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.20202.03

**Thành viên**

Phương Trung Đức – 20173030

Hoa Xuân Dương – 20173068

Phạm Bá Đức – 20173036

Lê Minh Dương - 20173070

*Hà Nội,* *ngày 05 tháng 06 năm 2021*

Mục lục

Mục lục 1

1 Tổng quan 4

1.1 Mục tiêu 4

1.2 Phạm vi 4

1.3 Danh sách thuật ngữ 5

1.4 Danh sách tham khảo 5

2 Đánh giá thiết kế cũ 6

2.1 Nhận xét chung 6

2.2 Đánh giá các mức độ coupling và cohesion 6

2.2.1 Coupling 6

2.2.2 Cohesion 8

2.3 Đánh giá việc tuân theo SOLID 9

2.3.1 SRP 9

2.3.2 OCP 9

2.3.3 LSP 10

2.3.4 ISP 10

2.3.5 DIP 10

2.4 Các vấn đề về Clean Code 11

2.4.1 Clear Name 11

2.4.2 Clean Function/Method 12

2.4.3 Clean Class 12

2.5 Các vấn đề khác 13

3 Đề xuất cải tiến 15

3.1 Vấn đề “Thêm mặt hàng Media mới: AudioBook” và giải pháp 15

3.2 Vấn đề “Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)” và giải pháp 16

3.3 Vấn đề “Thay đổi công thức tính phí vận chuyển” và giải pháp 18

3.4 Vấn đề “Thay đổi cách tính khoảng cách, sử dụng thư viện mới” và giải pháp 19

3.5 Vấn đề “Hệ thống chưa xử lý khi người dùng thay đổi các mặt hàng trong giỏ hàng” và giải pháp 22

3.6 Vấn đề “Cập nhật lại chức năng hủy đơn hàng” và giải pháp 25

3.7 Vấn đề “Các hàm không liên quan đến nghiệp vụ của một số lớp” và giải pháp 26

3.8 Vấn đề “Thêm màn hình: Xem chi tiết sản phẩm” và giải pháp 26

4 Tổng kết 29

4.1 Kết quả tổng quan 29

4.2 Các vấn đề tồn đọng 29

5 Phân công công việc trong nhóm 30

**Danh sách các ảnh minh họa**

[Hình 2.1 : Áp dụng template method 13](#_Toc74081200)

[Hình 3.1 Biểu đồ lớp của lớp Media và các lớp con 15](#_Toc74081201)

[Hình 3.2 Biểu đồ lớp của lớp AudioBook 16](#_Toc74081202)

[Hình 3.3 Biểu đồ lớp của thiết kế cũ liên quan đến thẻ Credit 17](#_Toc74081203)

[Hình 3.4 Biểu đồ lớp interface Card khi thêm phương thức thanh toán 18](#_Toc74081204)

[Hình 3.5 Biểu đồ lớp InterbankInterface 18](#_Toc74081205)

[Hình 3.6 Biểu đồ lớp thể hiện mối quan hệ giữa DistanceCalculator với class khác 19](#_Toc74081206)

[Hình 3.7 Biểu đồ lớp thể hiện thay đổi cách tính phí vận chuyển 19](#_Toc74081207)

[Hình 3.8 Biểu đồ lớp sử dụngDistanceCalculator 20](#_Toc74081208)

[Hình 3.9 Biểu đồ lớp sử dụng Strategy Pattern để thay đổi cách tính khoảng cách 20](#_Toc74081209)

[Hình 3.10 Biểu đồ lớp Sử dụng Adapter Pattern để thay đổi cách tính khoảng cách 21](#_Toc74081210)

[Hình 3.11 Phương thức gọi đến thư viện mới 21](#_Toc74081211)

[Hình 3.12 Sử dụng thư viện mới trong class DeliveryInfo 22](#_Toc74081212)

[Hình 3.13 Biểu đồ lớp các lớp views 22](#_Toc74081213)

[Hình 3.14 Biểu đồ lớp các lớp view sau khi sửa 23](#_Toc74081214)

[Hình 3.15 Biểu đồ lớp CartHandler với Observer Pattern 24](#_Toc74081215)

[Hình 3.16 Biểu đồ sử dụng State Pattern để tạo chức năng Hủy đơn hàng 25](https://husteduvn-my.sharepoint.com/personal/duong_hx173068_sis_hust_edu_vn/Documents/DP-03.docx#_Toc74081216)

[Hình 3.17 Biểu đồ lớp giao diện Home 26](#_Toc74081217)

[Hình 3.18 Biểu đồ lớp của lớp Media với các lớp con 27](#_Toc74081218)

[Hình 3.19 Mã nguồn vi phạm control coupling 27](#_Toc74081219)

[Hình 3.20 Biểu đồ lớp Media với phương thức getAllProp 28](#_Toc74081220)

**Danh sách các bảng**

[Bảng 2.1 Bảng đánh giá về Coupling 6](#_Toc74081221)

[Bảng 2.2 Bảng đánh giá về Cohesion 8](#_Toc74081222)

[Bảng 2.3 Bảng đánh giá về SRP 9](#_Toc74081223)

[Bảng 2.4 Bảng đánh giá về OCP 9](#_Toc74081224)

[Bảng 2.5 Bảng đánh giá về LSP 10](#_Toc74081225)

[Bảng 2.6 Bảng đánh giá về ISP 10](#_Toc74081226)

[Bảng 2.7 Bảng đánh giá về DIP 11](#_Toc74081227)

[Bảng 2.8 Bảng đánh giá về Clear Name 11](#_Toc74081228)

[Bảng 2.9 Bảng đánh giá về Clean Function/Method 12](#_Toc74081229)

[Bảng 2.10 Bảng đánh giá về Clean Class 12](#_Toc74081230)

[Bảng 2.11 Bảng chi tiết về các vấn đề khác 14](#_Toc74081231)

[Bảng 5.1 Bảng phân công công việc 30](#_Toc74081232)

# Tổng quan

Báo cáo này tập trung phân tích, đánh giá và chỉnh sủa lại mã nguồn theo các nguyên lý clean code, các mẫu design pattern đã học nhằm xây dựng một hệ thống tối ưu linh hoạt đối với các yêu cầu thay đổi trong tương lai.

## Mục tiêu

Báo cáo cung cấp các thông tin liên quan đến thiết kể của các module, các method, cách thước đáp ứng sự thay đổi trong tương lai. Các thông itn này là hữu ích cho các hoạt động bảo trì hệ thống sau nàynày, khi có các yêu cầu phát triển mới, giúp nhanh chóng phát triển các tính năng của hệ thống mà không cần can thiệp vào mã nguồn ban đầu.

## Phạm vi

* Phần mềm: AIMS - An Internet Media Store
* Chức năng của phần mềm:
* Tìm kiếm sản phẩm
* Xem danh sách sản phẩm
* Xem chi tiết sản phẩm
* Đặt hàng
* Xem giỏ hàng
* Xóa sản phẩm khỏi giỏ hàng
* Cấu trúc mã nguồn:
* Common package: chứa các exception sử dụng cho ứng dụng
* Controller package: xử lý các nghiệp vụ logic của bài toán
* Dao package: chứa các method sử dụng để nhập xuất với Database
* Entity package: các thực thể được sử dụng trong bài toán
* Subsystem package: chứa hệ thống con làm nhiệm vụ giao tiếp với hệ thống ngân hàng giả định khi thanh toán
* Utils package: định nghĩa các class, method sử dụng khi giao tiếp với server
* Views.creen package: chứa các màn hình của ứng dụng
* App.java: class bắt đầu ứng dụng
* Resources package: các tài nguyên được sử dụng trong ứng dụng
* Các yêu cầu thêm cần cân nhắc khi tái cấu trúc:
* Thêm mặt hàng Media mới: Audio Book
* Thêm màn hình: Xem chi tiết sản phẩm
* Thay đổi yêu cầu khi load giao diện
* Thay đổi cách tính khoảng cách, sử dụng thư viện mới
* Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)
* Cập nhật lại chức năng hủy đơn hàng
* Các hoạt động thực thi trên mã nguồn: Mã nguồn sẽ được đánh giá, sửa đổi lại đồng thời thêm mới các chức năng clean nhất có thể đồng thời thỏa mãn hiệu năng của hệ thống.
* Kết quả dự kiến: Hoàn thiện các yêu cầu thay đổi, sửa đỗi các phần mã nguồn chưa clean, hệ thống có thể bảo trì tốt và dễ dàng sửa đổi đối với các sự thay đổi trong tương lai.

## Danh sách thuật ngữ

* SRP : Single Responsibility Principle
* OCP : Open/Close Principle
* LSP : Liskov Substitution Principle
* ISP : Interface Segregation principle.
* DIP : Dependency Inversion Principle.

## 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

3. Lý thuyết các mẫu thiết kế phần mềm: <https://gpcoder.com/>

# Đánh giá thiết kế cũ

## Nhận xét chung

Mã nguồn hiện tại hoàn toàn có thể giúp hệ thống vận hành một cách bình thường. Tuy nhiên chưa tối ưu, rất nhiều vị trí mã nguồn chưa tuân thủ theo các nguyên tắc coupling va cohesion có thể khiến hệ thống vận hành sai, khó bảo trì và thêm các yêu cầu trong tương lai.

## Đánh giá các mức độ coupling và cohesion

Coupling: Mức độ quan hệ phụ thuộc giữa các module với nhau

Cohension: Mức độ liên quan, tập trung về mặt chức năng trong một module

Một thiết kết tốt (Loose coupling & Tight cohension) là thiết kế làm cho mức độ phụ thuộc giữa các module ít nhất có thể và mức độ tập trung

### Coupling

Trong mã nguồn có một số vị trí khiến mức độ quan hệ giứa các module tăng cao và chưa tối ưu.

Bảng 2.1 Bảng đánh giá về Coupling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Lý do* |
| 1 | Stamp | InterbankPayloadConverter | Phương thức convertToRequestPayload(CreditCard card, int amount, String contents) | Khi có sự thay đổi về thẻ thanh toán như có thêm loại thẻ phải thay đổi tham số CreditCard trong code |
| InterbankSubsystemController | refund(CreditCard card, int amount, String contents) | Vì truyền cả đối tượng credit card trong params |
| DeliveryInfo | Phương thức: processDeliveryInfo(HashMap info) | Truyền vào param info dạng Hash Map nhưng thực tế chỉ sử dụng các trường: name, phone, province, address, instructions |
| PlaceOrderController | Phương thức: validateDeliveryInfo(HashMap<String, String> info) | Truyền vào param dạng hash map nhưng thực tế chỉ sử dụng đến các trường: phone, name, address |
| BaseController | checkMediaInCart(Media media) | Truyền cả đối tượng Media trong khi chỉ sử dụng mỗi trường Id |
| DeliveryInfo | calculateShippingFee(Order order) | Truyền vào order nhưng không sử dụng |
| 2 | Control |  |  |  |
| 3 | Common | SessionInformation |  | Các thuộc tính đều ở dạng public, tuy nhiên không cung cấp các phương thức get và set để thay đổi chúng nên các module sử dụng đều phải truy cập trực tiếp vào thuộc tính để thay đổi. Gây ra cả content và common ở một số module |
|  |  | ShippingConfigs | public static String[] PROVINCES | Việc để biến static không có final làm cho các module sử dụng chung có thể thay đổi biến chung này |
| 4 | Content | SessionInformation |  | Các thuộc tính đều ở dạng public, tuy nhiên không cung cấp các phương thức get và set để thay đổi chúng nên các module sử dụng đều phải truy cập trực tiếp vào thuộc tính để thay đổi. Gây ra cả content và common ở một số module |

### Cohesion

Bảng 2.2 Bảng đánh giá về Cohesion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Lý do* |
| 1 | Coincidental | ViewsConfig | getCurrencyFormat(int num) | Vì hàm không liên quan đến Class |
| InterbankPayloadConverter | getToday() | Thực hiện chức năng không liên quan đến các hàm khác của class |
| PlaceOrderController |  | Cac phuong thuc validate khong lien quan den nghiep vu cua lop, nen tach ra thanh mot lop rieng de xu ly |
| PaymentController | Method: getExpirationDate(String date) | Hàm này không liên quan đến các hàm khác và không liên quan đến nghiệp vụ xử lí của lớp |
| 2 | Logical | PlaceOrderController |  | Các hàm validate chỉ liên quan với nhau về mặt logic nhưng không liên quan đến nhau về mặt chức năng |
| InterbankPayloadConverter | hàm convertJSONResponse() | hàm chỉ liên quan với nhau về mặt logic nhưng không liên quan đến nhau về mặt chức năng |
| 3 | Temporal | InterbankPayloadConverter | hàm extractPaymentTransaction() |  |
| 4 | Procedural |  |  |  |
| 5 | Communicational |  |  |  |
| 6 | Functional | Cart |  | Các phương thức đều sử dụng chung data là lstCartItem và đều thực hiện các chức năng liên quan đến giỏ hàng |

## Đánh giá việc tuân theo SOLID

*Mã nguồn hiện tại chưa tuân thủ hoàn toàn nguyên lý SOLID.*

### SRP

Nguyên lý SRP: Mỗi module chỉ nên chịu trách nhiệm cho một tính năng duy nhất. Hiện tại một số vị trí trong mã nguồn chưa thỏa mãn điều kiện trên

Bảng 2.3 Bảng đánh giá về SRP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | InterbankPayloadConverter | getToday() | Thực hiện chức năng không liên quan đến các hàm khác của class |
| 2 | PlaceOrderController |  | Các phương thức validate không liên quan đến nghiệp vụ của lớp |
| 3 | PaymentController | Method: getExpirationDate(String date) | Hàm này không liên quan đến các hàm khác và không liên quan đến nghiệp vụ xử lí của lớp |

### OCP

Nguyên lý OCP: Hạn chế việc điều chỉnh code bên trong một module thay vào đó linh hoạt tạo mới hoặc bổ sung các module mà không cần sửa đổi mã nguồn

Bảng 2.4 Bảng đánh giá về OCP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *#* | *Module* | | *Mô tả* | | *Lý do* | |
| 1 | | DeliveryInfo | | calculateShippingFee | | vì khi muốn đổi sang cách tính phí giao hàng khác thì phải sửa các lớp khác | |
| 2 | | PlaceOrderController | | hàm processDeliveryInfo | | vì khi muốn đổi sang cách tính phí giao hàng khác thì phải sửa các lớp khác | |
| 3 | | CreditCard card; | | Khi thay đổi thẻ thanh toán phải thay đổi trực tiếp code trong class này | |
| 4 | | PaymentTransaction,InterbankInterface,  InterbankSubsystemController | | CreditCard card; | | Khi thay đổi thẻ thanh toán phải thay đổi trực tiếp code trong class này | |
| 5 | | DeliveryInfo | | protected DistanceCalculator distanceCalculator; | | Phụ thuộc trực tiếp vào class DistanceCalculator, khi class có sự thay đổi kéo theo lớp cũng cần thay đổi code | |

### LSP

Nguyên lý LSP: Các thể hiện của lớp con vẫn hoạt động đúng đối với các chứng nang mà lớp cha cung cấpcấp

Bảng 2.5 Bảng đánh giá về LSP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | extend BaseController | Kế thừa từ BaseController nhưng hai phương thức của Base không hề được override lại hay sử dụng ở bất kì đâu |

### ISP

Nguyên lý ISP: chia nhỏ interface với nhiều mục đích cụ thể.

Bảng 2.6 Bảng đánh giá về ISP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 |  |  |  |
| 2 |  |  |  |

### DIP

Nguyên lý DIP: Mỗi module chi nên phụ thuộc vào abtractions, không nên phụ thuộc vào các chi tiết cụ thể

Bảng 2.7 Bảng đánh giá về DIP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | PaymentController, PaymentTransaction,  InterbankInterface,  InterbankSubsystemController | Phụ thuộc trực tiếp vào class CreditCard | CreditCard là module mức thấp nên khi có bất kì sự thay đổi nào ở class này, PaymentController cũng phải thay đổi theo |
| 2 | DeliveryInfo | protected DistanceCalculator distanceCalculator; | Class phụ thuộc vào class DistanceCalculator là module mức thấp |

## Các vấn đề về Clean Code

### Clear Name

Mã nguồn ban đầu chưa đáp ứng clean name. Tên một số module, class chưa được đặt tên hợp lý gây khó hiểu, không rõ ý nghĩa

Bảng 2.8 Bảng đánh giá về Clear Name

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | MessageDigest md = MessageDigest.getInstance("MD5");  Đổi tên md thành md5 | Đặt tên thành md5 để biết rõ instance của message digest là loại md5 |
| 2 | ViewsConfig | public static final float FONT\_SIZE = 24 | Để làm rõ ý nghĩa của hằng số 24 |
| 3 | CartScreenHandler | List listCartMedia = getBController().getListCartMedia();  for (Object cartMedia : listCartMedia) | Đặt lại tên biến cho dễ hiểu hơn |
| 4 | PopupScreen | public static final double TIME = 0.8; | Để hiểu rõ hằng số 0.8 có ý nghĩa là gì |
| 5 | PopupScreen | Đổi tên từ method error thành showErrorPopup | Đổi tên để làm rõ vai trò của các method như hiển thị thông báo thành công/ thất bại hoặc tạo ra 1 màn hình loading. |
| Đổi tên từ method success thành showSuccessPopup |
| Đổi tên từ method loading thành createLoadingScreen |
| 6 |  | Đổi tên các method getBController() thành getBaseController() | Làm rõ ý nghĩa của method |

### Clean Function/Method

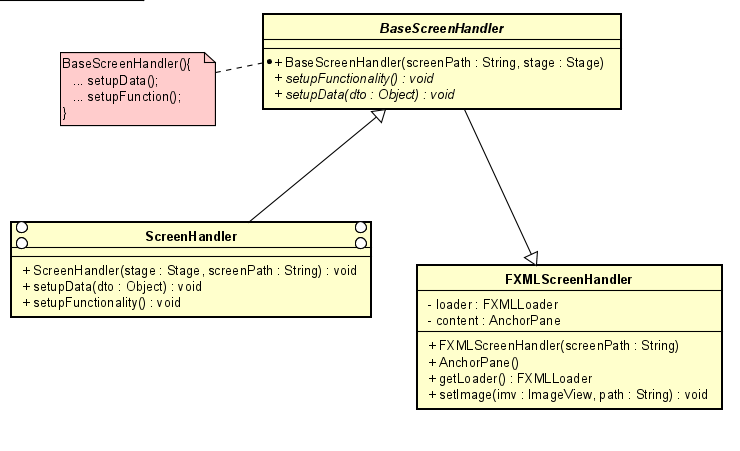
Bảng 2.9 Bảng đánh giá về Clean Function/Method

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | InterbankPayloadConverter | Hàm getToday() được tách ra thành lớp CommonFunctionDate, có nhiệm vụ cung cấp các hàm liên quan đến ngày tháng | Do hàm getToday() không có liên quan gì đến lớp InterbankPayloadConverter. Tạo lớp CommonFunctionDate để sau này có thể tái sử dụng các hàm liên quan đến ngày tháng |
| 2 | ViewsConfig | Hàm getCurrencyFormat() được tách ra ở lớp … | Do hàm getCurrencyFormat() không liên quan về chức năng với lớp ViewsConfig |

### Clean Class

Bảng 2.10 Bảng đánh giá về Clean Class

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | PlaceOrderController | Một số method bị duplicate | Mặc dù chức năng như nhau nhưng method được đặt tên khác nhau. |
| 2 | Package views.screen | Các lớp screen có các thức hoạt động tương tự nhau (constructor) lặp đi lặp lại các đoạn mã nguồn ở lớp con thay vì khai báo cách thức hoạt động ở lớp cha BaseScreenHandler | Có thể sử dụng template method để tối ưu mã nguồn cho các lớp con. (Hình minh họa) |



Hình 2.1 : Áp dụng template method

## Các vấn đề khác

* Rất nhiều lớp hiện đang được khởi tạo đối tượng ở nhiều nơi nhưng với nghiệp vụ thực tế chỉ nên khởi tạo 1 lần duy nhất

+ AuthenticationController : Công việc của class là làm các nhiệm vụ liên quan đến xác thực người dùng, đăng kí, đăng nhập. Vì vậy chỉ cần khởi tạo 1 instance duy nhất và sử dụng xuyên suốt chương trình là đủ đảm bảo.

+ Cart: Xử lý các nghiệp vụ liên quan đến giỏ hàng mà với một tài khoản chỉ sở hữu 1 giỏ hàng. Do vậy chỉ nên tồn tại 1 instance giỏ hàng

* Hiện tại trong giỏ hàng, khi xóa hoặc thay đổi số lượng các mặt hàng chưa có xử lí nhận biết sự thay đổi này. Điều này có thể gây ra một số lỗi như khi khách hàng không may ấn xóa mặt hàng, hệ thống không cung cấp bất kì một phản hồi nào nhằm xác nhận khách hàng có thực sự muốn xóa hay là thao tác nhầm.

Bảng 2.11 Bảng chi tiết về các vấn đề khác

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | AuthenticationController | Sử dụng mẫu thiết kế Singleton để khởi tạo 1 instance duy nhất và sử dụng xuyên suốt chương trình là đủ đảm bảo | Công việc của class là làm các nhiệm vụ liên quan đến xác thực người dùng, đăng kí, đăng nhập |
| 2 | Cart | Sử dụng mẫu thiết kế Singleton để khởi tạo 1 instance giỏ hàng | Xử lý các nghiệp vụ liên quan đến giỏ hàng mà với một tài khoản chỉ sở hữu 1 giỏ hàng |

# Đề xuất cải tiến

## Vấn đề “Thêm mặt hàng Media mới: AudioBook” và giải pháp

* Hiện tại, phần mềm đang thiết kế tất cả các Media: CD, DVD, Book,.. đều kế thừa từ 1 lớp cha là Media và các lớp, các phương thức làm việc với các đối tượng media đều được đại diện bằng class Media. Khi thêm 1 mặt hàng

Diagram

Description automatically generated

Hình 3.1 Biểu đồ lớp của lớp Media và các lớp con

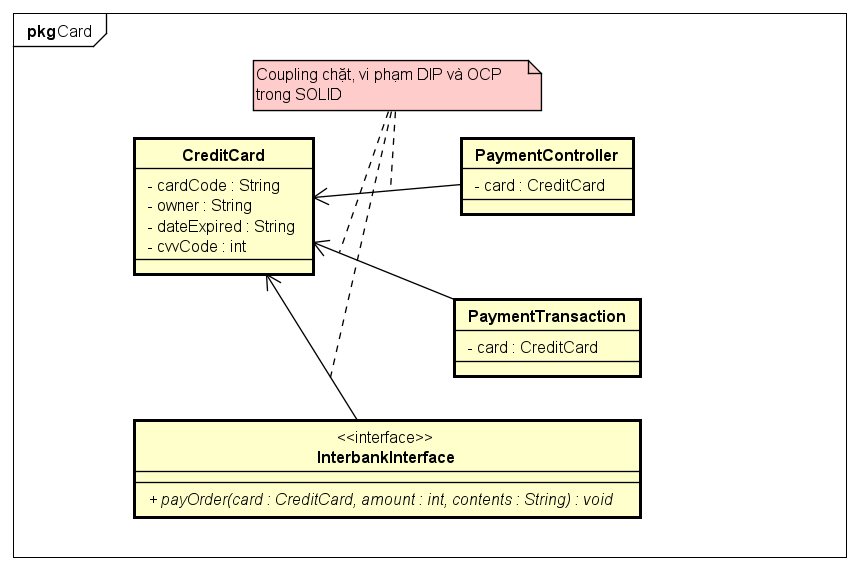
Diagram

Description automatically generated

Hình 3.2 Biểu đồ lớp của lớp AudioBook

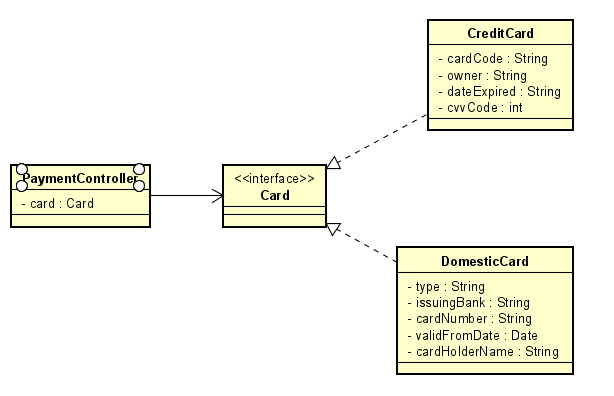
## Vấn đề “Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)” và giải pháp

Với các vấn đề liên quan đến thẻ thanh toán (class CreditCard trên codebase), rất nhiều lớp và phương thức đang phụ thuộc trực tiếp vào class này.



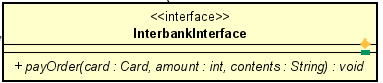
Hình 3.3 Biểu đồ lớp của thiết kế cũ liên quan đến thẻ Credit

* Để khắc phục các vi phạm coupling, SOLID cũng như để đáp ứng tốt nếu có yêu cầu thêm một thẻ thanh toán mới. Bổ sung 1 interface mới cung cấp một giao diện chung cho tất cả các lớp muốn sử dụng thẻ giao tiếp qua đó, các loại thẻ sinh ra đều phải implement interface này.



Hình 3.4 Biểu đồ lớp interface Card khi thêm phương thức thanh toán

Với việc sử dụng một interface làm giao diện để các lớp khác giao tiếp với các loại thẻ giúp hệ thống “flex” hơn khi có thêm các thẻ mới. Đảm bảo không phải thay đổi các lớp đang sử dụng instance của interface này. Thích ứng với yêu cầu chương trình vẫn kết nối với API của interbank mà không phải thay đổi gì.

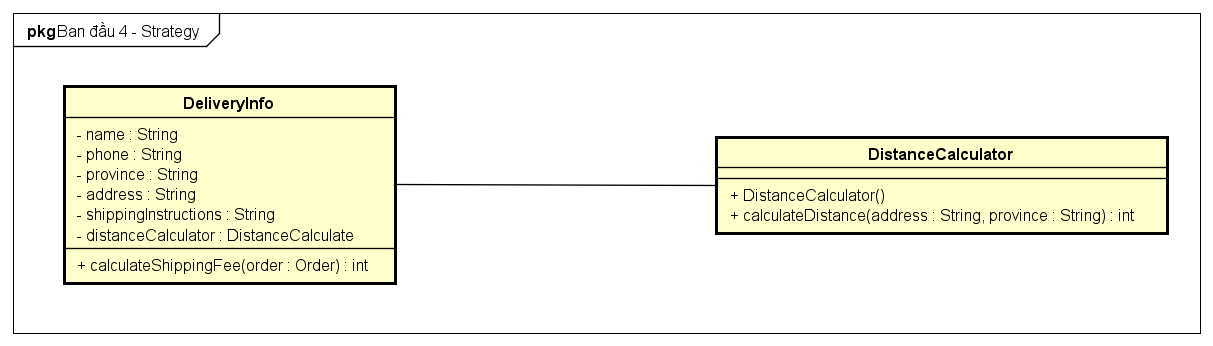


Hình 3.5 Biểu đồ lớp InterbankInterface

Giải quyết được các vấn đề SOLID như OCP và DIP.

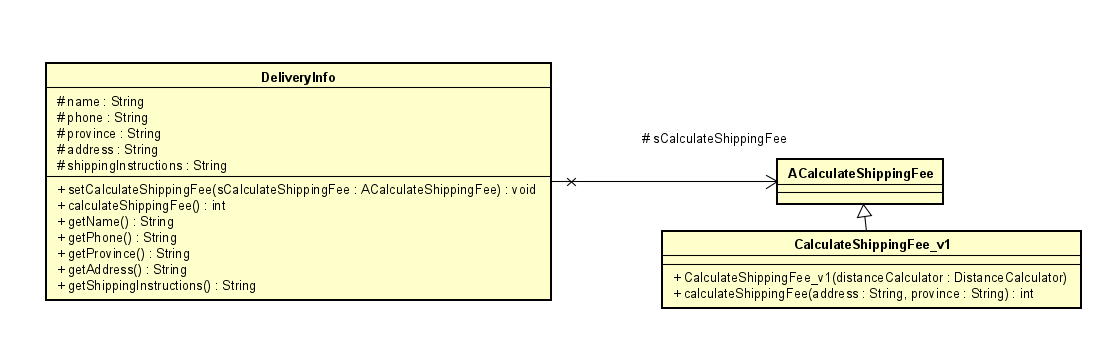
## Vấn đề “Thay đổi công thức tính phí vận chuyển” và giải pháp

Hiện tại, phí vận chuyển trong codebase được tính sau khi xác định được địa chỉ của đơn hàng trong lớp DeliveryInfo dựa vào hàm “int calculateShippingFee()”.



Hình 3.6 Biểu đồ lớp thể hiện mối quan hệ giữa DistanceCalculator với class khác

Khi có bất kì thay đổi sự thay đổi nào về cách tính phí vận chuyển, với thiết kế cũ, ta cần thay đổi trực tiếp code trong hàm calculateShippingFee, đồng thời không thể thay đổi cách tính phí cho đơn hàng trong thời gian chạy. Để cải tiến thiết kế này, nhóm áp dụng strategy pattern để khắc phục các nhược điểm trên:



Hình 3.7 Biểu đồ lớp thể hiện thay đổi cách tính phí vận chuyển

Với việc cải tiến theo mẫu thiết kế sẽ giải quyết được vấn đề phải thay đổi code của class DeliveryInfo mỗi khi có yêu cầu thay đổi cách tính phí vận chuyển – chỉ cần tạo 1 class mới và hiện thực hóa lớp trừu tượng ACalculateShippingFee và setCalCulateShippingFee cho class DeliveryInfo.

## Vấn đề “Thay đổi cách tính khoảng cách, sử dụng thư viện mới” và giải pháp

Trong code base đang sử dụng thư viện tính khoảng cách DistanceCalculator, và các class muốn tính khoảng sử dụng thư viện. Tuy nhiên chính vì vậy, rất nhiều class đang bị phụ thuộc chặt vào class DistanceCalculator vi phạm DIP và OCP

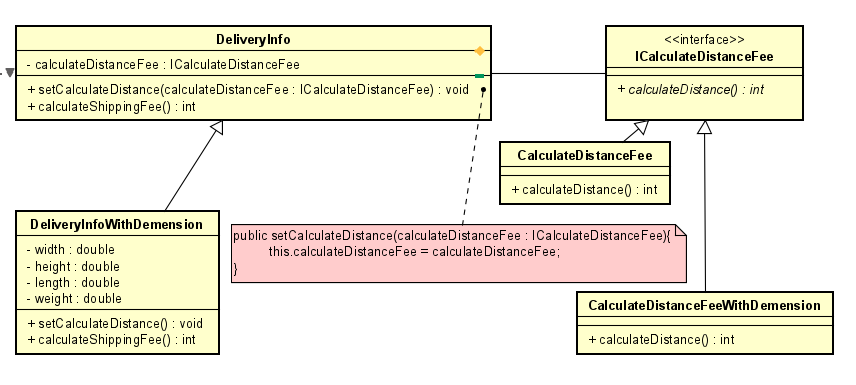
Diagram

Description automatically generated

Hình 3.8 Biểu đồ lớp sử dụngDistanceCalculator

Để giải quyết vấn đề này nhóm đưa ra 2 giải pháp:

* **Giải pháp 1**: Sử dụng Strategy Pattern để thay đổi cách tính khoảng cách



Hình 3.9 Biểu đồ lớp sử dụng Strategy Pattern để thay đổi cách tính khoảng cách

Với cách làm này, ta cần bỏ đi thuộc tính DistanceCalculator distanceCalculator hiện đang được sử dụng và thay thế bằng 1 thuộc tính là instance của interface ICalculateDistance – đóng vai trò là Strategy, phương thức setCalculateDistance() trong class DeliveryInfo sẽ giúp quyết định class nào sẽ được thực thi giúp giải quyết được vấn đề thay đổi thư viện tính khoảng cách.

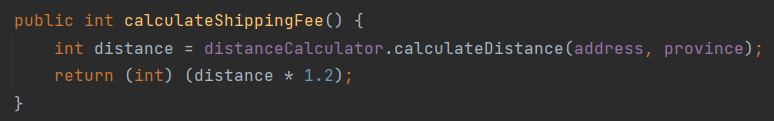
* Khuyết điểm: Với cách cài đặt này chúng ta phải thay đổi trực tiếp codebase bằng việc thay đổi các thuộc tính như đã nêu ở trên.
* **Giải pháp 2**: Sử dụng Adapter Pattern để thay đổi cách tính khoảng cách

Diagram

Description automatically generated

Hình 3.10 Biểu đồ lớp Sử dụng Adapter Pattern để thay đổi cách tính khoảng cách

Với cách sử dụng Adapter như đã minh họa, ta không cần thay đổi codebase gốc của class DeliveryInfo, thay vào đó, ta sẽ khởi tạo đối tượng distanceCalculate chính là thực thể của lớp DistanceCalculateAdapter, khi đó với phương thức tính khoảng cách rất dễ dàng để áp dụng phương thức tính mới.



Hình 3.11 Phương thức gọi đến thư viện mới

Các bước thực hiện:

* Tạo lớp DistanceCalculateAdapter extends DistanceCalculator đóng vai trò chuyển đổi giữa 2 class DistanceCalculator và AlternativeDistanceCalculator
* Override phương thức calculateDistance để gọi phương thức calculateDistance của AlternativeDistanceCalculator

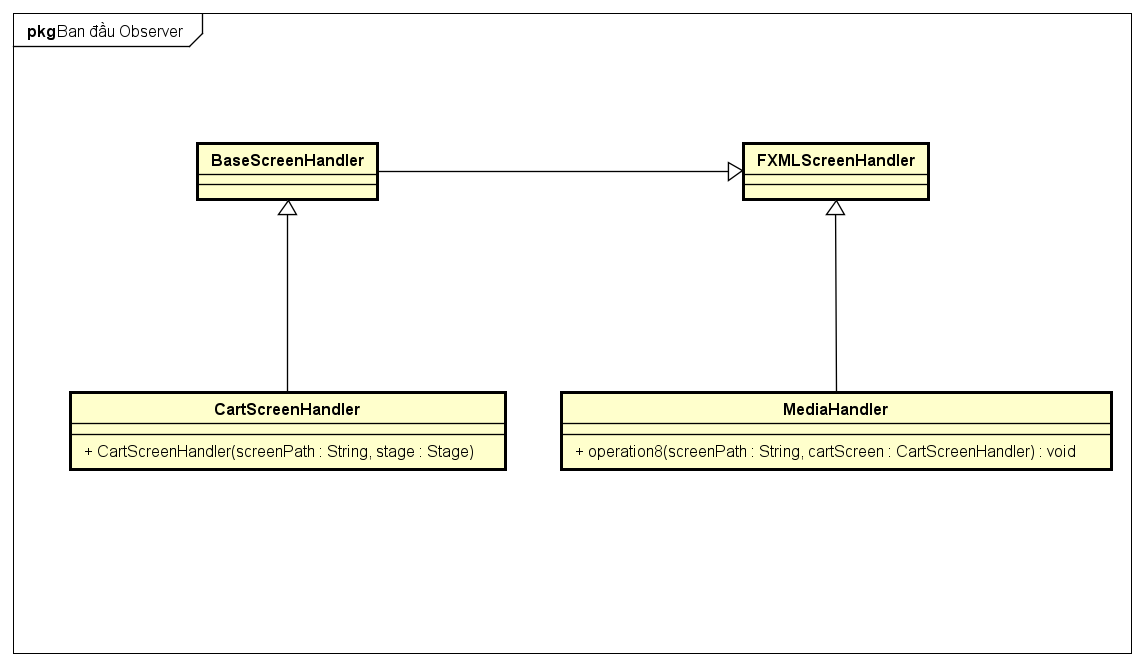
Với thiết kế cũ, khi muốn thay đổi thư viện, ta cần khởi tạo class DistanceCalculateAdapter cùng với thư viện AlternativeCalculator ở lớp DeliveryInfo.

Text

Description automatically generated

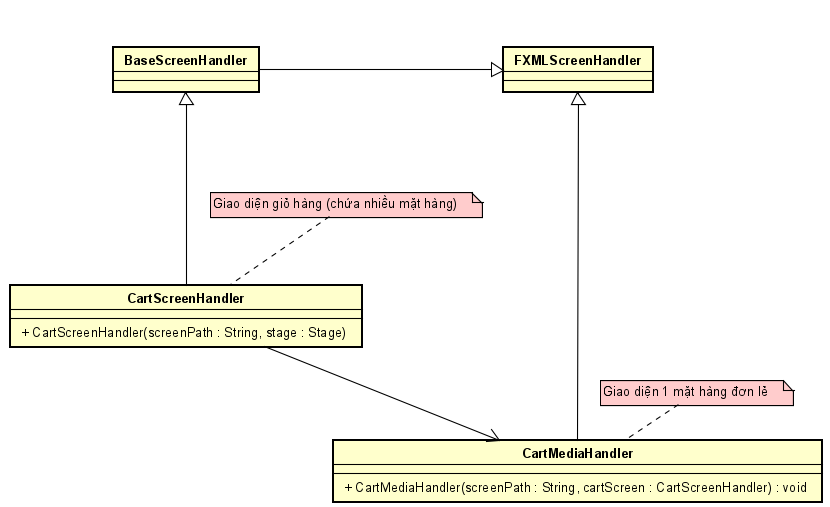
Hình 3.12 Sử dụng thư viện mới trong class PlaceOrderController

## Vấn đề “Hệ thống chưa xử lý khi người dùng thay đổi các mặt hàng trong giỏ hàng” và giải pháp



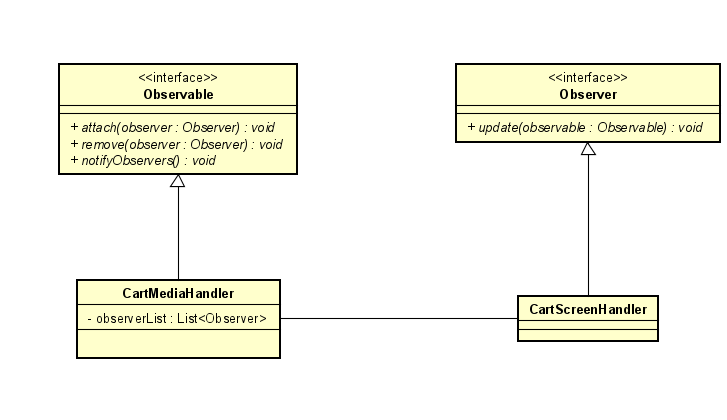
Hình 3.13 Biểu đồ lớp các lớp views

Với vấn đề như đã nêu, với tình huống nếu người dùng vô tình ấn nhầm vào nút xóa mặt hàng khỏi giỏ hàng, hệ thống chưa hề có cơ chế đáp ứng các thay đổi mặt hàng trong giỏ hàng.



Hình 3.14 Biểu đồ lớp các lớp view sau khi sửa

Với vấn đề này, từ màn hình giỏ hàng cần nhận biết được những sự thay đổi từ các mặt hàng đơn lẻ như các yêu cầu xóa, tăng, giảm của một mặt hàng. Để giải quyết, nhóm áp dụng Observer Pattern với thiết kế cụ thể như sau:



Hình 3.15 Biểu đồ lớp CartHandler với Observer Pattern

Với thiết kế này, mỗi khi một mặt hàng đơn lẻ có sự thay đổi như xóa, tăng hay giảm số lượng, class CartMediaHandler sẽ gọi đến hàm notifyObservers để thông báo về sự thay đổi này, khi đó CartScreenHandler (tương ứng với màn hình giỏ hàng) sẽ nhận biết có sự thay đổi và đưa ra các thông báo hoặc bất kì hành động nào tùy theo yêu cầu cài đặt

## Vấn đề “Cập nhật lại chức năng hủy đơn hàng” và giải pháp

Diagram

Description automatically generated Trong tương lai có thể phát sinh yêu cầu cho khách hàng có thể hủy đơn sau khi đã hoàn tất quá trình đặt hàng. Với yêu cầu này, hệ thống chưa có khả năng đáp ứng, cần phải bổ sung thay đổi rất nhiều trong codebase. Do vậy, để đáp ứng nếu có yêu cầu này trong tương lai, nhóm áp dụng State Pattern để thêm trạng thái cho các đơn hàng.

Hình . Biểu đồ sử dụng State Pattern để tạo chức năng Hủy đơn hàng

State Pattern giúp cho việc chuyển trạng thái có trình tự. Các thay đổi:

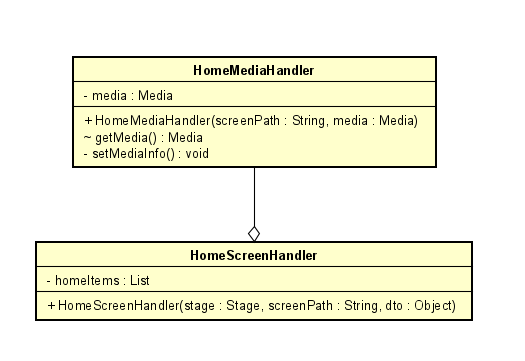
* Tạo interface InvoiceState
* Class Invoice: thay đổi trạng thái của đơn bằng hàm changeState(state)
* Tạo các lớp implements InvoiceState thể hiện các trạng thái: Đang chờ duyệt (CreateInvoiceState), Đã hủy đơn và Đã phê duyệt
* Thực hiện chuyển trạng thái có trình tự:
* CreateInvoiceState.cancelOrder(), CreateInvoiceState.confirmOrder()

## Vấn đề “Các hàm không liên quan đến nghiệp vụ của một số lớp” và giải pháp

Trong codebase vẫn tồn tại rất nhiều hàm xử lí không liên quan đến nghiệp vụ của lớp gây ra các lỗi Coincidental rất “ngớ ngẩn”. Vì vậy, nhóm đã bổ sung thêm các class CommonFunctionDate, CommonFunctionCurrency để chứa các method này. Đặc điểm chung của các method này là các hàm này đều có thể được sử dụng ở bất kì lớp nào do xử lí các vấn đề rất chung như: getToday (Lấy ra ngày hôm nay theo đúng định dạng), getCurrencyFormat (lấy ra định dạng tiền tệ),… Chúng được để ở dạng static trở thành các phương thức của lớp để có thể sử dụng dễ dàng ở bất kì đâu, giảm cohesion cho codebase.

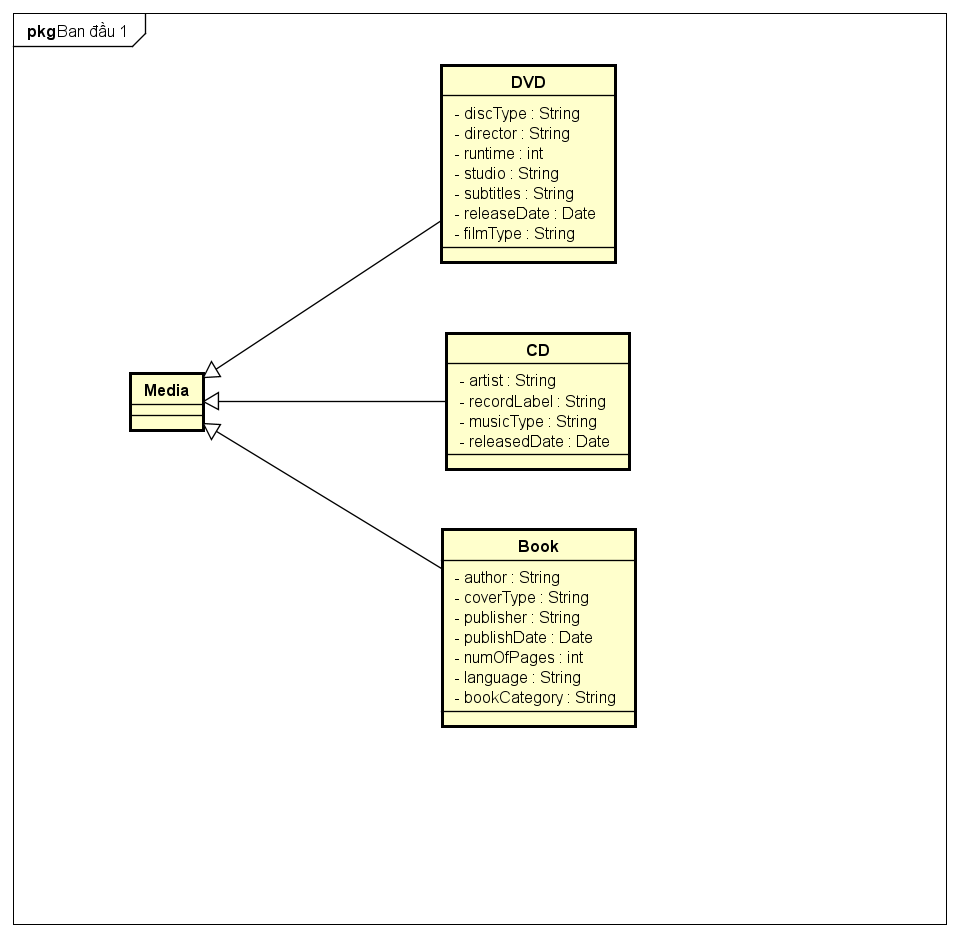
## Vấn đề “Thêm màn hình: Xem chi tiết sản phẩm” và giải pháp

Hiện tại, giao diện trang chủ bao gồm rất nhiều các sản phẩm được xử lý trong class HomeScreenHandler với mỗi 1 sản phẩm sẽ được xử lý riêng trong class HomeMediaHandler.



Hình 3.17 Biểu đồ lớp giao diện Home

Cụ thể, trong mỗi class HomeMediaHandler chứa 1 instance Media chứa thông tin của đối tượng hiển thị trên lớp đó ví dụ: CD, DVD, Book,…Các thông tin được hiển thị đối với các media đều là các thông tin chung có trong lớp Media.



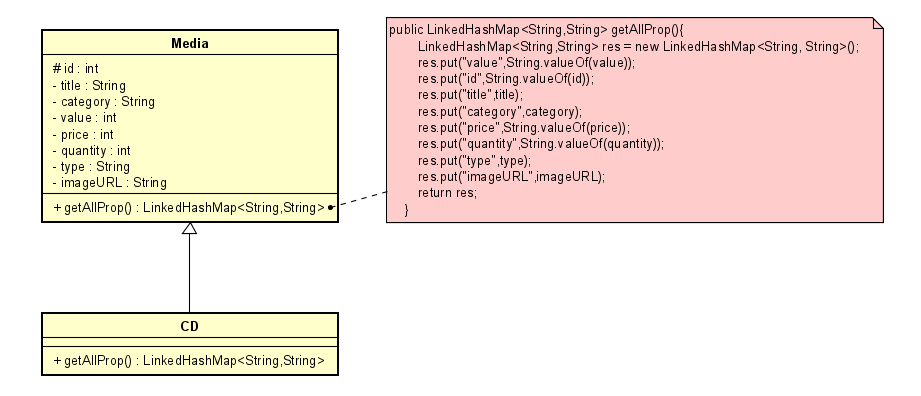
Hình 3.18 Biểu đồ lớp của lớp Media với các lớp con

Với yêu cầu thêm nút xem chi tiết đơn hàng, hiển thị đầy đủ thông tin của đối tượng, ta cần lấy được đầy đủ thông tin của đối tượng media thuộc lớp đó. Để giải quyết vấn đề này, cách đơn giản nhất là kiểm tra xem Media là instance của lớp nào bằng cách:



Hình 3.19 Mã nguồn vi phạm control coupling

Tuy nhiên, với cách xử lý như vậy sẽ gặp vấn đề control coupling khi phải kiểm tra tất cả loại media đang có. Mỗi khi có thêm media mới tiếp tục phải bổ sung code phần kiểm tra này. Để đáp ứng yêu cầu này, bổ sung thêm 1 phương thức trong class Media : getAllProp() sẽ trả lại thông tin chi tiết của mỗi đối tượng media.



Hình 3.20 Biểu đồ lớp Media với phương thức getAllProp

# Tổng kết

## Kết quả tổng quan

Với dự kiến ban đầu, kết quả đã thực hiện được hoàn thiện khoảng 90%. Dựa trên các kiến thức đã học về coupling, cohesion và các mẫu thiết kế, mã nguồn hiện tại đã đáp ứng được các yêu cầu thay đổi trong tương lai.

## Các vấn đề tồn đọng

Trong quá trình nghiên cứu và tìm hiểu các kiến thức học được trên lớp, nhóm đã cố gắng vận dụng để cải tiến mã nguồn của hệ thống. Tuy nhiên, các vấn đề được nêu trong báo cáo là các vấn đề mà nhóm đã nhìn ra và cố gắng sửa chữa vì vậy không tránh khỏi các thiếu sót và hoàn thiện tuyệt đối. Rất mong thầy cô và các bạn đóng góp ý kiến để hoàn thiện báo cáo này hơn nữa. Chúng em xin chân thành cảm ơn!

# Phân công công việc trong nhóm

Bảng 5.1 Bảng phân công công việc

|  |  |  |
| --- | --- | --- |
| Thành viên | MSSV | Công việc đảm nhận |
| Hoa Xuân Dương | 20173068 | Coupling, Cohesion, Clean code, SOLID, Các vấn đề khác, Đề xuất cải tiến 3.4, 3.6 |
| Phương Trung Đức | 20173030 | Coupling, Cohesion, Clean code, SOLID, Các vấn đề khác, Đề xuất cải tiến 3.1, 3.3, 3.5, 3.8 |
| Phạm Bá Đức | 20173036 | Coupling, Cohesion, Clean code, SOLID, Các vấn đề khác, Đề xuất cải tiến 3.2, 3.7 |
| Lê Minh Dương | 20173070 | Coupling |