ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

**Nhóm 01**

|  |  |  |
| --- | --- | --- |
| STT | Họ tên | MSSV |
| 1 | Nguyễn Đức Anh | 20172956 |
| 2 | Lê Xuân An | 20172931 |
| 3 | Nguyễn Đình Quang Anh | 20172936 |
| 4 | Nguyễn Mạnh Cường | 20172989 |

*Hà Nội, Tháng 6 Năm 2021*

**Mục lục**

[**Tổng quan**](#_heading=h.tyjcwt) **2**

[Mục tiêu](#_heading=h.3dy6vkm) 3

[Phạm vi](#_heading=h.1t3h5sf) 3

[Danh sách thuật ngữ](#_heading=h.4d34og8) 4

[Danh sách tham khảo](#_heading=h.2s8eyo1) 5

[**Đánh giá thiết kế cũ**](#_heading=h.17dp8vu) **5**

[Nhận xét chung](#_heading=h.3rdcrjn) 6

[Đánh giá các mức độ coupling và cohesion](#_heading=h.26in1rg) 6

[Coupling](#_heading=h.lnxbz9) 6

[Cohesion](#_heading=h.35nkun2) 7

[Đánh giá việc tuân theo SOLID](#_heading=h.1ksv4uv) 9

[SRP](#_heading=h.44sinio) 9

[OCP](#_heading=h.2jxsxqh) 9

[LSP](#_heading=h.z337ya) 10

[Các vấn đề về Clean Code](#_heading=h.3j2qqm3) 11

[Clear Name](#_heading=h.1y810tw) 11

[Clean Function/Method](#_heading=h.4i7ojhp) 12

[Clean Class](#_heading=h.2xcytpi) 12

[Các vấn đề khác](#_heading=h.1ci93xb) 13

[**Đề xuất cải tiến**](#_heading=h.3whwml4) **13**

[Vấn đề clean code, coupling, cohesion, SOLID và giải pháp](#_heading=h.2bn6wsx) 14

[Vấn đề tạo nhiều instance Cart và hướng giải quyết](#_heading=h.vyd9yjwajqqn) 15

[Vấn đề xem chi tiết sản phẩm và hướng giải quyết](#_heading=h.3sdzcjd5g2eo) 16

[Vấn đề thay đổi tính khoảng cách và cách giải quyết](#_heading=h.t2how8wz4ni2) 17

[Vấn đề 5 và cách giải quyết](#_heading=h.siq49fljeblu) 18

[Vấn đề thêm mới cách tính phí vận chuyển và cách giải quyết](#_heading=h.5as5429goexc) 19

[Vấn đề cập nhật chức năng hủy đơn hàng và cách giải quyết](#_heading=h.y9s6oyw740b3) 20

[Vấn đề xóa item trong Cart](#_heading=h.okw5bf1slmnl) 21

[**Tổng kết**](#_heading=h.3as4poj) **22**

[Kết quả tổng quan](#_heading=h.1pxezwc) 23

[Các vấn đề tồn đọng](#_heading=h.49x2ik5) 23

**Danh sách các minh họa**

**Error! No table of figures entries found.**

**Danh sách các bảng**

**Error! No table of figures entries found.**

# Tổng quan

Báo cáo Mẫu thiết kế phần mềm(Design Pattern) là báo cáo tổng hợp các vấn đề của codebase đề ra, song song với đó là các phần giải quyết vấn đề mà nhóm một đã tổng hợp và cùng nhau giải quyết

## Mục tiêu

Mục đích sử dụng của báo cáo mẫu thiết kế phần mềm: Tổng hợp báo cáo, đưa ra chi tiết các vấn đề cần giải quyết và cách giải quyết vấn đề như thế nào.

Đối tượng đọc báo cáo:

* Giảng viên dạy học khóa học này
* Các thành viên trong nhóm
* Các người đọc quan tâm có thể sử dụng như một bộ tài liệu tham khảo

Mô tả khái quát nội dung báo cáo:

- Tổng quan: cái nhìn cung của báo cáo

- Đánh giá thiết kế cũ: Đánh giá lại những lỗi về clean code, coupling, cohesion của codebase cũ

- Đề xuất cải tiến: Đưa ra những cái tiến mới cho những vấn đề đặt ra

## Phạm vi

*Trong mục này, cần làm rõ:*

1. *Mô tả khái quát phần mềm*
2. *Các chức năng chính của phần mềm*
3. *Cấu trúc mã nguồn*
4. *Các yêu cầu thêm cần cân nhắc cùng quá trình tái cấu trúc*
5. *Các hoạt động (Ví dụ: review, refactor) thực thi trên mã nguồn để đạt được mục tiêu kể trên*
6. *Kết quả dự kiến*>

Khái quát phần mềm:

Phần mềm AIMS là phần mềm giúp người dùng mua sách, đĩa CD, đĩa DVD. Sử dụng ngôn ngữ lập trình Java cùng thư viện JavaFX để tạo giao diện. Hệ quản trị cơ sở dữ liệu sử dụng là sqlite.

Các chức năng chính:

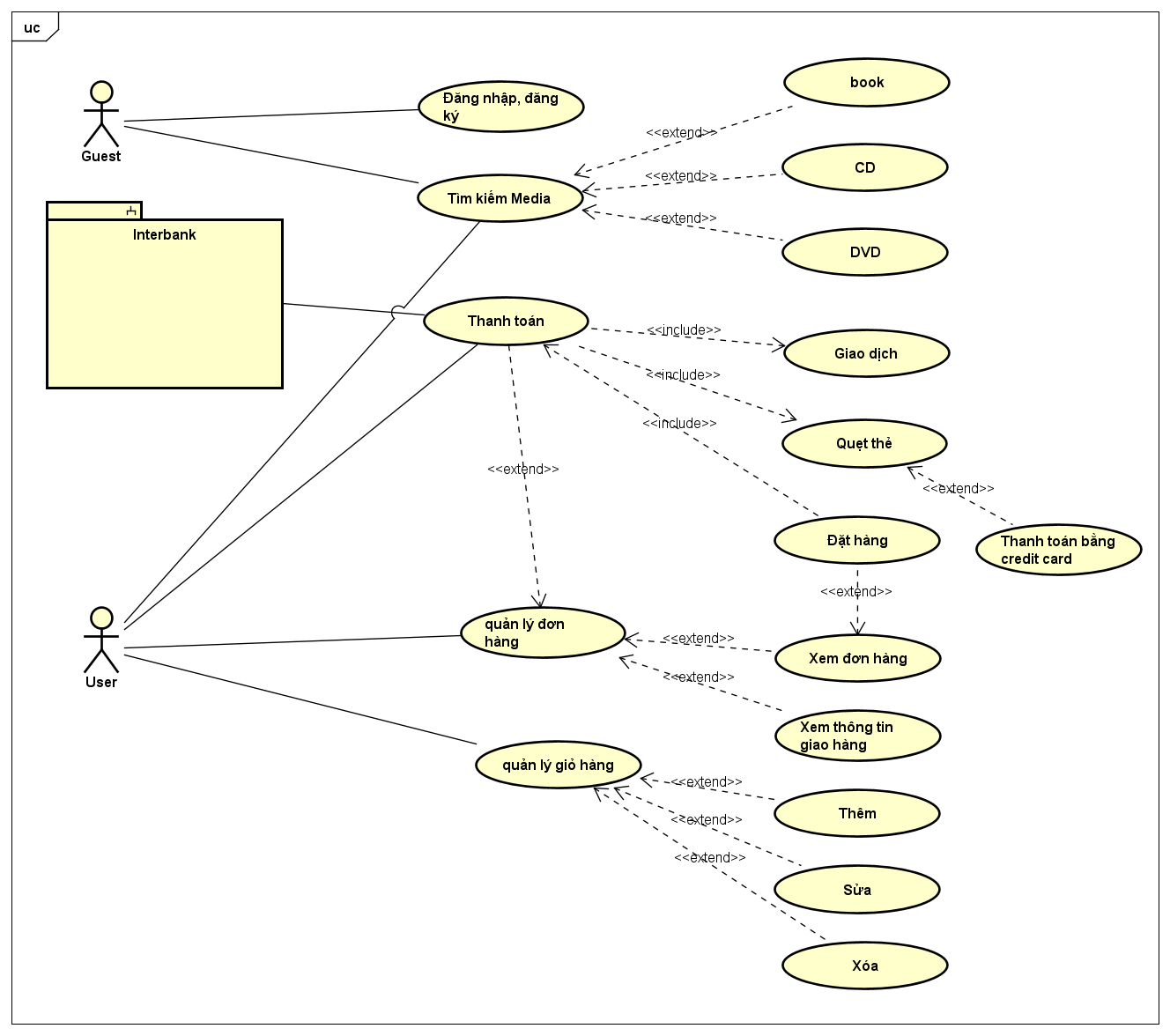
Tìm kiếm sản phẩm theo tên, theo loại sản phẩm (book, cd, dvd)

Thêm sản phẩm vào giỏ hàng

Xóa sản phẩm trong giỏ hàng

Thanh toán giỏ hàng

Biểu đồ Use Case tổng quan cho hệ thống



## Danh sách thuật ngữ

SOLID: 5 chữ cái đầu của 5 nguyên lý Single responsibility principle, Open/closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle

SRP: Single responsibility principle

OCP: Open/closed principle

LSP: Liskov substitution principle

ISP: Interface segregation principle

DIP: Dependency inversion principle

## Danh sách tham khảo

1. <https://refactoring.guru/design-patterns>

2. Dive Into Design Patterns by Alexander Shvets

# Đánh giá thiết kế cũ

## Nhận xét chung

Mã nguồn của case study ban đầu tồn tại khá nhiều vi phạm về các nguyên lý thiết kế ví dụ như SOLID, chưa áp dụng những design pattern để thiết kế gây ra các khó khăn khi có những update về tính năng, giao diện trong tương lai.

## Đánh giá các mức độ coupling và cohesion

### Coupling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Common Coupling | entity.db.AIMSDB | method static getConnection có thể truy cập ở mọi nơi | Khi sử dụng chung nguồn tài nguyên connection như thế dễ dẫn đến việc ghi đè dữ liệu hoặc các connection block nhau |
| 2 | Common Coupling | ViewConfigs | các biến, hằng đề ở static có thể truy cập nhiều nơi | Không ảnh hưởng nhiều lắm |
| 3 | Stamp Coupling | InterbankSubsystemController | hàm refund chưa thực hiện gì cả | Có các đối tượng truyền vào nhưng không được sử dụng => dẫn đến khó hiểu code |
| 4 | Stampl Couping | DeliveryInfo | method calculateShippingFee mặc dù truyền vào Order nhưng không sử dụng | Code không cần sử dụng |
|  |  |  |  |  |

### Cohesion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Logical Cohesion | dao.media.MediaDAO | getAllMedia và getMediaById đều là lấy thông tin nhưng không sử dụng dữ liệu của nhau | Code trong class dài, khó đọc. |
| 2 | Temporal Cohesion | controller.AuthenticationController | Xác thực quá nhiều thông tin trong 1 class, nhiều cấp xác thực trong 1 class | Khó đọc, khó bảo trì code. |
| 3 | Temporal Cohesion | Controller.  PaymentController | Thực hiện kiểm tra ngày hết hạn của thẻ và thanh toán trong 1 class | 2 method không sử dụng dữ liệu của nhau, là độc lập nhau nên khiến class khó đọc, dài. |
| 4 | Temporal Cohesion | Controller.  PlaceOrderController | Các hàm validate các dữ liệu khác nhau cùng trong 1 class | Class quá nhiều method, khó đọc khó bảo trì. |
| 5 | Coincidental cohesion | dao.media.BookDAO | Trong method getMediaByID thực hiện 2 việc truy vấn dữ liệu và xử lí dữ liệu trả về | Method getMediaByID quá dài, khó đọc. Khi có thay đổi khó bảo trì. |
| 6 | Coincidental cohesion | dao.media.MediaDAO | Trong method getMediaByID thực hiện 2 việc truy vấn dữ liệu và xử lí dữ liệu trả về | Method getMediaByID quá dài, khó đọc. Khi có thay đổi khó bảo trì. |
| 7 |  |  |  |  |

## Đánh giá việc tuân theo SOLID

### SRP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | PlaceOrderController | Lớp này thực hiện quá nhiều method trong cùng một class, đó là validate | Giả sử có một lớp khác cũng có validate tương tự thì dẫn đến duplicate code, nên tách ra validate thành 1 class |
| 2 | PaymentController | Lớp này thực hiện quá nhiều method trong cùng 1 class: getExpirationDate(), isValidMonthandYear(),  payOrder(), emptyCart(), | Làm không rõ mục đích chính của class, nên tách method isValidMonthandYear() ra thành class mới vì nhiều class khác cũng cần sử dụng hoặc là thêm một phương thức validateTimeOrder ở trong class Validate |
| 3 | AuthenticationController | Lớp sử dụng xác thực nhưng lại chứa các phương thức validate, phương thức algo md5 | không làm đúng nhiệm vụ của lớp, cần tách các phương thức ngoài nhiệm vụ của lớp sang các lớp khác |

### OCP

*<Nhận xét chung về tình trạng thiết kế và mã nguồn trên nguyên lý OCP>*

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | PaymentTransaction | lớp này có constructor PaymentTransaction nhưng lại truyền vào các giá trị cứng, nên chỉ thanh toán được với một loại card | Nếu tương lai phát sinh thêm các loại thanh toán bằng card mới thì sẽ dẫn đến việc phải modify lại code của class này |
| 2 | DeliveryInfo | Lớp này có method calculateShippingFee tính toán bằng 1 cách | Trong tương lai nếu phát sinh nhiều cách tính phí các thì hẳn nhiên phải thay đổi lại method này |
| 3 | payment/Card.java | Chỉ có 1 class áp dụng cho phương thức thanh toán bằng credit Card | Nếu phát sinh thêm phương thức thanh toán mới thì bắt buộc phải dùng 1 interface cho mọi phương thức thanh toán và class Card phải implement lại method đó |

### LSP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | controler.BaseControler | Lớp BaseCotroller được kế thừa nhiều nơi và có 2 method checkMediaInCart và getListCartMedia nhưng không được sử dụng mọi nơi | Các lớp kế thừa không sử dụng gây khó hiểu khó maintain |
| 2 | src/main/java/dao/media/MediaDAO.java | 3 lớp con BookDAO, CDDAO, DVDDAO chưa dùng được mọi hành vi của lớp cha | Nếu người dùng yêu cầu thêm phương thức getAllBook, getAllCD, getAllDVD nếu ko viết lại getAllMedia ở 3 lớp con thì ko thể thực hiện được |
| 3 | PaymentTransaction | Class này phụ thuộc vào một class không pha |  |

### ISP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 |  |  |  |
| 2 |  |  |  |

### DIP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | InterbankInterface | lớp này có phương thức payOrder có param là Card không phải một lớp trừu tượng | tương lai nếu thay đổi các loại Card dẫn đến việc sửa lại code, và lớp interface lại phụ thuộc vào một lớp rõ ràng |
| 2 | Order | Class này |  |
| 3 | PaymentTransaction | Class này phụ thuộc vào một class mà đã được extend từ một class khác | Thay thế lớp Card cho lớp CreditCard |

## Các vấn đề về Clean Code

### Clear Name

**-** Clean name trong codebase đầu còn nhiều chỗ chưa được rõ ràng

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Class* | *Mô tả* | *Chỉnh sửa* |
| 1 | AuthenticationController | Các Biến không mô tả rõ hoặc viết tắt | md5 -> md5Encryption |
| 2 |  | sb -> digestSB |
| 3 | BaseScreenHandler | Biến viết tắt | getBController  -> getBaseCotroller |
| 4 | InterbankSubsystem | Biến viết tắt dễ gây hiểu nhầm | ctrl -> interbankSubsystemController |
| 5 | PaymentController | Biến đặt tên không mô tả được nội dung | isValidMonthAndYear ->  isValidTimeOrder |

### Clean Function/Method

*< Nhận xét tình trạng mã nguồn ban đầu có đáp ứng clean function/method hay không?>*

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Class* | *Mô tả* | *Chỉnh sửa* |
| 1 | InterbankPayloadConverter | Các mã code chuyển từ 01-07 sang hằng số | Success = '01'  InvalidCard = '02',  ... |
| 2 | AIMSDB | Các string có trong code được chuyển về thành các hằng số | NAME\_JDBC  DATABASE\_URL |
| 3 | BaseScreenHandler | getBController | getBaseCotroller |
| 4 | ViewCartController | return bằng một dòng, không tạo thêm biến trung gian |  |

### Clean Class

*<Tương tự>*

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Class* | *Mô tả* | *Chỉnh sửa* |
| 1 | MediaDAO | Trong tương lai nhiều clients có thể sử dụng getMediaById | Tạo 1 interface MediaInterface rồi sau đó MediaDAO sẽ override lại getMedia(byId) |
| 2 | MediaDetailInterface | Class interface này không được sử dụng mà thay vào đó là một abstract class |  |
|  |  |  |  |
|  |  |  |  |

## Các vấn đề khác

Không có

# Đề xuất cải tiến

## Vấn đề clean code, coupling, cohesion, SOLID và giải pháp

* Clean class và clean method trong class: ***AuthenticationController***

Lớp ***AuthenticationController*** có nhiều vấn đề về Cohesion, và vi phạm nguyên lý SRP ở các phương thức md5, validation

Giải pháp giải quyết vấn đề ở đây là: Tạo mới một lớp dành riêng cho việc xử lý Security cho md5, một lớp dành riêng cho validation

Lớp Validate:

public class Validate {

private static Validate single\_instance = null;

public void validateDeliveryInfo(HashMap<String, String> info){}

public boolean validatePhoneNumber(String phoneNumber) {}

public boolean validateName(String name) {}

public boolean validateAddress(String address) {}

public boolean validateTimeOrder(int Month, int Year) {}

}

Hơn nữa nghiệp vụ của Validate chỉ cần một instance nên hoàn toàn sử dụng được Singleton

Lớp Security:

public class Security {

private static Security single\_instance = null;

public String md5Encryption(String message) {

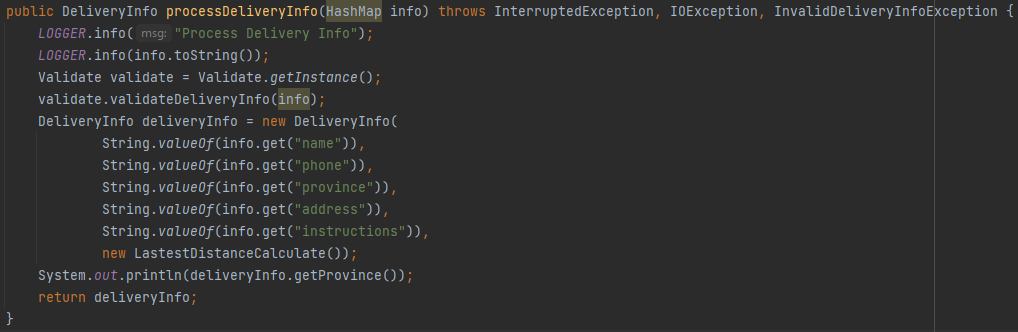
}

}

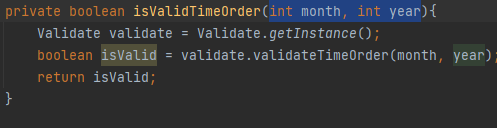
Lớp Security cũng tương tự như thế chỉ cũng chỉ cần một instance để thực hiện nhiệm vụ: sử dụng Singleton

* Clean class PaymentController, PlaceOrderController

Đặt lại tên biến và sử dụng class Validate cho PlaceOrderController:



Đặt lại tên biến và sử dụng class Validate cho PaymentController:



## Vấn đề tạo nhiều instance Cart và hướng giải quyết

Vì chỉ có một giả hàng nên sử dụng Singleton để giải quyết vấn đề:

public class Cart {

private static Cart instance;

private Cart() {

}

public synchronized static Cart getInstanse() {

if (instance == null) {

instance = new Cart();

}

return instance;

}

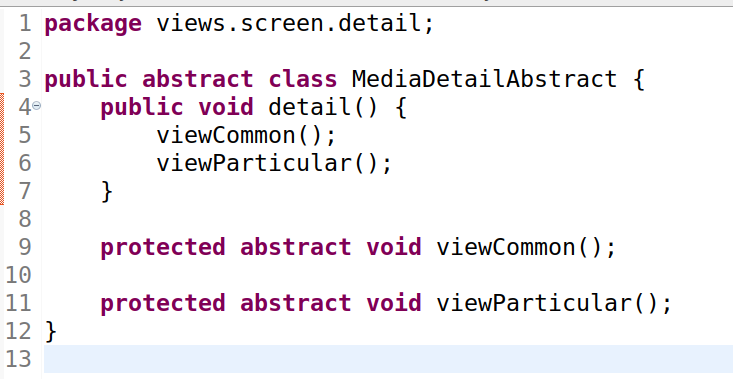
Singleton giúp giải quyết nghiệp vụ bài toán: Chỉ sinh ra duy nhất môt instance của một class

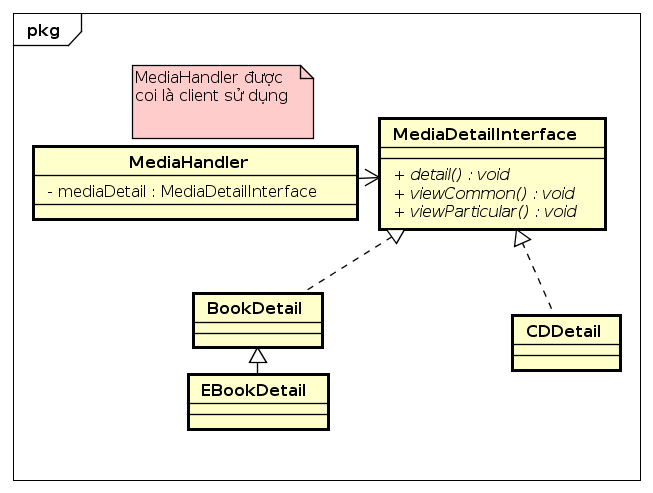
## Vấn đề xem chi tiết sản phẩm và hướng giải quyết

Khi thêm tính năng xem chi tiết sản phẩm, ta thấy thuật toán xem chi tiết sản phẩm khá giống nhau nên em đề nghị sử dụng template method để xử lý

Thiết kế của giải pháp:

MediaHandler là lớp xử lý hiển thị Media sẽ được coi là client, MediaDetailAbstract làm nhiệm vụ cung cấp method detail là trình tự xử lý để đưa ra detail của sản phẩm



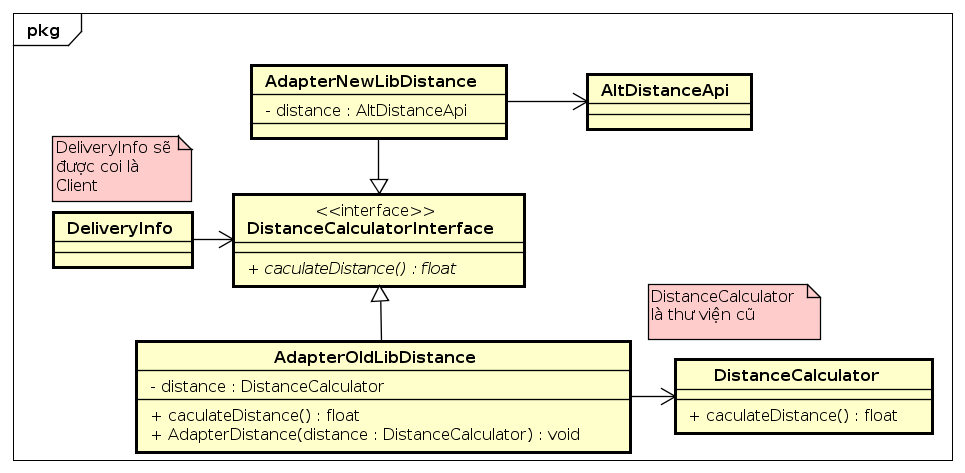


## Vấn đề thay đổi tính khoảng cách và cách giải quyết

***Đặt vấn đề: Thay đổi cách tính khoảng cách, sử dụng thư viện mới***

Ngoài vấn đề thay đổi tính khoảng cách, code tính khoảng cách đang bị vi phạm OCP vì code tính nằm ngay trong class DeliveryInfo,

=> Cách giải quyết vấn đề ứng dụng Adapter Design Pattern



Adapter Pattern giữ vai trò trung gian giữa hai lớp, chuyển đổi interface của một hay nhiều lớp có sẵn thành một interface khác, thích hợp cho lớp đang viết.

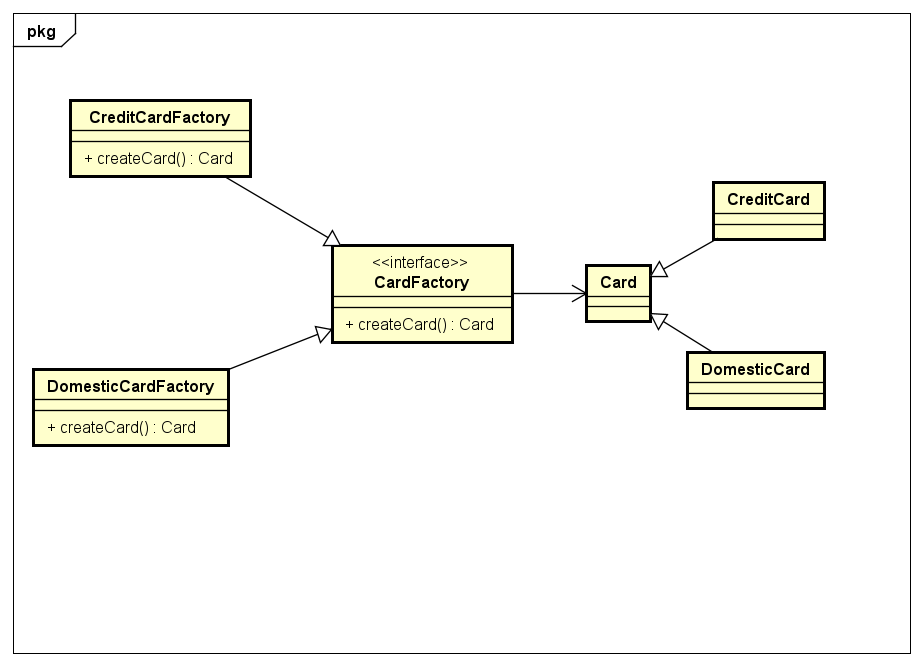
Ở đây ta thấy khi cập nhật một loại Lib cho Distance mới thì interface có thể bị thay đổi, nên ta sử dụng Adapter Design Pattern thì Client (DeliveryInfo) sẽ không phải quan tâm Lib thay thế là gì, việc nó quan tâm là giao tiếp với DistanceCalculatorInterface

## Vấn đề 5 và cách giải quyết

Thêm phương thức thanh toán mới

Khi thêm phương thức thanh toán mới sẽ có các thuộc tính khác nhau với mỗi loại thẻ. Mỗi lần thay đổi phương thức thanh toán

Vấn đề này có thể được giải quyết bằng cách tạo ra một interface CardFactory 1 lớp cha Card, các lớp CreditCard, DomesticCard kế thừa Card còn CreditFactory và DomesticCardFactory implement interface CardFactory. Như vậy sau này có thêm phương thức thanh toán mới cũng tiện hơn trong việc code.

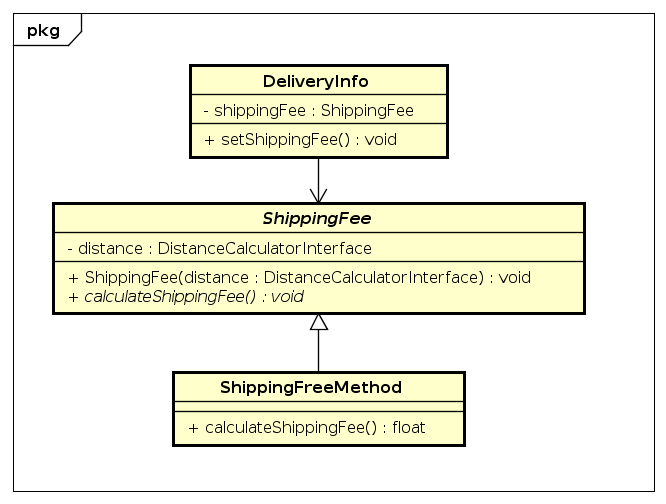


## Vấn đề thêm mới cách tính phí vận chuyển và cách giải quyết

Thêm mới phí vận chuyển cũng làm vi phạm OCP của class DeliveryInfo, vì khi thay đổi sẽ phải trực tiếp trong class này

Khi thêm mới cách tính phí vận chuyển, ta có một solution đơn giản như sau:

Tạo một Abstract class ShippingFee, có method calculateShippingFee để DeliveryInfo (coi là Client) có thể sử dụng được, việc của Client là sử dụng phương thức tính như nào thì truyền vào class tương ứng, và lib để tính khoảng cách tương ứng

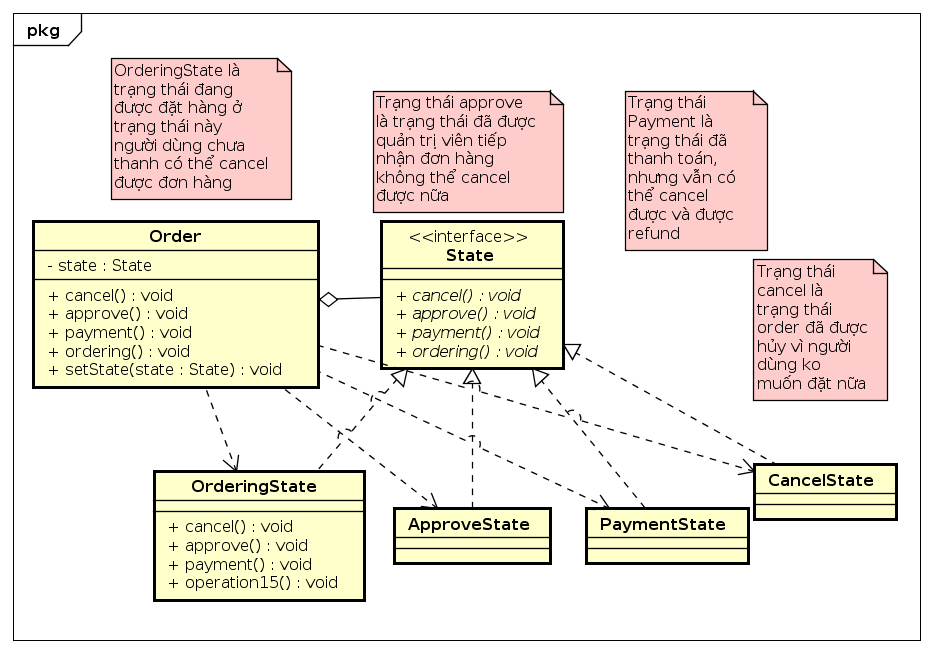


## Vấn đề cập nhật chức năng hủy đơn hàng và cách giải quyết

Cập nhật chức năng hủy đơn hàng: Một đơn hàng có nhiều trạng thái như: Đang order, đã order đã thanh toán, hủy hàng, đã chấp nhận, với yêu cầu đã chấp nhận sẽ không được phép hủy hàng nữa.

Với các trạng thái như vậy của đơn hàng cách dễ nhất là sử dụng State Design Pattern cập nhật trạng thái của đơn hàng mỗi khi có State mới

Thiết kế của giải pháp:



Order sẽ có các method để người dùng sử dụng: cancel, approve,...

Client sẽ truyền vào state khi khởi tạo hoặc truyền vào các State của đơn hàng, khi đó các State sẽ tự thực hiện các công việc của nó

## Vấn đề xóa item trong Cart

Vấn đề:

Mọi app thương mại điện tử khác, khi hàng đã ở trong giỏ, trước khi xóa mặt hàng đó, hệ thống sẽ hỏi lại "Bạn có chắc chắn việc xóa vật phẩm này không"

Ngoài ra còn có vấn đề về coupling, hai lớp CartScreenHandler và MediaHandler phụ thuộc nhiều vào nhau nếu có sự cập nhật từ CartScreenHandler

=> Giải quyết vấn đề: Sử dụng Observer Design Pattern để đăng ký CartScreenHandler với MediaHandler để có thể thông báo cho người dùng biết mỗi khi có sự update ở MediaHandler

Ở đây lớp MediaHandler sẽ được coi là lớp Observable hay còn là Subject sẽ duy trì CartScreenHandler là lớp observe, mỗi khi data Media bị thay đổi, CartScreenHandler sẽ nhận được thông mới từ MediaHandler truyền sang và CartScreenHandler sẽ hiển thị ra hoặc thông báo ra các update này

Lớp CartScreenHandler:

@Override

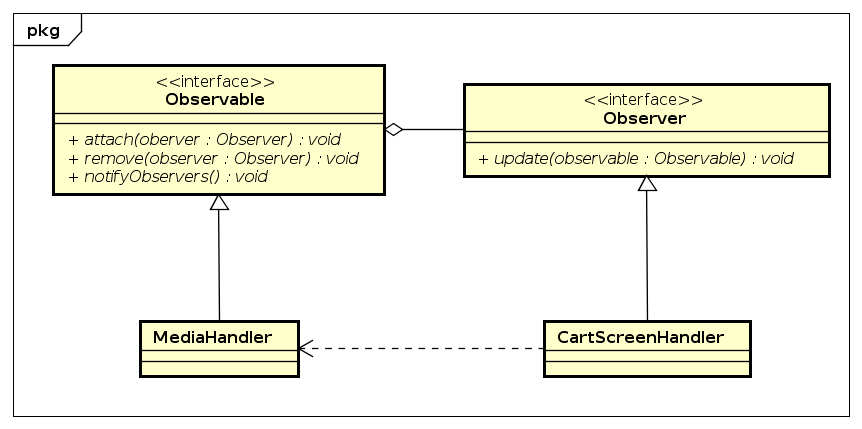
public void update(Observable observable) {

if (observable instanceof MediaHandler) {

// TODO Update Value, anything of Cart in here

}

}



# Tổng kết

## Kết quả tổng quan

*<So sánh với dự kiến ban đầu, kết quả đã thực hiện được hoàn thiện được bao nhiêu kỳ vọng*>

Với yêu cầu môn học, thời lượng, kiến thức tiếp thu được cũng như nhân lực trong nhóm cố gắng hoàn thành mini-project chúng em tự tin nhận thấy cải thiện được khá nhiều tính năng dựa theo các yêu cầu mới đưa ra. Tuy vậy mới chỉ là các yêu cầu đưa ra chưa đủ để có thể trở thành một sản phẩm hoàn chỉnh. Nhóm vẫn cần bỏ ra nhiều thời gian công sức cũng như đầu tư một cách rõ ràng hơn để có thể làm được sản phẩm tốt hơn.

Đánh giá: Hoàn thành 80% yêu cầu đặt ra

## Các vấn đề tồn đọng

- Chưa tìm được hết các giải pháp clean code

- Chưa giải quyết được vấn đề vi phạm OCP ở lớp PaymentTransaction