



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

BÁO CÁO MẪU THIẾT KẾ PHẦN MỀM

DP.20202.02

GVHD: TS. Nguyễn Thị Thu Trang
TS. Bùi Thị Mai Anh

Danh sách thành viên

Nhóm DP.20202.02

Họ tên	Mã số sinh viên
Vũ Quang Đại	20172993
Lê Minh Đức	20173043
Nguyễn Đình Đức	20173041
Nguyễn Mạnh Đức	20173039

Nội dung trình bày

1. Tổng quan

- Mục tiêu
- Phạm vi

2. Đánh giá thiết kế cũ

- Nhận xét chung
- Coupling & Cohesion
- SOLID
- Clean Code
- Các vấn đề khác

3. Đề xuất cải tiến

4. Tổng kết

- Kết quả thu được
- Vấn đề tồn đọng

1. Tổng quan



Tổng quan

Mục tiêu
Phạm vi

1.1 Mục tiêu

- Đánh giá thiết kế cũ một cách tổng quan và chi tiết từng phần, bao gồm:
 - Đánh giá về mức độ coupling, cohesion
 - Đánh giá việc tuân theo nguyên lý SOLID
 - Đánh giá mức độ clean code.
- Đưa ra một số đề xuất cải tiến cho các vấn đề gặp phải liên quan đến các mục đánh giá ở trên.

1.2 Phạm vi

Mô tả phần mềm:

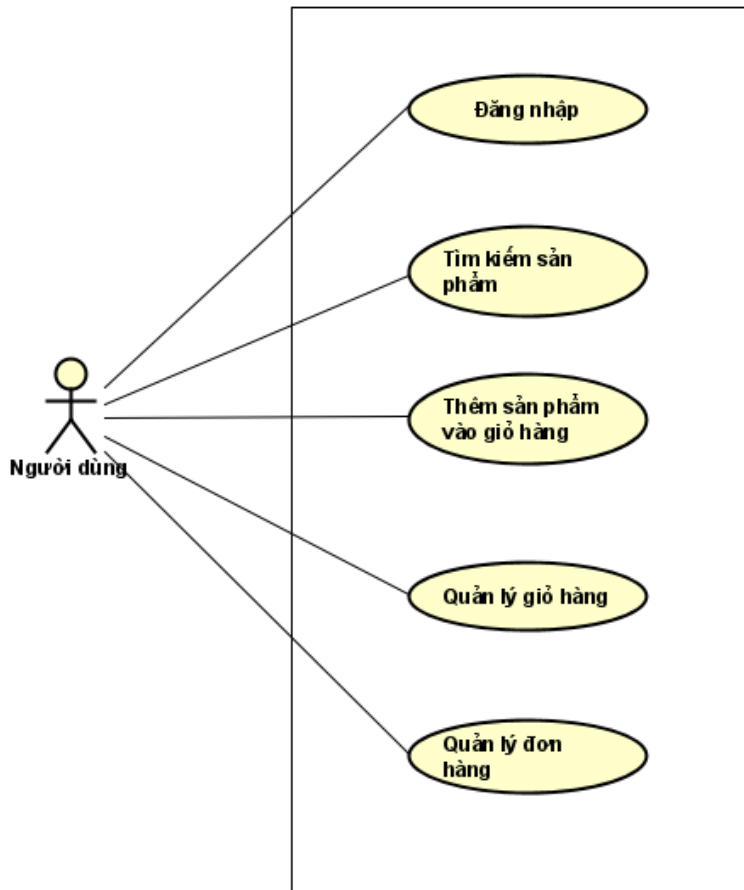
The screenshot displays the AIMS Project e-commerce interface. At the top, there is a navigation bar with the AIMS logo, a search bar, a shopping cart icon labeled '0 media', and a 'Login' button. Below the navigation bar, the main content area is a grid of 12 product listings, each with a cover image, title, price, availability, a quantity selector, and 'Add ...' and 'Detail' buttons.

Product ID	Product Name	Price	Avail
book2	The Secretariat of Remi	32 đ	12
book6	The Great Indian Novel	66 đ	12
book4	The Hitchhiker's Guide to the Galaxy	57 đ	12
dvd12	Mortal	82 đ	12
book9	Like Brothers	21 đ	12
cd7	Tax Collector	24 đ	12
cd3	CD (unlabeled)	66 đ	12
book3	Harry Potter and the Deathly Hallows	25 đ	12
book10	The Famous Five Big Book	73 đ	12
book12	Stephen King Misery	50 đ	12
book1	Sunset	79 đ	12
dvd10	Dead Still	75 đ	12

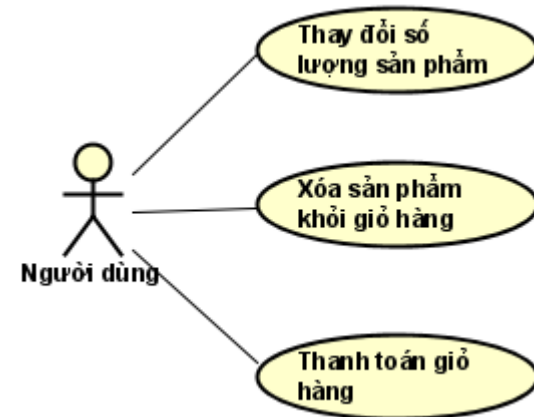
Hệ thống thương mại điện tử AIMS Project, cho phép mua bán sản phẩm phương tiện truyền thông, bao gồm sách, đĩa CD, đĩa DVD.

1.2 Phạm vi

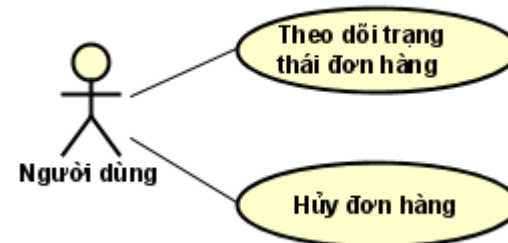
Biểu đồ use cases



Phân rã use case “Quản lý giỏ hàng”


















Phân rã use case “Quản lý đơn hàng”



1.2 Phạm vi

Cấu trúc mã nguồn

- >  src/main/java
 - ▼  > (default package)
 - >  > App.java
 - ▼  common
 - >  exception
 - >  interfaces
 - >  > controller
 - >  dao
 - >  entity
 - ▼  subsystem
 - >  interbank
 - >  InterbankInterface.java
 - >  InterbankSubsystem.java
 - >  utils
 - >  > views.screen

2. Đánh giá thiết kế cũ

2

Đánh giá
thiết kế cũ

Nhận xét chung

Coupling

Cohesion

SOLID

Clean Code

Các vấn đề khác

2.1 Nhận xét chung

Nhận xét chung:

- Chất lượng mã nguồn hiện tại ở mức khá, vẫn còn xuất hiện một số nơi có mức độ coupling cao và cohesion thấp.
- Một số nơi còn vi phạm nguyên lý SOLID cần phải tái cấu trúc lại cho phù hợp.
- Với các thay đổi trong tương lai: Một số yêu cầu (thay đổi cách tính chi phí vận chuyển, theo dõi đơn hàng) không cần thay đổi cấu trúc mã nguồn mà chỉ cần sử dụng một số cách và design pattern để thêm các tính năng phù hợp. Còn lại, hầu hết các yêu cầu mới đều cần phải thay đổi lại cấu trúc code base để có thể thêm tính năng phù hợp với yêu cầu mới.

2.2 Coupling

Đánh giá chung về mức độ Coupling

- Sự phụ thuộc lẫn nhau giữa các thành phần trong CodeBase là không quá lớn
- Đa số chỉ ở mức từ common đến stamp coupling

2.2.1 Common Coupling

Vị trí: entity.order.Order, method Order(Cart cart)
controller.PlaceOrderController, method createOrder()
views.screen.home.HomeScreenHandler, method update() và show()

Mô tả:

Các module này
đều cần truy cập
đến thông tin của
giỏ hàng qua
SessionInformation.cartInstance
không thuộc
module của mình.

```
public Order(Cart cart) {  
    List<OrderItem> orderItems = new ArrayList<>();  
    for (Object object : SessionInformation.cartInstance.getListMedia()) {  
        CartItem cartItem = (CartItem) object;  
        OrderItem orderItem = new OrderItem(cartItem.getMedia(),  
            cartItem.getQuantity(),  
            cartItem.getPrice());  
        orderItems.add(orderItem);  
    }  
    this.orderMediaList = Collections.unmodifiableList(orderItems);  
    this.subtotal = cart.calSubtotal();  
    this.tax = (int) (ViewsConfig.PERCENT_VAT/100) * subtotal;  
}
```

Lý do:

Các phương thức này truy cập đến SessionInformation.cartInstance của module khác.

2.2.2 Control Coupling

Vị trí: `views.screen.home.HomeScreenHandler`, method `update(MediaHandler mediaHandler)`
`views.screen.popup.PopupScreen`, method `popup(... Boolean undecorated)`

Mô tả:

Sử dụng tham số đầu vào điều khiển luồng hoạt động của một module khác

```
private static PopupScreen popup(String message, String imagePath,
                                  Boolean undecorated) throws IOException{
    PopupScreen popup = new PopupScreen(new Stage());
    if (undecorated) popup.stage.initStyle(StageStyle.UNDECORATED);
    popup.message.setText(message);
    popup.setImage(imagePath);
    return popup;
}
```

Lý do:

Phương thức `update` ở class `HomeScreenHandler` dùng tham số `mediaHandler` truyền vào để chia ra 2 trường hợp xử lý.

Phương thức `popup` ở class `PopupScreen` dùng tham số truyền vào `undecorated` để quyết định hành động.

2.2.3 Stamp Coupling

Vị trí: `entity.cart.Cart`, method `checkMediaInCart(Media media)`
`controller.PlaceOrderController`, method `setupData(Object dto)`
`views.screen.home.HomeScreenHandler`, method `validateDeliveryInfo(...)`

Mô tả:

Vì phạm xảy ra tại vị trí sử dụng các tham số truyền vào (`media`, `dto`, `info`)

```
public CartItem checkMediaInCart(Media media){  
    for (CartItem cartItem : lstCartItem) {  
        if (cartItem.getMedia().getId() == media.getId()) return cartItem;  
    }  
    return null;  
}
```

Lý do:

Các phương thức trên đều truyền vào một tham số lớn nhưng chỉ sử dụng đến một phần nhỏ của tham số đó

2.3 Cohesion

Đánh giá chung về mức độ Cohesion:

- Mức độ liên kết chặt chẽ giữa các thành phần trong CodeBase chưa cao.
- Mức độ coincidental cohesion còn xuất hiện nhiều.

2.3.1 Coincidental Cohesion

Vị trí: controller.AuthenticationController, md5(String message)

Mô tả:

Các yếu tố của thành phần hoàn toàn không liên quan với nhau, chỉ liên quan theo vị trí của chúng trong module

```
private String md5(String message) {  
    String digest = null;  
    try {  
        MessageDigest md = MessageDigest.getInstance("MD5");  
        byte[] hash = md.digest(message.getBytes(StandardCharsets.UTF_8));  
        // converting byte array to Hexadecimal String  
        StringBuilder sb = new StringBuilder(2 * hash.length);  
        for (byte b : hash) {  
            sb.append(String.format("%02x", b & 0xff));  
        }  
        digest = sb.toString();  
    } catch (NoSuchAlgorithmException ex) {  
        Utils.getLogger(Utils.class.getName());  
        digest = "";  
    }  
    return digest;  
}
```

Lí do:

Phương thức md5 không có chức năng liên quan đến class. Phương thức này nên đặt trong lớp tiện ích khác.

2.3.1 Coincidental Cohesion

Vị trí: controller.PaymentController, method getExpirationDate(String date)

Mô tả:

Phương thức getExpirationDate(String date) không liên quan đến class.

```
private String getExpirationDate(String date) throws InvalidCardException {
    String[] strs = date.split("/");
    if (strs.length != 2) {
        throw new InvalidCardException();
    }

    String expirationDate = null;
    int month = -1;
    int year = -1;

    try {
        month = Integer.parseInt(strs[0]);
        year = Integer.parseInt(strs[1]);
        if (validateDate(month, year)) {
            throw new InvalidCardException();
        }
        expirationDate = strs[0] + strs[1];
    } catch (Exception ex) {
        throw new InvalidCardException();
    }

    return expirationDate;
}
```

2.3.2 Logical Cohesion

Vị trí: dao.media.MediaDAO, method getAllMedia(), getMediaById()

Mô tả:

Các thành phần được nhóm lại vì chúng phù hợp với logic danh mục, các thành phần độc lập với nhau

Hai phương thức này đều là lấy thông tin nhưng không sử dụng dữ liệu của nhau.

```
public List getAllMedia() throws SQLException {
    Statement stm = AIMSDB.getConnection().createStatement();
    ResultSet res = stm.executeQuery("select * from Media");
    ArrayList medium = new ArrayList<>();
    while (res.next()) {
        Media media = new Media(
            res.getInt("id"),
            res.getString("title"),
            res.getInt("quantity"),
            res.getString("category"),
            res.getString("imageUrl"),
            res.getInt("price"),
            res.getString("type"));
        medium.add(media);
    }
    return medium;
}
```

```
public Media getMediaById(int id) throws SQLException {
    String sql = "SELECT * FROM Media ";
    Statement stm = AIMSDB.getConnection().createStatement();
    ResultSet res = stm.executeQuery(sql);

    if (res.next()) {
        return new Media(
            res.getInt("id"),
            res.getString("title"),
            res.getInt("quantity"),
            res.getString("category"),
            res.getString("imageUrl"),
            res.getInt("price"),
            res.getString("type"));
    }
}
```

2.3.3 Temporal Cohesion

Vị trí: view.screen.popup.PopupScreen

Mô tả:

Các thành phần được nhóm lại theo thời điểm thực thi

Lí do:

Chứa cả method popup và xử lí exception

```
public static void success(String message) throws IOException{
    popup(message, ViewsConfig.IMAGE_PATH + "/" + "tickgreen.png", true)
        .show(true);
}

public static void error(String message) throws IOException{
    popup(message, ViewsConfig.IMAGE_PATH + "/" + "tickerror.png", false)
        .show(false);
}

public static PopupScreen loading(String message) throws IOException{
    return popup(message, ViewsConfig.IMAGE_PATH + "/" + "loading.gif", true);
}

public void setImage(String path) {
    super.setImage(icon, path);
}

public void show(Boolean autoClose) {
    super.show();
    if (autoClose) close(0.8);
}

public void show(double time) {
    super.show();
    close(time);
}
```

2.3.4 Procedural Cohesion

Vị trí: controller.PlaceOrderController

Mô tả:

Các method
creatOrder, creatorder,
placeorder trong class
được nhóm lại với
nhau vì chúng được
thực hiện theo trình tự

Các thành phần được
nhóm lại vì chúng được
thực hiện theo trình tự
của chúng
Tính liên kết không tốt

```
public class PlaceOrderController extends BaseController {  
    /**  
     * Just for logging purpose  
     */  
    private static Logger LOGGER = utils.Utils.getLogger(PlaceOrderController.class.getName());  
  
    //Nguyen Dinh Duc. Common Coupling. Truy cập ở đây: SessionInformation.getInstance().cartInstance  
    /**  
     * This method checks the availability of product when user click PlaceOrder button  
     * @throws SQLException  
     */  
    public void placeOrder() throws SQLException {  
        SessionInformation.getInstance().cartInstance.checkAvailabilityOfProduct();  
    }  
  
    //Nguyen Dinh Duc. Common Coupling. Truy cập ở đây: SessionInformation.getInstance().cartInstance  
    /**  
     * This method creates the new Order based on the Cart  
     * @return Order  
     * @throws SQLException  
     */  
    public Order createOrder() throws SQLException {  
        return new Order(SessionInformation.getInstance().cartInstance);  
    }  
  
    /**  
     * This method creates the new Invoice based on order
```

2.3.5 Communicational Cohesion

Vị trí: entity.shipping.DeliveryInfo, calculateShippingFee(Order order)

Mô tả:

Phương thức này không liên quan đến class, chỉ sử dụng tham số distanceCalculator

```
public int calculateShippingFee(Order order) {  
    int distance = distanceCalculator.calculateDistance(address, province);  
    return (int) (distance * 1.2);  
}  
  
public String getName() {  
    return name;  
}  
  
public String getPhone() {  
    return phone;  
}  
  
public String getProvince() {  
    return province;  
}  
  
public String getAddress() {  
    return address;  
}  
  
public String getShippingInstructions() {  
    return shippingInstructions;  
}
```

2.4 Các nguyên lý SOLID

Bản thiết kế vi phạm các nguyên lý SOLID ở một vài module:

- Một số class phải thực hiện thêm chức năng mà người thiết kế chưa biết đặt ở đâu (SRP)
- Một số chức năng được thiết kế chưa phù hợp cho việc mở rộng sau này (OCP)
- Một số module đang phụ thuộc trực tiếp vào module cấp thấp hơn (DIP)
- Một số class kế thừa nhưng không thật sự là con của class cha (LSP & ISP)

2.4.1 Single Responsibility Principle

Vị trí: controller.AuthenticationController
controller.PaymentController
controller.PlaceOrderController
entity.cart.Cart

Mô tả:

Vì phạm ở hàm md5 của AuthenticationController

```
private String md5(String message) {  
    String digest = null;  
    try {  
        MessageDigest md = MessageDigest.getInstance("MD5");  
        byte[] hash = md.digest(message.getBytes(StandardCharsets.UTF_8));  
        // converting byte array to Hexadecimal String  
        StringBuilder sb = new StringBuilder(2 * hash.length);  
        for (byte b : hash) {  
            sb.append(String.format("%02x", b & 0xff));  
        }  
        digest = sb.toString();  
    } catch (NoSuchAlgorithmException ex) {  
        Utils.getLogger(Utils.class.getName());  
        digest = "";  
    }  
    return digest;  
}
```

Lý do:

Các class này đều có nhiệm vụ chính theo như tên gọi, nhưng chứa những hàm không giúp ích gì cho nó

2.4.2 Open-Close Principle

Vị trí: entity.shipping.DeliveryInfo
controller.PaymentController

Mô tả:

```
public class PaymentController extends BaseController {  
  
    /**  
     * Represent the card used for payment  
     */  
    private CreditCard card;  
  
    /**  
     * Represent the Interbank subsystem  
     */  
    private InterbankInterface interbank;  
  
    /**
```

Lý do:

Class PaymentController vi phạm nếu sau này muốn thêm hình thức thanh toán khác.
Class DeliveryInfo vi phạm nếu sau này muốn thêm thông tin vận chuyển khác.

2.4.3 Liskov Substitution Principle và Interface Segregation Principle

Vị trí: Views.screen.home.HomeScreenController
controller.AuthenticationController

Mô tả:

Các class này kế thừa từ class cha nhưng không hoàn toàn là con của class cha

```
public class AuthenticationController extends BaseController {  
  
public class HomeScreenHandler extends BaseScreenHandler
```

Lý do:

Class AuthenticationController không liên quan gì đến 2 phương thức của class BaseController (checkMediaInCart và getListCartMedia)

Class HomeScreenHandler không dùng đến 2 phương thức của class BaseScreenHandler (setPreviousScreen và getPreviousScreen)

2.4.4 Dependency Inversion Principle

Vị trí: entity.order.Order
controller.AuthenticationController

Mô tả:

Các class này
phụ thuộc trực
tiếp vào class
cấp thấp hơn

```
public void setDeliveryInfo(DeliveryInfo deliveryInfo) {  
    this.deliveryInfo = deliveryInfo;  
    this.shippingFees = deliveryInfo.calculateShippingFee(this);  
}  
  
public int calculateShippingFee(Order order) {  
    int distance = distanceCalculator.calculateDistance(address, province);  
    return (int) (distance * 1.2);  
}
```

Lý do:

Class AuthenticationController phụ thuộc vào UserDAO

Class Order phụ thuộc vào hàm calculateShippingFee của class DeliveryInfo

2.5 Clean Code

Đánh giá chung về mức độ clean code:

- Phần clear name đã được thực hiện tốt.
- Các method và class chưa thực sự clean, cần refactor lại nhiều phần.

2.5 Clean Code

Đánh giá chung về mức độ clean code:

- Phần clear name đã được thực hiện tốt.
- Các method và class chưa thực sự clean, cần refactor lại nhiều phần.

2.5.1 Clear Name

Vị trí: default.App, method start()

Mô tả:

```
// Finish splash with fade out effect
FadeTransition fadeOut = new FadeTransition(Duration.seconds(1), introScreen.getContent());
```

Các phần khởi tạo FadeTransition, tham số đầu tiên truyền vào thời gian cố định. Trong trường hợp cần sửa đổi sẽ gây ra khó khăn.

Nên đưa các tham số này thành thuộc tính static final

```
private static final Duration FADE_IN_DURATION = Duration.seconds(2);
private static final Duration FADE_OUT_DURATION = Duration.seconds(1);

// Load splash screen with fade in effect
FadeTransition fadeIn = new FadeTransition(FADE_IN_DURATION, introScreen.getContent());
```

2.5.2 Clean Function/Method

Vị trí: controller.PaymentController, method getExpirationDate()

Mô tả:

Phương thức còn chứa một biểu thức điều kiện phức tạp cần tạo thành một boolean function

```
private String getExpirationDate(String date) throws InvalidCardException {
    String[] strs = date.split("/");
    if (strs.length != 2) {
        throw new InvalidCardException();
    }

    String expirationDate = null;
    int month = -1;
    int year = -1;

    try {
        month = Integer.parseInt(strs[0]);
        year = Integer.parseInt(strs[1]);
        if (month < 1 || month > 12 || year < Calendar.getInstance().get(Calendar.YEAR) % 100 || year > 100) {
            throw new InvalidCardException();
        }
        expirationDate = strs[0] + strs[1];
    } catch (Exception ex) {
        throw new InvalidCardException();
    }

    return expirationDate;
}
```

2.5.2 Clean Function/Method

Vị trí: entity

Mô tả:

Thuộc tính của các class: Book, CD, DVD chưa có access modifier

```
public class Book extends Media {  
  
    String author;  
    String coverType;  
    String publisher;  
    Date publishDate;  
    int numOfPages;  
    String language;  
    String bookCategory;  
}
```

2.5.2 Clean Function/Method

Vị trí: `views.screen.cart.CartMediaHandler`, method `initializeSpinner()`

Mô tả:

```
private void initializeSpinner(){
    SpinnerValueFactory<Integer> valueFactory = //
        new SpinnerValueFactory.IntegerSpinnerValueFactory(1, 100, cartItem.getQuantity());
    spinner = new Spinner<Integer>(valueFactory);
    spinner.setOnMouseClicked( e -> {
        try {
            int numOfProd = this.spinner.getValue();
            int remainQuantity = cartItem.getMedia().getQuantity();
            LOGGER.info("NumOfProd: " + numOfProd + " -- remainOfProd: " + remainQuantity);
        }
    });
}
```

Lý do:

Phương thức này có 2 nhiệm vụ:

- Tạo ra Spinner
- Xử lý sự kiện khi click vào spinner.

Việc này làm giảm tính dễ đọc của code vì đây là hai nhiệm vụ riêng biệt.

2.5.3 Clean Class

Vị trí: controller

Mô tả:

Trong mã nguồn ban đầu có một số Large class và các thành phần bên trong các class đó chưa thực hiện cùng một nhiệm vụ như:

AuthenticationController, PaymentController, PlaceOrderController

AuthenticationController

```
+ isAnonymousSession() : boolean  
+ getMainUser() : User  
- validateUser() : boolean  
+ login(email : String, password : String) : void  
+ logout() : void
```

PlaceOrderController

```
+ placeOrder() : void  
+ createOrder() : Order  
+ createInvoice(order : Order) : Invoice  
+ processDeliveryInfo(info : HashMap) : DeliveryInfo
```

PaymentController

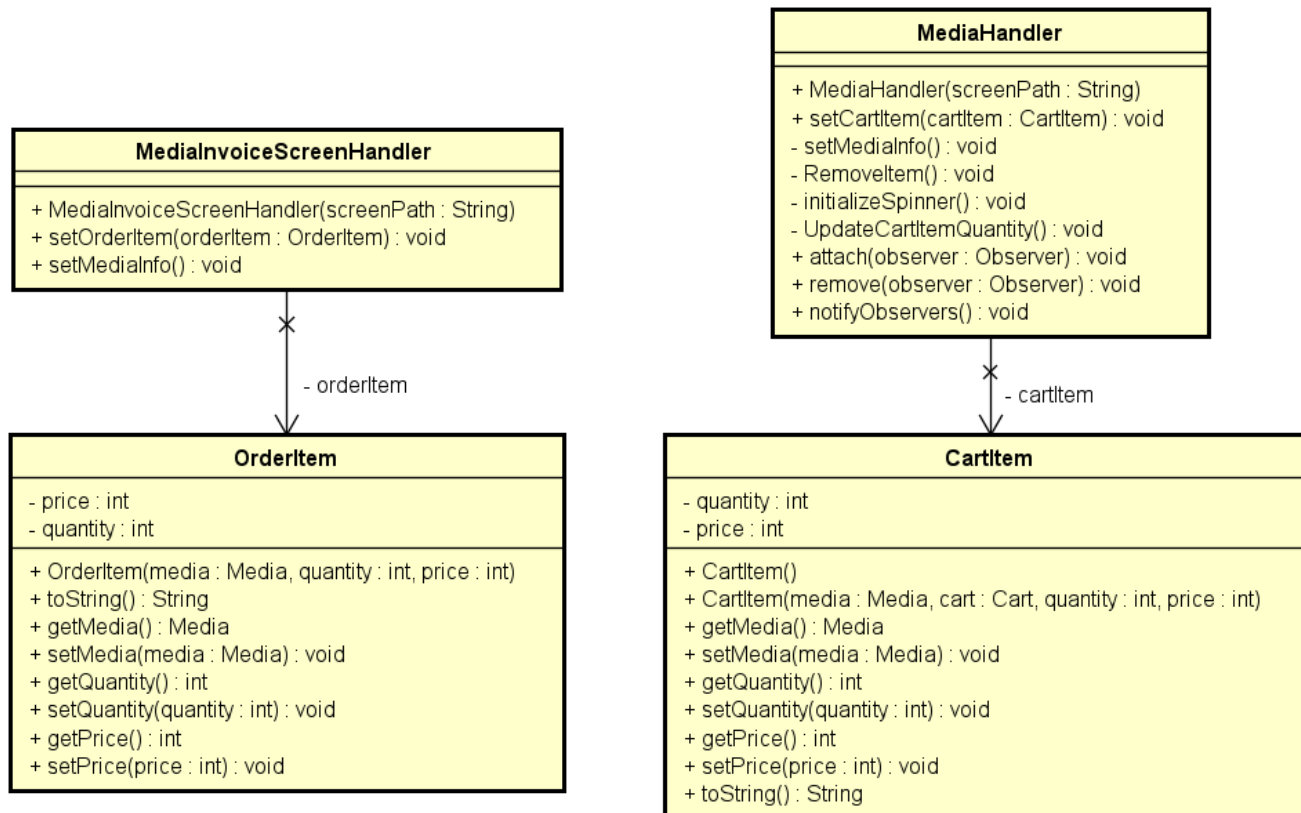
```
+ payOrder(amount : int, contents : String, cardNumber : String, cardHolderName : String, expirationDate : String, securityCode : String) : Map<String,String>  
+ payOrder(amount : int, contents : String, cardType : String, issuingbank : String, cardNumber : String, cardholderName : String, validFromDate : String) : Map<String,String>  
+ emptyCart() : void
```

2.5.3 Clean Class

Vị trí: views.screen.cart.MediaHandler, entity.cart.CartItem
views.screen.invoice.MediaInvoiceScreenHandler, entity.order.OrderItem

Coupling giữa hai class CartItem và MediaHandler.

Coupling giữa hai class OrderItem và MediaInvoiceScreenHandler.



2.5.3 Clean Class

Vị trí: default.App, method loadContent()

Mô tả:

```
private void loadContent(Stage primaryStage) {  
    try {  
        HomeScreenHandler homeHandler = new HomeScreenHandler(primaryStage, ViewsConfig.HOME_PATH);  
        homeHandler.setScreenTitle("Home Screen");  
        homeHandler.setImage();  
        homeHandler.show();  
    } catch (IOException e1) {  
        e1.printStackTrace();  
    }  
}
```

HomeScreenHandler đưa ra các phương thức public setImage(), trong khi hoàn toàn có thể được gọi từ bên trong lớp đó, không có tác dụng gì khi gọi từ bên ngoài.



3

Đề xuất
cải tiến

3.1 Class SessionInformation chứa các thuộc tính static

Vấn đề:

Vị trí: controller.SessionInformation

Nhiều class tham chiếu đến
các thuộc tính static của class
SessionInformation

SessionInformation
+ <u>mainUser : User</u> + <u>cartInstance : Cart</u> + <u>expiredTime : LocalDateTime</u>

Giải pháp:

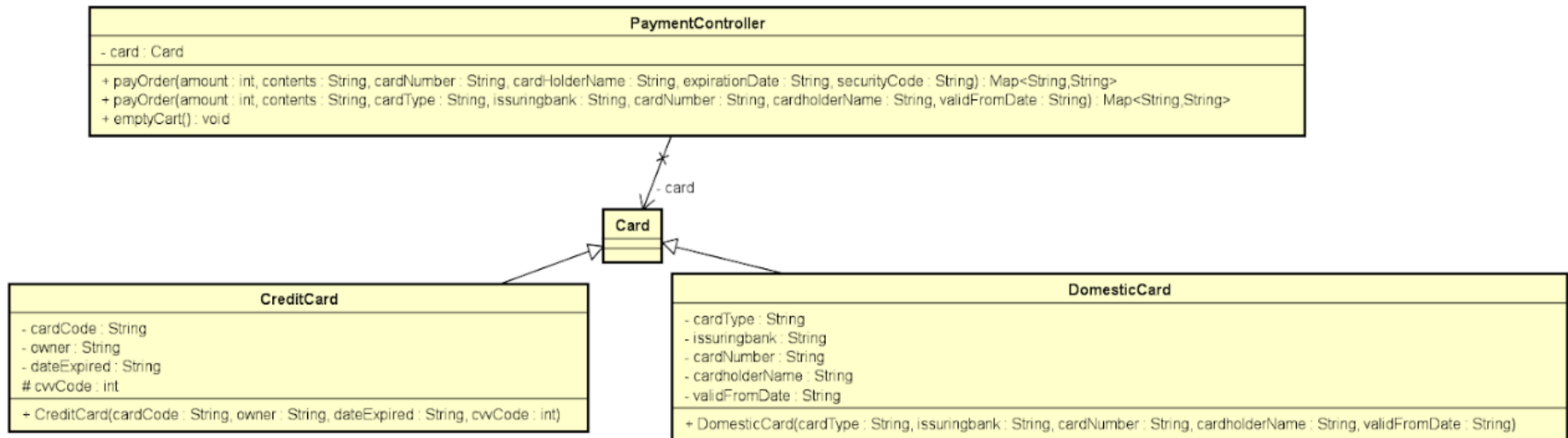
Sử dụng mẫu thiết kế
Singleton

```
public class SessionInformation {  
    private static SessionInformation instance;  
  
    private SessionInformation() {  
  
    };  
  
    public static synchronized SessionInformation getInstance() {  
        if (instance == null) {  
            instance = new SessionInformation();  
        }  
  
        return instance;  
    }  
  
    public User mainUser;  
    public Cart cartInstance = new Cart();  
    public LocalDateTime expiredTime;  
}
```

3.2 Thêm phương thức thanh toán mới (Domestic Card)

Giải pháp:

Sử dụng mẫu thiết kế Strategy



3.2 Thêm phương thức thanh toán mới (Domestic Card)

Giải pháp:

Khi khách muốn thanh toán, trong method của class PaymentController sẽ setCard là CreditCard và setPaymentController là InterbankSubsystemCreditCardController tương ứng với CreditCard và ngược lại

```
public Map<String, String> payOrder(int amount, String contents, String cardNumber, String cardHolderName,
    String expirationDate, String securityCode) {
    Map<String, String> result = new Hashtable<String, String>();
    result.put("RESULT", "PAYMENT FAILED!");

    Validate validate = new Validate();

    try {
        this.card = new CreditCard(
            cardNumber,
            cardHolderName,
            validate.getExpirationDate(expirationDate),
            Integer.parseInt(securityCode));

        if (this.interbank == null) this.interbank = new InterbankSubsystem();
        this.interbank.setPaymentController(new InterbankSubsystemCreditCardController());
        PaymentTransaction transaction = interbank.payOrder(card, amount, contents);

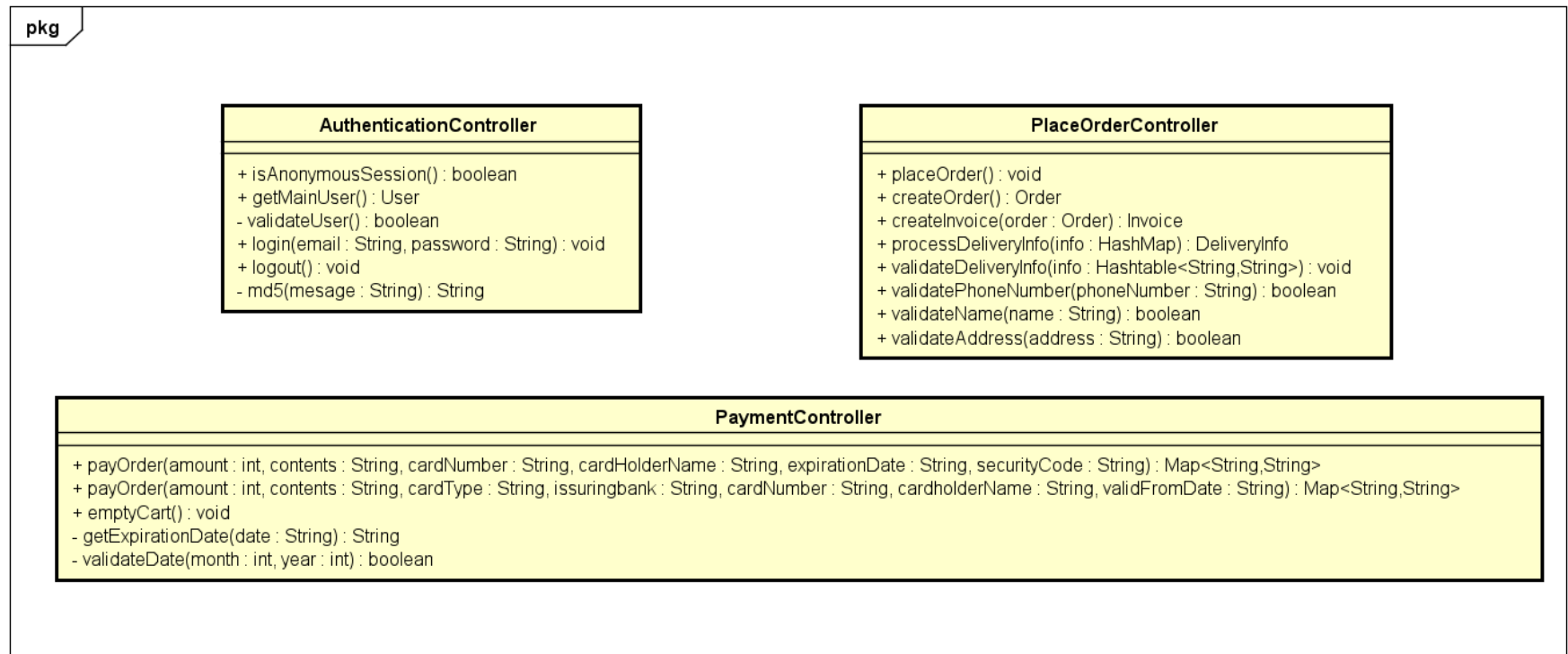
        result.put("RESULT", "PAYMENT SUCCESSFUL!");
        result.put("MESSAGE", "You have successfully paid the order!");
    } catch (PaymentException | UnrecognizedException ex) {
        result.put("MESSAGE", ex.getMessage());
    }
    return result;
}
```

3.3 Các Large class

Vấn đề:

Vị trí: `controller.AuthenticationController`, `controller.PaymentController`,
`controller.PlaceOrderController`

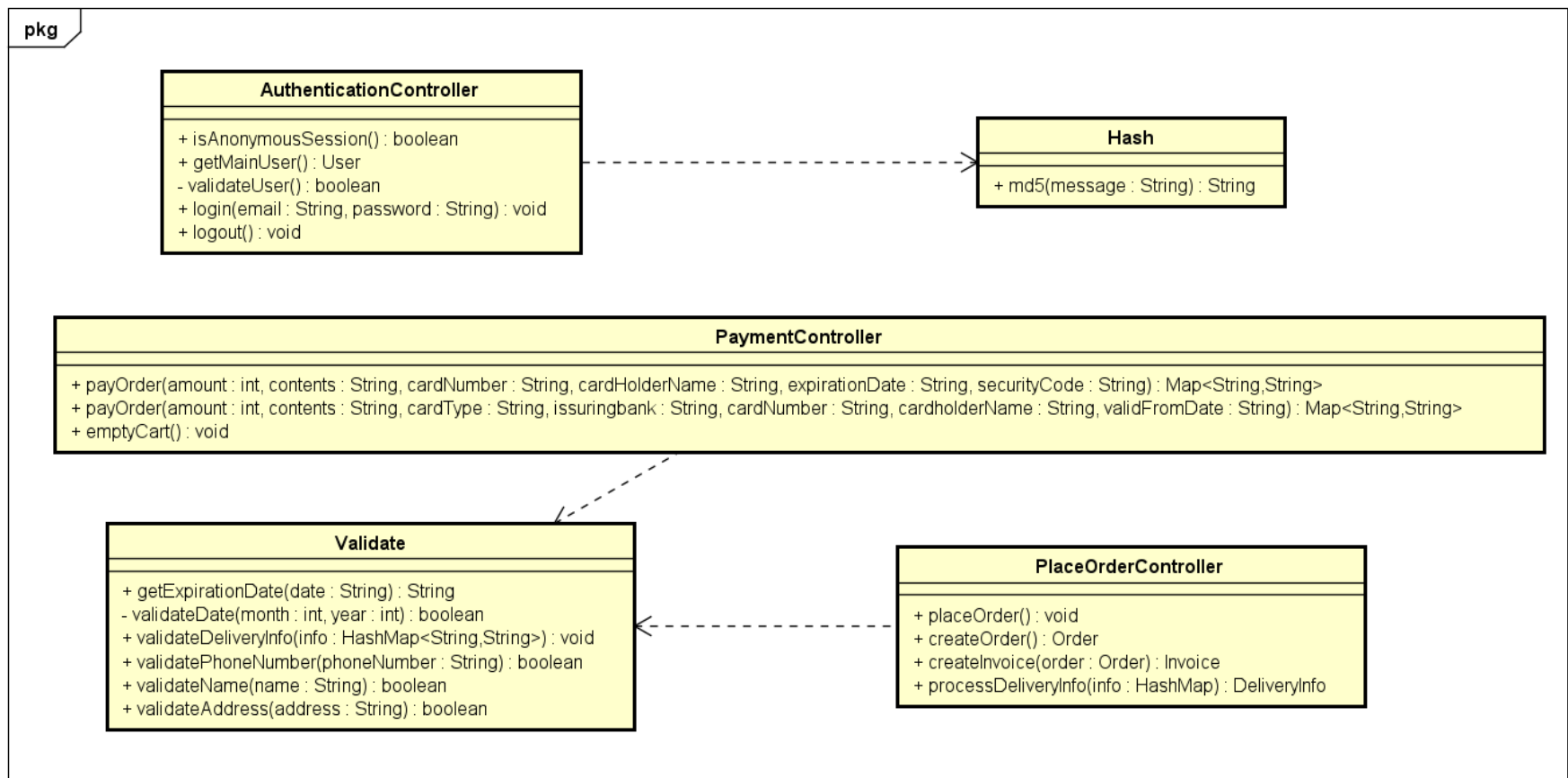
Các lớp này chứa một số phương thức không liên quan.



3.3 Các Large class

Giải pháp:

Chuyển các method không liên quan đến class đó sang 1 class khác

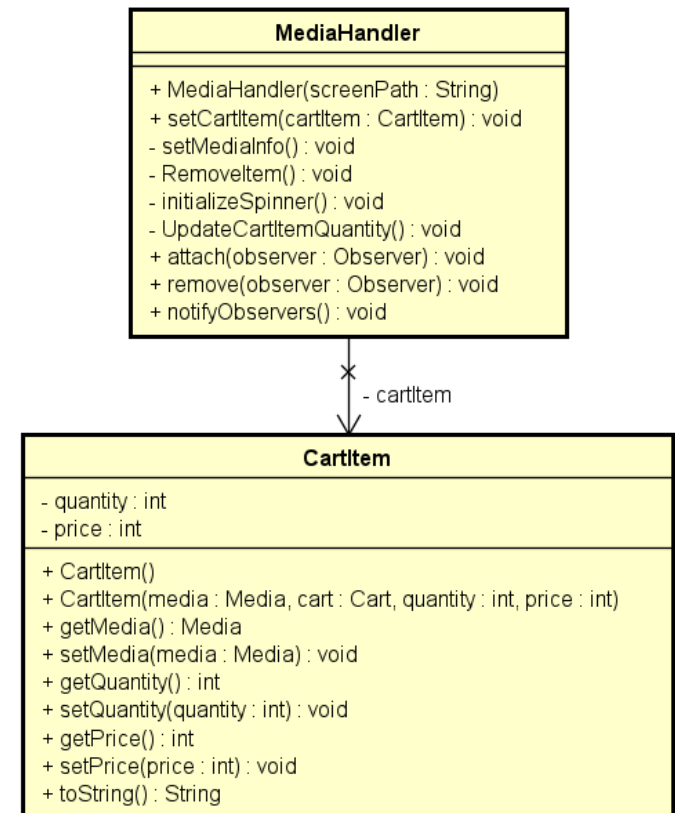
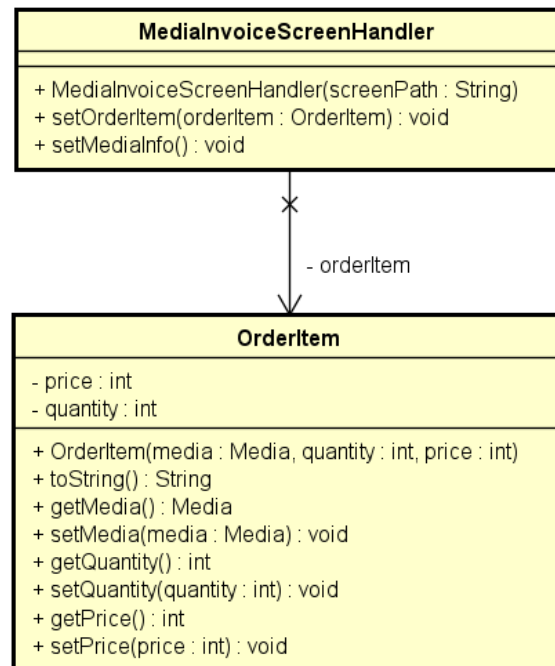


3.4 Vấn đề coupling giữa class CartItem và MediaHandler

Vấn đề:

Vị trí: views.screen.cart.MediaHandler, entity.cart.CartItem
views.screen.invoice.MediaInvoiceScreenHandler, entity.order.OrderItem

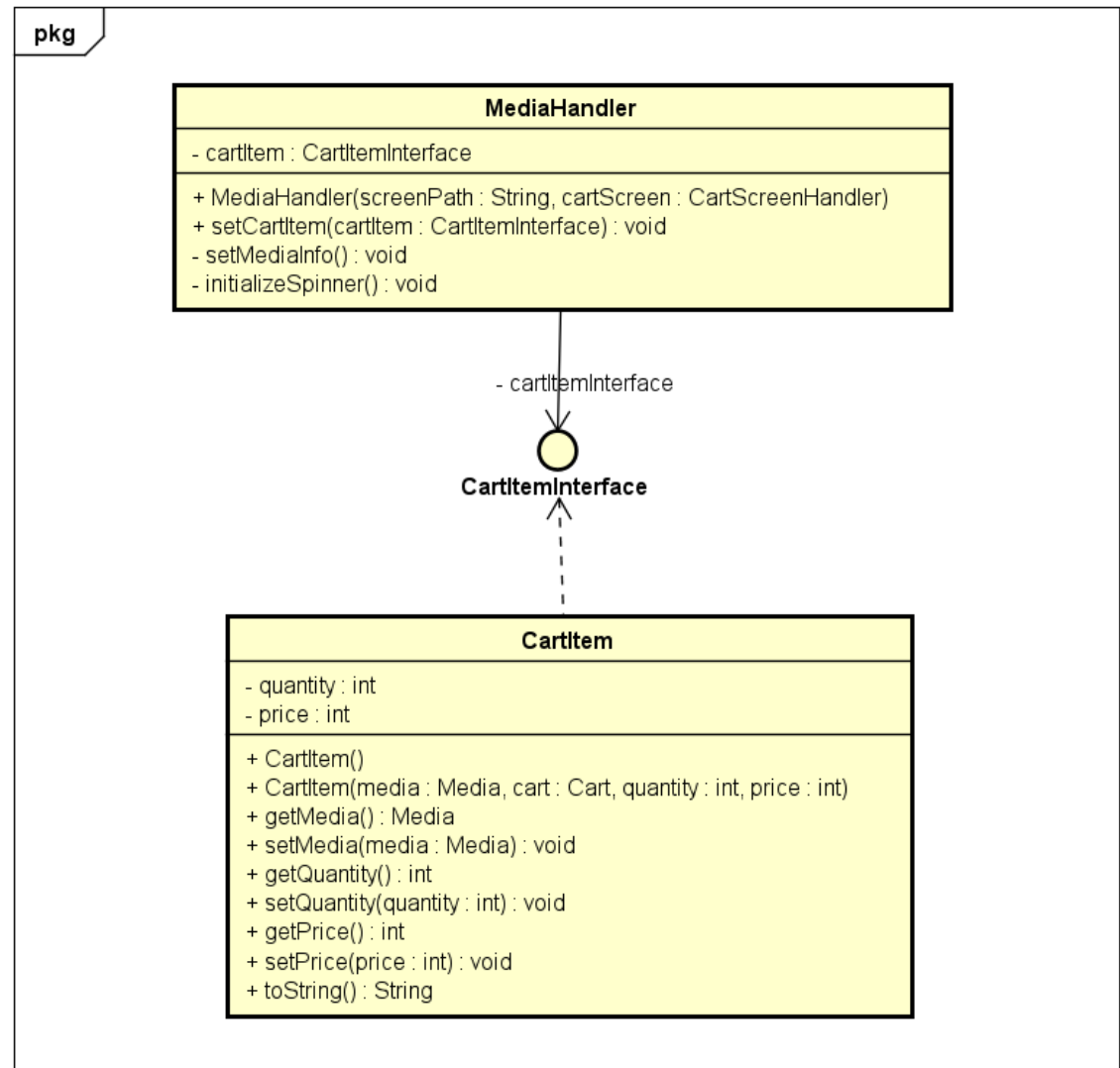
Hai class CartItem và MediaHandler,
hai class OrderItem và
MediaInvoiceScreenHandler coupling với nhau.



3.4 Vấn đề coupling giữa class CartItem và MediaHandler

Giải pháp:

Tạo interface mới là CartItemInterface và cho class CartItem implement



3.5 Vấn đề clean method tại CartMediaHandler và giải pháp

Vấn đề:

Vị trí: `views.screen.cart.CartMediaHandler`, method `initializeSpinner()`

Phương thức này có 2 nhiệm vụ: tạo ra Spinner và xử lý sự kiện khi click vào spinner, làm giảm tính dễ đọc của code

```
private void initializeSpinner(){
    SpinnerValueFactory<Integer> valueFactory = //
        new SpinnerValueFactory.IntegerSpinnerValueFactory(1, 100,
    spinner = new Spinner<Integer>(valueFactory);
    spinner.setOnMouseClicked( e -> {
        try {
            int numOfProd = this.spinner.getValue();
            int remainQuantity = cartItem.getMedia().getQuantity();
            LOGGER.info("NumOfProd: " + numOfProd + " -- remainOfPr
```

Giải pháp:

Tách phần xử lý khi click spinner thành một private method riêng.

```
private void initializeSpinner(){
    SpinnerValueFactory<Integer> valueFactory = //
        new SpinnerValueFactory.IntegerSpinnerValueFactory(1, 100, cartItem.getQuar
    spinner = new Spinner<Integer>(valueFactory);
    spinner.setOnMouseClicked( e -> {
        UpdateCartItemQuantity();
    });
    spinnerFX.setAlignment(Pos.CENTER);
    spinnerFX.getChildren().add(this.spinner);
}

private void UpdateCartItemQuantity() {
    try {
        int numOfProd = this.spinner.getValue();
        int remainQuantity = cartItem.getMedia().getQuantity();
```

3.6 Vấn đề clean method tại CartMediaHandler và giải pháp

Vấn đề:

Vị trí: `views.screen.cart.CartMediaHandler`, method `setMediaInfo()`

Phương thức này có 2 nhiệm vụ: thiết lập thông số đồ họa và xử lý sự kiện khi click vào nút delete, làm giảm tính dễ đọc.

```
// add delete button
btnDelete.setFont(ViewsConfig.REGULAR_FONT);
btnDelete.setOnClickListener(e -> {
    SessionInformation.getInstance().cartInstance.removeCartMedia(cartItem); //
    notifyObservers();
    LOGGER.info("Deleted " + cartItem.getMedia().getTitle() + " from the cart");
});
```

Giải pháp:

Tách phần xử lý khi click nút Delete thành một phương thức riêng.

```
// add delete button
btnDelete.setFont(ViewsConfig.REGULAR_FONT);
btnDelete.setOnClickListener(e -> {
    RemoveItem();
});

initializeSpinner();
}

private void RemoveItem() {
    SessionInformation.getInstance().cartInstance.removeCartMedia(cartItem); //
    notifyObservers();
    LOGGER.info("Deleted " + cartItem.getMedia().getTitle() + " from the cart");
}
```

3.7 Vấn đề clean class tại HomeScreenHandler

Vị trí: default.App, method loadContent(), views.screen.home.HomeScreenhandler

Mô tả:

```
private void loadContent(Stage primaryStage) {  
    try {  
        HomeScreenHandler homeHandler = new HomeScreenHandler(primaryStage, ViewsConfig.HOME_PATH);  
        homeHandler.setScreenTitle("Home Screen");  
        homeHandler.setImage();  
        homeHandler.show();  
    } catch (IOException e1) {  
        e1.printStackTrace();  
    }  
}
```

HomeScreenHandler đưa ra các phương thức public setImage(), trong khi hoàn toàn có thể được gọi từ bên trong lớp đó, không có tác dụng gì khi gọi từ bên ngoài.

Giải pháp:

Đưa phương thức setImage() về private, gọi từ hàm khởi tạo.

```
public HomeScreenHandler(Stage stage, String screenPath) throws IOException{  
    super(stage, screenPath, null);  
    setImage();  
}
```

3.8 Vấn đề coupling và giải pháp dùng observer pattern

Vấn đề:

Vị trí: views.screen.cart.MediaHandler, views.screen.cart.CartScreenHandler

Hai lớp MediaHandler, CartScreenHandler có sự phụ thuộc lẫn nhau.

Tại CartScreenHandler, phương thức displayCartWithMediaAvailability():

```
try {
    for (Object cm : lstMedia) {

        // display the attribute of vboxCart media
        CartItem cartItem = (CartItem) cm;
        MediaHandler mediaCartScreen = new MediaHandler(ViewsConfig.CART_MEDIA_PATH, this);
        mediaCartScreen.setCartItem(cartItem);

        // add spinner
        vboxCart.getChildren().add(mediaCartScreen.getContent());
    }
}
```

Tại MediaHandler:

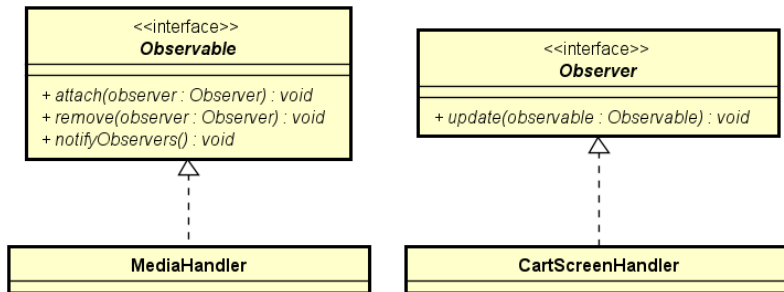
```
private CartItem cartItem;
private Spinner<Integer> spinner;
private CartScreenHandler cartScreen;

public MediaHandler(String screenPath, CartScreenHandler cartScreen) throws IOException {
    super(screenPath);
    this.cartScreen = cartScreen;
    hboxMedia.setAlignment(Pos.CENTER);
}
```

3.8 Vấn đề coupling và giải pháp dùng observer pattern

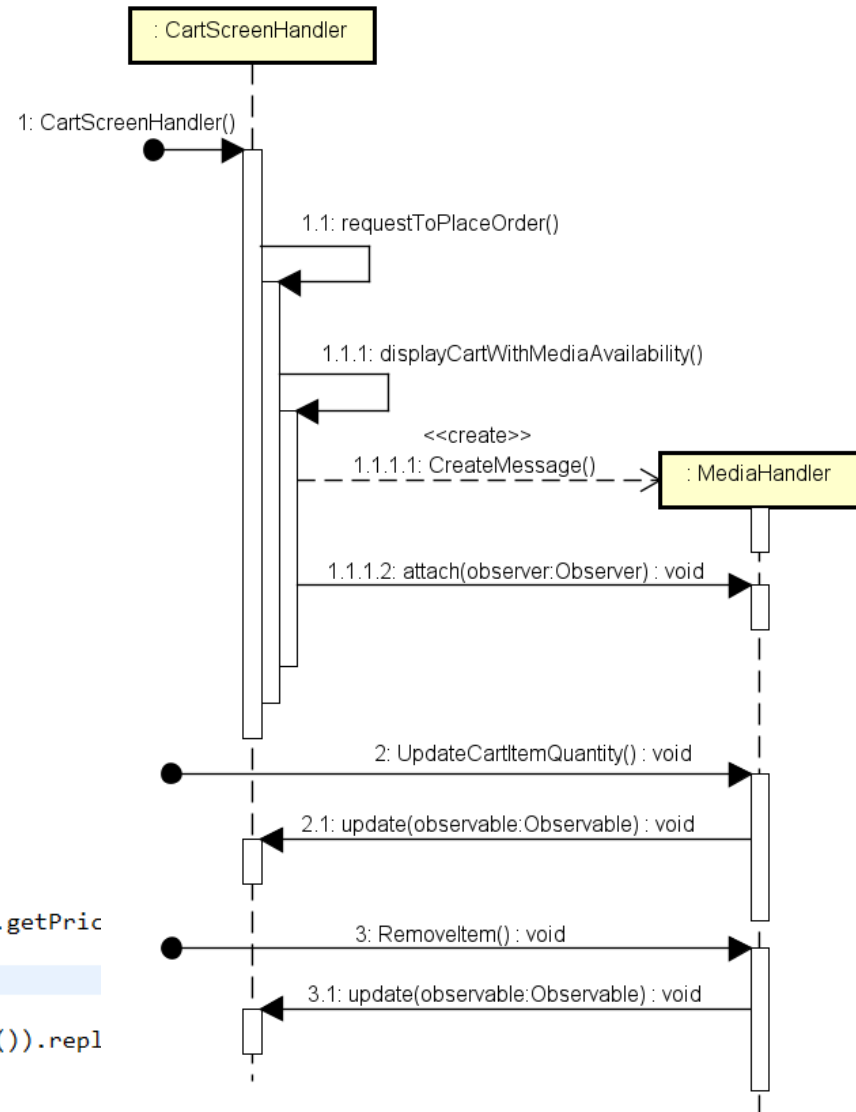
Giải pháp:

Sử dụng Observer pattern, với
MediaHandler implement Observable và
CartScreenHandler implement Observer.



MediaHandler có hai sự kiện cần thông báo cho Observer: xóa mặt hàng và thay đổi số lượng một mặt hàng. Ta gọi tới `notifyObservers()` tại các hàm xóa và thay đổi số lượng.

```
// update quantity of mediaCart in useCart
cartItem.setQuantity(numOfProd);
// update the total of mediaCart
price.setText(ViewsConfig.getCurrencyFormat(numOfProd* cartItem.getPric
// update subtotal and amount of Cart
notifyObservers();
} catch (SQLException e1) {
    throw new MediaUpdateException(Arrays.toString(e1.getStackTrace()).repl
}
```



3.8 Vấn đề coupling và giải pháp dùng observer pattern

Giải pháp:

Tại MediaHandler (Observable): set null cho cartItem sau khi xóa, đồng thời cung cấp phương thức get cho cartItem.

```
private void RemoveItem() {  
    SessionInformation.getInstance().cartInstance.removeCartMedia((CartItem)cartItem);  
    LOGGER.info("Deleted " + cartItem.getMedia().getTitle() + " from the cart");  
    cartItem = null;  
    notifyObservers();  
}
```

Tại ScreenCartHandler (Observer):

```
@Override  
public void update(Observable observable) {  
    if (observable instanceof MediaHandler) {  
        update((MediaHandler)observable);  
    }  
}  
  
private void update(MediaHandler mediaHandler) {  
    if (mediaHandler.getCartItem() == null) {  
        vboxCart.getChildren().remove(mediaHandler.getContent());  
    }  
    updateCartAmount();  
}
```

3.9 Yêu cầu thay đổi công thức tính vận chuyển

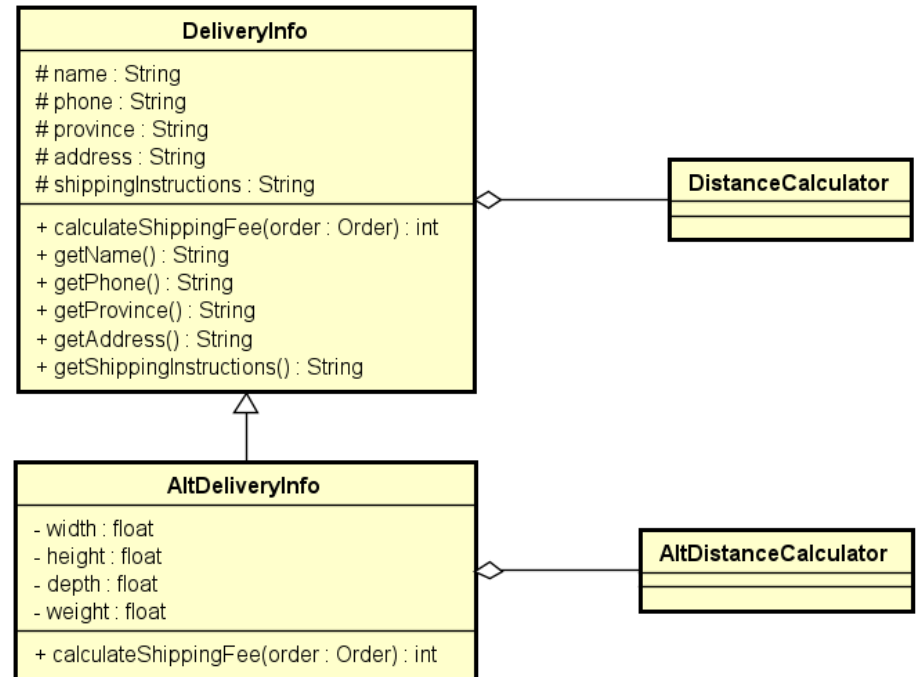
Vấn đề:

Công thức vận chuyển mới yêu cầu làm việc với thư viện mới.

Giải pháp:

Tạo adapter class kế thừa DeliveryInfo, và override lại calculateShippingFee() để sử dụng tới thư viện mới.

```
//      DeliveryInfo deliveryInfo = new DeliveryInfo(  
//          String.valueOf(info.get("name")),  
//          String.valueOf(info.get("phone")),  
//          String.valueOf(info.get("province")),  
//          String.valueOf(info.get("address")),  
//          String.valueOf(info.get("instructions")),  
//          new DistanceCalculator());  
AltDeliveryInfo deliveryInfo = new AltDeliveryInfo(  
    String.valueOf(info.get("name")),  
    String.valueOf(info.get("phone")),  
    String.valueOf(info.get("province")),  
    String.valueOf(info.get("address")),  
    String.valueOf(info.get("instructions")),  
    new AlternativeDistanceCalculator());  
deliveryInfo.setDepth(1);  
deliveryInfo.setHeight(2);  
deliveryInfo.setWeight(3);  
deliveryInfo.setWeight(4);  
System.out.println(deliveryInfo.getProvince());  
return deliveryInfo;
```



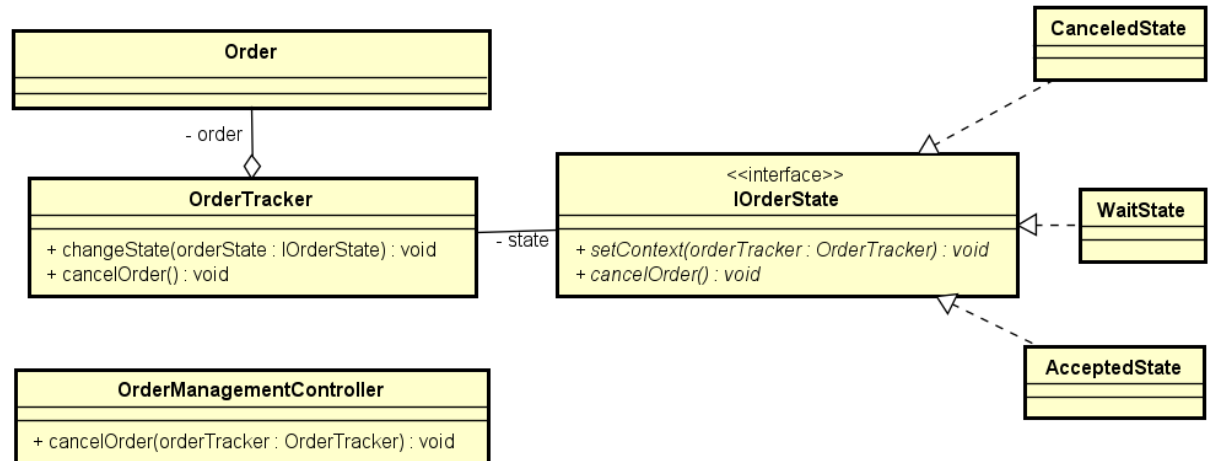
AltDeliveryInfo có thêm các thuộc tính mới cùng hàm set.

Tại client (PlaceOrderController), sửa lại cách khởi tạo DeliveryInfo.

3.10 Yêu cầu cập nhật chức năng hủy đơn hàng

Vấn đề:

Chức năng hủy đơn hàng, chỉ cho phép một số trạng thái của đơn được hủy.



Giải pháp:

Tạo lớp **OrderTracker**, lưu thông tin trạng thái của 1 order.
Sử dụng state pattern với **OrderTracker** là context và các state kế thừa từ **IOrderState**.

```
public class StateAccepted implements IOrderState {
    private OrderTracker context;

    @Override
    public void SetContext(OrderTracker context) {
        this.context = context;
    }

    @Override
    public void cancelOrder() throws CancelOrderNotAllowedException {
        throw new CancelOrderNotAllowedException();
    }
}
```

```
public void cancelOrder(OrderTracker orderTracker) {
    try {
        orderTracker.cancelOrder();
    } catch (CancelOrderNotAllowedException e) {
        e.printStackTrace();
    }
}
```

Tại client, gọi hàm `cancelOrder` trong **OrderTracker** và bắt exception nếu không được phép hủy.



4

Tổng kết

Kết quả thu được
Vấn đề tồn đọng

4.1 Kết quả thu được

- Đánh giá tổng quan về các mức độ coupling, cohesion, clean code.
- Tìm kiếm và sửa chữa mã nguồn, đạt được mức coupling và cohesion tốt hơn.
- Đánh giá và sửa chữa được một số lỗi trong mã nguồn không tuân theo nguyên lý SOLID.
- Thực hiện tái cấu trúc lại mã nguồn để thỏa mãn hầu hết các yêu cầu phát sinh của hệ thống.
- Chương trình hoạt động tốt, không có lỗi phát sinh sau quá trình tái cấu trúc.

4.2 Vấn đề tồn đọng

- Một số lỗi về việc không tuân thủ nguyên lý SOLID, gây ra high coupling và low cohesion vẫn chưa được sửa chữa một cách hoàn thiện.
- Cần thêm các hoạt động clean code để mã nguồn dễ đọc, dễ bảo trì hơn.



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Thanks for listening

A decorative geometric pattern in the bottom right corner, consisting of a grid of squares in orange and purple colors, arranged in a way that creates a sense of depth and perspective.