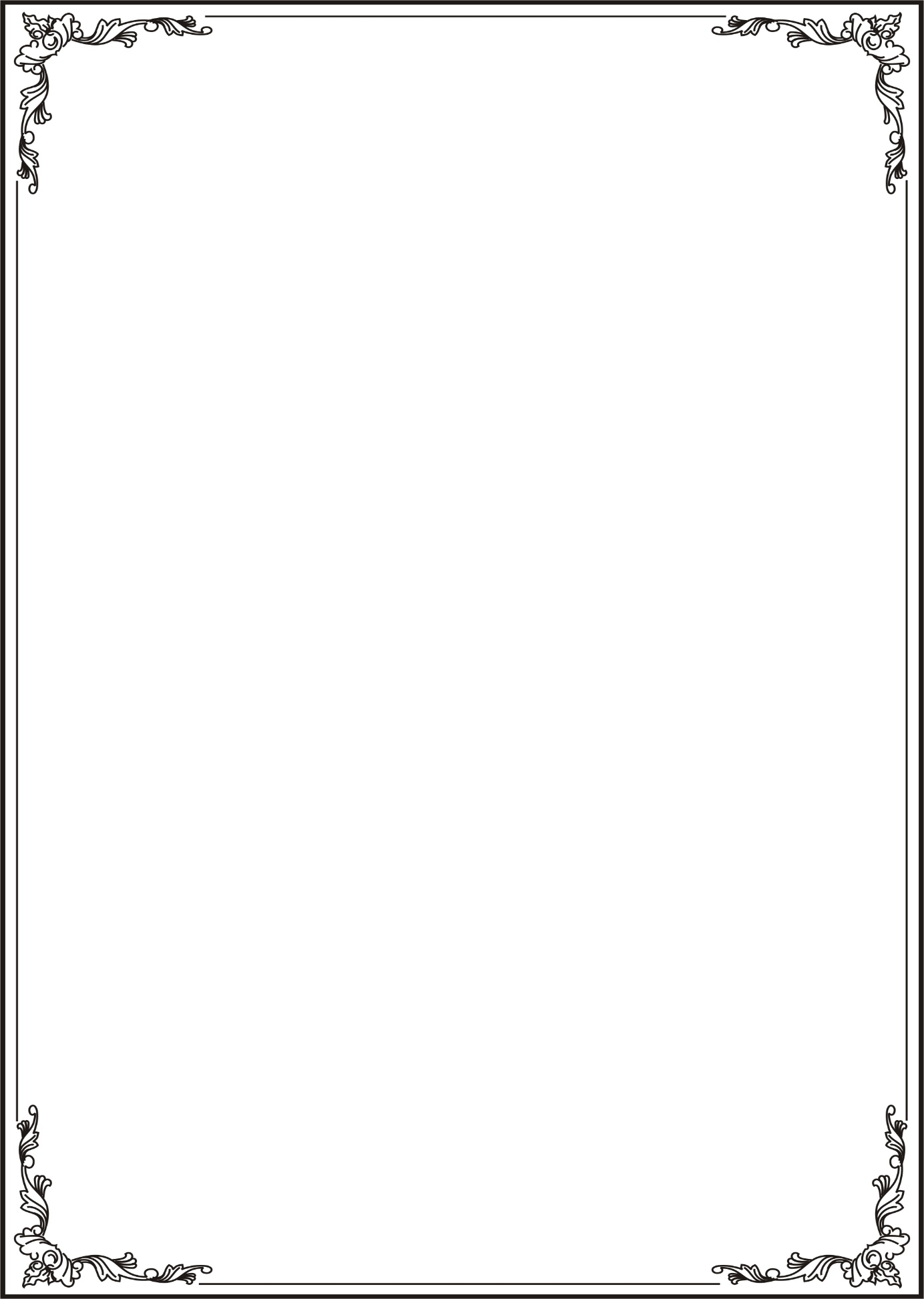
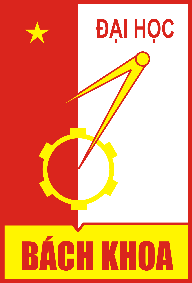
****ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông



Báo cáo Mẫu thiết kế phần mềm

Version 1.0

|  |  |  |
| --- | --- | --- |
| **Giảng viên hướng dẫn:** | GV.Nguyễn Thị Thu Trang  GV.Bùi Thị Mai Anh |  |
| **Nhóm 12:** | Lê Anh Thành | 20170115 |
|  | Hoàng Minh Tiến | 20173400 |
|  | Ngô Huy Thao | 20173381 |
|  | Đặng Đình Thọ  Phạm Văn Thành | 20173388  20173380 |

**Mục Lục**

[1 Tổng quan 1](#_Toc74075675)

[1.1 Mục tiêu 1](#_Toc74075676)

[1.2 Phạm vi 1](#_Toc74075677)

[1.2.1. Mô tả phần mềm 1](#_Toc74075678)

[1.2.2. Các chức năng chính của phần mềm 2](#_Toc74075679)

[1.2.3. Cấu trúc mã nguồn 3](#_Toc74075680)

[1.2.4. Các yêu cầu thêm cần cân nhắc cùng quá trình tái cấu trúc 3](#_Toc74075681)

[1.2.5. Các hoạt động thực thi mã nguồn để đạt được mục đích trên 3](#_Toc74075682)

[1.2.6. Kết quả dự kiến 3](#_Toc74075683)

[1.3 Danh sách thuật ngữ 4](#_Toc74075684)

[1.4 Danh sách tham khảo 4](#_Toc74075685)

[2 Đánh giá thiết kế cũ 5](#_Toc74075686)

[2.1 Nhận xét chung 5](#_Toc74075687)

[2.2 Đánh giá các mức độ coupling và cohesion 5](#_Toc74075688)

[2.2.1 Coupling 5](#_Toc74075689)

[2.2.2 Cohesion 6](#_Toc74075690)

[2.3 Đánh giá việc tuân theo SOLID 7](#_Toc74075691)

[2.3.1 SRP 8](#_Toc74075692)

[2.3.2 OCP 8](#_Toc74075693)

[2.3.3 LSP 9](#_Toc74075694)

[2.3.4 ISP 9](#_Toc74075695)

[2.3.5 DIP 10](#_Toc74075696)

[2.4 Các vấn đề về Clean Code 10](#_Toc74075697)

[2.4.1 Clear Name 10](#_Toc74075698)

[2.4.2 Clean Function/Method 13](#_Toc74075699)

[2.4.3 Clean Class 14](#_Toc74075700)

[2.5 Các vấn đề khác 16](#_Toc74075701)

[3 Đề xuất cải tiến 17](#_Toc74075702)

[3.1 Vấn đề Clear Name và giải pháp 17](#_Toc74075703)

[3.1.1 Tóm tắt vấn đề 17](#_Toc74075704)

[3.1.2 Giải pháp 17](#_Toc74075705)

[3.1.3 Ưu - nhược điểm 17](#_Toc74075706)

[3.2 Một số vấn đề Clean Function/Method + Clean Class và giải pháp 18](#_Toc74075707)

[3.2.1 Tóm tắt vấn đề 18](#_Toc74075708)

[3.2.2 Giải pháp 18](#_Toc74075709)

[3.2.3 Ưu - nhược điểm 18](#_Toc74075710)

[3.3 Vấn đề vi phạm SRP và giải pháp 18](#_Toc74075711)

[3.3.1 Tóm tắt vấn đề 18](#_Toc74075712)

[3.3.2 Giải pháp 19](#_Toc74075713)

[3.3.3 Ưu - nhược điểm 19](#_Toc74075714)

[3.4 Vấn đề các lớp chỉ sử dụng một thực thể và giải pháp 19](#_Toc74075715)

[3.4.1 Tóm tắt vấn đề 19](#_Toc74075716)

[3.4.2 Giải pháp 19](#_Toc74075717)

[3.4.3 Ưu - nhược điểm 20](#_Toc74075718)

[3.5 Vấn đề lặp một số phương thức trong các lớp giao diện và giải pháp 20](#_Toc74075719)

[3.5.1 Tóm tắt vấn đề 20](#_Toc74075720)

[3.5.2 Giải pháp 20](#_Toc74075721)

[3.5.3 Ưu - nhược điểm 22](#_Toc74075722)

[3.6 Vấn đề mối liên hệ giữa lớp MediaHandler và lớp CartScreenHandler trong package views.screen.cart và giải pháp 22](#_Toc74075723)

[3.6.1 Tóm tắt vấn đề 22](#_Toc74075724)

[3.6.2 Giải pháp 22](#_Toc74075725)

[3.6.3 Ưu - nhược điểm 24](#_Toc74075726)

[3.7 Vấn đề thay đổi yêu cầu khi load giao diện và giải pháp 25](#_Toc74075727)

[3.7.1 Tóm tắt vấn đề 25](#_Toc74075728)

[3.7.2 Giải pháp 25](#_Toc74075729)

[3.7.3 Ưu - nhược điểm 26](#_Toc74075730)

[3.8 Vấn đề thay đổi thư viện sử dụng và giải pháp 26](#_Toc74075731)

[3.8.1 Tóm tắt vấn đề 26](#_Toc74075732)

[3.8.2 Giải pháp 26](#_Toc74075733)

[3.8.3 Ưu - nhược điểm 29](#_Toc74075734)

[3.9 Vấn đề thay đổi phương thức thanh toán và giải pháp 29](#_Toc74075735)

[3.9.1 Tóm tắt vấn đề 29](#_Toc74075736)

[3.9.2 Giải pháp 29](#_Toc74075737)

[3.9.3 Ưu - nhược điểm 30](#_Toc74075738)

[3.10 Vấn đề thay đổi công thức tính phí vận chuyển và giải pháp 31](#_Toc74075739)

[3.10.1 Tóm tắt vấn đề 31](#_Toc74075740)

[3.10.2 Giải pháp 31](#_Toc74075741)

[3.10.3 Ưu - nhược điểm 32](#_Toc74075742)

[3.11 Vấn đề cập nhật chức năng hủy đơn hàng và giải pháp 33](#_Toc74075743)

[3.11.1 Tóm tắt vấn đề 33](#_Toc74075744)

[3.11.2 Giải pháp 33](#_Toc74075745)

[3.11.3 Ưu - nhược điểm 34](#_Toc74075746)

[4.Tổng kết 35](#_Toc74075747)

[4.1.Kết quả tổng quan 35](#_Toc74075748)

[4.2.Các vấn đề tồn đọng 35](#_Toc74075749)

[5.Phân chia công việc 36](#_Toc74075750)

[5.1.Trong quá trình thực hiện bài tập theo tuần 36](#_Toc74075751)

[5.2.Trong quá trình thực hiện báo cáo 40](#_Toc74075752)

**DANH MỤC HÌNH ẢNH**

[Hình 1.Use-Case Hệ Thống 2](#_Toc74075665)

[Hình 2 Clear Name 17](#_Toc74075666)

[Hình 3 Template Pattern 21](#_Toc74075667)

[Hình 4 Observer Pattern 23](#_Toc74075668)

[Hình 5 Strategy Pattern 1 25](#_Toc74075669)

[Hình 6 Adapter Pattern 27](#_Toc74075670)

[Hình 7 Factory Pattern 30](#_Toc74075671)

[Hình 8 Strategy Pattern 31](#_Toc74075672)

[Hình 9 State Pattern 33](#_Toc74075673)

# Tổng quan

Nội dung báo cáo bao gồm: Tìm kiếm,phát hiện mã nguồn vi phạm các các vấn đề về Design Concept, Design Principles, Clean Code và các yêu cầu phát sinh trong tương lai yêu cầu việc sử dụng Design Pattern để tối ưu hóa mã nguồn. Từ các vấn đề đó, đưa ra các giải pháp để tái cấu trúc mã nguồn được sạch, tăng tính tái sử dụng và tính linh hoạt dựa trên các các khuôn mẫu thiết kế đã được học.

## Mục tiêu

Việc xây dựng báo cáo thiết kế phần mềm của nhóm nhằm mục đích tổng hợp lại các kiến thức về các vấn đề Design Concept, Design Principles, Clean Code, Design Pattern trong quá trình học, vận dụng các kiến thức trên để giải quyết các vấn đề tồn trong mã nguồn, đưa ra được các đề xuất để cải tiến để loại bỏ các yếu tố khiến mã nguồn trở lên không linh hoạt, phụ thuộc, khó mở rộng. Báo cáo gồm 3 phần chính :

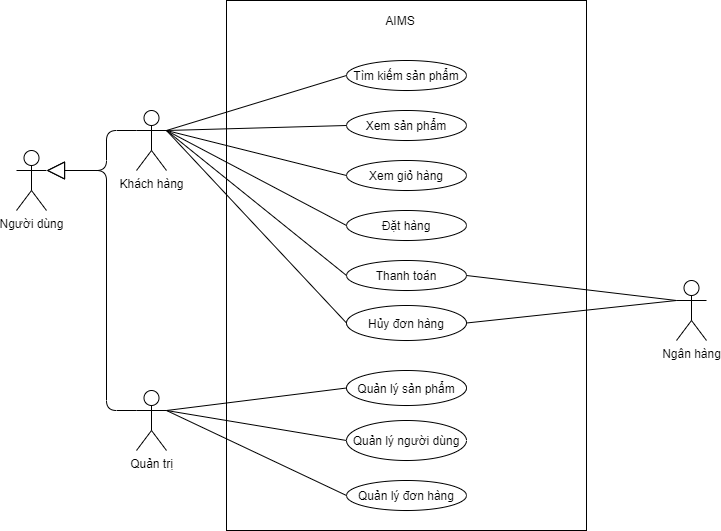
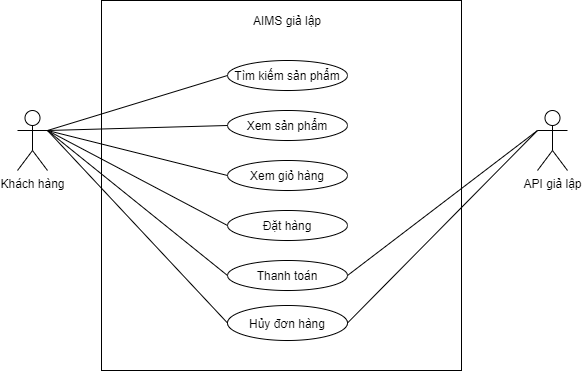
* Tìm kiếm, phát hiện, đánh giá mã nguồn cũ theo các yêu cầu trên
* Đề xuất cải tiến,đánh giá các vấn đề tìm được trong mã nguồn
* Tổng kết vấn đề, đánh giá kết quả thực hiện

## Phạm vi

### 1.2.1. Mô tả phần mềm

Phần mềm được sử dụng là AIMS(An Internet Media Store) là một hệ thống giả lập cho phép người dùng tham gia mua bán các sản phẩm trên hệ thống, bao gồm đầy đủ các quy trình từ lúc đăng ký/ đăng nhập,tìm kiếm sản phẩm, xem giỏ hàng, … đến quá trình thanh toán được tích hợp API để xử lý

### 1.2.2. Các chức năng chính của phần mềm

****

Hình .Use-Case Hệ Thống

### 1.2.3. Cấu trúc mã nguồn

Hệ thống bao gồm 3 thư mục chính:

* Src: mã nguồn của hệ thống
* Src.main.resources: chứa image,db,fxml file
* Src.main.java.common: chứa các thành phần sử dụng chung cho hệ thống bao gồm: Exception,Interface(Observable,Observer)
* Src.main.java.controller: chứa các lớp xử lý nghiệp vụ
* Src.main.java.DAO: chứa logic lưu trữ dữ liệu trên các đối tượng
* Src.main.java.entity: mô tả các thực thể của hệ thống
* Src.main.java.subsystem: xử lý nghiệp vụ giao dịch
* Src.main.java.utils: chứa các lớp chức năng tự viết
* Src.main.java.utils: chứa các lớp màn hình hiển thị
* App: lớp chạy ứng dụng
* Lib: thư viện, dependencies được sử dụng
* Test: phục vụ cho quá trình Testing

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

Các yêu cầu thêm đối với Project (yêu cầu phát sinh) sẽ được xem xét đánh giá đến tính ảnh hưởng đến mã nguồn hiện tại, đảm bảo việc mở rộng mà không vi phạm tới các nguyên lý thiết kế, tăng tính tái sử dụng và linh hoạt của mã nguồn đồng thời Refactor lại mã nguồn hiện tại dựa trên các vấn đề tìm được.

### 1.2.5. Các hoạt động thực thi mã nguồn để đạt được mục đích trên

Trước tiên, cần quan sát các module trong mã nguồn, hiểu được chức năng, nhiệm vụ của từng module, nghiệp vụ của hệ thống và luồng vận hành của mã nguồn. Tiếp đến là các thành viên trong nhóm sẽ tìm kiếm cụ thể trong từng lớp các vấn đề liên quan tới Design Concepts, Design Principles, Clean Code và các vấn đề khác để có được một bản đánh giá cụ thể. Từ đó đưa ra các giải pháp áp dụng trên các vấn đề trên sử dụng kiến thức về Design Pattern.Áp dụng Refactor mã nguồn theo Design ở bước trên cùng với báo cáo đánh giá, review code của từng subteam cho lẫn nhau để tạo ra được một sản phẩm cuối cùng.

### 1.2.6. Kết quả dự kiến

Tìm ra được các vấn đề tiềm tàng, quan trọng nhất cần phải được thiết kế lại theo các yêu cầu về thiết kế và giải pháp xử lý, refactor mã nguồn đồng thời đưa ra các hướng giải quyết các yêu cầu phát sinh trong tương lai.

## Danh sách thuật ngữ

|  |  |  |
| --- | --- | --- |
| STT | Tên | Ý nghĩa |
| 1 | Coupling | Sự liên kết giữa các module |
| 2 | Cohesion | Sự gắn kết bên trong module |
| 3 | SOLID | Design Principle |
| 4 | SRP | Single Responsibility Principle |
| 5 | OCP | Open/Closed Principle |
| 6 | LSP | Liskov Substitution Principle |
| 7 | ISP | Interface Segregation Principle |
| 8 | DIP | Dependency Inversion Principle |
| 9 | Clean Code | Mã nguồn sạch |
| 10 | Refactor Code | Tái cấu trúc mã nguồn |
| 11 | Design Pattern | Khuôn mẫu thiết kế |

## Danh sách tham khảo

1. Nguyễn Thị Thu Trang, Slide mẫu thiết kế phần mềm, <https://www.dropbox.com/sh/uidda3nhd8p7bhg/AABp62_smSg2vae4gMRlW9Aua/Slides?dl=0&subfolder_nav_tracking=1>

2. Refactoring, <https://refactoring.guru/>

3. Mã nguồn project, <https://github.com/0pdo1Tall/DP.20202.12/tree/main/src/main>

# Đánh giá thiết kế cũ

## Nhận xét chung

Mã nguồn case study hiện tại còn rất nhiều vấn đề bên trong, liên quan đến việc các module khác nhau phụ thuộc chồng chéo và bản thân bên trong module không đảm bảo liên kết. Mã nguồn vi phạm các nguyên lý thiết kế SOLID, chủ yếu là các nguyên lý SRP, OCP, DIP.

Mã nguồn chưa thực sự dễ đọc, dễ hiểu do những quy chuẩn liên quan đến việc đặt tên, sử dụng các biến vẫn còn nhiều mập mờ, chưa làm rõ được vai trò sử dụng.

Với mã nguồn hiện tại, việc ứng biến với thay đổi các yêu cầu trong tương lai sẽ gặp nhiều khó khăn vì gây ra phá vỡ cấu trúc thiết kế hiện tại, các thiết kế chưa đảm bảo được ứng phó với các yêu cầu mới.

## Đánh giá các mức độ coupling và cohesion

Coupling là tiêu chí đánh giá mức độ phụ thuộc kết dính giữa các module với nhau.

Cohesion là tiêu chí đánh giá mức độ liên kết chặt chẽ giữa các submodule bên trong một module.

Hai tiêu chí này dùng để đánh giá mức độ thiết kế của hệ thống có đảm bảo tốt không, có đảm bảo cho việc tái sử dụng, bảo trì, nâng cấp, mở rộng,... trở nên dễ dàng hơn hay không.

Một thiết kế tốt nếu đảm bảo coupling thấp nhất và cohesion cao nhất có thể. Khi đó việc tái sử dụng, bảo trì, nâng cấp, mở rộng,... sẽ trở nên dễ dàng.

### Coupling

Trong thiết kế hiện tại, vẫn còn một số module có mức độ phụ thuộc kết dính cao, nhiều module còn có khả năng sửa trực tiếp dữ liệu liệu của module khác.

Một số vấn đề được liệt kê trong danh sách sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Lý do* |
| 1 | Content Coupling | AuthenticationController | Phương thức login(),logout() | Sửa trực tiếp vào các trường mainUser, expiredTime trong lớp SessionInformation |
| 2 | Stamp Coupling | PlaceOrderController  (package controller) | validateDeliveryInfo() | Do info có nhiều hơn 3 trường dữ liệu mà chỉ sử dụng có 3 trường name, phone, address. |
| 3 | Stamp Coupling | MediaDAO | updateFieldByID() | Tham số tbname được truyền vào hàm nhưng không được sử dụng |
| 4 | Stamp Coupling | Cart | checkMediaInCart() | Truyền vào hàm đối tượng kiểu Media nhưng chỉ sử dụng trường id trong Media |
| 5 | Stamp Coupling | InterbankSubsystemController  Interbank  (package subsystem) | refund() | Tham số được truyền vào nhưng không sử dụng. |
| 6 | Content Coupling | ApplicationProgramInterface  (package utils) | allowMethods() | Do đã sửa đổi nội dung của class khác thông qua việc setAccessible(true) cho thuộc tính. |
| 7 | Stamp Coupling | InvoiceScreenHandler | setupInfoInvoice() | Truyền vào đối tượng Order nhưng chỉ sử dụng 3 trường dữ liệu |

### Cohesion

Về cơ bản, sự liên kết bên trong các module là tương đối chặt chẽ, tuy nhiên vẫn còn một số module có liên kết giữa các submodule rời rạc, và thậm chí là không có tính liên kết nào. Đó là những module thường có những chức năng không đáp ứng đúng mục đích sử dụng của module đó.

Các vấn đề về mức độ liên kết chặt chẽ giữa các submodule được liệt kê dưới đây:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Lý do* |
| 1 | Temporal | App | Phương thức start() | Các thành phần như fadeIn, fadeOut không liên quan, được ghép vào do thực thi theo thời gian |
| 2 | Coincidental | AuthenticationController | Phương thức md5() | Không liên quan đến phương thức khác, không phục vụ mục đích của lớp |
| 3 | Coincidental | MyMap | Phương thức getNextTerm() | Không có chức năng theo tên class mà chỉ hỗ trợ xử lý các hàm khác |
| 4 | Coincidental | PaymentController | phương thức getExpirationDate(Date date), phương thức emptyCart() | Có 3 phương thức không liên quan tới nhau, phương thức getExpirationDate(Date date) nên được để sang một lớp khác vì chỉ xử lý đến kiểm tra ngày tháng phương thức emptyCart() thì nên để bên ViewCartController xử lý, vì chỉ có Cart mới có quyền quản lý việc làm trống giỏ hàng |
| 5 | Coincidental | PlaceOrderController | Các phương thức validate | do các phương thức validate không thực hiện chức năng cho PlaceOrderController |
| 6 | Coincidental | nterbankPayloadConverter | convertJSONResponse | Đọc response từ interbank server không liên quan đến module InterbankConverter |
| 7 | Logical | Order |  | Tất cả các phương thức đều liên quan đến tác vụ order |

## Đánh giá việc tuân theo SOLID

Với mã nguồn hiện tại, nếu chỉnh sửa theo các yêu cầu phát sinh thì mã nguồn vi phạm các nguyên lý thiết kế SOLID khá nhiều. Các nguyên lý thường bị vi phạm là SRP, OCP và DIP. Cụ thể hơn sẽ được nhắc đến trong các phần kế tiếp.

### SRP

Hầu hết các module đều được thiết kế để tránh vi phạm nguyên lý thiết kế SRP. Tuy nhiên vẫn có những module vi phạm, chủ yếu là các module có liên quan tới điều hướng hoặc thực hiện các nhiệm vụ bổ trợ.

Chi tiết về vi phạm được liệt kê ở trong bảng sau:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Phương thức md5() | Ngoài điều hướng authentication, còn thực hiện nhiệm vụ mã hóa md5. |
| 2 | PaymentController | Phương thức getExpirationDate() | Thực hiện việc xác thực ngày quá hạn, không thuộc phạm vi quản lý |
| 3 | PlaceOrderController | Phương thức validateDeliveryInfo(), validateName(), validateAddress(), validatePhoneNumber() | Thực hiện việc kiểm tra tính hợp lệ của dữ liệu, ngoài kiểm soát đặt hàng |
| 4 | ApplicationProgramInterface  (package utils) | Vi phạm SRP | Vì thực hiện nhiều hơn một chức năng. Một chức năng là thực hiện get, post các đường dẫn url; hai là tạo ra kết nối HTTP thông qua phương thức setupConnection. |
| 5 | MyMap  (package utils) | Vi phạm SRP | Do thực hiện 2 chức năng, một là liên quan đến xử lý MyMap, 1 là chức năng liên quan đến phương thức getNextTerm để tìm cụm tiếp theo trong xâu. |

### OCP

Mã nguồn hiện tại chưa vẫn vi phạm nguyên lý OCP ở một số lớp. Hầu hết đều do việc phụ thuộc vào lớp cụ thể thay vì lớp trừu tượng, khi đó nếu có thay đổi sẽ phải sửa trực tiếp phần mã nguồn ở lớp phụ trách việc đó.

Chi tiết cụ thể về các vi phạm được nêu trong bảng sau:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Phương thức login(), md5() | Khi thay đổi phương thức mã hóa, không dùng md5 thì phải sửa code |
| 2 | PaymentController | Phương thức payOrder() | Do payOrder chỉ có thanh toán theo CreditCard. Do đó nếu thêm phương thức thanh toán thì phải sửa code phần này. |
| 3 | PlaceOrderController | Phương thức validateDeliveryInfo(), processDeliveryInfo() | Do phương thức validateDeliveryInfo sẽ phải thay đổi khi muốn thêm thuộc tính cần phải validate, phương thức processDeliveryInfo phụ thuộc trực tiếp vào cấu trúc của info. |
| 4 | PaymentTransaction(Package entity/payment) | Phương thức khởi tạo | Khi có loại card mới cần thay đổi code |

### LSP

Mã nguồn gần như không vi phạm nguyên lý LSP.

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | PaymentController |  | Kế thừa BaseController nhưng không sử dụng hết các phương thức kế thừa |

### ISP

Về cơ bản, mã nguồn đã tránh được vi phạm nguyên lý thiết kế ISP.

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | Không có |  |  |

### DIP

Mã nguồn hiện tại vẫn vi phạm nguyên lý DIP ở một số module. Hầu hết đều do việc các lớp phụ thuộc vào lớp cụ thể chứ không phải phụ thuộc vào lớp trừu tượng. Dẫn tới việc khó khăn trong quá trình bảo trì, mở rộng.

Chi tiết được liệt kê ở bảng dưới đây:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Phương thức login() | Hàm này phụ thuộc vào phương thức mã hóa md5 không phải là abstract hay interface nên khi có thay đổi về phương thức mã hóa trong tương lai thì sẽ phải modify lại hàm này. |
| 2 | Order.java(Package order) | Phương thức setDeliveryInfo() | Phụ thuộc vào 1 class là deliveryInfo không phải là interface/abstract |
| 3 | DeliveryInfo.java(Package Shipping) | Phương thức calculateShippingFee() | Phụ thuộc vào DistanceCalculator |
| 4 | PaymentTransaction(Package Payment) | Phương thức khởi tạo | Phụ thuộc vào 1 class là CreditCard không phải là interface/abstract |
| 5 | PaymentController | Phương thức payOrder() | Do phương thức payOrder() phụ thuộc là CreditCard, lớp chi tiết InterbankSubsystem |
| 6 | PlaceOrderController.java | Phương thức processDeliveryInfo() | Do phụ thuộc vào DistanceCalculator() và lớp deliveryInfo không phải là interface/abstract |

## Các vấn đề về Clean Code

### Clear Name

Mã nguồn ban đầu chưa đáp ứng được clear name, trong nhiều module có nhiều thuộc tính được đặt tên dưới dạng tên viết tắt, hoặc không mô tả được ý nghĩa đầy đủ của chúng. Một vấn đề khác là khá nhiều module sử dụng hằng số nhưng không thể hiện được ý nghĩa tại sao lại dùng hằng số đó, và sử dụng chúng có mục đích gì. Có một số phương thức đặt tên sai quy chuẩn.

Các vấn đề về Clear Name được đưa ra cụ thể trong bảng sau:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | Module | Mô tả | Lý do |
| 1 | App.java |  | Cần tạo các biến constant thay cho các hệ số. |
| 2 | BookDAO  (package dao) | Phương thức getMediaByID() | String: sql là chuỗi ứng với câu lệnh queryBook với tham số là id nên đặt tên biến như vậy là ko hợp lý.  Tương tự với tên biến: stm, res, title, price... |
| 3 | CDDAO  (package dao) | Phương thức getMediaByID() | String: sql là chuỗi ứng với câu lệnh queryBook với tham số là id nên đặt tên biến như vậy là ko hợp lý.  Tương tự với tên biến: res, title, price, ... |
| 4 | DVDDAO  (package dao) | Phương thức getMediaByID() | String: sql là chuỗi ứng với câu lệnh queryBook với tham số là id nên đặt tên biến như vậy là ko hợp lý.  Tương tự với tên biến: res, title, price |
| 5 | MediaDAO  (package dao) | Phương thức getMediaByID() | String: sql là chuỗi ứng với câu lệnh queryBook với tham số là id nên đặt tên biến như vậy là ko hợp lý.  Tương tự với tên biến: stm, res |
| 6 | Cart  (package entity) | Phương thức addCartMedia(), getTotalMedia(), calSubtotal() | Các tên biến viết tắt: cm, total, obj |
| 7 | DeliveryInfo  (package entity) | Phương thức calculateShippingFee() | Chưa định nghĩa hệ số nhân 1.2. |
| 8 | AIMSBD  (package entity) | Phương thức getConnection() | Tên biến là tên viết tắt: db.  Biến cục bộ url.(nên tạo biến final static String) |
| 9 | User  (package entity) | Phương thức setusername() | Tên phương thức không phù hợp với thuộc tính. |
| 10 | InterbankSubsystem  (package subsystem) | Trong thân lớp InterbankSubsystem | Tên biến là tên viết tắt: ctrl |
| 11 | ApplicationProgramInterface  (package utils) | Phương thức get() | Tên biến là tên viết tắt: respon |
| 12 | MyMap  (package utils) | Phương thức toJson(), toMyMap() | Tên biến là tên viết tắt: sb, i, max. |
| 13 | CartScreenHandler  (package views) | Phương thức setupFunctionality(), displayCartWithMediaAvailablity() | Tên biến là tên viết tắt: cm, im |
| 14 | MediaHandler  (package views/screen/home) | Phương thức setMediaInfo() | Cần tạo các biến constant thay cho các hệ số. |
| 15 | InvoiceScreenHandler  (package views/screen/invoice) | Phương thức setupData() | Tên biến là tên viết tắt: mis. |
| 16 | MediaInvoiceScreenHandler  (package views/screen/invoice) | Phương thức setMediaInfo() | Cần tạo các biến constant thay cho các hệ số. |
| 17 | PaymentScreenHandler  (package views/screen/payment) | Phương thức confirmToPayOrder() | Tên biến là tên viết tắt: ctrl |

### Clean Function/Method

Mã nguồn ban đầu chưa đáp ứng được yêu cầu về clean function/method. Vẫn còn một số module không đáp ứng được, và hầu hết có liên quan tới việc một phương thức thực hiện quá nhiều công việc trong quá trình thực hiện vai trò của phương thức đó.

Chi tiết các vấn đề được liệt kê bên trong bảng dưới đây.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | CartScreenHandler | Phương thức requestToPlaceOrder | Phương thức này sẽ gồm các công việc như check sản phẩm còn hay hết, hiển thị cart, tạo order,hiển thị form giao hàm. Mỗi công việc trên tương ứng với 1 chức năng nên ko thể viết tập trung lại trong hàm requestToPlaceOrder được |
| 2 | MyMap | Phương thức toMyMap | Sử dụng else không cần thiết vì throw new đã thoát ra khỏi hàm |
| 3 | ViewCartController | Phương thức getCartSubtotal() | Biến cục bộ cartSubtotal không cần thiết |
| 4 | Cart | Phương thức checkAvailabilityOfProduct() | requiredQuantity > availableQuantity for an item --> throw Exception immediately remove variable allProductAvailable sử dụng không cần thiết |
| 5 | DeliveryInfo | Phương thức calculateShippingFee | Biến cụ bộ distance sử dụng không cần thiết |
| 6 | BookDao | Phương thức getBookByIdQuery | Phương thức đảm nhận nhiều công việc con có thể tách ra |
| 7 | CDDAO | Phương thức getCDByIdQuery() | Phương thức đảm nhận nhiều công việc con có thể tách ra |
| 8 | DVDDAO | Phương thức getDVDByIdQuery | Phương thức đảm nhận nhiều công việc con có thể tách ra |
| 9 | MediaDao | Phương thức updateMediaFieldById | Phương thức đảm nhận nhiều công việc con có thể tách ra |
| 10 | UserDAO | authenticate | Phương thức đảm nhận nhiều công việc con có thể tách ra |
| 11 | InterbankPayloadConverter | Phương thức extractPaymentTransaction | Vì errorCode được lấy từ response nên cần kiểm tra luôn, nếu có lỗi, các câu lệnh sau đó không cần khởi tạo. |
| 12 | MediaHandler | Phương thức setMediaInfo | Phương thức đảm nhận quá nhiều việc, cần tách ra |
| 13 | InvoiceScreenHandler | Phương thức setupData | Phương thức đảm nhận nhiều công việc con có thể tách ra |
| 14 | App | Hàm Start | Hàm Start đảm nhận nhiều công việc con khác nhau có thể tách ra được |

### Clean Class

Mã nguồn hiện tại chưa đáp ứng được yêu cầu về clean class. Có một số module không đảm bảo các yêu cầu này, và hầu hết có liên quan tới việc lớp đảm nhiệm quá nhiều nhiệm vụ (vi phạm SRP) và phụ thuộc vào các lớp cụ thể chứ không phải lớp trừu tượng gây ảnh hưởng tới yêu cầu mở rộng (vi phạm OCP, DIP).

Các vấn đề được liệt kê trong bảng dưới đây:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | Module | Mô tả | Lý do |
| 1 | Book | Phương thức getId() | Phương thức này trùng với phương thức getId() của lớp cha(Media) |
| 2 | AuthenticationController | Large class | Do class chứa chức năng khác, không cần thiết |
| 3 | BaseController | Phương thức checkAvailabilityOfProduct() trong ViewCartController, phương thức placeOrder() trong PlaceOrderController | Phương thức checkAvailabilityOfProduct() trong class ViewCartController và phương thức placeOrder() trong class PlaceOrderController cùng thực hiện một nhiệm vụ nhưng khác tên.. |
| 4 | PaymentController | Large class | Do class chứa hàm getExpirationDate không phù hợp với chức năng |
| 5 | PlaceOrderController | Large class | - Do class chứ hàm validate không phù hợp với chức năng  - Phương thức placeOrder(), checkAvailabilityOfProduct() trong lớp ViewCartController thực hiện cùng nhiệm vụ nhưng khác tên |
| 6 | ViewCartController | Phương thức checkAvailabilityOfProduct(), placeOrder() | Phương thức checkAvailabilityOfProduct() và phương thức placeOrder() trong lớp PlaceOrderController cùng thực hiện một thứ tuy tên khác nhau |
| 7 | DVDDAO(Package media) | Phương thức getAllMedia(), getMediaById() | Do nếu media có thêm 1 trường mới, ta phải lấy kết quả từ getAllMedia và getMediaById. |
| 8 | MediaDAO | Phương thức getAllMedia(), getMediaById() | Do nếu media có thêm 1 trường mới, ta phải lấy kết quả từ getAllMedia và getMediaById. |
| 9 | InterbankInterface |  | Do sự phụ thuộc từ concrete class (CreditCard) |
| 10 | InterbankSubsystem |  | Do sự phụ thuộc từ concrete class (CreditCard) |
| 11 | InterbankPayloadConverter(Package interbank) | Phương thức convertToRequestPayload(), extracPaymentTransaction () | Nếu một phương thức payment khác, ta phải thay đổi convertToRequestPayload, extracPaymentTransaction |
| 12 | InterbankSubsystemController(Package interbank) | Phương thức convertToRequestPayload(), extracPaymentTransaction () | Nếu một phương thức payment khác, ta phải thay đổi convertToRequestPayload, extracPaymentTransaction |

## Các vấn đề khác

Một vấn đề khác có liên quan đến hiệu suất cũng như liên quan đến việc sử dụng tài nguyên. Trong một số trường hợp, việc một lớp chỉ cho phép tồn tại tối đa một đối tượng đóng vai trò quan trọng. Một số lớp có liên quan đến vấn đề đó được liệt kê trong bảng dưới đây:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | Module | Mô tả | Lý do |
| 1 | Package controller | Phương thức khởi tạo | Tại một thời điểm chỉ cần một thực thể duy nhất của lớp này |
| 2 | AIMSDB | Phương thức khởi tạo | Chỉ cần một thực thể duy nhất để tiến hành kết nối tới cơ sở dữ liệu |
| 3 | Cart | Phương thức khởi tạo | Chỉ cho phép tồn tại tối đa một giỏ hàng tại một thời điểm |

# Đề xuất cải tiến

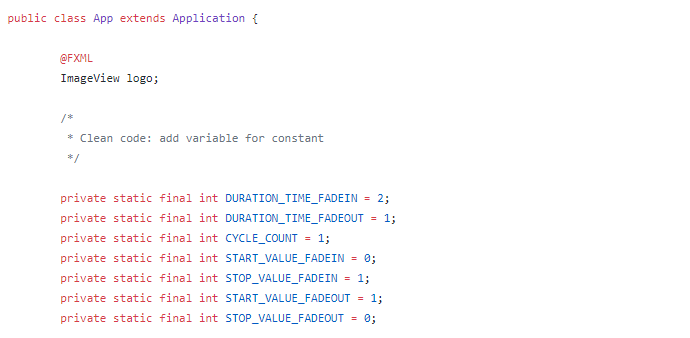
## Vấn đề Clear Name và giải pháp

### 3.1.1 Tóm tắt vấn đề

Trong rất nhiều các module, các tên hàm, tên biến được đặt không rõ ràng, rành mạch theo quy chuẩn rất dễ dẫn tới hiểu nhầm. Một số hằng số không được đặt tên dẫn tới khó khăn trong việc tìm kiếm và bảo trì. Cần đặt lại tên để cho rõ nghĩa và làm rõ được mục đích tồn tại của các thuộc tính, phương thức, lớp đó.

### 3.1.2 Giải pháp

Đối với tên các biến viết tắt, cần đặt tên lại một cách chi tiết, sao cho tên các biến diễn tả đúng ý nghĩa, vai trò của các biến đó trong phương thức hoặc lớp. Đối với các hằng số chưa được đặt tên, cần tạo các biến constant có tên mang ý nghĩa và vai trò của các hằng số đó và thay thế các hằng số đó.

Ví dụ như trong class App, cần tạo các biến constant thay cho các hệ số: 

Hình Clear Name

Hay như trong class BookDAO, có các biến có tên viết tắt như stm, res, title, price,... thì cần đặt tên lại sao cho cụ thể hơn: stm -> bookStatement, title -> bookTitle, price -> bookPrice, res -> bookResultSet, ...

### 3.1.3 Ưu - nhược điểm

Ưu điểm của giải pháp là dễ đọc, nắm được ý nghĩa, dễ bảo trì hơn. Dẫn tới việc code trở nên linh hoạt, thích ứng với thay đổi và tương lai trong việc tái sử dụng code sẽ dễ dàng.

## Một số vấn đề Clean Function/Method + Clean Class và giải pháp

### 3.2.1 Tóm tắt vấn đề

Trong một số module có một số Function/Method đảm nhiệm rất nhiều nhiệm vụ mà ta hoàn toàn có thể tách ra thành các Function/Method nhỏ hơn, hoặc các Function/Method sử dụng các biến local không cần thiết, hoặc các rẽ nhánh không xảy ra.

Một số lớp thì có các phương thức trùng với lớp cha mà nó kế thừa, hoặc có các phương thức không liên quan đến nó hoặc phụ thuộc giữa các lớp không hợp lý.

### 3.2.2 Giải pháp

* Với vấn đề “Một số Function/Method đảm nhiệm nhiều nhiệm vụ”: Phương pháp giải quyết là tách các nhiệm vụ ra thành các Function/Method riêng.
* Với vấn đề “Function/Method sử dụng các biến local không cần thiết”: Phương pháp giải quyết là loại bỏ biến local không cần thiết, trả về hoặc sử dụng giá trị của biểu thức tính toán luôn.
* Với vấn đề “Function/Method có rẽ nhánh không xảy ra”: Phương pháp giải quyết là loại bỏ phần rẽ nhánh không xảy ra đó.
* Vấn đề “Lớp con có phương thức trùng với lớp cha”: Phương pháp giải quyết là loại bỏ phương thức ở lớp con.
* Vấn đề “Lớp có phương thức không liên quan”: Phương pháp giải quyết là đưa phương thức đó về đúng lớp của nó hoặc tách ra thành lớp riêng.
* Vấn đề “Sự phụ thuộc giữa các lớp không hợp lí”: Phương pháp giải quyết là tạo ra các interface và các class giao tiếp với nhau thông qua những interface này.

### 3.2.3 Ưu - nhược điểm

* Ưu điểm: Clean Function/Method + Clean Class giúp code trở nên logic hơn, loại bỏ các thành phần dư thừa không cần thiết, giúp code dễ đọc, dễ hiểu.

## Vấn đề vi phạm SRP và giải pháp

### 3.3.1 Tóm tắt vấn đề

Trong một số module, có một số chức năng không liên quan hoặc nằm ngoài phạm vi nhiệm vụ của module đó, khiến cho chúng thực hiện nhiều hơn một nhiệm vụ. Vì thế đã vi phạm nguyên lý thiết kế SRP. Cần phải có giải pháp để giải quyết vấn đề này.

### 3.3.2 Giải pháp

Xác định rõ trách nhiệm của một lớp, mỗi lớp chỉ nên chịu trách nhiệm về một tính năng duy nhất. Nếu xuất hiện đoạn code không thuộc về trách nhiệm của lớp, cần tách đoạn code đó ra một lớp khác xử lý.

Cụ thể hơn, phương thức md5() trong class AuthenticationController. Ta sẽ sửa lại bằng cách tách phương thức md5() ra khỏi class AuthenticationController, do chưa có class nào phù hợp nên ta sẽ tạo một class mới để chứa phương thức md5() và đặt tên class là Encryption đặt trong thư mục Utils, đặt tên phương thức là encryptMd5().

Tương tự như vậy đối với phương thức getExpirationDate() trong class PaymentController, ta tách phương thức trên ra khỏi lớp PaymentController, tạo một class mới đặt tên là Date trong package Utils và đưa phương thức getExpirationDate() vào.

Tách các phương thức valiedateName(), validatePhoneNumber(), validateAddress(), validateDeliveryInfo() ra khỏi class PlaceOrderController và nhóm chúng vào một lớp mới, đặt tên là Validation được đặt trong package Utils.

### 3.3.3 Ưu - nhược điểm

Việc tách bạch nhiệm vụ giữa các lớp khiến code trở nên rành mạch, dễ đọc, dễ bảo trì, dễ mở rộng hơn trong các trường hợp trong tương lai, có thể đáp ứng được nhiều yêu cầu khi chương trình được mở rộng.

## Vấn đề các lớp chỉ sử dụng một thực thể và giải pháp

### 3.4.1 Tóm tắt vấn đề

* Một số lớp chỉ có một thực thể duy nhất trong chương trình nhưng phương thức khởi tạo có thể tạo ra các thực thể khác nhau khi khởi tạo, việc này làm mất tính logic cũng như gây nhập nhằng khi sử dụng thực thể của class.

### 3.4.2 Giải pháp

* Áp dụng Singleton cho các class chỉ có một thực thể duy nhất, hàm khởi tạo chỉ tạo ra một thực thể static, các reference sẽ chỉ sử dụng thực thể duy nhất này. Cụ thể như sau:
* Với class Cart:

private static Cart instance;

private List<CartItem> lstCartItem;

private Cart(){

lstCartItem = new ArrayList<>();

};

public static Cart getCart(){

if(instance == null){

instance = new Cart();

}

return instance;

}

* Class AIMSDB:

private static AIMSDB db;

private AIMSDB(){}

public static AIMSDB getInstance() {

if(db == null) db = new AIMSDB();

return db;

}

### 3.4.3 Ưu - nhược điểm

* Việc tạo một thực thể duy nhất giúp việc sử dụng các phương thức và thuộc tính của class (thực thể) trở nên thống nhất hơn, với kết nối database thì còn giúp cải thiện hiệu quả truy xuất.

## Vấn đề lặp một số phương thức trong các lớp giao diện và giải pháp

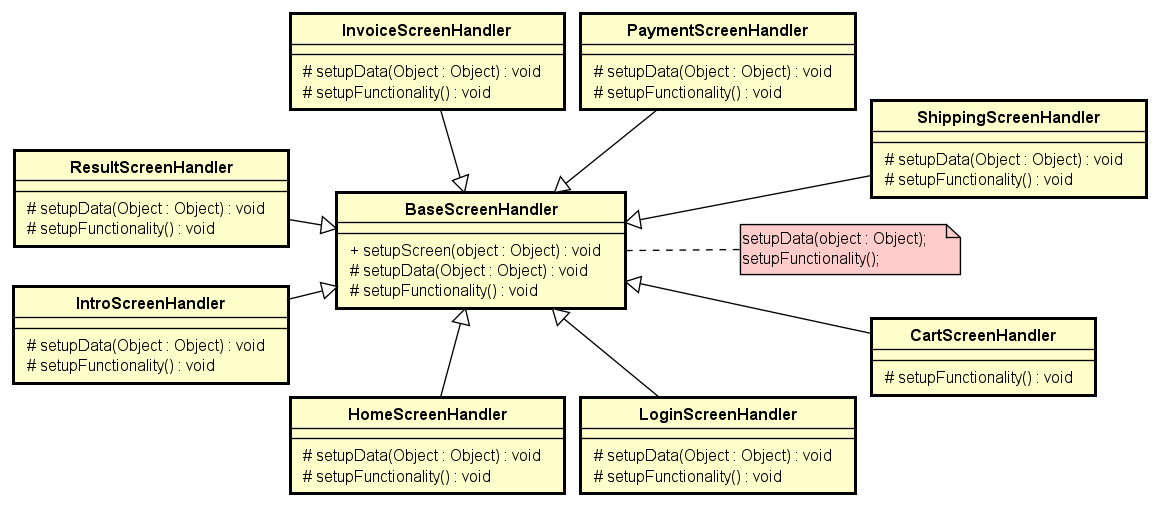
### 3.5.1 Tóm tắt vấn đề

Một số lớp giao diện đều có một khuôn mẫu chung trong việc khởi tạo, đó là đều thiết đặt giao diện rồi đến thiết lập dữ liệu và thiết lập các chức năng cho giao diện đó. Cần thiết kế sao cho trong tương lai khi tăng thêm các giao diện khác thì có thể tái sử dụng được khuôn mẫu thực hiện trong quá trình khởi tạo đó.

### 3.5.2 Giải pháp

Áp dụng Template Method để tạo ra một phương thức khuôn mẫu trên lớp cha của các lớp giao diện (BaseScreenHandler) và thiết lập trật tự thực hiện các phương thức trong quá trình khởi tạo giao diện, sau đó các lớp con kế thừa sẽ ghi đè các phương thức thiết lập để thực hiện cài đặt riêng cho từng lớp đó.

Ý tưởng thiết kế:



Hình Template Pattern

Trong đó phương thức đóng vai trò làm Template là phương thức setupScreen() được cài đặt trong lớp cha BaseScreenHandler. Phương thức Template này sẽ gọi các phương thức khác theo một trình tự xác định. Các phương thức khác đó có thể được ghi đè lại bên trong các lớp con để sử dụng phù hợp với lớp con đó.

Minh họa mã nguồn:

Tại lớp cha (BaseScreenHandler) sẽ định nghĩa ra các phương thức:

abstract protected void setupData(Ọbject object);

abstract protected void setupFunctionality();

Và một phương thức Template là:

public void setupScreen(Object object) {

setupData(object);

setupFunctionality();

}

Tại các lớp con, chỉ cần ghi đè lại phương thức nào cần thiết đối với hai phương thức được định nghĩa ở trên để có thể thiết lập theo ý muốn. Sau đó khi khởi tạo giao diện tương ứng với lớp đó thì chỉ cần gọi đến phương thức Template là có thể thực hiện theo phù hợp với yêu cầu sử dụng.

### 3.5.3 Ưu - nhược điểm

Ưu điểm của phương pháp này là tận dụng được khuôn mẫu trong các bước khởi tạo đối với giao diện, sau này nếu có thêm giao diện mới thì chỉ cần thiết đặt và ghi đè các phương thức cần thiết để tiến hành khởi tạo giao diện thật tốt. Giúp tăng khả năng tái sử dụng lại một phần mã nguồn, và đảm bảo được tính linh hoạt trong việc sử dụng chương trình. Chương trình sẽ tự động gọi các phương thức phù hợp để thiết lập lớp mà không cần tới sự can thiệp của người dùng.

## Vấn đề mối liên hệ giữa lớp MediaHandler và lớp CartScreenHandler trong package views.screen.cart và giải pháp

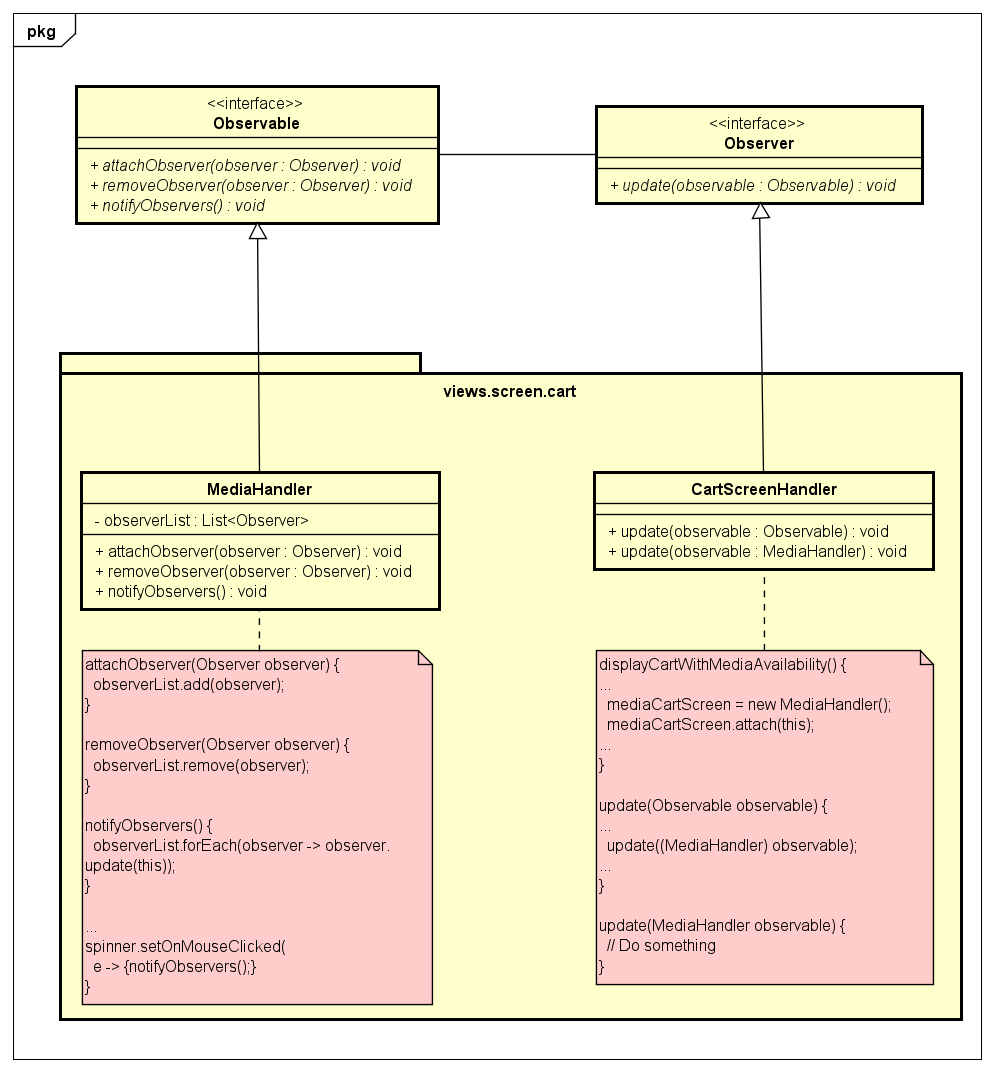
### 3.6.1 Tóm tắt vấn đề

Trong thiết kế cũ, khi có thay đổi về số lượng hoặc xóa đi một sản phẩm nào đó trong giỏ hàng, thì sản phẩm đó sẽ gọi tới giỏ hàng và cập nhật lại các thông số. Điều này chưa thực sự hợp lý nếu như việc thay đổi đó sẽ cần cập nhật lại ở nhiều nơi khác nhau, làm tăng mức độ coupling giữa các lớp. Do đó cần có một giải pháp để khắc phục điều này.

### 3.6.2 Giải pháp

Sử dụng mẫu thiết kế Observer, khi đó nếu có một sản phẩm nào đó có sự thay đổi trong số lượng hoặc bị xóa khỏi giỏ hàng thì đối tượng ứng với sản phẩm đó chỉ cần gửi thông báo thay đổi tới cho giỏ hàng, và khi đó đối tượng quản lý hiển thị cho giỏ hàng sẽ tự chịu trách nhiệm cập nhật lại giỏ hàng sao cho phù hợp. Trong thiết kế này, đối tượng sản phẩm sẽ có vai trò Observable (đối tượng được quan sát) còn đối tượng về hiển thị giỏ hàng sẽ có vai trò Observer (đối tượng quan sát).

Ý tưởng thiết kế:



Hình Observer Pattern

Trong đó lớp CartScreenHandler sẽ cài đặt giao diện Observer còn lớp MediaHandler sẽ cài đặt giao diện Observable, rồi tiến hành ghi đè lại các phương thức cần thiết để có thể sử dụng được cơ chế Observer-Observable.

Minh họa mã nguồn:

Sử dụng hai interface đã có sẵn một vài phương thức được cung cấp sẵn như trên hình minh họa là Observer và Observable.

Lớp MediaHandler cài đặt interface Observable, và do đó phải cài đặt lại một số phương thức đã được cung cấp trong interface.

Ba phương thức cần cài đặt là:

public void attachObserver(Observer observer) {

// Thêm observer vào danh sách

}

public void removeObserver(Observer observer) {

// Loại bỏ observer vào danh sách

}

public void notifyObservers() {

// Đưa ra thông báo thay đổi dữ liệu cho tất cả observer trong danh sách

}

Khi đó khi thực hiện một thao tác nào đó làm thay đổi số lượng hay nội dung thì chỉ cần gọi đến phương thức notifyObservers() thì dữ liệu sẽ được gửi tới cho các Observer.

Một ví dụ là:

spinner.setOnMouseClicked({

e -> {notifyObservers()};

})

Khi đó nếu nhấn chuột vào đối tượng thì thông báo thay đổi sẽ được gửi đến cho các Observer đang quan sát đối tượng này.

Lớp CartScreenHandler cài đặt interface Observer, và do đó phải cài đặt lại các phương thức được cung cấp trong interface.

Cụ thể là cài đặt lại phương thức update():

public void update(Observable observable) {

// Do something

}

Sau đó tiến hành các cập nhật cần thiết bên trong phương thức update() này.

### 3.6.3 Ưu - nhược điểm

Với thiết kế mới này, chúng ta đã giảm được mức độ coupling giữa hai module, ngoài ra trong tương lai nếu có phát sinh thêm một đối tượng cần liên kết và có phát sinh yêu cầu thay đổi sẽ dễ dàng trong việc tự động cập nhật lại thông tin khi sử dụng cơ chế của Observer Pattern.

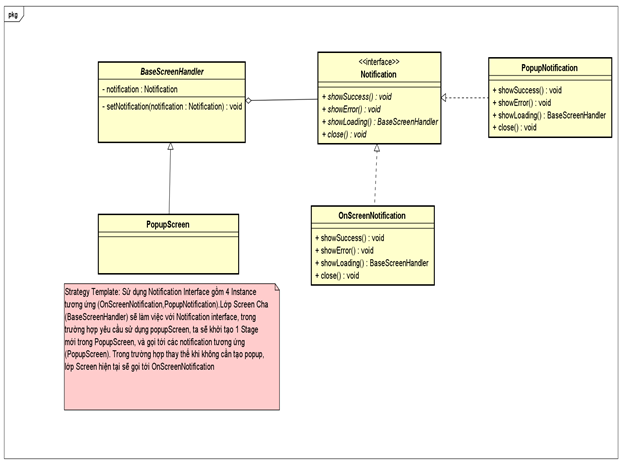
## Vấn đề thay đổi yêu cầu khi load giao diện và giải pháp

### 3.7.1 Tóm tắt vấn đề

Các lớp Screen đều có thể yêu cầu các loại thông báo khác nhau bao gồm : OnScreenNotification, PopupNotification ứng với các trường hợp cần phải đưa thông báo cho người dùng: khi xử lý thành công,thất bại hay đang load dữ liệu. Cần thiết kế sao cho khi mở rộng các yêu cầu trên (loại thông báo, kiểu thông báo) có thể tái sử dụng được mã nguồn mà không cần phải sửa đổi mã nguồn.

### 3.7.2 Giải pháp

Ý tưởng thiết kế:



Hình Strategy Pattern 1

Trong sơ đồ trên ta: Notification đóng vai trò là strategy pattern, các loại thông báo khác nhau đóng vai trò là Strategies sẽ implement Notification và thực hiện các loại thông báo khác nhau (thông báo thành công, thất bại,...), các lớp triển khai của Notification sẽ có các chiến lược thực hiện khác nhau. Để đảm bảo tính tái sử dụng trên các lớp Screen, tại lớp cha BaseScreenHandler cần phải có 1 tham chiếu tới Notification, lớp này không cần quan tâm sẽ làm việc với loại thông báo nào, nó chỉ làm việc với tất cả các Strategy thông qua một interface Notification. Các lớp màn hình con khi kế thừa lớp cha, khi cần đưa ra thông báo sẽ chỉ việc tham chiếu tới loại thông báo tương ứng.

Minh họa mã nguồn:

* Interface Notification sẽ khai báo các phương thức tương ứng với kiểu thông báo:

Void showSuccess(); // show thông báo thành công

Void showError(); // show thông báo thất bại

Void showLoading(); // show thông báo đang load dữ liệu

Void close(); // đóng thông báo

* BaseScreenHandler sẽ chứa tham chiếu với Interface Notification:

Notification notification;

setNotification(Notification notification) this.notification = notification;

### 3.7.3 Ưu - nhược điểm

Ưu điểm của phương pháp này là ta không cần phải quan tâm tới các Strategies cụ thể, mà chỉ cần giao tiếp với Strategy Interface, khi có nhiều thay đổi của Strategies trong tương lai, lớp ngữ cảnh(các lớp Screen) cũng sẽ không cần phải thay đổi cấu trúc, đảm bảo tính tái sử dụng và tính linh hoạt khi có thể kết hợp sử dụng nhiều Strategies khác nhau chỉ với 1 tham chiếu duy nhất.

## Vấn đề thay đổi thư viện sử dụng và giải pháp

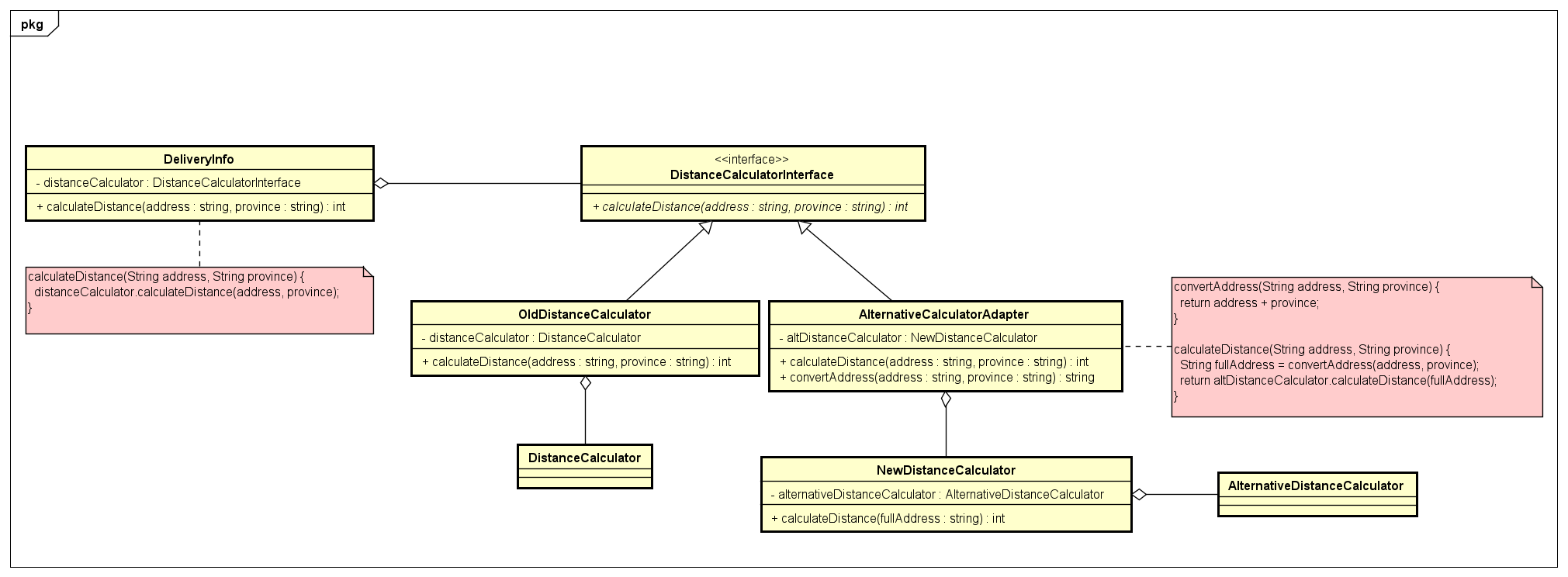
### 3.8.1 Tóm tắt vấn đề

Với yêu cầu trong tương lai có thể sử dụng thư viện mới (AlternativeDistanceCalculator) để tính khoảng cách thay cho thư viện cũ đang được sử dụng (DistanceCalculator). Thư viện mới không dùng cùng interface với thư viện cũ, cho nên không thể áp dụng trực tiếp do sự không tương thích giao tiếp giữa client với thư viện.

### 3.8.2 Giải pháp

Áp dụng mẫu thiết kế Adapter Pattern để tạo một lớp adapter trung gian cho phép hệ thống sẵn có có thể giao tiếp với thư viện mới mà không cần phải chỉnh sửa lại phần mã nguồn đã có, để không vi phạm nguyên lí OCP.

Ý tưởng thiết kế:



Hình Adapter Pattern

Khởi tạo lớp interface để đại diện cho cách tính khoảng cách, truyền như 1 thuộc tính trong lớp cần tính, các lớp con kế thừa có thể sử dụng phương thức chuyển đổi để chuyển đổi về đúng dữ liệu thích hợp.

Minh họa mã nguồn: (vài đoạn mã nguồn mẫu)

Lớp interface DistanceCalculatorInterface đại diện cho việc tính khoảng cách

Public interface DistanceCalculatorInterface {

Public int calculateDistance(String address, String province);

}

Lớp con AlternativeCalculatorAdapter cụ thể hóa cho lớp DistanceCalculatorInterface, có thêm phương thức convertAddress để chuyển đổi cho phù hợp

public class AlternativeCalculatorAdapter implements DistanceCalculatorInterface{

NewDistanceCalculator newDistanceCalculator;

public AlternativeCalculatorAdapter() {

this.newDistanceCalculator = new NewDistanceCalculator();

}

@Override

public int calculateDistance(String address, String province) {

// TODO Auto-generated method stub

String fullAddress = convertAddress(address, province);

return newDistanceCalculator.calculateDistance(fullAddress);

}

public String convertAddress(String address, String province) {

return province + address;

}

}

Lớp NewDistanceCalculator (1 cách tính khoảng cách mới) thể hiện cho 1 cách tính khoảng, là 1 thuộc tính trong lớp AlternativeCalculatorAdapter vì cần format dữ liệu cho phù hợp

public class NewDistanceCalculator {

protected AlternativeDistanceCalculator alternativeDistanceCalculator;

public NewDistanceCalculator() {

this.alternativeDistanceCalculator = new AlternativeDistanceCalculator();

}

public int calculateDistance(String fullAddress) {

return alternativeDistanceCalculator.calculateDistance(fullAddress);

}

}

Lớp OldDistanceCalculator (1 cách tính khoảng cách cũ) do không cần chuẩn hóa dữ liệu được kế thừa thẳng từ lớp interface DistanceCalculatorInterface

public class OldDistanceCalculator implements DistanceCalculatorInterface{

protected DistanceCalculator distanceCalculator;

public OldDistanceCalculator() {

this.distanceCalculator = new DistanceCalculator();

}

@Override

public int calculateDistance(String address, String province) {

// TODO Auto-generated method stub

return distanceCalculator.calculateDistance(address, province);

}

}

### 3.8.3 Ưu - nhược điểm

Khi sử dụng giải pháp này, client đang giao tiếp với thư viện cũ có thể giao tiếp với thư viện mới mà không cần phải sửa lại phần mã nguồn, không vi phạm OCP. Trong tương lai, giả sử một lần nữa phải thay đổi thư viện sử dụng để tính khoảng cách vận chuyển thì chỉ cần xây dựng một adapter khác để giao tiếp với thư viện mới.

Nhược điểm là nếu thay đổi quá nhiều thì số adapter sẽ tăng lên, khó quản lý hơn.

## Vấn đề thay đổi phương thức thanh toán và giải pháp

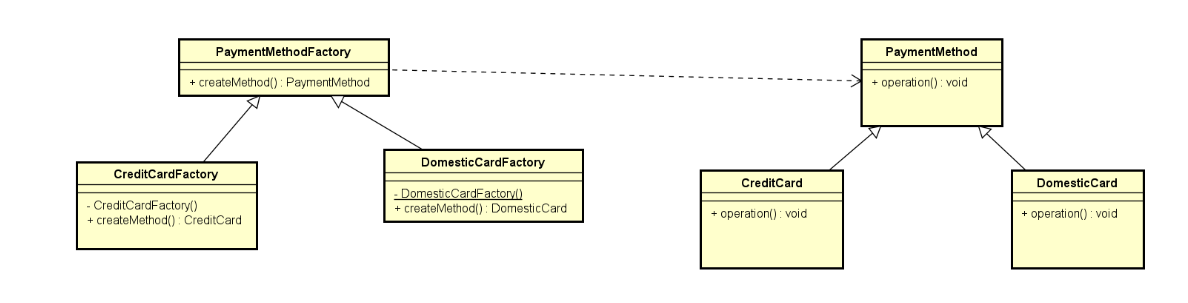
### 3.9.1 Tóm tắt vấn đề

Với yêu cầu trong tương lai, thay vì dùng CreditCard để thanh toán, khách hàng có thể lựa chọn sử dụng thẻ nội địa (DomesticCard) để thanh toán với các thông tin khai báo có đôi chút khác biệt với CreditCard. Hệ thống thẻ mới vẫn sử dụng API cũ để giao tiếp và thanh toán. Cần thiết kế mã nguồn sao cho việc thay đổi này không làm vi phạm các nguyên lý thiết kế.

### 3.9.2 Giải pháp

Ý tưởng thiết kế:

Để loại bỏ sự phụ thuộc vào 1 phương thức thanh toán duy nhất hiện có(CreditCard) vi phạm nguyên tắc OCP(phải sửa đổi mã nguồn mỗi khi cập nhật) và DIP(phụ thuộc vào lớp không phải là Abstract hay Interface) ta cần 1 lớp cha tổng quát hóa là PaymentMethod sẽ bao gồm các lớp con là các phương thức thanh toán như CreditCard, DomesticCard. Đến đây sự phụ thuộc đã bị loại bỏ hay phải sửa đổi mã nguồn đã bị loại bỏ nhưng lại phát sinh thêm vấn đề mới là việc khởi tạo các phương thức thanh toán trở lên quá tràn lan,ở đâu liên quan tới Payment Process đều cần đến quá trình khởi tạo, việc khởi tạo này có thể dẫn tới các hệ quả như tốn kém tài nguyên và đồng thời việc khởi tạo sẽ bao gồm rất nhiều nghiệp vụ bên trong bao gồm rất nhiều bước để tạo được 1 đối tượng hoàn chỉnh hoặc cũng có thể hệ thống không cho phép việc gọi tới các hàm khởi tạo trên 1 số đối tượng cụ thể. Tất cả điều trên, dẫn tới việc phải sử dụng Factory Pattern để đóng gói quá trình khởi tạo này, tận dụng những đối tượng có sẵn mà không phải khởi tạo lại. Lớp cha PaymentMethod sẽ tương ứng với lớp Abstract PaymentMethodFactory, các lớp con của PaymentMethod cũng sẽ tương ứng với 1 lớp con của PaymentMethodFactory.



Hình Factory Pattern

Minh họa mã nguồn:

* Lớp Abstract PaymentMethod: gồm có các thành phần : type, cardCode ,owner cùng với thành phần khởi tạo
* Lớp Abstract PaymentMethodFactory:
* Abstract PaymentMethod CreateMethod(...) để khởi tạo đối tượng thanh toán tương ứng
* Các lớp con kế thừa của PaymentMethodFactory:
* PaymentMethod createMethod(...) ghi đè để khởi tạo đối tượng thanh toán tương ứng
* Bên cạnh đó các lớp con nên áp dụng Singleton vì các lớp này chỉ cần 1 thực thể duy nhất

### 

### 3.9.3 Ưu - nhược điểm

Ưu điểm của phương thức này sẽ loại bỏ sự phụ thuộc vào các lớp cố định, khiến cho chi phí thay đổi lớn, bên cạnh đó việc đóng gói quá trình khởi tạo sẽ tiết kiệm được nhiều tài nguyên, đơn giản hóa quá trình khởi tạo.

Nhược điểm: Nhưng bên cạnh đó, mỗi khi có thêm 1 đối tượng thanh toán mới, sẽ phát sinh thêm 1 lớp khởi tạo tương ứng, khi có quá nhiều lớp được thêm vào sẽ khiến cho mã nguồn lớn và khó kiểm soát hơn.

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

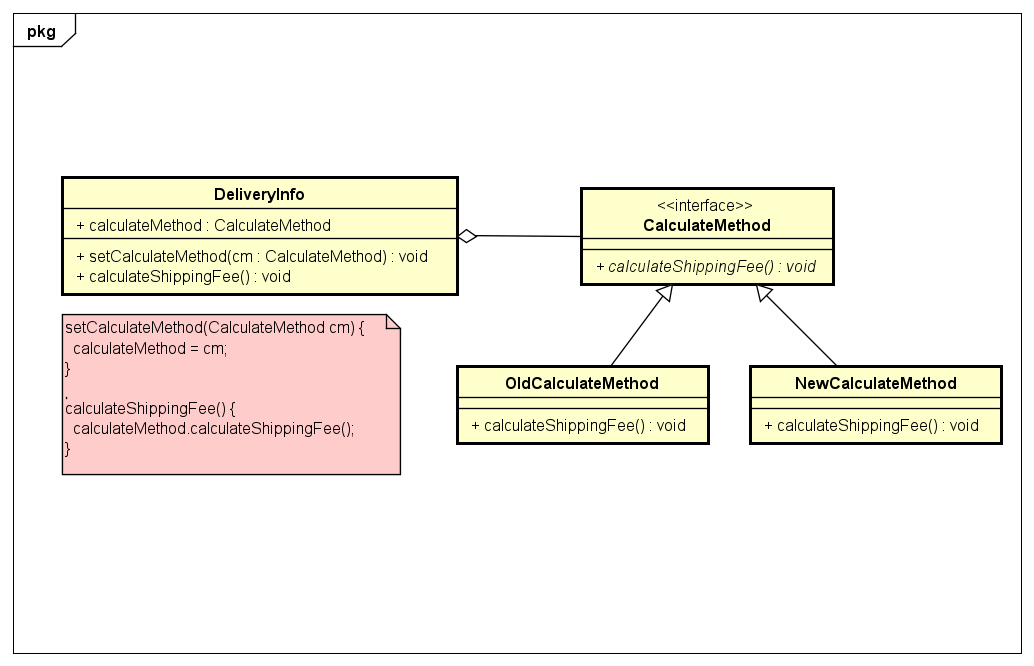
### 3.10.1 Tóm tắt vấn đề

Với yêu cầu trong tương lai thay đổi công thức tính phí vận chuyển của đơn hàng giữa trên nhiều yếu tố, bao gồm cả khoảng cách lẫn độ cồng kềnh của đơn hàng. Việc thay đổi công thức nếu không thiết kế đủ tốt sẽ gây khó khăn trong việc bảo trì.

### 3.10.2 Giải pháp

Áp dụng mẫu thiết kế Strategy để có thể thay đổi chiến lược tính phí vận chuyển một cách linh hoạt mà không vi phạm các nguyên lý thiết kế

Ý tưởng thiết kế:



Hình Strategy Pattern

Khởi tạo một lớp interface đại diện cho việc tính phí vận chuyển, bên lớp cần dùng sẽ có thuộc tính và hàm để set cách tính phí, khi muốn thêm 1 cách tính phí chỉ cần tạo class kế thừa lớp interface và truyền vào thuộc tính trong lớp cần dùng.

Minh họa mã nguồn: (vài đoạn mã nguồn mẫu)

Lớp interface Calculate:

Public interface CalculateMethod{

Public int calculateShippingFee(int distance, Order order);

}

Lớp OldCalculateMethod cụ thể hóa cho lớp CalculateMethod

Public class OldCalculateMethod implements CalculateMethod {

Private static final double MULTIPLIER = 1.2;

Public int calculateShippingFee(int distance, Order order){

Return (int)(distance \* MULTIPLIER );

}

}

Lớp Delivery

Public class DeliveryInfo{

…

Private CalculateMethod calculateMethod;

…

Public void setCalculateMethod(CalculateMethod \_calculateMethod){

this.calculateMethod = \_calculateMethod

}

...

}

Lớp CalculateMethod đại diện cho việc tính phí, lớp OldCalculateMethod sẽ kế thừa và override lại hàm tính phí calculateShippingFee. Bên lớp cần dùng DeliveryInfo chỉ cần gọi hàm setCalculate để thay đổi cách tính phí theo thuộc tính calculateMethod.

### 3.10.3 Ưu - nhược điểm

Ưu điểm của giải pháp này là nó cung cấp sự thay thế cho các lớp con, định nghĩa mỗi hành vi trong lớp riêng, loại bỏ sự cần thiết cho các câu lệnh có điều kiện, nó giúp mở rộng và kết hợp hành vi mới mà không làm thay đổi ứng dụng.

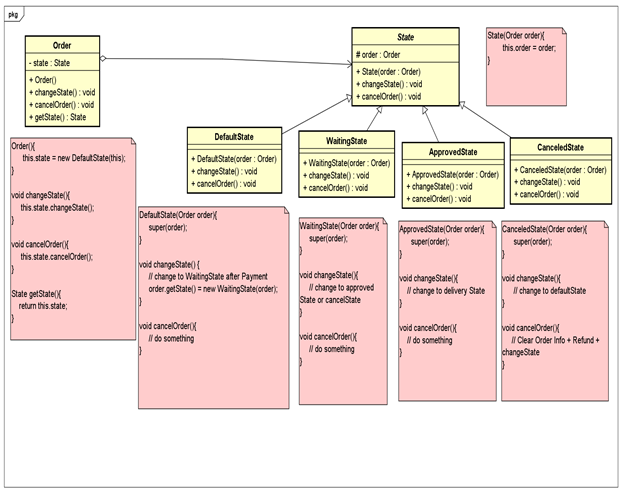
## Vấn đề cập nhật chức năng hủy đơn hàng và giải pháp

### 3.11.1 Tóm tắt vấn đề

Trong tương lai, sẽ có thể có thay đổi về chức năng liên quan đến đơn hàng. Nếu như hiện tại sau khi đặt hàng thì khách hàng có thì hủy đơn hàng và được hoàn lại tiền thì trong yêu cầu mới, sau khi quản trị viên đã phê duyệt đơn hàng, thì khách hàng không thể hủy đơn hàng được nữa. Với mã nguồn hiện tại, để thay đổi mã nguồn thì sẽ vi phạm đến các nguyên lý thiết kế, cần thiết kế và tái cấu trúc lại mã nguồn để thuận tiện hơn trong quá trình thay đổi trong tương lai.

### 3.11.2 Giải pháp

Ý tưởng thiết kế:



Hình State Pattern

Trong sơ đồ trên:

Abstract class State đóng vai trò biểu diễn trạng thái của lớp ngữ cảnh (Order). Quá trình giao hàng sẽ bao gồm một sự chuyển đổi qua lại giữa các trạng thái bắt đầu là: DefaultState, khi đã thanh toán xong yêu cầu sẽ được gửi tới cho quản trị viên phê duyệt và chuyển vào trạng thái chờ (WaitingState), khi được quản trị viên phê duyệt sẽ được chuyển vào ApprovedState và chờ để sang trạng thái giao hàng, ngược lại khi quản trị hủy yêu cầu do yêu cầu từ phía người mua, hoặc do một số lý do khác, khách hàng sẽ được hoàn tiền, hệ thống sẽ hủy bỏ đơn hàng và chuyển đơn hàng vào trạng thái hủy bỏ (CanceledOrder). Lớp ngữ cảnh Order sẽ chứa 1 tham chiếu tới trạng thái hiện tại của Order, tương ứng với mỗi trạng thái sẽ quyết định các hành động cụ thể có thể chuyển đổi sang State tiếp theo. Lưu ý, các lớp State sẽ nhận vào tham chiếu của lớp ngữ cảnh để thuận tiện cho việc xử lý các hành động.

Minh họa mã nguồn: Có trên sơ đồ trên

### 3.11.3 Ưu - nhược điểm

Ưu điểm của việc sử dụng State Pattern là ta có thể quan sát được trạng thái của lớp ngữ cảnh, từ đó đưa ra quyết định tương ứng và có thể chuyển đổi trạng thái một cách linh hoạt kể cả khi có thêm các trạng thái,tăng tính tái sử dụng code khi các hành động có liên quan đến điều kiện. Trong State Pattern ta có thể biết được sự chuyển đổi giữa các trạng thái, điều mà Strategy Pattern không có.

## 

# 4.Tổng kết

Về cơ bản, sau khi tái cấu trúc, hiệu năng trở nên tốt hơn. Mã nguồn sáng sủa, dễ đọc, dễ bảo trì và thích ứng với mở rộng hơn. Tuy nhiên vẫn còn vấn đề tồn đọng.

## 4.1.Kết quả tổng quan

So với ban đầu, hầu hết các vấn đề được xác định đều đã được giải quyết với cách thiết kế và tái cấu trúc mã nguồn mới đảm bảo hiệu quả hơn. Mức độ coupling giữa các module đã giảm xuống và mức độ cohesion bên trong từng module tăng lên. Nhờ đó hiệu năng của thiết kế cũng tăng lên.

Các thiết kế mới đã làm cho hệ thống trở nên trong suốt hơn, các module được thiết kế để giảm thiểu sự vi phạm vào các nguyên lý thiết kế, mã nguồn trở nên sáng sủa, dễ đọc, dễ bảo trì, dễ mở rộng hơn khi có yêu cầu so với tình trạng ban đầu.

Tuy nhiên vẫn còn một số vấn đề chưa giải quyết được.

## 4.2.Các vấn đề tồn đọng

Trong mã nguồn vẫn còn một số vấn đề chưa được giải quyết triệt để.

Về yêu cầu thêm một mặt hàng Media mới, nhóm mới chỉ giải quyết được vấn đề khởi tạo và phụ thuộc vào lớp trừu tượng, chưa giải quyết được việc sử dụng và thay đổi các lớp cụ thể sẽ ảnh hưởng đến các lớp khác. Tuy nhiên việc này có thể xử lý được, bằng cách sử dụng các mẫu thiết kế đã được học.

Về yêu cầu thêm màn hình xem chi tiết sản phẩm, nhóm chưa xác định được các yếu tố ảnh hưởng liên quan đến mã nguồn, và cách thay đổi. Đánh giá khả năng xử lý vấn đề này là không cao, cần phải nghiên cứu thêm.

Về yêu cầu liên quan tới thay đổi phương thức thanh toán, vấn đề đã được giải quyết, tuy nhiên có thể giải quyết một cách triệt để hơn bằng cách kết hợp với mẫu thiết kế Strategy trong quá trình quyết định lựa chọn phương thức thanh toán của khách hàng. Vấn đề này trong khả năng xử lý được.

# 5.Phân chia công việc

## 5.1.Trong quá trình thực hiện bài tập theo tuần

**Design Pattern - Coupling**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Tìm Coupling trên các package : controller,dao,entity,subsystem,utils,views.screen,class App
* Subteam 2: Ngô Huy Thao, Đặng Đình Thọ, Hoàng Minh Tiến: Tìm Coupling trên các package controller, dao, entity, subsystem, utils, views.screen, class App

**Design Pattern - Cohesion**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Tìm Cohesion lên các class: PaymentController, App, PlaceOrderController, Observable(common.interfaces), Observer(commmon.interfaces), ApplicationProgrammingInterface(java.utils), MyMap(java.utils), Utils(java.utils), AuthenticationController.
* Subteam 2: Ngô Huy Thao, Đặng Đình Thọ, Hoàng Minh Tiến:Tìm Cohesion cho các class: PlaceOrderController, AuthenticationController, BaseController, HomeController, PaymentController, ViewCartController(java.controller), Cart(entity.cart), cartItem(entity.cart), AIMSDB(java.entity.db), Invoice(java.entity.invoice), Book,DVD,Media(java.entity.media), Order(java.entity.order), DeliveryInfo(java.entity.shipping), User, BaseScreenHandler, CartScreenHandler, HomeScreenHandler(views.screen)

**Design Pattern - SOLID**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Tìm SOLID trên các class: AuthenticationController, PaymentController, PlaceOrderController, MyMap, ApplicationProgrammingInterface, HomeController.
* Subteam 2: Ngô Huy Thao, Đặng Đình Thọ, Hoàng Minh Tiến: Tìm SOLID cho các class: Cart, CD, CartScreenHandler, HomeScreenHandler, LoginScreenHandler, ShippingScreenHandler, Media, Order, PaymentTransaction, DeliveryInfo, AuthenticationController, PaymentController, InterbankSubsystem.

**Design Pattern - Singleton**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Singleton cho các class: AuthenticationController, HomeController, PaymentController, PlaceOrderController, ViewCartController (package controller)
* Subteam 2: Ngô Huy Thao, Đặng Đình Thọ, Hoàng Minh Tiến: Áp dụng Singleton cho các class: AuthenticationController (package controller), AIMSDB (package entity.db), Cart (package entity.cart), Utils (package utils)

**Design Pattern - Template Method + Factory Method**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Template Method cho các class: CartScreenHandler, HomeScreenHandler, LoginScreenHandler, IntroScreenHandler, InvoiceScreenHandler, PaymentScreenHandler, ResultScreenHandler, ShippingScreenHandler với lớp cha là BaseScreenHandler.
* Subteam 2: Hoàng Minh Tiến, Đặng Đình Thọ,Ngô Huy Thao: Áp dụng Factory Method cho Media và các lớp con: Book,CD,DVD. Tạo lớp cha PaymentMethod của CreditCard trong payment,tạo lớp cha PaymentMethodFactory và lớp con CreditCardFactory ứng với PaymentMethod và CreditCard

**Clean Code - Meaningful Names**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Meaningful Names cho các class: AuthenticationController, PaymentController (package controller); Cart (entity.cart); DeliveryInfo (entity.shipping); InterbankSubsystem (interbank); CartScreenHandler (views.screen.cart); MediaHandler (views.screen.home); InvoiceScreenHandler, MediaInvoiceScreenHandler (views.screen.invoice); PaymentScreenHandler (views.screen.payment); App (default package).
* Subteam2 : Hoàng Minh Tiến, Đặng Đình Thọ, Ngô Huy Thao Áp dụng Meaningful Names trong các class sau: AuthenticationController, PaymentController, PlaceOrderController, ViewCartController, BookDAO, CDDAO, DVDDAO, MediaDAO, UserDAO, Cart, AIMSDB, ApplicationProgrammigInterface, MyMap, Utils

**Clean Code - Method Refactoring**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Method Refactoring cho các class: PaymentController, PlaceOrderController, ViewCartController (package controller); BookDAO, CDDAO, DVDDAO, MediaDAO (package dao.media); UserDAO (package dao.user); Cart (package entity.cart); AIMSDB (package entity.db); Order (package entity.order); DeliveryInfo (package entity.shipping); User (package entity.user); App (default package).
* Subteam 2: Hoàng Minh Tiến, Đặng Đình Thọ, Ngô Huy Thao Áp dụng Method Refactoring cho các class: BookDAO,DVDDAO,MediaDAO,UserDAO,PaymentTransaction,InterbankPayloadConverter

**Clean Code - Clean Class, Clean Test**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Clean Class cho các class: AuthenticationController, BaseController, PaymentController, PlaceOrderController, ViewCartController (package controller); Book (package entity.media). Áp dụng Clean code cho các class: MediaHandler(Searchable names) (package views.screen.home). Áp dụng Clean code để loại bỏ sự phụ thuộc trực tiếp vào ConcreteClass (cụ thể là CreditCard) để phụ thuộc vào AbstractClass (cụ thể là PaymentMethod - superclass của CreditCard) và ứng dụng cả FactoryMethod trong các class: PaymentController (package controller); InterbankInterface, InterbankSubsystem (package subsystem); InterbankPayloadConverter, InterbankSubsystemController (package subsystem.interbank); PaymentScreenHandler (package views.screen.payment).
* Subteam 2: Hoàng Minh Tiến, Đặng Đình Thọ, Ngô Huy Thao Áp dụng Clean Class cho các class: DVDDao,MediaDao,InterbankPayloadConverter,InterbankSubsystemController

**Design Pattern - Strategy Pattern**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Strategy Method cho các class: DeliveryInfo (package entity.shipping), tạo interface CalculateMethod (package entity.shipping) để ứng dụng Strategy Method cho class trên phần tính chi phí vận chuyển (method calculateShippingFee).
* Subteam 2: Hoàng Minh Tiến, Đặng Đình Thọ, Ngô Huy Thao Áp dụng Strategy Method cho các class: BaseScreenHandler, tạo interface Notification với 2 implementation OnScreenNotification(sử dụng thông báo trong màn hiện tại),PopupNotification(sử dụng thông báo trong cửa sổ Popup)

**Design Pattern - Observer Pattern**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Observer Pattern cho class MediaHandler - sử dụng Interface Observable, class CartScreenHandler - sử dụng Interface Observer (package views.screen.cart).
* Subteam 2: Hoàng Minh Tiến, Đặng Đình Thọ,Ngô Huy Thao Áp dụng Observer Pattern cho class MediaHandler - sử dụng Interface Observable, class CartScreenHandler - sử dụng Interface Observer

**Design Pattern - Adapter Pattern, State Pattern**

* Subteam 1: Lê Anh Thành, Phạm Văn Thành: Áp dụng Adapter Pattern giải quyết yêu cầu số 4. Cho class DeliveryInfo (package entity.shipping) liên quan tới sự thay đổi thư viện tính toán khoảng cách. Tạo mới một interface DistanceCalculatorInterface, ba class OldDistanceCalculator, AlternativeCalculatorAdapter, NewDistanceCalculator (package entity.shipping).
* Subteam 2: Ngô Huy Thao, Đặng Đình Thọ, Hoàng Minh Tiến: Áp dụng State Pattern giải quyết yêu cầu số 7. Cho class Order liên quan tới yêu cầu hủy đơn hàng. Tạo mới abstract class State, với các thành phần con kế thừa DefaultState, WaitingState, ApprovedState, CanceledState.

## 5.2.Trong quá trình thực hiện báo cáo

|  |  |
| --- | --- |
| **Vấn đề và giải pháp** | **Người thực hiện** |
| 1.Clear Name | Ngô Huy Thao |
| 2. Clean Function, Clean Code, Clean Class | Đặng Đình Thọ |
| 3. Vi phạm SRP | Ngô Huy Thao |
| 4. Các lớp chỉ sử dụng một thực thể | Đặng Đình Thọ |
| 5. Vấn đề lặp một số phương thức trong lớp giao diện | Lê Anh Thành |
| 6. Mối liên hệ giữa lớp MediaHandler và CartScreenHandler | Lê Anh Thành |
| 7. Vấn đề thay đổi yêu cầu khi load giao diện | Hoàng Minh Tiến |
| 8. Vấn đề thay đổi thư viện sử dụng | Phạm Văn Thành |
| 9. Vấn đề thay đổi phương thức thanh toán | Hoàng Minh Tiến |
| 10. Vấn đề thay đổi công thức tính phí vận chuyển | Phạm Văn Thành |
| 11. Vấn đề cập nhật chức năng hủy đơn hàng | Hoàng Minh Tiến |