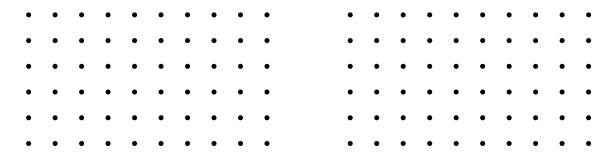




Hanoi University of  
Science and Technology



# **DESIGN PATTERN**

## **GROUP 10**

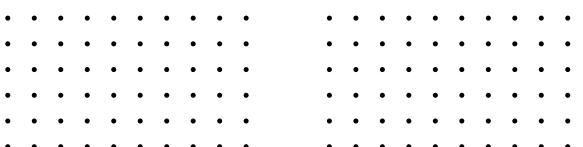
### **Thành viên**

Trịnh Đức Tiệp 20194185

Trần Văn Phúc 20194139

Nguyễn Đức Thắng 20194170

Chu Thành Đô 20194018



# **NỘI DUNG TRÌNH BÀY**

- **TỔNG QUAN DỰ ÁN**
- **ĐÁNH GIÁ THIẾT KẾ CŨ**
- **ĐỀ XUẤT CẢI TIẾN**
- **TỔNG KẾT**

# I. TỔNG QUAN DỰ ÁN

**Phần mềm AIMS (An Internet Media Store)** là một phần mềm viết bằng Java, giúp người dùng tìm kiếm và mua các loại mặt hàng liên quan đến các vật phẩm truyền thông đa phương tiện như sách, đĩa CD, đĩa DVD, ...



# I. TỔNG QUAN DỰ ÁN

## Các chức năng chính

- Theo dõi và tham quan cửa hàng trực tuyến, tìm kiếm và lọc ra các loại mặt hàng truyền thông đa phương tiện có trong hệ thống.
- Đăng ký tài khoản và đăng nhập vào hệ thống.
- Quản lý giỏ hàng cá nhân: Bao gồm các hoạt động thêm, xóa sản phẩm vào giỏ hàng.
- Thanh toán đơn hàng.

# I. TỔNG QUAN DỰ ÁN

## Cấu trúc thư mục mã nguồn

- **Package common:** chứa các file mã nguồn với mục đích sử dụng cho việc dùng chung ở nhiều nơi như các ngoại lệ (exception) và các interface chức năng (như Observer, Observable, ...)
- **Package controller:** chứa các file mã nguồn của các controller được sử dụng trong hệ thống
- **Package dao:** chứa các file thực hiện nhiệm vụ truy xuất tới dữ liệu local sử dụng thư viện SQLite.
- **Package entity:** chứa các file định nghĩa ra các model sử dụng bên trong hệ thống

# I. TỔNG QUAN DỰ ÁN

## Cấu trúc thư mục mã nguồn

- **Package subsystem**: một hệ thống con chứa giao diện interbank hỗ trợ giao dịch liên kết ngân hàng thông qua các API
- Package utils: chứa các file công cụ chức năng được sử dụng nhiều nơi khác nhau
- **Package views**: là nơi định nghĩa ra các trang, giao diện người dùng và các màn hình của hệ thống

# I. TỔNG QUAN DỰ ÁN

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

- Thêm một mặt hàng mới: Audio Book (Sách nói)
- Thêm màn hình xem chi tiết sản phẩm
- Thay đổi yêu cầu khi load giao diện
- Thêm cách tính khoảng cách, sử dụng thư viện mới
- Thêm phương thức thanh toán mới: Thẻ nội địa (Domestic Card)
- Thay đổi công thức tính phí vận chuyển
- Cập nhật lại chức năng hủy đơn hàng

# I. TỔNG QUAN DỰ ÁN

## Hoạt động cần thực hiện

- Review codebase. Xem xét và đánh giá lại mã nguồn
- Refactor codebase. Tái cấu trúc lại mã nguồn
- Optimization. Chọn mô hình và phương pháp tái cấu trúc cho lợi ích tối ưu
- Testing. Kiểm thử lại các tính năng của dự án sau khi tái cấu trúc
- Scalability. Mở rộng dự án

# I. TỔNG QUAN DỰ ÁN

## Kết quả dự kiến

- Làm sạch mã nguồn (Không có code smell,...)
- Tái cấu trúc mã nguồn, đảm bảo chúng khoa học, dễ hiểu
- Tăng cường khả năng mở rộng trong tương lai



## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Coupling - Thống kê

#	<i>Các mức độ về Coupling</i>	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
1	Content	<u>entity.cart</u> (Cart)	<u>getListMedia()</u>	Trả về trực tiếp tham chiếu List<CartItem> lstCartItem của lớp thay vì trả về bản copy
2	Content	<u>entity.order</u> (Order)	<u>getOrderMediaList()</u>	Trả về trực tiếp tham chiếu List orderMediaList của lớp thay vì trả về bản copy
3	Common	<u>controller(Se ssionInfomati on.java)</u>	Nhiều class sử dụng chung các trường và method của SessionInformation	Các class có sử dụng chung: AuthenticationController.java, BaseController.java, PaymentController.java, ViewCartController.java.

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Coupling - Thống kê

#	<i>Các mức độ về Coupling</i>	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
4	Common	<a href="#"><u>entity.shipping</u></a> (ShippingConfigs)	Nhiều class sử dụng chung các trường và method của ShippingConfigs	Các class có sử dụng chung: <a href="#"><u>ShippingScreenHandler.java</u></a> ...
5	Control	InterbankPayloadConverter	hàm <a href="#"><u>extractPaymentTransactio</u></a> <a href="#"><u>n()</u></a> có phân xử lý để throw ra exception vi phạm	Dùng switch case với từng mã lỗi và xử lý tương ứng, trong khi hàm này có trách nhiệm khác
6	Stamp	MediaDAO.java (updateMediaFieldById), Cart.java (checkMeddiaInCart.java), CartItem.java (Constructor CartItem),...	Vi phạm stamp coupling ở nhiều nơi	Nhiều tham số truyền vào thừa mà không sử dụng

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Cohesion - Thống kê

#	Các mức độ về Cohesion	Module	Mô tả	Lý do
1	Coincidental	utils (Utils.java)	DATE_FORMATTER, LOGGER	Trong lớp Utils đang thực hiện quá nhiều công việc không liên quan đến nhau (God class)
2	Coincidental	<a href="#">views.screen</a> (ViewsConfig)	Các thuộc tính đường dẫn <u>PATH</u> , <u>PERCENT</u> VAT, REGULAR_FONT, getCurrencyFormat()	Trong lớp ViewsConfig chứa nhiều nhóm thuộc tính, phương thức không liên quan đến nhau (God class)
3	Logical	<a href="#">views.screen.popup</a> (PopupScreen)	Các method <u>success()</u> , <u>error()</u> , <u>loading()</u>	Các method <u>success()</u> , <u>error()</u> , <u>loading()</u> đều là xử lý các dạng popup khác nhau nhưng thật sự không liên quan tới nhau

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Cohesion - Thống kê

#	Các mức độ về Cohesion	Module	Mô tả	Lý do
4	Temporal	<u>views.screen</u> (Các lớp ScreenHandler)	<u>setupData()</u> và <u>setupFunctionality()</u> vi phạm temporal	Hai method <u>setupData()</u> và <u>setupFunctionality()</u> được thiết lập chạy theo thứ tự khi bắt đầu khởi tạo các màn hình
5	Procedural	<u>Controller(PaymentController)</u>	Các method <u>getExpirationDate()</u> , <u>payOrder()</u>	Hai method trên thực hiện theo tuần tự cụ thể (Kiểm tra ngày hết hạn thẻ, rồi mới trả tiền order)
6	Communicational	<u>Controller(SessionInformation.java)</u>	Nhiều class có sử dụng chung các property và method của class này	Các class của controller có sử dụng: BaseController.java, ViewCartController.java, ...

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Cohesion - Thống kê

#	Các mức độ về Cohesion	Module	Mô tả	Lý do
7	Communicational	<u>dao.media</u> (Các class thuộc package này)	Các method getMediaById vi phạm	Các method của BookDao, CDDAO hoạt động trên data cùng loại (Thuộc <u>entity.media</u> )
8	Communicational	<u>subsystem.interbank</u> (InterbankBoundary.java)	Trả về của hàm query vi phạm cohesion	Hàm query trả về kiểu dữ liệu có liên quan tới class ApplicationProgrammingInterface
9	Sequential	<u>views.screen.invoice</u> (MediaInvoiceScreenHandler.java)	setOrderItem và setMediaInfo vi phạm Sequential	setOrderItem sẽ thiết lập orderItem trong đối tượng, sau đó gọi setMediaInfo xử lý dựa trên giá trị đó

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Coupling & Cohesion - Nhận xét

- **Hiện tại, trong code base vẫn còn tồn lại nhiều class vi phạm các mức độ cao của coupling và cohesion.**
- Có vài class chỉ vi phạm ở mức độ nhỏ (Như Stamp coupling, sequential cohesion,...), tuy nhiên có nhiều class đang vi phạm coupling và cohesion ở mức độ cao hơn (Coincidental cohesion, logical cohesion, hay content coupling, ... ), đòi hỏi cần áp dụng các biện pháp để khắc phục triệt để.

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Single responsibility principle

#	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
1	<u>entity.cart</u> (Cart.js)	Class Cart đang nắm nhiều trách nhiệm	Class cart vừa đảm nhiệm chức năng quản lý cart thông thường (thêm, gỡ bỏ <u>media...</u> ), vừa có trách nhiệm thông báo tình hình cart (Có trống hay không, số lượng các media trong cart, tổng tiền, hay media có đang bán hay không), việc này có thể giao cho ViewCartController quản lý.
2	<u>entity.shipping</u> (Deli veryInfo.java)	Class DeliveryInfo đang nắm 2 trách nhiệm	Tại thông tin khoảng cách, thì class nên tính luôn thông tin phí khoảng cách (Dựa trên thông tin address và province, rồi bind vào property distanceFee). Việc tạo phương thức calculateShippingFee khiến cho deliveryInfo đóng vai trò vừa lưu giữ thông tin vận chuyển, vừa cung cấp công cụ tính giá shipping.
3	<u>subsystem.interbank</u> <u>k</u> (InterbankPayloadC onverter.java)	Class đóng vai trò convert interbank payload, nhưng có hàm getToday (Lấy ngày hôm nay), không hợp nhất với mục đích của class	Class có chứa hàm getToday, hàm này khá ngắn và chỉ được dùng duy nhất chỉ trong class này, nên được gộp chung với convertToRequestPayload

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Single responsibility principle

#	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
4	Utils (ApplicationProgrammingInterface.java) a)	Class đóng vai trò xử lý nhiều HTTP method khác nhau	Class đóng vai trò xử lý nhiều HTTP method khác nhau (Get, Post). Việc xử lý hai method get và post nên được tách riêng ra
5	<u>utils</u> (Util.java)	Class đóng vai trò <u>god</u> class (Xử lý nhiều các tác vụ không liên quan)	Lớp này là GodClass khi nhận đa trách <u>nhiệm</u> : <u>DateFormat</u> và <u>Logger</u>
6	<u>views.screen.popup</u> (PopupScreen.java)	Class đảm nhiệm nhiều trách nhiệm như xử lý nhiều loại popup không giống nhau	Ba hàm success, error, loading có 3 trách nhiệm khác nhau tương ứng với các loại popup khác nhau
7	<u>views.screen</u> (ViewsConfig.java)	Class đang nắm nhiều trách nhiệm	Class đang chứa các config về PATH, VAT, FONT vốn không liên quan tới nhau

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Single responsibility principle

**Nhận xét:** Có thể thấy, trong source code của dự án hiện tại đang có nhiều class đảm nhiệm nhiều vai trò khác nhau cùng một lúc. Điều này sẽ gây nhiều khó khăn trong việc xác định chính xác tính năng cần chỉnh sửa, hay cập nhật sau này nằm ở đâu, đồng thời gây rối và làm phức tạp không cần thiết cho source code.

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Open-Close principle

#	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
1	<u>controller</u> (PlaceOrderController)	Các hàm validate nếu có bổ sung trong tương lai sẽ phải chỉnh sửa lại hàm validate tổng trong class, chúng nên được tách riêng ra thành class riêng biệt	Các hàm validate nếu có bổ sung trong tương lai sẽ phải chỉnh sửa lại hàm validate tổng trong class, chúng nên được tách riêng ra thành class riêng biệt
2	<u>views.screen.popup</u> (PopupScreen.java)	Các chức năng liên quan tới các loại popup riêng đang được gộp chung trong một class	Các popup như loading, error, Success đang gộp chung trong một file, điều này gây khó khăn cho việc mở rộng thêm một loại popup mới sau này
3	utils (ApplicationProgrammingInterface)	Lớp API này đang chứa hai phương thức cố định là get và post	Mỗi khi thêm một phương thức mới như patch, delete, ... thì phải modify lại mã nguồn của lớp này. Vì thế các nghiệp vụ này nên được tách thành các lớp riêng biệt để có thể dễ dàng mở rộng

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Open-Close principle

**Nhận xét:** Source code vẫn tồn tại vài class vi phạm OCP. Điều này có thể gây khó khăn cho việc cập nhật các tính năng mới sau này.

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Liskov Substitution principle

#	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
1	<u>dao.media</u> (MediaDAO, BookDAO, CDDAO, DVDDAO)	3 lớp BookDAO, CD DAO, DVDDAO đang cùng kế thừa lớp MediaDAO. Lớp MediaDAO dùng để thao tác truy cập lên bảng Media nhưng các lớp kia lại truy cập lên các bảng cụ thể hơn → 3 lớp còn lại vi phạm LSP	Lớp MediaDAO không có lý do gì để làm cha của 3 lớp còn lại. Trong khi phương thức getAllMedia của MediaDAO trả về tất cả các Media thì getAllMedia của các lớp con chỉ trả về danh sách Media của mỗi lớp con ấy. Tương tự với các phương thức khác, các lớp con đã làm thay đổi hành vi của lớp cha và có thể sẽ làm chương trình chạy không đúng kết quả như đối với lớp cha nữa.

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### SOLID - Dependency inversion Principle

#	<i>Module</i>	<i>Mô tả</i>	<i>Lý do</i>
1	<u>view.screen</u> (BaseScreenHandler)	Lớp BaseScreen Handler đang có một thuộc tính là Home ScreenHandler và lớp HomeScreenHandler lại đang là con của lớp BaseScreen Handler.	Lớp BaseScreenHandler vi phạm nguyên lý DIP khi đang phụ thuộc vào lớp có mức độ trừu tượng thấp hơn
2	Các module đang sử dụng trực tiếp lớp CreditCard như: Payment Controller, InterbankInterface, Interbank Subsystem, ...	Lớp PaymentController có một thuộc tính thuộc kiểu CreditCard, còn các lớp, interface khác sử dụng CreditCard như là tham số trong các phương thức	Lớp CreditCard là một lớp có mức độ trừu tượng thấp hơn các lớp khác. Vì thế nếu có một loại thẻ mới cần thêm vào hệ thống thì sẽ không thực hiện được đối với code base hiện tại

## **II. ĐÁNH GIÁ THIẾT KẾ CŨ**

### **Clean code - Clean name**

#### **Vẫn tồn tại biến đặt với tên khó hình dung được ý nghĩa:**

- PaymentController.java: Line 48 - Biến strs không có ý nghĩa gì, khiến code khó hiểu
- MediaDAO.java, Line 54: Tham số tbname không thể hiện được ý nghĩa.

#### **Vài biến đặt tên sai quy tắc:**

- ViewCartController.java: Tên class không nên là động từ, nên chuyển thành CartController.java
- AIMSDB.java: Line 14 - Biến connect là một động từ, nên thay bằng connection

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Clean Function/Method

- **Gọi hàm không hợp lý:** PlaceOrderController.java (Line 78): Bỏ line này hoặc thay bằng LOGGER.info(), không nên dùng system.out.println.
- **Vài hàm có độ phức tạp lớn:** MyMap.java có phương thức toMyMap() quá phức tạp (Nhiều condition, try catch).
- **Nhiều hàm có lượng parameter lớn:** Book.java, CD.java, DVD.java: Constructor có hơn 7 parameter, phức tạp hơn so với mức khuyến cáo (7); ....

## II. ĐÁNH GIÁ THIẾT KẾ CŨ

### Clean Class

#### Nhiều class nắm quá nhiều trách nhiệm

- Class Utils ban đầu là một class rất chung chung và thực hiện nhiều nhiệm vụ khác nhau, có thể trở thành God class, cần phải tách ra
- Class ViewsConfig có nhiều hơn một lý do để thay đổi vì nó chứa: nhóm các thuộc tính Path, REGULAR\_FONT, PERCENT\_VAT, getCurrencyFormat(int num), cần phải tách ra.
- ;...

### **III. ĐỀ XUẤT CẢI TIẾN**

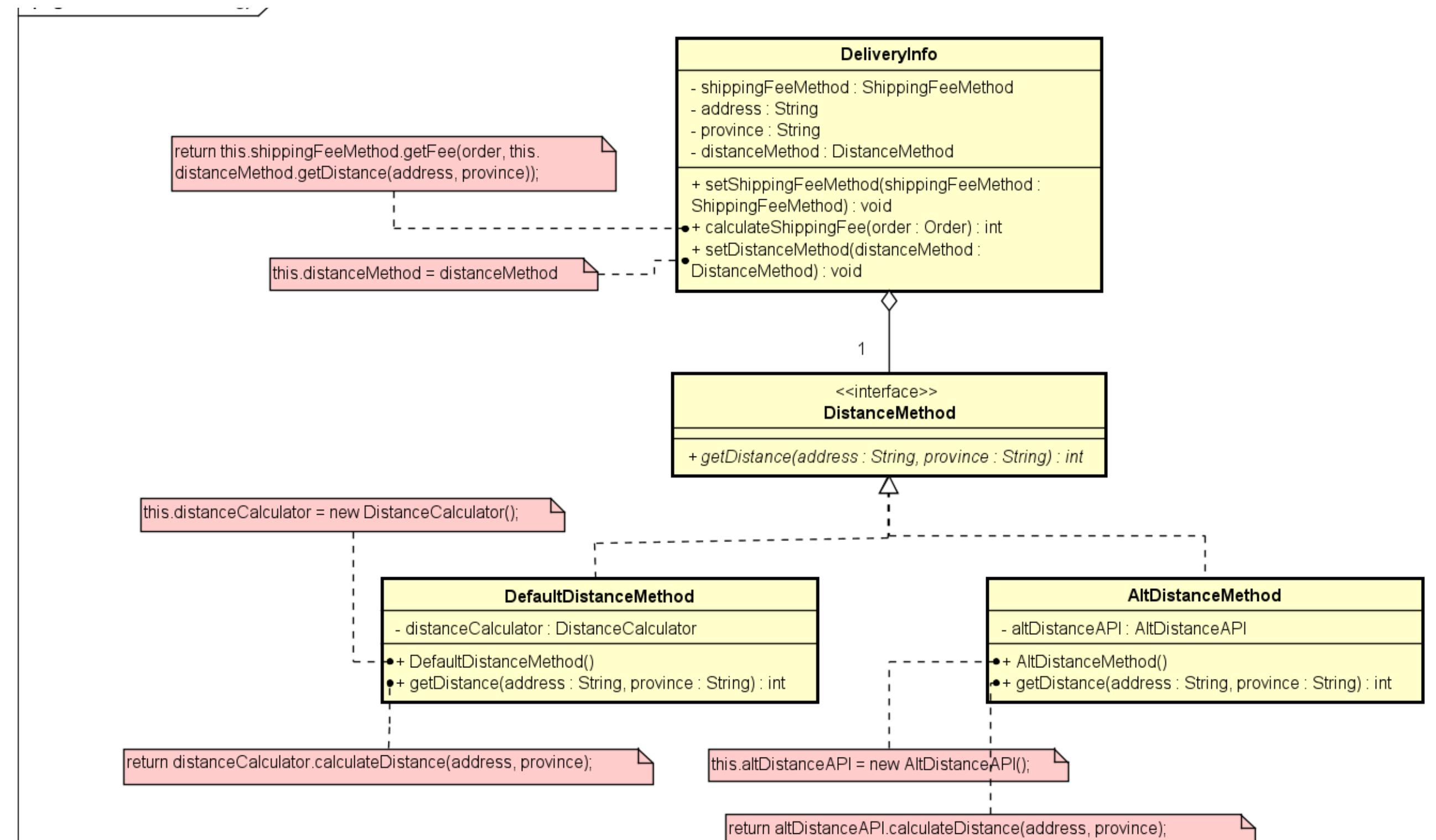
#### **3.1. Vấn đề thay đổi thư viện DistanceCalculator và tương thích với codebase.**

**Mô tả:** Trong tương lai, dự án AIMS có thể sẽ hướng tới thay thế thư viện DistanceCalculator, với cách thức tính khoảng cách mới trong giai đoạn tạo DeliveryInfo.

**Giải pháp:** Áp dụng Adapter Patern để tạo tính tương thích ngược với phiên bản cũ. Đồng thời áp dụng thêm Strategy patern, giúp ta có thể lựa chọn cách tính mới hoặc cách tính cũ.

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.1. Vấn đề thay đổi thư viện DistanceCalculator và tương thích với codebase



### **III. ĐỀ XUẤT CẢI TIẾN**

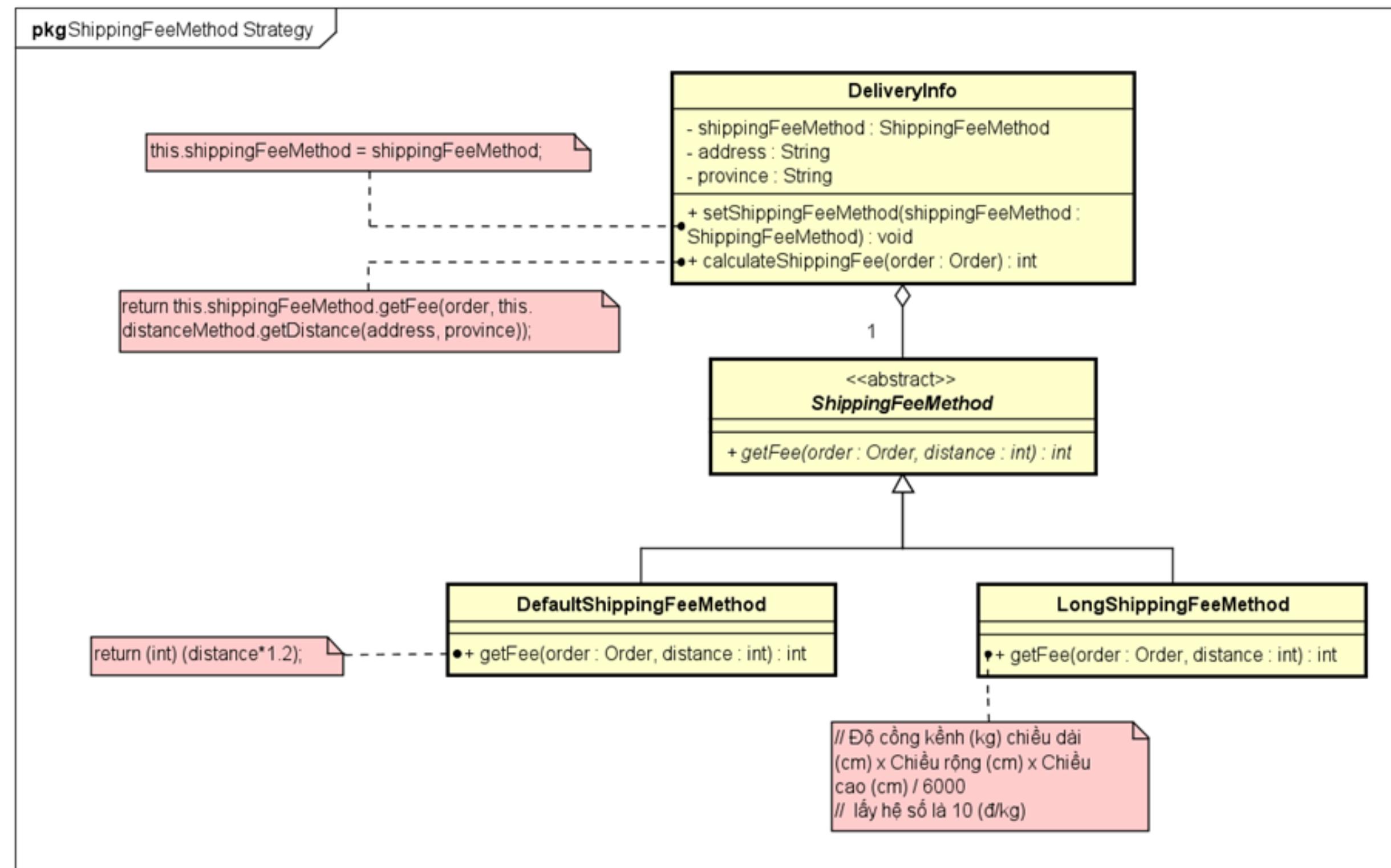
#### **3.2. Vấn đề về chỉnh sửa hệ số tính ShippingFee**

**Mô tả:** Hệ số nhân tính phí vận chuyển đang được đặt giá trị cố định là 1,2. Nếu tương lai ta muốn thay đổi hệ số nhân thì cần phải sửa lại code base.

**Giải pháp:** Dùng Strategy Patern, tạo abstract class ShippingFeeMethod với phương thức trừu tượng getFee(Order order, int distance).

### III. ĐỀ XUẤT CẢI TIẾN

### 3.2. Vấn đề về chỉnh sửa hệ số tính ShippingFee



### **III. ĐỀ XUẤT CẢI TIẾN**

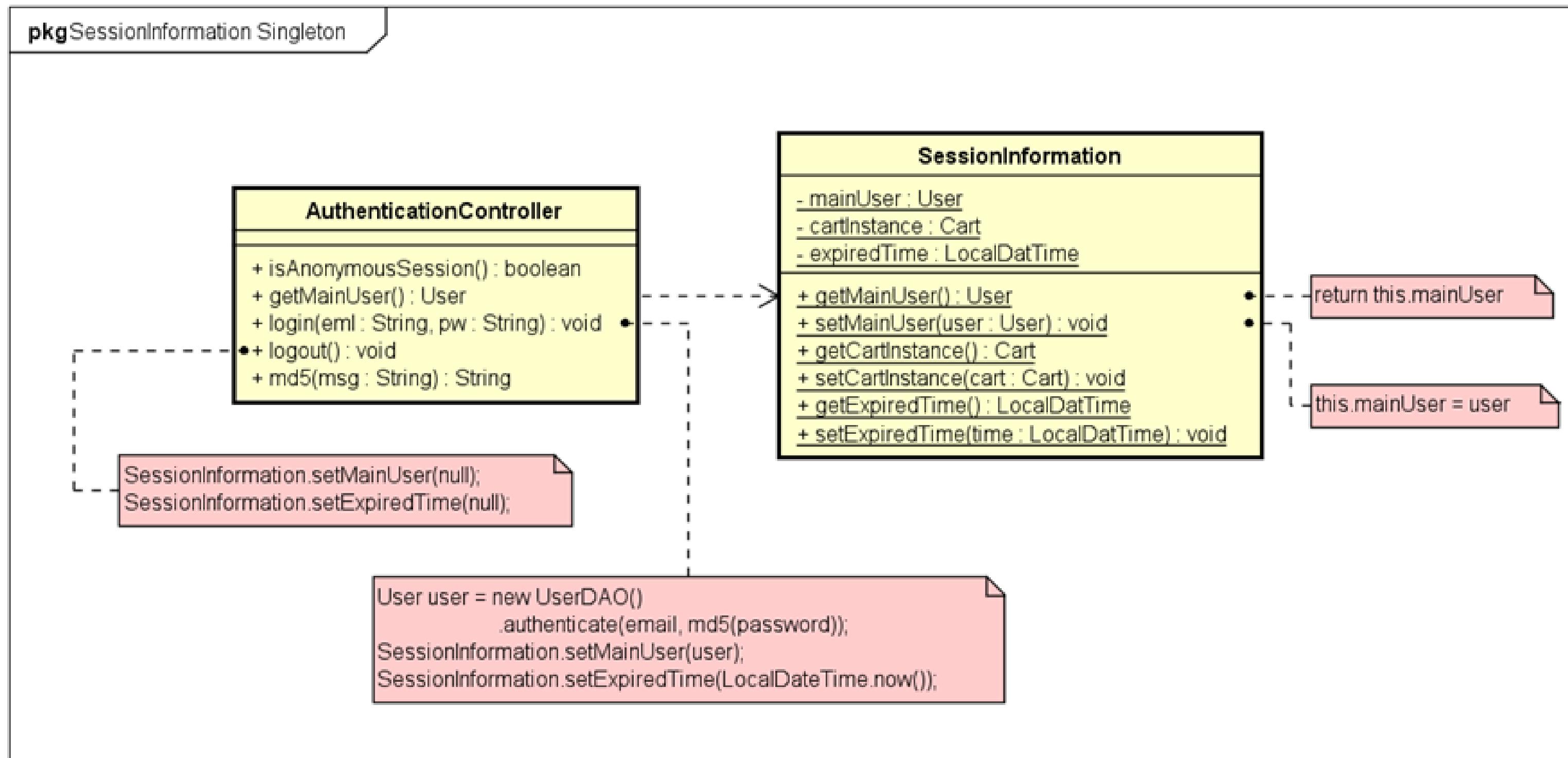
#### **3.3. Vấn đề Common Coupling của SessionInformation**

**Mô tả:** Các biến mainUser, cartInstance và expiredTime của lớp Session Information đang là các biến toàn cục (public static) và được truy cập trực tiếp từ rất nhiều nơi trong hệ thống. Điều này là vi phạm Common Coupling.

**Giải pháp:** Chuyển chỉ định truy cập của các biến toàn cục trên thành private và kiểm soát truy cập bằng cách phương thức getter và setter cho mỗi biến thành phần.

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.3. Vấn đề Common Coupling của SessionInformation



### **III. ĐỀ XUẤT CẢI TIẾN**

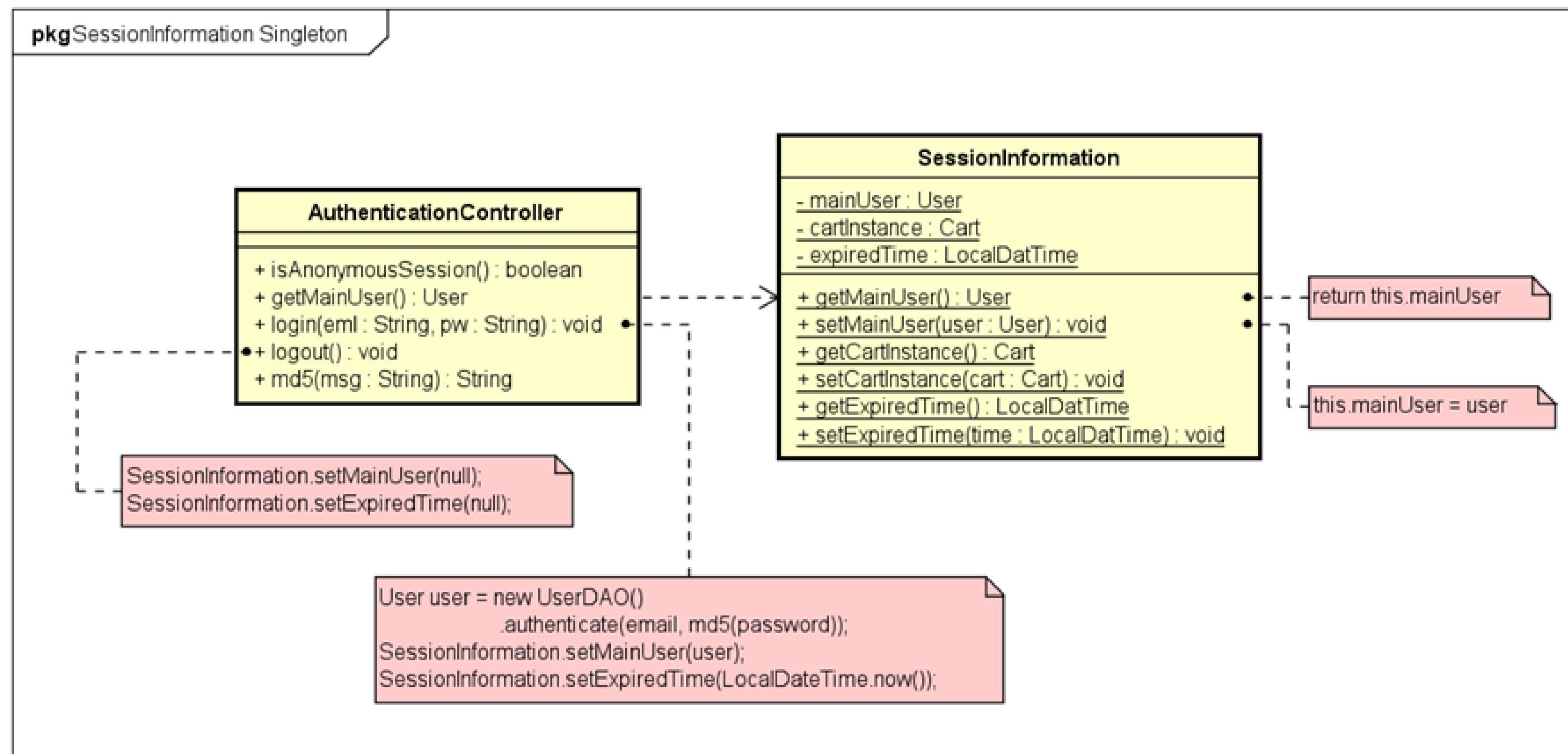
#### **3.4. Vấn đề sự tồn tại duy nhất và toàn cục của thông tin phiên người dùng trong hệ thống**

**Mô tả:** Thông tin về phiên làm việc của người dùng sau khi đã đăng nhập vào hệ thống phải là toàn cục (có thể được truy cập tại bất cứ đâu) và phải là duy nhất.

**Giải pháp:** Sử dụng mẫu thiết kế Singleton cho lớp User. Lớp UserDAO (1 trong những client) có thể sử dụng biến toàn cục cho thông tin phiên user này bằng cách gọi User.getInstance().

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.4. Vấn đề sự tồn tại duy nhất và toàn cục của thông tin phiên người dùng trong hệ thống



### **III. ĐỀ XUẤT CẢI TIẾN**

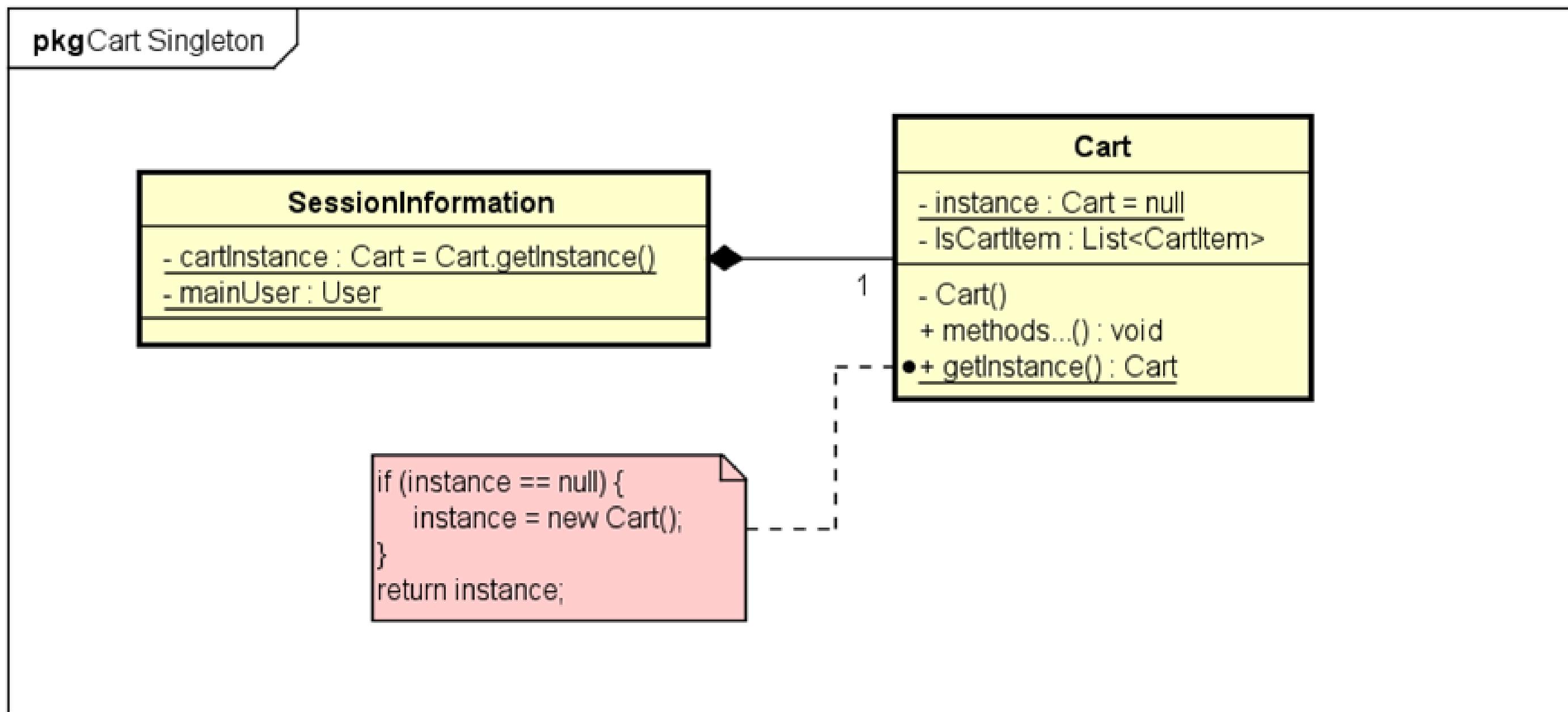
#### **3.5. Vấn đề sự tồn tại duy nhất và toàn cục của giỏ hàng trong hệ thống**

**Mô tả:** Thông tin về giỏ hàng của người dùng khi đã đăng nhập và sử dụng hệ thống phải là toàn cục để có thể lấy được các sản phẩm trong giỏ hàng ở bất cứ đâu và bất cứ khi nào. Giỏ hàng của người dùng cũng phải là duy nhất vì một người dùng không thể có hai giỏ hàng khi sử dụng hệ thống.

**Giải pháp:** Sử dụng mẫu thiết kế Singleton cho lớp Cart. Ở đây biến Cart được sử dụng làm thuộc tính trực tiếp và được khởi tạo ngay khi thông tin phiên được tạo ra bên trong lớp SessionInformation.

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.5. Vấn đề sự tồn tại duy nhất và toàn cục của giỏ hàng trong hệ thống



### **III. ĐỀ XUẤT CẢI TIẾN**

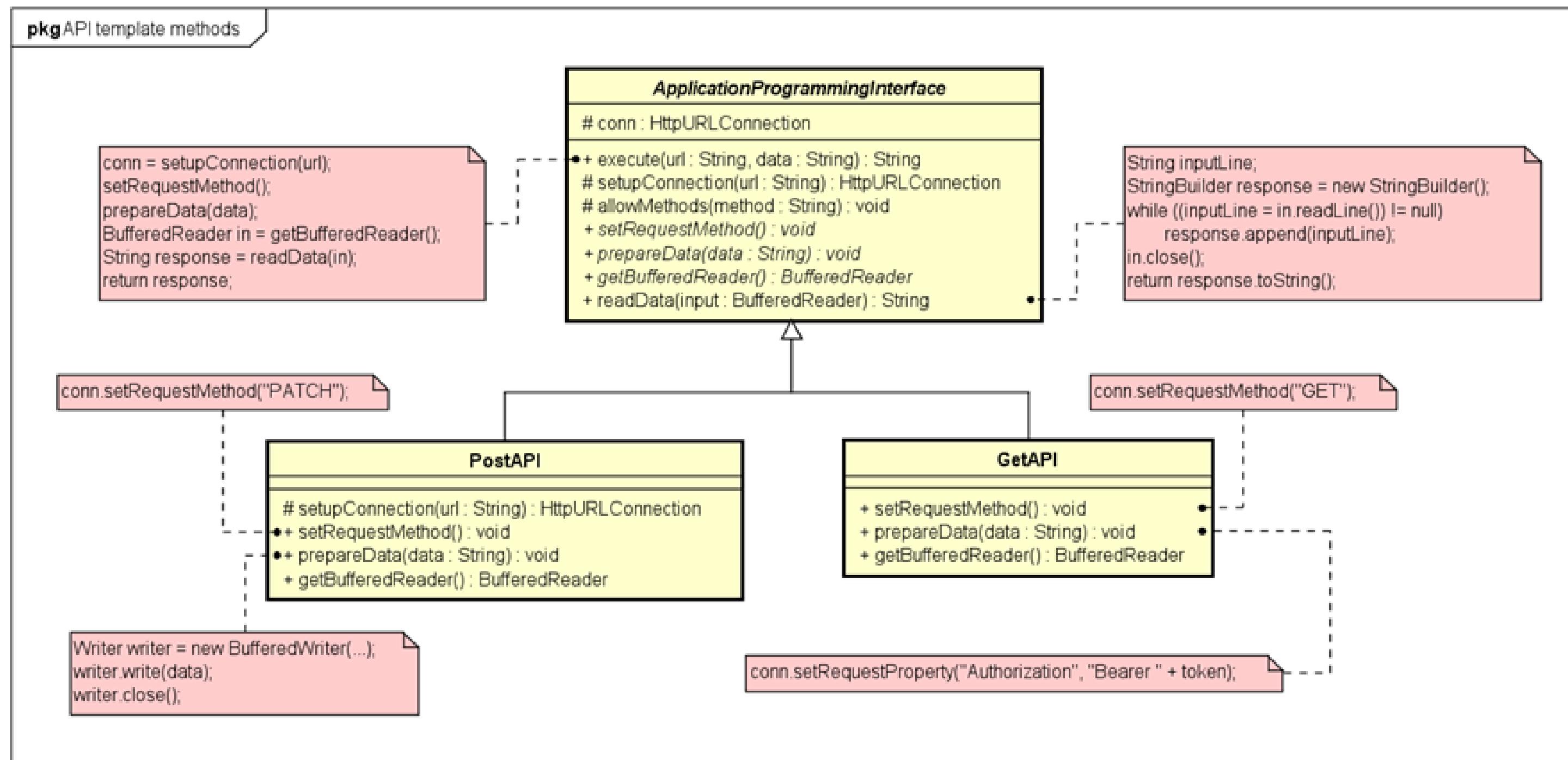
#### **3.6. Vấn đề SRP và OCP trong lớp ApplicationProgramming Interface**

**Mô tả:** Lớp ApplicationProgrammingInterface đang thực hiện cả hai trách nhiệm là post và get. Chưa phù hợp cho việc thêm các phương thức http mới trong tương lai (delete, head, ...)

**Giải pháp:** Chuyển lớp API thành abstract và tách các nghiệp vụ post, get ra thành các lớp riêng cùng kế thừa abstract. Đồng thời sử dụng Design pattern Template Method cho method execute(). Các step trong template method này là: setupConnection, setRequestMethod, prepareData, getBufferedReader, readData.

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.6. Vấn đề SRP và OCP trong lớp ApplicationProgramming Interface



### **III. ĐỀ XUẤT CẢI TIẾN**

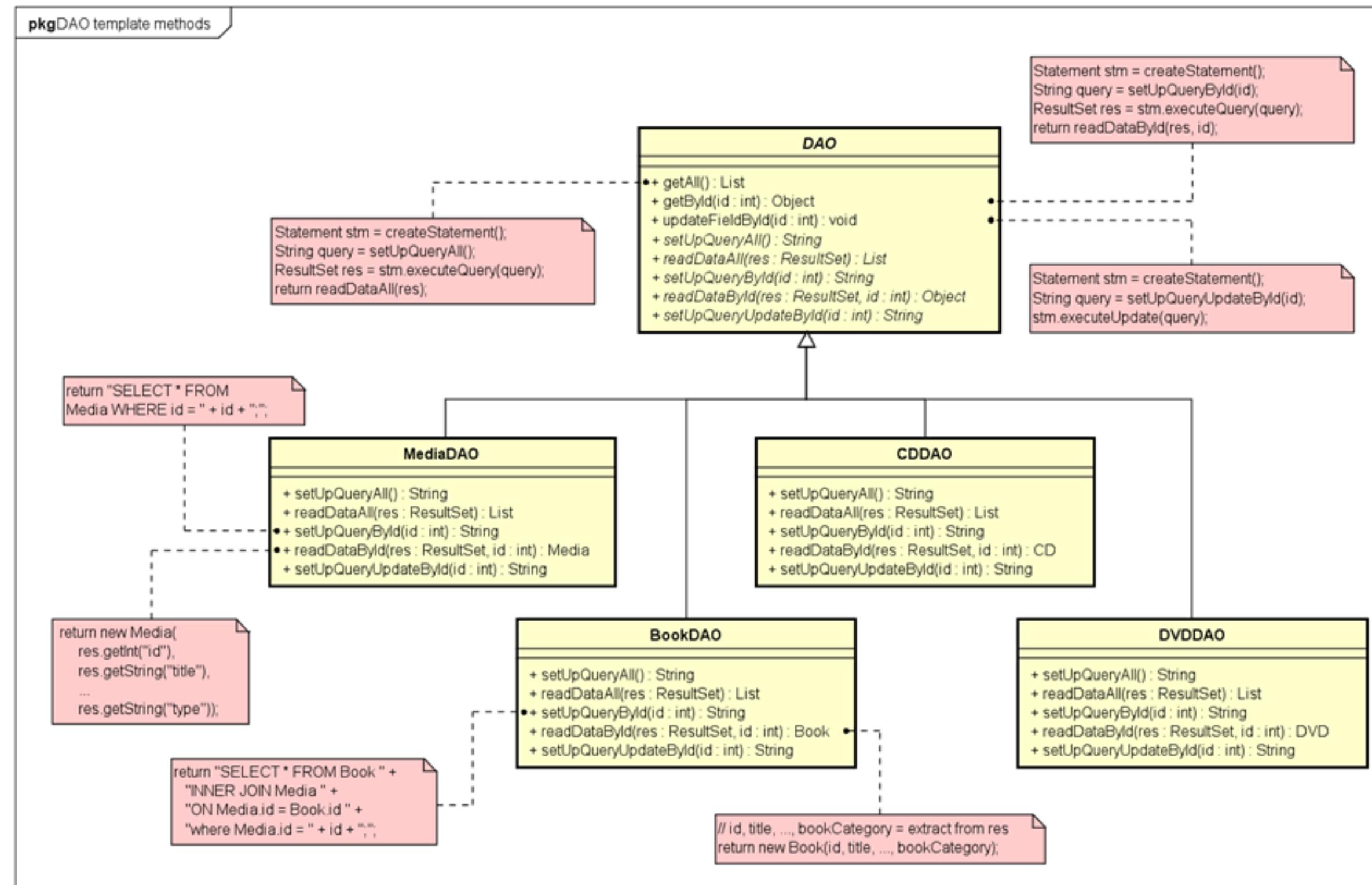
#### **3.7. Vấn đề LSP trong nhóm các lớp Media DAO**

**Mô tả:** Trong khi phương thức getAllMedia của MediaDAO trả về tất cả các Media thì getAllMedia của các lớp con chỉ trả về danh sách Media của mỗi lớp con ấy. Điều này khiến phương thức hoạt động không đúng kết quả như lớp cha đã định.

**Giải pháp:** Tái cấu trúc kế thừa các lớp, trong đó thành lập lớp trừu tượng DAO làm cha của 4 lớp MediaDAO, BookDAO, CDDAO và DVDDAO. Bên cạnh đó, sử dụng Design pattern Template Method cho 3 phương thức là getAll(), getById() và updateFieldById().

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.7. Vấn đề LSP trong nhóm các lớp Media DAO



### **III. ĐỀ XUẤT CẢI TIẾN**

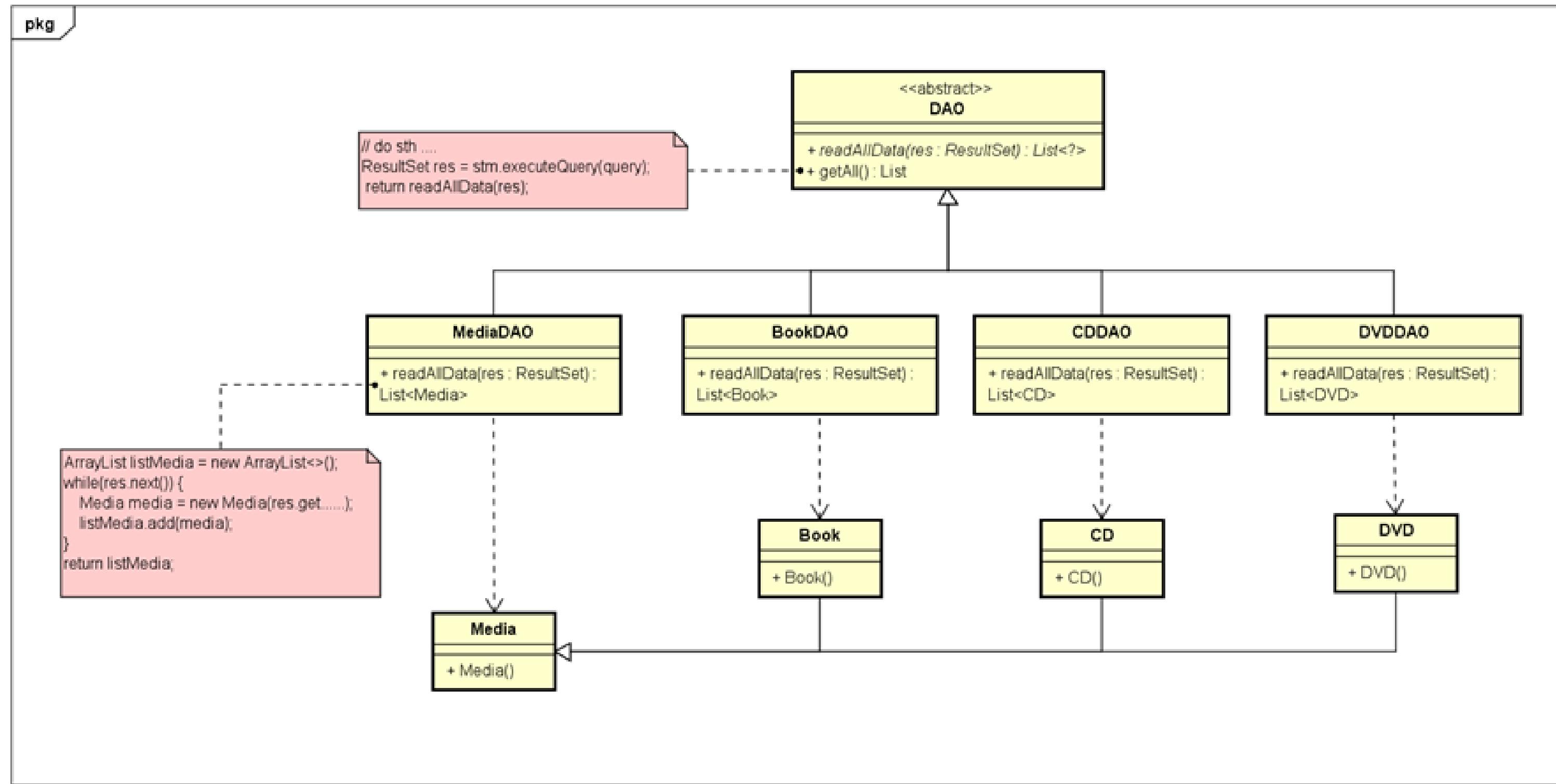
#### **3.8. Sử dụng factory method sau khi tái cấu trúc các lớp Media DAO**

**Mô tả:** Sau khi tái cấu trúc lại, các lớp DAO sẽ có chung method getAll() sẽ thực hiện setUpQuery để truy cập database và sau đó gọi readAllData(res) trả ra danh sách mọi đối tượng thuộc loại Media tương ứng

**Giải pháp:** Sử dụng Factory Method để khi sử dụng chỉ cần gọi phương thức readAllData(res) mà không quan tâm tới việc từng danh sách cho mỗi loại (Media, Book, CD, DVD) được tạo như nào.

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.8. Sử dụng factory method sau khi tái cấu trúc các lớp Media DAO



### **III. ĐỀ XUẤT CẢI TIẾN**

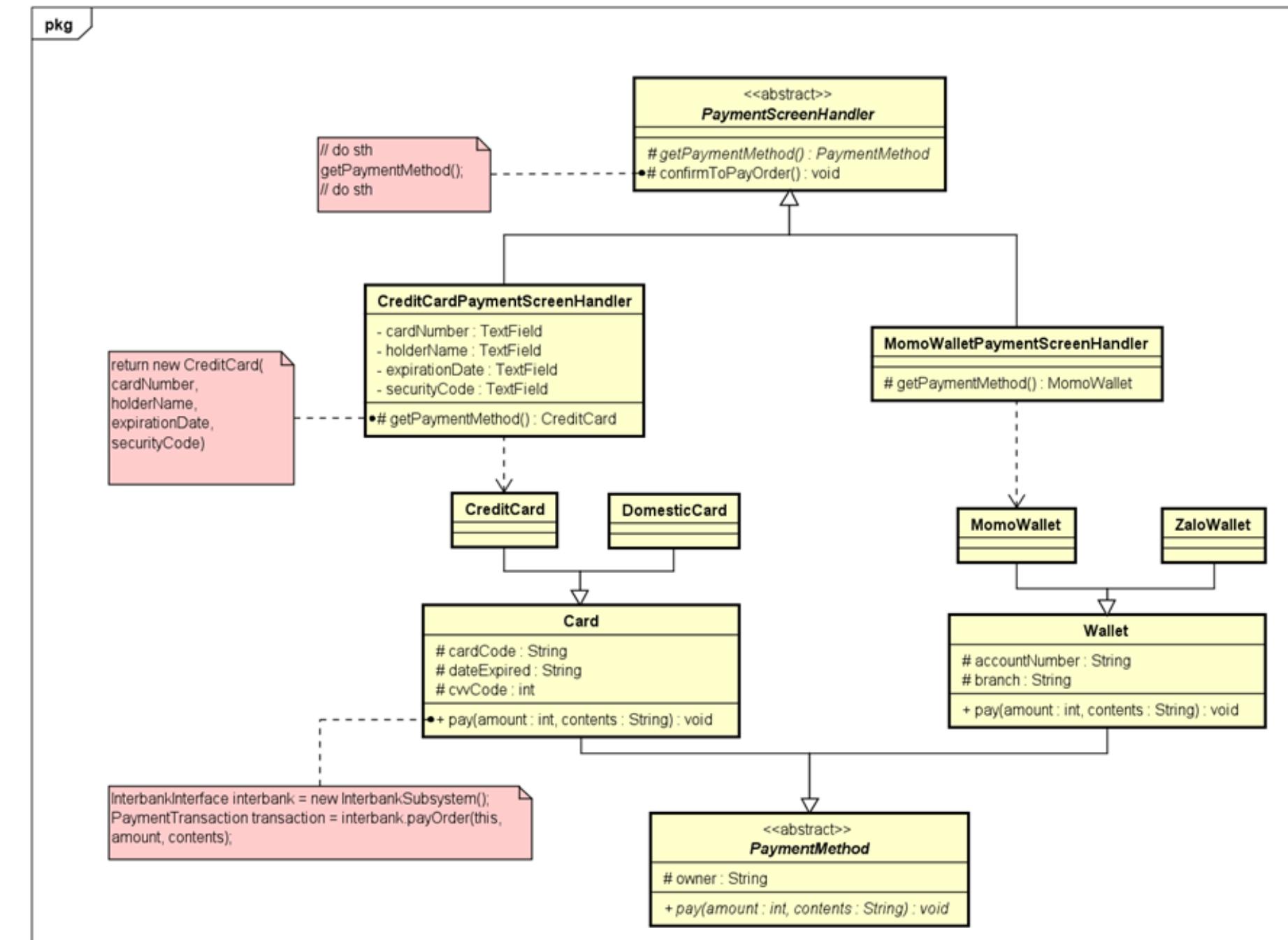
#### **3.9. Vấn đề DIP, OCP đối với các lớp ở mức cao phụ thuộc vào lớp cụ thể CreditCard**

**Mô tả:** Các lớp PaymentController, PaymentTransaction, InterbankPayloadConverter, InterbankInterface (mức cao hơn) lại phụ thuộc vào lớp CreditCard (lớp cụ thể hơn). Trong tương lai khi mở rộng cơ chế thanh toán với nhiều loại thẻ và có thể có thêm ví thanh toán, sẽ phải modify lại rất nhiều ở các lớp đó.

**Giải pháp:** Tái cấu trúc lại cây thanh toán và màn hình sử dụng của từng loại, sử dụng factory method.

### III. ĐỀ XUẤT CẢI TIẾN

#### 3.9. Vấn đề DIP, OCP đối với các lớp ở mức cao phụ thuộc vào lớp cụ thể CreditCard



## IV. TỔNG KẾT





Hanoi University of Science  
and Technology

**CẢM ƠN ĐÃ LẮNG NGHE**

Group 10