tổng quan, tình trạng mã nguồn ban đầu đã đáp ứng được clear name, các tên của biến, method, class đều mang ý nghĩa. Chỉ có một số vị trí biến nên sửa đổi tên biến để mang nhiều ý nghĩa hơn so với hiện tại

#	*Module*	*Mô tả*	*Lý do*

1	views\screen\home\HomeScreenHandler.java	Phương thức setImage()	Chưa thấy được sự phân biệt khác biệt giữa các biến **file1** và **file2, image1** và **image2** ⇒ nên sửa lại: - file1 ⇒ logoFile - file2 ⇒ cartFile - img1 ⇒ logoImage - img2 ⇒ cartImage
2	views\screen\home\HomeScreenHandler.java	Phương thức setupData()	Có biến **medium** bị sai chính tả. Nên đổi tên biến **medium** thành **mediaList**
3	views\screen\home\HomeScreenHandler.java	Phương thức setupData()	Biến **m** trong câu lệnh MediaHandler **m = new MediaHandler()** không rõ ý nghĩa. Nên đổi **m** ⇒ **mediaHandler**
4	views\screen\cart\CartScreenHandler.java	Phương thức setupFunctionality()	Có biến **im** được đặt tên không có ý nghĩa rõ ràng. Nên đổi **im ⇒ image**
5	views\screen\invoice\InvoiceScreenHandler.java	Phương thức setupData()	Có biến **mis** được đặt tên không có ý nghĩa rõ ràng, nên đổi **mis ⇒ mediaInvoiceScreenHandler**
6	views\screen\payment\PaymentScreenHandler.java	Phương thức confirmToPayOrder()	Có biến **ctrl** được đặt tên chưa có ý nghĩa rõ ràng, nên đổi **ctrl ⇒ paymentController**

7	views\screen\FXMLScreenHandler.java	Phương thức setImage()	Nhận tham số là biến **imv** được đặt tên chưa có ý nghĩa rõ ràng. Nên đổi **imv ⇒ imageView**
8	views\screen\ViewsConfig.java	Phương thức getCurrencyFormat()	Nhận tham số là biến **num** được đặt tên chưa có ý nghĩa rõ ràng. Nên đổi **num ⇒ number**
9	utils\ApplicationProgrammingInterface.java	Trong các phương thức của lớp	Có biến **in, conn,** được đặt tên có ý nghĩa chưa rõ ràng. Biến **respone** sai chính tả. Nên đổi **in ⇒ input, conn ⇒ connection, respone ⇒ response**
10	utils\MyMap.java	Trong các phương thức của lớp	Có biến **sb, e** (Entry), **it** được đặt tên có ý nghĩa chưa rõ ràng. Nên đổi **sb ⇒ stringBuffer, e ⇒ entry, it ⇒ iterator**
11	subsystem\InterbankSubsystem.java	Thuộc tính đối tượng bên trong lớp	Thuộc tính **ctrl** được đặt tên chưa có ý nghĩa rõ ràng. Nên đổi **ctrl ⇒ interbankSubsystemController**
12	controller\PaymentController.java	Method getExpirationDate có tên biến strs	Sửa tên biến strs thành dateSplitedStrs để thể hiện rõ ràng ý nghĩa của biến

Bảng 7. Clean Code - Clear Name

2.4.2. Clean Function/Method

#	*Module*	*Mô tả*	*Lý do*
1	views\screen\cart\CartScreenHandler.java	Phương thức setupFunctionality() được sử dụng để thiết lập các chức năng và	phương thức setupFunctionality chứa nhiều try-catch không cần thiết,setOnMouseClicked xử lý 2 sự kiện khác nhau

		xử lý sự kiện của các thành phần trong giao diện người dùng của CartScreenHandler	
2	dao.media. MediaDao	Method getCurrentQuantity có truyền vào tham số id của media nhưng không sử dụng để làm tham số cho câu truy vấn cơ sở dữ liệu. Hiện tại câu truy vấn cơ sở dữ liệu đang là "SELECT * FROM MEDIA"	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	controller\AuthenticationController	Method md5	Do md5 là tên một thuật toán dùng để mã hóa, nên cần sửa tên method md5 thành genDigestByMd5 để thể hiện rõ ràng nhiệm vụ của hàm này là sinh một message-digest sử dụng hàm md5.
4	views\screen\home\HomeScreenHandler	Method update có hai vị trí log ra cùng message giống nhau, nhưng không khai báo biến để dùng chung mà sử dụng hard-code	Tạo biến errorMessage để lưu các giá trị log giống nhau
5	controller\ViewCartController	Method getCartSubtotal	Sửa phần cài đặt của method thành inline function

6	views\screen\intro\IntroScreenHandler	Method setupFunctionality đang hard-code đường dẫn tới ảnh logo của ứng dụng	Chuyển giá trị đường dẫn tới logo của ứng dụng vào class ViewsConfig để tránh việc hard-code và có thể tái sử dụng hoặc thay đổi giá trị này dễ dàng hơn

Bảng 8. Clean Code - Clean Function/Method

2.4.3. Clean Class

Hiện tại, các class của mã nguồn ban đầu hầu hết đáp ứng được các tiêu chí về clean class, nhưng vẫn còn một số class có các attribute được khai báo nhưng không sử dụng, các import không sử dụng hoặc một số method không có phần cài đặt hoặc có phần cài đặt nhưng lại không được sử dụng tại bất kỳ vị trí nào trong hệ thống.

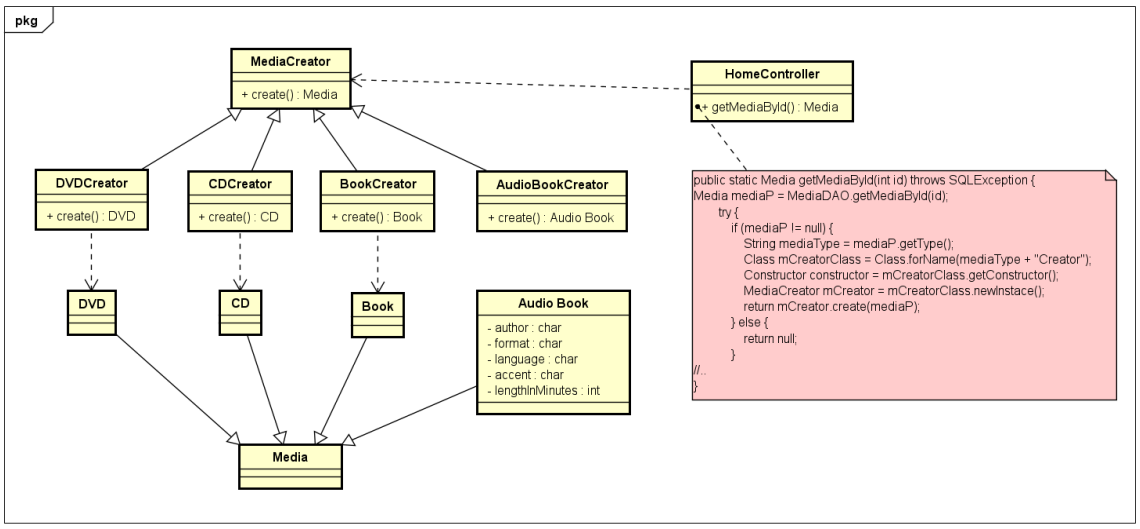
#	Vấn đề tồn tại	Danh sách class
1	Có các attribute được khai báo nhưng không được sử dụng	entity.media.Book entity.media.CD entity.media.DVD utils.Utils utils.ApplicationProgrammingInterface

2	Có các method được khai báo nhưng không được sử dụng	entity.media.Book entity.media.CD entity.media.DVD entity.media.Media entity.invoice.Invoice dao.media.MediaDAO entity.order.Order views.screen.popup.PopupScreen views.screen.shipping.ShippingScreenHandler
3	Có các import không được sử dụng	entity.media.Book views.screen.home.HomeScreenHandler App
4	Thiếu khoảng trắng giữa các ký tự syntax của ngôn ngữ (tuân thủ coding convention)	entity.cart.Cart entity.cart.CartItem common.exception. NotEnoughTransactionInfoException

Bảng 9. Clean Code - Clean Class

3. Đề xuất cải tiến

3.1. Vấn đề Thêm mặt hàng Media mới và giải pháp



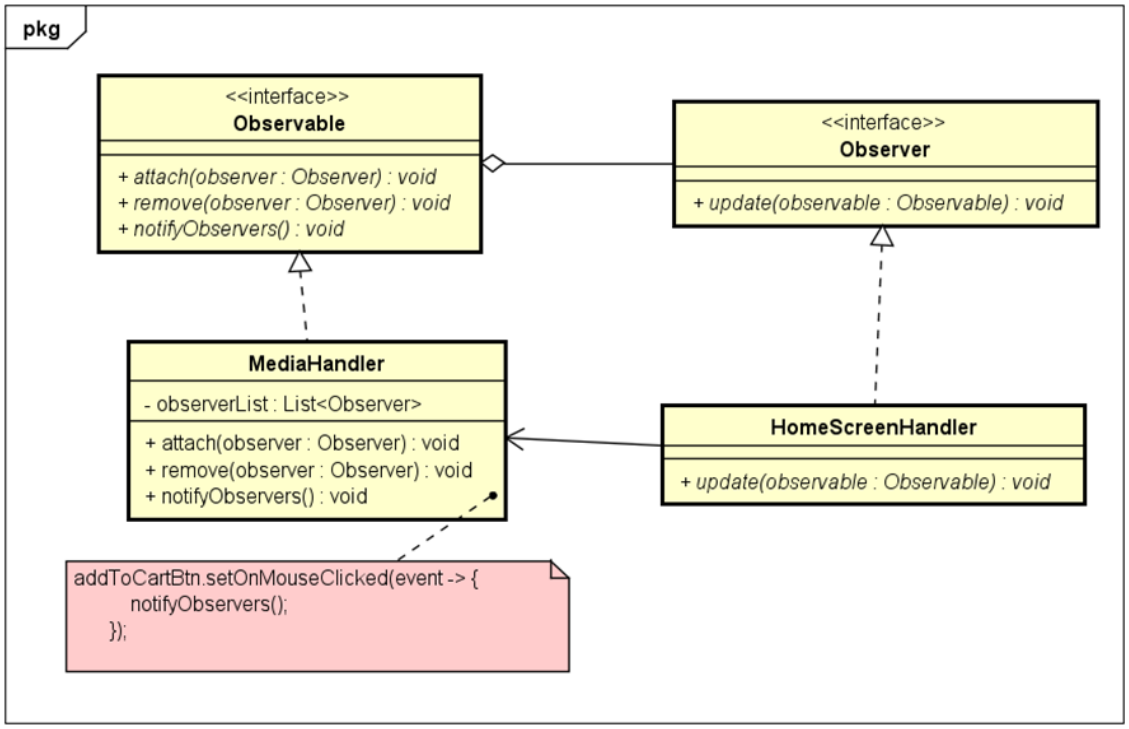
Hình 3. Biểu đồ lớp vấn đề Thêm mặt hàng Media

Ở đây, ta sử dụng Factory Method cho việc lấy thông tin của media theo từng loại. Ta tạo lớp Media Creator có các lớp con Book Creator, CD Creator, DVD Creator trả về đối tượng tương ứng với loại media. Tại HomeController, ta chỉ cần gọi lớp creator tương ứng với loại media. Từ đó ta sẽ lấy được thông tin loại media mà mình muốn.

Nếu muốn thêm 1 mặt hàng Media mới, ví dụ như AudioBook, ta chỉ cần thêm lớp AudioBook kế thừa Media và AudioBook Creator kế thừa Media Creator

3.2. Vấn đề Thêm màn hình: Xem chi tiết sản phẩm và giải pháp

- Vấn đề hiện tại:

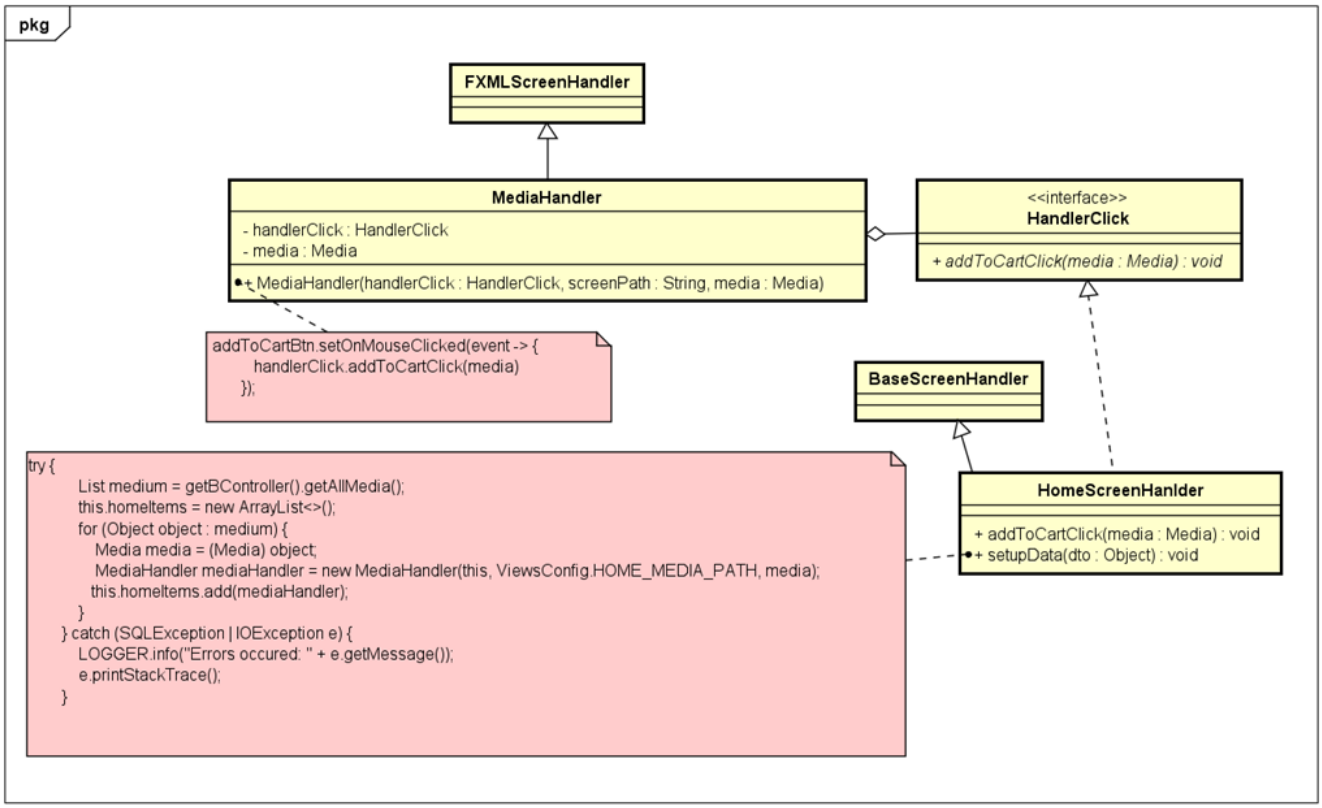


Hình 4. Biểu đồ lớp vấn đề Thêm màn hình - Xem chi tiết sản phẩm 1

- Áp dụng mẫu thiết kế Observer pattern có thể không phù hợp trong trường hợp này, vì hiện tại chỉ có lớp HomeScreenHandler attach MediaHandler mà không cần thiết phải áp dụng observer. Hơn nữa, MediaHandler hiện tại chỉ có thể notifyObservers() khi click vào button addToCartBtn. Trong tương lai, nếu có yêu cầu thêm về việc click, ví dụ như thêm button toDetailBtn, thì không thể xử lý được việc click button và thông báo cho HomeScreenHandler.
- Tuy nhiên, để giải quyết việc thêm button toDetailBtn mà vẫn áp dụng Observer pattern, ta có thể thêm tham số (tag, interface HandleClick) vào hàm update() của Observer và notifyObservers() của Observable để HomeScreenHandler biết sự thay đổi là do việc click nào tạo ra và sau đó xử lý. Tuy nhiên, thiết kế như vậy có thể dẫn tới control coupling trong hàm update().

Giải pháp:

- Một giải pháp khác là thay vì áp dụng Observer pattern để lắng nghe việc click button, ta có thể sử dụng interface HandleClick để HomeScreenHandler giao tiếp với MediaHandler. Trong đó, HomeScreenHandler sẽ implement interface HandleClick. Khi xảy ra sự kiện click bên MediaHandler, ta chỉ cần gọi phương thức tương ứng từ HandleClick để HomeScreenHandler xử lý.

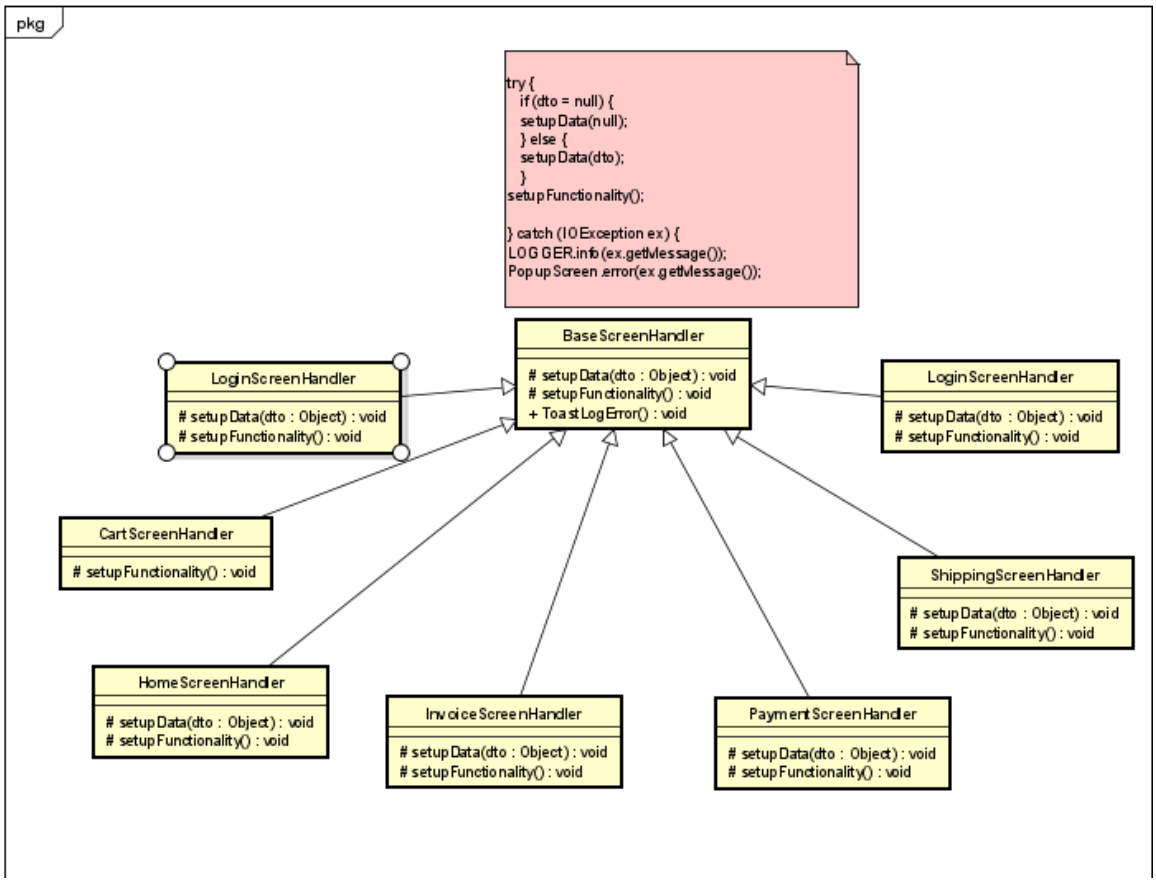


Hình 5. Biểu đồ lớp vấn đề Thêm màn hình - Xem chi tiết sản phẩm 2

3.3. Vấn đề Thay đổi yêu cầu khi load giao diện

Trong mã nguồn ban đầu, nếu chúng ta muốn thay đổi yêu cầu khi load giao diện, thay đổi cách xử lý khi xảy ra lỗi IOException, chúng ta cần vào tất cả các lớp ScreenHandler để có thể chỉnh sửa mã nguồn, điều này hạn chế khả năng tái sử dụng mã nguồn hiện tại.Để xử lý vấn đề này, ta sử dụng template method,gọi các phương thức setupData, setupFunctionality và thêm phương thức ToastLogError để hiển thị lỗi màu đỏ trong lớp BaseScreenHandler do các class dùng để xử lý logic giao diện

đang kế thừa nó.Khi đó nếu ta muốn sửa đổi trong các lớp ScreenHandler thì ta chỉ cần ghi đè (override) các phương thức trong template method



Hình 6. Biểu đồ lớp vấn đề 2 - Thay đổi yêu cầu khi load giao diện

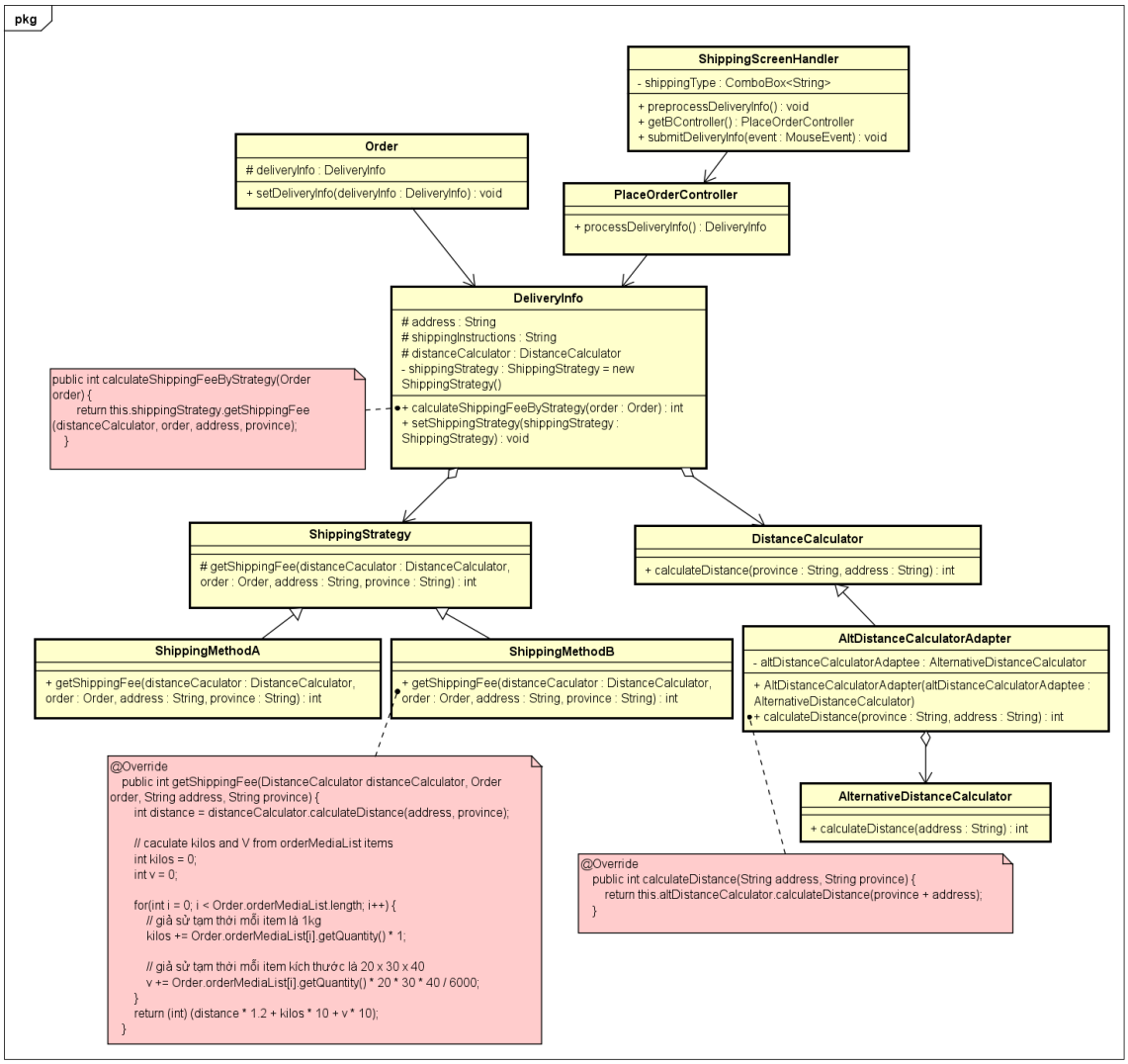
3.4. Vấn đề số 4 và 6: Thay đổi cách tính khoảng cách, sử dụng thư viện mới, thay đổi cách tính phí vận chuyển mới

Với yêu cầu về việc thay đổi công thức tính phí vận chuyển mới, ta đã sửa lại mã nguồn bằng cách thêm các lớp ShippingMethodB và ShippingMethodA kế thừa ShippingStrategy. Sử dụng strategy pattern cho 2 phương thức tính phí ship, trong đó ShippingMethodA là tính theo phí vận chuyển cũ, còn ShippingMethodB áp dụng cho công thức tính phí vận chuyển mới theo yêu cầu đề ra. Lớp strategy là ShippingStrategy sẽ được kết tập bên trong lớp ngữ cảnh là DeliveryInfo, phương tính vận chuyển sẽ được cái đặt qua phương thức setShippingStrategy() của lớp DeliveryInfo.

Với yêu cầu mở rộng về việc thay đổi cách tính khoảng cách theo thư viện tính khoảng cách mới, ta sử dụng Adapter object pattern. Trong đó, ta tạo ra lớp AltDistanceCalculatorAdapter với vai trò là lớp adapter, kế thừa lớp tính khoảng

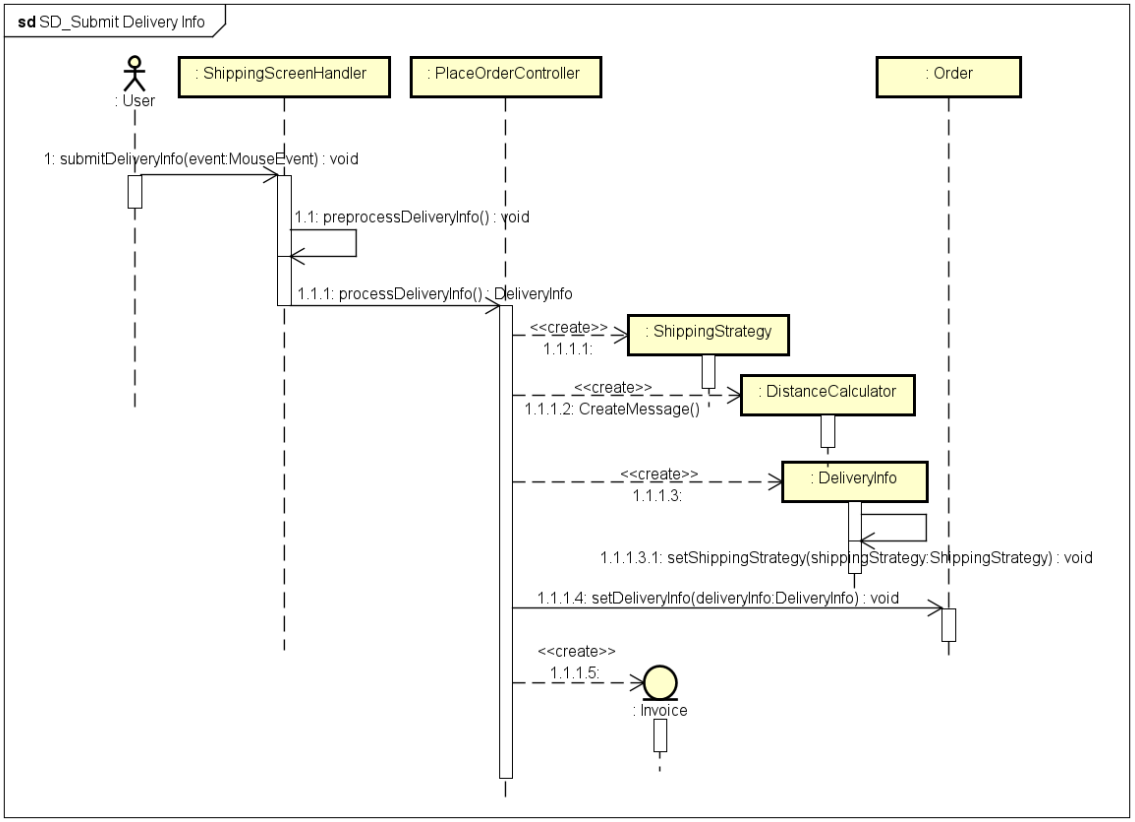
cách hiện tại là DistanceCalculator, đồng thời lớp AltDistanceCalculatorAdapter kết tập lớp tính khoảng cách theo thư viện mới là AlternativeDistanceCalculator (vai trò là lớp adaptee).

Dưới đây là biểu lớp sau đề xuất để cải thiện cho 2 vấn đề trên



Hình 7. Biểu đồ lớp vấn đề 4 và 6 -Thay đổi cách tính khoảng cách và thay đổi cách tính phí vận chuyển mới

- Biểu đồ tương tác

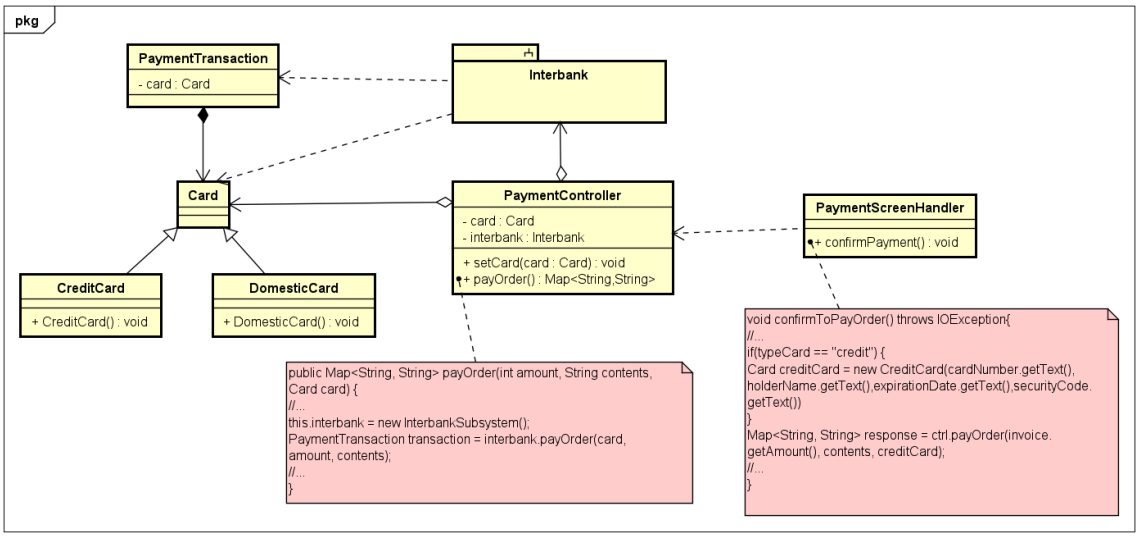


Hình 8. Biểu đồ tương tác vấn đề 4 và 6 -Thay đổi cách tính khoảng cách và thay đổi cách tính phí vận chuyển mới

3.5. Vấn đề Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)

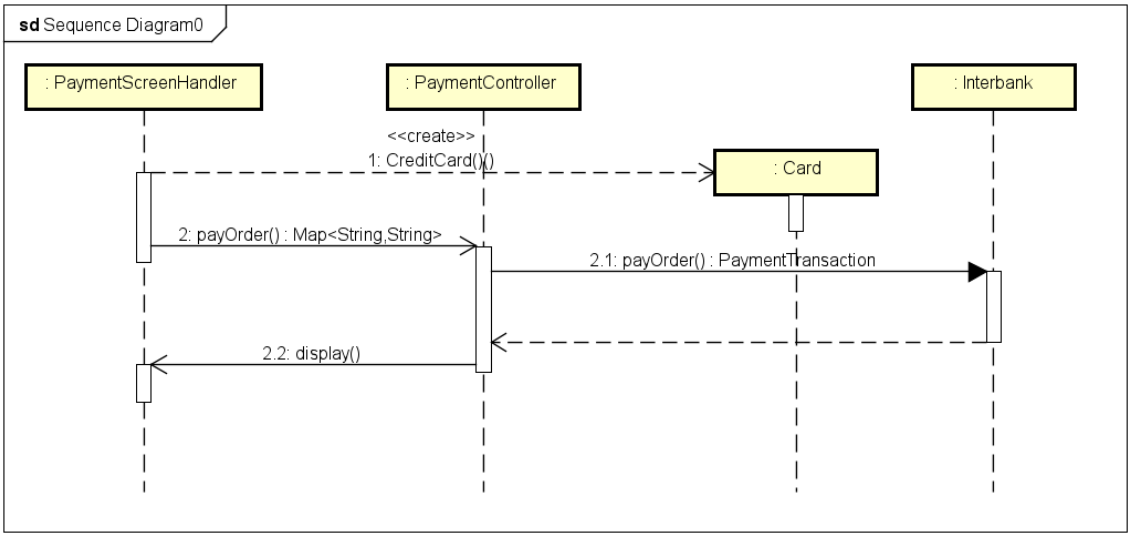
Ta refactor lại mã nguồn bằng cách tạo ra lớp Card mới và lớp CreaditCard kế thừa lớp Card. Nếu muốn thêm phương thức thanh toán mới như Domestic Card, ta chỉ cần thêm lớp DomesticCard kế thừa Card. Tại PaymentController, PaymentTransaction và InterbankSubsystem, thay vì truyền vào thông tin của Credit Card như cũ, ta chỉ cần truyền vào 1 instance của Card, tại PaymentScreenHandler ta sẽ xử lý xem loại thanh toán là gì và tạo đối tượng Card tương ứng rồi truyền cho PaymentController xử lý.

Biểu đồ lớp:



Hình 9. Biểu đồ lớp vấn đề 5 - Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)

Biểu đồ tương tác:

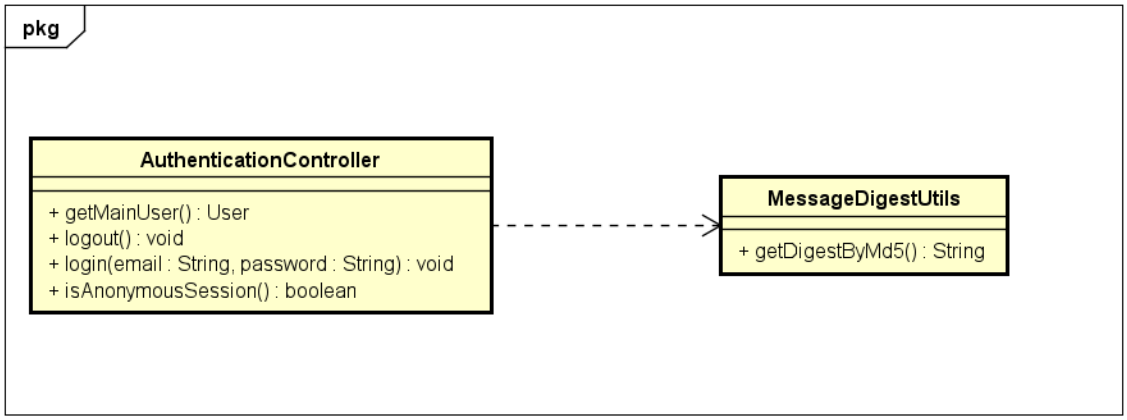


Hình 10. Biểu đồ tương tác vấn đề 5 - Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)

3.6. Giải quyết vấn đề vi phạm nguyên lý Single Responsibility Principle trong class AuthenticationController

Với thiết kế hiện tại, nhiệm vụ của class AuthenticationController là thực hiện các công việc liên quan tới xác thực người dùng. Nhưng trong class này hiện đang có method md5 chỉ có nhiệm vụ sinh message digest sử dụng hàm băm md5. Việc thiết kế như vậy đang vi phạm nguyên lý Single Responsibility Principle. Để khắc phục vấn đề này, chúng ta nên tạo riêng một class MessageDigestUtils riêng chỉ để thực

hiện nhiệm vụ sinh message digest từ message ban đầu và cài đặt method md5 ở class này. Khi đó, ta vừa có thể tái sử dụng method md5 ở nhiều nơi trong hệ thống nếu cần thiết, vừa có thể dễ dàng bổ sung thêm các method sử dụng các thuật toán khác để sinh message digest. Thiết kế sau khi refactor sẽ được mô tả trong biểu đồ class như sau:

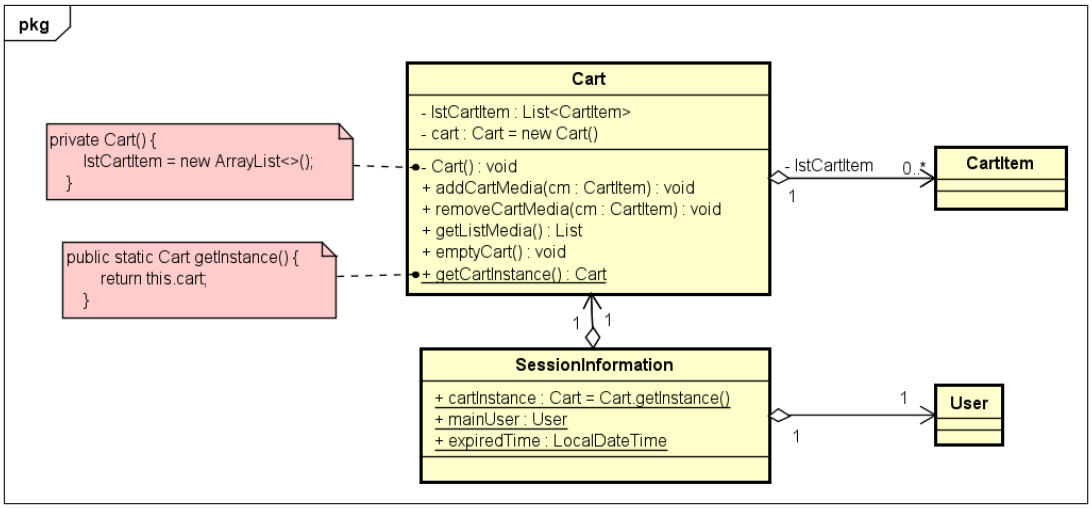


Hình 11. Biểu đồ lớp vấn đề vi phạm nguyên lý SRP trong class AuthenticationController

3.7. Giải quyết vấn đề tạo nhiều instance của lớp Cart mà không cần thiết bằng Singleton Pattern

Với thiết kế hiện tại, ta có thể tạo nhiều đối tượng lớp Cart tùy thích, điều này là không đúng vì thực tế thì mỗi người dùng từ đầu đến cuối chỉ có duy nhất một giỏ hàng mà thôi, hay nói cách khác, ta chỉ cần tạo một instance của lớp Cart xuyên suốt quá trình sử dụng. Để giải quyết vấn đề này, ta sử dụng Singleton Pattern. Phương thức khởi tạo sẽ cho là private, ta chỉ có thể khởi tạo đối tượng Cart bên trong lớp đó và mong muốn khởi tạo một lần duy nhất (ở đây nhóm lựa chọn cách làm khởi tạo đối tượng cart ngay tại thời điểm biên dịch). Các lớp khác muốn sử dụng đối tượng cart duy nhất này sẽ cần thông qua phương thức static getInstance() của lớp Cart.

Dưới đây là thiết kế để khắc phục vấn đề trên



Hình 12. Biểu đồ lớp vấn đề tạo nhiều instance của lớp Cart mà không cần thiết bằng Singleton Pattern

4. Tổng kết

Sau khi tiến hành tái cấu trúc mã nguồn, tuy vẫn còn một số hạn chế, nhưng thiết kế của hệ thống cơ bản đã đáp ứng được các nguyên lý SOLID, thích ứng được khi có hệ thống xảy ra các yêu cầu thay đổi phát sinh trong tương lai. Các vấn đề về clean name, clean method, clean class cũng đã được khắc phục. Các vấn đề về Coupling và Cohesion như đã phân tích ở trên gần như cũng được xử lý, đảm bảo hệ thống không còn bị vi phạm. Việc tiến hành tái cấu trúc một mã nguồn hệ thống hiện có giúp các thành viên trong nhóm có cơ hội áp dụng kiến thức được học trên lớp vào sản phẩm thực tế, biết cách xác định các vấn đề cần áp dụng mẫu thiết kế phần mềm để xử lý, cài đặt và triển khai các mẫu thiết kế.

4.3. Kết quả tổng quan

So với dự kiến ban đầu, kết quả thực hiện đã hoàn thiện được hầu hết các vấn đề mà nhóm đã tìm và đưa ra, tuy nhiên vẫn còn một số vấn đề, hạn chế tồn tại, hoặc nhóm chưa thực hiện được một cách tốt nhất.

4.4. Các vấn đề tồn đọng

Như đã nói, vẫn còn một số vấn đề, hạn chế mà nhóm chưa thực hiện được, ví dụ như yêu cầu thêm số 7. Một số pattern nhóm chưa thể tìm ra được giải pháp và đưa vào việc cải tiến mã nguồn. Bên cạnh đó, việc thay đổi mã nguồn mặc dù đã làm giảm đáng kể và tốt hơn nhiều so với thiết kế cũ, tuy nhiên, tại một số chỗ ta vẫn phải chịu vấn đề về control coupling, tuy nhiên việc xảy ra control coupling ở các lớp giao diện, UI là vấn đề có thể chấp nhận được. Ngoài ra, quá trình sửa đổi mã nguồn, cũng có những phát sinh, tồn đọng mà nhóm đã bỏ sót hoặc chưa phát hiện, kiểm soát, kính mong cô và mọi người thông cảm

4.5. Lời cảm ơn

Cuối cùng, chúng em xin chân thành cảm ơn cô Bùi Thị Mai Anh và cô Nguyễn Thị Thu Trang đã có những buổi chia sẻ kiến thức vô cùng bổ ích, có những góp ý, hướng dẫn tận tình để chúng em có thể hiểu được những vấn đề, những nguyên lý, những mẫu thiết kế áp dụng cho việc phát triển phần mềm. Cảm ơn cô đã giúp chúng em có những trải nghiệm về việc áp dụng những kiến thức về thiết kế xây dựng một phần mềm trong thực tế và đặc biệt là hoàn thành bài tập lớn lần này. Cuối cùng, chúng em xin chân thành cảm ơn hai cô và chúc hai cô luôn luôn mạnh khỏe, giữ mãi nhiệt huyết và truyền động lực, kiến thức cho thế hệ sinh viên chúng em.