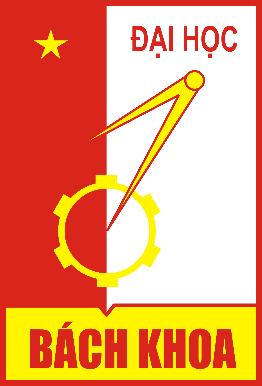
**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

----- □ 🕮 □ -----



**ĐỒ ÁN MÔN HỌC**

**MẪU THIẾT KẾ PHẦN MỀM**

**Giảng viên:** TS. Nguyễn Thị Thu Trang  
 TS. Bùi Thị Mai Anh

**Nhóm sinh viên thực hiện:**

Ong Thế Tùng 20204619

Ninh Thành Vinh 20200670

**Hà Nội, tháng 6 năm 2024**

**MỤC LỤC**

[1. Tổng quan 3](#_Toc169030700)

[1.1. Mục tiêu 3](#_Toc169030701)

[1.2. Phạm vi 4](#_Toc169030702)

[1.2.1. Mô tả khái quát phần mềm 4](#_Toc169030703)

[1.2.2. Các chức năng chính của phần mềm 4](#_Toc169030704)

[1.2.3. Cấu trúc mã nguồn 5](#_Toc169030705)

[1.2.4. Các yêu cầu cần cân nhắc thêm trong quá trình tái cấu trúc 6](#_Toc169030706)

[1.2.5. Các hoạt động review, refactor 7](#_Toc169030707)

[1.2.6. Kết quả dự kiến 7](#_Toc169030708)

[1.3. Danh sách thuật ngữ 7](#_Toc169030709)

[1.4. Danh sách tài liệu tham khảo 8](#_Toc169030710)

[2. Đánh giá thiết kế cũ 8](#_Toc169030711)

[2.1. Nhận xét chung 8](#_Toc169030712)

[2.2. Đánh giá các mức độ coupling và cohesion 9](#_Toc169030713)

[2.2.1. Mức độ coupling 9](#_Toc169030714)

[2.2.2. Mức độ cohesion 11](#_Toc169030715)

[2.3. Đánh giá việc tuân theo SOLID 12](#_Toc169030716)

[2.3.1. Single Responsibility Principle (SRP) 12](#_Toc169030717)

[2.3.2. Open Closed Principle (OCP) 13](#_Toc169030718)

[2.3.3. Liskov Substitution Principle (LSP) 14](#_Toc169030719)

[2.3.4. Interface Segregation Principle (ISP) 14](#_Toc169030720)

[2.3.5. Dependency Inversion Principle (DIP) 14](#_Toc169030721)

[2.4. Các vấn đề về Clean Code 15](#_Toc169030722)

[2.4.1. Clean name 15](#_Toc169030723)

[2.4.2. Clean function/method 15](#_Toc169030724)

[3. Đề xuất cải tiến 16](#_Toc169030725)

[3.1. Đề xuất khi phát sinh thêm một loại Media mới: AudioBook. 16](#_Toc169030726)

[3.2. Đề xuất khi thêm màn hình: Xem chi tiết sản phẩm 18](#_Toc169030727)

[3.3. Đề xuất khi thay đổi yêu cầu khi load giao diện 19](#_Toc169030728)

[3.4. Cải tiến vấn đề thay đổi phương thức tính khoảng cách sử dụng thư viện mới và thay đổi công thức tính phí vận chuyển 19](#_Toc169030729)

[3.5. Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card) 20](#_Toc169030730)

[3.6. Thay đổi công thức tính phí vận chuyển 22](#_Toc169030731)

[3.7. Kết quả tổng quan 23](#_Toc169030732)

[3.8. Các vấn đề còn tồn đọng 23](#_Toc169030733)

**Phân chia công việc**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thành viên** | **Công việc** |
| 1 | Ong Thế Tùng, Ninh Thành Vinh | Cohesion/Coupling |
| 2 | Ong Thế Tùng, Ninh Thành Vinh | Clean Code |
| 3 | Ong Thế Tùng, Ninh Thành Vinh | SOLID (tìm vấn đề) |
| 4 | Ong Thế Tùng, Ninh Thành Vinh | SOLID(giải pháp áp dụng Design Pattern) |
| 5 | Ong Thế Tùng, Ninh Thành Vinh | Implement code |
| 6 | Ong Thế Tùng, Ninh Thành Vinh | Review code |

## Tổng quan

### Mục tiêu

Tài liệu này được sử dụng làm báo cáo môn học Mẫu thiết kế phần mềm - IT4536 của Trường Công nghệ thông tin và truyền thông – Đại học Bách Khoa Hà Nội.

Đối tượng người đọc hướng đến những lập trình viên, những người đã có kiến thức cơ sở về lập trình hướng đối tượng, người thiết kế và xây dựng hệ thống phần mềm hoặc những người muốn tham khảo tài liệu để viết hướng dẫn nghiệp vụ.

Tài liệu này là kết quả làm việc của nhóm tác giả, nhằm tổng kết quá trình thực hiện và kết quả việc tái cấu trúc lại mã nguồn dựa trên một hệ thống phần mềm có sẵn được cung cấp bởi giảng viên giảng dạy môn học. Nhóm tác giả đã sử dụng các kiến thức về Cohesion, Coupling, SOLID, và các Design Pattern phổ biến được giảng dạy trong môn học để áp dụng vào phân tích và cải tiến thiết kế hiện tại.

### Phạm vi

#### Mô tả khái quát phần mềm

Phần mềm là một cửa hàng online giúp người dùng đặt hàng các sản phẩm đa phương tiện như sách, đĩa CD, đĩa DVD từ cửa hàng. Sau khi thực hiện đặt hàng, đơn hàng sẽ được thanh toán online qua Credit Card (Thẻ tín dụng) của người dùng và sau đó sản phẩm sẽ được vận chuyển tới địa chỉ mà khách hàng yêu cầu.

Về đánh giá sơ bộ, phần mềm đã đáp ứng được các chức năng cơ bản tại thời điểm hiện tại, có thể đưa vào sử dụng và vận hành. Tuy nhiên do chưa đảm bảo việc tuân thủ các nguyên lý SOLID cũng như các yêu cầu về thiết kế phần mềm nên phần mềm sẽ không có khả năng đáp ứng được một số yêu cầu thay đổi có thể phát sinh trong tương lai. Do vậy việc tái cấu trúc lại code phần mềm là việc cần thiết.

#### Các chức năng chính của phần mềm

Phần mềm AIMS có một số tính năng chính như sau:

* Khách hàng
  + Xem danh sách sản phẩm
  + Tìm kiếm sản phẩm: khách hàng dùng các đặc tính của sản phẩm để tìm kiếm
  + Quản lý giỏ hàng: khách hàng có thể xem, sửa hoặc bỏ sản phẩm khỏi giỏ hàng
  + Đặt hàng: khách hàng có thể đặt hàng trong giỏ hàng
  + Thanh toán: khách hàng chọn phương thức thanh toán và thanh toán đơn hàng
  + Hủy đơn hàng: khách có thể hủy đơn hàng đã đặt
* Quản trị viên
  + Quản lý sản phẩm: có thể thêm, xem, xóa sửa bất kỳ sản phẩm nào
  + Quản lý người dùng: chặn/bỏ chặn người dùng, tạo người dùng mới, xem cập nhật và xóa người dùng
  + Quản lý đơn hàng: xem đơn hàng, phê duyệt đơn hàng

A diagram of a person's diagram

Description automatically generated

**Hình 1:** Use case Diagram tổng quan

Phần mềm có các tác nhân: User, Admin và InterBank. User dù không đăng nhập vẫn có thể thực hiện việc đặt hàng.

User có thể thực hiện việc Xem danh sách sản phẩm, Thêm sản phẩm vào giỏ hàng, , sửa giỏ hàng, Đặt hàng, Tìm kiếm. Trong khi thực hiện đặt hàng, sẽ có nghiệp Thanh toán đơn hàng. Nghiệp vụ này có sự tham gia của tác nhân InterBank, là một tác nhân hệ thống, đại diện cho hệ thống thanh toán Online.

Admin có thể thực hiện quản lí người dùng, quản lí sản phẩm, quản lí đơn hàng.

#### Cấu trúc mã nguồn

Mã nguồn của phần mềm được thiết kế rõ ràng theo mô hình MVC, gồm ba tầng là Model, View và Controller.

Trong mô hình MVC, tầng View đảm nhận vai trò xử lý logic liên quan đến giao diện người dùng, bao gồm việc hiển thị các màn hình, lấy dữ liệu từ người dùng và chuyển tiếp cho tầng Controller, cũng như điều hướng giữa các màn hình. Tầng Controller xử lý logic liên quan đến luồng dữ liệu, bao gồm xác thực dữ liệu người dùng và tương tác với các hệ thống con để thực hiện thanh toán đơn hàng của người dùng, ở tầng này bao gồm cả Subsystem bên trong. Tầng Model chịu trách nhiệm lưu trữ dữ liệu của hệ thống và bao gồm hai thành phần chính là Data Access Object và Entity. Ngoài ra, hệ thống còn có một thành phần chung gọi là Common, bao gồm các Interfaces, Exceptions và Utils, được sử dụng chung trong toàn bộ hệ thống.

#### Các yêu cầu cần cân nhắc thêm trong quá trình tái cấu trúc

Việc tái cấu trúc mã nguồn của phần mềm với mục tiêu giúp phần mềm có khả năng đáp ứng với các thay đổi trong tương lai và giúp cho code “Clean” giúp cho các lập trình dễ dàng làm việc trên mã nguồn.

Việc tái cấu trúc đảm bảo tuân thủ SOLID, đặc biệt là OCP (Open-closed Principle) và DIP (Dependency Inversion Principle). Quá trình tái cấu trúc phần mềm để đáp ứng SOLID có thể bao gồm việc chia nhỏ, tái cấu trúc và tái thiết kế các module, lớp, và giao diện. Nó yêu cầu một quá trình phân tích kỹ lưỡng, đánh giá và thực hiện các thay đổi một cách cẩn thận để đảm bảo tính đúng đắn và hiệu quả của hệ thống.

Trong quá trình tái cấu trúc, yêu cầu không được thay đổi về mặc chức năng và phi chức năng của hệ thống ban đầu.

#### Các hoạt động review, refactor

Việc review với mã nguồn được thực hiện như sau:

* Xác định các mức độ coupling và cohesion
* Xác định các vi phạm nguyên lý SOLID
* Xác định các vấn đề về clean code

Việc refactor mã nguồn được thực hiện như sau:

* Tái cấu trúc các variable, method và class để đảm bảo clean code
* Tái cấu trúc những đoạn vi phạm nguyên lý SOLID
* Áp dụng các Design Pattern đã học vào hệ thống một cách có chọn lọc và phù hợp.

#### Kết quả dự kiến

Sau khi tái cấu trúc mã nguồn, codebase sẽ đảm bảo được các nguyên lý về SOLID, được áp dụng các Design Pattern phù hợp và đảm bảo clean code để dễ đọc dễ hiểu cho các lập trình viên sau này.

Từ đó, phần mềm sẽ có khả năng nâng cấp, mở rộng tốt hơn trong tương lai.

### Danh sách thuật ngữ

|  |  |  |
| --- | --- | --- |
| STT | Thuật ngữ | Giải thích |
| 1 | coupling | Mức độ liên kết giữa các class, các thành phần trong hệ thống |
| 2 | cohesion | Mức độ liên kết chi tiết trong module, class |
| 3 | Method | Phương thức của một lớp |
| 4 | SOLID | Nguyên tắc thiết kế nhằm làm cho các thiết kế hướng đối tượng trở nên dễ hiểu, linh hoạt và dễ bảo trì hơn |
| 5 | Refactor | Tái cấu trúc mã nguồn |
| 6 | Data Access Object (DAO) | Cung cấp giao diện trừu tượng cho một số loại cơ sở dữ liệu hoặc cơ chế lưu giữ khác. Bằng cách ánh xạ các cuộc gọi ứng dụng tới lớp kiên trì, DAO cung cấp các hoạt động dữ liệu mà không để lộ chi tiết cơ sở dữ liệu.. |
| 7 | Entity | Thực thể lưu trữ dữ liệu của hệ thống |
| 8 | Design Pattern | Mẫu thiết kế phần mềm - Những mô hình hoặc mẫu quy ước được phát triển để giải quyết các vấn đề chung trong thiết kế phần mềm. |
| 9 | Module | Các thành phần trong hệ thống phục vụ một mục đích chung |

### Danh sách tài liệu tham khảo

* Tài liệu, vở ghi trên lớp môn học Mẫu thiết kế phần mềm IT4536 - Trường Công nghệ Thông tin và Truyền thông - Đại học Bách Khoa Hà Nội.
* Các tài liệu online khác

## Đánh giá thiết kế cũ

### Nhận xét chung

Hệ thống hiện tại đã đảm bảo được luồng hoạt động chính. Tuy nhiên khi nhìn vào cách tổ chức hệ thống, ta sẽ có thể phát hiện ra nhiều vấn đề mà khi có các yêu cầu phát sinh hệ thống sẽ khó thích ứng: Các modules phụ thuộc trực tiếp vào nhau thay vì phụ thuộc ở mức trừu tượng. Điều này sẽ khiến khi thay đổi một module, các module có liên quan cũng phải được thay đổi theo. Thêm vào đó là các vấn đề khác như dư thừa dữ liệu, vấn đề clean code. Do đó việc tái cấu trúc và sửa đổi là cần thiết: Loại bỏ sự phụ thuộc trực tiếp mà thay vào đó là phụ thuộc trừu tượng, clean code, loại bỏ dữ liệu không cần thiết, … Từ đó hệ thống sẽ dễ bảo trì vào nâng cấp hơn

### Đánh giá các mức độ coupling và cohesion

Trong lĩnh vực thiết kế hệ thống phần mềm, mức độ coupling của một thiết kế được xác định bằng sự phụ thuộc, gắn kết giữa hai hay nhiều classes, modules. Một thiết kế được coi là low coupling khi một class, module này bị thay đổi, sẽ không gây nhiều ảnh hưởng tới các class, module khác mà nó liên kết tới. Ngược lại, một thiết kế được coi là high coupling khi các class, module bị kết dính quá chặt chẽ với nhau, một thay đổi của thành phần này gây ra nhiều sự thay đổi của các thành phần liên quan tới nó, dẫn tới việc hệ thống rất khó thay đổi và bảo trì.

Không giống với cohesion, mức độ cohesion của một class, module, package được xác định bằng độ liên kết chặt chẽ của các thành phần bên trong một class, module, package đó. Một thiết kế được coi là low cohesion khi module đó thực hiện quá nhiều công việc, mà không tập trung vào một công việc cụ thể nào, ví dụ như các package Utils trong thực tế thường được sử dụng để chứa nhiều hàm tiện ích của hệ thống mà có ít liên quan với nhau. Và ngược lại, một thiết kế được coi là high cohesion khi class, module, package chỉ tập trung vào việc thực hiện một nhiệm vụ, mục tiêu cụ thể và duy nhất.

Tổng kết lại, một thiết kế phần mềm tốt là một thiết kế có mức độ cohesion cao và coupling thấp (high cohesion và low coupling).

#### Mức độ coupling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Các mức độ về Coupling | Module | Mô tả | Lý do |
| 1 | Content coupling | entity\shipping\DeliveryInfo | access modifier là protected | Khi để access modifier là protected, các lớp con cùng package và lớp con có thể truy xuất data trực tiếp, vừa vi phạm content coupling, vừa vi phạm nguyên lý encapsulation. |
| 2 | Common coupling | views\screen\ViewsConfig | Các biến PERCENT\_VAT và REGULAR\_FONT là các biến dùng chúng cho hệ thống nhưng không được khai báo final | Khi khai báo biến static để dùng chung cho cả hệ thống, biến nên được khai báo là hằng số final để tránh việc bị thay đổi khi hệ thống vận hành |
| SessionInformation | Nhiều lớp có tham gia vào việc dùng và sửa đổi dữ liệu này | Việc nhiều lớp có thể thay đổi dữ liệu này khiến khó để quản lý trạng thái dữ liệu |
| 3 | Control coupling |  |  |  |
| 4 | Stamp coupling | entity\cart\Cart | Tham số của method checkMediaInCart đang có kiểu dữ liệu là Media | implement của hàm chỉ cần duy nhất trường id của Media, dẫn đến dư thừa dữ liệu |
| views\screen\  intro\  IntroScreenHandler.java | Tham số của method setupData đang có kiểu dữ liệu là Object | Hàm chưa được implement dẫn đến việc truyền dữ liệu vào hàm là dư thừa |
| views\screen\  home\  LoginScreenHandler | method backToHomeScreen truyền vào MouseEvent | Không dùng đến nên dư thừa |
| dao\media\MediaDAO | Tham số truyền vào method updateMediaFieldById có tbname | Trong implement của hàm không dùng tới nên dư thừa |
| entity\cart\CartItem | Hàm khởi tạo CartItem có tham số truyền vào là Cart | implement của hàm không cần nên dư thừa |
| entity\cart\Cart | Tham số truyền vào method checkMediaInCart là Media | implement của hàm này chỉ cần đến id của media đó nên dư thừa các dữ liệu còn lại |
| entity\shipping\DeliveryInfo.java | CalculateShippingFee có tham số truyền vào là Order | implement của hàm không cần nên dư thừa |
| 5 | Data coupling |  |  |  |

#### Mức độ cohesion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Các mức độ về Cohesion | Module | Mô tả | Lý do |
| 1 | Coincidental cohesion | subsystem\interbank\InterbankPayloadConverter | getToday() không cùng mục đích với class cha | Tách thành 1 helper method riêng |
| 2 | Logical cohesion | utils\ApplicationProgrammingInterface | các methods post và get liên quan logic với nhau | các methods post và get trong module chỉ liên kết với nhau về logic gọi API ngoài hệ thống chứ không liên kết về chức năng |
|  |  | views\screen\ PopupScreen | success(), error(), loading() liên quan logic với nhau | Các method thực hiện các thao tác liên quan đến hiển thị popup liên quan logic với nhau chứ không liên kết về chức năng |
| 3 | Temporal cohesion | controller\AuthenticationController | md5() không liên quan đến class chỉ thực hiện theo thứ tự thời gian bởi việc thực hiện login() sử dụng md5() | hàm md5 là hàm sinh ra digest, không phục vụ mục đích của lớp là xác thực người dùng, nên tách thành 1 hàm helper riêng |
| controller\PaymentController | getExpirationDate không liên quan đến class chỉ thực hiện theo thứ tự thời gian bởi việc thực hiện payOrder() sử dụng | Nên tách thành 1 hàm helper riêng |
|  |  | controller\ PlaceOrderController | validateDeliveryInfo, validatePhoneNumber, validateName, validateAddress chỉ thực hiện theo thứ tự thời gian bởi các method của lớp | Nên tách tiêng các hàm vào class validator riêng |
| 4 | Procedure cohesion |  |  |  |
| 5 | Communicational cohesion |  |  |  |
| 6 | Sequential cohesion |  |  |  |
| 7 | Functional cohesion |  |  |  |

### Đánh giá việc tuân theo SOLID

#### Single Responsibility Principle (SRP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | entity.shipping.  DeliveryInfo | Class DeliveryInfo đang có chứa một method dùng để tính toán chi phí vận chuyển. | DeliveryInfo đang thực hiện nhiều hơn 1 nhiệm vụ đó là cung cấp thông tin giao hàng và tính phí ship. Có xem xét tách phần tính phí ship ra thành class khác |
| 2 | controller.  AuthenticationController | Class AuthenticationController có method md5, | Class AuthenticationController có nhiệm vụ xử lý logic các yêu cầu authenticate của người dùng, vì vậy đang thực hiện nhiều hơn 1 nhiệm vụ đó là xác thực người dùng, quản lý session và mã hóa dữ liệu Có xem xét đưa md5() vào helper, và AuthenticationController gồm: isAnonymousSession(), login(), logout() |
| 3 | controller.  PlaceOrderController | Class PlaceOrderController vi phạm SRP do có nhiều hơn một vai trò (Cụ thể là vừa validate trường, vừa đặt hàng, tạo hóa đơn). | Class PlaceOrderController có nhiệm vụ xử lý yêu cầu đặt hàng, tạo hóa đơn của người dùng, việc validate không phải nhiệm vụ của nó.  Hãy tách hàm validate ra một class riêng và truyền vào PlaceOrderController. |
|  | controller.  PaymentController | PaymentController có method, getExpirationDate() xử lí thời gian thẻ, và emptyCart xử lí làm rỗng Cart | PaymentController đang thực hiện nhiều hơn 1 nhiệm vụ đó là thanh toán và quản lý giỏ hàng và lấy ngày hết hạn. Xem xét đưa xử lí giỏ hàng vào phần module liên quan đén giỏ hàng, getExpirationDate() đẩy vào helper |

#### Open Closed Principle (OCP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | * entity.payment.   PaymentTransaction   * entity.payment.   CreditCard   * controller.   PaymentController   * entity.shipping.   DeliveryInfo   * subsystem.InterbankInterface * subsystem.interbanl. InterbankPayloadConverter | Các class, interface bên đang có một phụ thuộc trực tiếp vào CreditCard | Có thể trong tương lai tương lai có mở rộng thêm phương thức thanh toán mới dẫn tới việc sẽ phải sửa đổi nhièu |
| 2 | entity.shipping.  DeleveryInfo | Class này đang phụ thuộc trực tiếp vào class DistanceCalculator | Do class đang phụ thuộc vào một thư viện tính toán khoảng cách bên ngoài DistanceCalculator. Việc này có nguy cơ nếu muốn thêm các cách tính khoảng cách khác thì bắt buộc phải thay đổi code. |

#### Liskov Substitution Principle (LSP)

Hệ thống hiện tại không vi phạm Liskov Substitution Principle (LSP).

#### Interface Segregation Principle (ISP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | * views.screen. BaseScreenHandler | BaseScreenHandler có method setupData, setupFunctionality | Một só class con không sử dụng đến 2 method này mà vẫn phải extent nó ví dụ PopupScreen |

#### Dependency Inversion Principle (DIP)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | * entity.payment.   PaymentTransaction   * entity.payment.   CreditCard   * controller.   PaymentController   * entity.shipping.   DeliveryInfo   * subsystem.InterbankInterface * subsystem.interbank. InterbankPayloadConverter | Đang phụ thuộc trực tiếp vào class CreditCard. | Trong tương lai, hệ thống khả năng cao sẽ mở rộng các phương thức giao dịch, thanh toán mới ví dụ như DomesticCard nên việc sử dụng và phụ thuộc vào CreditCard khiến cho việc mở rộng trở nên rất khó khăn. |
|  | * entity.shipping.   DeliveryInfo | Đang phụ thuộc trực tiếp vào thư viện khoảng cách. | Ảnh hưởng đến việc mở rộng nếu có thêm cách tính phí ship hay thư viện tính toán khác |

### Các vấn đề về Clean Code

#### Clean name

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Vị trí có tên chưa clear | Đề xuất sửa đổi |
| 1 | controller.PaymentController | Method getExpirationDate có tên biến strs | Sửa tên biến strs thành dateParts để thể hiện các phần trong date |
| 2 | controller.PlaceOrderController | Các method validate đang truyền trực tiếp giá trị các hằng số | Chuyển hàng số thành biến final có ý nghĩa |
| 3 | utils.ApplicationProgrammingInterface | Method get có tên biến BufferedReader là in | Nên đặt tên thành “bufferedReader” |
| 4 | views.screen.home.HomeScreenHandler | Method setupData có tên biến medium thể hiện chưa rõ ý nghĩa mà nó thực hiện | Đổi tên biến medium thành mediaList thể hiện danh sách các đối tượng media |

#### Clean function/method

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Vị trí có tên chưa clear | Đề xuất sửa đổi |
| 1 | controller.AuthenticationController | Phương thức md5 | Thay đổi tên phương thức md5 thành encryptMD5 để thể hiện nhiệm vụ của hàm này là sinh một message-digest bằng cách dùng thuật toán md5. |
| 4 | controller.ViewCartController | Phương thức getCartSubtotal | Bỏ việc đặt tên cho biến thay vào đó return luôn giá tra do hàm chỉ có 1 phép tính toán, chuyển về dạng inline-function |
| 5 | views.screen.intro.IntroScreenHandler | Phương thức setupFunctionality dùng hard-code cho đường dẫn dẫn logo đến khó thay đổi sau này | Chuyển giá trị đường dẫn tới logo của ứng dụng vào class ViewsConfig đặt tên là LOGO\_PATH và gán vào phương thức setupFunctionality |

## Đề xuất cải tiến

### Đề xuất khi phát sinh thêm một loại Media mới: AudioBook.

AudioBook kế thừa các đặc tính của Book.

### Đề xuất khi thêm màn hình: Xem chi tiết sản phẩm

Màn hình xem chi tiết màn hình là một màn hình nhỏ được gọi ra khi event drag chuột vào mỗi sản phẩm. Màn hình xem chi tiết sản phẩm gồm 2 phần, một phần chứa các thuộc tính cơ bản của Media, một phần chứa các thuộc tính đặc biệt theo từng lớp con của Media – Details. Tại script xử lý màn hình UI của xem chi tiết Media chứa VBOX chứa DetailsScreen, sử dụng Strategy để set media cho Details.

### Đề xuất khi thay đổi yêu cầu khi load giao diện

Áp dụng Template method: Đưa đoạn code bị lặp lên phương thức khởi tạo của lớp cha BaseScreenHandler. Lớp BaseScreenHandler có các phương thức setupData và setupFunctionality và các lớp con sẽ ghi đè các phương thức này.

Vấn đề vi phạm nguyên lý OCP khi có thay đổi: áp dụng Strategy Pattern. Lớp BaseScreenHandler có thêm thuộc tính errorNotify là đối tượng của lớp abstract ErrorNotify. Lớp này có phương thức notify thực hiện thông báo lỗi. Việc thông báo lỗi khi khởi tạo các màn hình được thực hiện bằng cách gọi errorNotify.notify(). Khi cần thay đổi cách thức thông báo lỗi, chỉ cần thay đổi thuộc tính errorNotify bằng setErrorNotify.

A diagram of a computer

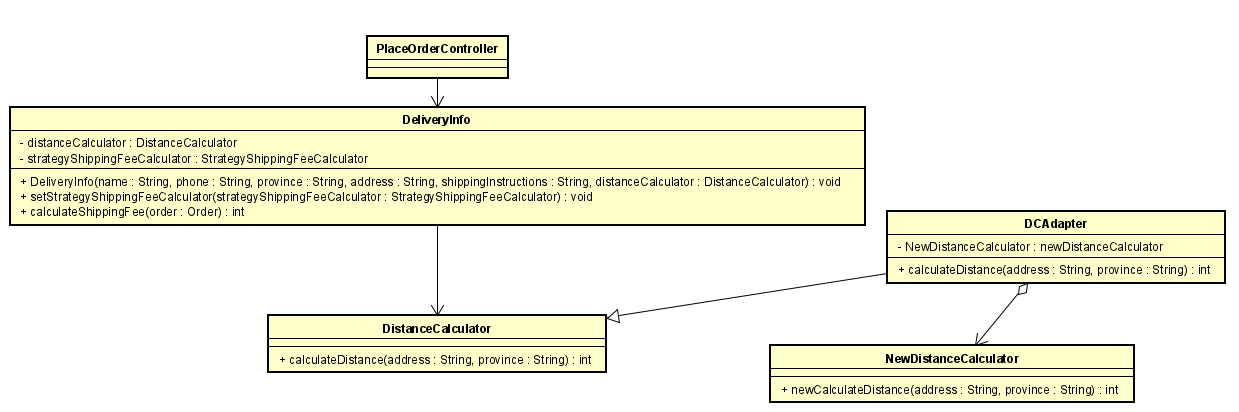
Description automatically generated with medium confidence

**Hình 1** Giải quyết vấn đề thay đổi thư viện tính khoảng cách và cách tính khoảng cách

### Cải tiến vấn đề thay đổi phương thức tính khoảng cách sử dụng thư viện mới và thay đổi công thức tính phí vận chuyển

Class DeliveryInfo hiện tại đang có một thuộc tính có kiểu dữ liệu là DistanceCalculator (thuộc thư viện Distance-API). Nếu như ta muốn thay đổi thư viện tính khoảng cách mới, ta phải sửa trực tiếp class DeliveryInfo.

Từ đó ta sẽ thiết kế lại, bằng cách sử dụng Adapter Pattern, tạo một interface cho phương thức tính khoảng cách, và class nào muốn sử dụng sẽ thực hiện việc implement interface này. Việc tách thành interface riêng sẽ giúp chúng ta khi có yêu cầu mới sẽ chỉ cần tạo một class implement interface này, các class đã implement từ trước sẽ không bị ảnh hưởng



**Hình 2** Giải quyết vấn đề thay đổi thư viện tính khoảng cách và cách tính khoảng cách

### Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)

Trong thiết kế cũ, lớp PaymentController có nhiệm vụ thực hiện việc xử lý thanh toán, và nó phụ thuộc trực tiếp vào thuộc tính card của lớp CreditCart cũng như ta đang cố định một phương thức thanh toán trong này. Thêm vào đó các lớp khác như ở trong pakage subsystem, class PaymentTransaction và InterbankPayloadConverter cũng phụ thuộc trực tiếp vào CreditCard. Như vậy rõ ràng trong tương lai nếu có thêm một loại thẻ mới, một phương thức thanh toán mới và Card mới này còn có cả các thông tin khác Card cũ thì ta sẽ sửa trực tiếp code trong những class trên. Như vậy đang có hai vấn đề liên quan tới loại card và phương thức thanh toán. Dưới đây là thiết kế cũ:

A screenshot of a computer screen

Description automatically generated

**Hình 3** Mã nguồn phải sửa code khi thay đổi phương thức thanh toán và loại thẻ thanh toán

Để khắc phục điều này ta làm như sau:

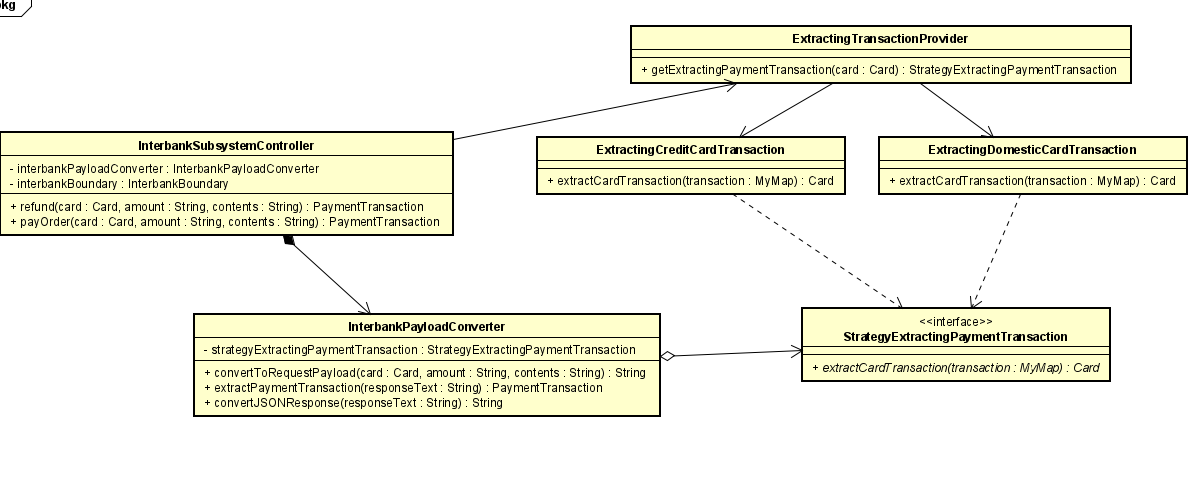
* Sử dụng design pattern Factory method thực hiện tạo loại thẻ mới dựa vào typeCard và chấp nhận bị control coupling cardFactoryProvider xử lí trả về CardFactory ( nhận dữ liệu là thông tin về CardDTO do mỗi loại card có các trường là khác nhau) – CardFactory xử lí trả về loại Card tương ứng được implement từ interface Card .

A diagram of a computer

Description automatically generated

**Hình 4** Mã nguồn sau khi sửa code khi thay đổi phương thức thanh toán và loại thẻ thanh toán

* Ngoài ra hiện tại việc xử lí giá trị trả về mà chỉ dựa vào chuỗi json do API trả về và phụ thuộc trực tiếp vào CreditCard nên cần giải pháp xử lí json này với tất cả các loại Card và xử lí loại Card nào sẽ cần phụ thuộc InterbankSubsystemController. Ở thiết kế này chấp nhận control coupling



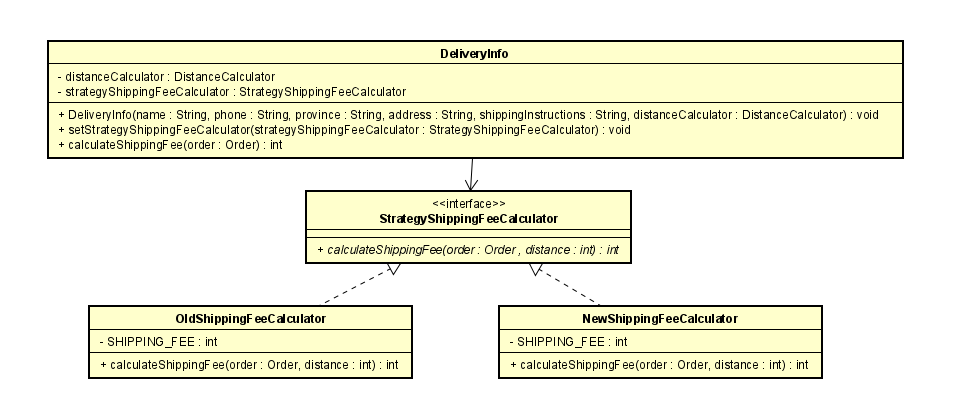
**Hình 5** Giải quyết vấn đề phát sinh xử lí dữ liệu trả về

### Thay đổi công thức tính phí vận chuyển

Class DeliveryInfo hiện tại đang tính phí ship dựa vào khoảng cách và giá trị 1.2. Nếu như ta muốn thay đổi công thức tính phí vận chuyển, ta phải sửa trực tiếp class DeliveryInfo.

Việc tính phí đang được thực hiện ở class DeliveryInfo. Khi có yêu cầu mới là tính phí dựa trên khối lượng, độ cồng kềnh và khoảng cách sẽ có vấn đề phát sinh, vì nó đang dựa vào Order.

Từ đó ta sẽ thiết kế lại, bằng cách sử dụng Strategy Pattern, tạo một interface cho phương thức tính khoảng cách dựa vào Order và khoảng cách được tính toán trước đó, và class nào muốn sử dụng sẽ thực hiện việc implement interface này. Việc tách thành interface riêng sẽ giúp chúng ta khi có yêu cầu mới sẽ chỉ cần tạo một class implement interface này, các class đã implement từ trước sẽ không bị ảnh hưởng.



**Hình 6** Giải quyết vấn đề thay đổi cách tính chi phí

Hiện tại, để tính chi phí chỉ cần biết khoảng điểm, nhưng tham số của hàm calculate trong interface ShippingFeeCalculatorStrategy được thiết kế với tham số là Order. Chúng ta chấp nhận mức vi phạm Stamp Coupling ở hệ thống hiện tại sau khi cải tiến, do trong công thức tính chi phí mới phải cần các thông tin khác trong Order để tính toán, nếu chỉ sử dụng hai tham số là điểm giao và nhận thì sẽ không tái sử dụng được interface này trong tương lai.

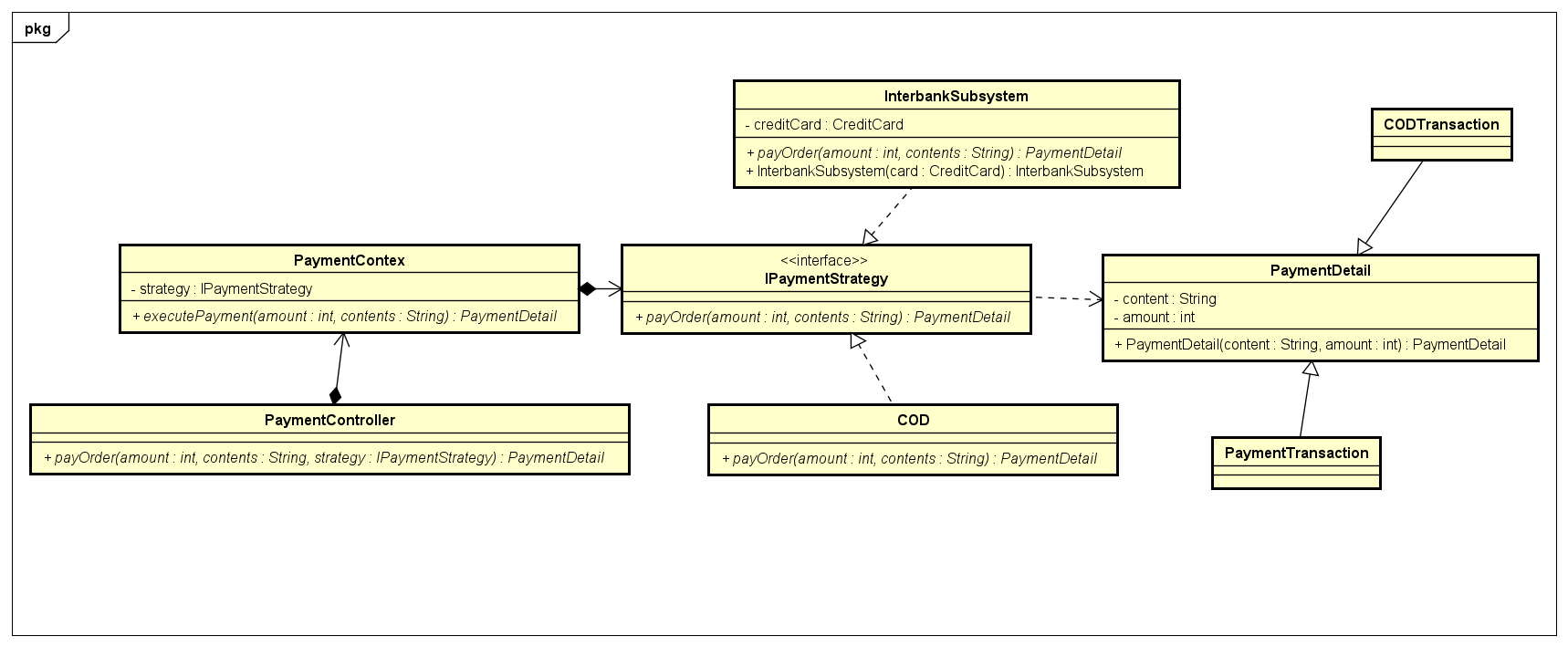
Khi yêu cầu phát sinh xảy ra, chúng ta chỉ cần cài đặt thêm class NewShippingFeeCalculator tương ứng với thay đổi về việc thay đổi công thức tính toán.

### Mở rộng phương thức thanh toán với Strategy Pattern

Ở mã nguồn gốc, phương thức thanh toán mặc định là credit card với tham số đầu vào là 1 card. Từ đấy hạn chế các phương thức thanh toán khác, đặc biệt là các phương thức thanh toán không sử dụng tới thẻ như COD hay QR.

Trên cơ sở đó, ta thiết kế lại dựa trên Strategy Pattern bằng cách tạo một *interface* IPaymentStrategy, các subsystem implement interface này. Ta giảm bớt tham số card ở phương thức payOrder tại PaymentController, thêm thuộc tính này tại interbankSubsystem thông qua setStrategy tại PaymentContext.

Từ đó, đối với những phương thức không yêu cầu card, ta có thể dễ dàng mở rộng hệ thống. Đối với PaymentTransaction, ta thành lập 1 class cha PaymentDetail để PaymentTransaction kế thừa nó. Khi gọi tới payOrder sẽ trả về PaymentDetail, đối với mỗi subsystem hiện tại và tương lai thêm vào, tạo thêm các class con của PaymentDetail có các cách xử lý phù hợp đối với từng loại phương thức được thêm vào.



**Hình 7**

Các phương thức khác thêm vào sẽ được thêm vào các tùy chọn để set tại các class handle UI, đảm bảo tránh được control coupling.

### Vấn đề quản lí giỏ hàng

Với mã nguồn gốc, SessionInformation quản lí giở hàng và mỗi lần truy cập đến giở hàng đều khởi tạo 1 instance của giỏ hàng. Đây là vấn đề liên quan đến tính đồng nhất do chỉ cần 1 giỏ hàng khi chương trình chạy và quản lí sự thống nhất cuả giỏ hàng trên toàn bộ chương trình.

Nhóm sử dụng Design pattern : Singleton cho phép chỉ tạo 1 instance của Cart trong suốt chương trình và dễ dang quản lí.

A screen shot of a computer code

Description automatically generated

**Hình 8** Mã nguồn sử dụng Singleton quản lí Cart

Bây giờ khi chương trình chạy chỉ khởi tạo instance của Cart đúng 1 lần, và có thể truy cập vào giỏ hàng bằng getInstance().

### Kết quả tổng quan

Sau khi hoàn thành bài tập lớn lần này, dự án cửa hàng online AIMS đã có thiết kế tốt hơn. Mã nguồn về cơ bản đã đáp ứng được các nguyên lý SOLID, từ đây trong tương lai nếu có những yêu cầu mới hay cần sửa đổi mã nguồn thì những việc phát sinh đó sẽ được xử lý dễ dàng hơn trước. Thêm vào đó các vấn đề clean code như clean name, clean method, clean class cũng đã được khắc phục. Do đã sửa đổi theo nguyên lý SOLID mà các vấn đề coupling cohesion trước đó cũng không còn nữa.

Qua việc làm bài tập lớn lần này đã giúp các thành viên trong nhóm cải thiện kiến thức một cách rõ rệt, mỗi thành viên đều đã biết vận dụng kiến thức trong việc tái cấu trúc mã nguồn, thiết kế mã nguồn làm sao cho hợp lý và hiệu quả nhất.

### Các vấn đề còn tồn đọng

Tuy đã có sửa đổi và bổ sung thêm các tính năng, nhưng việc thiếu sót vẫn có thể xảy ra. Nhóm cũng đã nhận thấy các vấn đề như: Khi tái cấu trúc để đáp ứng yêu cầu đổi tình phí vận chuyển, nhóm đã xây dựng hàm tính phí theo khối lượng có tham số đầu vào là order, nhưng trong thân hàm không sử dụng tới. Việc này dẫn tới vấn đề bị Stamp Coupling, do một Order có các thuộc tính không cần thiết trong việc tính toán chi phí vận chuyển, ví dụ như thuế (tax), phụ phí (subtotal). Tuy nhiên ở đây nhóm vẫn chấp nhận bị coupling vì các dữ kiện như khối lượng, độ cồng kềnh là các thuộc tính của order. Khi sử dụng strategy pattern để mở rộng phần payment, nhóm vẫn chưa hoàn toàn linh động khi vẫn sử dụng lại UI có sẵn được xây dựng thay vì UI và handle riêng cho từng phương thức. Điều này khiến việc thêm các phương thức mới trong tương lai gặp khó khăn khi phải xây dựng thêm UI và thay đổi một phần logic xử lý UI tại payment. Các dữ liệu PaymentTransaction nhiều khi không được sử dụng đến (như trong interbankSubsystem).