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 15

Danh sách thành viên: Nguyễn Minh Tuấn 20173436

Bùi Minh Tuấn 20173444

Nguyễn Văn Tuấn 20173440

Phạm Mạnh Tuấn 20173435

*Hà Nội, tháng 06 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 3

1.4 Danh sách tham khảo 3

2 Đánh giá thiết kế cũ 4

2.1 Nhận xét chung 4

2.2 Đánh giá các mức độ coupling và cohesion 4

2.2.1 Coupling 4

2.2.2 Cohesion 4

2.3 Đánh giá việc tuân theo SOLID 4

2.3.1 SRP 5

2.3.2 OCP 5

2.3.3 ... 5

2.4 Các vấn đề về Clean Code 5

2.4.1 Clear Name 5

2.4.2 Clean Function/Method 5

2.4.3 Clean Class 5

2.5 Các vấn đề khác 6

3 Đề xuất cải tiến 7

3.1 Vấn đề xxx và giải pháp 7

3.2 Vấn đề yyy và giải pháp 7

4 Tổng kết 8

4.1 Kết quả tổng quan 8

4.2 Các vấn đề tồn đọng 8

**Phân chia công việc**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thành viên** | **Công việc** |
| 1 | Nguyễn Văn Tuấn | Cohesion/Coupling |
| 2 | Bùi Minh Tuấn | Clean Code |
| 3 | Nguyễn Minh Tuấn | SOLID (tìm vấn đề) |
| 4 | Phạm Mạnh Tuấn | SOLID(giải pháp áp dụng Design Pattern) |

# Tổng quan

## Mục tiêu

Mục đích của báo cáo là chỉ ra những vi phạm trong thiết kế của mã nguồn. Người sử dụng báo cáo là lập trình viên, qua báo cáo này họ có thể xác định các vấn đề trong thiết kế của mã nguồn và cải tiến thiết kế của mã nguồn. Thiết kế của mã nguồn sau khi cải tiến cần đạt được low coupling và high cohesion, hạn chế vi phạm các nguyên lý SOLID và sử dụng các Design Pattern cần thiết

Nội dung báo cáo gồm 3 phần:

· Phần 1 Tổng quan: Mục nội dung sau mô tả khái quát nội dung báo cáo

· Phần 2 Đánh giá thiết kế cũ: Mục này bao gồm kết quả code review trên mã nguồn case study

· Phần 3 Đề xuất cải tiến: Mục này đưa ra những đề xuất cần cải tiến để khắc phục những vấn đề trên nếu có những yêu cầu phát sinh

## Phạm vi

<

*Trong mục này, cần làm rõ:*

1. *Mô tả khái quát phần mềm*

*AIMS Project là một hệ thống đa nền tảng hoạt động 24/7, cho phép người dùng mới có thể làm quen dễ dàng. Hệ thống này có thể cho phép phục vụ 1000 khách hàng cùng lúc mà hiệu suất không bị giảm đáng kể, đồng thời có thể hoạt động 300 giờ liên tục không hỏng hóc. Ngoài ra, hệ thống có thể hoạt động trở lại bình thường trong vòng 1 giờ sau khi xảy ra lỗi. Thời gian đáp ứng của hệ thống tối đa là 1 giây khi bình thường hoặc 2 giây lúc cao điểm.*

*Trong hệ thống thương mại điện tử AIMS Project, quản trị viên có thể thêm, xem, sửa, xóa bất kỳ sản phẩm nào. Tuy nhiên, quản trị viên chỉ có thể thêm hoặc sửa với một sản phẩm tại một thời điểm, nhưng lại có thể xóa tới 10 sản phẩm cùng một lúc. Ngoài ra, quản trị viên không thể xóa hoặc cập nhật quá 30 sản phẩm ví lý do bảo mật nhưng có thể thêm không giới hạn số sản phẩm trong một ngày.*

*Về phía khách hàng, khi khởi động, hệ thống sẽ hiện ra danh sách của 20 sản phẩm bất kỳ ở mỗi trang. Để tìm kiếm sản phẩm, khách hàng sử dụng các đặc tính của sản phẩm để tìm kiếm. Hệ thống sẽ hiện ra 20 sản phẩm liên quan trong mỗi trang tìm kiếm. Bên cạnh đó, khách hàng có thể sắp xếp sản phẩm theo giá cả hoặc có thể thêm sản phẩm với số lượng tương ứng vào giỏ hàng (cart) hiện tại.*

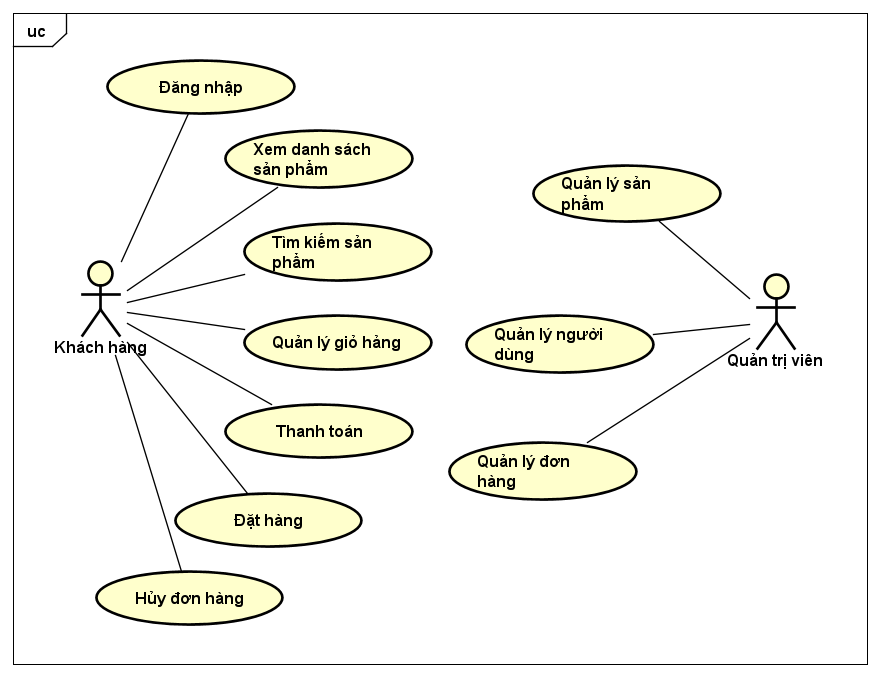
*Khi xem giỏ hàng, hệ thống sẽ hiện ra thông tin giỏ hàng, bao gồm tổng giá cả sản phẩm chưa bao gồm VAT, tổng giá cả sản phẩm đã bao gồm VAT, danh sách sản phẩm với thông tin sản phẩm (tên sản phẩm, số lượng, và giá cả). Đồng thời, hệ thống cũng thông báo tới khách hàng nếu số lượng hàng tồn trong kho bất kỳ sản phẩm không đủ và sẽ hiện ra số lượng này của từng sản phẩm bị thiếu. Khi thay đổi ý định, khách hàng cũng có thể bỏ sản phẩm khỏi giỏ hàng. Ngoài ra, chỉ có 1 giỏ hàng trong mỗi phiên bản chạy của hệ thống, đồng thời, giỏ hàng sẽ được làm trống sau khi thanh toán đơn hàng thành công. Từ giao diện xem giỏ hàng này, khách hàng có thể yêu cầu đặt hàng.*

*Trong AIMS Project, quá trình đặt hàng trải qua 5 bước chính: (1) đặt hàng (order placement), (2) thanh toán (order payment), (3) duyệt đơn hàng (order approval), (4) vận chuyển (delivery), và (5) hỗ trợ sau đặt hàng (post order support)*

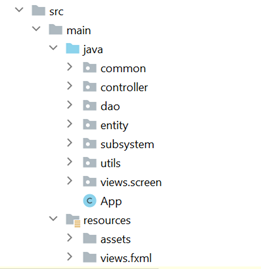
1. *Các chức năng chính của phần mềm*

* Khách hàng
* Đăng nhập
* 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

Biểu đồ usecase tổng quát:



1. *Cấu trúc mã nguồn*

**

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

*1. Thêm mặt hàng Media mới: AudioBook*

*2. Thêm màn hình: Xem chi tiết sản phẩm*

*3. Thay đổi yêu cầu khi load giao diện*

*4. Thay đổi cách tính khoảng cách, sử dụng thư viện mới*

*5. Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)*

*6. Thay đổi công thức tính phí vận chuyển*

*7. Cập nhật lại chức năng hủy đơn hàng*

1. *Các hoạt động (Ví dụ: review, refactor) thực thi trên mã nguồn để đạt được mục tiêu kể trên*

*· Review mã nguồn, tìm ra những nơi có mức độ coupling, cohesion chưa hợp tốt*

*· Tìm ra các nơi vi phạm nguyên lý SOLID trong mã nguồn*

*· Tìm ra các code smell trong mã nguồn*

*· Tìm ra các module có thể áp dụng Design Pattern để đáp ứng các yêu cầu được bỏ sung trong tương lai*

1. *Kết quả dự kiến*

· Tìm được các vấn đề, vi phạm trong mã nguồn và giải pháp cho chúng

· Tiến hành tái cấu trúc mã nguồn để thu được mã nguồn mới tối ưu hơn

## Danh sách thuật ngữ

## 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 case study hiện tại còn chưa tốt, hiệu năng ứng biến của code đối với các kế hoạch update trong tương lai còn kém

## Đánh giá các mức độ coupling và cohesion

Coupling đề cập đến vấn đề phụ thuộc lẫn nhau giữa các component. Low coupling, loose coupling có nghĩa là các component ít phụ thuộc vào nhau, sự thay đổi trong component này ít khi, hoặc không ảnh hưởng đến component kia. Ngược lại, high coupling và tight coupling cho thấy các component phụ thuộc nhiều vào nhau, khi thay đổi 1 component thì các component kia đều bị ảnh hưởng và có khả năng phải thay đổi theo. Low coupling là mục tiêu chúng ta cần hướng đến để đảm bảo cho hệ thống ít bị ảnh hưởng khi có thay đổi và do đó, tăng tốc độ thực hiện công việc và bảo trì.

Cohesion thể hiện mối quan hệ bên trong module. Nhiệm vụ của từng module càng rõ ràng và tách biệt thì cohesion càng cao (high cohesion), và đó là mục tiêu cần đạt tới khi thiết kế. Khi hệ thống đạt được high cohesion thì hệ thống sẽ có khả năng phát triển, mở rộng và bảo trì cực cao.

### Coupling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Coupling* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Stamp coupling | function checkMediaInCart trong Cart.java | Vì chỉ sử dụng 1 thuộc tính id trong biến media truyền vào | Khó tái sử dụng, có thể gây ra tác dụng phụ không mong muốn |
| 2 | Common Coupling | Function checkMediaInCart và getListCardMedia trong BaseController.java | Sử dụng chung biến cartInstance của lớp SessionInformation | Gây ảnh hưởng đến việc kiểm soát dữ liệu, khó xác định được tất cả các thành phần tác động đến dữ liệu. Tái sử dụng kém |
| 3 | Common coupling | Function Order trong Order.java | Sử dụng chung biến cartIntance của lớp SessionÌnomation | Gây ảnh hưởng đến việc kiểm soát dữ liệu, khó xác định được tất cả các thành phần tác động đến dữ liệu. Tái sử dụng kém |
| 4 | Stamp coupling | Function refund trong InterbankSubsystemController.java | Vì biến card không được sử dụng | Khó tái sử dụng, có thể gây ra tác dụng phụ không mong muốn |
| 5 | Content coupling | Function login trong AuthenticationController.java | Thay đổi trực tiếp dữ liệu lớp SessionInformation (mainUser) | Vi phạm tính đóng gói, có thể gây ra kết quả không mong muốn |
| 6 | Control coupling | Function validateDeliveryInfo trong PlaceOrderController.java | Do biến info điều khiển luồng của module được gọi | Khó hiểu và khó  bảo trì. Tái sử dụng  kém |
| 7 | Stamp coupling | Function caculateShippingDee trong DeliveryInfo.java | tham số order không được sử dụng | Khó tái sử dụng, có thể gây ra tác dụng phụ không mong muốn |

### Cohesion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Các mức độ về Cohesion* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Coincidental cohesion | Function getToday trong InterbankPayloadConverter | phương thức này không liên quan đến các phương thức khác trong class | Khó bảo trì và tái sử dụng |
| 2 | Communicational cohesion | getMediaById trong MediaDAO.java | vì getAllMedia và getMediaById cùng trả về 1 dữ liệu đầu ra | Tính tái sử dụng thấp |
| 3 | Procedural cohesion | updateMediaFieldById trong MediaDAO.java | vì phương thức updateMediaById được nhóm lại | Tính tái sử dụng thấp |
| 4 | Coincidental cohesion | các phương thức validate trong PlaceOrderController | vì các phương thức này có chức năng khác so với chức năng của lớp | Khó bảo trì và tái sử dụng |
| 5 | Coincidental cohesion | phương thức setImage trong FXMLScreenHandler | vì phương thức này không liên quan đến các phương thức còn lại của class | Khó bảo trì và tái sử dụng |
| 6 | Communicational cohesion | phương thức setDeliveryInfo trong DeliveryInfo | các lệnh trong phương thức này chỉ liên quan đến nhau vì sử dụng chung loại dữ liệu | Tính tái sử dụng thấp |

## Đánh giá việc tuân theo SOLID

### SRP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Lớp PlaceOrderController | Lớp làm 2 nhiệm vụ: xừ lí Order và validate các thông tin của Order | Chưa tuân thủ mỗi 1 lớp chỉ có 1 nhiệm vụ, gây khó khăn cho việc quản lí thêm các yêu cầu mới |
| 2 | Lớp PaymentController | Lớp làm 2 nhiệm vụ: xử lý thanh toán và kiểm tra thẻ hết hạn | Chưa tuân thủ mỗi 1 lớp chỉ có 1 nhiệm vụ, gây khó khăn cho việc quản lí thêm các yêu cầu mới |

### OCP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Phương thức PayOrder trong PaymentController | Phương thức này chứa các thông tin của Order, | khi thay đổi phương thức thanh toán sẽ phải sửa lại code |
| 2 | Phương thức caculatorShipFee trong DeliveryInfo | Phương thức nhận tham số là order và trả về giá ship | Nếu sau này sử dụng thư viện khác để tính khoảng cách thì sẽ phải sửa đổi trực tiếp mã nguồn |
| 3 | Phương thức extractPaymentTransaction trong IntebankPayloadInverter | Phương thức nhận response và trích rút thông tin và trả về 1 transaction | Nếu sau này thêm thông tin khi thay đổi cách thức, phương thức thanh toán |

### LSP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Lớp AuthencationController | Lớp AuthencationController không sử dụng 2 phương thức của lớp cha BaseController | Khó quản lí và bảo trì code |
| 2 |  |  |  |

### ISP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 |  |  |  |
| 2 |  |  |  |

### DIP

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Module* | *Mô tả* | *Ảnh hưởng* |
| 1 | Phương thức PayOrder trong PaymentController | Phương thức này chứa các thông tin của Order,  khi thay đổi phương thức thanh toán sẽ phải sửa lại code | khi thay đổi phương thức thanh toán sẽ phải sửa lại code |

## Các vấn đề về Clean Code

### Clear Name

- Thay thành các tên có ý nghĩ chính xác :

|  |  |  |
| --- | --- | --- |
| File thay đổi | Nội dung thay đổi | Lý do |
| PaymentController.java | private CreditCard card;  ð private CreditCard creditCard;    private InterbankInterface interbank;  ð private InterbankInterface interbankInterface; | Thuận tiện cho việc hiểu và bảo trì code |

- Tạo biến static final cho các hằng số sử dụng trong code:

|  |  |  |
| --- | --- | --- |
| File thay đổi | Nội dung thay đổi | Lý do |
| PlaceOrderController.java | int LENGTH\_MAX\_PHONENUMBER = 10;  String START\_PHONENUMBER\_CHARACTER = "0";  String PATTERN\_STRING = "^[a-zA-Z\\s]\*$"; | Tạo thành các biến có nghĩa, sau này muốn sửa đổi giá trị hằng số chỉ cần thay đổi giá trị của biến thay vì phải sửa đổi ở tất cả các nơi giá trị hằng số được sử dụng |
| Cart.java | int TOTAL\_INITIALIZATION = 0; |
| Order.java | int INITIALIZATION\_NUMBER = 0; |
| DeliveryInfo.java | double SHIPPING\_FEE = 1.2; |
| CartMediaHandler.java | int MEDIA\_INFO\_HEIGHT = 110;  int MEDIA\_INFO\_WIDTH = 92; |
| HomeMediaHandler.java | int IMAGE\_MEDIA\_HEIGHT = 160;  int IMAGE\_MEDIA\_WIDTH = 152; |
| AuthenticationController.java | int HOURS = 24; |
| PopupScreen.java | double AUTO\_CLOSE\_TIME = 0.8; |

### Clean Function/Method

|  |  |  |
| --- | --- | --- |
| Method/File | Nội dung thay đổi | Lý do |
| getMainUser()  AuthenticationController.java | Tạo các biến boolean để sử dụng trong hàm if |  |
| validateDeliveryInfo()  PlaceOrderController.java | Tạo các biến boolean để sử dụng trong hàm if |  |
| payOrder()  PaymentScreenHandler.java,  PaymentController.java | Tạo ra đối tượng payOrderInfo thay vì truyền nhiều tham số vào phương thức payOrder | Tránh tình trạng 1 method có quá nhiều tham số truyền vào |
| requestToPlaceOrder()  CartScreenHandler.java | Thêm phương thức getDisplayShippingForm xử lý các hành động liên quan đến shipping form handler |  |
| validatePhoneNumber(String phoneNumber)  validateName(String name)  validateAddress(String adrress)  PlaceOrderController.java | Tạo lớp ValidatorUtils để sử lý validate |  |

### Clean Class

* Tạo thêm interface Card, class CreditCard sẽ thực thi interface Cart. Các module trước đây giao tiếp với class CreditCard giờ sẽ giao tiếp với interface Cart
* Tạo thêm lớp MfDate chứa các phương thức liên quan đến Date (getToday)

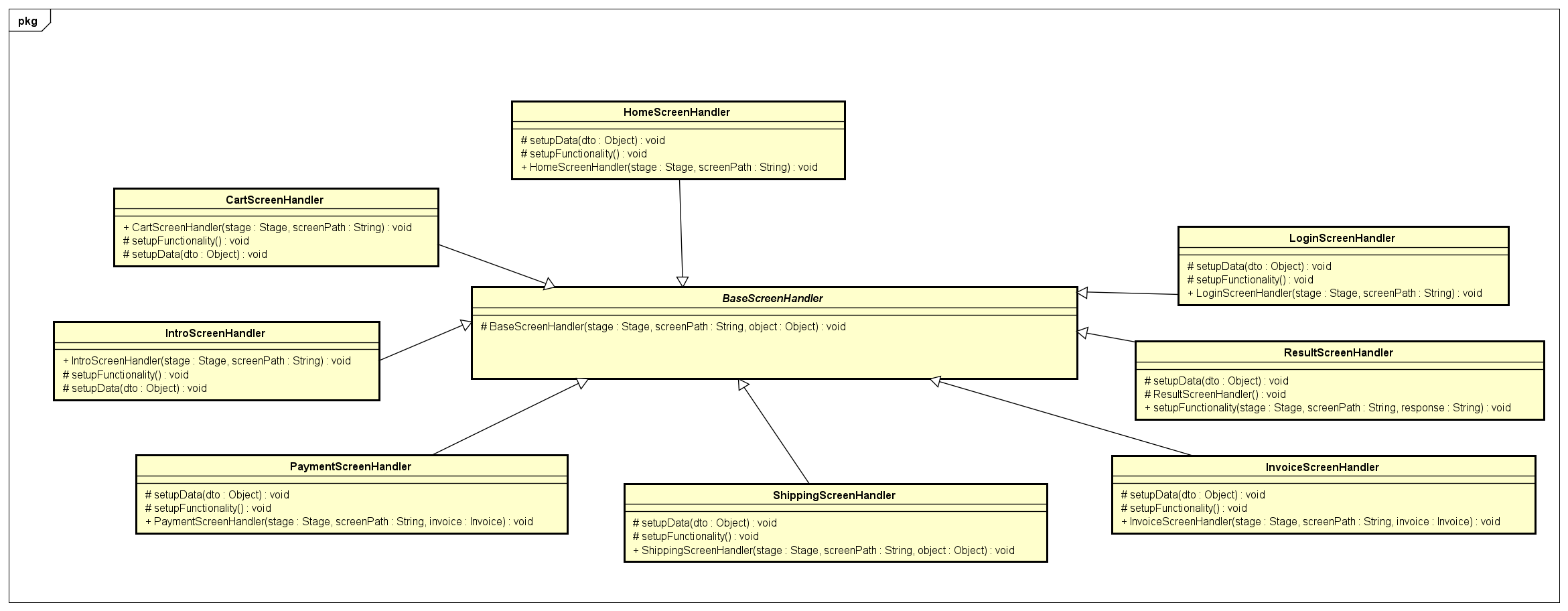
## Các vấn đề khác

* Vấn đề: Các lớp con của lớp BaseScreenHandler có phương thức khởi tạo gần giống nhau, nhưng lại không có phương thức chung ở lớp cha, do đó khi muốn thêm một màn hình mới sẽ phải lặp lại phương thức khởi tạo này dẫn đến lặp code.
* Vấn đề: Khi có yêu cầu mới cập nhật lại chức năng hủy đơn hàng

# Đề xuất cải tiến

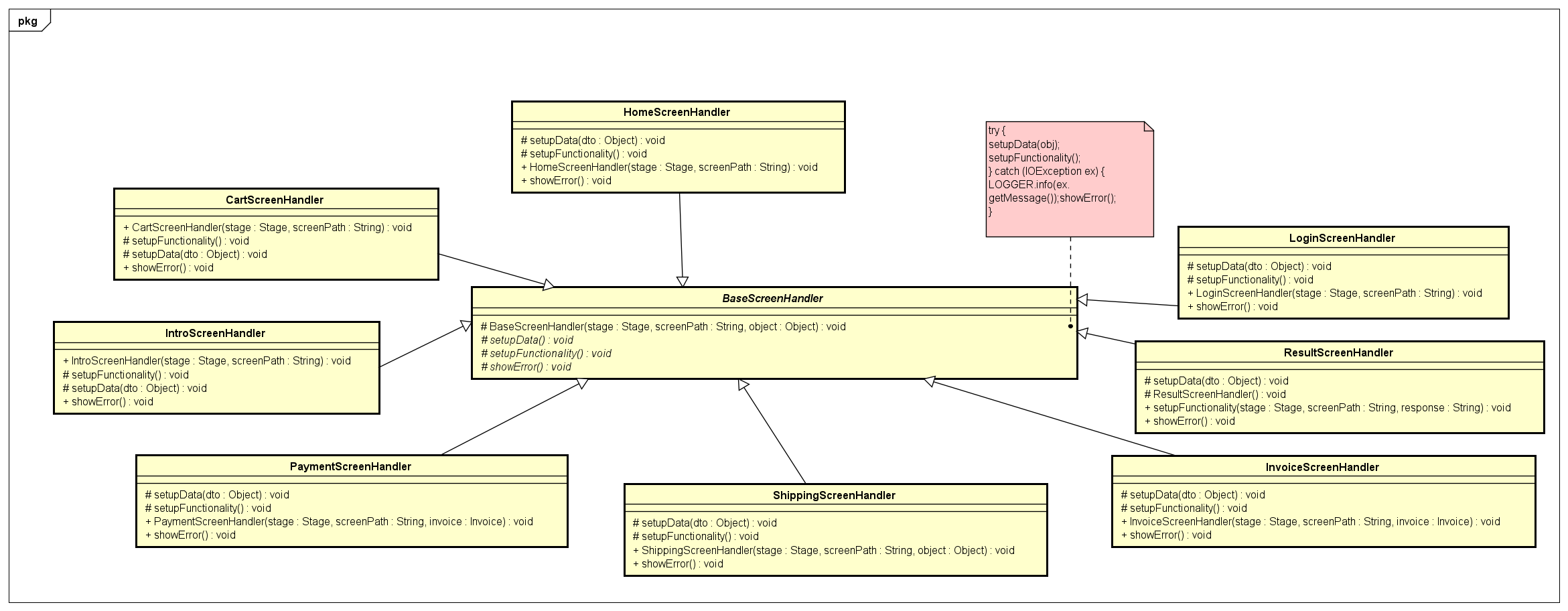
## Vấn đề lặp code trong các lớp con của lớp BaseScreenHandler và giải pháp

Vấn đề: Các lớp con của lớp BaseScreenHandler có phương thức khởi tạo gần giống nhau, nhưng lại không có phương thức chung ở lớp cha, do đó khi muốn thêm một màn hình mới sẽ phải lặp lại phương thức khởi tạo này dẫn đến lặp code.



Giải pháp: sử dụng Template Method

Tạo phương thức khởi tạo chung và các phương thức abstract(setupData, setupFunctionality và showError) ở lớp cha (lớp BaseScreenHandler). Các lớp con sẽ kế thừa phương thức khởi tạo và override lại các phương thức abstract



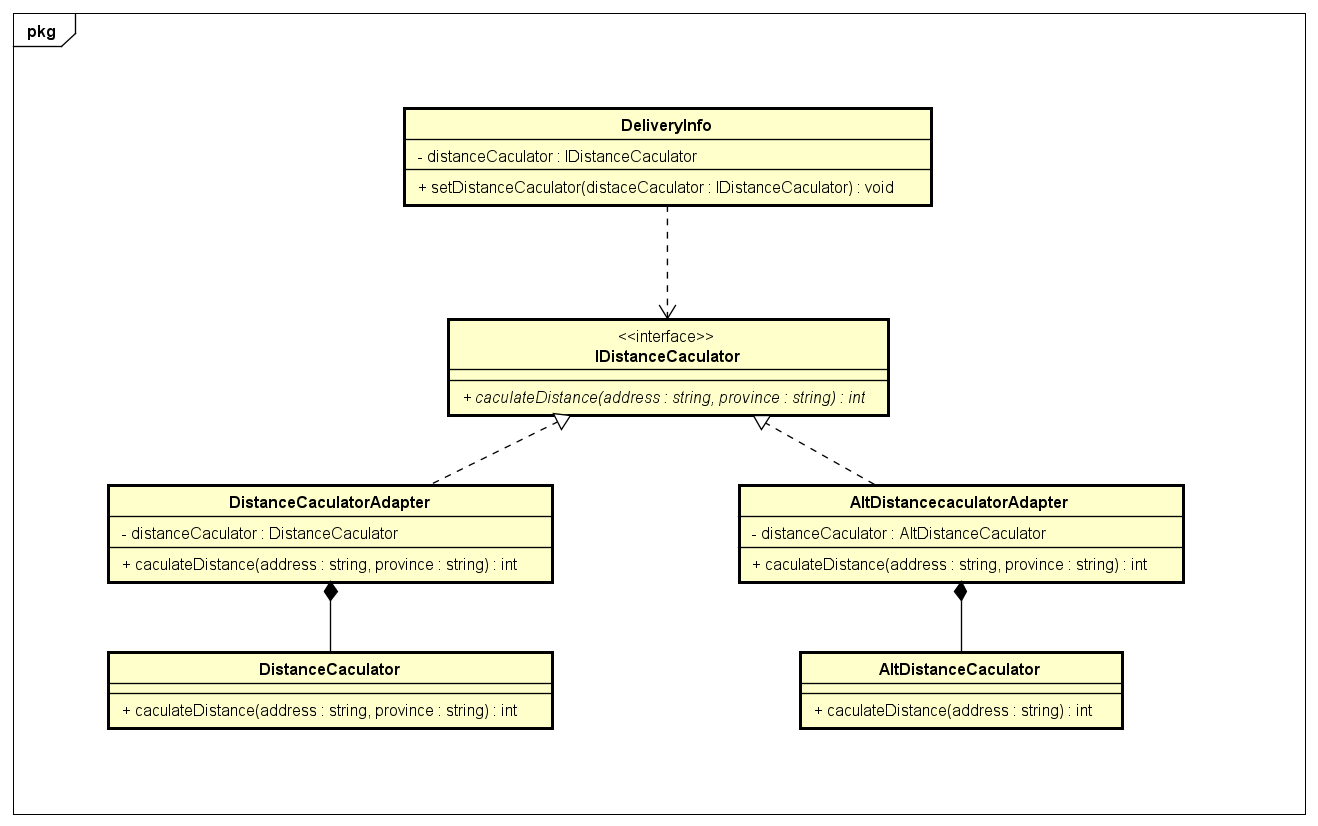
## Vấn đề vi phạm OCP khi thay đổi cách tính khoảng cách(sử dụng thư viện mới) và giải pháp

Vấn đề: Khi muốn sử dụng thư viện mới sẽ phải thay đổi trực tiếp code của lớp DeliveryInfo

Ảnh có chứa bàn

Mô tả được tạo tự động

Giải pháp: Sử dụng Adapter



Minh họa:

**public** **class** DeliveryInfo{

**protected** IDistanceCaculator distanceCalculator;

**public** **void** setDistanceCaculator (IDistanceCaculator caculateDistance) {

**this**.distanceCalculator = caculateDistance;

}}

**public** **interface** IDistanceCaculator {

**int** caculateDistance(String address,String province);

}

**public** **class** DistanceCaculatorAdapter **implements** IDistanceCaculator {

**private** DistanceCalculator distanceCalculator;

@Override

**public** **int** caculateDistance(String address, String provice) {

**return** distanceCalculator.calculateDistance(address, provice);

}

}

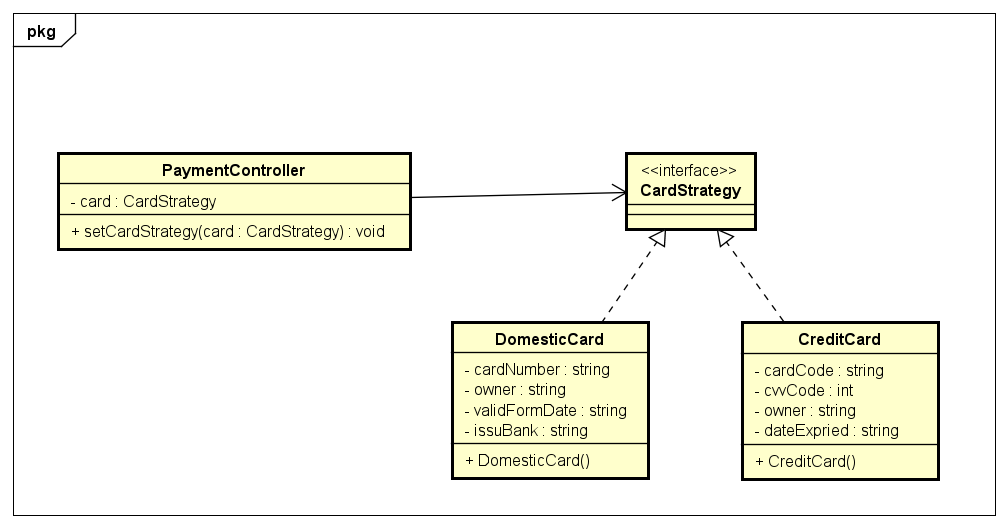
## Vấn đề vi phạm OCP khi thêm phương thức thanh toán mới và giải pháp

Vấn đề: Khi thêm phương thức thanh toán mới sẽ phải trực tiếp sửa code của lớp PaymentController

Ảnh có chứa bàn

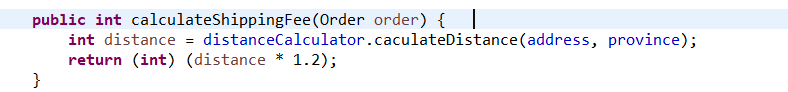
Mô tả được tạo tự động

Giải pháp: dùng mẫu thiết kế Strategy

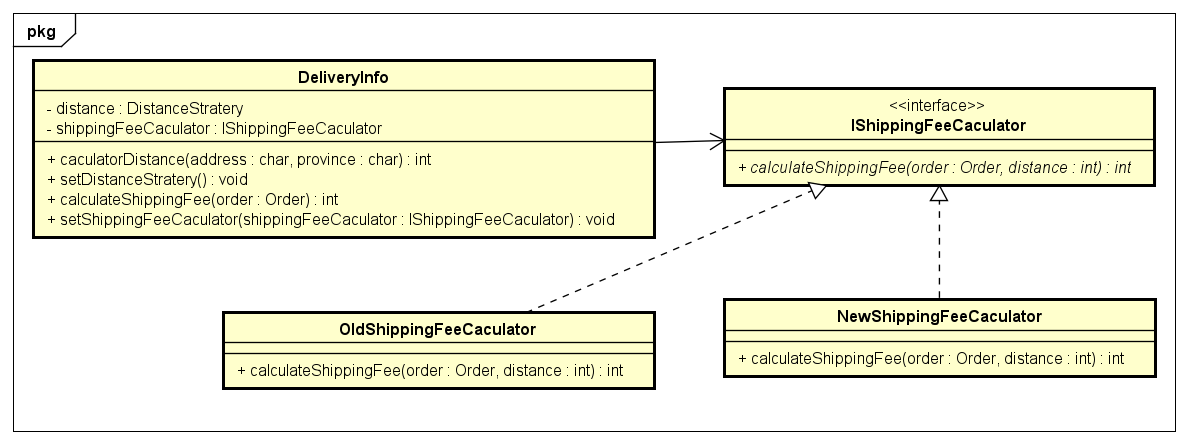


## Vấn đề vi phạm OCP khi thay đổi công thức tính phí vận chuyển và giải pháp

Vấn đề : Khi thay đổi công thức tính phí vận chuyển sẽ phải thay đổi trực tiếp code của lớp DeliveryInfo

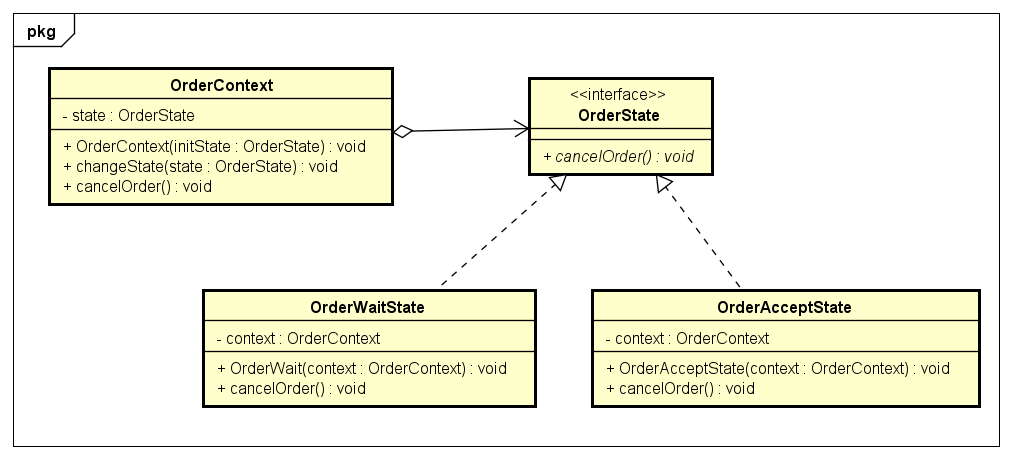


Giải pháp: dùng mẫu thiết kế Strategy



## Vấn đề cập nhật lại chức năng hủy đơn hàng và giải pháp

Giải pháp: dùng mẫu thiết kế State



Minh họa:

public class OrderContext {

OrderState state;

public OrderContext(OrderState initState) {

this.state = initState;

}

public void changeState(OrderState state) {

this.state = state;

}

public void cancelOrder () {

state.cancelOrder();

}

public interface OrderState() {

void cancelOrder();

}

public class OrderWaitState implement OrderState() {

OrderContext context;

public OrderWaitState (OrderContext context) {

this.context = context;

}

public void cancelOrder() {}

}

## Vấn đề vi phạm SRP tại lớp PlaceOrderController và giải pháp

Vấn đề: Lớp PlaceOrderController thực hiện nhiều chức năng khác nhau (vừa thực hiện chức năng đặt hàng vừa làm nhiệm vụ validate thông tin)

Giải pháp: Tách nhiệm vụ validate thông tin thành một lớp riêng (ValidatorUtils) và sử dụng mẫu thiết kế singleton cho lớp này vì lớp này chỉ cần khởi tạo duy nhất một lần

Minh họa:

**public** **class** ValidatorUtils { //Tao lop rieng de xu li validate

**private** **static** ValidatorUtils *singleton*;

**private** ValidatorUtils () {};

**public** **synchronized** **static** ValidatorUtils getInstance() {

**if**(*singleton* == **null**) *singleton* = **new** ValidatorUtils();

**return** *singleton*;

}

**public** **boolean** validatePhoneNumber(String phoneNumber) {

//Data-Level Refactoring/Introduce an intermediate variable

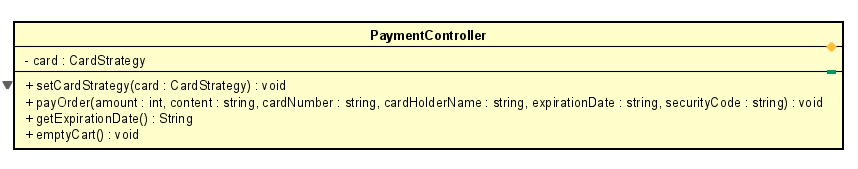
}

**public** **boolean** validateString(String name) {}

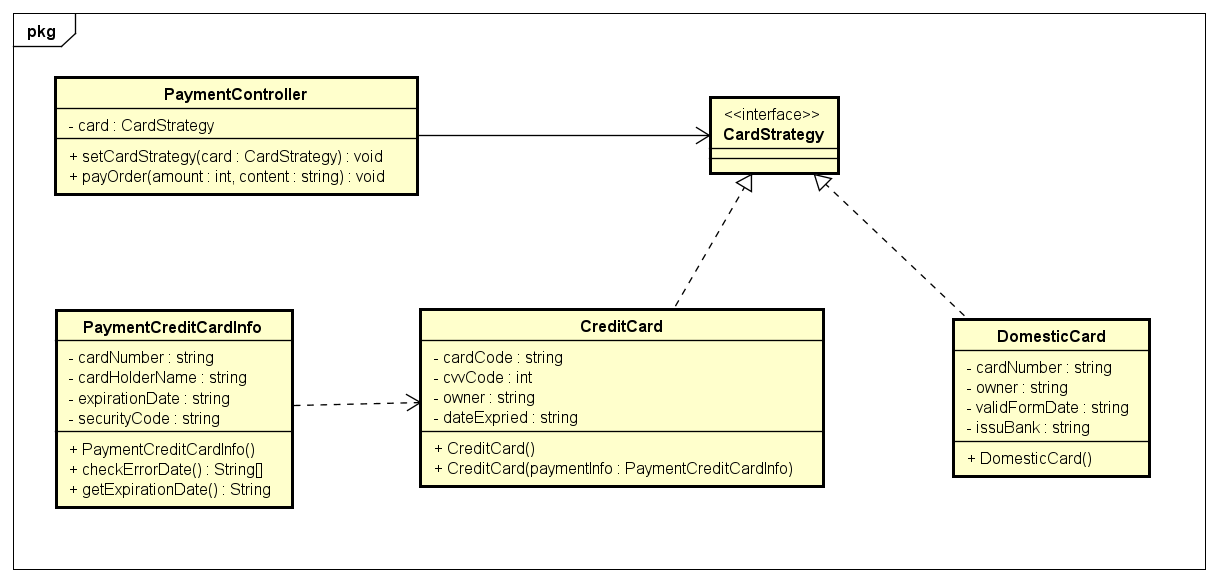
}

## Vấn đề vi phạm SRP tại lớp PaymentController và giải pháp

Vấn đề: lớp PaymentController vừa thực hiện chức năng thanh toán vừa kiểm tra ngày hết hạn của thẻ, hàm payOrder có quá nhiều tham số truyền vào



Giải pháp: Tạo ra lớp PaymentCreditCardInfo để chứa các thông tin khởi tạo thẻ CreditCard và làm nhiệm vụ kiểm tra ngày hết hạn của thẻ



# Tổng kết

## Kết quả tổng quan

Dự kiến ban đầu : Nhóm đề ra sẽ tìm ra tất cả cohesion, coupling.

Nhóm sẽ áp dụng các design pattern được học để xử lí và sửa lại toàn bộ những vi phạm trên. Tuy nhiên do thời gian có hạn, nhóm đã tìm ra được các vi phạm trong mã nguồn, đưa ra giải pháp cho một số vấn đề nhưng chưa xử lý hết. Các yêu cầu thêm vẫn còn chưa có giải pháp.

Kết quả : tuy còn nhiều chỗ chưa giải quyết nhưng mã nguồn sau khi được cải tiến cũng đã dễ dàng cho việc quản lí và mở rộng trong tương lai

## Các vấn đề tồn đọng

Requiemnt : Thêm mặt hàng Media mới : Chưa có giải pháp xử lí

Coupling : SessionInformation : Chưa có giải pháp xử lý