ĐẠI HỌC BÁCH KHOA HÀ NỘI

Trường Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

Version 1.0

Nhóm 5

Danh sách thành viên

Nguyễn Hồng Sơn	20194156
Lê Hồng Ưng	20194211
Thân Minh Nam	20194128
Hồ Hải Nam	20194123

Hà Nội, tháng 7 năm 2023

Mục lục

Danh sách các minh họa

Danh sách các bảng

1 Tổng quan

1.1 Mục tiêu

Tài liệu báo cáo mẫu thiết kế phần mềm là tài liệu được sử dụng để tổng hợp công việc, quá trình làm việc và kết quả đạt được của nhóm trong quá trình hoàn thiện project của học phần IT4536 – Mẫu thiết kế phần mềm. Đối tượng người đọc của tài liệu là Giảng viên, thành viên trong nhóm và những người có kiến thức cơ sở về lập tình hướng đối tượng, thiết kế hệ thống phần mềm. Người đọc có thể xem tài liệu này là một cách giải quyết cho một bài toán thực tế, sử dụng làm tài liệu tham khảo. Mọi đóng góp của người đọc có thể được gửi qua email sonnguyenhong291@gmail.com. Trong quy trình xử lý phần mềm, người đọc tài liệu này có thể là người quản lý kỹ thuật và những người phát triển trong dự án.

Nội dung khái quát của báo cáo bao gồm trình bày phạm vi của đề tài, đánh giá thiết kế cũ của codebase, chỉ ra những vi phạm về nguyên tắc thiết kế, đưa ra giải pháp khắc phục những vi phạm trên và chỉ rõ lý do tại sao lại sử dụng giải pháp này.

1.2 Phạm vi

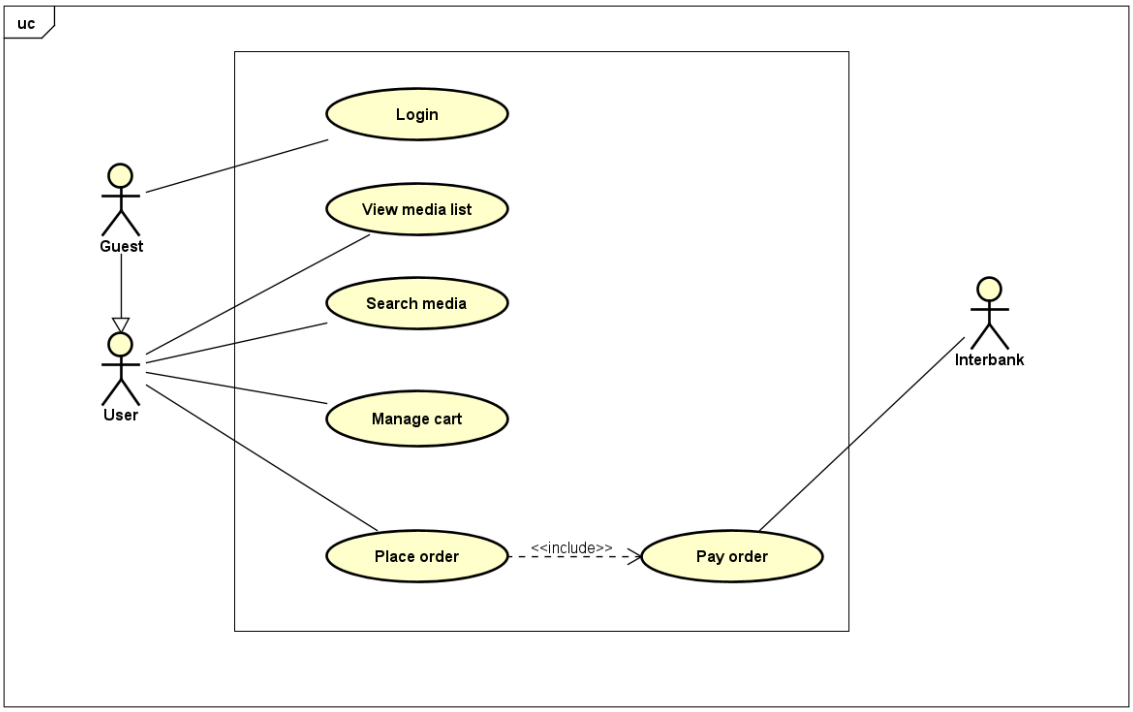
1.2.1 Mô tả khái quát phần mềm

Phần mềm AIMS là một phần mềm bán hàng online, chuyên bán các mặt hàng về Media (Book, CD, DVD). Người dùng có thể đặt mua hàng online thông qua phần mềm. Sau khi đặt hàng, người dùng có thể thanh toán online thông qua thẻ tín dụng. Sau đó đơn hàng sẽ được đơn vị vận chuyển giao cho người dùng tới địa chỉ mà họ cung cấp. Trong codebase ban đầu, phần mềm đã cơ bản đầy đủ các chức năng. Tuy nhiên, codebase còn nhiều vi phạm các nguyên lý thiết kế hướng đối tượng, khiến cho việc sửa đổi, mở rộng phần mềm trở nên khó khan khi có yêu cầu thay đổi có thể phát sinh trong tương lai.

1.2.2 Các chức năng chính của phần mềm

Các chức năng chính của phần mềm:

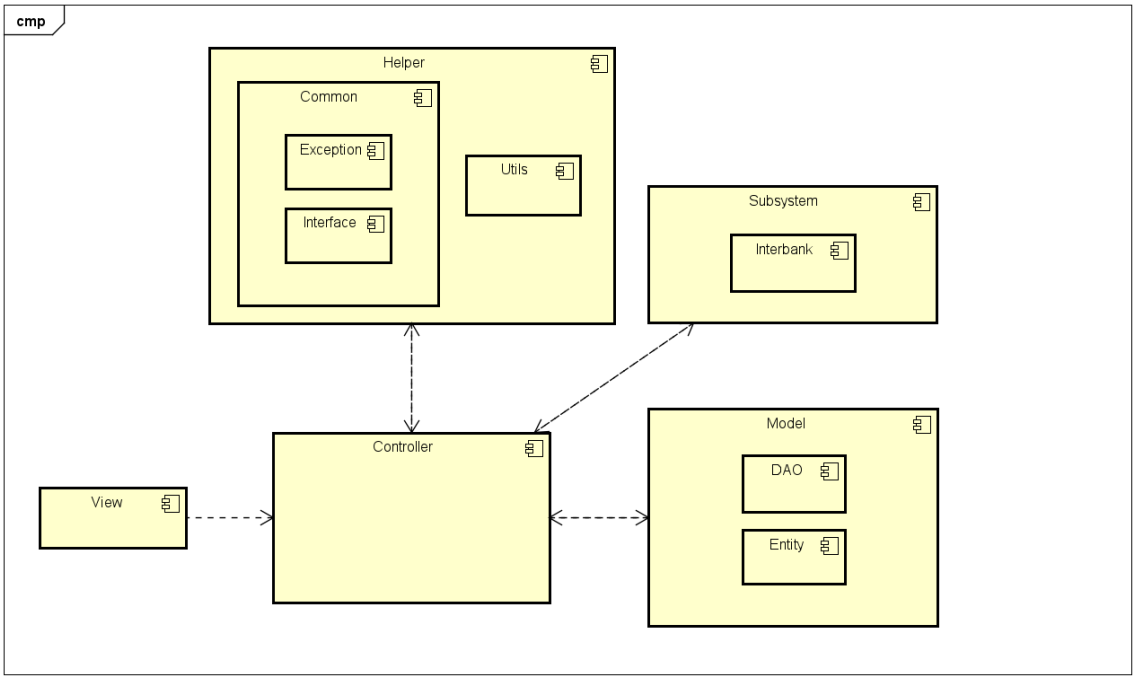
- Xem danh sách sản phẩm, tìm kiếm sản phẩm có trong cửa hàng
- Quản lý giỏ hàng
- Thực hiện mua hàng, thanh toán đơn hàng online



Hình 1. Biểu đồ usecase tổng quan AIMS

Hệ thống AIMS có các tác nhân: Guest User, User và Interbank. Guest User và User đều có thể thực hiện được các chức năng chính của phần mềm như Xem danh sách sản phẩm, tìm kiếm sản phẩm, quản lý giỏ hàng, đặt hàng. Usecase Pay order (thanh toán đơn hàng) được include bên trong usecase Place order (Đặt hàng) thể hiện đây là hành vi bắt buộc của người dùng nếu muốn hoàn thành usecase Place order. Muốn hoàn thành usecase Pay order này, hệ thống cần phải có sự tham gia tương tác của tác nhân Interbank, hỗ trợ việc thanh toán online qua ngân hàng.

1.2.3 Cấu trúc mã nguồn



Hình 2. Biểu đồ thành phần của hệ thống

Cấu trúc mã nguồn của hệ thống AIMS được thiết kế theo mô hình MVC (Model – View – Controller). Ở tầng View, mã nguồn bao gồm các đoạn mã giao diện xử lý việc hiển thị giao diện và xử lý logic ở màn hình cho người dùng. Thông tin của người dùng truyền từ tầng View sẽ được chuyển để tầng Controller để thực hiện xử lý logic, truy xuất dữ liệu tại tầng Model. Dữ liệu từ tầng Model trả về cho tầng Controller sẽ được trả về cho tầng View để xử lý và hiển thị kết quả cho người dùng. Ngoài 3 thành phần chính là Model – View – Controller, hệ thống còn có các thành phần bổ trợ như Subsystem (thể hiện các hệ thống con, ở đây bao gồm hệ thống Interbank phụ trách nhiệm vụ hỗ trợ thanh toán online) và các module Helper (chứa các chức năng chung để xử lý logic cho hệ thống).

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

Trong quá trình tái cấu trúc, mã nguồn cần được xử lý sao cho thỏa mãn tính high cohension và low coupling, đảm bảo tính clean code, không vi phạm các nguyên lý thiết kế như SOLID, Law of Demeter,… Ngoài ra, mã nguồn cần áp dụng các Mẫu thiết kế đã được học như Singleton, Template Method, Factory Method, Strategy, Observer, Adapter, có thể sử dụng các Mẫu thiết kế khác nhưng cần phải giải thích lý do tại sao sử dụng những mẫu thiết kế này.

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

Các hoạt động review mã nguồn:

- Xác định mức độ coupling và cohension
- Xác định các vi phạm nguyên lý SOLID

Các hoạt động refractor mã nguồn:

- Áp dụng các mẫu thiết kế để đưa ra thiết kế mới, phù hợp với các yêu cầu mở rộng có thể xảy ra trong tương lai.

1.2.6 Kết quả dự kiến

Mã nguồn sau khi được thực hiện review và refrator sẽ đảm bảo tính high cohension và low coupling, đảm bảo tính chất clean code, tuân thủ các nguyên tắc thiết kế như SOLID, Law of Demeter, …, có thể mở rộng dễ dàng nhờ vào việc áp dụng các Mẫu thiết kế.

1.3 Danh sách thuật ngữ

Số thứ tự	Thuật ngữ	Giải thích
1	Message digest	Hàm băm mật mã trả về một chuỗi ký tự sử dụng hàm băm một chiều
2	SOLID	Là tập hợp gồm 5 nguyên lý thiết kế: Single Responsibility, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle
3	MVC	Kiến trúc thiết kế Model – View - Controller

1.4 Danh sách tham khảo

- Tài liệu học phần Mẫu thiết kế phần mềm IT4536.
- Sách Head First Design Pattern: A Brain-Friendly Guide, O'Reilly

2 Đánh giá thiết kế cũ

2.1 Nhận xét chung

Mã nguồn của hệ thống phần mềm AIMS hiện tại cơ bản đã đáp ứng đầy đủ các chức năng yêu cầu cần thiết cho một hệ thống bán hàng. Tuy nhiên, với các yêu cầu mở rộng có thể phát sinh trong tương lai thì mã nguồn hiện tại chưa thể đáp ứng được việc mở rộng dễ dàng và hiệu quả. Mã nguồn hiện tại vẫn còn vi phạm các mức độ Coupling và Cohension, các nguyên lý thiết kế SOLID. Việc thay đổi hoặc mở rộng mã nguồn hiện tại để đáp ứng các yêu cầu mở rộng trở nên khó khăn do các module phụ thuộc lẫn nhau, việc thay đổi một module sẽ dẫn tới sự thay đổi của module khác khiến chúng ta khó kiểm soát được các sự thay đổi này. Các module nên phụ thuộc lẫn nhau ở mức độ trừu tượng, chỉ quan tâm đến các chức năng mà module khác cung cấp chứ không quan tâm đến cài đặt chi tiết của module đó. Thiết kế này sẽ giúp việc mở rộng hệ thống dễ dàng hơn.

2.2 Đánh giá các mức độ coupling và cohesion

Coupling và Cohension là 2 khái niệm được sử dụng để đánh giá một thiết kế phần mềm có tốt hay không. Một thiết kế được đánh giá là tốt nếu nó đáp ứng được các yêu cầu:

- Dễ dàng phát triển, kiểm thử
- Dễ dàng đọc hiểu mã nguồn, thiết kế
- Dễ dàng giao tiếp giữa các module
- Dễ dàng đáp ứng cho việc mở rộng các yêu cầu mới
- Dễ dàng bảo trì.

Mọi nguyên lý thiết kế hay các mẫu thiết kế được đưa ra đều để có thể đáp ứng được 2 yếu tố đó là Low Coupling và High Cohension.

Coupling là mức độ phụ thuộc lẫn nhau giữa các modules. Một thiết kế tốt là một thiết kế mà các module có sự phụ thuộc trực tiếp lẫn nhau một cách thấp nhất (Low Coupling). Các mức độ Coupling được sắp xếp từ thấp đến cao là:

- Data Coupling
- Stamp Coupling
- Control Coupling
- Common Coupling
- Content Coupling

Cohension là mức độ gắn kế giữa các thành phần trong một module. Một thiết kế tốt là một thiết kế mà các thành phần trong cùng một module có sự gắn kết với nhau một cách cao nhất (High Cohension). Các mức độ Cohension được sắp xếp từ thấp đến cao là:

- Coincidental
- Logical
- Temporal
- Procedural
- Communicational
- Functional

2.2.1 Coupling

#	*Các mức độ về Coupling*	*Module*	*Mô tả*	*Lý do*
1	Data Coupling	controller.AuthenticationController	Hàm login() chứa tham số email, là một thuộc tính của User	Hàm login() chỉ cần truyền vào thuộc tính email của đối tượng user thay vì truyền cả vào một đối tượng user.
2	Stamp Coupling	entity.cart.Cart	Hàm checkMediaInCart(Media media) nhận tham số đầu vào là 1 đối tượng media	Hàm checkMediaInCart chỉ sử dụng thuộc tính id của đối tượng media nhưng lại truyền vào 1 đối tượng media luôn.
3	Control Coupling	Không có	Không có	Không có
4	Common Coupling	controller.SessionInformation	Các biến mainUser và cartInstance là các biến global có thể được truy cập và sửa	Chúng ta khó kiểm soát được những module nào đã truy cập và sửa đổi lên dữ liệu đó. Do đó giảm khả năng bảo trì hay tái sử dụng.

			đổi bởi nhiều module	
5	Content Coupling	entity.cart.Cart	Lớp Cart có phương thức getListMedia() trả về tham chiếu đến ArrayList lstCartItem	Module khác có thể sử dụng hàm này để lấy tham chiếu đến lstCartItem và thực hiện thay đổi lên lstCartItem. Giải pháp là hàm getListMedia() nên trả về một bản copy của lstCartItem.
		entity.shipping.DeliveryInfo	Có thuộc tính protected	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		entity.media.Book	Có thuộc tính default	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		entity.media.CD	Có thuộc tính default	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		entity.media.DVD	Có thuộc tính default	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		entity.media.Media	Có thuộc tính protected	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		controller.SessionInformation	Có thuộc tính public static	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		entity.order.Order	Có thuộc tính protected ở trường deliveryInfo	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng

		entity.payment.CreditCard	Có thuộc tính protected ở trường cvvCode	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		entity.shipping.ShippingConfigs	Có thuộc tính public	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		subsystem.interbank.InterbankConfigs	Có thuộc tính public	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		utils.ApplicationProgrammingInterface	Có thuộc tính public	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		utils.Utils	Có thuộc tính public	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.cart.CartScreenHandler	Có thuộc tính default	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.cart.MediaHandler	Có thuộc tính protected	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.home.Logi	Có thuộc tính public	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng

		nScreen Handler		
		views.screen.home.MediaHandler	Có thuộc tính protected	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.popup.PopupScreen	Có thuộc tính default	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.BaseScreenHandler	Có thuộc tính protected	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.FXMLScreenHandler	Có thuộc tính protected	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		views.screen.ViewsConfig	Có thuộc tính public	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng
		App	Có thuộc tính default	Module khác có thể truy cập trực tiếp vào các thuộc tính này và sửa đổi chúng

Bảng 1. Các mức độ Coupling trong mã nguồn

2.2.2 Cohesion

#	*Các mức độ về Cohesion*	*Module*	*Mô tả*	*Lý do*

1	Coincidental	utils.Utils	Biến DATE_FORMATTER và LOGGER được khai báo nhưng không được sử dụng và cũng không có quan hệ gì với nhau.	Hai biến được khai báo trong cùng một lớp nhưng không có liên quan gì đến nhau.
		controller.AuthenticationController	Method md5 có nhiệm vụ sinh message digest, không liên quan tới nghiệp vụ thực hiện xác minh người dùng	Phương thức md5 được gộp vào trong lớp AuthenticationController nhưng không có nghiệp vụ liên quan đến Authentication
2	Logical	views.screen.popup.PopupScreen	Các hàm success, error, loading thực hiện các cách hiển thị Popup khác nhau	Các hàm success, errror, loading được gộp chung vào một class vì chúng có liên quan với nhau một cách logic nhưng cách thức thực hiện khác nhau.
3	Temporal	views.screen	Các phương thức setupData và setupFunctionality	Hai phương thức này được gộp vào trong một lớp vì chúng có liên quan tới nhau trong thời gian chạy.
4	Procedural	views.screen.cart.MediaHandler	Các phương thức setCartItem, setMediaInfo,	Các phương thức có liên quan tới nhau vì chúng được thực hiện theo một thứ tự: setCartItem =>

			initializeSpinner	setMediaInfo => initializeSpinner
5	Communicational	Không có	Không có	Không có
6	Functional	Không có	Không có	Không có

Bảng 2. Các mức độ Cohension trong mã nguồn

2.3 Đánh giá việc tuân theo SOLID

Hiện tại hệ thống cần đáp ứng 7 yêu cầu phát sinh

- Thêm mặt hàng Media mới (AudioBook): Với yêu cầu này, bản thiết kế và mã nguồn ban đầu đã đáp ứng được việc mở rộng thêm Media mới.
- Thêm màn hình xem chi tiết sản phẩm: Với yêu cầu này, bản thiết kế và mã nguồn ban đầu không bị vi phạm về nguyên lý SOLID
- Thay đổi yêu cầu khi load giao diện: Với yêu cầu này, bản thiết kế và mã nguồn hiện tại đang vi phạm nguyên lý OCP. Khi chúng ta cần thay đổi cách xử lý khi gặp lỗi IOException thì phải mở lại mã nguồn của các lớp ScreenHandler để sửa đổi.
- Thay đổi cách tính khoảng cách, sử dụng thư viện mới: Với yêu cầu này, bản thiết kế và mã nguồn hiện tại đang vi phạm nguyên lý OCP vì modules phụ thuộc trực tiếp vào thư viện tính khoảng cách, nếu thay đổi thư viện thì phải sửa mã nguồn.
- Thêm phương thức thanh toán mới (Thẻ nội địa – Domestic Card): Với yêu cầu này, bản thiết kế và mã nguồn hiện tại đang vi phạm nguyên lý OCP và DIP vì modules phụ thuộc trực tiếp vào phương thức thanh toán CreditCard, nếu chúng ta thêm phương thức thanh toán mới thì phải sửa mã nguồn trong những module đó.
- Thay đổi công thức tính phí vận chuyển: Với yêu cầu này, bản thiết kế và mã nguồn hiện tại đang vi phạm nguyên lý OCP và DIP vì việc thay đổi công thức tính phí vận chuyển sẽ phải sửa mã nguồn tại các module phụ trách việc đó.
- Cập nhật chức năng hủy đơn hàng: Việc cập nhật lại chức năng hủy đơn hàng vi phạm nguyên lý OCP vì chúng ta phải sửa đổi mã nguồn trong module thực hiện nhiệm vụ đó.

2.3.1 SRP

#	*Module*	*Mô tả*	*Lý do*
1	subsystem.interbank.InterbankPayloadConverter	Lớp InterbankPayloadConverter chứa các phương thức convertToRequestPayload, extractPaymentTransaction, convertJSONResponse, getToday	Các phương thức extractPaymentTransaction và getToday không nằm trong nghiệp vụ Convert.
2	utils.Utils	Biến DATE_FORMATTER và LOGGER được khai báo nhưng không được sử dụng và cũng không có quan hệ gì với nhau.	Hai biến được khai báo trong cùng một lớp nhưng không có liên quan gì đến nhau.
3	utils.ApplicationProgrammingInterface	Các phương thức setUpConnection và allowMethods	Lớp ApplicationProgrammingInterface chỉ nên chứa các phương thức get, post, ... nên tách thành các class ConnectionManager và AccessManager.
4	entity.shipping.DeliveryInfo	Lớp DeliveryInfo đang có chứa một method dùng để tính toán chi phí vận chuyển.	Lớp DeliveryInfo nên chỉ có nhiệm vụ chứa các thông tin về đơn hàng như tên người nhận, địa chỉ, hướng dẫn giao hàng… chứ không nên có cả nhiệm vụ tính toán chi phí vận chuyển.

5	controller.AuthenticationController	Lớp AuthenticationController đang có một method để sinh mã băm sử dụng hàm MD5	Lớp AuthenticationController nên chỉ thực hiện các nhiệm vụ Authentication, không nên có cả nhiệm vụ sinh mã băm.
6	controller.PlaceOrderController	Lớp PlaceOrderController đang có các method để thực hiện validate dữ liệu người dùng nhập vào.	Lớp PlaceOrderController nên chỉ thực hiện các nhiệm vụ về đặt hàng thay vì có thêm cả nhiệm vụ validate dữ liệu đầu vào.

Bảng 3. Vi phạm SRP trong mã nguồn

2.3.2 OCP

#	*Module*	*Mô tả*	*Lý do*
1	subsystem.interbank.InterbankPayloadConverter	Lớp InterbankPayloadConverter có chứa câu lệnh switch ... case ... để kiểm tra mã lỗi và trả về lỗi tương ứng	Khi có thêm một mã lỗi mới thì chúng ta phải mở lại mã nguồn của lớp InterbankPayloadConverter để chỉnh sửa mã nguồn
2	controller.PaymentController, subsystem.InterbankInterface và subsystem.InterbankSubsystem	Các lớp này phụ thuộc trực tiếp vào CreditCard	Trong tương lai nếu chúng ta có thêm một phương thức thanh toán khác ngoài CreditCard thì chúng ta sẽ phải sửa đổi trong các lớp này
3	controller.PlaceOrderController	Lớp PlaceOrderController này phụ thuộc trực tiếp	Trong tương lai nếu chúng ta đổi thư viện tính khoảng cách thì chúng ta

		vào class DistanceCaculator	sẽ phải sửa đổi bên trong lớp PlaceOrderController

Bảng 4. Vi phạm OCP trong mã nguồn

2.3.3 LSP

Mã nguồn hiện tại không có vi phạm nguyên lý LSP.

2.3.4 ISP

Mã nguồn hiện tại không có vi phạm nguyên lý ISP.

2.3.5 DIP

#	*Module*	*Mô tả*	*Lý do*
1	controller.PaymentController, subsystem.InterbankInterface và subsystem.InterbankSubsystem	Các lớp này phụ thuộc trực tiếp vào lớp CreditCard	Trong tương lai nếu chúng ta có thêm một phương thức thanh toán khác ngoài CreditCard thì chúng ta sẽ phải sửa đổi trong các lớp này
2	controller.PlaceOrderController	Lớp PlaceOrderController này phụ thuộc trực tiếp vào class DistanceCaculator	Trong tương lai nếu chúng ta đổi thư viện tính khoảng cách thì chúng ta sẽ phải sửa đổi bên trong lớp PlaceOrderController

Bảng 5. Vi phạm DIP trong mã nguồn

2.4 Các vấn đề về Clean Code

2.4.1 Clear Name

Mã nguồn hiện tại đã đáp ứng được nguyên tắc clear name. Các tên lớp, tên biến, thuộc tính và phương thức đã mang ý nghĩa giúp chúng ta dễ hiểu khi đọc code. Tuy nhiên, có một số tên biến vi phạm nguyên tắc cần sửa đổi để giúp người đọc dễ hiểu hơn:

#	Module	Vị trí có tên chưa clear	Đề xuất sửa đổi

1	controller. PaymentController	Phương thức getExpirationDate có tên biến strs	Sửa tên biến strs thành dateSplit
2	Controller. AuthenticationController	Phương thức md5 có tên biến md	Sửa tên phương thức thành messageDigest5 và tên biến thành messageDigest
3	views.screen.home. HomeScreenHandler	Phương thức setupData có tên biến m	Sửa tên biến thành mediaHandler
4	views.screen.home. HomeScreenHandler	Phương thức setImage có tên biến file1, img1, file2, img2	Sửa tên biến thành fileLogo, imgLogo, fileCart, imgCart
5	views.screen.cart. CartScreenHandler	Phương thức displayCartWithMediaAvailability() chứa tên biến cm	Sửa tên biến thành cartMedia
6	Subsystem. InterbankSubsystem	Có thuộc tính ctrl thuộc lớp InterbankSubsystemController	Sửa tên biến thành interbankSubsystemController

Bảng 6. Vi phạm clear name trong mã nguồn

2.4.2 Clean Function/Method

Trong mã nguồn ban đầu, đa số các function và method đã đáp ứng được các yêu cầu về clean function/method. Tuy nhiên chúng ta cần sửa đổi một số vị trí sau để có được mức độ clean cao hơn.

#	Module	Lý do	Đề xuất sửa đổi
1	views.screen.cart. CartScreenHandler	Phương thức requestToPlaceOrder có mức trừu tượng cao (High Level of Abstraction) (create	Tách từng phần nhiệm vụ của lớp thành các hàm riêng

		placeOrderController and process the order, display available media, create order, display shipping form)	
2	views.screen.cart. CartScreenHandler	Phương thức updateCartAmount() có mức trừu tượng cao (High Level of Abstraction) (calculate subtotal and amount, update subtotal and amount of Cart)	Tách từng phần nhiệm vụ của lớp thành các hàm riêng

Bảng 7. Vi phạm clear function/method trong mã nguồn

2.4.3 Clean Class

Trong mã nguồn, vấn đề về clean class chủ yếu xảy ra ở những lớp vi phạm nguyên lý SRP. Các lớp thực hiện nhiều nhiệm vụ trong cùng một class, các nhiệm vụ không liên quan tới nhau.

#	*Module*	*Mô tả*	*Lý do*
1	subsystem.interbank.InterbankPayloadConverter	Lớp InterbankPayloadConverter chứa các phương thức convertToRequestPayload, extractPaymentTransaction, convertJSONResponse, getToday	Các phương thức extractPaymentTransaction và getToday không nằm trong nghiệp vụ Convert.
2	utils.Utils	Biến DATE_FORMATTER và LOGGER được khai báo nhưng không	Hai biến được khai báo trong cùng một lớp nhưng không có liên quan gì đến nhau.

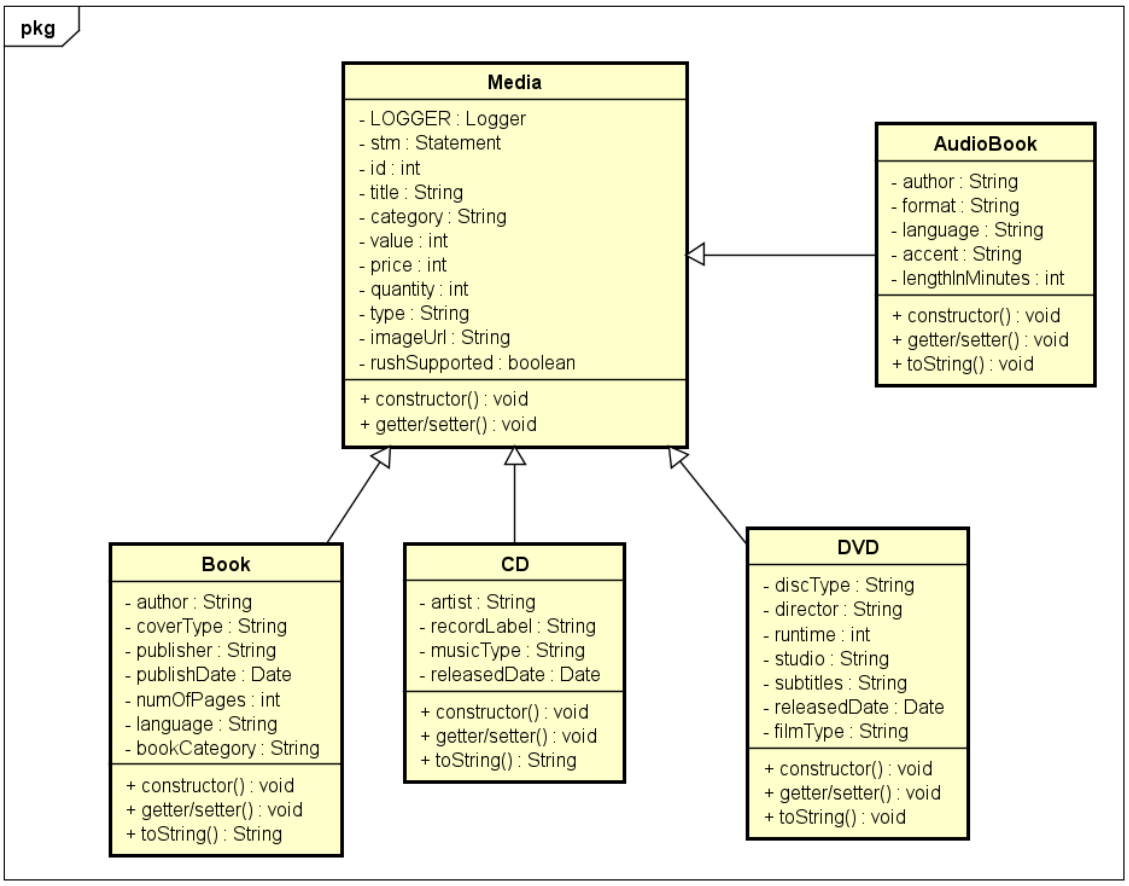
		được sử dụng và cũng không có quan hệ gì với nhau.	
3	utils.ApplicationProgrammingInterface	Các phương thức setUpConnection và allowMethods	Lớp ApplicationProgrammingInterface chỉ nên chứa các phương thức get, post, ... nên tách thành các class ConnectionManager và AccessManager.
4	entity.shipping.DeliveryInfo	Lớp DeliveryInfo đang có chứa một method dùng để tính toán chi phí vận chuyển.	Lớp DeliveryInfo nên chỉ có nhiệm vụ chứa các thông tin về đơn hàng như tên người nhận, địa chỉ, hướng dẫn giao hàng… chứ không nên có cả nhiệm vụ tính toán chi phí vận chuyển.
5	controller.AuthenticationController	Lớp AuthenticationController đang có một method để sinh mã băm sử dụng hàm MD5	Lớp AuthenticationController nên chỉ thực hiện các nhiệm vụ Authentication, không nên có cả nhiệm vụ sinh mã băm.
6	controller.PlaceOrderController	Lớp PlaceOrderController đang có các method để thực hiện validate dữ liệu người dùng nhập vào.	Lớp PlaceOrderController nên chỉ thực hiện các nhiệm vụ về đặt hàng thay vì có thêm cả nhiệm vụ validate dữ liệu đầu vào.

Bảng 8. Vi phạm clean class trong mã nguồn

3 Đề xuất cải tiến

3.1 Vấn đề thêm mặt hàng Media mới (Audio Book) và giải pháp

Việc thêm một mặt hàng Media mới, chúng ta chỉ cần extends lớp Media mới từ lớp cha Media.



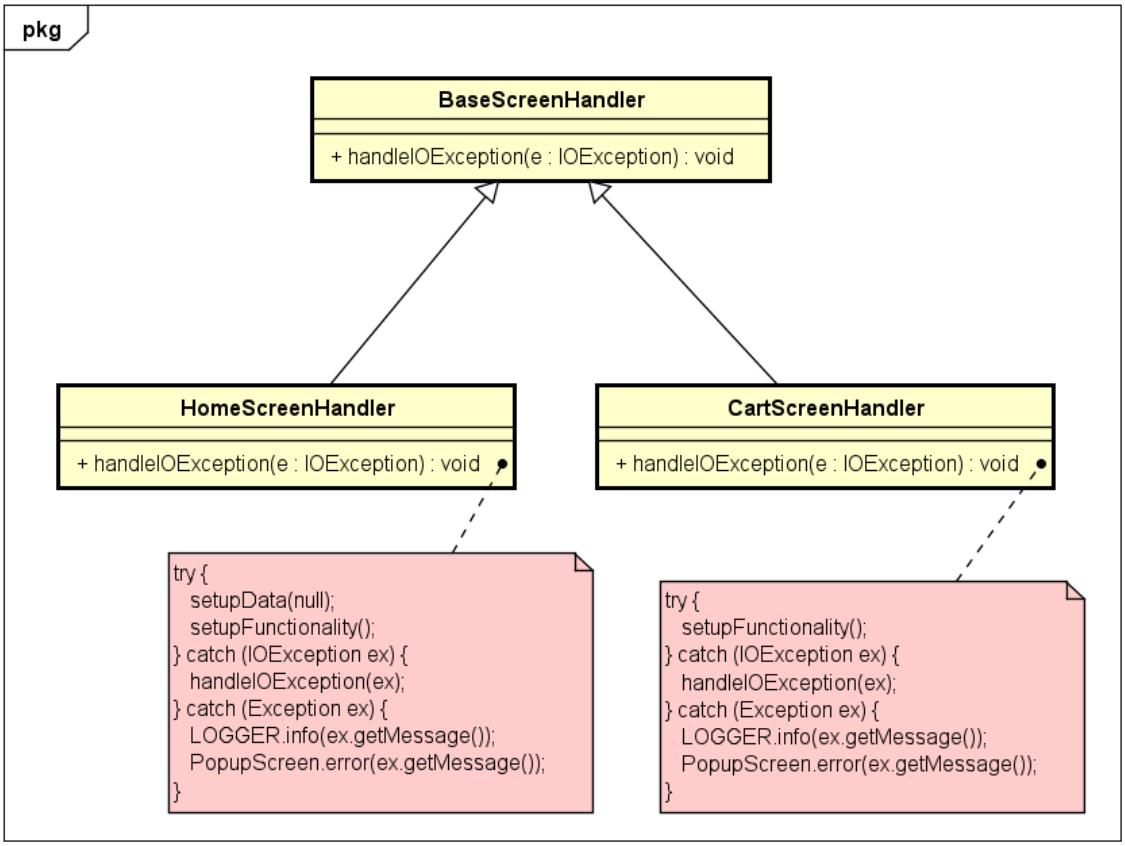
Hình 3. Giải pháp vấn đề thêm mặt hàng Media mới

3.2 Vấn đề thêm màn hình xem chi tiết sản phẩm và giải pháp

Khi thêm một màn hình chi tiết mới, do mỗi loại sản phẩm có các trường thông tin khác nhau nên màn hình chi tiết cũng phải hiển thị khác nhau ứng với mỗi loại sản phẩm. Hiện tại, giải pháp mà nhóm nghĩ tới là sủ dụng kỹ thuật Reflection trong Java để có thể truy cập vào chi tiết các thuộc tính bên trong một lớp. Tuy nhiên về giải pháp thiết kế, nhóm chưa đưa ra được một giải pháp có thể đáp ứng được các nguyên lý thiết kế.

3.3 Vấn đề thay đổi yêu cầu khi load giao diện và giải pháp

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 khắc phục vấn đề này, chúng ta sẽ tạo thêm một hàm handleIOException ở trong lớp BaseScreenHandler là một hàm chung để xử lý các lỗi về IOException trong tất cả các ScreenHandler kết thừa từ BaseScreenHandler. Mỗi khi chúng ta muốn thay đổi cách thức xử lý khi xảy ra lỗi, chúng ta chỉ cần sửa mã nguồn tại hàm này và mọi xử lý sẽ được các lớp cập nhật tương ứng.

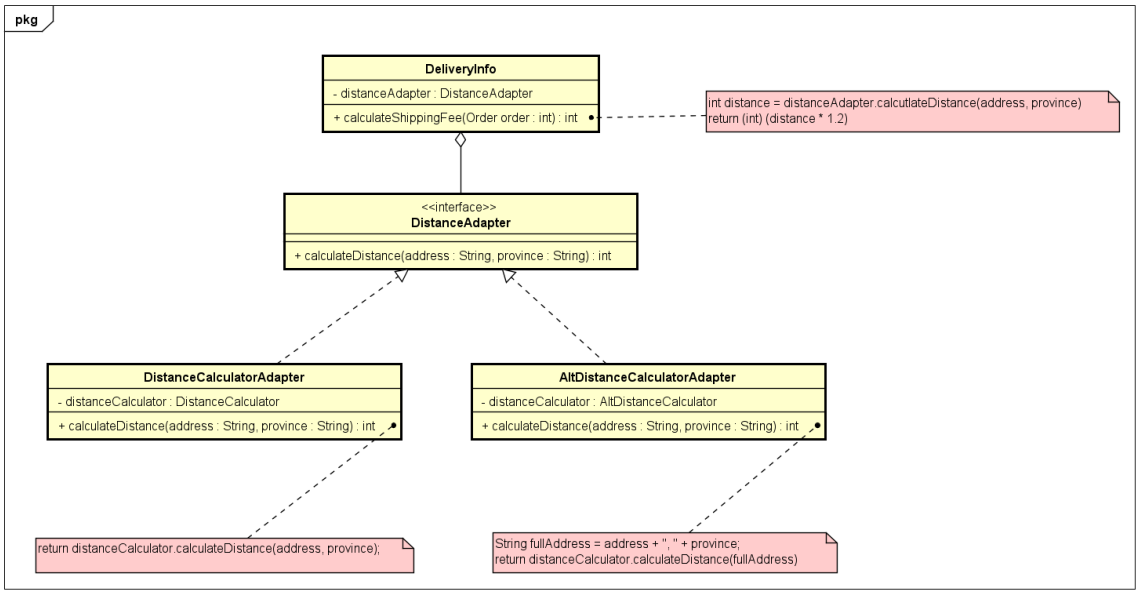


Hình 4. Giải pháp vấn đề thay đổi yêu cầu khi load giao diện

3.4 Vấn đề thay đổi thư viện tính khoảng cách mới và giải pháp

Trong mã nguồn ban đầu, nếu muốn thay đổi thư viện tính khoảng cách, chúng ta cần vào lại mã nguồn của lớp DeliveryInfo để sửa lại thuộc tính của lớp. Nếu có nhiều lớp sử dụng thư viện cũ, việc tìm kiếm và sửa mã nguồn ở các lớp này là vô cùng vất vả và khó khăn. Thiết kế ban đầu đã vi phạm nguyên lý OCP và DIP do nó

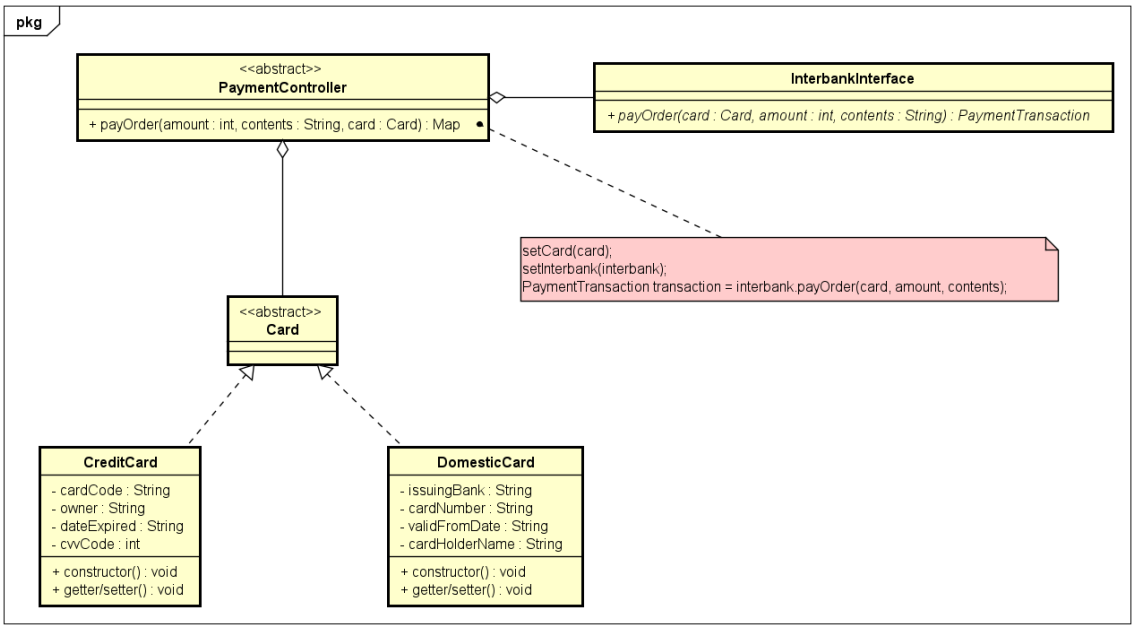
không đáp ứng được việc mở rộng dễ dàng cho chúng ta, các module phụ thuộc trực tiếp lẫn nhau chứ không thông qua một lớp trừu tượng. Để giải quyết vấn đề này, chúng ta sử dụng Adapter Design Pattern để có thể đáp ứng với nhu cầu sử dụng thư viện tính khoảng cách mới một cách dễ dàng, chỉ cần extends từ lớp DistanceAdapter các Adapter cho thư viện mới, không cần sửa đổi mã nguồn trong DeliveryInfo.



Hình 5. Giải pháp vấn đề thay đổi thư viện tính khoảng cách

3.5 Vấn đề thêm phương thức thanh toán mới (Thẻ nội địa – Domestic Card) và giải pháp

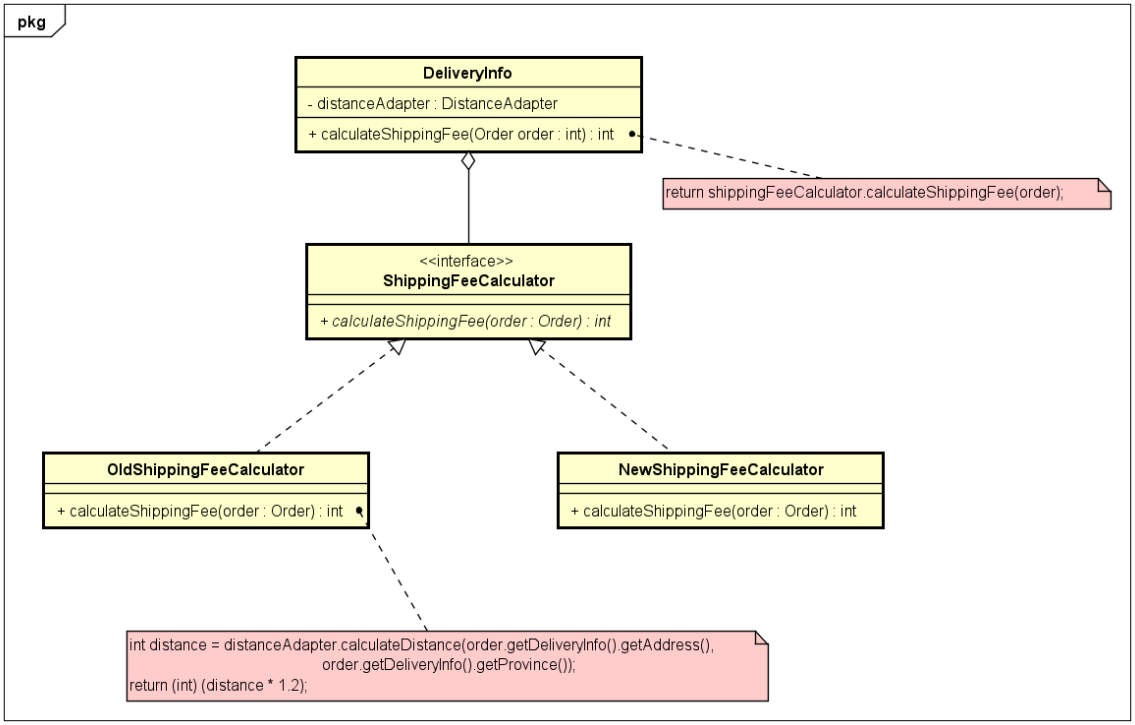
Trong mã nguồn ban đầu, việc thêm một phương thức thanh toán mới dẫn tới việc phải thay đổi mã nguồn ở nhiều module do các module đang phụ thuộc trực tiếp vào CreditCard. Điều này vi phạm nguyên lý OCP. Chúng ta cần tạo một liên kết abstraction giữa các module bằng cách tạo thêm một abstract class Card. Mọi loại thẻ thêm mới vào đều được extends từ lớp Card này.



Hình 6. Giải pháp vấn đề thêm phương thức thanh toán mới (DomesticCard)

3.6 Vấn đề thay đổi công thức tính phí vận chuyển và giải pháp

Trong mã nguồn ban đầu, để thay đổi công thức tính phí vận chuyển, chúng ta phải sửa đổi trực tiếp mã nguồn trong lớp DeliveryInfo. Để giải quyết vấn đề này, chúng ta sử dụng Strategy Design Pattern để tạo các "chiến lược" tính phí khác nhau, mọi phương pháp tính phí sẽ được implements từ interface ShippingFeeCalculator và các module cấp cao sẽ giao tiếp thông qua interface này.



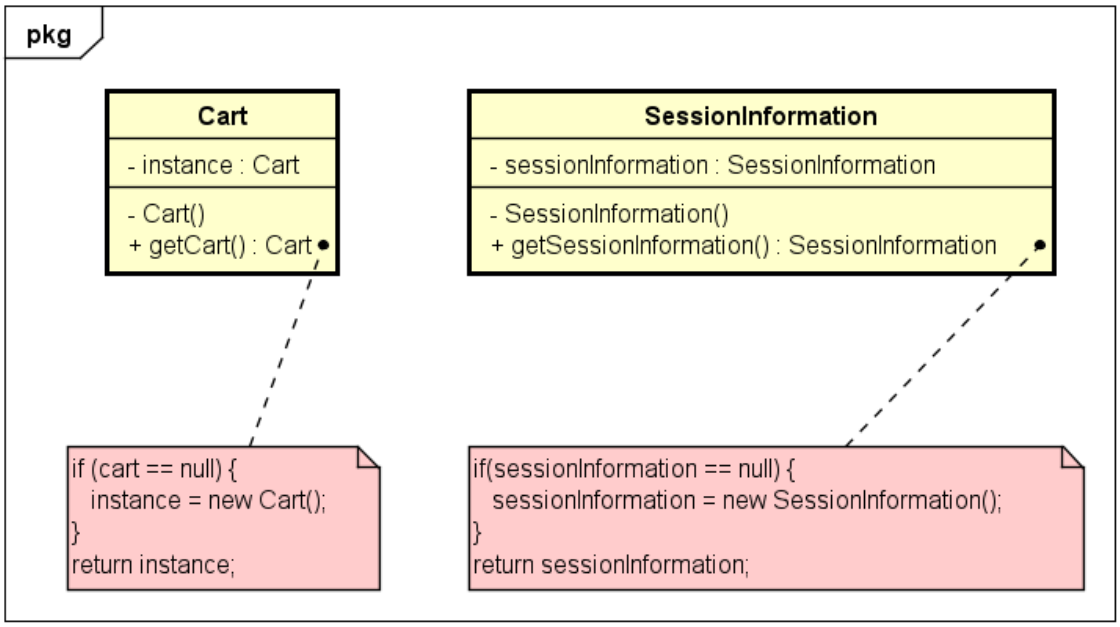
Hình 7. Giải pháp vấn đề thay đổi công thức tính phí vận chuyển

3.7 Vấn đề cập nhật lại chức năng hủy đơn hàng và giải pháp

Với vấn đề cập nhật lại chức năng hủy đơn hàng, nhóm vẫn chưa đưa ra được một giải pháp thiết kế đáp ứng được các nguyên lý thiết kế cơ bản.

3.8 Vấn đề tạo cart instance và session instance

Trong mã nguồn ban đầu, chúng ta không thể đảm bảo rằng trong chương trình chỉ tạo ra duy nhất một cart instance và một session instance. Vì vậy để giải quyết vấn đề này, chúng ta cần áp dụng Singleton Design Pattern để đảm bảo trong chương trình chỉ tạo ra duy nhất một cart instance và một session instance.



Hình 8. Áp dụng Singleton Design Pattern

4 Tổng kết

4.1 Kết quả tổng quan

Sau khi thực hiện quá trình tìm hiểu và kiểm tra mã nguồn, nhóm đã hoàn thành được các công việc sau:

- Chỉ ra được các mức độ vi phạm về Coupling và Cohension trong mã nguồn
- Chỉ ra được các vi phạm về nguyên lý SOLID trong mã nguồn
- Chỉ ra những vi phạm về clean code (clean name, clean function, clean class)
- Giải quyết được 5/7 vấn đề về yêu cầu mở rộng của mã nguồn.

Sau khi hoàn thành các công việc trên, tất cả các thành viên trong nhóm đều đã có cơ hội thực hiện ứng dụng những kiến thức đã được học trên lớp về Design Pattern để cải thiện mã nguồn của một sản phẩm.

4.2 Các vấn đề tồn đọng

Hiện tại, còn 2/7 vấn đề về yêu cầu mở rộng chức năng của phần mềm nhóm vẫn chưa đưa ra được một giải pháp thiết kế đáp ứng được các nguyên lý thiết kế. Nhóm sẽ tìm hiểu thêm và sẽ đưa ra các giải pháp trong tương lai.