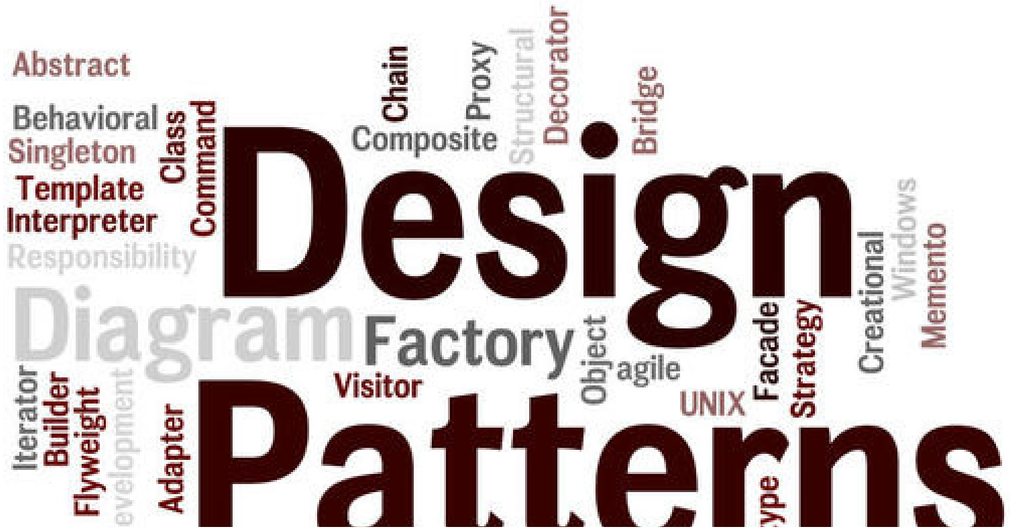


Template Method Pattern

Nhóm 1
Phan Hoàng Minh Quân – 20520713
Trần Gia Bảo – 21521863
Trần Đông Đông - 21521957

Nội dung

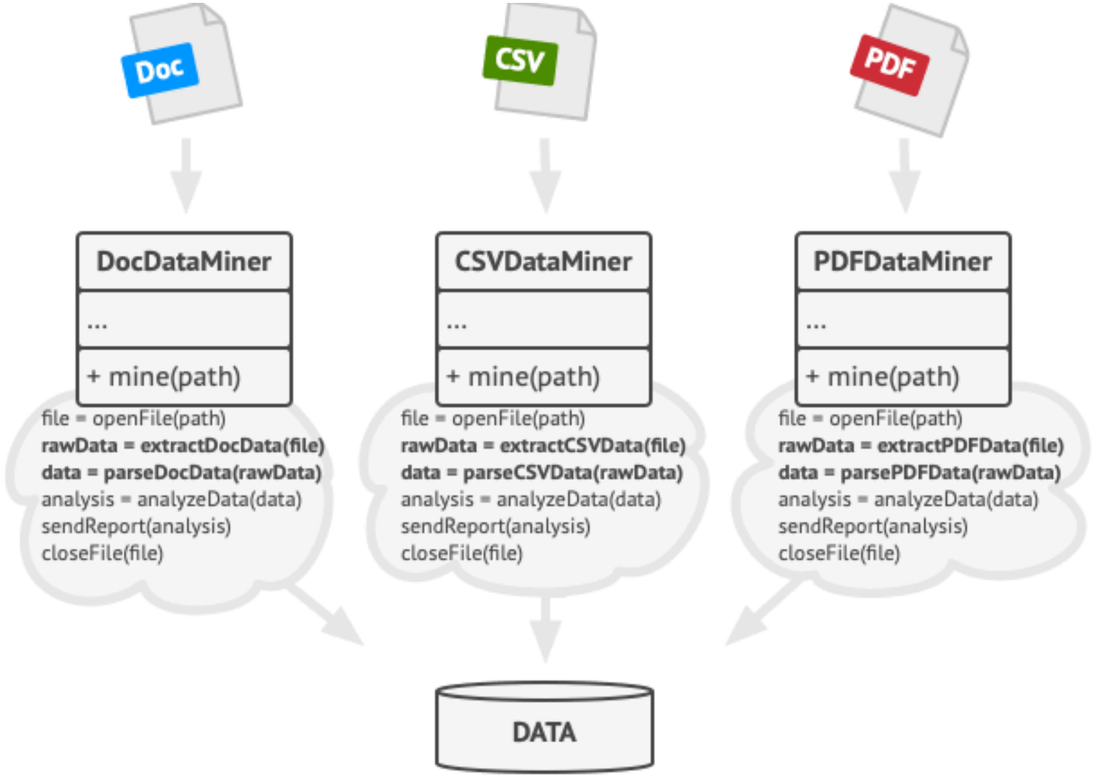
1. Tổng quan
   - Tên
   - Mô tả ngắn về mẫu
   - Phân loại
2. Motivation
3. SOLID principle in Builder
4. Cấu trúc mẫu và mô tả + ví dụ minh họa
5. Các bước hiện thực mẫu + code minh họa cho ví dụ trên
6. Ưu điểm
7. Nhược điểm
8. Áp dụng thực tiễn
9. Các mẫu liên quan
10. Link source code ví dụ và nguồn tài liệu

1. Tổng quan

- Tên mẫu: Proxy
- Phân Loại: Behavioral Pattern
- Mô tả:
  - Template Method Pattern là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern). Pattern này nói rằng “Định nghĩa một bộ khung của một thuật toán trong một chức năng, chuyển giao việc thực hiện nó cho các lớp con. Mẫu Template Method cho phép lớp con định nghĩa lại cách thực hiện của một thuật toán, mà không phải thay đổi cấu trúc thuật toán“.

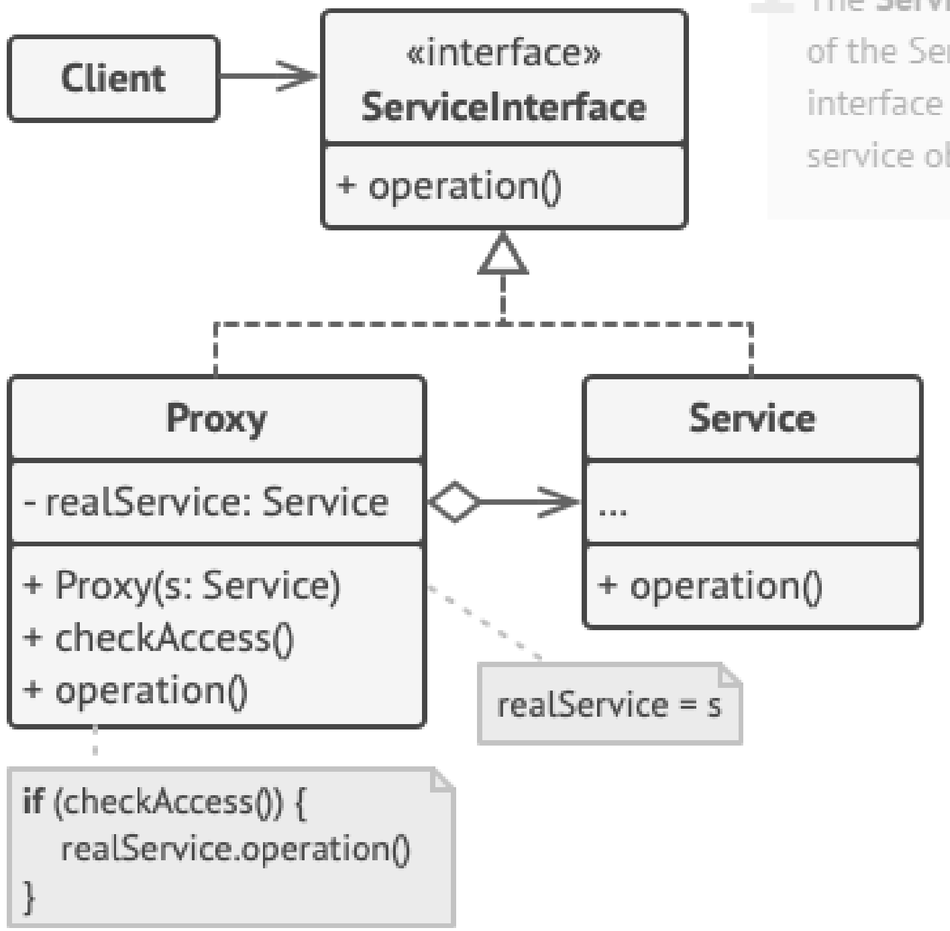
2. Motivation

- Trong quá trình phát triển ứng dụng, chúng ta có các component khác nhau có sự tương đồng đáng kể, nhưng chúng không sử dụng interface/ abstract class chung, dẫn đến code duplicate ở nhiều nơi. Nếu muốn thay đổi chung cho tất cả component, chúng ta phải đi sửa ở từng nơi trong component, làm tốn nhiều chi phí không cần thiết. Một trong những cách để giải quyết vấn đề này là sử dụng Template Method Pattern.



4. Cấu trúc

4 The **Client** should work with both services and proxies via the same interface. This way you can pass a proxy into any code that expects a service object.



1 The **Service Interface** declares the interface of the Service. The proxy must follow this interface to be able to disguise itself as a service object.

3 The **Proxy** class has a reference field that points to a service object. After the proxy finishes its processing (e.g., lazy initialization, logging, access control, caching, etc.), it passes the request to the service object.

Usually, proxies manage the full lifecycle of their service objects.

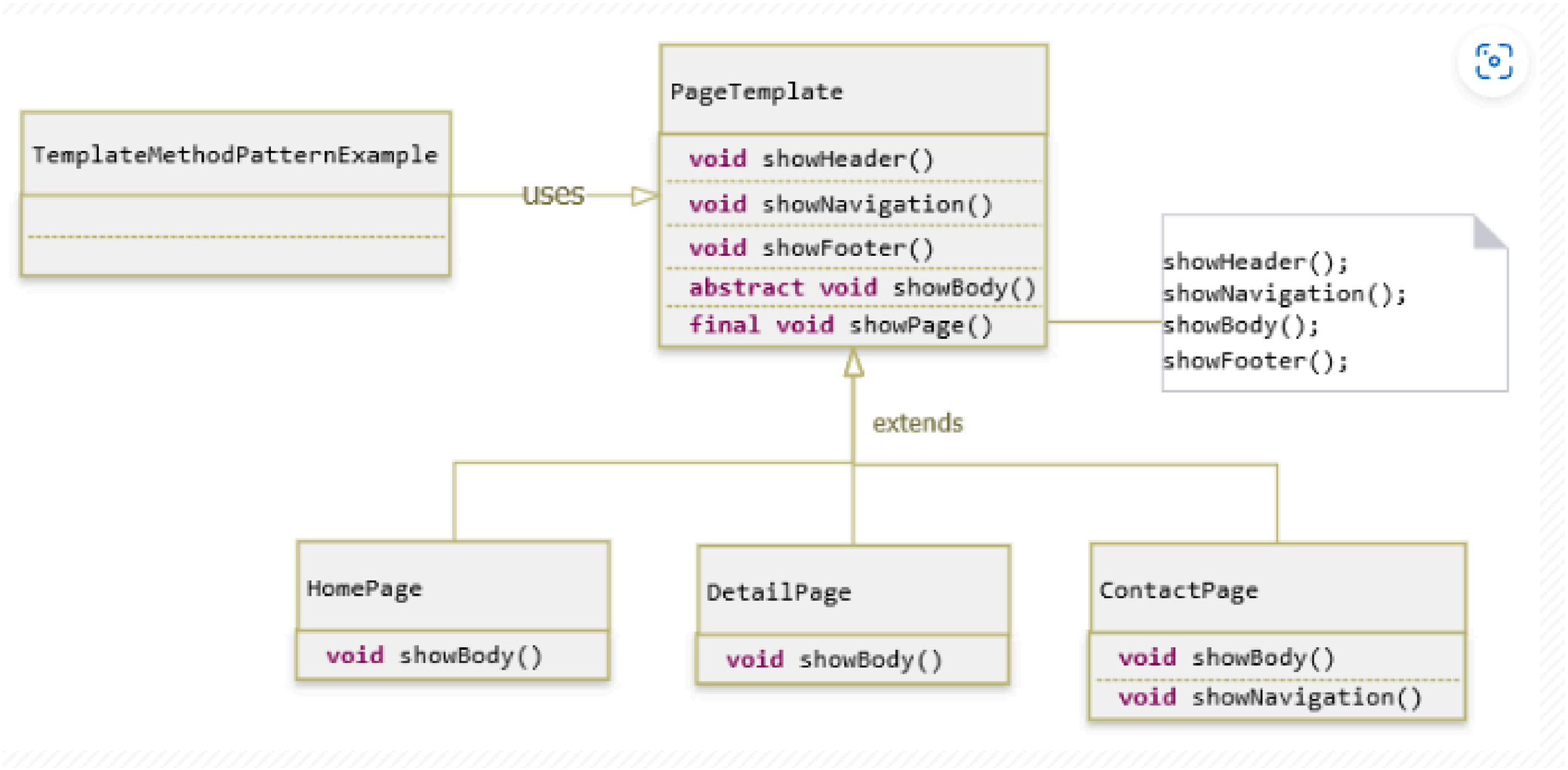
2 The **Service** is a class that provides some useful business logic.

4. Cấu trúc

Một Proxy Pattern bao gồm các thành phần sau:

- Lớp trừu tượng (**Abstract Class**) khai báo các phương thức đóng vai trò như các bước của một thuật toán, cũng như phương thức mẫu (template method) thực sự, phương thức này gọi các bước theo một thứ tự cụ thể. Các bước có thể được khai báo là trừu tượng hoặc có sẵn một số triển khai mặc định.
- Các lớp cụ thể (**Concrete Classes**) có thể ghi đè tất cả các bước, nhưng không được ghi đè phương thức mẫu (template method) chính.

5. Code Ví dụ (Cấu trúc)



5. Code Ví dụ

```
public abstract class PageTemplate {

    protected void showHeader() {
        System.out.println(" <header />");
    }

    protected void showNavigation() {
        System.out.println(" <nav />");
    }

    protected void showFooter() {
        System.out.println(" <footer />");
    }

    protected abstract void showBody();

    public final void showPage() {
        showHeader();
        showNavigation();
        showBody();
        showFooter();
    }
}
```

5. Code Ví dụ

```
public class HomePage extends PageTemplate {

    @Override
    protected void showBody() {
        System.out.println("Content of home page page");
    }
}
```

5. Code Ví dụ

```
public class DetailPage extends PageTemplate {

    @Override
    protected void showBody() {
        System.out.println("Content of detail");
    }
}
```

5. Code Ví dụ

```
public class ContactPage extends PageTemplate {

    @Override
    protected void showNavigation() {
        // Just do nothing
        // Because we don't want to show navigation bar on contact page
    }

    @Override
    protected void showBody() {
        System.out.println("Content of contact page");
    }
}
```

Extract Surround // ⋮

5. Code ví dụ (Kết quả)

```
public class TemplateMethodPattern {
    public static void main(String[] args) {

        PageTemplate homePage = new HomePage();
        homePage.showPage();

        System.out.println();
        PageTemplate detailPage = new DetailPage();
        detailPage.showPage();

        System.out.println();
        PageTemplate contactPage = new ContactPage();
        contactPage.showPage();
    }
}
```

5. Code ví dụ (Kết quả)

```
 <header />
 <nav />
Content of home page page
 <footer />

 <header />
 <nav />
Content of detail
 <footer />

 <header />
Content of contact page
 <footer />
```

6. Ưu điểm

- Tái sử dụng code (reuse), tránh trùng lặp code (duplicate): đưa những phần trùng lặp vào lớp cha (abstract class).
- Cho phép người dùng override chỉ một số phần nhất định của thuật toán lớn, làm cho chúng ít bị ảnh hưởng hơn bởi những thay đổi xảy ra với các phần khác của thuật toán.

7. Nhược điểm

- Một số khách hàng có thể bị hạn chế bởi khung thuật toán được cung cấp.
- Bạn có thể vi phạm Nguyên lý thay thế Liskov (Liskov Substitution Principle) bằng cách bỏ qua một bước mặc định trong lớp con.
- Các lớp cụ thể (Concrete Classes) có thể ghi đè tất cả các bước, nhưng không được ghi đè phương thức mẫu (template method) chính.

8. Áp dụng mẫu trong thực tiễn

Template Method với Spring Security

- Spring Security sử dụng Template Method Pattern trong nhiều phần:
- CORS Handling: Như đã đề cập, Spring Security định nghĩa quy trình xử lý yêu cầu CORS nhưng cho phép tùy chỉnh các bước như nguồn gốc, headers, và phương thức HTTP.
- Authentication and Authorization:
- Xác thực (authentication) người dùng.
- Kiểm tra quyền (authorization).
- Trả về kết quả (cho phép hoặc từ chối).
- Người dùng có thể tùy chỉnh các bước cụ thể như cách kiểm tra thông tin xác thực hoặc logic phân quyền.
-

9. Các mẫu liên quan

- Factory Method là một sự chuyên biệt hóa của Template Method. Đồng thời, một Factory Method có thể là một bước trong một Template Method lớn hơn.
- Template Method dựa trên kế thừa: nó cho phép bạn thay đổi các phần của một thuật toán bằng cách mở rộng các phần đó trong các lớp con.
- Strategy dựa trên kết hợp: bạn có thể thay đổi các phần của hành vi đối tượng bằng cách cung cấp cho nó các chiến lược khác nhau tương ứng với hành vi đó.
- Template Method hoạt động ở cấp độ lớp, vì vậy nó là tĩnh.
- Strategy hoạt động ở cấp độ đối tượng, cho phép bạn thay đổi hành vi tại thời điểm chạy (runtime).