ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Báo cáo Mẫu thiết kế phần mềm

Version 1.0

DP.04

Nguyễn Thị Duyên

Nguyễn Văn Dương

Nguyễn Khương Duy

Nguyễn Hữu Giang

*Hà Nội,* *tháng 6 năm 2021*

Mục lục

Mục lục 1

1 Tổng quan 3

1.1 Mục tiêu 3

1.2 Phạm vi 3

1.3 Danh sách thuật ngữ 4

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 10

2.3.3 LSP 11

2.3.4 ISP 11

2.3.5 DIP 11

2.4 Các vấn đề về Clean Code 12

2.4.1 Clear Name 12

2.4.2 Clean Function/Method 12

2.4.3 Clean Class 13

2.5 Các vấn đề khác 13

3 Đề xuất cải tiến 15

3.1 Vấn đề SRP#1 và giải pháp 15

3.2 Vấn đề SRP#2 và giải pháp 15

3.3 Vấn đề OCP#1 và giải pháp 15

3.4 Vấn đề OCP#2 và giải pháp 15

3.5 Vấn đề OCP#3 và giải pháp 15

3.6 Vấn đề OCP#4 (requirement 1: Thêm sản phẩm mới, requirement 2: thêm màn hình xem chi tiết sản phẩm) và giải pháp 16

3.7 Thay đổi cách tính khoảng cách, sử dụng thư viện mới (requirement 4) 16

3.8 Vấn đề thêm phương thức thanh toán mới (requirement 5) 17

3.9 Mở rộng của Requirement 7 18

3.10 Mở rộng của Requirement 6 19

4 Tổng kết 21

4.1 Kết quả tổng quan 21

4.2 Các vấn đề tồn đọng 21

**Danh sách các minh họa**

**Danh sách các bảng**

**Error! No table of figures entries found.**

# Tổng quan

## Mục tiêu

Mục đích sử dụng của báo cáo: Giới thiệu về đề tài, phân tích các vấn đề của mã nguồn hiện tại và đưa ra các giải pháp để khắc phục những vấn đề đó

Các đối tượng người đọc trong quy trình xử lý phần mềm:

* Những thành viên tham gia xử lý các vấn đề của mã nguồn cũ
* Những thành viên phát triển mã nguồn này trong tương lai
* Những người đọc có quan tâm đến mẫu thiết kế phần mềm, sử dụng làm tài liệu tham khảo

Nội dung của báo cáo bao gồm:

* Giới thiệu đề tài
* Đánh giá thiết kế cũ
* Các giải pháp để khắc phục những vấn đề còn tồn đọng

## Phạm vi

1.2.1 Mô tả khái quát phần mềm

Phần mềm AIMS là phần mềm giúp người dùng mua những sản phẩm sách, đĩa CD, đĩa DVD

Công nghệ sử dụng:

* Ngôn ngữ lập trình: Java
* Thư viện xây dựng giao diện: JavaFX
* Hệ quản trị cơ sở dữ liệu: Sqlite

1.2.2 Các chức năng chính của phần mềm

Diagram

Description automatically generated

Hình : Use Case tổng quan của hệ thống

1.2.3 Cấu trúc mã nguồn

1.2.4 Các yêu cầu cân nhắc cùng quá trình tái cấu trúc

* Đảm bảo chương trình vẫn hoạt động chính xác như cũ
* Đảm bảo mã nguồn sẽ dễ dàng tái sử dụng, bổ xung khi có thay đổi trong tương lai

1.2.5 Các hoạt động thực thi trên mã nguồn

* Review để đưa ra các vấn đề tồn đọng về thiết kế
* Refactor code theo thiết kế mới, đáp ứng các nguyên lý SOLID, Clean Code và áp dụng một số design pattern

1.2.6 Kết quả dự kiến

Xử lý các vấn đề cohesion, coupling và đảm bảo mã nguồn tuân thủ theo nguyên lý SOLID

Giải quyết được 7 yêu cầu đã được đề cập trong tài liệu yêu cầu dự án

## Danh sách thuật ngữ

1. **SOLID**: là viết tắt của 5 chữ cái đầu trong 5 nguyên tắc thiết kế hướng đối tượng
2. **SRP**: Single responsibility principle
3. **OCP**: Open/closed principle
4. **LSP**: Liskov substitution principle
5. **ISP**: Interface segregation principle
6. **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

# Đánh giá thiết kế cũ

## Nhận xét chung

Mã nguồn dựa trên thiết kế cũ còn tồn đọng khá nhiều vi phạm về các nguyên lý thiết kế SOLID, Clean Code và chưa áp dụng nhiều design pattern dẫn đến trong tương lai khi bổ xung thêm các tính năng mới hoặc tái sử dụng lại sẽ gây khó khăn cho những người phát triển sau.

## Đánh giá các mức độ coupling và cohesion

### Coupling

Mức độ phụ thuộc kết dính giữa các module khá chặt. Mức độ Stamp Coupling xảy ra nhiều giữa các module.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Lý do* |
| 1 | Stamp Coupling | Package entity | Class entity.order.Order -> Order(Cart cart) | Phương thức này chỉ cần giá trị phương thức CalSubtotal của đối tượng cart |
| 2 | Stamp Coupling | Package entity | Class entity.shipping.DeliveryInfor - >  calculateShippingFee(Order order) | Phương thức này không cần các thông tin nào của đối tượng order |
| 3 | Stamp Coupling | Package dao | Class dao.media.MediaDAO ->getMediaById( int id) | Tham số id không sử dụng khi truyền vào |
| 4 | Stamp Coupling | Package dao | Class dao.media.MediaDAO ->updateMediaFeildById(…) | Tham số tbname không sử dụng khi truyền vào |
| 5 | Stamp Coupling | Package entity | Class enity.cart.Cart -> checkMediaInCart(Media media) | Phương thức chỉ sử dụng kết quả phương thức getId() của đối tượng media |
| 6 | Stamp Coupling | Package views | Class views.screen.home.HomeScreenHandler ->update(MediaHandler mediaHandler) | Phương thức này chỉ cần hai giá trị của hai phương thức getRequestQuantity() và getMedia() của đối tượng mediaHandler |
| 7 | Stamp Coupling | Package views | Class views.screen.home.HomeScreenHandler ->setupData(Object dto) | Tham số dto không được sử dụng |
| 8 | Common Coupling | Package controller | Class PaymentController ->emptyCart() | Sử dụng trực tiếp các thuộc tính static của class SessionInformation |
| 9 | Common Coupling | Package controller | Class AuthenticationController ->getMainUser() | Sử dụng trực tiếp các thuộc tính static của class SessionInformation |
| 10 | Common Coupling | Package controller | Class BaseController ->checkMediaInCart() | Sử dụng trực tiếp các thuộc tính static của class SessionInformation |
| 11 | Common Coupling | Package controller | Class PlaceOrderController ->placeOrder() | Sử dụng trực tiếp các thuộc tính static của class SessionInformation |
| 12 | Common Coupling | Package controller | Class PlaceOrderController ->createOrder() | Sử dụng trực tiếp các thuộc tính static của class SessionInformation |
| 13 | Content coupling | Package controller | Class AuthenticationController ->login() | Sử dụng thuộc tính static của SessionInformation và gán giá trị thuộc tính cho user trực tiếp |
| 14 | Content coupling | Package controller | Class AuthenticationController ->logout() | Sử dụng trực tiếp các thuộc tính static của SessionInformation và gán bằng null |
| 15 | Common coupling | Package views | Class views.screen.shipping.ShippingScreenHandler -> setupData() | Phương thức có sử dụng biến Global của ShippingConfigs |
| 16 | Content coupling | Package utils | Class utils.AppplicationProgramming -> allowMethods() | Phương thức có sử dụng setAccessible(true) để thay đổi quyền truy cập vào data. |

### Cohesion

Mức độ liên kết chặt chẽ giữa các submodule khá lỏng lẻo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Lý do* |
| 1 | Coincidental Cohesion | Package controller | Class AuthenticationController -> isAnonymousSession() | Phương thức không liên quan đến xác thực. |
| 2 | Coincidental Cohesion | Package controller | Class AuthenticationController -> getMainUser | Phương thức không lên quan đến xác thực |
| 3 | Coincidental Cohesion | Package controller | Class AuthenticationController -> md5 | Phương thức không liên quan đến xác thực của người dùng. |
| 4 | Coincidental  Cohesion | Package subsystem.interbank | InterbankPayloadConverter ->getToday() | Phương thức không liên quan đến các phương thức khác |
| *5* | *Coincidental*  *cohesion* | *Package utils* | *ApplicationProgrammingInterface ->HttpURLConnection()* | *Phương thức này không liên quan gì đến các phương thức/thuộc tính khác trong class* |
| *6* | *Logical cohesion* | *Package views* | *Class views.screen.payment.ResultScreenHandler ->confirmPayment()* | *Phương thức này chỉ liên quan về mặt logic, không phù hợp để đặt trong lớp này về mặt chức năng.* |
| 7 | Logical cohesion | Package views | Class views.screen.invoice.InvoiceScreenHandler | Tất cả các phương thức bên trong class này chỉ liên quan với nhau về mặt logic không sử dụng dữ liệu gì của nhau để tính toán. |
| 8 | Communicational cohesion | Package entity | Class entity.order.Order | Các phương thức bên trong class sử dụng chung thuộc tính để thực hiện luồng công việc |
| 9 | Logical cohesion | Package views | Class views.screen.Popup.PopupScreen | Hai phương thức success, error chỉ liên quan về mặt logic chứ không sử dụng chung dữ liệu gì để tính toán. Có thể tách riêng ra độc lập. |

## Đánh giá việc tuân theo SOLID

### SRP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | controller | PlaceOrderController vừa thực hiện tạo đơn hàng, vừa validate thông tin | lớp này làm hơn 1 nhiệm vụ của nó, (có thêm những phương thức validate trong nó) |
| 2 | controller | AuthenticationController vừa thực hiện xác thực đăng nhập, vừa thực hiện mã hoá md5 | Thực hiện nhiều nhiệm vụ |

### OCP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | controller | Method payOrder() của PaymentController có nhiệm vụ thanh toán đơn hàng | Khi thêm phương thức thanh toán thì cần sửa đổi code |
| 2 | entity | DeliveryInfor class là thông tin về giao hàng, bao trong đó có calculateShippingFee() dùng để tính phí giao hàng | khi thay đổi cách tính khoảng cách( hoặc thêm cách tính) thì cần sửa đổi code của calculateShippingFee() |
| 3 | subsystem | extractPaymentTransaction() của InterbankPayloadConverter | Nếu thêm phương thức thanh toán mới thì cần sửa đổi |
| 4 | dao | Khi có yêu cầu xem chi tiết sản phẩm thì cần sử dụng hàm getMediaById() để lấy thông tin sản phẩm. Với mỗi sản phẩm thì có một hàm getMediaById() khác nhau. | Khi thêm chức năng xem chi tiết mặt hàng, cần tạo ra các instance khác nhau cho mỗi loại sản phẩm. Nếu trong tương lai có thêm một loại sản phẩm mới (thêm audioBook như yêu cầu 1) thì sẽ phải sửa đổi code-> vi phạm OCP |

### LSP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Lý do* |
| 1 | controller | AuthenticationController extends từ BaseController. BaseController có 2 phương thức checkMediaInCard() và getListCartMedia() nhưng AuthenticationController lại không sử dụng | Không sử dụng phương thức của lớp cha |
| 2 |  |  |  |

### ISP

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

### DIP

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

## Các vấn đề về Clean Code

### Clear Name

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Vấn đề* | *Giải quyết* |
| 1 | subsystem. InterbankSubsystem | Sử dụng từ viết tắt để đặt tên biến | Thay thế tên biến từ ctrl thành interbankSubsystemController |
| 2 | views.screen. BaseScreenHandler | Sử dụng từ viết tắt để đặt tên biến, tên hàm | Thay thế tên biến bController thành baseController, tên hàm getBController thành getBaseController, tên hàm setBController thành setBaseController |
| 3 | views.screen.home. LoginScreenHandler,  views.screen.home. HomeScreenHandler,  views.screen.cart. CartScreenHandler, views.screen.shipping. ShippingScreenHandler,  views.screen.invoice. InvoiceScreenHandler,  views.screen.payment. PaymentScreenHandler | Các lớp này kế thừa từ class BaseScreenHandler nên gặp phải vấn đề về đặt tên hàm | Thay thế tên hàm getBController thành getBaseController |

### Clean Function/Method

- Trong class controller.PaymentController, phương thức getExpirationDate đang ở mức 2 level of abstraction

- Trong class utils.MyMap, phương thức getNextTerm đang ở mức 2 level of abstraction

- Trong class subsystem.interbank.InterbankPayloadConverter, tại phương thức extractPaymentTransaction có vi phạm Data-Level Refactoring, ta có thể thay các mã code bằng các hằng số được định nghĩa sẵn

- Trong class entity.shipping.DeliveryInfo, tại phương thức calculateShippingFee có vi phạm Data-Level Refactoring, ta có thể thay hệ số bằng một hằng số định nghĩa sẵn

- Trong class views.screen.popup.PopupScreen, tại phương thức show có vi phạm Data-Level Refactoring, ta có thể thay hệ số bằng một hằng số định nghĩa sẵn

### Clean Class

- Trong class controller.PlaceOrderController bên cạnh các phương thức đặt hàng còn chứa các phương thức xác thực: validateDeliveryInfo, validatePhoneNumber, validateName và validateAddress. Ở đây ta nên tách thành một class Validate riêng.

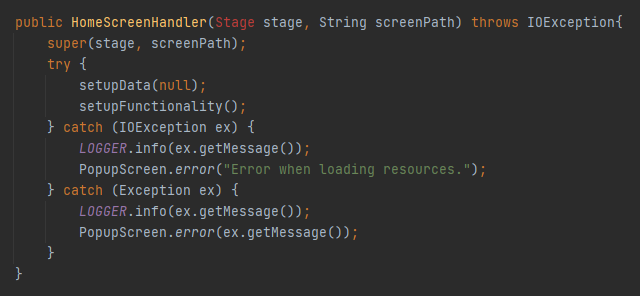
- Trong class controller.AuthenticationController bên cạnh các phương thức authentication còn chứa các phương thức mã hoá md5. Ở đây ta nên tách thành một class để mã hoá md5 riêng.

## Các vấn đề khác

**2.5.1. Áp dụng Template Method**

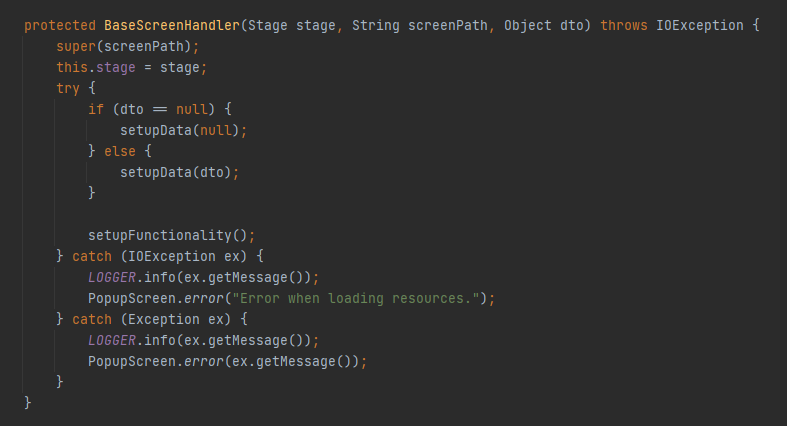
Trong các phương thức khởi tạo của các class ScreenHandler nói chung đều gồm chung một luồng thực hiện trong khối try catch là thực hiện mặc định chức năng setupData, setupFunctionality sau đó sẽ bắt các lỗi này bằng cách dùng PopupScreen.

Ví dụ: Constructor của class HomeScreenHandler (views.screen.home.HomeScreenHandler)



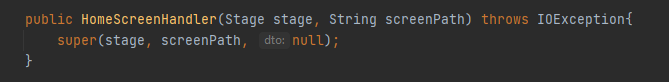
Vì các class ScreenHandler đều kế thừa từ class BaseScreenHandler cho nên tại phương thức khởi tạo Constructor của BaseScreenHandler chúng ta sẽ tạo một template gọi luồng thực thi chung tổng quát cho các ConcreteClass, và các hàm setupData, setupFunctionality sẽ implement theo cách riêng trong các ConcreteClass cụ thể.

Implement cho constructor của super class BaseScreenHandler.



Constructor của các concrete class bây giờ không còn phức tạp và bị duplicate nữa.

Ví dụ: Constructor của class HomeScreenHandler (views.screen.home.HomeScreenHandler) sau khi áp dụng template method



**2.5.2. Có thể sử dụng BuilderPattern cho các Media cụ thể**

Do các concrete Media (Book,CD,DVD) hàm constructor nhận vào rất nhiều tham số, mỗi khi khởi tạo có sẽ truyền vào phải truyền theo một thứ tự và cũng không biết tham số này sẽ cho thuộc tính nào.

Vì vậy để tăng tính dễ đọc của code, có thể áp dụng BuilderPattern vào trong các class này. Giúp cho mỗi khi khởi tạo một đối tượng chúng ta có thể tạo theo thuộc tính từng loại rõ ràng chứ không nhất thiết phải theo một thứ tự.

# Đề xuất cải tiến

Vấn đề về coupling

Common coupling : Các phương thức sử dụng trực tiếp các thuộc tính static của SessionInformation.

Giải pháp: áp dụng mẫu Singleton, set private cho các thuộc tính và cung cấp phương thức getter/setter.

## Vấn đề SRP#1 và giải pháp

- PlaceOrderController trong module controller vừa thực hiện tạo đơn hàng, vừa validate thông tin => vi phạm SRP.

- Giải pháp: tách chức năng validate thông tin thành một class Validate rồi đưa vào module utils.

## Vấn đề SRP#2 và giải pháp

- AuthenticationController vừa thực hiện xác thực đăng nhập, vừa thực hiện mã hoá md5=> vi phạm SRP

- Giải pháp: tách chức năng mã hoá md5 sang một class chuyên về mã hoá: Cipher trong utils

## Vấn đề OCP#1 và giải pháp

- PaymentController trong module controller cần sửa đổi code khi thêm phương thức thanh toán, cụ thể là method payOrder() cần được sửa đổi.

- Giải pháp: sử dụng Factory Method . (xem mục 3.8)

## Vấn đề OCP#2 và giải pháp

- DeliveryInfor trong entity là class mô tả thông tin về đơn giao hàng, vận chuyển, trong đó có calculateShippingFee() dùng để tính phí giao hàng. Khi thay đổi phương thức tính khoảng cách thì cần sửa đổi code.

- Giải pháp: xem mục 3.7

## Vấn đề OCP#3 và giải pháp

- extractPaymentTransaction() của InterbankPayloadConverter trong subsystem cần sửa đổi code khi thêm phương thức thanh toán mới.

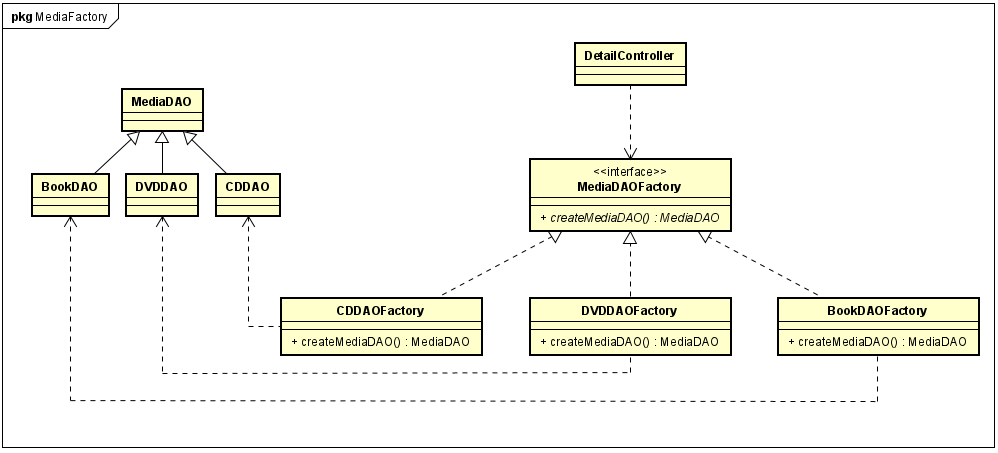
Giải pháp: xử lí cùng lúc với OCP#1

## Vấn đề OCP#4 (requirement 1: Thêm sản phẩm mới, requirement 2: thêm màn hình xem chi tiết sản phẩm) và giải pháp

Cách giải quyết: Sử dụng Factory Method để thiết kế lại:

* Tạo interface MediaDAOFactory chứa abstract method createMediaDAO()
* Tạo ra các concrete Factory implement MediaDAOFactory rồi override createMediaDAO()
* Thêm một class DetailController trong module controller để xử lý nhiệm vụ xem chi tiết sản phẩm.

Như vậy, khi thêm một sản phẩm mới chỉ cần tạo concrete Factory mới mà không cần sửa đổi code cũ



Hình : Áp dụng Factory Method cho yêu cầu thêm sản phẩm mới cũng như tạo trang xem chi tiết sản phẩm

## Thay đổi cách tính khoảng cách, sử dụng thư viện mới (requirement 4)

Yêu cầu: Thay vì sử dụng thư viện cũ DistanceCalculator, ta sử dụng một thư viện mới là

alt-distance-api.jar có interface gần giống với distance-api.jar

Cách giải quyết: Sử dụng Adapter Pattern

- Tạo interface DistanceCalculatorInterface chứa phương thức calculateDistance(String province, String address)

- Tạo 2 Adapter: AdapterDistanceCalculator (sử dụng thư viện distance-api.jar) và AdapterAlternativeDistanceCalculator (sử dụng thư viện alt-distance-api.jar)

- Cho 2 Adapter trên implements DistanceCalculatorInterface và ghi đè phương thức calculateDistance của DistanceCalculatorInterface như sau:

* Đối với AdapterDistanceCalculator thì sẽ trả về kết quả là distanceCalculator.calculateDistance(address, province)
* Đối với AdapterAlternativeDistanceCalculator thì sẽ trả về kết quả là alternativeDistanceCalculator.calculateDistance(province, address)

Diagram

Description automatically generated

Hình : Áp dụng Adapter Pattern cho yêu cầu về tính khoảng cách

## Vấn đề thêm phương thức thanh toán mới (requirement 5)

Vấn đề ta gặp phải khi thêm các phương thức thanh toán mới là mỗi phương thức thanh toán sẽ có các thuộc tính khác nhau.

Vì vậy ở đây ta sẽ sử dụng Factory Method để giải quyết vấn đề này.

Đầu tiên ta sẽ tạo 1 interface Card. Lớp con CreditCard (hoặc về sau là DomesticCard) sẽ implements interface Card này.

Tiếp theo ta sẽ tạo abstract class CardFactory chứa abstract function createCreditCard (về sau sẽ có thêm createDomesticCard). Ta tạo class CreditCardFactory kế thừa CardFactory, trong class này ghi đè phương thức createCreditCard từ CardFactory và trả về là CreaditCard với các thuộc tính của CreaditCard.

Diagram, timeline

Description automatically generated

Hình : Áp dụng Factory Method cho yêu cầu phương thức thanh toán mới

## Mở rộng của Requirement 7

Yêu cầu mở rộng: Muốn đơn hàng có khả năng cho phép khách hàng hủy bỏ. Nhưng một khi Admin đã phê duyệt thì không thể hủy bỏ đơn hàng của khách hàng nữa.

Yêu cầu trên đã đặt ra một thiết kế bổ sung cho class Order sử dụng StatePattern để thêm state vào đơn hàng.

Trạng thái đơn hàng được miêu tả như sau:

Khi khách hàng chưa thanh toán. Đơn hàng có trạng thái thái mặc định là DefaultState

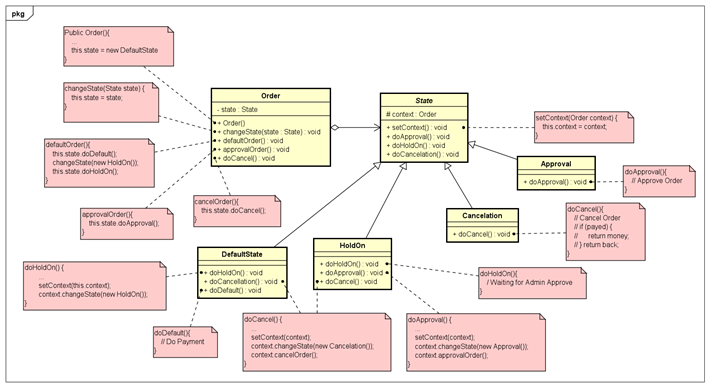
Sau khi khách hàng xác nhận các thủ tục chuyển phát và thanh toán. Đơn hàng có các trạng thái như sau:

Trạng thái chờ (HoldOn): Đơn hàng ở trạng thái chờ người quản trị xác nhận đơn hàng.

Trạng thái đã phê duyệt (Approval): Khi người quản trị phê duyệt đơn hàng xong, đơn hàng sẽ ở trạng thái đã phê duyệt. Lúc này đơn hàng sẽ tự động chuyển sang trạng thái xác nhận thành công (Success) và khách hàng không thể sửa đổi trạng thái của đơn hàng.

Trạng thái hủy đơn hàng (Cancellation): Khi người quản trị chưa thực hiện thao tác phê duyệt đơn hàng. Khách hàng có thể hủy bỏ đơn hàng để lấy lại tiền.

Class Diagram cho mẫu thiết kế StatePattern được thể hiện như hình dưới đây. (Code đã được implement vào source code)



Hình : Áp dụng State Pattern cho yêu cầu cập nhật chức năng huỷ đơn hàng

## Mở rộng của Requirement 6

Từ yêu cầu mở rộng muốn tính phí ship theo một công thức mới.

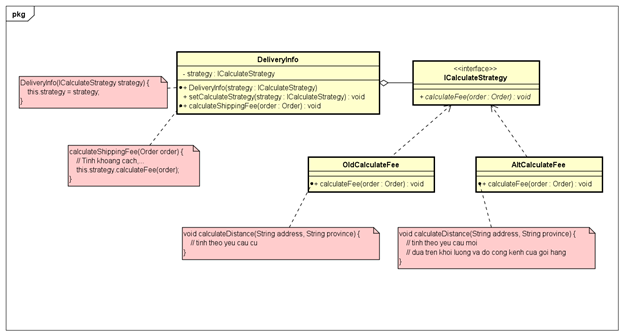
Thiết kế cũ:

Công thức tính phí ship chỉ dựa trên khoảng cách. Muốn sửa đổi chúng ta cần phải sửa trực tiếp vào trong phương thức calculateShippingFee() của class DeliveryInfo. Nếu làm vậy sẽ làm cho code bị vi phạm nguyên lý OCP (cụ thể là Close for modification). Hơn nữa, nếu mai này lại có yêu cầu thay đổi chúng ta lại phải sửa đổi lại rất mất thời gian.

Thiết kế mới

Áp dụng mẫu thiết kế Strategy Pattern

Chúng ta có thể lựa chọn công thức tính bất kỳ chúng ta muốn, kể cả chuyển đổi cách tính công thức trong quá trình runtime để lựa chọn theo từng loại đơn hàng chẳng hạn



Hình : Áp dụng Strategy Pattern cho yêu cầu thay đổi cách tính phí vận chuyển

# Tổng kết

## Kết quả tổng quan

Mã nguồn đã được tinh chỉnh đảm bảm các nguyên lý SOLID, Clean Code và giải quyết được các yêu cầu đã đề ra

## Các vấn đề tồn đọng

Không có.