**Tên Yorn Bunthoeurn**

**Mssv: 20170287**

**Lớp CNTT 09 – K62**

**Thực hành 5:**

**Review design của AIMS (và UC "Place rush order")**

**và chỉ ra được các vấn đề:**

* Coupling và Cohesion
* Vi phạm nguyên lý SOLID hay không sau khi thêm các yêu cầu mở rộng.

### Coupling và Cohesion

* 1. Coupling
     1. Content coupling

Content coupling: xảy ra khi 1 module xử dụng code của 1 module khác, điều này sẽ không đúng với những khái niệm thiết kế cơ bản là đóng gói thông tin

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Không có |  |  |

* + 1. Common coupling

Common coupling: Xảy ra khi các module cùng truy cập vào cùng data global, và điều này có thể dẫn đến việc không thể kiểm soát được lỗi và sẽ xảy ra những lỗi không mong muốn

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Không có |  |  |

* + 1. Control coupling

Control coupling: là khi 1 module đang control luồng hoạt động của 1 module khác thông qua các tham số đầu vào, và nó có thể dẫn đến 1 vài trường hợp không mong muốn khi các tham số truyền vào không được đúng

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| paymentMethod | Thông thường, sẽ sử dụng control structure như if-else và sẽ gặp phải vấn đề về control coupling | Tách phương thức  Áp dụng các design patterns như strategy hoặc factory pattern |

* + 1. Stamp coupling

Stamp Coupling: xảy ra khi các module chỉ sử dụng các thành phần cấu trúc dữ liệu được chia sẻ với nhau, và trong trường hợp này, khi thay đổi 1 thuộc tính của 1 module có thể sẽ ảnh hưởng đến các module khác đang phụ thuộc vào nó

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Hai lớp khi sử dung cùng nhau | Hai lớp được coi là vi phạm stamp coupling nếu một lớp gửi một collection hoặc một object dưới dạng tham số và chỉ một vài phần data được sử dụng ở lớp thứ hai | Để giảm thiểu level coupling này, chúng ta truyền đủ những tham số cần thiết trong function gọi. Chú ý rằng, stamp coupling là mức có thể chấp nhận được |

* + 1. Data coupling

Data coupling: xảy ra khi 2 hoặc nhiều module độc lập với nhau, tương tác, chia sẻ tài nguyên cho nhau và thông thường là qua tham số đầu vào

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Không có |  |  |

* + 1. Uncoupled

Nếu các modules là uncoupled, vui lòng tách chương trình của bạn thành các module độc lập.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Không có |  |  |

* 1. Cohesion
     1. Coincidental cohesion
* Các sub component đặt trong 1 component vì tính ngẫu nhiên
* Rõ ràng, chúng ta có thể thấy loại cohesion này trong class Config hoặc class Utils trong package utils

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Class Config hoặc class Utils | Có 1 component trong class |  |

* + 1. Logical cohesion

Các thành phần trong 1 module có liên quan đến nhau nhưng về mặt logic chứ không phải chức năng.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| PaymentMethod | nếu chia đoạn code thành 2 methods và đưa chúng vào trong cùng một lớp, chúng ta có thể đối mặt với logical cohesion | Tách phương thức class hoặc package khác |

* + 1. Temporal cohesion
* Các sub component đặt trong một component vì chúng liên quan đến nhau về mặt thời gian chứ không phải về mặt chức năng.
* Thông thường, chúng ta cần loại cohesion này trong các module khởi tạo hoặc clean-up.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Không có |  |  |

* + 1. Procedural cohesion

Các thành phần đặt trong 1 module vì nó có quan hệ chặt chẽ với nhau theo một thứ tự nào đó chứ không liên hệ với nhau về mặt chức năng.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| PlaceOrderController | validate các trường dữ liệu từng bước một trong package control | validation |

* + 1. Communicational cohesion

Các thành phần đặt trong cùng một module vì cùng thực hiện trên cùng một dữ liệu.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| InterbankSubsystemController | các thành phần trong InterbankSubsystemController đều nhận các dữ liệu đầu vào giống nhau và dữ liệu đầu ra cùng trả về kiểu PaymentTransaction |  |

* + 1. Sequential cohesion
* Trong một module, output của thành phần này là input của thành phần kia.
* Chỉ có một vấn đề nhỏ ở đây. Nếu chúng ta có 1 trình tự, chúng ta có thể chia nó thành nhiều phần khác nhau. Có nghĩa là các class được tạo ra có thể làm nhiều hơn một chức năng hoặc hoàn toàn ngược lại - chỉ một phần của chức năng.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Không có |  |  |

* + 1. Informational cohesion

Các operation có tính độc lập (có input và output riêng nhưng chúng có thể thao tác trên một tập dữ liệu chung là attribute của lớp đó)

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Entity | Cohesion này ở trong các lớp entity như Media và Order |  |

* + 1. Functional cohesion

Mỗi một subcomponent thực hiện một công việc nào đó và hướng đến mục đích chung của component đó.

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Class API | Đầu ra phương thức setUpConnection() là đầu vào cho phương thức get(). | Refactor |

### Nguyên lý thiết kế SOLID

Đối với Design Principles ,SOLID đó là 5 chữ cái viết tắt của 5 nguyên lý cơ bản trong thiết kế: Single responsibility principle, Open/Closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle.

* 1. Single Responsibility Principle

Một class chỉ nên giữ 1 trách nhiệm duy nhất (Chỉ có thể sửa đổi class với 1 lý do duy nhất)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Related modules | Description | Improvement |
| 1 | Không có |  |  |

* 1. Open/Closed Principle

Có thể thoải mái mở rộng 1 class, nhưng không được sửa đổi bên trong class đó (open for extension but closed for modification).

|  |  |  |  |
| --- | --- | --- | --- |
| # | Related modules | Description | Improvement |
| 1 | PlaceOrderController | sẽ tạo ra một interface ShippingFeeCalculator với phương thức trừu tượng là calculateShippingFee | ShippingFeeCalculate |

* 1. Liskov substitution principle

Trong một chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình

|  |  |  |  |
| --- | --- | --- | --- |
| # | Related modules | Description | Improvement |
| 1 | getAllMedia | Chương trình sau đó vẫn chạy vì phương thức getAllMedia() của lớp cha Media không phải phương thức abstract, không bắt buộc phải override | Xóa đoạn code Override của lớp Media |

* 1. Interface segregation principle

Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể

|  |  |  |  |
| --- | --- | --- | --- |
| # | Related modules | Description | Improvement |
| * 1. 1   1 | Payorder, refund đều được lớp InterfaceSystemController implement. | có một số hệ thống interbank khác không hoàn tiền cho khách hàng mà chỉ thanh toán, lúc này phương thức refund của InterbankInterface trở nên dư thừa đối với interbanksubsystem đó |  |

* 1. Dependency inversion principle
* Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.
* Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Related modules | Description | Improvement |
| 1 | PaymentTransaction | PaymentTransaction đang phụ thuộc chặt chẽ vào lớp CreditCard, sau này giả sử không sử dụng CreditCard để thanh toán mà sử dụng một loại phương thức thanh toán khác | tạo một lớp abstract là PaymentCard và lớp PaymentTransaction chỉ quan tâm đến lớp PaymentMethod này. |