TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - CƠ - TIN HỌC

BÁO CÁO CUỐI KÌ

CÁC THÀNH PHẦN PHẦN MỀM

Giảng viên: Quản Thái Hà

Đề tài:

DESIGN PATTERN(Nhóm 1)

Thành viên

Lê Thị Thùy Dung - 20001895
Bùi Khánh Duy - 20001898

Mục lục

1 Tổng quan về Design Pattern

1.1 Khái niệm

Design Pattern là một kỹ thuật quan trọng trong lập trình hướng đối tượng(OOP). Design pattern là các giải pháp tổng thể đã được tối ưu hóa, được tái sử dụng cho các vấn đề thường gặp trong thiết kế phần mềm. Nó cung cấp cho chúng ta các "mẫu thiết kế", giải pháp để giải quyết các vấn đề chung, thường gặp trong lập trình.

Với sự đúc kết và công nhận từ nhiều nhà nghiên cứu, Design Pattern là mẫu chuẩn tối ưu nhất, có thể áp dụng để giải quyết không chỉ một vấn đề mà nhiều vấn đề có tính chất tương tự nhau, lặp đi lặp lại nhiều lần trong lập trình OOP.

Design Pattern không dành riêng cho một ngôn ngữ lập trình cụ thể nào. Nó có thể được áp dụng trong hầu hết các ngôn ngữ lập trình OOP như: PHP, Java, Python và nhiều ngôn ngữ khác.

1.2 Tại sao nên sử dụng Design Pattern?

Đầu tiên, chúng ta phải nói đến tác dụng tăng tốc độ phát triển phần mềm. Design pattern cung cấp những giải pháp ở dạng tổng quát, đã được tối ưu hóa, giúp tăng tốc độ phát triển phần mềm bằng cách đưa ra các mô hình test, mô hình phát triển đã qua kiểm nghiệm. Việc này giúp tiết kiệm rất nhiều thời gian so với việc tự viết ra những giải pháp chưa thực sụ tối ưu.

Design Pattern giúp hạn chế lỗi tiềm ẩn. Việc sử dụng giải pháp đã được kiểm chứng chắc chắn sẽ ít rủi ro hơn là tự mình thử nghiệm giải pháp mới.

Không chỉ thế, Design Pattern sẽ hỗ trợ tái sử dụng mã lệnh. Các mẫu thiết kế có thể được sử dụng nhiều lần mà không xảy ra lỗi. Chúng ta cũng dễ dàng mở rộng, nâng cấp và bảo trì để đáp ứng được các yêu cầu thay đổi liên tục của dự án.

Sử dụng Design Pattern giúp cho code dễ đọc, dễ hiểu hơn. Vì nó đã theo khuôn mẫu nên nó giúp cho các lập trình viên tìm được tiếng nói chung, có thể hiểu code của người khác một cách nhanh chóng.

1.3 Để học Design Pattern cần có gì?

Muốn học Design Pattern, bạn cần phải có kiến thức vững chắc về lập trình OOP, cụ thể là:

- 4 đặc tính: Trừu tượng, Đóng gói, Đa hình, Kế thừa.
- 3 khái niệm: abstract class, interface và static.
- Tư duy hoàn toàn theo OOP.

1.4 Phân loại

Hệ thống các mẫu Design pattern hiện có gồm 23 mẫu được định nghĩa trong cuốn “Design patterns Elements of Reusable Object Oriented Software” và được chia thành 3 nhóm:

- Creational Pattern (Nhóm khởi tạo)

  Những design pattern loại này cung cấp một giải pháp để tạo ra và che giấu được logic của việc tạo ra object, thay vì sử dụng method new. Điều này giúp cho chương trình trở nên mềm dẻo hơn trong việc quyết định object nào cần được tạo ra trong những tình huống được đưa ra.

  Gồm 5 design pattern:

  1. Abstract Factory
  2. Builder
  3. Factory
  4. Prototype
  5. Singleton

- Structural Pattern (Nhóm cấu trúc)

  Những Design pattern loại này liên quan tới class và các thành phần của object. Nó dùng để thiết lập, định nghĩa quan hệ giữa các đối tượng.

  Gồm 7 design pattern:

  1. Adapter
  2. Bridge
  3. Composite
  4. Decorator
  5. Facade
  6. Flyweight
  7. Proxy

- Behavioral Pattern (Nhóm tương tác/ hành vi)

  Nhóm này dùng trong thực hiện các hành vi của đối tượng, sự giao tiếp giữa các object với nhau.

  Gồm 11 design pattern:

  1. Chain of responsibility
  2. Command
  3. Interpreter
  4. Iterator
  5. Mediator
  6. Memento
  7. Observer
  8. State
  9. Strategy
  10. Template method
  11. Visitor

2 Một số Design Patterns

2.1 Singleton

2.1.1 Định nghĩa, mục đích sử dụng và mô hình cấu trúc

"The Singleton Pattern ensures a class has only one instance, and provides a global point of access to it."

(Head First Design Pattern)

Singleton là 1 trong 5 design pattern của nhóm Creational Design Pattern. Singleton pattern là một design pattern mà nó đảm bảo rằng một class chỉ có duy nhất một instance và nó cung cấp một cách toan cục để truy cập tới instance đó.

Mục đích sử dụng Singleton là đảm bảo rằng 1 class chỉ có 1 instance duy nhất và class này luôn sẵn sàng để sử dụng ở bất kỳ thời điểm hoặc vị trí nào trong phần mềm ứng dụng. Singleton cũng giúp cho quản lý truy cập tốt hơn vì chỉ có một instance duy nhất. Có thể quản lý số lượng instance của một lớp trong giới hạn chỉ định.

Một Singleton Pattern bao gồm các thành phần cơ bản sau:

- Hàm constructor phải có access modifier là private để hạn chế truy cập từ class bên ngoài.
- Khai báo private static variable để lưu trữ instance của class Singleton.
- Định nghĩa một thuộc tính là private static và đó là instance duy nhất của class này.
- Định nghĩa public static getInstance() dùng để khởi tạo đối tượng (hàm accessor). Phương thức getInstance() sẽ kiểm tra xem lớp đó có instance nào chưa, nếu chưa (== null) thì sẽ tạo một đối tượng mới, nếu có rồi thì sẽ trả về đối tượng đó.
- Thực hiện lazy initialization trong hàm accessor: chỉ khi gọi mới khởi tạo instance.
- Client chỉ có thể gọi hàm accessor khi muốn có instance của class

2.1.2 Code minh họa

1. Singleton.java: Singleton

```
public final class Singleton {

 // Biến uniqueInstance là biến static, là instance duy nhất của class Singleton.
 private static Singleton uniqueInstance;
 public String value;

 private Singleton(String value) {
   // Contructor Singleton() được khai báo là private
   // để hạn chế khả năng truy cập của các class bên ngoài.
   // Chỉ Singleton mới có thể khởi tạo lớp này.
   // Đoạn code dưới đây sẽ bắt thread phải chờ 1s trước khi gán giá trị.
   try {
     Thread.sleep(1000);
   } catch (InterruptedException ex) {
     ex.printStackTrace();
   }
   this.value = value;
 }

 public static Singleton getInstance(String value) {
   // Nếu instance đang là null thì khởi tạo,
   // ngược lại sẽ trả về nó.
   if (uniqueInstance == null) {
     uniqueInstance = new Singleton(value);
   }
   return uniqueInstance;
 }
}
```

2. DemoSingleThread.java: Client code

```
public class DemoSingleThread {

//  Class này đểkiểm tra xem Singleton có thật sự hoạt động hay không,
//  có thật sự chỉ tồn tại 1 instance hay không?
 public static void main(String[] args) {
   System.out.println(
       "If you see the same value, then singleton was reused (yay!)"
```

Ưu điểm	Nhược điểm
ff	fff

```
            + "\n"
            + "If you see different values, then 2 singletons were created
                (booo!!)"
            + "\n\n"
            + "RESULT:"
            + "\n");
    Singleton singleton = Singleton.getInstance("FOO");
    Singleton anotherSingleton = Singleton.getInstance("BAR");
    System.out.println(singleton.value);
    System.out.println(anotherSingleton.value);
  }
}
```

3. OutputDemoSingleThread.txt: Execution result

```
If you see the same value, then singleton was reused (yay!)
If you see different values, then 2 singletons were created (booo!!)
RESULT:

FOO
FOO
```