**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

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



**BÁO CÁO MÔN HỌC**

**MẪU THIẾT KẾ PHẦN MỀM**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | TS. Nguyễn Thị Thu Trang |
|  | TS. Bùi Thị Mai Anh |
| Nhóm sinh viên thực hiện: |  |
| Nguyễn Đình Hùng | 20183548 |
| Phạm Trung Hiếu | 20183535 |
| Phạm Hữu Anh Quốc | 20183616 |
| Nguyễn Minh Tuân | 20183652 |
| Mã lớp học: | 132681 |

Hà Nội, ngày 09 tháng 09 năm 2022

**MỤC LỤC**

[1. Tổng quan 5](#_Toc108340771)

[1.1. Mục tiêu 5](#_Toc108340772)

[1.2. Phạm vi 5](#_Toc108340773)

[1.2.1. Mô tả khái quát phần mềm 5](#_Toc108340774)

[1.2.2. Các chức năng chính của phần mềm 5](#_Toc108340775)

[1.2.3. Cấu trúc mã nguồn 6](#_Toc108340776)

[1.2.4. Các yêu cầu thêm cần cân nhắc cùng quá trình tái cấu trúc 7](#_Toc108340777)

[1.2.5. Các hoạt động review, refactor thực thi trên mã nguồn để đạt được mục tiêu 7](#_Toc108340778)

[1.2.6. Kết quả dự kiến 8](#_Toc108340779)

[1.3. Danh sách thuật ngữ 8](#_Toc108340780)

[1.4. Danh sách tài liệu tham khảo 8](#_Toc108340781)

[2. Đánh giá thiết kế cũ 8](#_Toc108340782)

[2.1. Nhận xét chung 8](#_Toc108340783)

[2.2. Đánh giá các mức độ coupling và cohesion 9](#_Toc108340784)

[2.2.1. Mức độ coupling 9](#_Toc108340785)

[2.2.2. Mức độ cohesion 11](#_Toc108340786)

[2.3. Đánh giá việc tuân theo SOLID 13](#_Toc108340787)

[2.3.1. Single Responsibility Principle (SRP) 14](#_Toc108340788)

[2.3.2. Open Closed Principle (OCP) 15](#_Toc108340789)

[2.3.3. Liskov Substitution Principle (LSP) 16](#_Toc108340790)

[2.3.4. Interface Segregation Principle (ISP) 16](#_Toc108340791)

[2.3.5. Dependency Inversion Principle (DIP) 16](#_Toc108340792)

[2.4. Các vấn đề về Clean Code 17](#_Toc108340793)

[2.4.1. Clear name 17](#_Toc108340794)

[2.4.2. Clean function/method 17](#_Toc108340795)

[2.4.3. Clean class 18](#_Toc108340796)

[3. Đề xuất cải tiến 19](#_Toc108340797)

[3.1. Giải quyết các vấn đề với yêu cầu phát sinh thêm một loại mặt hàng Media mới 19](#_Toc108340798)

[3.2. Giải quyết các vấn đề với yêu cầu phát sinh thêm màn hình mới để xem chi tiết sản phẩm 20](#_Toc108340799)

[3.3. Giải quyết các vấn đề với yêu cầu phát sinh sửa lỗi khi load giao diện 21](#_Toc108340800)

[3.4. Giải quyết các vấn đề với yêu cầu phát sinh thay đổi cách tính phí vận chuyển và sử dụng thư viện mới 22](#_Toc108340801)

[3.5. Giải quyết các vấn đề với yêu cầu phát sinh thêm phương thức thanh toán mới là thẻ nội địa 24](#_Toc108340802)

[3.6. Giải quyết các vấn đề với yêu cầu phát sinh cập nhật lại chức năng hủy đơn hàng 25](#_Toc108340803)

[3.7. Giải quyết các vấn đề vi phạm nguyên lý Single Responsibility Principle trong class AuthenticationController và PlaceOrderController 27](#_Toc108340804)

[3.7.1. Refactor class AuthenticationController 27](#_Toc108340805)

[3.7.2. Refactor class PlaceOrderController 28](#_Toc108340806)

[4. Tổng kết 28](#_Toc108340807)

[4.1. Kết quả tổng quan 28](#_Toc108340808)

[4.2. Các vấn đề còn tồn đọng 28](#_Toc108340809)

**DANH MỤC HÌNH ẢNH**

[Hình 1 Biểu đồ use case tổng quan 6](#_Toc108340751)

[Hình 2 Thiết kế tổng quan của hệ thống ban đầu 7](#_Toc108340752)

[Hình 3 Sử dụng Java Reflections để xử lý hard-code trong class HomeScreenHandler 20](#_Toc108340753)

[Hình 4 Thiết kế lớp Media khi xảy ra yêu cầu thêm màn hình mới 21](#_Toc108340754)

[Hình 5 Giải quyết vấn đề xử lý IOException khi load màn hình 22](#_Toc108340755)

[Hình 6 Giải quyết vấn đề thay đổi cách tính chi phí và thư viện tính khoảng cách 23](#_Toc108340756)

[Hình 7 Thiết kế khi xảy ra thay đổi thư viện tính khoảng cách và công thức tính phí 24](#_Toc108340757)

[Hình 8 Giải quyết vấn đề phát sinh phương thức thanh toán mới 25](#_Toc108340758)

[Hình 9 Biểu đồ máy trạng thái của đơn hàng với yêu cầu hiện tại 25](#_Toc108340759)

[Hình 10 Áp dụng State Design Pattern cho việc quản lý trạng thái của đơn hàng 26](#_Toc108340760)

[Hình 11 Biểu đồ máy trạng thái của đơn hàng khi phát sinh thay đổi 26](#_Toc108340761)

[Hình 12 Cập nhật các lớp trong việc thiết kế State Design Pattern để quản lý trạng thái của đơn hàng 27](#_Toc108340762)

[Hình 13 Refactor class AuthenticationController 27](#_Toc108340763)

[Hình 14 Refactor class PlaceOrderController 28](#_Toc108340764)

1. **Tổng quan**
   1. **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 là những người đã có kiến thức cơ sở về lập trình hướng đối tượng sử dụng ngôn ngữ Java, người thiết kế hệ thống phần mềm, người có kiến thức cơ bản về công nghệ thông tin. Báo cáo là kết quả làm việc của nhóm tác giả, nhằm tổng kết kết quả, cách thức thực hiện việc tái cấu trúc, refactor mã nguồn dựa trên một hệ thống phần mềm có sẵn được cung cấp bởi giảng viên giảng dạy môn học, sử dụng các mẫu thiết kế phần mềm trong môn học để áp dụng vào phân tích và cải tiến thiết kế hiện tại.

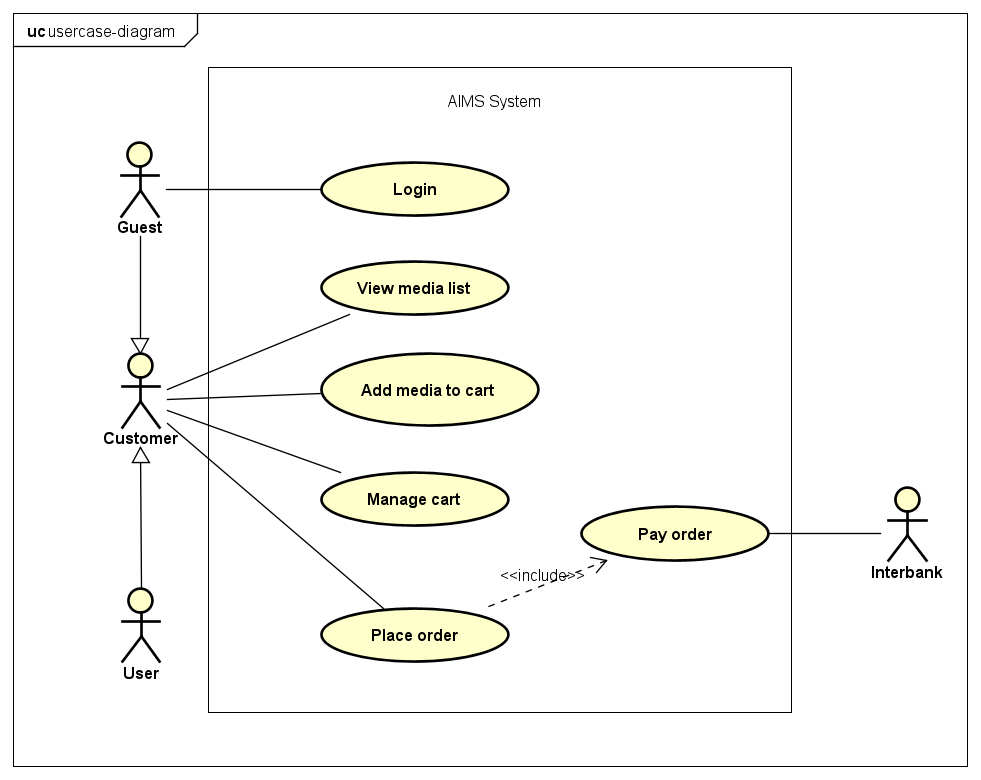
* 1. **Phạm vi**
     1. **Mô tả khái quát phần mềm**

Phần mềm AIMS là một cửa hàng online, có chức năng đặt hàng các loại media (sách, đĩa CD, đĩa DVD) từ cửa hàng. Sau khi đặt hàng, đơn hàng sẽ được thanh toán online bởi thẻ tín dụng của người dùng và sau đó sản phẩm sẽ được vận chuyển tới địa chỉ mà khách hàng yêu cầu. Phần mềm hiện tại đã đáp ứng được các chức năng chính tại thời điểm hiện tại, nhưng chưa đảm bảo việc tuân thủ các nguyên lý SOLID và không có khả năng đáp ứng được một số yêu cầu thay đổi có thể phát sinh trong tương lai.

* + 1. **Các chức năng chính của phần mềm**

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

* Xem, tìm kiếm danh sách các sản phẩm đang có trong cửa hàng
* Thanh toán đơn hàng, xác định địa chỉ giao hàng và hướng dẫn giao hàng



Hình 1 Biểu đồ use case tổng quan

Biểu đồ use case tổng quan của hệ thống AIMS được thể hiện ở Hình 1.

Hệ thống AIMS có các tác nhân: Customer, Guest, User và Interbank. Guest, User đều có chung một số hành vi nghiệp vụ khi tương tác với hệ thống AIMS và đều là nhóm người sử dụng có thể đem lại doanh thu cho cửa hàng thông qua việc đặt mua, thanh toán đơn hàng. Do đó nhóm bổ sung tác nhân Customer, là tác nhân trừu tượng, tác nhân Guest và tác nhân User kế thừa tác nhân trừu tượng này. Guest là vai trò của người sử dụng chưa đăng nhập vào hệ thống, do đó Guest có thể thực hiện Login (đăng nhập). User là vai trò của người sử dụng đã đăng nhập vào hệ thống. Cả Guest và User đều có thể thực hiện View Media List (xem danh sách sản phẩm), Add media to cart (thêm sản phẩm vào giỏ hàng), Manage cart (quản lý giỏ hàng, bảo gồm xem, sửa giỏ hàng), Place order (đặt hàng). Trong khi thực hiện đặt hàng, sẽ có nghiệp vụ Pay order (thanh toán đơn hàng). Nghiệp vụ này có sự tham gia tương tác của tác nhân Interbank, là một tác nhân hệ thống, đại diện cho hệ thống thanh toán online.

* + 1. **Cấu trúc mã nguồn**

Mã nguồn của phần mềm hiện tại đang được thiết kế theo mô hình MVC, gồm ba tầng chính là Model, View và Controller. Trong đó, tầng View bao gồm các xử lý logic ở các màn hình, lấy dữ liệu từ người người dùng và chuyển cho tầng Controller và thực hiện điều hướng giữa các màn hình với nhau; tầng Controller bao gồm việc xử lý logic của luồng dữ liệu, xác thực dữ liệu người dùng và tương tác với hệ thống subsystem để thực hiện thanh toán tiền cho đơn hàng của người dùng; tầng Model sẽ thực hiện lưu dữ liệu của hệ thống và bao gồm hai modules chính là Data Access Object và Entity. Ngoài ra hệ thống còn có một các thành phần dùng chung, gọi chung là Common, bao gồm các Interfaces, các Exceptions và Utils. Thiết kế tổng quan ban đầu của hệ thống phần mềm được module hóa như biểu đồ miêu tả sau đây:

Diagram

Description automatically generated

Hình 2 Thiết kế tổng quan của hệ thống ban đầu

* + 1. **Các yêu cầu thêm cần cân nhắc cùng quá trình tái cấu trúc**

Việc tiến hành tái cấu trúc mã nguồn của hệ thống hiện tại chỉ nhằm mục đích giúp hệ thống có khả năng đáp ứng với các thay đổi trong tương lai một các tốt hơn, đảm bảo các nguyên lý trong SOLID, đặc biệt là nguyên lý Open-closed Principle. Việc tái cấu trúc này phải đảm bảo các tính năng của hệ thống hiện tại không bị thay đổi cả về mặt chức năng và phi chức năng.

* + 1. **Các hoạt động review, refactor thực thi trên mã nguồn để đạt được mục tiêu**

Các hoạt động review mã nguồn hiện tại bao gồm:

* Xác định các mức độ coupling và cohension của mã nguồn
* Xác định các vi phạm nguyên lý SOLID

Các hoạt động refactor mã nguồn bao gồm:

* Refactor methods, class để đảm bảo các tiêu chí về clean code
* Refactor những vi phạm nguyên lý SOLID đã phát hiện được trong bước review
* Áp dụng các mẫu thiết kế phần mềm đã học vào hệ thống một cách phù hợp
  + 1. **Kết quả dự kiến**

Kết quả dự kiến sau khi refactor mã nguồn là hệ thống đáp ứng được tiêu chí High Cohesion và Loose Coupling, loại bỏ những vi phạm các nguyên lý trong SOLID, giúp hệ thống có khả năng thích ứng với các yêu cầu trong tương lai tốt hơn, có khả năng mở rộng tốt hơn so với mã nguồn ban đầu bằng các sử dụng các mẫu thiết kế, các kiến thức về clean code trên lớp.

* 1. **Danh sách thuật ngữ**

|  |  |  |
| --- | --- | --- |
| Số thứ tự | Thuật ngữ | Giải thích |
| 1 | Message digest | Hàm băm mật mã trả về một chuỗi ký tự sử dụng hàm băm một chiều |
| 2 | Method | Phương thức của một lớp |
| 3 | Attribute | Thuộc tính của một lớp |
| 4 | Refactor | Tái cấu trúc mã nguồn để đạt được mã nguồn có thiết kế tốt hơn |
| 5 | Data Access Object | Đối tượng có nhiệm vụ truy cập vào cơ sở dữ liệu để truy xuất dữ liệu phục vụ logic ứng dụng |
| 6 | Entity | Đối tượng lưu trữ dữ liệu của hệ thống |

* 1. **Danh sách tài liệu tham khảo**
* Sách Design Patterns: Elements of Reusable Object-Oriented Software
* Tài liệu, vở ghi trên lớp môn học Mẫu thiết kế phần mềm IT4536

1. **Đánh giá thiết kế cũ**
   1. **Nhận xét chung**

Về mặt chức năng hiện tại của thiết kế của hệ thống, mã nguồn cũ đã đáp ứng được các yêu cầu. Tuy nhiên với các yêu cầu có thể phát sinh trong tương lai, hệ thống hiện tại lại tỏ ra chưa có khả năng thích ứng tốt. Mã nguồn hiện tại vẫn còn các modules phụ thuộc vào nhau một ở mức thấp thay vì chỉ phụ thuộc vào nhau ở mức trừu tượng. Khi phụ thuộc vào nhau ở level thấp, nếu một module thay đổi thì sẽ gây ra nhiều thay đổi ở các module mà phụ thuộc vào nó hay ngược lại. Còn khi thiết kế chỉ thuộc vào nhau ở mức trừu tượng, nếu một module thay đổi thì các module còn lại do chỉ biết về class đó một cách trừu tượng nên sẽ ít bị ảnh hưởng hơn. Hơn nữa, như đã phân tích ở trên, thiết kế cũ vẫn còn tồn đọng nhiều vấn đề về clean code, cần phải tiến hành tái cấu trúc và sửa đổi để đảm bảo mã nguồn dễ bảo trì và mở rộng hơn.

* 1. **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).

* + 1. **Mức độ coupling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Các mức độ về Coupling | Module | Mô tả | Lý do |
| 1 | Content coupling | entity.shipping.  DeliveryInfo | Các attributes của class đang để access modifier là protected. | Khi để access modifier là protected, các classes khác trong cùng package có thể truy cập vào các attributes một cách 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 đang được dùng chung cho cả hệ thống được khai báo không có từ khóa final. | Khi khai báo biến common static để dùng chung cho cả hệ thống, nếu được khai báo là hằng số (với từ khóa final), dẫn tới các module có thể cùng đọc và cùng thay đổi dữ liệu dùng chung này, gây ra sự không rõ ràng về vai trò của dữ liệu, khó xác định được thành phần nào thay đổi giá trị của biến trong quá trình vận hành hệ thống. |
| 3 | Control coupling | Không có | Không có | Không có |
| 4 | Stamp coupling | entity.cart.Cart | Tham số của method checkMediaInCart đang có kiểu dữ liệu là Media | Tham số truyền vào cho method checkMediaInCart bị dư thừa về mặt dữ liệu, method này chỉ cần dùng một trường dữ liệu duy nhất của Media là id để có thể thực hiện logic nghiệp vụ. |
| views.screen.  intro.  IntroScreenHandler | Tham số của method setupData đang có kiểu dữ liệu là Object | Tham số Object truyền vào method setupData đang không được sử dụng trong phần cài đặt của method này, dẫn tới việc dư thừa về mặt dữ liệu. |
| views.screen.  home.  LoginScreenHandler | Tham số của method backToHomeScreen đang có kiểu dữ liệu là MouseEvent | Tham số MouseEvent truyền vào method backToHomeScreen đang không được sử dụng trong phần cài đặt của method này, dẫn tới việc dư thừa về mặt dữ liệu. |
| 5 | Data coupling | Không có | Không có | Không có |

* + 1. **Mức độ cohesion**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Các mức độ về Cohesion | Module | Mô tả | Lý do |
| 1 | Co-incidental cohesion | utils.Utils | Biến DATE\_FORMATTER và LOGGER được khai báo nhưng không được sử dụng và cũng không có quan hệ gì với nhau. | Biến được khởi tạo nhưng không được sử dụng và không có chức năng phục vụ xử lý logic nào liên quan tới chức năng của class. |
| views.screen.home.  MediaHandler | Biến LOGGER được khởi tạo nhưng không được sử dụng. | Tương tự như trên |
| views.screen.  popup.  Popupscreen | Các methods loading, setupData và setupFunctionality được khai báo nhưng không được sử dụng. | Tương tự như trên |
| views.screen.  shipping.  ShippingScreen  Handler | Method notifyError của class được khai báo nhưng không được sử dụng và cũng không có phần cài đặt. | Tương tự như trên |
| utils.  ApplicationInterface | Biến DATE\_FORMATTER được khai báo nhưng không được sử dụng | Tương tự như trên |
| Package dao | Class OrderDAO, OrderItemDAO, InvoiceDAO được khai báo nhưng không có phần cài đặt và cũng không được sử dụng |  |
| 2 | Logical cohesion | utils.  ApplicationInterface | Các methods get và post của class đang có mức độ logical cohesion | Các methods vừa liệt kê của class ApplicationInterface chỉ có liên kết với nhau một cách logic khi cùng thực hiện việc gọi tới một API bên ngoài hệ thống chứ không phải liên kết chặt chẽ với nhau về mặt chức năng |
| 3 | Temporal cohesion | Không có | Không có | Không có |
| 4 | Procedure cohesion | controller.  PlaceOrder  Controller | Các methods validatePhoneNumber, validateDeliveryInfo, validateName và validateAddress đang có mức độ procedure cohesion | Các methods vừa liệt kê có liên kết với nhau về mặt chức năng khi cùng thực hiện việc validate dữ liệu người dùng nhập vào, nhưng đây lại không phải nhiệm vụ của lớp PlaceOrderController. Các methods này đang được đặt trong class PlaceOrderController chỉ bởi vì chúng có liên quan trình tự thực thi liên tiếp nhau. |
| 5 | Communicational cohesion | Không có | Không có | Không có |
| 6 | Sequential cohesion | Không có | Không có | Không có |
| 7 | Functional cohesion | Không có | Không có | Không có |

* 1. **Đánh giá việc tuân theo SOLID**

Với yêu cầu phát sinh là thay đổi cách hiển thị thông báo lỗi khi xảy ra IOException trên các màn hình, với thiết kế hiện tại khi xảy ra thay đổi trên sẽ phải sửa lại code trong tất cả 8 khối catch IOException của các class, gây ra việc vi phạm nguyên lý Open Closed Principle.

Với yêu cầu phát sinh là thay đổi các tính chi phí vận chuyển, thay vì chỉ tính dựa trên khoảng cách như hiện tại, cách tính mới yêu cầu việc tính chi phí dựa vào các yếu tố bổ sung là khối lượng thực tế của hàng hóa và độ cồng kềnh của kiện hàng, thiết kế hiện tại của hệ thống chưa tuân theo nguyên lý thiết kế Single Responsibility Principle, Open Closed Principle và Dependency Inversion Principle.

Với yêu cầu phát sinh là thay đổi cách tính khoảng cách, sử dụng thư viện mới, thiết kế hiện tại đang vi phạm nguyên lý Open Closed Principle và Dependency Inversion Principle. Do class DeliveryInfo đang có một attribute là DistanceCalculator là một class của thư viện bên ngoài nên khi cần thay đổi thư viện mới, chúng ta cần phải sửa attribute này thành class của thư viện mới. Việc sử dụng một thư viện mới có interface khác với interface của thư viện hiện tại cũng là dấu hiệu của việc phải sử dụng mẫu thiết kế Adapter Design Pattern để có thể refactor code của hệ thống hiện tại.

Với yêu cầu phát sinh là thêm phương thức thanh toán mới là thẻ nội địa (DomesticCard), thiết kế hiện tại cũng đang vi phạm nguyên lý Open Closed Principle và Dependency Inversion Principle do thiết kế đang sử dụng class CreditCard không phải abstract class, có các attributes mà DomesticCard không có, dẫn tới việc khó mở rộng, phát triển thêm khi có yêu cầu thêm loại thẻ DomesticCard.

Với yêu cầu phát sinh là cập nhật lại chức năng hủy đơn hàng, thiết kế hiện tại đang vi phạm nguyên lý Open Closed Principle do hiện tại chưa quản lý các trạng thái của đơn hàng một cách cụ thể. Yêu cầu mới đặt ra cần phải sử dụng State Design Pattern để cài đặt và khắc phục việc vi phạm nguyên lý Open Closed Principle.

Với yêu cầu phát sinh là thêm màn hình xem chi tiết sản phẩm, tại màn hình này, tất cả thông tin của sản phẩm được hiển thị đầy đủ để khách hàng xem xét. Do hệ thống có nhiều loại sản phẩm, mỗi loại lại có những thuộc tính khác nhau, nên để xử lý với yêu cầu phát sinh này, việc bổ sung một method giúp hiển thị thông tin sản phẩm ra màn hình trong class Media là cần thiết. Với yêu cầu phát sinh này, thiết kế hiện tại của hệ thống không bị vi phạm nguyên lý nào trong SOLID.

Với yêu cầu phát sinh là thêm mặt hàng Media mới là AudioBook, hiện tại trên màn hình chính của ứng dụng có liệt kê các loại sản phẩm (bao gồm Book, DVD, CD) để người dùng có thể lọc và tìm kiếm, và phần hiển thị tên các loại sản phẩm này đang bị hardcode trong codebase. Khi thêm loại Media mới sẽ phải tiến hành sửa đoạn code này (method setupFunctionality trong class HomeScreenHandler), điều này đang vi phạm nguyên lý Open Closed Principle.

* + 1. **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. | Class DeliveryInfo là class entity, nên chỉ có nhiệm vụ chứa các thông tin về đơn hàng như tên người nhận, địa chỉ, hướng dẫn giao hàng… chứ không nên có cả nhiệm vụ tính toán chi phí vận chuyển. |
| 2 | controller.  AuthenticationController | Class AuthenticationController đang có một method để sinh mã băm sử dụng hàm MD5 | Class AuthenticationController là class controller, có nhiệm vụ xử lý logic các yêu cầu authenticate người dùng, không nên bao gồm cả nhiệm vụ sinh mã băm như hiện tại. Nên tách method sinh mã băm ra một class riêng và class controller này có một attribute là class sinh mã băm đó. |
| 3 | controller.  PlaceOrderController | Class PlaceOrderController đang có các method để thực hiện validate dữ liệu người dùng nhập vào. | Class PlaceOrderController là class controller, có nhiệm vụ xử lý logic khi người dùng đặt hàng, không nên chứa cả các method thực hiện validate dữ liệu người dùng nhập vào. Việc validate nên tách riêng ra cho một class validator thực thi, và class controller này sẽ có một attribute là class validator đó. |

* + 1. **Open Closed Principle (OCP)**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | entity.payment.  PaymentTransaction và entity.payment.  CreditCard | Class DeliveryInfo đang có một attribute là CreditCard | Do có yêu cầu phát sinh trong tương lai phải thêm loại thẻ mới có các attributes mà CreditCard hiện tại không có, dẫn tới việc sẽ phải sửa đổi attribute CreditCard trong class DeliveryInfo, chứng tỏ thiết kế hiện tại đang vi phạm Open Closed Principle. |
| 2 | controller.  PlaceOrderController | Class này đang phụ thuộc trực tiếp vào class DistanceCalculator | Do class controller này phụ thuộc trực tiếp vào class DistanceCaculator - class thư viện bên ngoài sử dụng tính khoảng cách, mà trong tương lai có yêu cầu phát sinh phải thay đổi thư viện tính khoảng cách, dẫn tới việc sẽ phải sửa class controller phụ thuộc vào thư viện mới, chứng tỏ thiết kế hiện tại đang không thỏa mãn Open Closed Principle |

* + 1. **Liskov Substitution Principle (LSP)**

Với thiết kế hiện tại và các thay đổi có thể phát sinh trong tương lại, hệ thống đang không có vi phạm Liskov Substitution Principle.

* + 1. **Interface Segregation Principle (ISP)**

Với thiết kế hiện tại và các thay đổi có thể phát sinh trong tương lại, hệ thống đang không có vi phạm Interface Segregation Principle.

* + 1. **Dependency Inversion Principle (DIP)**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Mô tả | Lý do |
| 1 | controller.  PaymentController và  subsystem.  InterbankInterface và subsystem.  InterbankSubsystem | Các class và interface này hiện tại đang phụ thuộc trực tiếp vào class CreditCard | Do có yêu cầu phát sinh trong tương lai phải thêm loại thẻ mới có các attributes mà CreditCard hiện tại không có, dẫn tới việc sẽ phải sửa đổi attribute CreditCard trong class controller và interface này. Dẫn tới việc vi phạm Dependency Inversion Principle. Các class controller và interface này nên chỉ phụ thuộc vào abstract class của card trong hệ thống. |

* 1. **Các vấn đề về Clean Code**
     1. **Clear name**

Về tổng quan, tình trạng mã nguồn ban đầu đã đáp ứng được clear name, các tên của biến, method, class đều mang ý nghĩa. Chỉ có một số vị trí biến nên sửa đổi tên biến để mang nhiều ý nghĩa hơn so với hiện tại.

|  |  |  |  |
| --- | --- | --- | --- |
| # | 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 dateSplitedStrs để thể hiện rõ ràng ý nghĩa của biến |
| 2 | utils.  ApplicationProgramming  Interface | Method post có hàm LOGGER đang ghi log bị sai chính tả từ “respone” | Sửa thông tin log từ “respone” thành “response” |
| 3 | views.screen.home.  HomeScreenHandler | Method setupData có tên biến medium bị sai chính tả | Đổi tên biến medium thành media |

* + 1. **Clean function/method**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Module | Vị trí có tên chưa clear | Đề xuất sửa đổi |
| 1 | controller.  AuthenticationController | Method md5 | Do md5 là tên một thuật toán dùng để mã hóa, nên cần sửa tên method md5 thành genDigestByMd5 để thể hiện rõ ràng nhiệm vụ của hàm này là sinh một message-digest sử dụng hàm md5. |
| 2 | dao.media.  MediaDao | Method getCurrentQuantity có truyền vào tham số id của media nhưng không sử dụng để làm tham số cho câu truy vấn cơ sở dữ liệu. Hiện tại câu truy vấn cơ sở dữ liệu đang là “SELECT \* FROM MEDIA” | Sửa đổi câu truy vấn cơ sở dữ liệu trở thành “SELECT \* FROM MEDIA WHERE ID = ?” và truyền tham số id của method vào. |
| 3 | views.screen.home.  HomeScreenHandler | Method update có hai vị trí log ra cùng message giống nhau, nhưng không khai báo biến để dùng chung mà sử dụng hard-code | Tạo biến errorMessage để lưu các giá trị log giống nhau |
| 4 | controller.  ViewCartController | Method getCartSubtotal | Sửa phần cài đặt của method thành inline function |
| 5 | views.screen.intro.  IntroScreenHandler | Method setupFunctionality đang hard-code đường dẫn tới ảnh logo của ứng dụng | Chuyển giá trị đường dẫn tới logo của ứng dụng vào class ViewsConfig để tránh việc hard-code và có thể tái sử dụng hoặc thay đổi giá trị này dễ dàng hơn |

* + 1. **Clean class**

Hiện tại, các class của mã nguồn ban đầu hầu hết đáp ứng được các tiêu chí về clean class, nhưng vẫn còn một số class có các attribute được khai báo nhưng không sử dụng, các import không sử dụng hoặc một số method không có phần cài đặt hoặc có phần cài đặt nhưng lại không được sử dụng tại bất kỳ vị trí nào trong hệ thống.

|  |  |  |
| --- | --- | --- |
| # | Vấn đề tồn tại | Danh sách class |
| 1 | Có các attribute được khai báo nhưng không được sử dụng | entity.media.Book  entity.media.CD  entity.media.DVD  utils.Utils  utils.ApplicationProgrammingInterface |
| 2 | Có các method được khai báo nhưng không được sử dụng | entity.media.Book  entity.media.CD  entity.media.DVD  entity.media.Media  entity.invoice.Invoice  dao.media.MediaDAO  entity.order.Order  views.screen.popup.PopupScreen  views.screen.shipping.ShippingScreenHandler |
| 3 | Có các import không được sử dụng | entity.media.Book  views.screen.home.HomeScreenHandler |
| 4 | Thiếu khoảng trắng giữa các ký tự syntax của ngôn ngữ (tuân thủ coding convention) | entity.cart.Cart  entity.cart.CartItem  common.exception.  NotEnoughTransactionInfoException |
| 5 | Class có các đoạn log có cùng giá trị nhưng đang hard-code thay vì tạo biến để dùng chung | utils.MyMap |

1. **Đề xuất cải tiến**
   1. **Giải quyết các vấn đề với yêu cầu phát sinh thêm một loại mặt hàng Media mới**

Như đã trình bày ở mục đánh giá thiết kế hiện tại của hệ thống theo các nguyên lý SOLID, mỗi khi hệ thống cần thêm mặt hàng mới sẽ phát sinh vấn đề phải sửa danh sách tên các mặt hàng của hệ thống để người dùng tìm kiếm tại màn hình chính. Việc sửa đổi danh sách tên này đang vi phạm nguyên lý Open Closed Principle. Để khắc phục việc vi phạm này, chúng ta có thể sử dụng Java Reflections để liệt kê các class Media đang có trong hệ thống.

Diagram

Description automatically generated

Hình 3 Sử dụng Java Reflections để xử lý hard-code trong class HomeScreenHandler

* 1. **Giải quyết các vấn đề với yêu cầu phát sinh thêm màn hình mới để xem chi tiết sản phẩm**

Như đã xác định ở trên, với yêu cầu phát sinh là thêm màn hình xem chi tiết sản phẩm, thiết kế hiện tại của hệ thống hiện đang không vi phạm nguyên lý nào trong SOLID. Khi cần thêm màn hình mới, chúng ta chỉ cần thêm method getAttributesAndValues() tại class Media, method này sẽ trả về tên attribute và giá trị của attribute đó của object. Do các loại sản phẩm khác nhau đều kế thừa class Media này và có thể override lại method đó để có những các hiển thị riêng biệt.

Text

Description automatically generated

Hình 4 Thiết kế lớp Media khi xảy ra yêu cầu thêm màn hình mới

* 1. **Giải quyết các vấn đề với yêu cầu phát sinh sửa lỗi khi load giao diện**

Với thiết kế hiện tại của hệ thống, mỗi lần xảy ra lỗi IOException, các method sẽ gọi tới một hàm static PopupScreen.error(“Error when loading resources.”). Điều này dẫn tới việc khi muốn thay đổi cách xử lý khi xảy ra lỗi IOException, chúng ta sẽ phải thay đổi tất cả các vị trí đang xử lý logic này.

Do các class dùng để xử lý logic giao diện đang kế thừa class BaseScreenHandler, nên để giải quyết vấn đề yêu cầu phát sinh này, chúng ta có thể viết một method handleIOException trong class BaseScreenHandle, từ đó các class con có thể dùng method này thay vì xử lý độc lập như hiện nay. Khi có yêu cầu phát sinh, chúng ta chỉ cần thay đổi việc cài đặt method này thay vì sửa tại mọi chỗ xảy ra IOException.

Biểu đồ lớp thiết kế sau khi bổ sung method handleIOException như sau:

Diagram

Description automatically generated

Hình 5 Giải quyết vấn đề xử lý IOException khi load màn hình

* 1. **Giải quyết các vấn đề với yêu cầu phát sinh thay đổi cách tính phí vận chuyển và sử dụng thư viện mới**

Với thiết kế hiện tại, trong class DeliveryInfo có một attribute có kiểu dữ liệu là DistanceCalculator (là một class của thư viện Distance-API), việc sử dụng trực tiếp class như vậy dẫn tới việc khi muốn thay đổi thư viện tính khoảng cách thành Alt-Distance-API, chúng ta sẽ phải sửa đổi lại class DeliveryInfo này. Hơn nữa, trong yêu cầu phát sinh thay đổi cách tính khoảng cách dựa vào khối lượng hàng hóa, độ cồng kềnh của kiện hàng thì cách thiết kế hiện tại sẽ không đáp ứng được, do các thông tin này được lưu trong class Order.

Để giải quyết vấn đề thay đổi thư viện mới, chúng ta sẽ áp dụng mẫu thiết kế Adapter Design Pattern, tạo ra một interface để thực hiện việc composition thư viện sử dụng vào class implement interface đó (thiết kế chi tiết sẽ được thể hiện trong biểu đồ lớp bên dưới). Còn đối với vấn đề thay đổi công thức tính toán chi phí giao hàng, chúng ta sẽ sử dụng Strategy Design Pattern, chuyển sự phụ thuộc vào class tính toán chi phí từ class DeliveryInfo sang class Order do các thông tin để xác định khối lượng hàng hóa, độ cồng kềnh được lưu trong class Order. Khi yêu cầu phát sinh cần thay đổi hàm tính toán, chúng ta sẽ tạo một class mới và implement interface ShippingFeeCalculatorStrategy (các class được thể hiện trong biểu đồ lớp bên dưới).

Biểu đồ lớp thiết kế sau khi sửa đổi như sau:

Diagram

Description automatically generated

Hình 6 Giải quyết vấn đề thay đổi cách tính chi phí và thư viện tính khoảng cách

Hiện tại, để tính chi phí chỉ cần biết tọa độ điểm nhận, 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 OrderDistanceBaseStrategy và class AltDistanceCbalculator tương ứng với thay đổi về việc thay đổi công thức tính toán và thay đổi thư viện tính khoảng cách. Thiết kế khi xảy ra cả hai thay đổi sẽ như sau:

Diagram

Description automatically generated

Hình 7 Thiết kế khi xảy ra thay đổi thư viện tính khoảng cách và công thức tính phí

* 1. **Giải quyết các vấn đề với yêu cầu phát sinh thêm phương thức thanh toán mới là thẻ nội địa**

Với thiết kế hiện tại, chỉ có một lớp CreditCard duy nhất lưu trữ thông tin về thẻ ngân hàng mà người dùng sử dụng để thanh toán khi mua hàng của hệ thống. Điều đó dẫn tới việc vi phạm nguyên tắc Dependency Inversion Principle khi các module khác như sub-system hay class PaymentTransaction đều phụ thuộc vào class CreditCard. Dẫn tới việc sẽ rất khó sửa đổi khi có thay đổi phát sinh một phương thức mới.

Để khắc phục vấn đề này, chúng ta sẽ tạo ra một abstract class PaymentCard chứa các thông tin cơ bản mà bất kỳ loại thẻ nào cũng phải có, sau đó cho class CreditCard kế thừa abstract class này và bổ sung thêm các attributes riêng biệt của CreditCard. Các module ở trước kế trước bị phụ thuộc trực tiếp vào class CreditCard sẽ được sửa đổi để chỉ phụ thuộc vào abstract class PaymentCard.

Diagram

Description automatically generated

Hình 8 Giải quyết vấn đề phát sinh phương thức thanh toán mới

* 1. **Giải quyết các vấn đề với yêu cầu phát sinh cập nhật lại chức năng hủy đơn hàng**

Do chức năng hiện tại của hệ thống cho phép người dùng hủy đơn hàng ở bất cứ thời điểm nào, nên việc xử lý logic khi chuyển đổi giữa các state hiện tại (đơn hàng đang chờ thanh toán; đơn hàng đang vận chuyển; đơn hàng bị hủy) vẫn còn đơn giản và chưa cần áp dụng design pattern. Nhưng do trong tương lai có yêu cầu phát sinh cần bổ sung thêm trạng thái đơn hàng được quản trị viên phê duyệt, và logic để chuyển đổi giữa trạng thái của đơn hàng phức tạp hơn nhiều, nên chúng ta nên áp dụng State Design Pattern để giải quyết vấn đề với yêu cầu phát sinh này.

Biểu đồ máy trạng thái (state machine diagram) của đơn hàng (class Order) tại thời điểm hiện tại:

*Diagram

Description automatically generated*

Hình 9 Biểu đồ máy trạng thái của đơn hàng với yêu cầu hiện tại

Khi áp dụng State Design Pattern, biểu đồ lớp của class Order và các class liên quan tới design pattern này sẽ được thiết kế như sau:

Diagram

Description automatically generated

Hình 10 Áp dụng State Design Pattern cho việc quản lý trạng thái của đơn hàng

Khi phát sinh yêu cầu mới, đơn hàng sẽ bổ sung thêm trạng thái phê duyệt, biểu đồ máy trạng thái của đơn hàng lúc này sẽ như sau:

Diagram

Description automatically generated

Hình 11 Biểu đồ máy trạng thái của đơn hàng khi phát sinh thay đổi

Diagram

Description automatically generated

Hình 12 Cập nhật các lớp trong việc thiết kế State Design Pattern để quản lý trạng thái của đơn hàng

* 1. **Giải quyết các vấn đề vi phạm nguyên lý Single Responsibility Principle trong class AuthenticationController và PlaceOrderController**
     1. **Refactor class AuthenticationController**

Với thiết kế hiện tại, nhiệm vụ của class AuthenticationController là thực hiện các công việc liên quan tới xác thực người dùng. Nhưng trong class này hiện đang có method md5 chỉ có nhiệm vụ sinh message digest sử dụng hàm băm md5. Việc thiết kế như vậy đang vi phạm nguyên lý Single Responsibility Principle. Để khắc phục vấn đề này, chúng ta nên tạo riêng một class MessageDigestUtils riêng chỉ để thực hiện nhiệm vụ sinh message digest từ message ban đầu và cài đặt method md5 ở class này. Khi đó, ta vừa có thể tái sử dụng method md5 ở nhiều nơi trong hệ thống nếu cần thiết, vừa có thể dễ dàng bổ sung thêm các method sử dụng các thuật toán khác để sinh message digest. Thiết kế sau khi refactor sẽ được mô tả trong biểu đồ class như sau:

Diagram

Description automatically generated

Hình 13 Refactor class AuthenticationController

* + 1. **Refactor class PlaceOrderController**

Tương tự với vấn đề ở class AuthenticationController, class PlaceOrderController đang có nhiệm vụ thực hiện khởi tạo Order, Invoice và tiến hành xử lý logic khi người dùng chọn thanh toán. Tuy nhiên ở class này đang có ba methods thực hiện việc xác định dữ liệu của người dùng nhập vào có đúng định dạng hay không. Việc thiết kế như vậy đang vi phạm nguyên lý Single Responsibility Principle. Để khắc phục vấn đề này, chúng ta nên tạo một class Validator riêng chỉ để thực hiện nhiệm vụ xác thực dữ liệu người dùng nhập vào hệ thống. Khi đó, nếu có sự thay đổi về logic xác thực dữ liệu, chúng ta có thể dễ dàng thay đổi chúng trong class Validator này. Thiết kế sau khi refactor sẽ được mô tả trong biểu đồ class như sau:

Diagram

Description automatically generated

Hình 14 Refactor class PlaceOrderController

1. **Tổng kết**
   1. **Kết quả tổng quan**

Sau khi tiến hành tái cấu trúc mã nguồn, tuy vẫn còn một số hạn chế, nhưng thiết kế của hệ thống cơ bản đã đáp ứng được các nguyên lý SOLID, thích ứng được khi có hệ thống xảy ra các yêu cầu thay đổi phát sinh trong tương lai. Các vấn đề về clean name, clean method, clean class cũng đã được khắc phục. Các vấn đề về Coupling và Cohension như đã phân tích ở trên cũng được xử lý, đảm bảo hệ thống không còn bị vi phạm. Việc tiến hành tái cấu trúc một mã nguồn hệ thống hiện có giúp các thành viên trong nhóm có cơ hội áp dụng kiến thức được học trên lớp vào sản phẩm thực tế, biết cách xác định các vấn đề cần áp dụng mẫu thiết kế phần mềm để xử lý, cài đặt và triển khai các mẫu thiết kế.

* 1. **Các vấn đề còn tồn đọng**

Việc tái cấu trúc thiết kế để đáp ứng được với yêu cầu phát sinh thay đổi công thức tính phí vận chuyển, sử dụng Strategy Design Pattern đang có mức độ Stamp Coupling ở tham số của hàm tính phí. Hiện tại với yêu cầu của hệ thống, tham số của hàm tính phí chỉ cần là địa điểm người nhận của đơn hàng. Tuy nhiên do yêu cầu thay đổi công thức tính phí vận chuyển mới yêu cầu phí vận chuyển sẽ được tính dựa vào khối lượng thực tế, khoảng cách và độ cồng kềnh của kiện hàng, nên nhóm đang cài đặt tham số cho hàm tính phí này là Order (thông tin của cả đơn hàng). Kể cả khi có thay đổi công thức tính chi phí, việc truyền tham số là Order vẫn có mức độ 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) hay trạng thái của order (OrderState).