Assignment 10

CÁC KHÁI NIỆM VÀ NGUYÊN LÝ THIẾT KẾ PHẦN MỀM

1. MỤC ĐÍCH VÀ NỘI DUNG

- Trong bài thực hành này, người học sẽ làm quen với các khái niệm thiết kế (Design Concepts) và các nguyên lý cơ bản trong thiết kế (Design Principles)
- Đối với Design Concepts, người học sẽ được tiếp cận với 2 khái niệm chính là cohesion và coupling.
- Đối với Design Principles, người học sẽ được làm quen với 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.
- Sau bài thực hành, người học sẽ có thêm những ý tưởng để tạo ra một bản thiết kế tối ưu, dễ bảo trì, mở rộng…

2. CHUẨN BỊ

- Người học cần hiểu được các kiến thức lý thuyết về Design Concept và Design Principle đã được học ở trên lớp trước khi bắt tay vào thực hành.
- Lưu ý các hướng dẫn dưới đây chỉ là 1 số gợi ý chứ không phải là tất cả những gì cần tối ưu cho bản thiết kế và mã nguồn.
- Bản thiết kế và mã nguồn được xem xét trong bài thực hành này có thể clone trên https://github.com/leminhnguyen/AIMS-Student

3. NỘI DUNG CHI TIẾT

1. Một số yêu cầu mở rộng

Dưới đây là danh sách các yêu cầu mở rộng:

a) Thay đổi phí ship:

- Phí vận chuyển hiện nay cũng phụ thuộc vào trọng lượng thực tế, khoảng cách và độ cồng kềnh của sản phẩm (tức là kích thước của sản phẩm: chiều dài, chiều rộng, chiều cao)
- Công thức chuyển đổi được đưa ra như sau:
- Alternative weight (kg) =Length (cm) × Width (cm) × Height (cm) / 6000
- Trọng lượng của sản phẩm bằng trọng lượng thực tế cộng với trọng lượng thay thế.

b) Thêm một cách tính tiền mới:

- Domestic debit card:
  - Issuing bank, e.g., VietinBank
  - Card number: 16 digits
  - Valid-from date, e.g., 12/33
  - Cardholder's name, e.g., VU DUY MANH
- Đối với các yêu cầu mở rộng này, giả sử API vẫn giữ nguyên, giả sử thông tin thẻ được thay đổi.

c) Sử dụng interbank khác

- Sử dụng một interbank khác với giao thức kết nối và API thay đổi.

2. Coupling và Cohesion: các khái niệm cơ bản trong thiết kế

2.1. Coupling

- Để có một bản thiết kế tốt thì cần phải thiết kế sao cho coupling lỏng để khi có sự thay đổi ở một module thì ảnh hưởng ít nhất có thể đến các module khác. Coupling càng lỏng lẻo, càng tốt. Trong phần này, chúng ta sẽ xem xét về các mức coupling giữa các module trong một project mẫu.

a) Content coupling

- May mắn thay, project của chúng ta hiện tại chưa có loại mức độ coupling này.
- Để minh chứng cho điều này, hãy xem class Order trong package entity.order. Hiện tại, lớp này có thuộc tính deliveryInfo có kiểu HashMap và một getter public: getDeliveryInfo() cho thuộc tính này. Trong tương lai, có thể có một module quản lý để lấy đối tượng deliveryInfo bằng cách gọi getDeliveryInfo() của lớp Order và sau đó thay đổi deliveryInfo bằng cách gọi put() của lớp HashMap. Do đó, giá trị của deliveryInfo sẽ bị thay đổi trong khi object Order không biết gì về việc sửa đổi thuộc tính của nó từ bên ngoài.
- Để giảm mức độ ghép nối, chúng ta có thể:
  - Bỏ qua những accessors/getters dư thừa
  - Chọn chỉ định truy cập phù hợp.
  - Đóng gói dữ liệu bằng một lớp / đối tượng chuyên biệt.
  - Sử dụng design pattern như builder patterns.
- Hãy nhớ nghĩ đến mức độ phụ thuộc giữa các mô-đun khi thiết kế.

b) Common Coupling

- Đây là trượng hợp 2 module cùng chia sẻ chung dữ liệu, 2 global structure có thể cùng vào chỉnh sửa, truy cập hoặc có chung những khối mã nguồn thì sẽ vi phạm common coupling.
- Tuy nhiên, object oriented programming không có common data. Tất cả data đều thuộc về lớp. Ví dụ khi dùng C, bạn có data structure khai báo là global, sau đó sẽ có một số phương thức ghi vào structure, các phương thức khác đọc từ đó, trường hợp này sẽ vi phạm common coupling. Còn trong Object Oriented, không có bất cứ data nào mà không thuộc về lớp. Do đó, project hiện tại (JAVA) không vi phạm common coupling.

c) Control Coupling

- Một module vi phạm control coupling khi nó truyền tham số điều khiển cho các module khác thông qua việc gọi method. Điều này không tốt vì thành phần được gọi sẽ biết được cấu trúc bên trong thành phần gọi và khi cấu trúc này bị thay đổi thì thành phần được gọi sẽ phải thay đổi theo.
- Trong trường hợp sau đây, điều gì sẽ xảy ra nếu có nhiều phương thức thanh toán.
- Thông thường, chúng ta sẽ sử dụng control structure như if-else và sẽ gặp phải vấn đề về control coupling như sau:

```
if (paymentMethod == "NGAN_LUONG" ){
      NGANLUONGSubsystem nganLuong = new NGANLUONGSubsystem();
      nganLuong.pay(args);
} else {
      OceanBankSubsystem oceanbank = new OceanBankSubsystem();
      oceanbank.pay(args);
}
```

- Cách xử lý:
  - Tách phương thức
  - Áp dụng các design patterns như strategy pattern hoặc factory pattern

d) Stamp coupling

- 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.

e) Data coupling

- Nếu các module của bạn ở mức data coupling, thiết kế của bạn là thiết kế tốt. Đây là level mà chúng ta hướng đến.

f) Uncoupled

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

2.2. Cohesion

- Nhìn chung, để tăng mức độ (level) cohesion, chúng ta có thể thử đặt mỗi phần vào một module khác phù hợp hơn (tạo module mới nếu cần)

a) 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

b) Logical cohesion

- Các thành phần trong 1 module có liên quan đến nhau nhưng về mặt logic chứ không phải chức năng.
- Quan sát lại phần Control Coupling bên trên, nếu chúng ta chia đoạn code thành 2 methods và đưa chúng vào trong cùng một lớp, chúng ta có thể đối mặt với logical cohesion. Do đó, chúng ta cần xem xét đưa chúng vào trong những class hoặc package khác.

c) 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.
- Ví dụ: viết một phương thức khởi tạo tất cả các thành phần của hệ thống ➔ vi phạm Temporal cohesion: thành phần này đi khởi tạo dữ liệu cho thành phần khác, thay vì vậy ta nên gọi đến thành phần khởi tạo của từng thành phần.

d) 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.
- Chúng ta có thể thấy loại cohesion này ở trong class PlaceOrderController trong package control. Chúng ta validate các trường dữ liệu từng bước một với các phương thức validation.

```
public void validateDeliveryInfo(HashMap<String, String> info) throws InterruptedException, IOException{

}

public boolean validatePhoneNumber(String phoneNumber) {
    // TODO: your work
    return false;
}

public boolean validateName(String name) {
    // TODO: your work
    return false;
}

public boolean validateAddress(String address) {
    // TODO: your work
    return false;
}
```

e) Communicational cohesion

- Các thành phần đặt trong cùng một module vì cùng thực hiện trên cùng một dữ liệu.
- Các module theo kiểu cohesion này hoạt động trên cùng một đầu vào hoặc trả về cùng một đầu ra (ví dụ: các thành phần trong InterbankSubsystemController đều nhận các dữ liệu đầu vào giống nhau và dữ liệu đầu ra cùng trả về kiểu PaymentTransaction)

```
/**
 * @see InterbankInterface#payOrder(entity.payment.CreditCard, int,
 *      java.lang.String)
 */
public PaymentTransaction payOrder(CreditCard card, int amount, String contents) {
    PaymentTransaction transaction = ctrl.payOrder(card, amount, contents);
    return transaction;
}

/**
 * @see InterbankInterface#refund(entity.payment.CreditCard, int,
 *      java.lang.String)
 */
public PaymentTransaction refund(CreditCard card, int amount, String contents) {
    PaymentTransaction transaction = ctrl.refund(card, amount, contents);
    return transaction;
}
```

f) 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.

g) 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)
- Chúng ta có thể thấy rõ loại cohesion này ở trong các lớp entity như là Media hay Order.

h) Functional cohesion

- Mỗi một subcomponent thực hiện một công việc nào đó và hướng đến mục đích chung của component đó.
- Nhìn lại class API sau khi đã được refactor ở bài lab trước, bạn có thể thấy rõ được, đầu ra phương thức setUpConnection() là đầu vào cho phương thức get().

```
public static String get(String url, String token) throws Exception {
    LOGGER.info("Request URL: " + url + "\n");
    URL line_api_url = new URL(url);
    HttpURLConnection conn = setUpConnection(token,"GET", line_api_url);
    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String inputLine;
    StringBuilder respone = new StringBuilder(); // ising StringBuilder for the sake of memory and performance
    while ((inputLine = in.readLine()) != null)
        System.out.println(inputLine);
    respone.append(inputLine + "\n");
    in.close();
    LOGGER.info("Respone Info: " + respone.substring(0, respone.length() - 1).toString());
    return respone.substring(0, respone.length() - 1).toString();
}

private static HttpURLConnection setUpConnection(String token, String method, URL line_api_url)
        throws IOException, ProtocolException {
    HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod(method);
    conn.setRequestProperty("Content-Type", "application/json");
    conn.setRequestProperty("Authorization", "Bearer " + token);
    return conn;
}
```

3. BÀI TẬP CÁ NHÂN

Review Design sau khi đã thêm UC “Place Rush Order”

3.1. Coupling và Cohesion

Trong phần này bạn cần:

- Review lại các mức coupling và cohesion và xác định các mức độ này trong modules của thiết kế cho"UC Place Rush Order". Nếu thiết kế chưa tốt, hãy đề xuất giải pháp cải thiện.

- Viết báo cáo cho các vấn đề trên và sửa lại thiết kế và source code tương ứng với đề xuất.

Sau khi hoàn thành xong, nộp lại kết quả theo Assignment:
Dưới đây là một báo cáo mẫu:

1. Coupling

1.1. Content coupling

Related modules	Description	Improvement

1.2. ...

2. Cohesion

2.1. Coincidental cohesion

Related modules	Description	Improvement

2.2. ...

4. BÀI TẬP NHÓM

Trong phần này bạn cần:

- Review lại các mức coupling và cohesion và xác định các mức độ này trong modules của thiết kế cho bài tập lớn môn học của nhóm. Nếu thiết kế chưa tốt, hãy đề xuất giải pháp cải thiện.

- Viết báo cáo cho các vấn đề trên (theo mẫu ở trên) và sửa lại thiết kế và source code tương ứng với đề xuất. Sau khi hoàn thành xong, nộp lại kết quả vào thư mục "GoodDesign/DesignConcepts."

- Đây là một trong các nội dung quan trọng trong đánh giá kết quả bài tập lớn.

HẾT