ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

Version 1.0

DP.20202.02

Danh sách thành viên:

Vũ Quang Đại - 20172993

Lê Minh Đức - 20173043

Nguyễn Đình Đức - 20173041

Nguyễn Mạnh Đức - 20173039

*Hà Nội, 06, 2021*

# Phân công công việc

|  |  |  |
| --- | --- | --- |
| **Họ tên** | **MSSV** | **Công việc** |
| Vũ Quang Đại | 20172993 | - Phân chia công việc, giám sát tiến độ của các thành viên  - Xử lý các yêu cầu thêm số 4, 6, 7 của hệ thống  - Cải thiện mã nguồn bằng observer pattern, state pattern, adapter pattern.  - Làm, tổng hợp báo cáo, slide, video, mã nguồn |
| Lê Minh Đức | 20173043 | - Xử lý yêu cầu thêm số 5 của hệ thống  - Cải thiện mã nguồn bằng singleton pattern, factory pattern  - Đánh giá mức độ clean method, clean class  - Làm báo cáo, slide, video |
| Nguyễn Đình Đức | 20173041 | - Đánh giá mức độ coupling  - Đánh giá mã nguồn theo SOLID  - Làm báo cáo, slide, video |
| Nguyễn Mạnh Đức | 20173039 | - Đánh giá mức độ cohesion  - Đánh giá mức độ clear name  - Làm báo cáo, slide, video |

Mục lục

Phân công công việc 1

Mục lục 2

1 Tổng quan 5

1.1 Mục tiêu 5

1.2 Phạm vi 5

1.2.1 Mô tả phần mềm: 5

1.2.2 Các chức năng chính của phần mềm 5

1.2.3 Cấu trúc mã nguồn 6

1.2.4 Các yêu cầu thêm của hệ thống 7

1.2.5 Các hoạt động thực thi trên mã nguồn 7

1.2.6 Kết quả dự kiến 8

1.3 Danh sách thuật ngữ 8

1.4 Danh sách tham khảo 8

2 Đánh giá thiết kế cũ 9

2.1 Nhận xét chung 9

2.2 Đánh giá các mức độ coupling và cohesion 9

2.2.1 Coupling 9

2.2.2 Cohesion 10

2.3 Đánh giá việc tuân theo SOLID 11

2.3.1 SRP 11

2.3.2 OCP 11

2.3.3 DIP 12

2.3.4 LSP và ISP 12

2.4 Các vấn đề về Clean Code 12

2.4.1 Clear Name 12

2.4.2 Clean Function/Method 12

2.4.3 Clean Class 14

2.5 Các vấn đề khác 15

3 Đề xuất cải tiến 16

3.1 Vấn đề nhiều class tham chiếu đến các thuộc tính static của class SessionInformation. 16

3.2 Vấn đề cập nhật yêu cầu: Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card) 17

3.3 Vấn đề clean class các large class: AuthenticationController, PaymentController, PLaceOrderController 18

3.4 Vấn đề clean class giữa hai class CartItem và MediaHandler của gói views.screen.cart, giữa hai class OrderItem và MediaInvoiceScreenHandler của gói views.screen.invoice 19

3.5 Vấn đề clean method tại CartMediaHandler và giải pháp 20

3.6 Vấn đề clean class tại HomeScreenHandler và giải pháp 21

3.7 Vấn đề coupling tại CartScreenHandler - MediaHandler và giải pháp dùng observer pattern 22

3.8 Vấn đề cập nhật yêu cầu: Thay đổi công thức tính vận chuyển 25

3.9 Vấn đề cập nhật yêu cầu: Cập nhật chức năng hủy đơn hàng 26

4 Tổng kết 28

4.1 Kết quả tổng quan 28

4.2 Các vấn đề tồn đọng 28

**Danh sách các minh họa**

[Hình 1: Biểu đồ use cases các chức năng đã có trong hệ thống 6](#_Toc74080988)

[Hình 2: Coupling giữa các cặp class 14](#_Toc74080989)

[Hình 3: Cấu trúc SessionInformation ban đầu 16](#_Toc74080990)

[Hình 4: Cấu trúc SessionInformation sau khi áp dụng mẫu singleton 16](#_Toc74080991)

[Hình 5: Các phương thức trong một số controller 18](#_Toc74080992)

[Hình 6: Các phương thức tại mỗi lớp sau khi đã tái cấu trúc code 19](#_Toc74080993)

[Hình 7: Coupling giữa hai lớp MediaInvoiceScreenHandler-OrderItem, Mediahandler-CartItem 19](#_Toc74080994)

[Hình 8: Bổ sung interface giữa hai lớp MediaHandler và CartItem 20](#_Toc74080995)

[Hình 9: Áp dụng observer pattern vào tái cấu trúc các lớp MediaHandler và CartScreenHandler 23](#_Toc74080996)

[Hình 10: Biểu đồ trình tự, thể hiện tương tác giữa hai lớp CartScreenHandler và MediaHandler. 24](#_Toc74080997)

[Hình 11: Lớp adapter AltDeliveryInfo 25](#_Toc74080998)

[Hình 12: Biểu đồ các lớp phục vụ chức năng theo dõi đơn hàng 27](#_Toc74080999)

**Danh sách các bảng**

[Bảng 1: Đánh giá mức độ coupling 10](#_Toc74081000)

[Bảng 2: Đánh giá mức độ cohesion 10](#_Toc74081001)

[Bảng 3: Các phần vi phạm nguyên lý SRP 11](#_Toc74081002)

[Bảng 4: Các phần vi phạm nguyên lý OCP 11](#_Toc74081003)

[Bảng 5: Các phần vi phạm nguyên lý DIP 12](#_Toc74081004)

[Bảng 6: Các phần vi phạm nguyên lý LSP và ISP 12](#_Toc74081005)

# Tổng quan

## Mục tiêu

Mục đích của báo cáo mẫu thiết kế phần mềm là để đánh giá thiết kế cũ một cách tổng quan và chi tiết từng phần, bao gồm (i) đánh giá về mức độ coupling, cohesion, (ii) đánh giá việc tuân theo nguyên lý SOLID, (iii) đánh giá mức độ clean code. Đồng thời, trong bản báo cáo cũng đưa ra một số đề xuất cải tiến cho các vấn đề gặp phải liên quan đến các mục đánh giá ở trên.

Đối tượng hướng tới của báo cáo này là kỹ sư thiết kế, nhân viên phát triển và nhân viên bảo trì phần mềm, thông báo về những vấn đề gặp và đưa ra giải pháp sửa chữa.

## Phạm vi

### Mô tả phần mềm:

Hệ thống thương mại điện tử AIMS Project, cho phép mua bán sản phẩm phương tiện truyền thông vật lý. Với sản phẩm vật lý (physical good), cần có mã vạch barcode, mô tả sản phẩm (descripUon, ví dụ: hàng mới, hàng cũ, màu chủ đạo, điều kiện hoàn trả), số lượng sản phẩm (quanUty), ngày nhập kho, kích thước và khối lượng (product dimensions). Sản phẩm vật lý có thể là sách quyển, đĩa CD, đĩa than LP, hay đĩa DVD.

### Các chức năng chính của phần mềm

**Chức năng Đăng nhập**: Khi đăng nhập, hệ thống sẽ tự động lưu và lấy thông tin giao hàng mặc định, thông tin thẻ và sản phẩm hiện có trong giỏ hàng vào trong tài khoản khách hàng.

**Chức năng Danh sách sản phẩm**: Khi khởi động, hệ thống sẽ hiện ra danh sách của 20 sản phẩm bất kỳ. Mỗi sản phẩm trong danh sách bao gồm ảnh, tên sản phẩm, giá cả và số lượng sản phẩm hiện có trong kho.

**Chức năng Tìm kiếm**: cho phép người dùng tìm kiếm theo loại mặt hàng, bao gồm sách (Book), CD và DVD.

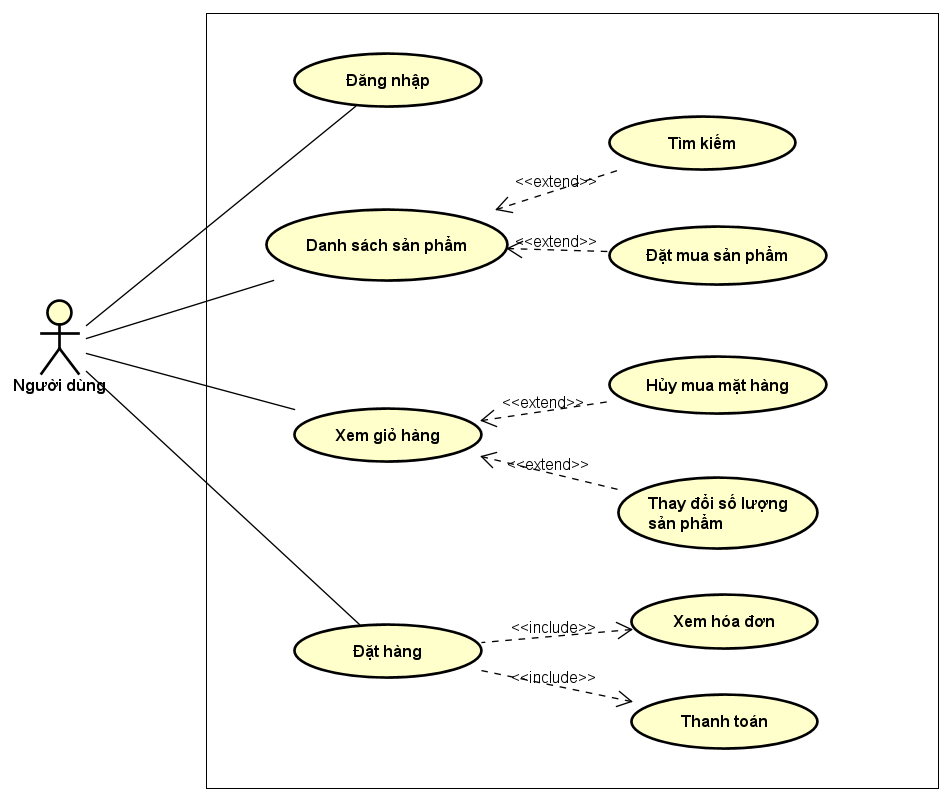
**Chức năng Đặt mua sản phẩm**: Người dùng chọn số lượng và nhấn đặt mua để thêm mặt hàng vào giỏ hàng.

**Chức năng Xem giỏ hàng**: Liệt kê danh sách các mặt hàng, số lượng đã đặt mua, chi phí cho mỗi mặt hàng. Hiển thị tổng giá tiền của toàn bộ giỏ hàng.

**Chức năng Hủy mua mặt hàng**: Người dùng hủy mặt hàng sẽ loại bỏ mặt hàng đó khỏi giỏ hàng.

**Chức năng Thay đổi số lượng sản phẩm**: Người dùng có thể thêm hoặc bớt số lượng sản phẩm định mua trong giới hạn nhất định, tối thiểu là 1 và tối đa không vượt quá số hàng trong kho.

**Chức năng Đặt hàng**: Người dùng điền thông tin về tên, điện thoại, địa chỉ và các hướng dẫn thêm khi giao hàng. Hệ thống hiển thị hóa đơn và cho phép người dùng lựa chọn phương thức thanh toán phù hợp.



Hình 1: Biểu đồ use cases các chức năng đã có trong hệ thống

### Cấu trúc mã nguồn

Mã nguồn được chia thành 8 package lớn.

Package default: gồm App, trong đó chứa hàm main(), là điểm bắt đầu của chương trình.

Package common: khai báo các loại exceptions sử dụng trong chương trình, đồng thời chứa interface Observer và Observable dùng trong observer pattern.

Package controller: bao gồm các controller tương ứng với các chức năng lớn của hệ thống.

Package dao: chứa các data access object dùng để giao tiếp với cơ sở dữ liệu.

Package entity: chứa các lớp đại diện cho các thực thế có trong hệ thống.

Package subsystem: hệ thống con dùng để xử lý việc thanh toán bằng thẻ.

Package utils: gồm các lớp cung cấp tiện ích như validate dữ liệu, cung cấp hàm hash, chuyển dữ liệu từ dạng json sang cấu trúc map.

Package views: chứa các lớp xử lý giao diện.

### Các yêu cầu thêm của hệ thống

Hệ thống có 7 yêu cầu thêm:

**Yêu cầu 1**: Thêm mặt hàng Media mới: AudioBook. AudioBook bao gồm các trường thông tin về author, format, language, accent, lengthInMinutes.

**Yêu cầu 2**: Thêm màn hình xem chi tiết sản phẩm

Tại màn hình chính (home screen), người dùng có thể xem danh sách các sản phẩm. Bên cạnh nút “Add to cart” tại mỗi sản phẩm, chương trình cần bổ sung thêm nút “View details”. Nếu khách hàng quan tâm sản phẩm nào, có thể chọn để xem chi tiết. Tại màn hình xem chi tiết, tất cả thông tin của sản phẩm được hiển thịđầy đủđể khách hàng xem xét, kèm theo lựa chọn số lượng và nút “Add to cart”.

**Yêu cầu 3**: Thay đổi yêu cầu khi load giao diện lỗi. Thay vì hiển thị thông báo lỗi trên màn hình popup, chương trình hiển thị trực tiếp lỗi màu đỏ trên phía trên cùng của các trang. Thay vì thông báo “Error when loading resources”, cần bổ sung thêm lỗi khi thao tác với resource nào, và cụ thể lỗi (ví dụ: không tìm thấy,...).

**Yêu cầu 4**: Thay đổi cách tính khoảng cách, sử dụng thư viện mới. Thay vì sử dụng thư viện cũ DistanceCalculator, ta sử dụng một thư viện mới là alt-distance-api.jar có interface gần giống với distance-api.jar.

**Yêu cầu 5**: Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card). Loại thẻ này bao gồm các trường thông tin về type, issuring bank, card number, valid-from date. cardholder’s name. Với thẻ nội địa, chương trình vẫn kết nối với API của Interbank, chỉ thay đổi thông tin của phương thức thanh toán.

**Yêu cầu 6**: Thay đổi công thức tính phí vận chuyển. Phí vận chuyển sẽđược tính dựa theo khối lượng thực tế, khoảng cách, và độcồng kềnh của kiện hàng.Ví dụ: Kích thước của kiện hàng: chiều dài, chiều rộng, chiều cao. Độ cồng kềnh của kiện hàng có thể lấy tượng trưng: Độ cồng kềnh (kg) = Chiều dài (cm) x Chiều rộng (cm) x Chiều cao (cm) / 6000. Khối lượng và độ cồng kềnh (đơn vị: kg) lấy hệ số là 10 (đ/kg); khoảng cách (đơn vị: km, mặc định trả về từ thư viện Distance-API) lấy hệ số là 1.

**Yêu cầu 7**: Cập nhật lại chức năng hủy đơn hàng. Chức năng hủy đơn hàng sẽ bị vô hiệu hóa sau khi đơn hàng đã được quản trị viên chấp nhận. Nếu đơn hàng vẫn đang ở trạng thái chờ phê duyệt, khách hàng vẫn có thể hủy đơn hàng như bình thường.

### Các hoạt động thực thi trên mã nguồn

Thực hiện việc đánh giá về mức độ coupling, cohesion, mức độ tuân theo nguyên lý SOLID của thiết kế trong code base, tìm kiếm những nơi xảy ra code smell, đánh giá về mức độ sạch của mã nguồn theo ba mức độ: clear name, clean method và clean class.

Đưa ra giải pháp cho các vấn đề coupling, cohesion, vi phạm SOLID. Làm sạch code bằng cách refactor.

Đưa ra giải pháp để việc thay đổi theo các yêu cầu thêm.

### Kết quả dự kiến

Mã nguồn hoạt động tốt, chương trình vận hành bình thường, không có lỗi phát sinh thêm.

Làm sạch code, giảm được coupling về mức data coupling và cohesion về mức functional cohesion trên tất cả các lớp trong code base.

Áp dụng triệt để nguyên lý SOLID vào mã nguồn.

Tái cấu trúc mã nguồn sao cho các yêu cầu thêm có thể thực hiện dễ dàng, không gây ra thêm coupling và cohesion.

## Danh sách thuật ngữ

SRP - Single responsibility principle

OCP - Open/closed principle

LSP - Liskov substitution principle

ISP - Interface segregation principle

DIP - Dependency inversion principle

## Danh sách tham khảo

# Đánh giá thiết kế cũ

## Nhận xét chung

Chất lượng mã nguồn hiện tại ở mức khá, vẫn còn xuất hiện một số nơi có mức độ coupling cao và cohesion thấp. Một số nơi còn vi phạm nguyên lý SOLID cần phải tái cấu trúc lại cho phù hợp.

Với các thay đổi trong tương lai: Một số yêu cầu (thay đổi cách tính chi phí vận chuyển, theo dõi đơn hàng) không cần thay đổi cấu trúc mã nguồn mà chỉ cần sử dụng một số cách và design pattern để thêm các tính năng phù hợp. Còn lại, hầu hết các yêu cầu mới đều cần phải thay đổi lại cấu trúc code base để có thể thêm tính năng phù hợp với yêu cầu mới.

## Đánh giá các mức độ coupling và cohesion

Coupling và cohesion là hai thước đo quan trọng để đánh giá về chất lượng của bản thiết kế cùng mã nguồn. Đồng thời, đây cũng là những dấu hiệu quan trọng để nhận biết các vấn đề của mã nguồn, từ đó giúp đưa ra được giải pháp để tái cấu trúc, cải thiện mã nguồn.

### Coupling

Sự phụ thuộc lẫn nhau giữa các thành phần trong CodeBase là không quá lớn. Đa số chỉ ở mức từ common đến stamp coupling.

Đánh giá chi tiết về mức độ coupling:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Lý do* |
| 1 | Common Coupling | Order | Phương thức Order(Cart cart) | Sử dụng đến SessionInformation.cartInstance của module khác |
| HomeScreenHandler | Phương thức show() và update() | Sử dụng đến SessionInformation.cartInstance của module khác |
| PlaceOrder Controller | Phương thức placeOrder() và createOrder() | Sử dụng đến SessionInformation.cartInstance của module khác |
| 2 | Control Coupling | HomeScreenHandler | Phương thức update(MediaHandler mediaHandler) | Sử dụng tham số truyền vào mediaHandler làm tham số điều khiển |
| PopupScreen | Phương thức popup(... Boolean undecorated) | Sử dụng tham số undecorated làm tham số điều khiển |
| 3 | Stamp Coupling | Cart | Phương thức checkMediaInCart(Media media) | Phương thức này nhận tham số truyền vào là toàn bộ media trong khi chỉ sử dụng media.getID() |
| HomeScreenHandler | Phương thức setupData(Object dto) | Truyền vào dto nhưng không dùng đến |
| PlaceOrder Controller | Phương thức validateDeliveryInfo( HashMap<String, String> info) | Truyền toàn bộ tham số info nhưng chỉ sử dụng 3 thuộc tính của info |

Bảng 1: Đánh giá mức độ coupling

### Cohesion

Đánh giá chung, mức độ liên kết chặt chẽ giữa các submodule chưa cao. Mức độ coincidental cohesion còn xuất hiện nhiều.

Chi tiết về các mức độ cohesion:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Lý do* |
| 1 | Coincidental Cohesion | Authentication Controller | Phương thức md5(String message) | Md5() không có liên hệ về mặt chức năng với AuthenticationController |
| Payment Controller | Phương thức getExpirationDate (String date) | Phần xác thực chuỗi không có liên hệ về mặt chức năng với payment. |
| PlaceOrder Controller | Phương thức createOrder, placeOrder | Các method createOrder, createOrder, placeOrder trong class được nhóm lại với nhau vì chúng được thực hiện theo trình tự |
| Interbank Payload Converter | Phương thức getToday() | Phương thức này không liên quan đến class |
| 2 | Logical cohesion |  |  |  |
| 3 | Temporal cohesion | PopupScreen | Phương thức success(String message) | Chứa cả method pop up + xử lí exception |
| 4 | Procedural Cohesion | PlaceOrder Controller | Phương thức createOrder, placeOrder | Các method createOrder, createOrder, placeOrder trong class được nhóm lại với nhau vì chúng được thực hiện theo trình tự |
| 5 | Communi- cational Cohesion | DeliveryInfo | Phương thức calculateShippingFee (Order order) | Phương thức này không liên quan đến class, chỉ sử dụng tham số distanceCalculator của class |

Bảng 2: Đánh giá mức độ cohesion

## Đánh giá việc tuân theo SOLID

Bản thiết kế còn vi phạm các nguyên lý SOLID ở một vài module:

* Một số class phải thực hiện thêm chức năng mà người thiết kế chưa biết đặt ở đâu (SRP)
* Một số chức năng được thiết kế chưa phù hợp cho việc mở rộng sau này (OCP)
* Một số module đang phụ thuộc trực tiếp vào module cấp thấp hơn (DIP)
* Một số class kế thừa nhưng không thật sự là con của class cha (LSP & ISP)

Trong trường hợp chỉnh sửa mã nguồn theo yêu cầu phát sinh, mã nguồn hiện tại có thể chưa tuân theo nguyên lý SOLID.

### SRP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Class có nhiệm vụ xác thực người dùng | Class thực hiện thêm nhiệm vụ mã hóa md5 không phù hợp đặt ở class này |
| 2 | PaymentController | Class có nhiệm vụ điều khiển thanh toán | Class thực hiện thêm nhiệm vụ xác thực và trả về ngày hết hạn đúng định dạng |
| 3 | PlaceOrderController | Class có nhiệm vụ điều khiển quá trình của hóa đơn | Class thực hiện thêm nhiệm vụ xử lý thông tin vận chuyển từ người dùng |
| 4 | Cart | Class làm khung cho một hóa đơn | Class thực hiện thêm nhiệm vụ kiểm tra sản phẩm có sẵn hay không |

Bảng 3: Các phần vi phạm nguyên lý SRP

### OCP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | PaymentController | Class thực hiện nhiệm vụ điều khiển thanh toán | Có 2 hình thức thanh toán là CreditCard và Interbank đều đặt ở phương thức payOrder, nếu sau này có thêm hình thức khác sẽ phải sửa lại class này |
| 2 | DeliveryInfo | Class mô tả các thông tin vận chuyển | Nếu sau này muốn thêm các thông tin khác sẽ phải sửa lại class này |

Bảng 4: Các phần vi phạm nguyên lý OCP

### DIP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Class có nhiệm vụ xác thực người dùng | Class này phụ thuộc trực tiếp vào class cấp thấp hơn là UserDAO |
| 2 | Order | Class xử lý các thông tin của đơn hàng | Phương thức setDeliveryInfo phụ thuộc trực tiếp vào hàm calculateShippingFee của class cấp thấp hơn |

Bảng 5: Các phần vi phạm nguyên lý DIP

### LSP và ISP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Class có nhiệm vụ xác thực người dùng | Class này kế thừa từ class BaseController nhưng không liên quan gì đến 2 phương thức của BaseController |
| 2 | HomeScreenHandler | Class điều khiển màn hình bắt đầu Home | Class này kế thừa từ class BaseScreenHandler nhưng không dùng đến phương thức setPreviousScreen và getPreviousScreen |

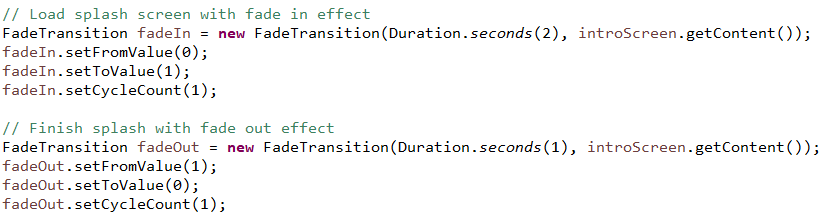
Bảng 6: Các phần vi phạm nguyên lý LSP và ISP

## Các vấn đề về Clean Code

### Clear Name

Nhìn chung, các tên hàm, phương thức trong mã nguồn đã được đặt rõ ràng, phù hợp.

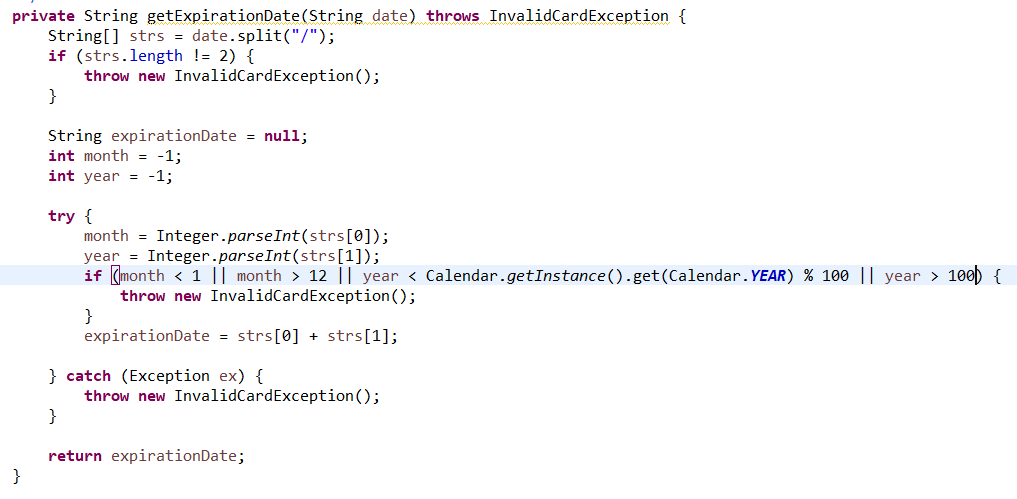
Các phần khởi tạo FadeTransition, tham số đầu tiên truyền vào thời gian cố định. Trong trường hợp cần sửa đổi sẽ gây ra khó khăn.



### Clean Function/Method

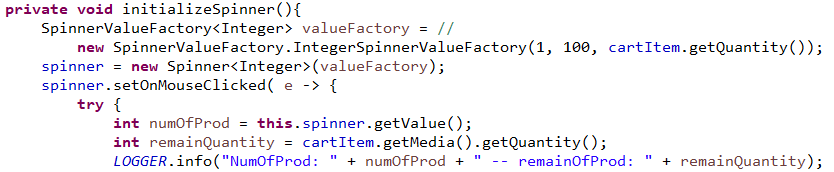
Trong mã nguồn ban đầu, các method đa phần đáp ứng clear method. Tuy nhiên, có một số chỗ chưa đáp ứng như sau:

Trong mã nguồn ban đầu, có method getExpirationDate() của class PaymentController còn chứa một biểu thức điều kiện phức tạp cần tạo thành một boolean function.

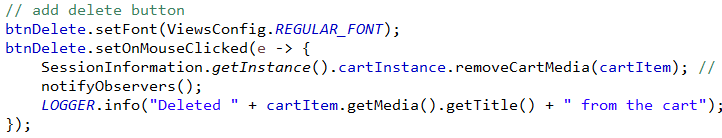


Thuộc tính của các class: Book, CD, DVD chưa có access modifier.

Tại class views.screen.cart.MediaHandler, method initializeSpinner(), có 2 nhiệm vụ, (i) tạo ra Spinner, (ii) xử lý sự kiện khi click vào spinner. Việc này làm giảm tính dễ đọc của code.



Class views.screen.cart.MediaHandler, method setMediaInfo() có 2 nhiệm vụ, vừa thiết lập các thông số về đồ họa, vừa xử lý sự kiện khi click vào nút Delete. Việc này làm giảm tính dễ đọc của code.

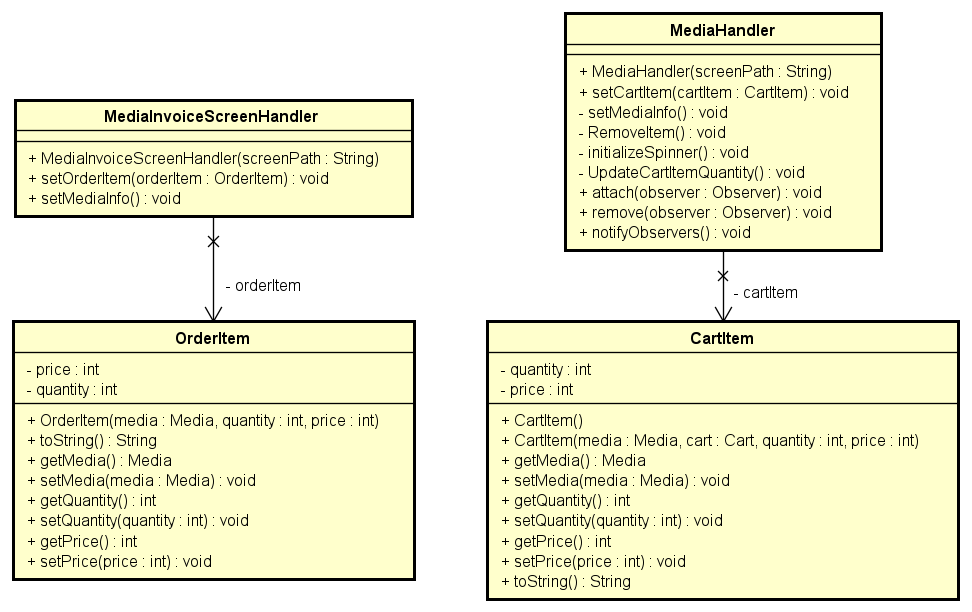
****

### Clean Class

**Vấn đề**: Trong mã nguồn ban đầu có một số Large class và các thành phần bên trong các class đó chưa thực hiện cùng một nhiệm vụ như: AuthenticationController, PaymentController, PLaceOrderController

**Vấn đề**: coupling giữa hai class CartItem và MediaHandler của gói views.screen.cart. Khi class CartItem thay đổi thì cũng phải thay đổi code hiển thị sản phẩm trong class MediaHandler

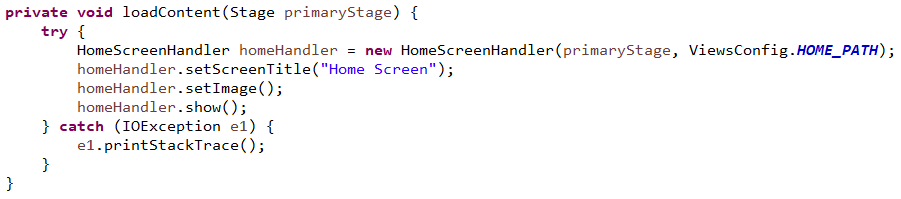
Tương tự vấn đề coupling giữa hai class OrderItem và MediaInvoiceScreenHandler của gói views.screen.invoice. Khi class OrderItem thay đổi thì cũng phải thay đổi code hiển thị sản phẩm trong class MediaInvoiceScreenHandler



Hình 2: Coupling giữa các cặp class

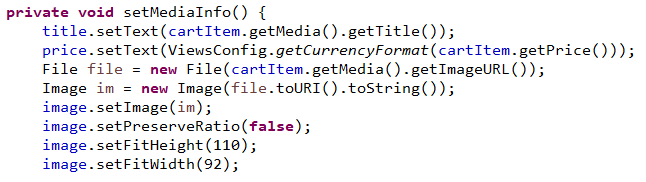
**Vấn đề:** HomeScreenHandler đưa ra các phương thức public setImage(), trong khi hoàn toàn có thể được gọi từ bên trong lớp đó, không có tác dụng gì khi gọi từ bên ngoài.

Trong ví dụ dưới đây, tại phương thức loadContent() của lớp App là lớp duy nhất có gọi tới hàm setImage() của HomeScreenHandler.



## Các vấn đề khác

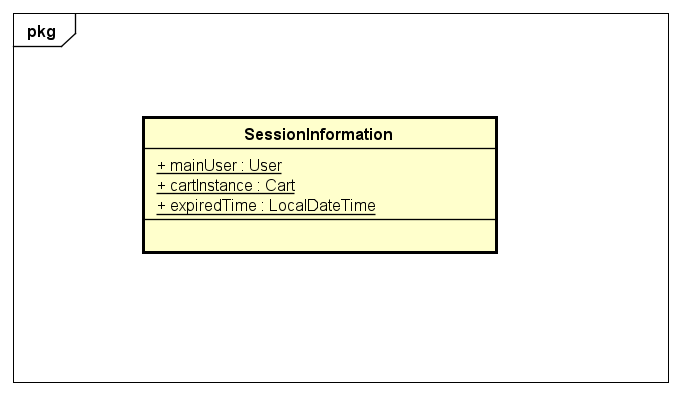
Class views.screen.cart.MediaHandler, method setMediaInfo() có chứa các đoạn mã thiết lập chiều dài và chiều rộng của ảnh với giá trị cố định. Điều này vi phạm nguyên lý OCP, vì các thông số này có khả năng cao sẽ thay đổi trong tương lai, nếu muốn thay đổi sẽ cần sửa lại code của class này. Đồng thời, các thông số này có thể thay đổi trong lúc thực thi chương trình, trong trường hợp ứng dụng muốn responsive cho nhiều loại kích cỡ màn hình.



# Đề xuất cải tiến

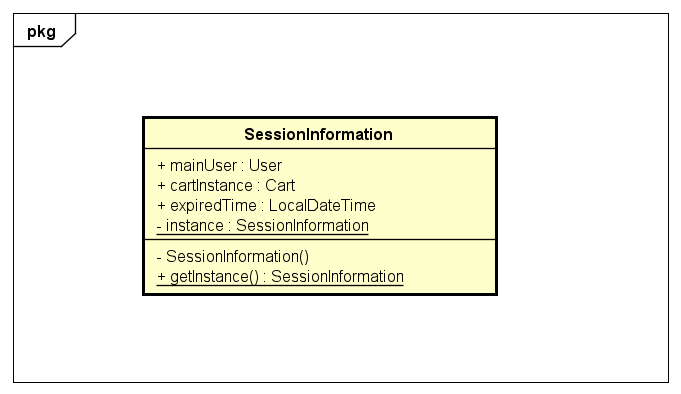
## Vấn đề nhiều class tham chiếu đến các thuộc tính static của class SessionInformation.

**Vấn đề:** Class SessionInformation sử dụng các thuộc tính static như: mainUser, cartInstance, expiredTime.



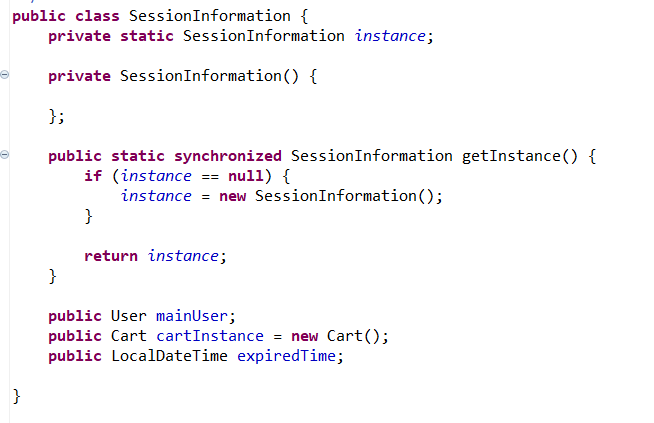
Hình 3: Cấu trúc SessionInformation ban đầu

**Giải pháp:** Sử dụng mẫu thiết kế Singleton



Hình 4: Cấu trúc SessionInformation sau khi áp dụng mẫu singleton

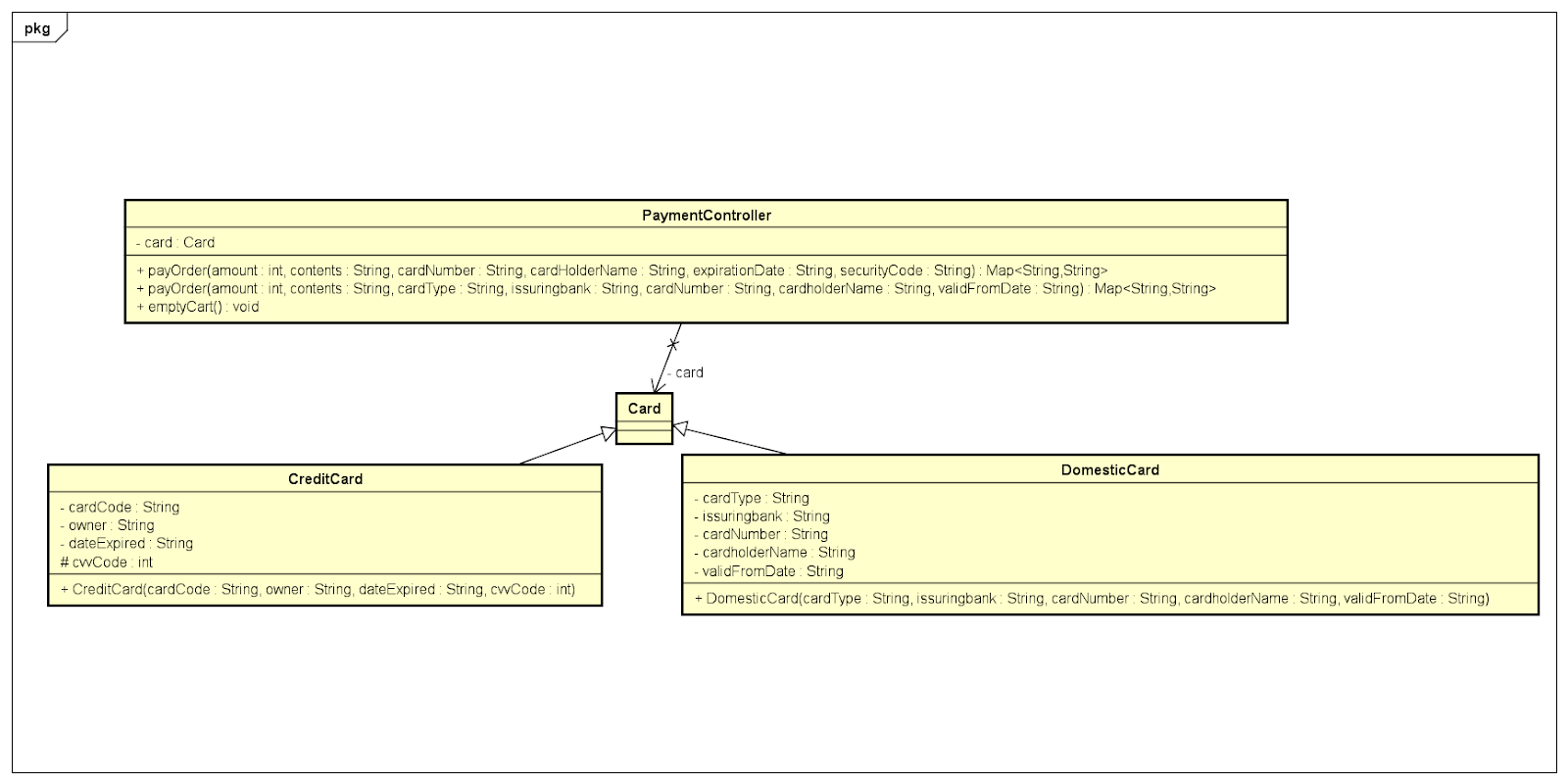
Mã nguồn của lớp SessionInformation theo singleton pattern:



## Vấn đề cập nhật yêu cầu: Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)

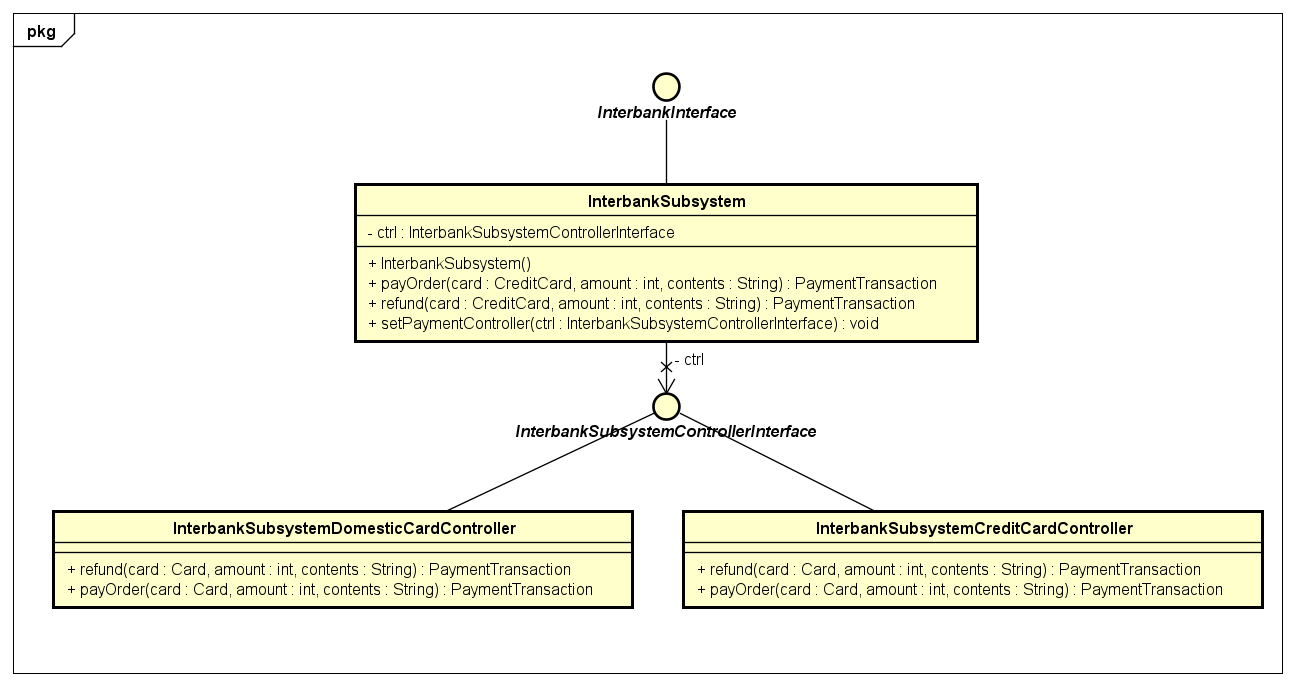
**Giải pháp**: Sử dụng mẫu thiết kế Strategy

Class PaymentController có thuộc tính Strategy là: private Card card;

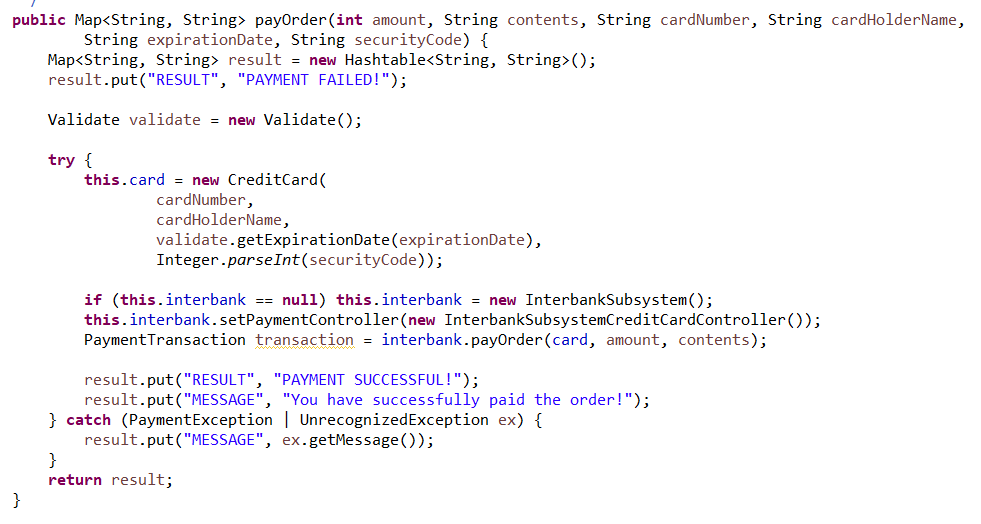


Trong class InterbankSubsystem có thuộc tính Strategy là:

private InterbankSubsystemControllerInterface ctrl;

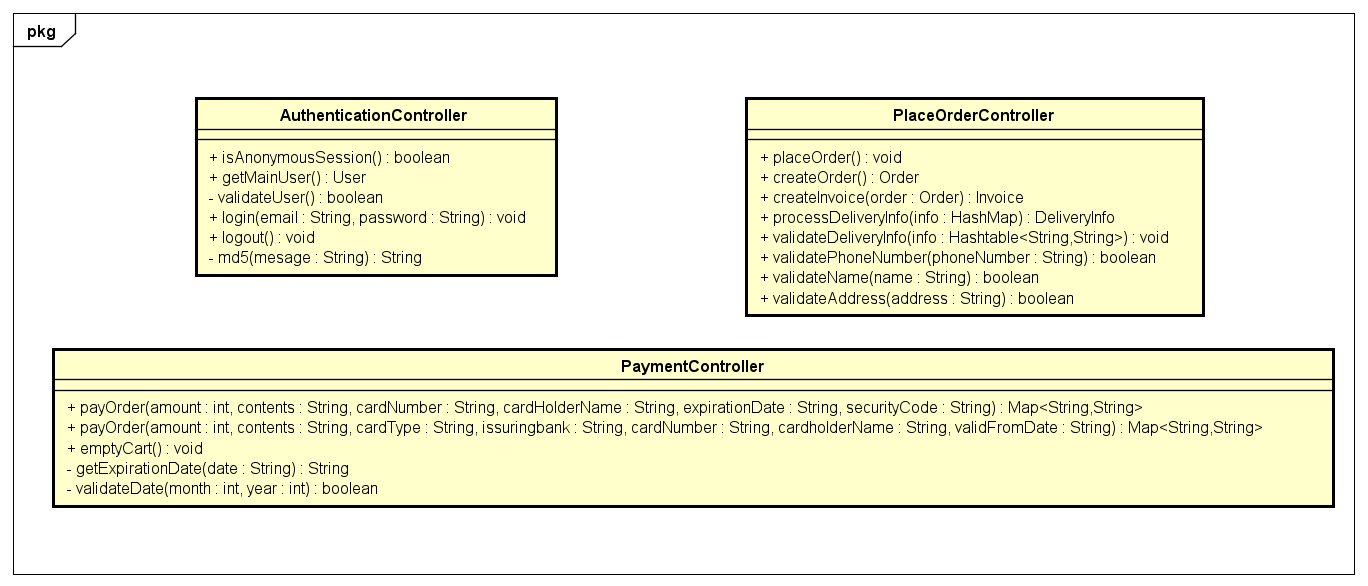


Khi khách muốn thanh toán, trong method payOrder() của class PaymentController sẽ setCard là CreditCard và setPaymentController là InterbankSubsystemCreditCardController tương ứng với CreditCard hoặc setCard là DomesticCard và setPaymentController là InterbankSubsystemDomesticCardController tương ứng với DomesticCard



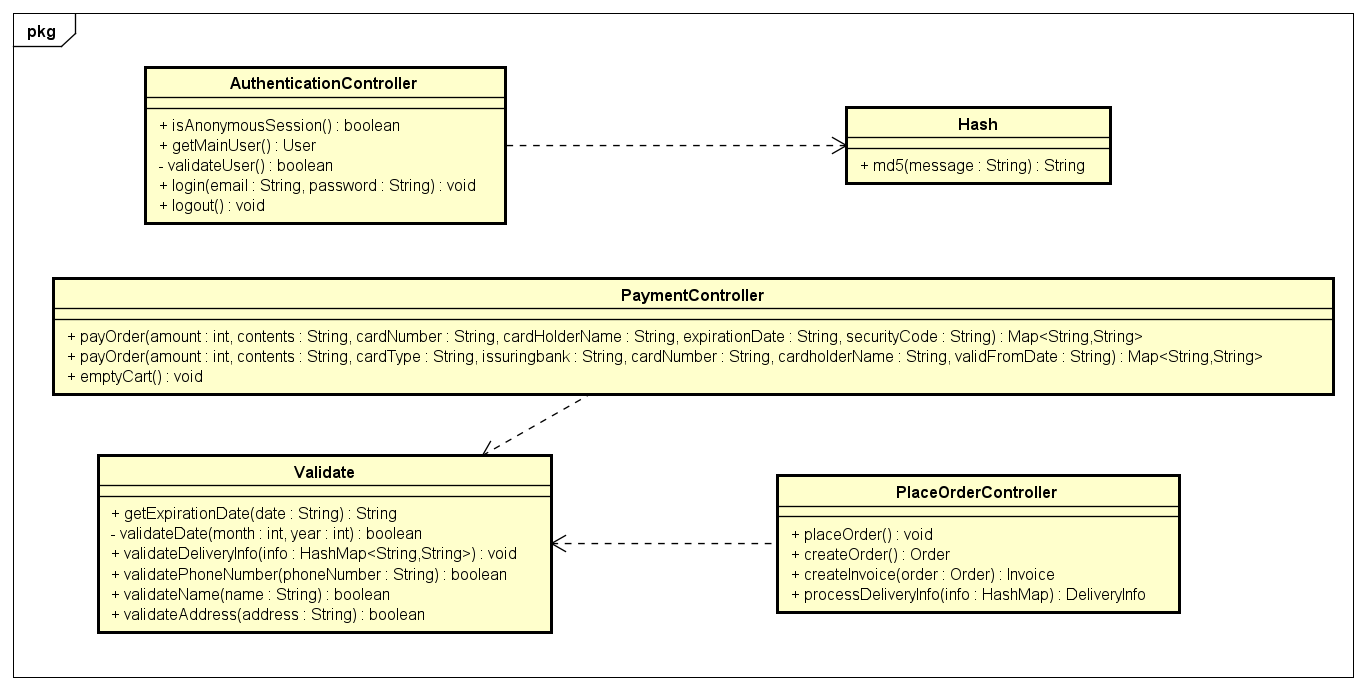
## Vấn đề clean class các large class: AuthenticationController, PaymentController, PLaceOrderController

**Vấn đề**: Class Authentication vừa làm nhiệm vụ Authentication, vừa làm nhiệm vụ mã hóa md5, class PlaceOrderController vừa làm nhiệm vụ đặt hàng vừa có cả phương thức validate thông tin chuyển phát, class PaymentController, ngoài method liên quan đến payment thì còn có isValidMonthandYear,



Hình 5: Các phương thức trong một số controller

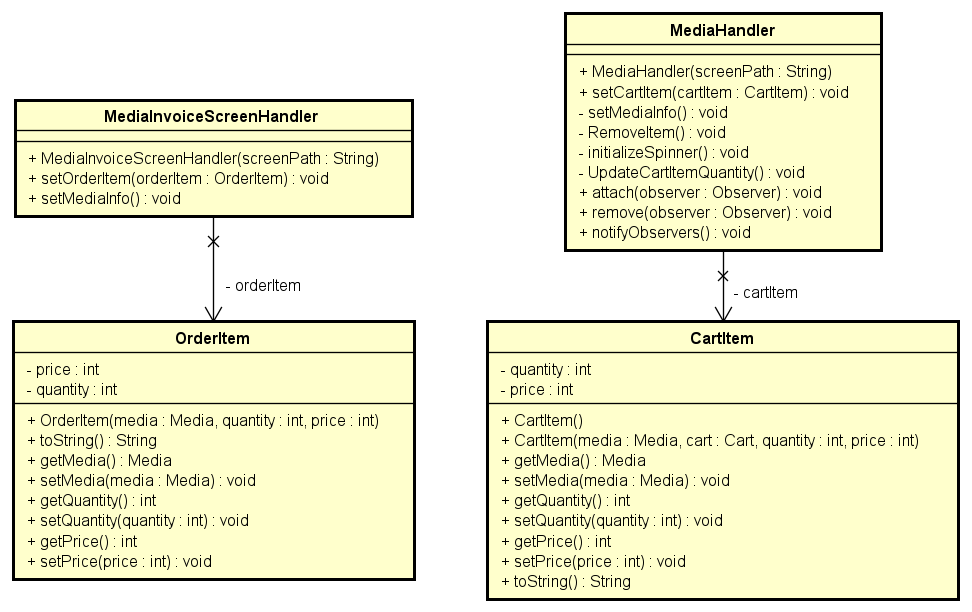
**Giải pháp**: Chuyển các method không liên quan đến class đó sang 1 class khác



Hình 6: Các phương thức tại mỗi lớp sau khi đã tái cấu trúc code

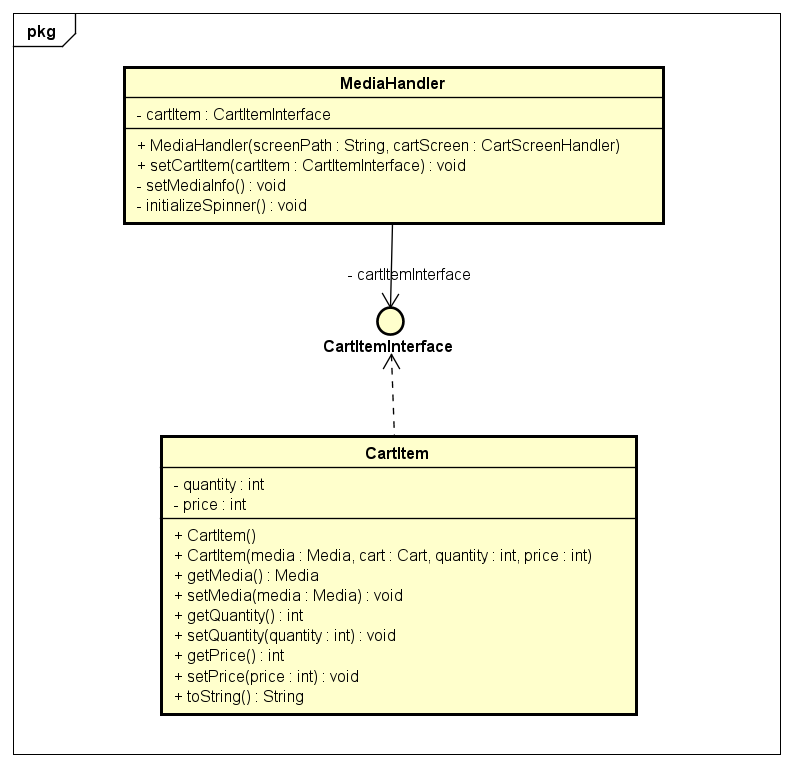
## Vấn đề clean class giữa hai class CartItem và MediaHandler của gói views.screen.cart, giữa hai class OrderItem và MediaInvoiceScreenHandler của gói views.screen.invoice

**Vấn đề**: Hai class CartItem và MediaHandler của gói views.screen.cart, hai class OrderItem và MediaInvoiceScreenHandler của gói views.screen.invoice coupling với nhau. Khi class CartItem thay đổi thì cũng phải thay đổi code hiển thị sản phẩm trong class MediaHandler.



Hình 7: Coupling giữa hai lớp MediaInvoiceScreenHandler-OrderItem, Mediahandler-CartItem

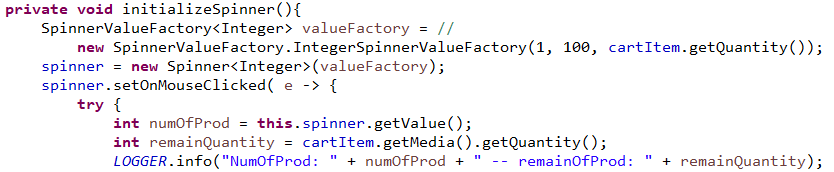
**Giải pháp:** Tạo interface mới là CartItemInterface và cho class CartItem implement. Khi đó, class MediaHandler sẽ không phụ thuộc vào class CartItem nữa.



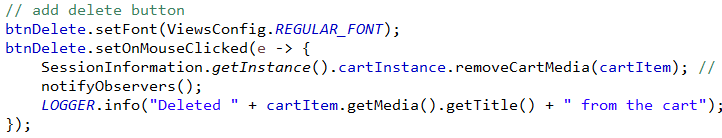
Hình 8: Bổ sung interface giữa hai lớp MediaHandler và CartItem

## Vấn đề clean method tại CartMediaHandler và giải pháp

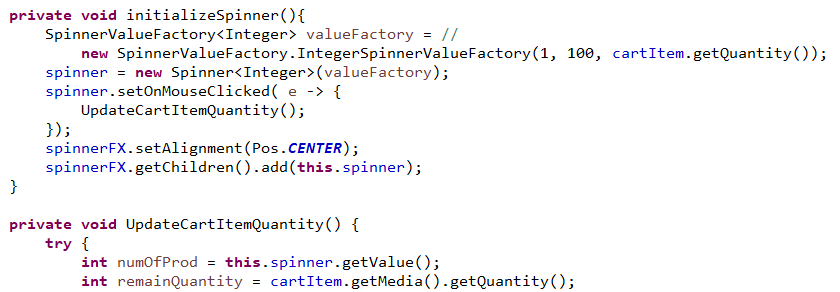
**Vấn đề:** Class views.screen.cart.MediaHandler, method initializeSpinner(), có 2 nhiệm vụ, (i) tạo ra Spinner, (ii) xử lý sự kiện khi click vào spinner. Việc này làm giảm tính dễ đọc của code.



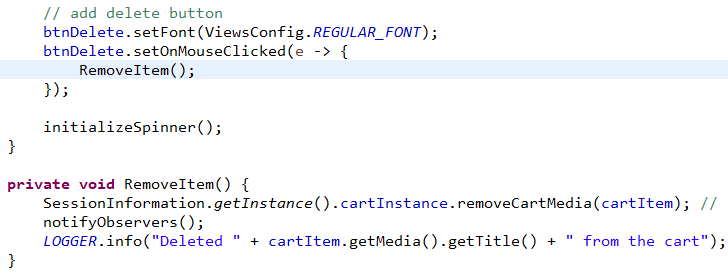
Class views.screen.cart.MediaHandler, method setMediaInfo() có 2 nhiệm vụ, vừa thiết lập các thông số về đồ họa, vừa xử lý sự kiện khi click vào nút Delete. Việc này làm giảm tính dễ đọc của code.

****

**Giải pháp:** Xử lý ở method-level, bằng cách tách phần xử lý khi click spinner thành một private method riêng.



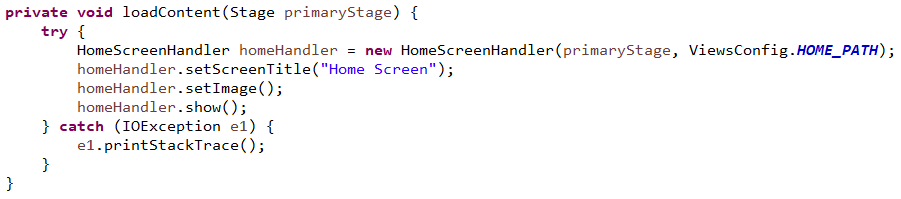
Tách phần xử lý khi click nút Delete thành một phương thức riêng.



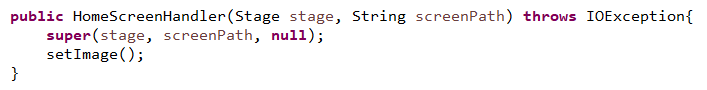
## Vấn đề clean class tại HomeScreenHandler và giải pháp

**Vấn đề:** HomeScreenHandler đưa ra các phương thức public setImage(), trong khi hoàn toàn có thể được gọi từ bên trong lớp đó, không có tác dụng gì khi gọi từ bên ngoài.

Trong ví dụ dưới đây, tại phương thức loadContent() của lớp App là lớp duy nhất có gọi tới hàm setImage() của HomeScreenHandler.



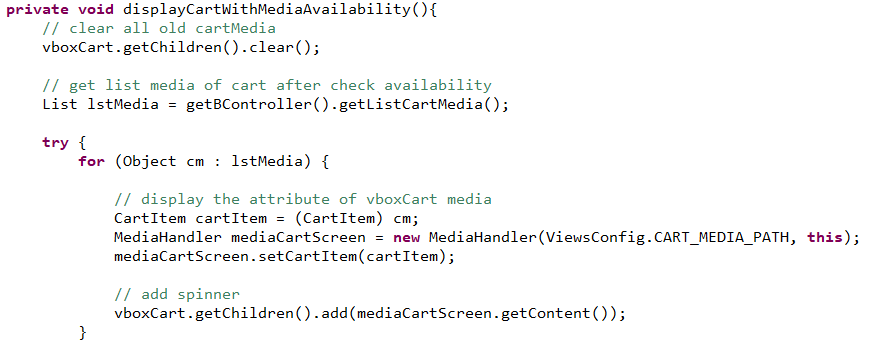
Giải pháp: Đưa phương thức setImage() về private, gọi từ hàm khởi tạo.



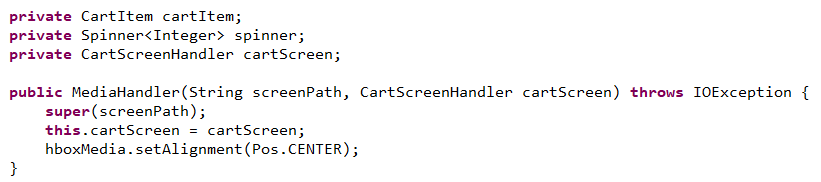
## Vấn đề coupling tại CartScreenHandler - MediaHandler và giải pháp dùng observer pattern

**Vấn đề:** Hai lớp views.screen.cart.MediaHandler, views.screen.cart.CartScreenHandler có sự phụ thuộc lẫn nhau.

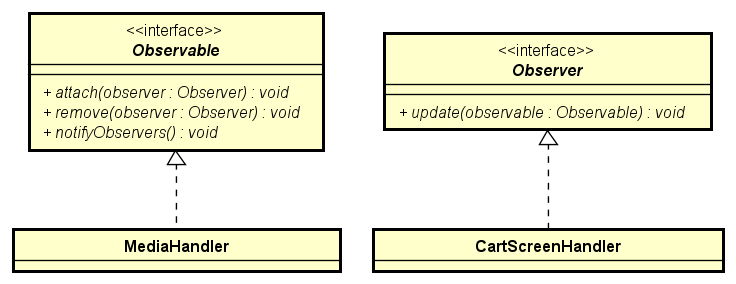
Tại CartScreenHandler, phương thức displayCartWithMediaAvailability() có chưa đoạn mã sinh ra MediaHandler.



Đồng thời, MediaHandler cũng chứa reference tới CartScreenHandler.

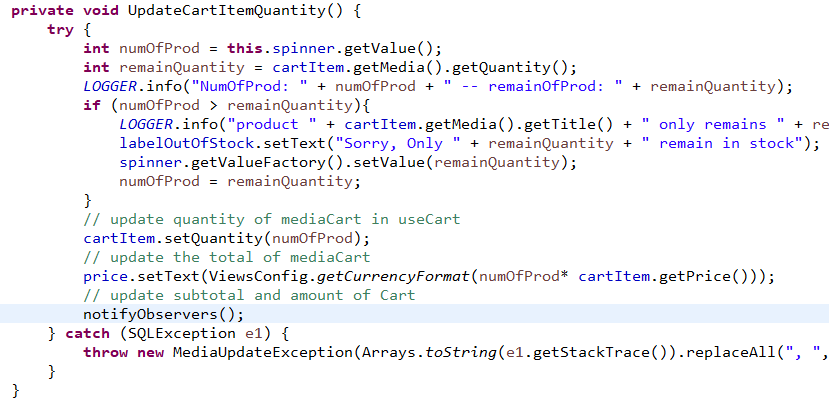


**Giải pháp:** sử dụng Observer pattern, với MediaHandler implement Observable và CartScreenHandler implement Observer.

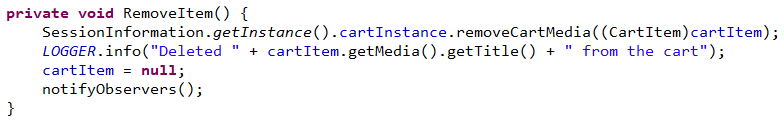


Hình 9: Áp dụng observer pattern vào tái cấu trúc các lớp MediaHandler và CartScreenHandler

MediaHandler có hai sự kiện cần thông báo cho Observer: (i) khi người dùng xóa một mặt hàng và (ii) khi người dùng thay đổi số lượng một mặt hàng.

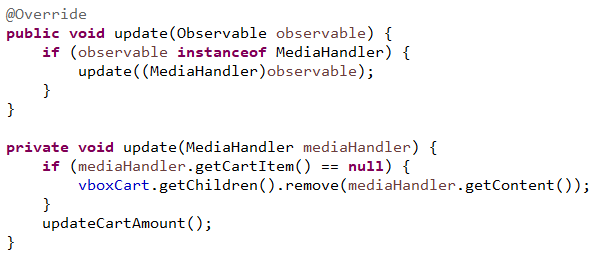


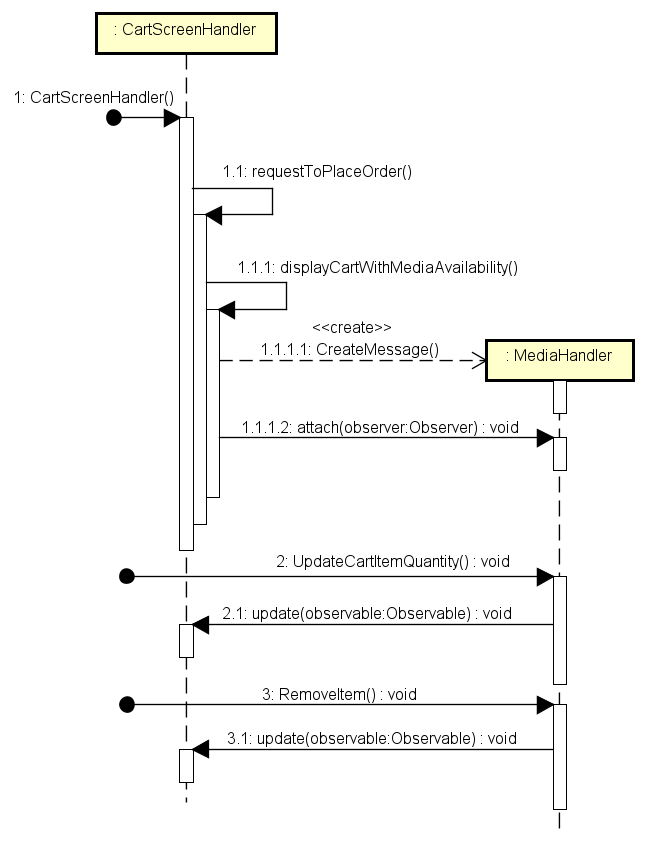
Cần có cách để nhận biết khi MediaHandler thông báo update() tới Observer, đó là cập nhật cho việc xóa hay thay đổi số lượng mặt hàng. Ta sẽ nhận biết thay đổi này thông qua thuộc tính cartItem. Khi xóa, ta set cartItem=null, đồng thời cung cấp phương thức get cho thuộc tính này.



Trong code base ban đầu, tại CartScreenHandler, mỗi lần delete một mặt hàng, ta phải tạo lại danh sách hiển thị các mặt hàng từ đầu, sử dụng hàm updateCart(). Điều này không cần thiết, làm giảm hiệu năng của chương trình. Thay vì gọi hàm updateCart(), ta chỉ cần xóa đối tượng hiển thị trong vboxCart.

Cả hai sự kiện xóa và thay đổi số lượng mặt hàng đều cần cập nhật lại tổng giá tiền, do đó luôn gọi hàm updateCartAmount() trong hàm update().

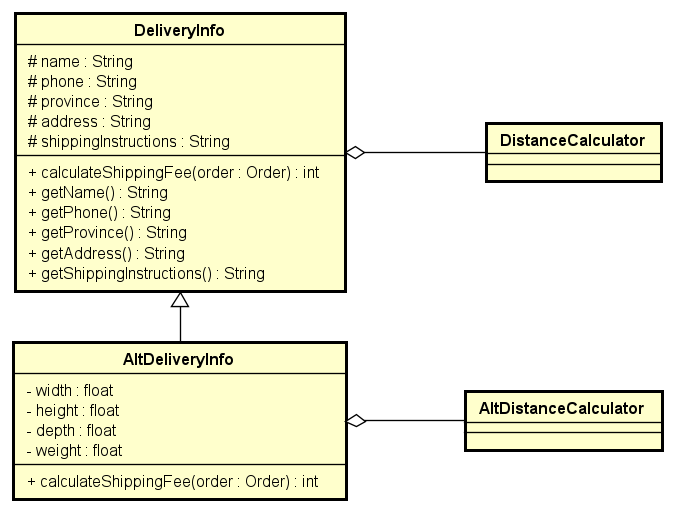




Hình 10: Biểu đồ trình tự, thể hiện tương tác giữa hai lớp CartScreenHandler và MediaHandler.

## Vấn đề cập nhật yêu cầu: Thay đổi công thức tính vận chuyển

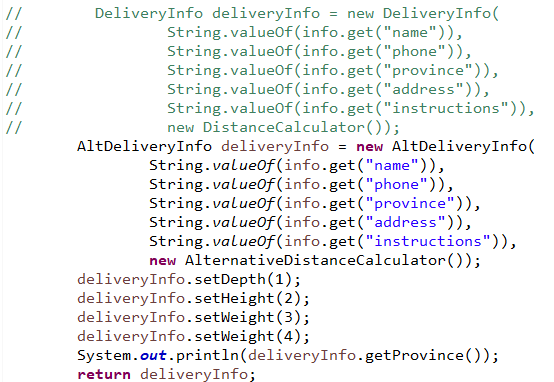
Vấn đề: Yêu cầu thay đổi công thức tính vận chuyển. Với việc sử dụng thư viện tính khoảng cách mới, nếu không muốn sửa lại các lớp hiện có, ta cần sử dụng adapter.



Hình 11: Lớp adapter AltDeliveryInfo

Ta tạo thêm lớp AltDeliveryInfo kế thừa từ DeliveryInfo, bổ sung thêm các thuộc tính cần thiết và override lại calculateShippingFee(). Bên trong phương thức này sẽ gọi tới thư viện mới để thực hiện tính toán.

Tại client, cụ thể là PlaceOrderController, tại phương thức processDeliveryInfo(), ta sửa lại cách khởi tạo DeliveryInfo:



Phương thức này vẫn giữ nguyên giá trị trả về là DeliveryInfo, do đó các thành phần khác của hệ thống sẽ không bị ảnh hưởng.

## Vấn đề cập nhật yêu cầu: Cập nhật chức năng hủy đơn hàng

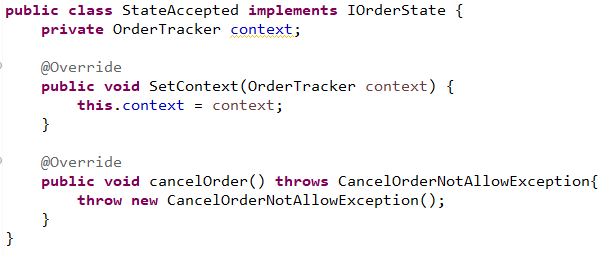
Yêu cầu mới là cập nhật chức năng hủy đơn hàng.

Đây là chức năng mới, chưa có trong hệ thống và có đặc điểm khác với các use case đã có. Do đó, ta cần tạo một controller mới là OrderManagementController.

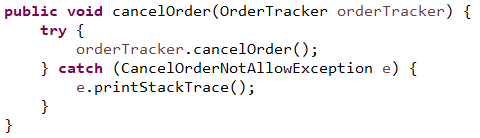
Để nhận biết được đơn hàng đã được quản trị viên chấp nhận hay chưa, ta cần thêm một trường thông tin về trạng thái đơn hàng. Để không phải chỉnh sửa lại lớp Order, ta tạo thêm một lớp OrderTracker có chứa đối tượng Order bên trong.

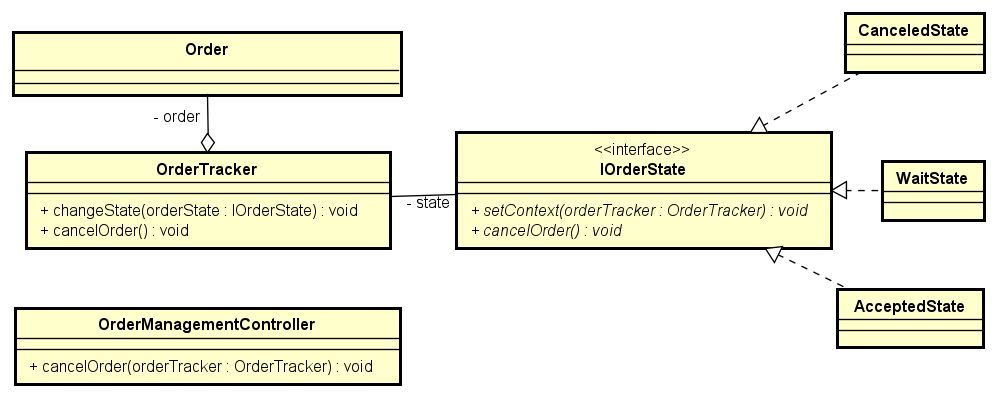
Ta sử dụng state pattern với OrderTracker là context và tạo thêm các state kế thừa từ một interface là IOrderState.

OrderTracker cung cấp phương thức cancelOrder() cho phép người dùng hủy đơn hàng, tùy thuộc vào trạng thái đơn hàng hiện tại sẽ có cách xử lý lệnh cancelOrder() khác nhau. Trường hợp không cho phép hủy đơn hàng, hàm cancelOrder() tại các state này sẽ trả về exception.



Controller sẽ bắt các exception này để xử lý, đưa thông báo cho người dùng.





Hình 12: Biểu đồ các lớp phục vụ chức năng theo dõi đơn hàng

# Tổng kết

## Kết quả tổng quan

Đánh giá được một cách tổng quan về các mức độ coupling, cohesion, clean code. Tìm kiếm và sửa chữa được những phần gây ra high coupling và low cohesion. Mã nguồn sau khi refactor đạt được mức coupling và cohesion tốt hơn.

Đánh giá và sửa chữa được một số lỗi trong mã nguồn không tuân theo nguyên lý SOLID.

Thực hiện tái cấu trúc lại mã nguồn để thỏa mãn hầu hết các yêu cầu phát sinh của hệ thống.

Chương trình hoạt động tốt, không có lỗi phát sinh sau quá trình tái cấu trúc.

## Các vấn đề tồn đọng

Một số lỗi về việc không tuân thủ nguyên lý SOLID, gây ra high coupling và low cohesion vẫn chưa được sửa chữa một cách hoàn thiện.