Báo cáo học tập - Thứ 5 - 15 / 09 / 2022

Kiến trúc Phần mềm

Tái cấu trúc mã nguồn cho chương trình Java

---

1 Tổng quan về Tái cấu trúc mã nguồn

Trước khi tham gia buổi học , chúng em thực hiện viết chương trình Java "Bài tập chuẩn bị cho BTVN2" chủ yếu bằng kinh nghiệm và thói quen lập trình trước đây .

Những dòng lệnh không thực sự được liên kết , tối ưu và thông minh theo yêu cầu một chương trình phần mềm cần đảm bảo .

Và vì thế chúng em nhận được sự hướng dẫn của giảng viên - thầy Võ Đình Hiếu về Tái cấu trúc mã nguồn .

Sau khi tham khảo và triển khai , chương trình của chúng em trở lên tối ưu và thông minh hơn bao giờ hết .

Nhóm chúng em cố gắng vẫn dụng toàn bộ quy tắc của Tái cấu trúc mã nguồn vào chương trình và cụ thể từng yếu tố hoạt động như thế nào sẽ được trình bày trong bài báo cáo học tập dưới đây

Chúng em có chuẩn bị đầy đủ mã nguồn trước và sau khi tái cấu trúc để thầy có thể dễ dàng so sánh

Vì một đoạn mã nguồn có thể sử dụng nhiều quy tắc tái cấu trúc nên đôi khi sẽ có sự lặp lại nhất định về mặt hình ảnh , tuy nhiên chúng em sẽ cố gắng tối ưu phần trình bày sao cho dễ hiểu và hợp lý nhất .

Nhóm chúng em gồm 3 thành viên :

1. **Nguyễn Tiến Hoàng - 21020123**
2. **Nguyễn Đức Minh - 20020060**
3. **Sùng Mí Và - 19020162**

Rất mong bản báo cáo của nhóm chúng em có thể gây ấn tượng với thầy cùng các bạn trong lớp học phần .

2 Composing Methods

2.1 Extract Method && Inline Method

2.1.1 Mô tả

Theo quan điểm của chúng em , việc viết ra các dòng lệnh thực chất là việc chuyển từ ngôn ngữ đọc hiểu của quá trình phân tích và thiết kế cấu trúc sang ngôn ngữ lập trình

Mỗi hàm cần thực hiện một công việc xác định , chứa những công việc nhỏ hơn . Cho đến khi mỗi công việc đơn giản đến mức có thể thực hiện bằng một vài dòng lệnh .Thể hiện Logic theo chiều sâu giúp các nhà phát triển dễ dàng truy vấn , đọc hiểu , đồng thời vẫn giữ được tính độc lập nhất định .

Mỗi hàm được đặt tên bằng một động từ chỉ mục đích của phương thức và có thể tái sử dụng khi cần thiết

Ngược lại , nếu Phương thức chỉ dùng để ủy thác , bao bọc cho 1 phương thức khác mà không tính toán , xử lý thêm thì nên được loại bỏ khỏi chương trình

2.1.2 Triển khai

Trong hàm main() , chương trình chỉ thực hiện một công việc lớn lao và duy nhất là "Tính toán".
Việc tính toán như thế nào , có thể được xem xét bằng cách truy vấn tới các hàm con được triển khai trong lớp Customer , Customer xử dụng kết quả từ lớp Play để tính toán totalAmount và volumnCredits một cách riêng biệt

```
customer.calculate();
```

```
public void calculate() {
    for (Play child : this.getPlayList()) {

        calTotalAmount(child);

        calVolumeCredits(child);

    }
```

```
private void calTotalAmount(Play child) {

    child.calPlayAmount();
```

2.2 Extract Variable && Inline Temp && Split Temporary Variable && Replace Term with Query

2.2.1 Mô tả

Sử dụng danh từ để đặt tên biến để thể hiện một kiểu giá trị , một đối tượng riêng biệt và tạo biến trung gian để lưu trữ cho các tham số khó hiểu

Các biến và phương thức cùng nhau thể hiện "Một cái gì đó đang làm gì đó" một cách rõ ràng

Tuy nhiên chương trình cần phải tạo nhiều biến hơn , nhưng điều này là đánh đổi được vì các đoạn mã sẽ dễ đọc hơn , ít phát sinh lỗi và việc không phải gọi lại hàm thậm chí còn tối ưu hơn .

Ngược lại , nếu một biến trung gian chỉ để lưu kết quả của một hành động đơn giản nào đó thì ta có thể bỏ qua nếu không ảnh hưởng đến hiệu suất.

Đồng thời , Nếu một biến được dùng tạm để tính toán trả về cho phương thức , nên chuyển biến đó thành một phương thức truy vấn , để có thể lấy ra sử dụng bất cứ lúc nào , tái sử dụng khi cần đến biến đó một lần nữa ở một hàm khác

Việc phải tối bộ nhớ cho truy vấn là không đáng kể cho lợi ích mà nó mang lại

2.2.2 Triển khai

Thuộc tính totalAmount , volumeCredits , result , ... được thêm vào danh sách thuộc tính của lớp Customer để sử dụng khi cần , thay vì phải khởi tạo và gọi nhiều lần trong các hàm tính toán.

```java
public class Customer {
    private ArrayList<Play> playList = new ArrayList<Play>();
    private String name;
    private int totalAmount;
    private int volumeCredits;
    private String result;
```

Before

```java
int totalAmount = 0;
int volumeCredits = 0;
String result =  "Statement for " + customer.getName() + "\n";
```

2.3 Remove Assignment to Parameter

2.3.1 Mô tả

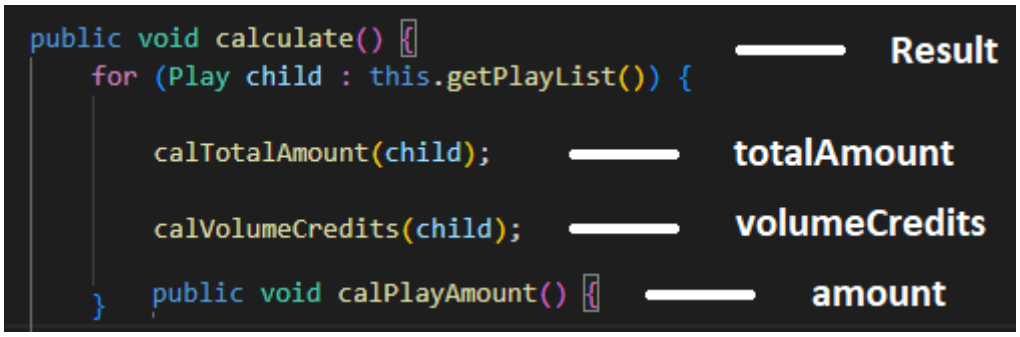
Tạo 1 biến nội bộ trong hàm khi truyền tham số vào hàm nếu có cần thay đổi giá trị của tham số trong quá trình tính toán

Để đảm bảo tham số toàn cục truyền vào không bị thay đổi giá trị khi gọi hàm .

Một phương thức chỉ nên chịu trách nhiệm cho 1 thứ thay vì phải chịu những hệ quả không đáng có

2.3.2 Triển khai

Tất cả các hàm calculate() chỉ chịu trách nhiệm cho việc tính toán , cập nhật giá trị các tham số cần tính toán , mà không làm thay đổi giá trị của các tham số hỗ trợ bên lề truyền vào .



2.4 Replace Method with Method Object

2.4.1 Mô tả

Những methods cần sử dụng nhiều biến cục bộ có liên hệ với nhau sẽ khó khăn trong việc mong muốn chỉ giải quyết 1 vấn đề cụ thể

Khi đó nên chuyển phương thức thành một lớp mới , các biến cục bộ trở thành thuộc tính của class và sử dụng chúng trong triển khai phương thức của đối tượng một cách dễ dàng .

2.4.2 Triển khai

Trước khi tái cấu trúc , hàm main() khai báo totalAmount = 0 , VolumnCredits = 0 , thisAmount = 0 , ...

Các biến được khai báo để xử lý Customer và các Play , tuy nhiên không có tính hệ thống , làm hàm sử lý logic không được chuyên biệt .

Vì thế chúng em đã chuyển totalAmount , VolumnCredits thành thuộc tính của customer và thisAmount thành thuộc tính của Play để xử chuyên biệt cho từng đối tượng , dễ dàng viết các phương thức calculate và mở rộng thêm nhiều Customer và Play khi cần thiết .

2.5 Substitue Algorithms

2.5.1 Mô tả

Thay thế thuật toán cũ bằng một thuật toán mới hiệu quả , thông minh , logic hơn .

2.5.2 Triển khai

Trước khi tái cấu trúc , chương trình cần sinh đối tượng Play liên tục , gắn vào các biến p1,p2,p3,... và Customer phải thêm từng Play vào trong danh sách , tuy nhiên việc đó rất thiếu tối ưu , khó mở rộng , không phải một thuật toán hiệu quả , thông minh và logic .

Chúng em đã tạo 1 mảng Play , sinh liên tục các đối tượng , và dành 1 vòng for duy nhất cho Customer để tự động thêm khi có đối tượng mới

```
Customer customer = new Customer(name: "BigCo");

Play[] p = {
        new Play(name: "Hamlet", type: "tragedy", audiance: 55),
        new Play(name: "As You Like It", type: "comedy", audiance: 35),
        new Play(name: "Othello", type: "tragedy", audiance: 40)
};

for (Play child : p) {
    customer.addPlay(child);
}
```

Before

```
Customer customer = new Customer(name: "BigCo");

Play p1 = new Play(name: "Hamlet",type: "tragedy",audiance: 55);
Play p2 = new Play(name: "As You Like It",type: "comedy",audiance: 35);
Play p3 = new Play(name: "Othello",type: "tragedy",audiance: 40);
customer.addPlay(p1);
customer.addPlay(p2);
customer.addPlay(p3);
```

3 Moving Features between Objects

3.1 Move Method

3.1.1 Mô tả

Những phương thức cụ thể thường được sử dụng chủ yếu ở một đối tượng xác định , hãy khai báo phương thức đó cho đối tượng chính , và nếu các đối tượng khác cần sẽ tham chiếu tới và sử dụng .

Việc này sẽ làm quan hệ hoạt động của các thành phần trở lên logic hơn

3.1.2 Triển khai

Trước khi tái cấu trúc , phương thức formatUSD() và rounding() được sử dụng ở cả lớp Customer và Play để hỗ trợ cho các hoạt động tính toán.

Tuy nhiên vì được sử dụng chủ yếu ở Customer , sau khi đã có kết quả từ Play trả về , nên để để hẳn ở lớp Customer , lớp Play hầu như không cần sử dụng nữa , nếu cần sẽ tham chiếu tới

```
// ----- Support function -----------------------

private double rounding(int i) {
    return (double) Math.round(i * 10) / 10;
}

private String formatUSD(int i) {
    Locale usa = new Locale(language: "en", country: "US");
    Currency dollars = Currency.getInstance(usa);
    NumberFormat dollarFormat = NumberFormat.getCurrencyInstance(usa);
    return dollarFormat.format(i);
}
```

3.2 Move Field

3.2.1 Mô tả

Tương tự , các trường thuộc tính cũng nên ở lớp được sử dụng nhiều và trở thành thuộc tính đặc trưng của lớp đó , những lớp khác cần sử dụng thì điều hướng tới thông qua các phương thức getter / setter trên giao diện.

3.2.2 Triển khai

Đối tượng Customer sử dụng thuộc tính của các Play để tính toán thông qua getter thay vì chứa cả các thuộc tính đó , chúng chỉ nên là đặc trưng của đối tượng Play

3.3 Extract class && Inline class

3.3.1 Mô tả

Một lớp chỉ nên thực hiện công việc của lớp đó , thay vì đa nhiệm nhiều công việc

Khi lớp lớn có chứa thành phần thể hiện bằng nhiều thành phần con nên tách riêng thành phần đó ra 1 lớp mới , có liên hệ với lớp lớn

Và ngược lại , những thành phần con không quá phức tạp thì tốt hơn hết vẫn nên là thuộc tính , phương thức của lớp lớn , để tránh tạo ra nhiều loại đối tượng phức tạp không cần thiết

3.3.2 Triển khai

Lớp Customer chứa nhiều Play , totalValue và volumeCredits .

Play là một thành phần lớn , có nhiều thuộc tính và nhiều làm xử lý , tốt nhất nên tách ra thành 1 lớp riêng , liên hệ với lớp Customer thông qua quan hệ has-a

Còn totalValue và volumeCredits chỉ là 1 thuộc tính giá trị nhỏ , không cần tách thành đối tượng .

```
public class Customer {
    private ArrayList<Play> playList = new ArrayList<Play>();
    private String name;
    private int totalAmount;
    private int volumeCredits;
    private String result;
```

3.4 Hide delegate && Remove Middle man

3.4.1 Mô tả

Nếu có một lớp cha sử dụng kết quả của lớp con và lớp con cần kế thừa các thuộc tính , phương thức của lớp cha , thì nên có một mối quan hệ cha – con .

Ngược lại , nếu chỉ đơn giản là có liên hệ với nhau hoặc sử dụng kết quả một chiều , không quá liên kết , nên để 2 lớp riêng biệt , có một chút quan hệ điều hướng tới nhau

3.4.2 Triển khai

Trước khi tái cấu trúc , chúng em đã suy nghĩ đến việc để lớp Play là lớp con của Customer vì theo lẽ hiểu thông thường , Customer chứa các Play .

Tuy nhiên vì chỉ có lớp Customer sử dụng kết quả tính toán từ Play do đó 2 lớp này nên là riêng biệt và có quan hệ liên quan ở mức has – a

3.5 Introduction Foreign Method && Introduction Local Extension

3.5.1 Mô tả

Nếu chương trình có một đoạn mã phức tạp , lặp lại nhiều lần trong 1 lớp mà không thể đơn giản hóa

Nên triển khai một phương thức hỗ trợ , phục vụ cho công việc bạn đang cần và truyền đối tượng vào bên trong

Nếu thực sự cần thiết có thể tạo một lớp mới , chứa các phương thức bổ trợ và kế thừa phương thức cha để sử dụng

3.5.2 Triển khai

Chúng em tạo riêng phương thức hỗ trợ formatUSD() and rounding() để tránh việc phức tạp code trong hàm tính toán và phải sử dụng lại nhiều lần Những phương thức thành không cần tách riêng ra thành các đối tượng vì chúng không quá phức tạp và không liên hệ mật thiết đến đối tượng Customer

```
// ----- Support function ----------------------

private double rounding(int i) {
    return (double) Math.round(i * 10) / 10;
}

private String formatUSD(int i) {
    Locale usa = new Locale(language: "en", country: "US");
    Currency dollars = Currency.getInstance(usa);
    NumberFormat dollarFormat = NumberFormat.getCurrencyInstance(usa);
    return dollarFormat.format(i);
}
```

4 Organizing Data

4.1 Change Value to Reference && Change Reference to Value && Replace Data Value with Object

4.1.1 Mô tả

Dữ liệu và hành vi của một lớp nên chỉ liên quan đến 1 lớp duy nhất .

Nếu đối tượng có thể bị thay đổi trong chương trình , nên thể hiện chúng dưới dạng Reference để có thể dễ dàng cập nhật trong từng đối tượng

Còn nếu không bị thay đổi , nên thể hiện dưới dạng Value , để có thể cập nhật 1 lần dùng cho tất cả , không phải quản lý ở tất cả mọi nơi

4.1.2 Triển khai

Trước khi tái cấu trúc , chúng em có suy nghĩ không tạo lớp Customer mà chỉ có lớp Play , mang thuộc tính customerName vì đề bài chỉ có 1 khách hàng tên "Bigco"

Tuy nhiên trong tương lai có thể có thêm nhiều khách hàng và lớp Customer là đối tượng chứa các đối tượng Play vì thế nên cần tách ra 1 lớp riêng , có thuộc tính name riêng , liên kết với lớp Play trên quan hệ has-a

4.2 Duplicate Observed Data

Ta cần tách riêng dữ liệu và GUI để có thể tách bạch và xử lý chuyên biệt rõ việc phân tích , tính toán và việc triển khai đồ họa

4.3 Encapsulation field && Self- encapsulation field

4.3.1 Mô tả

Mỗi đối tượng cần có tính đóng gói nhất định , đó là một trong 4 nguyên tắc của lập trình hướng đối tượng .

Chúng ta cần triển khai việc truy cập các thuộc tính của đối tượng thông qua giao diện bằng phương thức getter / setter để các đối tượng có thể mang tính độc lập nhất định

Đồng thời bản thân cái phương thức bên trong đối tượng cũng nên sử dụng getter / setter khi cần truy cập đến thuộc tính của chính nó

Thói quen này giúp cho chương trình trở lên linh hoạt hơn khi chúng ta muốn Ghi đè , Thừa kế , Khởi tạo lười biếng , xác thực , ...

4.3.2 Triển khai

Ngôn ngữ Java hỗ trợ triển khai đối tượng mang tính đóng gói nhất định .

Trước khi tái cấu trúc , các phương thức của lớp Customer và lớp Play truy cập trực tiếp vào thuộc tính của chính nó .

Mặc dù không cần thiết lắm vì bài toán nhỏ , những chúng em em đã đổi hết qua truy cập thông qua phương thức getter / setter , để chương trình dễ dàng mở rộng hơn .

```
result += "Amount owed is " + formatUSD(this.getTotalAmount() / 100) + "\n";
result += "You earned " + this.getVolumeCredits() + " credits \n";
```

Before

```
result += "Amount owed is " + formatUSD(totalAmount/100) + "\n";
result += "You earned " +  volumeCredits + " credits \n";
```

4.4 Encapsulation Collector

4.4.1 Mô tả

Với các thuộc tính được thể hiện dưới dạng Collector hoặc Danh sách .

Việc getter / setter và triển khai các hàm liên quan (xóa hoặc thêm) mà vẫn đảm bảo tính Đóng gói sẽ phức tạp hơn một chút .

Tuy nhiên lại đem đến nhiều hiệu quả hơn cho chương trình như kiểm tra null , kiểm tra define , truyền Set vào danh sách , ...

4.4.2 Triển khai

Tái cấu trúc Getter & Setter thuộc tính kiểu Danh sách trong lớp customer .

```
public ArrayList<Play> getPlayList() {
    return playList;
}

public void setPlayList(Collection<Play> p) {

    this.playList.clear();
    if (p != null) {
        this.playList.addAll(p);
    }
}
```

Before

```
public void setPlayList(ArrayList<Play> playList) {
    this.playList = playList;
}
```

4.5 Replace Array with Object

Khi sử dụng các ngôn ngữ không hướng đối tượng , Chúng ta có thói quen thể hiện đối tượng có nhiều thuộc tính thông qua các cấu trúc dữ liệu Prev hoặc Dictionary hay thậm chí là Array .

Việc làm này là chưa tối ưu , khó truy cập và xử lý , nên tạo ra các đối tượng , mang những thuộc tính và phương thức cần có .

Đó có lẽ là lí do vì sao chúng em được yêu cầu chuyển ngôn ngữ lập trình khác (JavaScript) qua ngôn ngữ Java .

4.6 "Change Unidirectional Association to Bidirectional" && "Change Bidirectional Association to Unidirectional"

4.6.1 Mô tả

Trong những trường hợp 2 lớp có liên hệ mật thiết với nhau , xử dụng kết quả của nhau , ta nên có mối liên hệ 2 chiều giữa hai lớp này thay vì liên hệ 1 chiều rồi cố gắng tính toán ngược lại

Tuy nhiên , việc liên kết 2 chiều cho nhiều lợi ích với bài toán lớn những cũng cần quản lý nhiều hơn .

4.6.2 Triển khai

Trong bài tập , em chỉ sử dụng kết quả của lớp Play cho lớp Customer mà không cần chiều ngược lại nên chỉ cần liên kết 1 chiều là đã đáp ứng được yêu cầu bài toán .

```
private void calTotalAmount(Play child) {

    child.calPlayAmount();

    totalAmount += child.getAmount();
```

4.7 Replace Magic Number with Symbolic Constant

Những hằng số mang ý nghĩa nhất định nên được khai báo như một biến final và một tên đại diện cho hằng số đó .

Người đọc có thể dễ dàng hiểu được hằng số đó là gì và tái sử dụng khi cần thiết .

5 Simplifying Conditional Expressions

5.1 Consolidate Duplicate Conditional Fragments

Những hoạt động sẽ xảy ra trong cả hai trường hợp của cấu trúc điều kiện thì nên được triển khai như các câu lệnh bình thường

```
case "comedy":
    amount = 30000;
    if (this.getAudiance() > 20) {
        amount += 1000 + 500 * (this.getAudiance() - 20);
    }
    amount += 300 * this.getAudiance();
```

5.2 Replace Conditional with Polymorphism

5.2.1 Mô tả

Cũng giống như "Replace Type Code with subClass" , nguyên tắc này xác định các trạng thái của đối tượng như một Đa hình

Việc thể hiện dưới dạng các Đa hình sẽ giúp chương trình dễ dàng sử lý chuyên biệt từng loại đối tượng

5.2.2 Triển khai

Tuy nhiên việc sử lý logic trong từng trạng thái của đối tượng Play là không nhiều nên không cần thiết phải đa hình các trạng thái nếu không phát triển theo hướng đó , thế nên em vẫn dùng switch case các trạng thái để tính toán .

```
public void calPlayAmount() {
    switch (type) {
        case "tragedy":
            amount = 40000;
            if (this.getAudiance() > 30) {
                amount += 1000 * (this.getAudiance() - 30);
            }
            break;
        case "comedy":
            amount = 30000;
            if (this.getAudiance() > 20) {
                amount += 1000 + 500 * (this.getAudiance() - 20);
            }
            amount += 300 * this.getAudiance();
            break;
        default:
            // throw Exception
    }
}
```

6 Simplifying Method Calls

6.1 Introduce Parameter & Preserve Whole Object

6.1.1 Mô tả

Phương thức bao gồm một nhóm tham số lặp lại , nên được tóm gọn truyền vào một loại tham số để dễ quản lý

6.1.2 Triển khai

Thay vì truyền vào type và audience của Play child thì truyền cả child vào để khi cần các thuộc tính khác còn có thể lập tức xử lý

```
calVolumeCredits(child);
```

Before

```
calVolumeCredits(child.getAudiance() , child.getType());
```

6.2 Tất cả quy tắc còn lại trong phần này

Tất cả các quy tắc còn lại trong Simplifying Method Calls đều là các nguyên tắc đại cương mà hiển nhiên một nhà phát triển phần mềm vào đều cần phải hiểu và đảm bảo .

7 Tổng kết

Vì chương trình bài tập về nhà ở quy mô còn nhỏ nên chưa thực sự cần sử dụng toàn bộ các quy tắc trong Tái cấu trúc mã nguồn .

Nhưng khi được tái cấu trúc phần nào đã trở lên tối ưu và thông minh hơn .

Qua lần luyện tập này giúp chúng em có tư duy hơn trong việc viết ra các chương trình cũng như phân tích và thiết kế cấu trúc của các hệ thống phần mềm.

Các quy tắc dần trở thành kinh nghiệm của một lập trình viên , mỗi dòng lệnh viết ra đều có sự thông minh và tinh tế nhất định .

8 Tài liệu

[1] Bài giảng lớp Kiến trúc Phầm mềm - Thầy Võ Đình Hiếu https://courses.uet.vnu.edu.vn/mod/assign/view.php?id=112068.

[2] Refactoring Guru https://refactoring.guru/refactoring/techniques.