Link github: <https://github.com/BacDung/NguyenVanDung_20183900_710808.git>

# Mục lục

[I. Yêu cầu mở rộng 2](#_Toc91425789)

[1. Thay đổi phí ship 2](#_Toc91425790)

[2. Sử dụng interbank khác 3](#_Toc91425791)

[II. Nguyên lý thiết kế SOLID 4](#_Toc91425792)

[1. Single Responsibility 4](#_Toc91425793)

[2. Opent/Closed 4](#_Toc91425794)

[3. Liskov Substitution 4](#_Toc91425795)

[4. Interface Segregation 4](#_Toc91425796)

[5. Dependency Inversion 4](#_Toc91425797)

## Yêu cầu mở rộng

### Thay đổi phí ship

Tạo một interface

Text

Description automatically generated

Trong interface này có phương thức calculateshipingfee, phương thức này để tính phí ship.

Text

Description automatically generated Text

Description automatically generated

Cài đặt interface này bằng 1 class ShippingFee1

Trong PlaceOrderController sử dụng calculateShippingFee mà không cần biết nó được cài đặt như thế nào

### Sử dụng interbank khác

Trong package interbank ta thêm 1 subsystem của interbanknew và implement lại interface interbank

Text

Description automatically generated

Tương tự, cài đặt thêm 1 subsystem

Text

Description automatically generated

## Nguyên lý thiết kế SOLID

### Single Responsibility

* Sau khi thêm UC Place Rush order ta thêm 1 class PlaceRushOrderController.
* Lớp này để đáp ứng mở rộng giao hàng nhanh
* Và chỉ có 1 nhiệm vụ đó là điều khiển giao hàng nhanh nên sau khi thêm không làm ảnh hưởng nhiều đến thiết kế chung

### Opent/Closed

* 1 vài yêu cầu mở rộng như thêm 1 interbank khác nó sẽ làm ảnh hưởng khá nhiều đến hệ thống nếu như không tìm ra phương án thiết kế tối ưu.
* Việc chúng ta thiết kế các subsystem giúp các modul dễ dàng mở rộng và bảo trì
* Trong thiết kế trong prj này khi thêm 1 interbank khác chúng ta thêm 1 subsystemInterbankNew mà không làm ảnh hưởng đến thiết kế cũ
* Đảm bảo nguyên lý Opent/Closed

### Liskov Substitution

* Nguyên tắc này nói rằng, các đối tượng của class con có thể thay thế cho lớp cha ở mọi tính huống mà không gây ra lỗi, hoặc nếu không, chúng ta có sự trừu tượng sai.
* Có thể thấy hệ thống phân cấp kế thừa ở class BaseController đã tuân theo nguyên lý này.
* Class PlaceRushOrder kế thừa PlaceOrder nhưng khi ta thay thế 2 class này cho nhau ở mọi trường hợp thì cũng không sảy ra l

### Interface Segregation

* Nguyên lý này nói rằng, thay vì một sử dụng một interface quá lớn, quá nhiều phương thức thì chúng ta sẽ tách nhỏ ra thành các interface con với mục đích cụ thể. Vì khi để một interface quá to, các lớp implement sẽ phải implement các phương thức mà bản thân nó không cần dùng đến.
* Thiết kế hiện tại về cơ bản đã đáp ứng được nguyên tắc này, ví dụ với InterbankInterface, cả 2 phương thức payOrder và refund đều được lớp InterfaceSystemController implement.

### Dependency Inversion

* Có thể hiểu nguyên lý này như sau: những thành phần trong một chương trình chỉ nên phụ thuộc vào những cái trừu tượng. Những thành phần trừu tượng không nên phụ thuộc vào một thành phần mang tính cụ thể mà nên ngược lại.