**Nguyên lý thiết kế SOLID**

1. ***Single Responsibility***

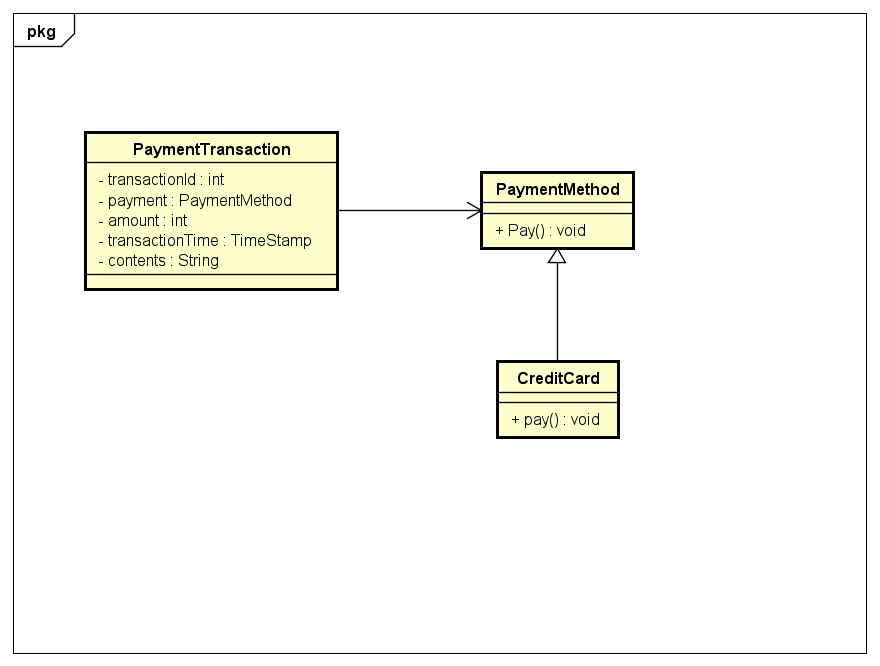
* Hiện tại các lớp được thiết kế và triển khai cho usecase “Place Rush Order” đang đáp ứng được việc mỗi lớp chỉ chịu trách nhiệm cho một nhiệm vụ nhất định.

1. ***Open/Closed***

* Khi thiết kế subsystem cho Interbank nếu sau khi mở rộng, chúng ta có thể sử dụng Interbank khác với giao thức kết nối và API thay đổi thì chúng ta chỉ cần viết một subsystem khác implement các phương thức pay().

**Giải pháp:**

* Tạo ra một lớp abstract là PaymentMethod và lớp PaymentTransaction chỉ quan tâm đến lớp PaymentMethod này mà không bị phụ thuộc vào lớp CreditCard cụ thể.
* Dễ thấy, giải pháp dưới đây chính là một ví dụ cho cách áp dụng Strategy Pattern vào bản thiết kế.



1. ***Liskov Substitution***

* Với các lớp được thiết kế và triển khai cho “UC Place Rush Order” đang đáp ứng được việc các lớp con có thể thay thế cho lớp cha ở mọi tình huống mà không gây ra lỗi.
* Chẳng hạn như việc 3 lớp entity DVD, Book, CD cùng kế thừa entity Media thì những phương thức được viết chung trong lớp Media thì các lớp kế thừa trên không phải viết lại, tức bỏ override trên mỗi phương thức dùng lại của lớp cha đó.

1. ***Interface Segregation***

Vấn đề xảy ra tại InterbankInterface với hai phương thức payOrder() và refund() đều được các lớp con implement lại. Giả sử trong tương lai hệ thống của chúng ta kết nối tới một Interbank khác mà không dùng tới phương thức refund() thì khi đó thiết kế của chúng ta sẽ bị vi phạm nguyên tắc này.

**Giải pháp:**

* Tách InterbankInterface thành hai interface khác là PayInterface với một phương thức payOrder() và RefundInterface với một phương thức refund(), khi đó vấn đề mà chúng ta đang đặt ra đã được giải quyết mà không phải thay đổi lại thiết kế ban đầu của chúng ta.

1. ***Dependency Inversion***

Vấn đề của thiết kế này đã được nêu và đưa ra giải pháp giống như trong nguyên lý Open/Closed phía trên.