ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

Version 1.0

Nhóm DP.20202.09

Danh sách thành viên

|  |  |
| --- | --- |
| Họ và tên | MSSV |
| Lương Tiến Mạnh | 20173254 |
| Hoàng Văn Mạnh | 20173252 |
| Hoàng Trọng Minh | 20173259 |
| Đỗ Hoàng Nam | 20173270 |

*Hà Nội, tháng 6, năm 2021*

*<Tất cả chú thích trong dấu ngoặc nhọn không thuộc về nội dung của báo cáo cuối, chỉ sử dụng nhằm mục đích hướng dẫn. Khi sử dụng mẫu báo cáo này, cần loại bỏ hết tất cả chú thích và/hoặc thay thế bằng nội dung tương ứng với chú thích.>*

*<Mẫu báo cáo này thuộc về GV. Nguyễn Thị Thu Trang và GV. Bùi Thị Mai Anh, được sử dụng cho các sinh viên tham gia môn Mẫu thiết kế phần mềm. Bất kỳ chỉnh sửa và/hoặc sử dụng nằm ngoài sự đồng ý của tác giả đều hoàn toàn nghiêm cấm.>*

**Mục lục**

Mục lục 1

1 Tổng quan 3

1.1 Mục tiêu 3

1.2 Phạm vi 3

1.3 Danh sách thuật ngữ 3

1.4 Danh sách tham khảo 3

2 Đánh giá thiết kế cũ 4

2.1 Nhận xét chung 4

2.2 Đánh giá các mức độ coupling và cohesion 4

2.2.1 Coupling 4

2.2.2 Cohesion 4

2.3 Đánh giá việc tuân theo SOLID 4

2.3.1 SRP 5

2.3.2 OCP 5

2.3.3 ... 5

2.4 Các vấn đề về Clean Code 5

2.4.1 Clear Name 5

2.4.2 Clean Function/Method 5

2.4.3 Clean Class 5

2.5 Các vấn đề khác 6

3 Đề xuất cải tiến 7

3.1 Vấn đề xxx và giải pháp 7

3.2 Vấn đề yyy và giải pháp 7

4 Tổng kết 8

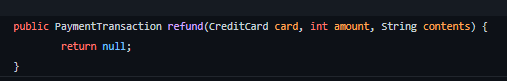
4.1 Kết quả tổng quan 8

4.2 Các vấn đề tồn đọng 8

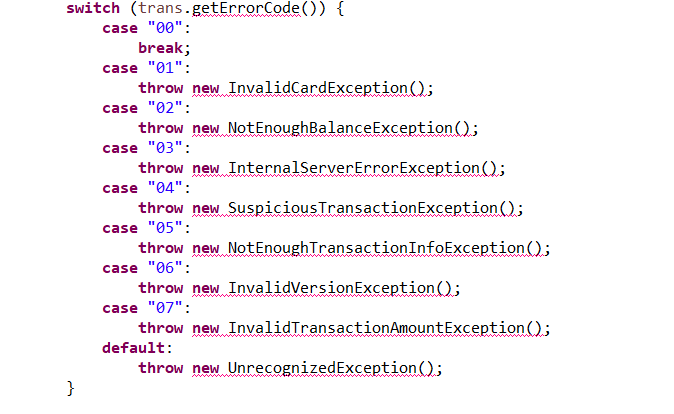
**Danh sách các minh họa**

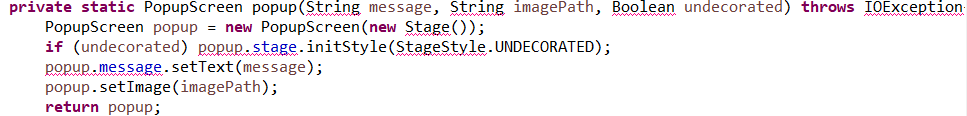


Hình 1.1

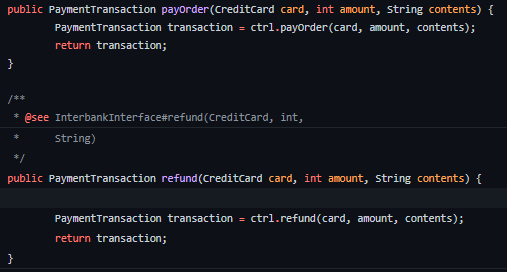


Hình 1.2

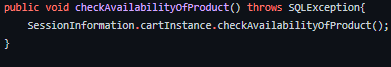
Hình 1.3



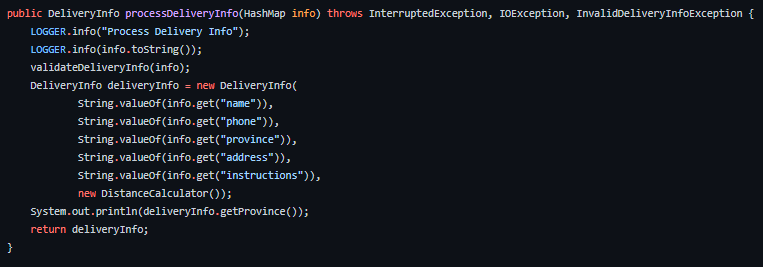
Hình 1.4



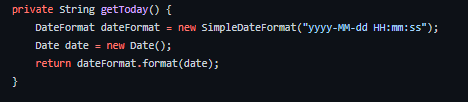
Hình 1.5



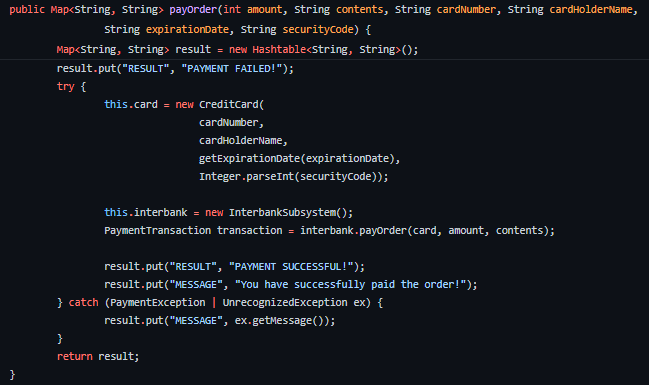
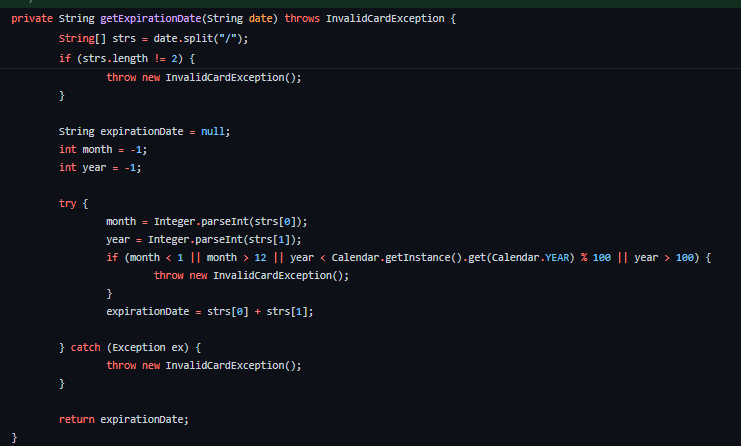
Hình 1.6



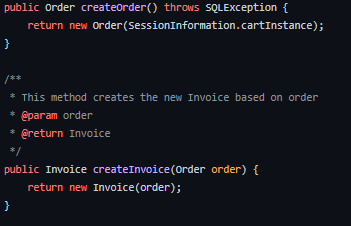
Hình 1.7



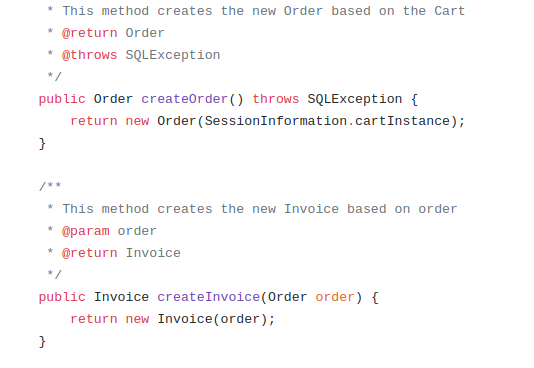
Hình 1.8



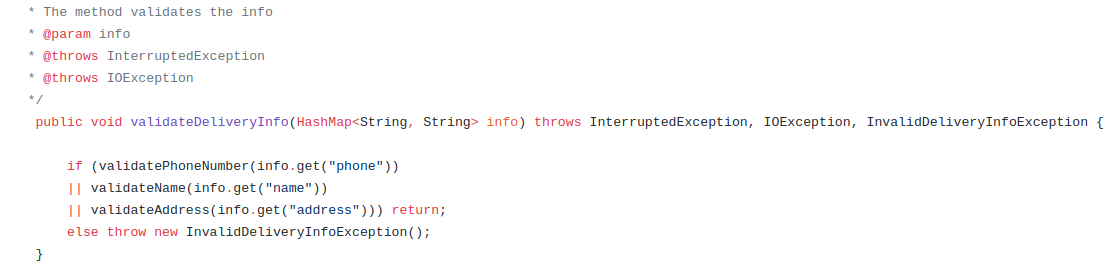
Hình 1.9



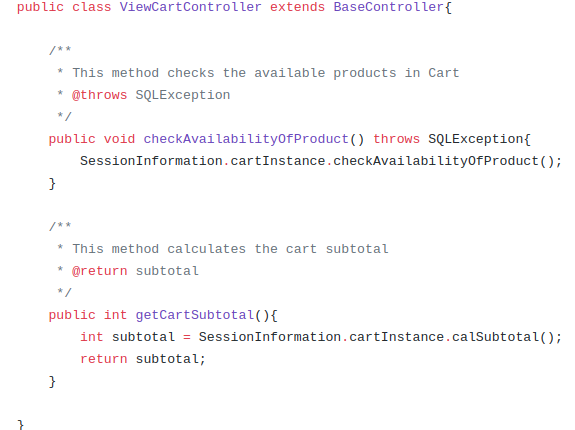
Hình 1.10



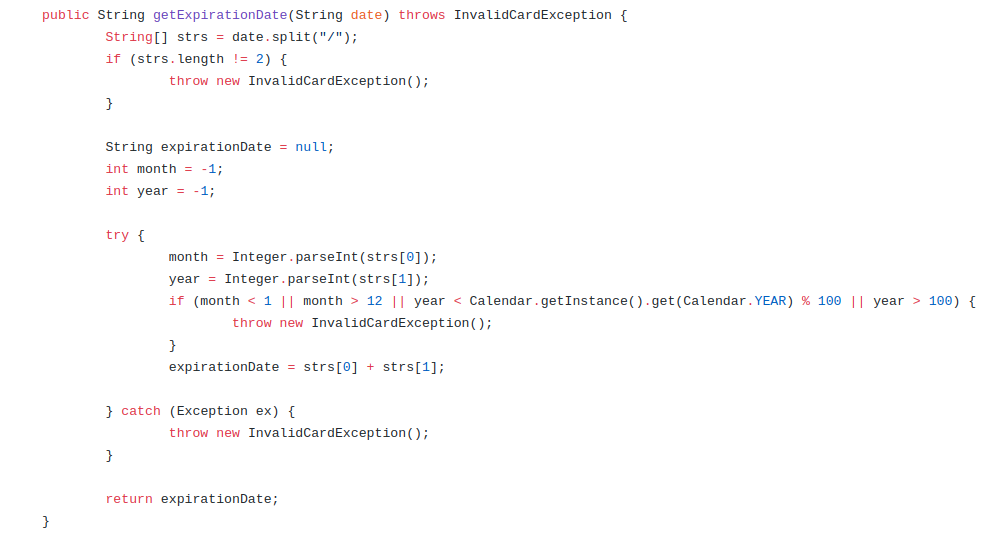
*Hình 2.1*



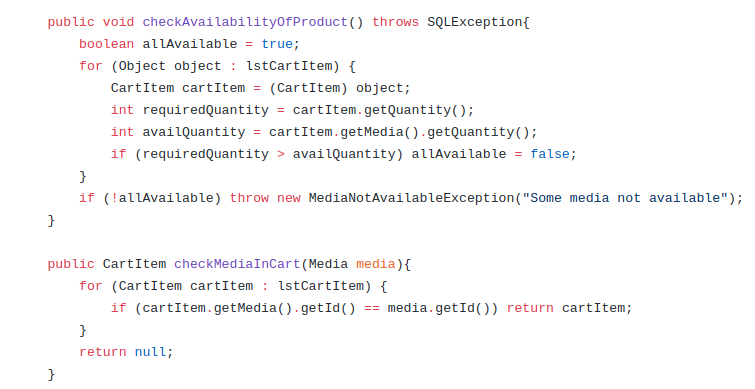
Hình 2.2



*Hình 2.3*



*Hình 2.4*



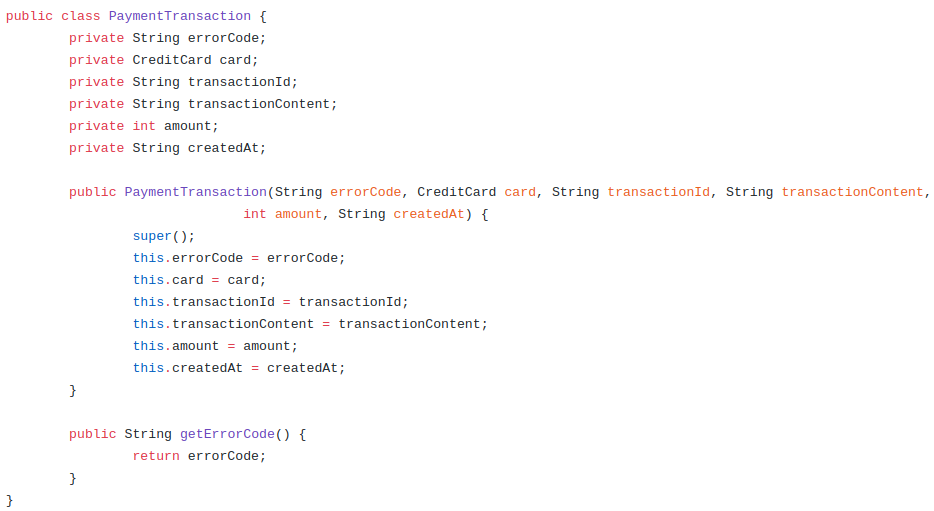
*Hình 2.5*



*Hình 2.6*



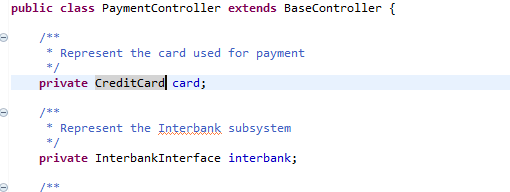
*Hình 2.7*



*Hình 2.8*



*Hình 2.9*

*Hình 2.10*

**Danh sách các bảng**

# Tổng quan

## Mục tiêu

Tổng hợp các vấn đề còn tồn đọng về thiết kế và code của mã nguồn cũ. Từ đó đưa ra giải pháp để cải tiến về thiết kế và tái cấu trúc mã nguồn giúp project tốt hơn.

Đối tượng: Developer, Giáo viên, người phân tích thiết kế hệ thống, ...

## Phạm vi

* Mô tả khái quát phần mềm: Đây là phần mềm có tên là An Internet Media Store (AIMS) giúp người dùng mua hàng là các mặt hàng phương tiện truyền thông online.
* Chức năng chính:
  + Đăng nhập
  + Xem các mặt hàng
  + Thêm vào giỏ hàng
  + Xem giỏ hàng
  + Đơn hàng
  + Thanh toán qua thẻ
  + Đặt hàng
* Cấu trúc mã nguồn
* Các yêu cầu thêm cần cân nhắc:
  + Thay đổi cách tính khoảng cách, sử dụng thư viện mới
  + Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)
* Các hoạt động:
  + Đọc hiểu code
  + Báo cáo file word ghi rõ: Mã nguồn cũ có vấn đề gì, cải tiến như thế nào? Nên sử dụng mẫu thiết kế nào?
  + Thêm file astah cho thiết kế mới
  + Refactor code
  + Các thành viên review lại code
* Kết quả dự kiến:
  + Code chạy không bị lỗi
  + Code sạch
  + Giảm thiểu hết các coupling, cohesion, không vi phạm SOLID
  + Code đúng mẫu thiết kế mới đã đề ra

## Danh sách thuật ngữ

* Refactor: Cấu trúc lại
* AIMS: An Internet Media Store

## Danh sách tham khảo

1. Centers for Medicare & Medicaid Services. (n.d.). *System Design Document Template.* Retrieved from Centers for Medicare & Medicaid Services: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SystemDesignDocument.docx

2. Cornell University How We Refactor and How We Document it? On the Use of Supervised Machine Learning Algorithms to Classify Refactoring Documentation

Retrived from www.elsevier.com/locate/eswa

# Đánh giá thiết kế cũ

Đánh giá:

* Thiết kế cũ kém hiệu quả khiến cho việc thêm một số chức năng gặp nhiều khó khăn, phải chỉnh sửa nhiều code cũ

## Nhận xét chung

* Code đã được refactor đạt đúng yêu cầu đề ra
* Việc update các mục khác, các yêu cầu khác sẽ không ảnh hưởng nhiều đến code cũ
* Developer đọc hiểu code dễ hơn, đỡ mất thời gian và tiện lợi cho việc update thường xuyên và lâu dài.

## Đánh giá các mức độ coupling và cohesion

Coupling và cohesion là 2 khái niệm trong thiết kế phần mềm để chỉ sự tương tác, liên kết với nhau trong các module hay các component trong 1 module.

* ***Coupling***: mức độ coupling giữa các module càng thấp, software càng được đánh giá cao và độ linh động giữa các module.

Có 5 khái niệm chính trong Coupling được sắp xếp theo thứ tự từ cao xuống thấp để đánh giá các cấp độ liên kết, tương tác giữa các module:

* + Content coupling: Xảy ra khi 1 module sử dụng code của 1 module khác, điều này sẽ không đúng với những khái niệm thiết kế cơ bản là đóng gói thông tin
  + Common coupling: Xảy ra khi các module cùng truy cập vào cùng data global, và điều này có thể dẫn đến việc không thể kiểm soát được lỗi và sẽ xảy ra những lỗi không mong muốn.
  + Control coupling: là khi 1 module đang control luồng hoạt động của 1 module khác thông qua các tham số đầu vào, và nó có thể dẫn đến 1 vài trường hợp không mong muốn khi các tham số truyền vào không được đúng.
  + Stamp coupling: xảy ra khi các module chỉ sử dụng các thành phần cấu trúc dữ liệu được chia sẻ với nhau, và trong trường hợp này, khi thay đổi 1 thuộc tính của 1 module có thể sẽ ảnh hưởng đến các module khác đang phụ thuộc vào nó.
  + Data coupling: xảy ra khi 2 hoặc nhiều module độc lập với nhau, tương tác, chia sẻ tài nguyên cho nhau và thông thường là qua tham số đầu vào.
* ***Cohesion***: nhiệm vụ của từng module càng rõ ràng và tách biệt thì cohesion càng cao và đó là mục tiêu cần đạt tới khi thiết kế.

Có 5 khái niệm chính trong Cohesion được sắp xếp theo thứ tự từ cao xuống thấp:

* + Functional cohesion: Khi các phần của 1 module được nhóm lại vì tất cả chúng đề đóng góp vào 1 nhiệm vụ được xác định rõ ràng của module.
  + Communication/informational cohesion: Khi các phần của module được nhóm lại vì chúng hoạt động trên cùng một dữ liệu.
  + Procedural cohesion: Khi các phần tử của module được nhóm lại vì chúng luôn tuân theo 1 trình tự thực thi nhất định.
  + Temporal cohesion: Các phần của module được nhóm lại theo thời điểm chúng được xử lý - các phần được xử lý tại một thời điểm cụ thể trong quá trình thực thi chương trình.
  + Logical cohesion: Khi các phần của module được nhóm lại vì chúng được phân loại một cách hợp lý để làm cùng một việc mặc dùng chúng khác nhau về bản chất.
  + Coincidental cohesion: Khi các phần của module được nhóm một cách tuỳ ý, mối quan hệ duy nhất giữa các phần là chúng đã được nhóm lại với nhau.

### Coupling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Lý do* |
| 1 | Stamp Coupling | entity/cart/Cart.java | Hình 1.1 | Method checkMediaInCart() chỉ sử dụng đến mediaId việc truyền cả Media là thừa thãi |
| 2 | Stamp Coupling | subsystem/interbank | Hình 1.2 | Method refund truyền param nhưng không sử dụng |
| 3 | Control coupling | subsystem/interbank/InterbankPayloadConverter | Hình 1.3 | Sử dụng Switch case |
| views/screen/popup/PopupScreen | Hình 1.4 | Su dung tham so undecorated lam tham so dieu khien |

### Cohesion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Lý do* |
| 1 | Communicational Cohesion | /subsystem/InterbankSubsystem | Hình 1.5 | Các param truyền vào payOrder và refund và output, input 2 method này là giống nhau |
| 2 | Coincidental Cohesion | controller/ViewCartController | Hình 1.6 | checkAvailabilityOfProduct nên phụ thuộc vào từng Product |
| controller/PlaceOrderController | Hình 1.7 | Các hàm validateString() được đặt trong class để thực hiện method processDeliveryInfo() mà về chức năng chúng không liên quan đến nhau |
| subsystem/interbank/InterbankPayloadConverter | Hình 1.8 | getToday() không liên quan các method còn lại nên đặt trong common |
| 3 | Procedural Cohesion | controller/PaymentController | Hình 1.9 | Các hàm getExpirationDate() được đặt trong class để thực hiện method payOrder() mà về chức năng chúng không liên quan đến nhau |
| controller/PlaceOrderController | Hình 1.7 | Các hàm validateString() được đặt trong class để thực hiện method processDeliveryInfo() mà về chức năng chúng không liên quan đến nhau |
| 4 | Temporal Cohesion | controller/PlaceOrderController | Hình 1.10 | createOrder() and createInvoice() chỉ có ý nghĩa với nhau về mặt thời gian |

## Đánh giá việc tuân theo SOLID

Nếu phải chỉnh sửa mã nguồn theo các yêu cầu phát sinh thì bản thiết kế và mã nguồn ban đầu không còn tuân thủ các nguyên lý thiết kế SOLID. Các vi phạm được trình bày ở dưới đây.

### SRP

*Các class PlaceOrderController, ViewCartController, PaymentController, Cart đang thực hiện nhiều việc mà bản chất không phải class đó đảm nhiệm*

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | controller/PlaceOrderController.java | *Hình 2.1*  *Hình 2.2* | Vi phạm SRP vì PlaceOrderController thực hiện hơn một tác vụ vừa tạo order, invoice, xử lý , validate deliveryInfo |
| 2 | controller/ViewCartController.java | *Hình 2.3* | Class đang thực hiện thêm checkAvailabilityOfProduct() mà bản chất không phải nhiệm vụ class |
| 3 | controller/PaymentController.java | *Hình 2.4* | Lớp thực hiện cả việc getExpirationDate() |
| 4 | entity/cart/Cart.java | *Hình 2.5* | việc checkAvailabilityOfProduct() nên để cho media tự kiểm tra số lượng |

### OCP

*Khi mà có thêm các yêu cầu như thêm loại thẻ thanh toán mới, thêm cách tính khoảng cách , phí ship các class PaymentController, DeliveryInfo, InterbankPayloadConverter, PaymentTransaction đều phải thay đổi.*

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | controller/PaymentController.java | *Hình 2.10* | + Khi thêm các ngân hàng mới (bank) , các loại thẻ mới (card) lớp payOrder phải modified |
| 2 | entity/shipping/DeliveryInfo.java | *Hình 2.6* | Khi thêm cách tính distance mới việc thay đổi lại code là thiết yếu, và việc giao tiếp cụ thể phải chỉnh sửa |
| 3 | subsystem/interbank/InterbankPayloadConverter.java | *Hình 2.7* | Thêm card mới phải sửa lại code |
| 4 | entity/payment/PaymentTransaction.java | *Hình 2.8* | Khi ta thêm mới một card thì việc refactor lại code là chắc chắn |

### LSP

*Ở class InterbankSubsystem có method refund nếu chỉ thanh toán mà không trả lại thì method bị dư thừa*

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | subsystem/InterbankSubsystem.java | *Hình 2.9* | Khi cửa hàng chỉ thanh toán mà không trả lại, thì method refund sẽ dư thừa |

### ISP

* Chưa tìm thấy trong codebase có trường hợp vi phạm ISP

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

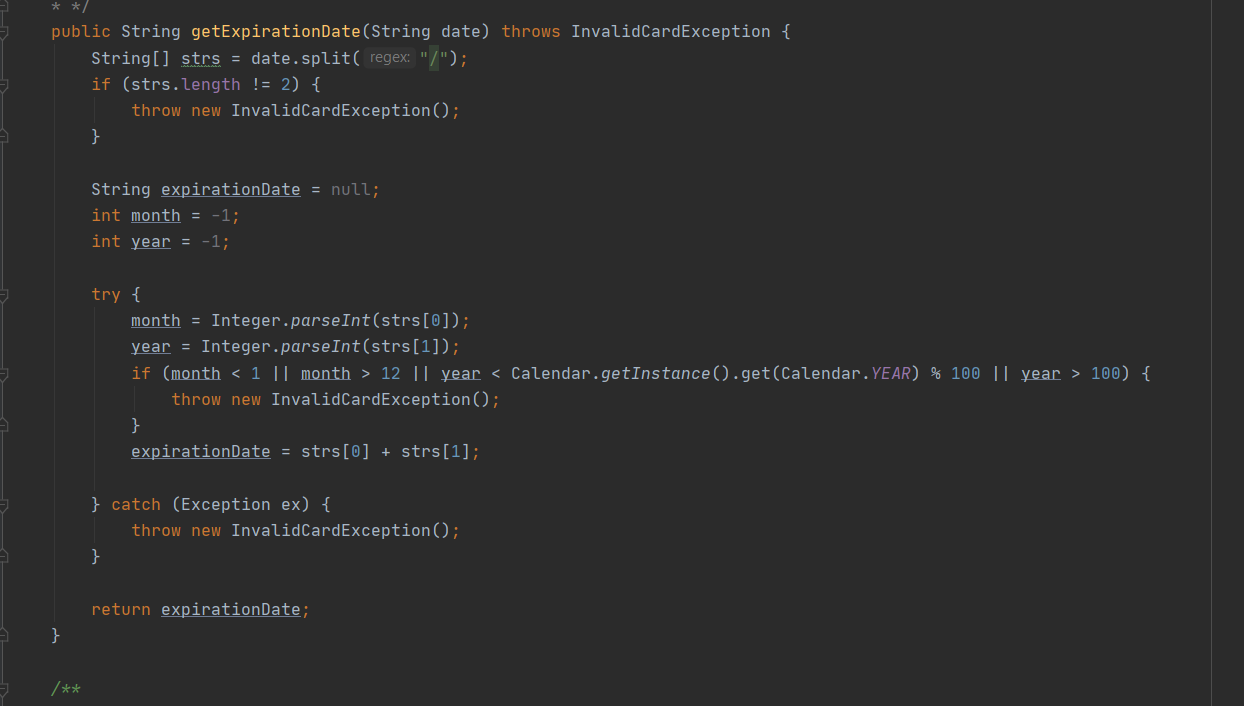
### DIP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | controller/PaymentController.java | *Hình 2.10* | Phải giao tiếp với nhau qua AbstractClass hoặc Interface( ở đây lại gọi thẳng class CreditCard) |
| 2 | entity/payment/PaymentTransaction.java | *Hình 2.8* | Khi ta thêm mới một card thì việc refactor lại code là chắc chắn, Do vậy chỉ nên giao tiếp qua interface ở card |
| 3 | entity/shipping/DeliveryInfo.java | *Hình 2.6* | Khi thêm cách tính distance mới việc thay đổi lại code là thiết yếu, vì vậy các class giao tiếp cụ thể phải chỉnh sửa |
| 4 | subsystem/InterbankSubsystem.java | *Hình 2.9* | Khi có card mới phải thay đổi code -> Nên giao tiếp qua interface |
| 5 | subsystem/interbank/InterbankPayloadConverter.java | *Hình 2.7* | Nên giao tiếp qua interface vì khi thêm card mới phải sửa lại code cũ |

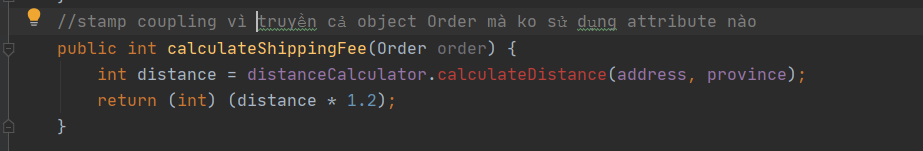
## Các vấn đề về Clean Code

### Clean Function/Method

1. Hiện tại trong source code có một số giá trị vẫn các giá trị cụ thể chưa để thành các const.



Hình ảnh trên là một ví dụ về Name Convention



Một ví dụ khác trong Order Class hệ số 1.2 nên để constant.

1. Giải pháp:

Chuyển tất cả các giá trị nguyên thủy thành các biến const để dễ kiểm soát và tránh trường hợp sai lệch khi modify.

### Clean Name

* Hiện tại về Name được đảm bảo về convention.

### Clean Class

Với nghiệp vụ hiện tại thì thì mã nguồn ban đầu đảm bảo được clean class.

Nhưng khi yêu cầu thêm các vấn đề ở mục 2.5 thì Clean Class không còn được đảm bảo nữa. Cụ thể

* Lớp Payment Controller giao tiếp trực tiếp với Credit Card. Khi thêm loại card mới sẽ phải refactor lại code và không đảm bảo OCP, DIP. Ảnh hưởng các class của InterBank.
* Lớp DeliveryInfo chứ một class cụ thể DistanceCalculatort. Và khi mở rộng thêm một cách tính phí Ship mới cũng không đảm bảo và phải mở rộng lớp.
* Quá nhiều lớp phụ thuộc CreditCard, Khi có một phương thức thanh toán mới tất cả các class phải thay đổi.

## Các vấn đề khác

|  |  |  |  |
| --- | --- | --- | --- |
| *STT* | *Module* | *Vấn đề* | *Mô tả cụ thể* |
| 1 | Media | Thêm một loại Media mới là Audio Book | Khi thêm một loại AudioBook mới phải đảm bảo không bị vi phạm các nguyên tắc coupling, cohesion, SOLID |
| 2 | Screen | Thêm chức năng xem chi tiết của một sản phẩm. | Khi ở màn chi tiết. Với mỗi sản phẩm thêm một button “Xem chi tiết”. Khi click vào “Xem chi tiết” sẽ hiển thị ra một trang chi tiết của sản phẩm chứa đầy đủ thông tin chi tiết của sản phẩm đấy, trong trang chi tiết có một button “Add to cart” để thêm sản phẩm vào giỏ hàng và cũng lập tức quay trở lại màn danh sách sản phẩm. |
| 3 | Screen | Thay đổi yêu cầu khi load giao diện lỗi | Hiện tại khi xảy ra lỗi (Exception) thì chương trình xuất hiện các Popup lỗi cảnh báo. Bây giờ mong muốn thêm các cách hiển thị thông báo lỗi mới. Thay vì hiển thị popup sẽ hiển thị một dòng chữ đỏ trên đầu trang là thông báo lỗi. Lỗi phải cụ thể của các màn. Vd: Với màn A lỗi “Màn A có lỗi” |
| 4 | Payment | Thay đổi cách tính khoảng cách sử dụng thư viện mới và công thức mới | Hiện tại đang sử dụng thư viện DistanceCalcultor để tính phí Ship.  Nhưng bây giờ add thêm một thư viện tính phí ship theo cả độ cồng kềnh của sản phẩm. |
| 5 | Card | Thêm một loại hình thanh toán mới. | Hiện tại hệ thống đang chỉ thanh toán bằng CreditCard. Mở rộng thêm một cách thanh toán mới là DomesticCard  Với các thông tin khác với CreditCard |
| 6 | Invoice | Cập nhật lại chức năng hủy đơn hàng | Theo chức năng cũ, sau khi hoàn thành đặt hàng có thể theo dõi trạng thái đơn hàng của mình, Bên cạnh đó, khách hàng có thể hủy đơn hàng và được hoàn lại tiền bằng giá trị đơn hàng.  Thêm chức năng hủy đơn hàng . Sẽ bị vô hiệu hóa khi đơn hàng được quản trị viên “chấp nhận”. Nếu đơn hàng ở trạng thái “chờ phê duyệt”, khách hàng vẫn có thể hủy đơn hàng. |

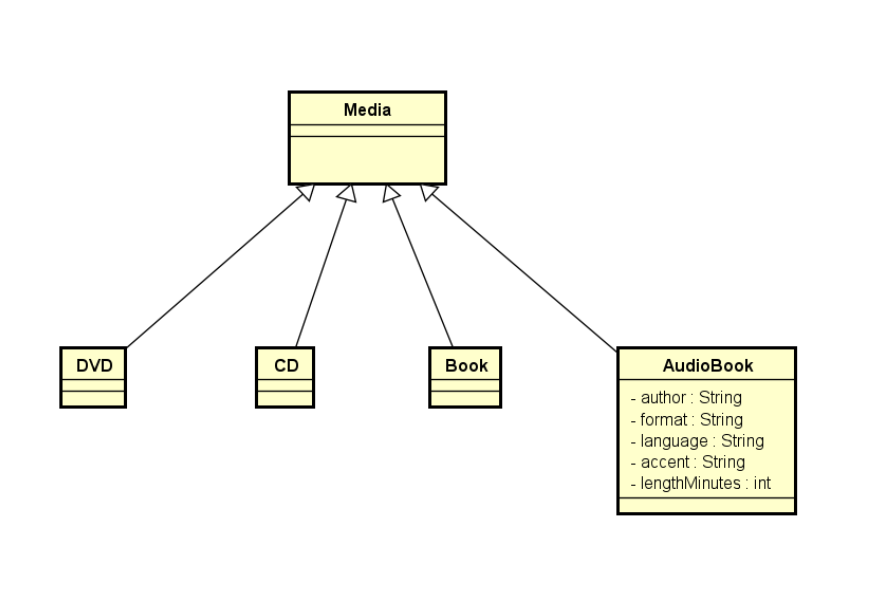
Đề xuất cải tiến

## Vấn đề 1 - Thêm một loại Media mới là AudioBook

2.6.1. Giải pháp

Tạo ra lớp mới AudioBook kế thừa từ lớp Media đã có sẵn.

2.6.2. Biểu đồ



2.6.3. Mô tả mã nguồn

public class AudioBook extends Media{

private String author;

private String format;

private String language;

private String accent;

private int lengthMinutes;

// contructor

// getter and setter

}

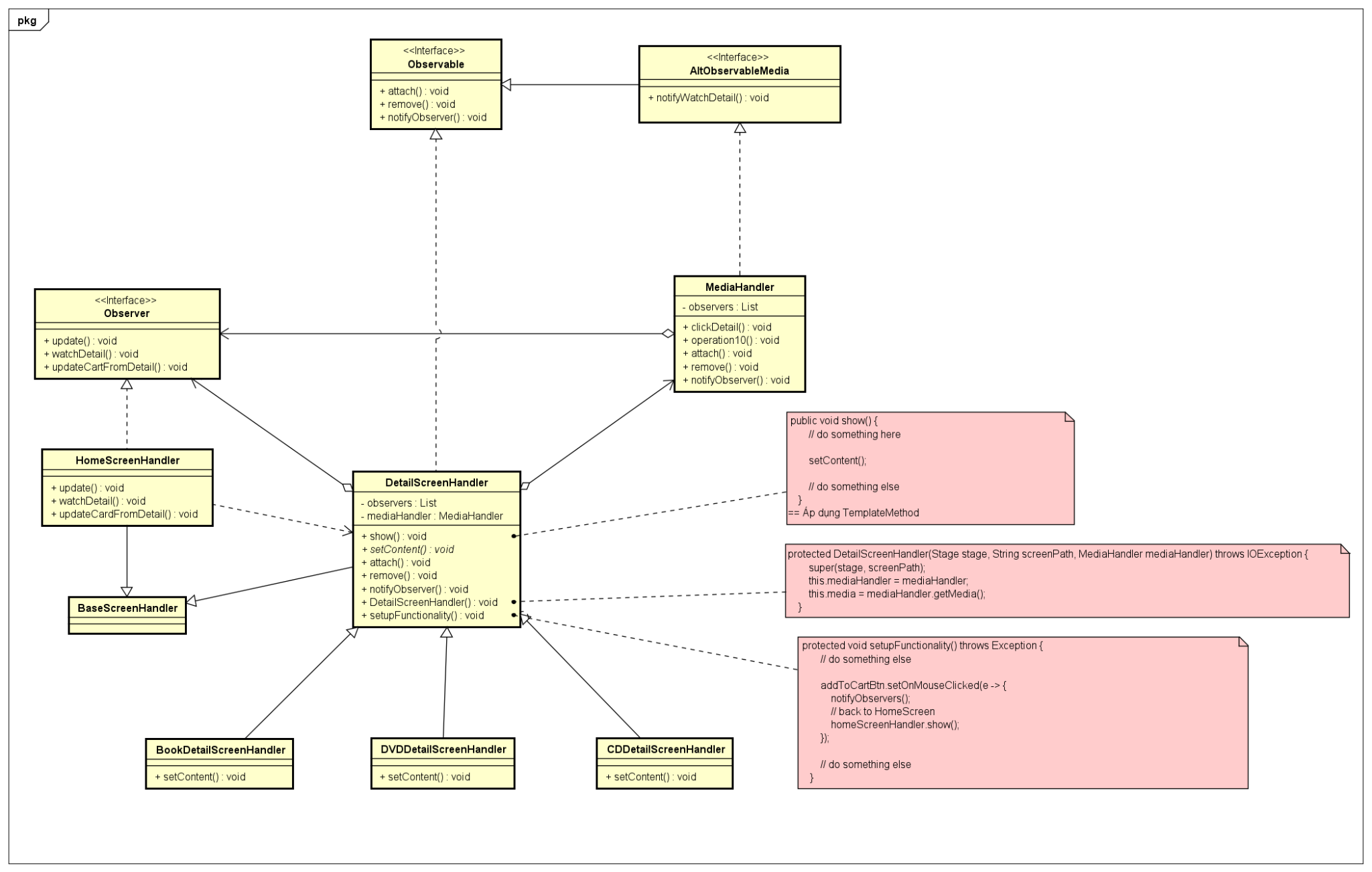
## Vấn đề 2 - Thêm chức năng xem chi tiết của sản phẩm

2.7.1. Giải pháp

Khi có thêm chức năng chi tiết thêm một button vào trong lớp MediaHandler - Lớp này thông tin của mỗi sản phẩm trên màn chi tiết. Ngoài ra trong màn chi tiết cũng có sự kiện “AddToCart” .Giải pháp ở đây chúng ra nghĩ đến Observer Pattern.

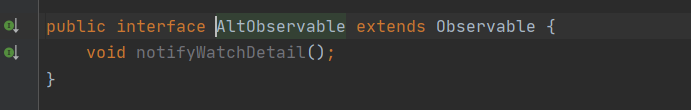
Màn Home sẽ subscribe MediaHandler và DetailScreen . Khi có sự kiện AddToCard sẽ thêm sản phẩm vào giỏ hàng. Nếu sự kiện “Xem chi tiết” thì màn Home điều hướng trang chi tiết sản phẩm. Với từng loại sản phẩm sẽ có một trang chi tiết riêng. Ở đây ta sẽ áp dụng Template Pattern cho từng trang chi tiết của sản phẩm. Lý do : Các trang chi tiết sản phẩm chỉ khách nhau setContent(), còn lại với phương thức khởi tạo, setBasicContent sẽ là như nhau.

2.7.2. Biểu đồ



2.7.3. Mô tả mã nguồn

* MediaHandler thêm một event notifyWatchDetail() do đó tạo một interface AltObservableMedia kế thừa Observable đã có đảm bảo tính ISP trong SOLID, và MediaHandler implement lớp AltObservableMedia



Class MediaHandler extends FXMLScreenHandler implements AltObservable {

@Override

Public void attach (Observer observer){ obserserList.add(observer)}

@Override

Public void remove(Observer observer){ obserserList.remove(observer)}

@Override

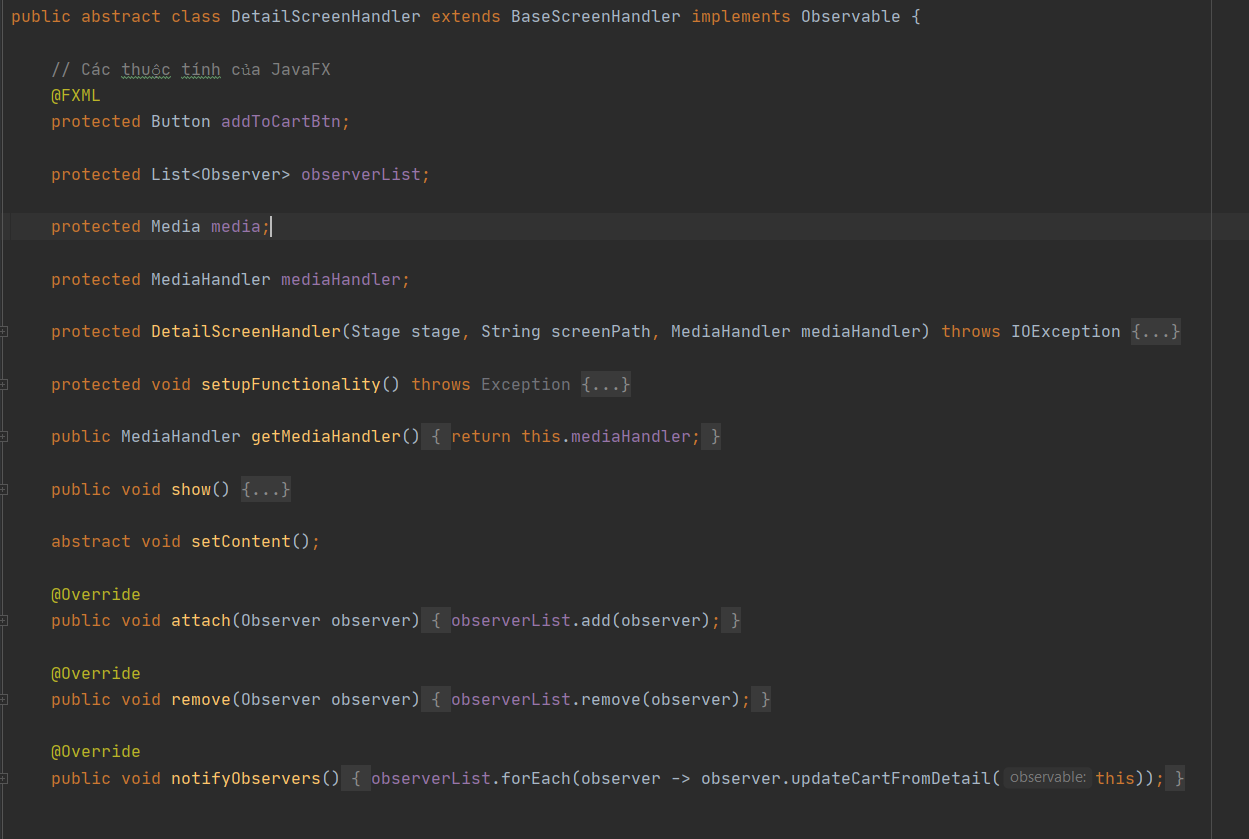
Public void notifyObservers(){observerList.forEach(observer -> observer.update(this))}

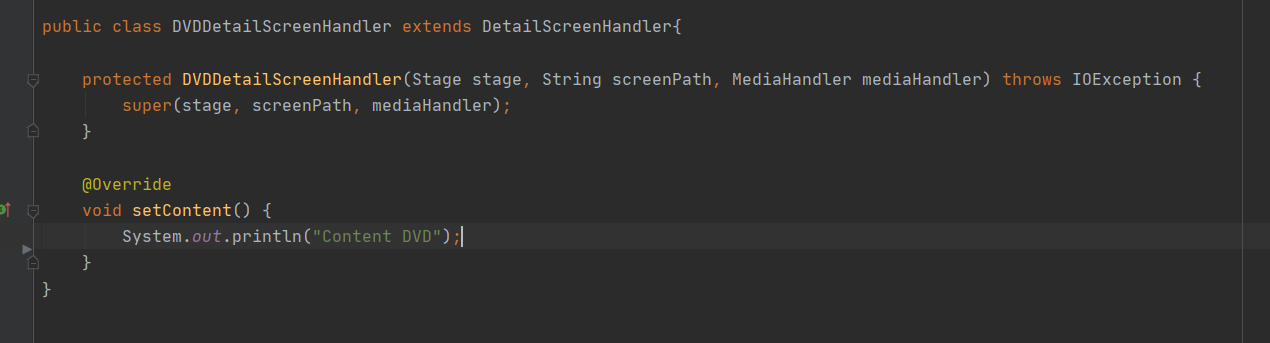
@Override

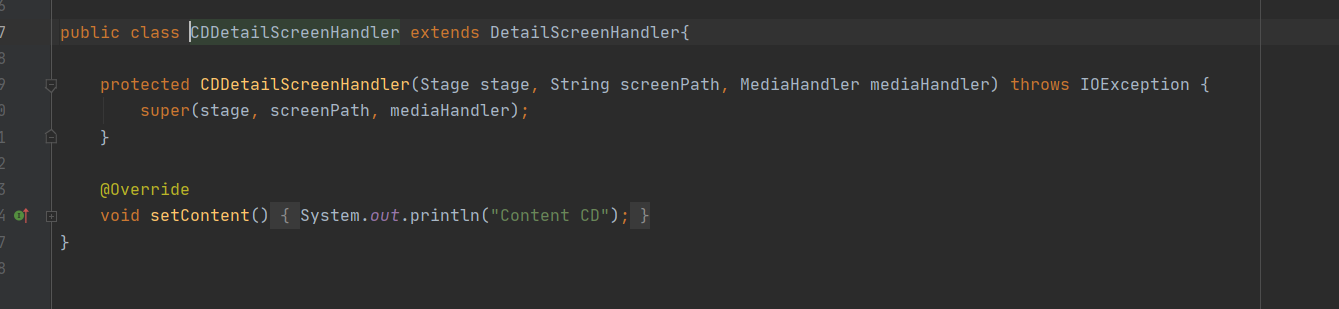
Public void notifyWatchDetail(){observerList.forEach(observer -> observer.watchDetail(this))}

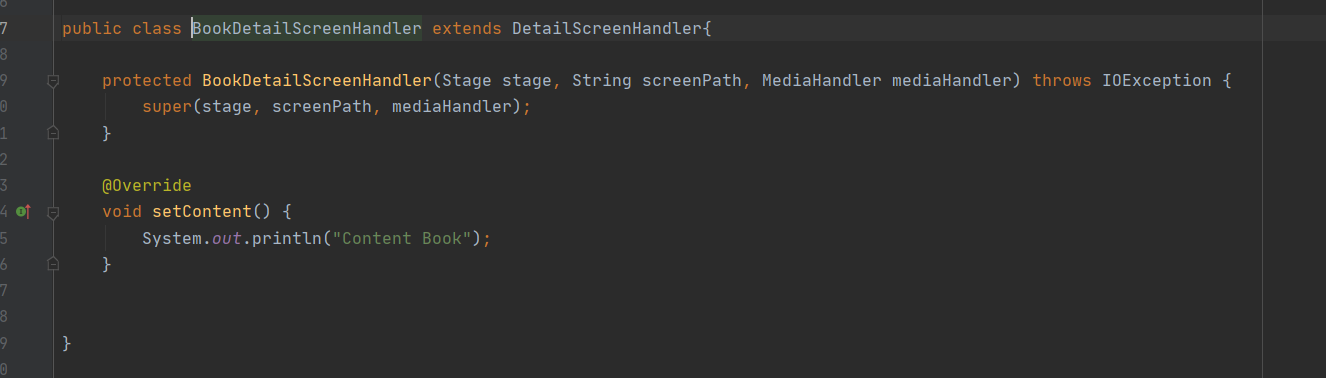
}

* Tạo ra Abstract Class DetailScreenHandler phụ trách việc hiển thị trang chi tiết, theo đó có BookDetailScreenHandler, DVDDetailScreenHandler, CDDetailScreenHanlder kế thừa DetailScreenHandler - Ở đây áp dụng Template Method. Đồng thời DetailScreenHandler là một Observable nên kế thừa Observable đồng thời chưa các Observer và HomeScreenHandler lúc subscribe DetailScreenHandler nhận sự kiện “Add To Card”

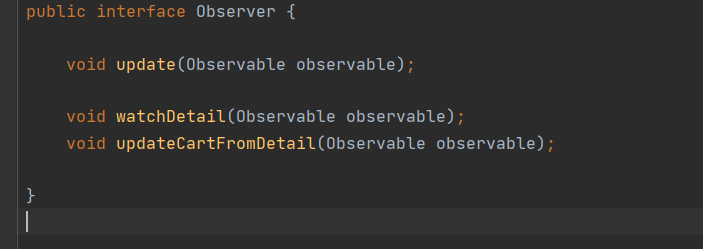




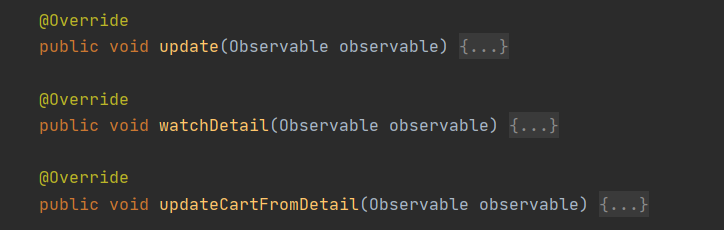




* Observer ngoài method update thêm các method watchDetail(), updateCardFromDetail()



Tại màn HomeScreenHandler override lại các phương thức

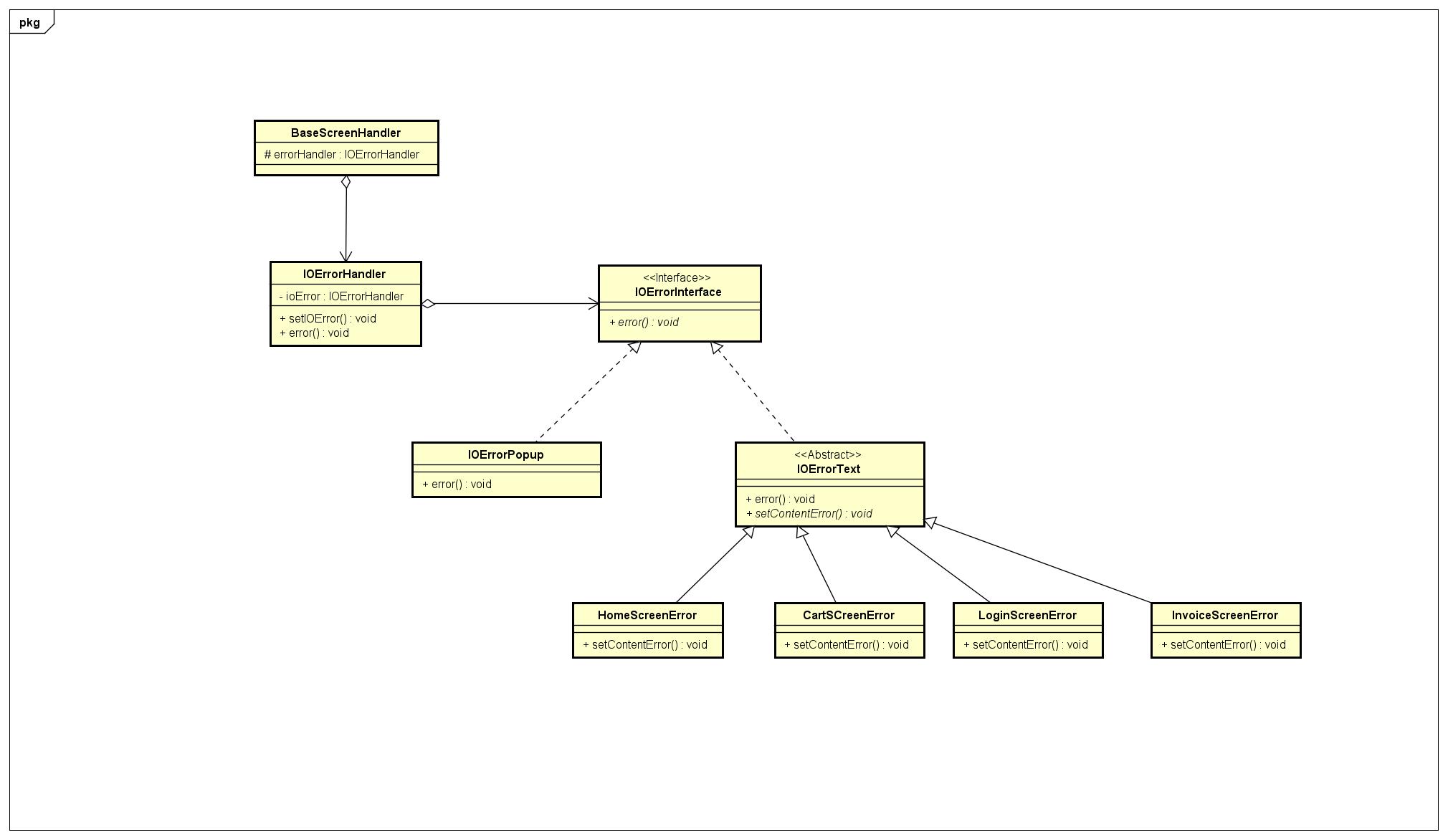


## Vấn đề 3 - Thay đổi yêu cầu khi load giao diện.

2.8.1. Giải pháp

Khi có thay đổi yêu cầu hiển thị thông báo lỗi từ hiển thị popup sang hiển thị cụ thể lỗi ở phía trên cùng của trang, sẽ phải sửa lại code hiện tại. Vì vậy giải pháp ở đây, ta sử dụng Strategy Pattern cho các loại hiển thị lỗi với yêu cầu hiện tại, hoặc tương lai nếu có thêm các loại hiển thị lỗi khác.

2.8.2. Biểu đồ

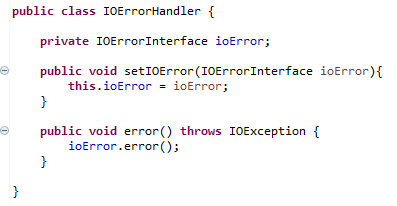


2.8.3. Mô tả mã nguồn

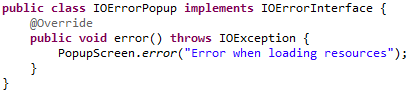
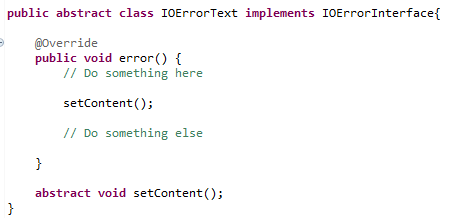
* Tạo Interface IOErrorInterface



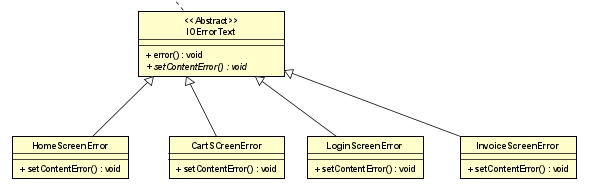
* Tạo class IOErrorHandler:



* Tạo 2 class IOErrorPopup và IOErrorText implements Interface IOErrorInterface cho việc hiển thị lỗi kiểu popup hay kiểu Text



* Tạo các class ứng với lỗi ở từng màn hình kế thừa class IOErrorText

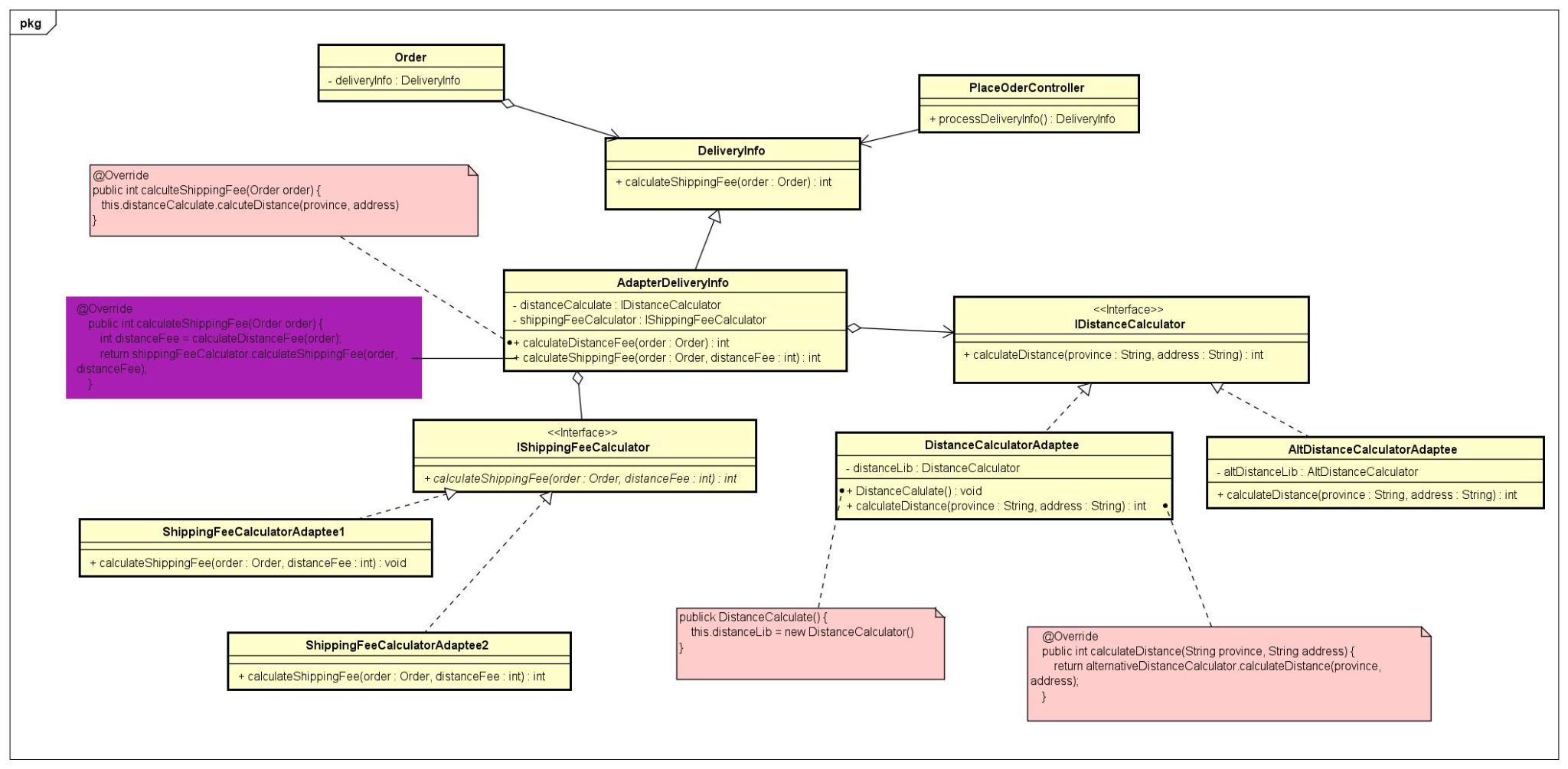


## Vấn đề 4 - Thay đổi khoảng cách sử dụng giao diện mới và công thức tính khoảng cách mới.

2.9.1. Giải pháp

Khi có thay đổi cách tính khoảng cách sử dụng thư viện mới hoặc thay đổi công thức tính phí vận chuyển,cần phải thay đổi vào code hiện tại ở method calculateShippingFee trong class DeliveryInfo. Điều này dẫn đến vi phạm nguyên tắc OCP . Các thư viện mới có thể có giao diện khác với giao diện của thư viện tính khoảng cách hiện tại nên giải pháp ở đây, ta sẽ sử dụng Adapter Pattern.

2.9.2. Biểu đồ



2.9.3. Mô tả mã nguồn

* Sửa lại class DeliveryInfo thành abstract và method calculateShippingFee thành abstract
* Tạo Interface IDistanceCalculator:



* Tạo Interface IShippingFeeCalculator:



* Tạo class AdapterDeliveryInfo kế thừa từ DeliveryInfo có các phương thức như sau:

public class AdapterDeliveryInfo extends DeliveryInfo{

private IDistanceCalculator distanceCalculator;

private IShippingFeeCalculator shippingFeeCalculator;

public AdapterDeliveryInfo(String name, String phone, String province, String address, String shippingInstructions, IDistanceCalculator distanceCalculator) {

super(name, phone, province, address, shippingInstructions);

this.distanceCalculator = distanceCalculator;

}

public void setDistanceCalculator(IDistanceCalculator distanceCalculator) {

this.distanceCalculator = distanceCalculator;

}

public int calculateDistanceFee(Order order) {

return distanceCalculator.calculateDistance(province, address);

}

public void setShippingFeeCalculator(IShippingFeeCalculator shippingFeeCalculator) {

this.shippingFeeCalculator = shippingFeeCalculator;

}

@Override

public int calculateShippingFee(Order order) {

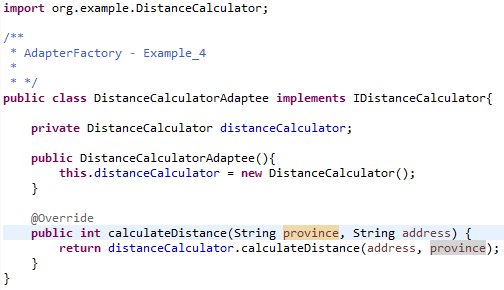
int distanceFee = calculateDistanceFee(order);

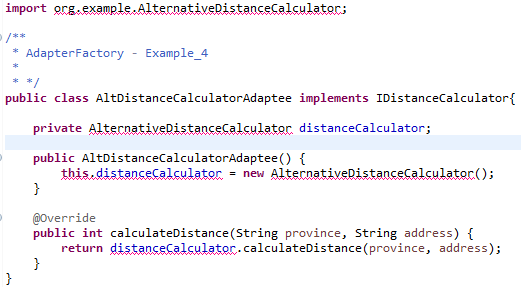
return shippingFeeCalculator.calculateShippingFee(order, distanceFee);

}

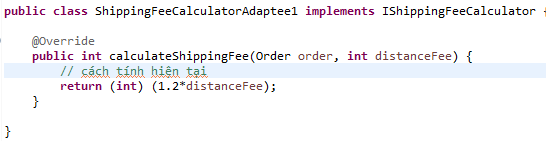
}

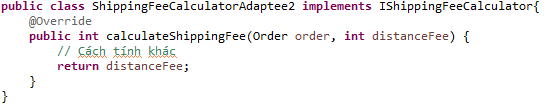
* Tạo 2 lớp DistanceCalculatorAdaptee và AltDistanceCalculatorAdaptee như sau để thực hiện việc sử dụng các thư viện tính khoảng cách khác nhau





* Tạo 2 class ShippingFeeCalculatorAdaptee1, ShippingFeeCalculatorAdaptee2 để thực hiện việc cách tính khoảng cách khác nhau





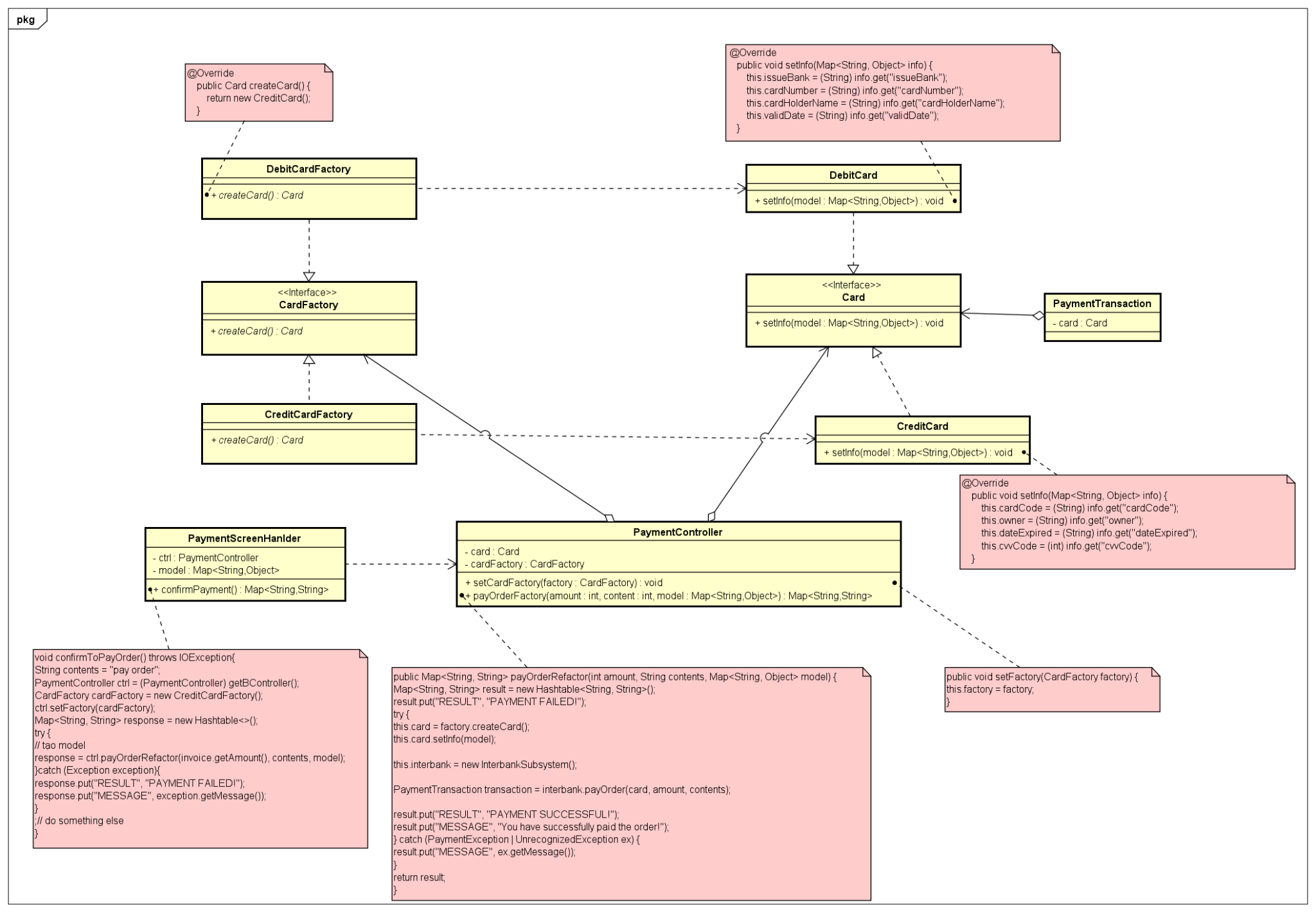
## Vấn đề 5 - Thêm một loại hình thanh toán mới.

2.10.1. Giải pháp

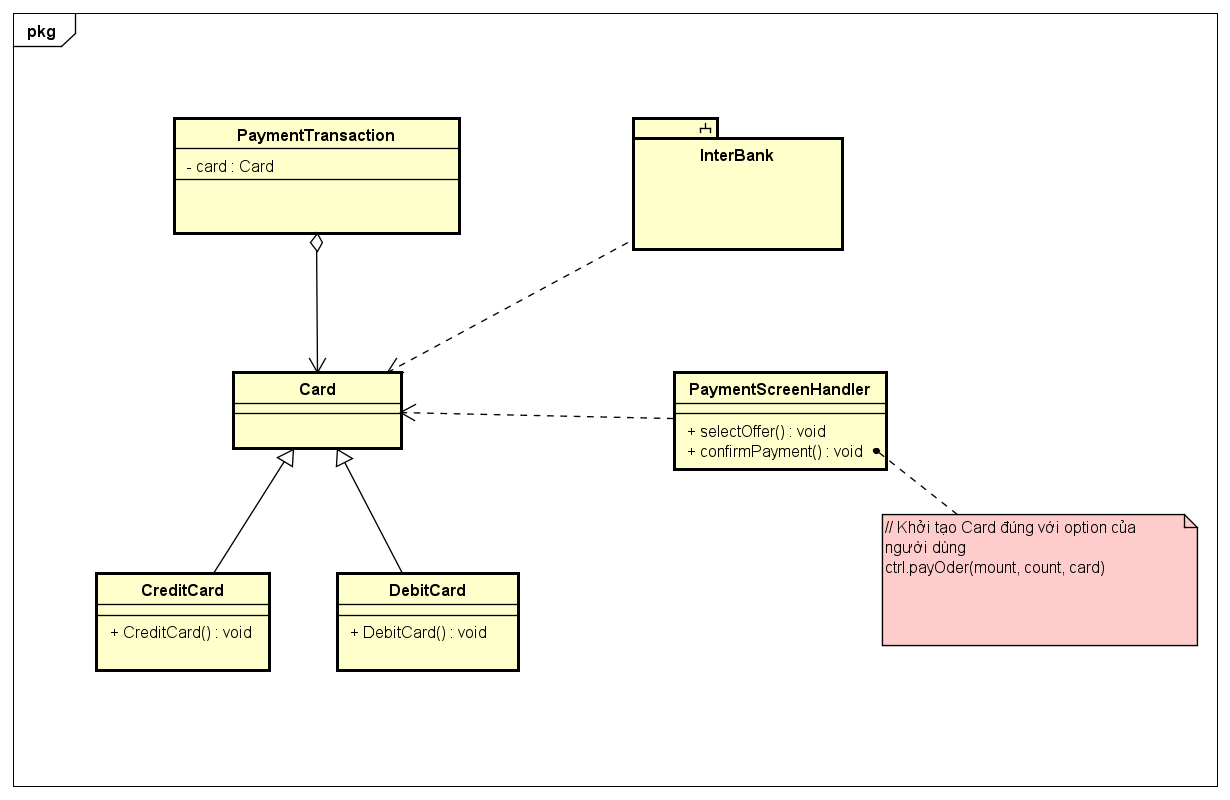
Hiện tại chỉ có một phương thức thanh toán bằng CreditCard , Lớp Payment Transaction chứa lớp CreditCard cụ thể. Khi thêm một phương thức mới DebitCard thì PaymentTransaction không được dùng nữa hoặc là phải sửa lại code, điều đó dẫn đến các lớp PaymentController, InterBank phụ thuộc vào PaymentTransaction sẽ phải thay đổi lại code . Tất cả điều đó vi phạm OCP và DIP. Giải pháp đưa ra.

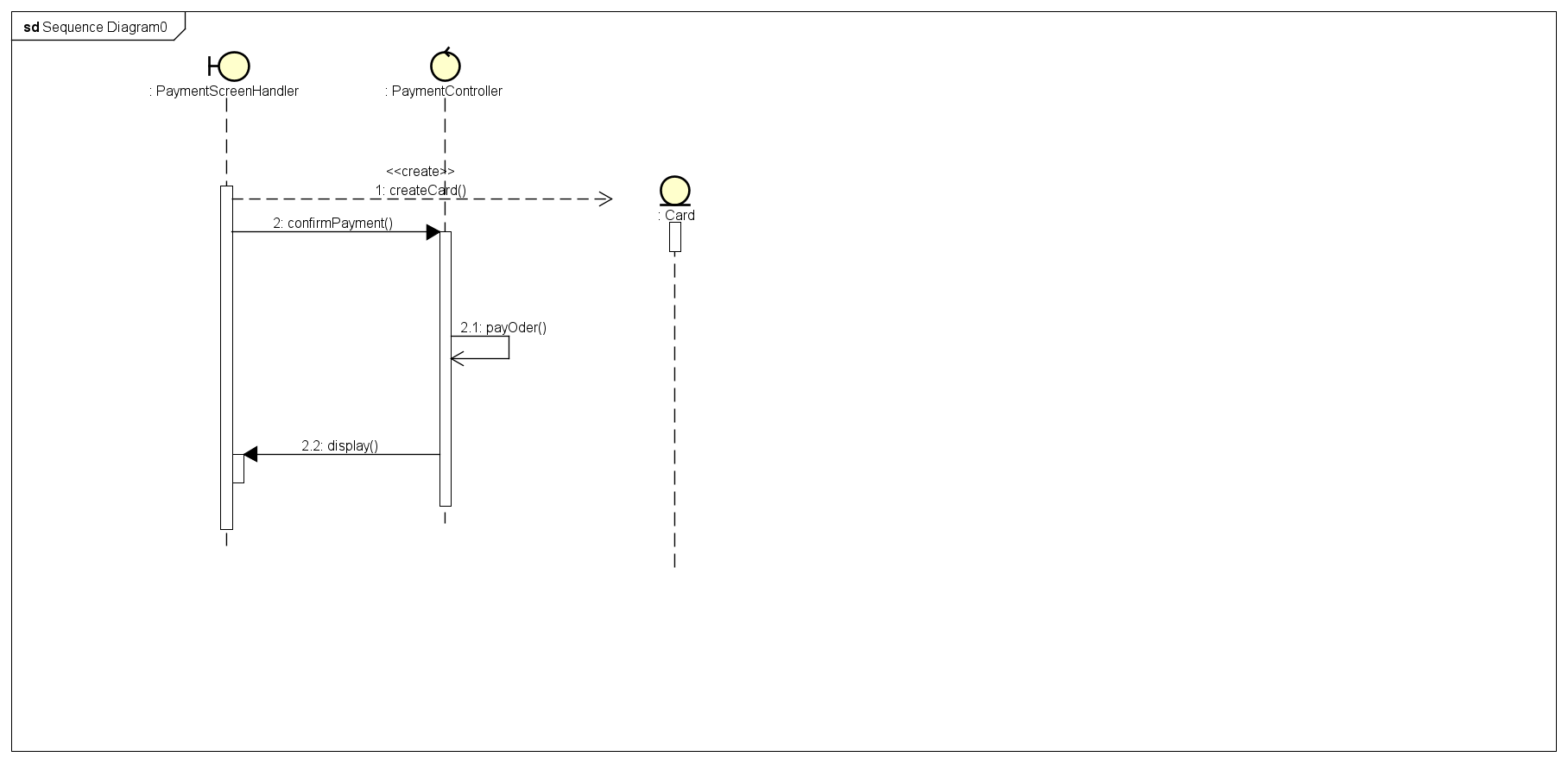
* Trừu tượng hóa các lớp CreditCard và DometicCard thành Interface Card
* PaymentController chứa Card
* Khi ở phía client người dùng thực hiện thao tác chọn card offer. Ta cần phải khởi tạo ra đúng loại card người dùng đang chọn. Ở đây ta sẽ áp dụng Factory Method hoặc Stratey Pattern vào trong trường hợp mở rộng này.

2.10.2. Biểu đồ

- Class Diagram Abstract Factory 

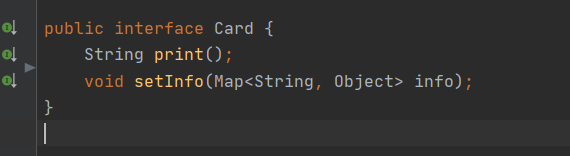
- Class Digram Strategy Pattern

****

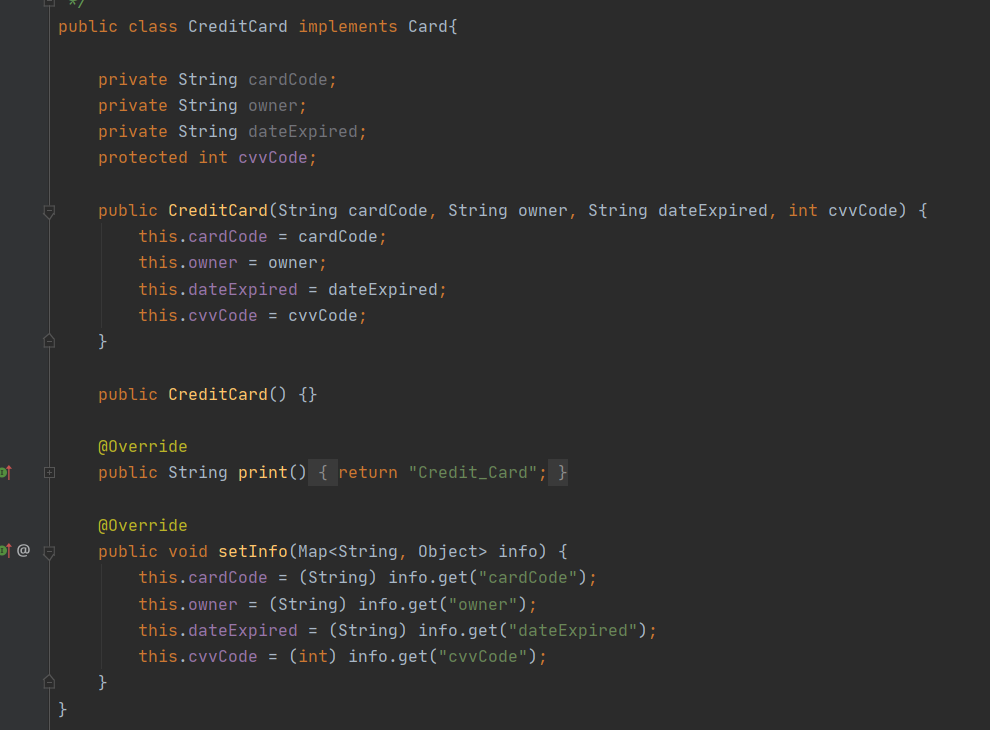


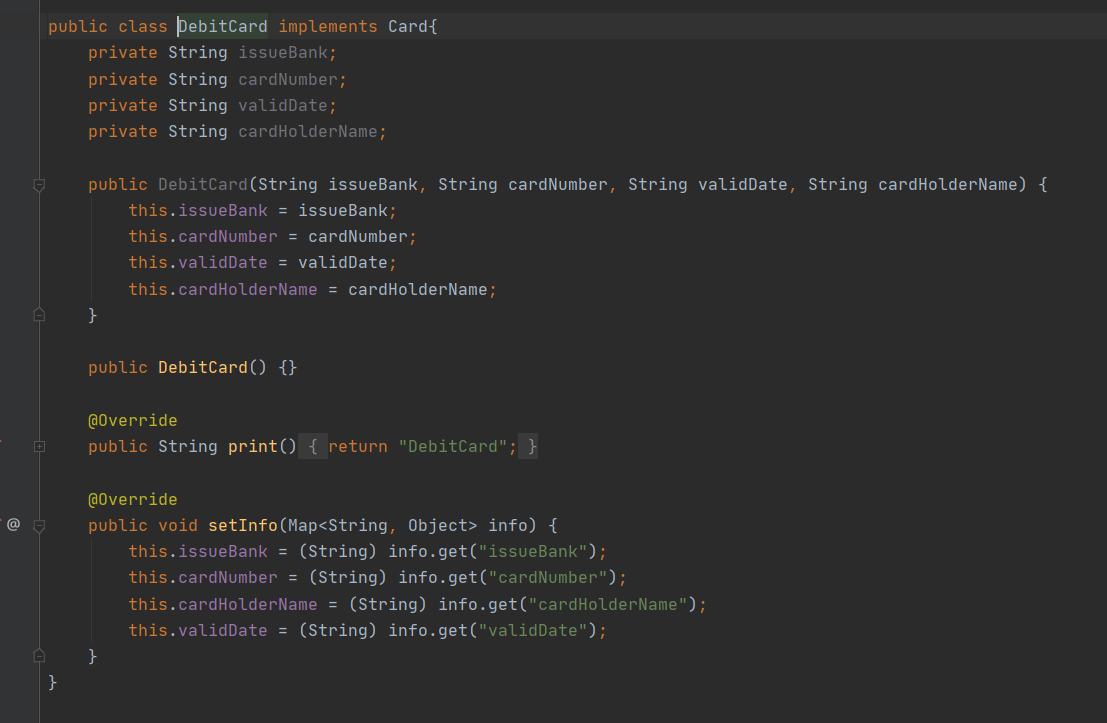
2.10.3. Mô tả mã nguồn cho Strategy

* Tạo ra Interface Card



* DebitCard và CreditCard implements Card





Class PaymentController {

Map<String, String> payOrderRefactor(int amount, String contents, Card card) {

// Tiền xử lý

PaymentTransaction transaction = interbank.payOrder(card, amout, contents)

// Logic else….

}

}

* Class PaymentScreen khác hàng chọn offer tạo ra Card phù hợp truyền vào PaymentController

Class PaymentScreenHandler {

confirmPayment() {

// lấy ra option khách chọn

// Khởi tạo instance Card

ctrl.payOder(amount, contents, card)

}

}

## Vấn đề 6 - Cập nhật lại chức năng hủy đơn hàng.

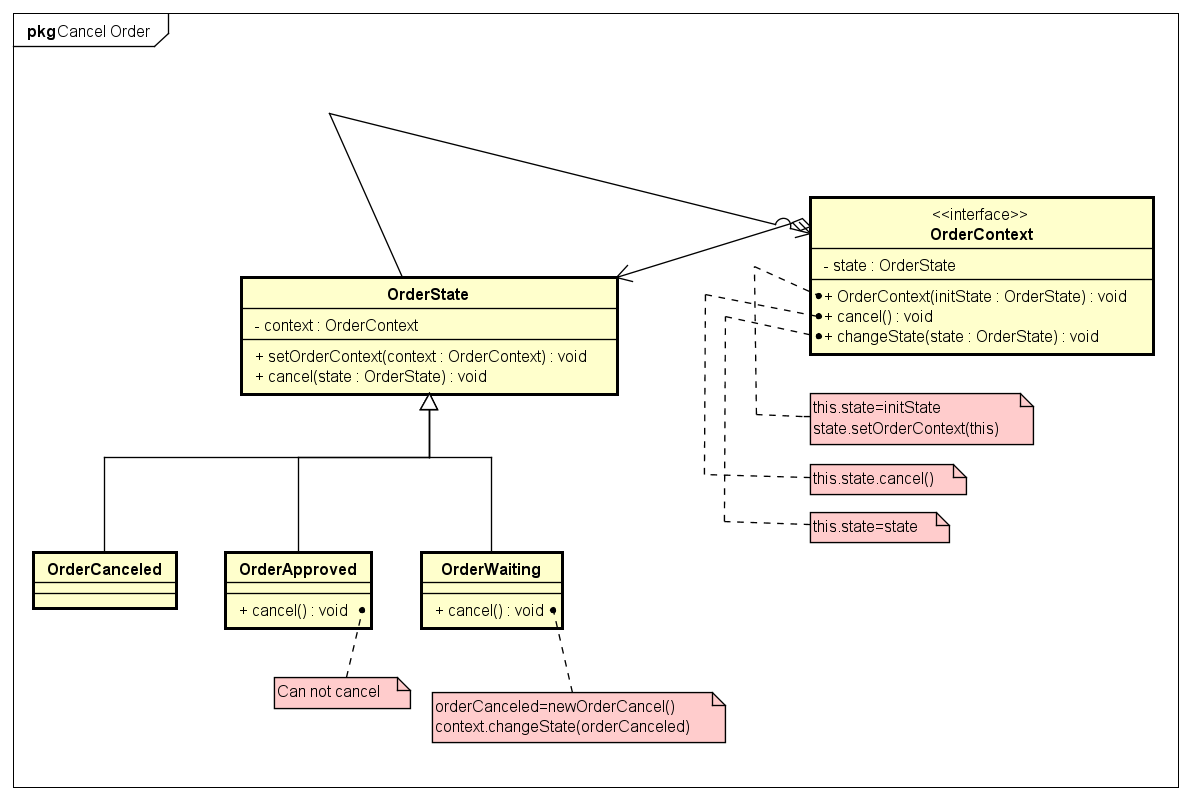
2.11.1 Giải pháp

* Với mã nguồn cũ, sau khi đặt hàng và thanh toán, khách hàng có thể huỷ đơn hàng bất cứ lúc nào và được hoàn tiền. Do đó vấn đề này đã vi phạm OCP.
* Sau khi refactor lại code, tính năng huỷ đơn hàng chỉ thực hiện được khi admin chưa duyệt đơn hàng
* Giải pháp đưa ra: khi khách hàng tiến hàng hủy đơn hàng, hệ thống sẽ check xem đơn hàng đang ở trạng thái nào và phản hồi thông báo tới user.

→ Sử dụng State Pattern:

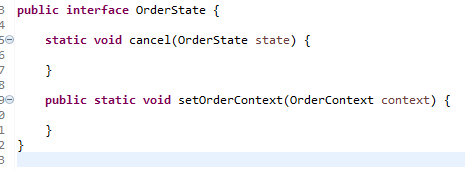
* Tạo class OrderContext để lưu trạng thái hiện tại của đơn và có chứa method cancelOrder
* Tạo Interface OrderState là trạng thái của đơn hàng, có method cancel để huỷ đơn hàng.
* Ngoài ra các class khác OrderWaiting, OrderApproved, OrderCanceled implements OrderState
* Lúc đó trạng thái đơn hàng sẽ chuyển sang OrderCanceled → thay đổi state ở OrderContext sang OrderCanceled.

2.12.2 Biểu đồ

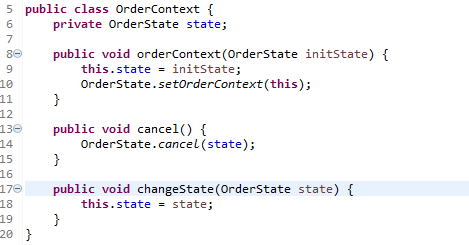


2.12.3 Mô tả mã nguồn

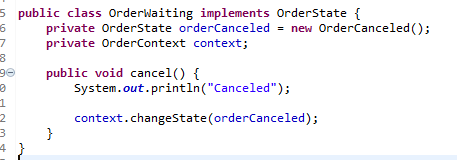
* Tạo Interface OrderState có 2 method



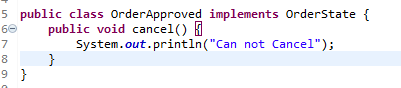
* Tạo class OrderContext để lưu trữ state đơn hàng:

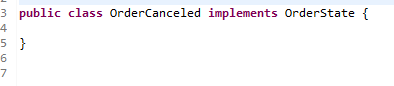


* Lần lượt tạo 3 class OrderWaiting, OrderApproved, OrderCanceled implements OrderState.



Chỉ có đơn hàng đang waiting mới có thể cancel





# 3. Tổng kết

## Kết quả tổng quan

* Sau lần tái cấu trúc mã nguồn đã làm code trở lên lỏng lẻo hơn giữa các class dễ ràng mở rộng thêm chức năng mới.
* Đã tiến tới Low Coupling , High Cohesion, cũng như thiết kế lại các lớp tuân thủ các nguyên tắc SOLID.
* Các vấn đề đặt ra cũng đã giải quyết nhờ các nguyên lý Design Pattern

## Các vấn đề tồn đọng

* Đến thời điểm hiện tại code đã đảm bảo được hết những vấn đề cần giải quyết trong tương lai.