TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐÀO TẠO CHẤT LƯỢNG CAO

NGÀNH CÔNG NGHỆ THÔNG TIN

ꝊꝊ

BÁO CÁO THIẾT KẾ PHẦN MỀM HƯỚNG ĐỐI TƯỢNG

Đề tài: Singleton và Adapter Pattern

GVHD: ThS. Nguyễn Trần Thi Văn

SVTH: Lâm Phước Bảo 16110016

Lê Thiện Duy 16110034

Phan Thanh Nam 16110162

Nguyễn Thiên Quốc 16110191

Bùi Quốc Thanh 16110209

Lớp: 16110CL3 **Nhóm 04 – Thứ 6**

TP. Hồ Chí Minh, tháng 11 năm 2019

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Giáo Viên Hướng Dẫn

Nguyễn Trần Thi Văn

LỜI CÁM ƠN

Trước hết, chúng em tỏ lòng biết ơn sâu sắc đến Thầy Nguyễn Trần Thi Văn đã tận tình hướng dẫn trong môn học Thiết kế phần mềm hướng đối tượng. Với những kiến thức, nhận xét đã giúp chúng em chỉnh sửa những thiếu sót và hoàn thành phần chuẩn bị giữa kì.

Do các kỹ năng của nhóm cũng như kiến thức còn ít ỏi nên thông qua lần báo cáo giữa kì này, chúng em rất mong được nhận thêm nhiều ý kiến đóng góp hữu ích hơn từ thầy để giúp chúng em hoàn thiện bản thân hơn.

Chúng em xin chân thành cảm ơn!

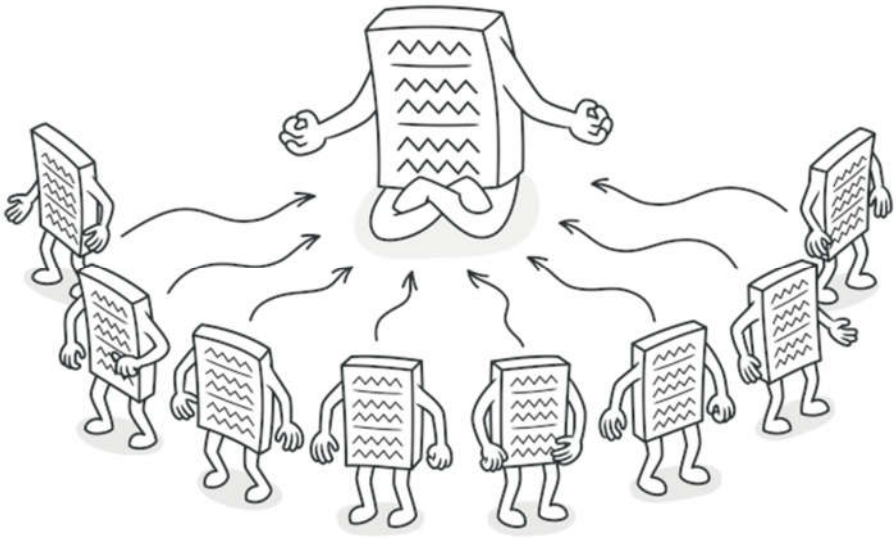
Mục Lục

DANH MỤC CÁC HÌNH

NỘI DUNG

I. Giới thiệu về Singleton Design Pattern

1. Mục đích sử dụng của mẫu

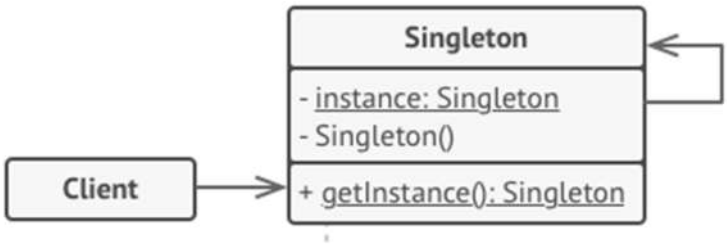


Hình 1. Singleton Design Pattern [1]

Theo Gang of Four patterns thì Single Pattern là một design pattern trong số 5 design pattern thuộc nhóm Creational Design Pattern. (Creational patterns cung cấp các cách để khởi tạo các đối tượng đơn lẻ hoặc các nhóm đối tượng liên quan).

Mẫu singleton đảm bảo rằng chỉ có một đối tượng của một lớp cụ thể được tạo. Tất cả các tham chiếu thêm đến các đối tượng của lớp singleton đều đề cập đến cùng một thể hiện bên dưới.

2. Cấu trúc, các lớp/đối tượng tham gia, ý nghĩa và vai trò của từng lớp trong mẫu

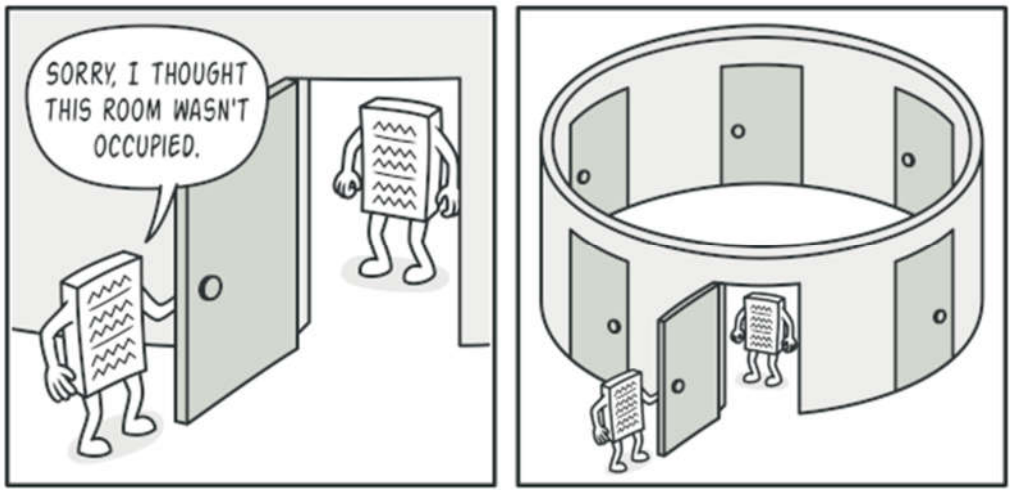


Hình 2. Cấu trúc Singleton [1]

Lớp Singleton khai báo phương thức tĩnh getInstance trả về cùng thể hiện lớp của chính nó.

Hàm tạo Singleton nên được ẩn khỏi mã máy khách. Gọi phương thức getInstance sẽ là cách duy nhất để có được đối tượng Singleton.

3. Những tính chất đặc thù của mẫu



Hình 3. Client thậm chí có thể không nhận ra rằng họ luôn làm việc với cùng một đối tượng [1]

Đảm bảo rằng một class chỉ có duy nhất một instance.

Cung cấp một cách toàn cầu để truy cấp tới instance đó.

4. Lĩnh vực áp dụng và các hệ quả

Bảng 1. Lĩnh vực áp dụng và các hệ quả

Lĩnh vực áp dụng	Hệ quả
Khi một lớp trong chương trình của bạn chỉ có một instance duy nhất cho tất cả các clients.	Singleton vô hiệu hóa tất cả các phương tiện khác để tạo các đối tượng của một lớp ngoại trừ phương thức tạo đặc biệt. Phương thức này hoặc tạo một đối tượng mới hoặc trả về một đối tượng hiện có nếu nó đã được tạo.
Sử dụng mẫu Singleton khi bạn cần kiểm soát chặt chẽ hơn các biến toàn cục.	Không giống như các biến toàn cục, Singleton đảm bảo rằng chỉ có một thể hiện của một lớp. Không có gì, ngoại trừ

	chính lớp Singleton, có thể thay thế thể hiện được lưu trữ.

5. Những ưu, khuyết điểm của mẫu

5.1. Ưu điểm

- Bạn có thể chắc chắn rằng một lớp chỉ có một thể hiện duy nhất.
- Bạn có được một điểm truy cập toàn cầu vào instance đó.
- Đối tượng singleton chỉ được khởi tạo khi nó yêu cầu lần đầu tiên.

5.2. Khuyết điểm

- Vi phạm nguyên tắc trách nhiệm duy nhất. Các mô hình giải quyết hai vấn đề tại thời điểm đó.
- Mẫu Singleton có thể che dấu thiết kế xấu cho một instance, khi các thành phần của chương trình biết quá nhiều về nhau.
- Mẫu này yêu cầu xử lý đặc biệt trong môi trường đa luồng để nhiều luồng giành được tạo ra một đối tượng đơn lẻ nhiều lần.
- Có thể khó cho unit test. Vì hàm tạo của lớp singleton là riêng tư và việc ghi đè các phương thức tĩnh là không thể trong hầu hết các ngôn ngữ.

6. Các mẫu thiết kế có liên quan

- Một Facade thường có thể chuyển đổi thành Singleton vì một đối tượng Facade là đủ trong hầu hết các trường hợp.
- Flyweight sẽ giống với Singleton nếu bằng cách nào đó quản lý để giảm tất cả các trạng thái được chia sẻ của các đối tượng thành chỉ một đối tượng flyweight. Nhưng có hai sự khác biệt cơ bản giữa các mẫu này:
    - Chỉ nên có một cá thể Singleton, trong khi một lớp Flyweight có thể có nhiều thể hiện với các trạng thái intrinsic khác nhau.
    - Đối tượng Singleton có thể thay đổi. Đối tượng Flyweight là bất biến.
- *Abstract Factories*, *Builders* và *Prototypes* tất cả có thể được thực hiện như là một *Singletons*.

7. Ví dụ thực tế

Ứng dụng trong phần mềm window:

Giả sử trong ứng dụng có chức năng bật tắt nhạc nền, khi người dùng mở app thì ứng dụng sẽ tự động mở nhạc nền. Nếu người dùng muốn tắt thì phải vào setting trong app để tắt nó. Trong setting của app, người dùng có thể quản lí việc mở hay tắt nhạc, và bạn sẽ cần sử dụng singleton để quản lí việc này.

Chắc chắn bạn phải cần duy nhất một instance để có thể ra lệnh bật hay tắt. Tại sao lại như vậy? Vì đơn giản bạn không thể tạo một instance để mở nhạc rồi sau đó lại tạo một instance khác để tắt nhạc, lúc này sẽ có hai instance được tạo ra, hai instance này không liên quan đến nhau nên không thể thực hiện việc cho nhau được. Bạn phải hiểu rằng instance nào bật thì chỉ có instance đó mới được phép tắt nên dẫn đến phải cần một instance. [2]

Ứng dụng trong thư viện java:

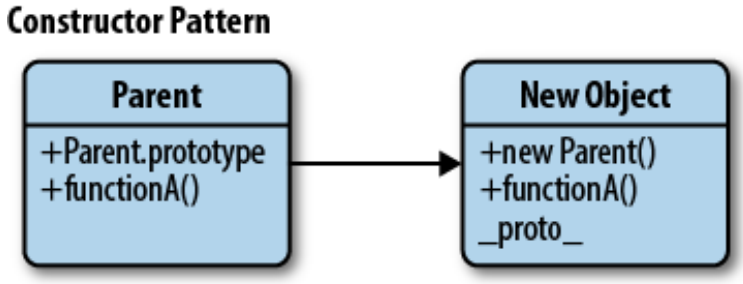
Các thư viện java dưới đây đều chỉ trả về duy nhất một instance trong các lần gọi đến nó:

- java.lang.Runtime#getRuntime()
- java.awt.Desktop#getDesktop()
- java.lang.System#getSecurityManager()

II. Giới thiệu về Constructor Pattern

1. Constructor pattern là gì?

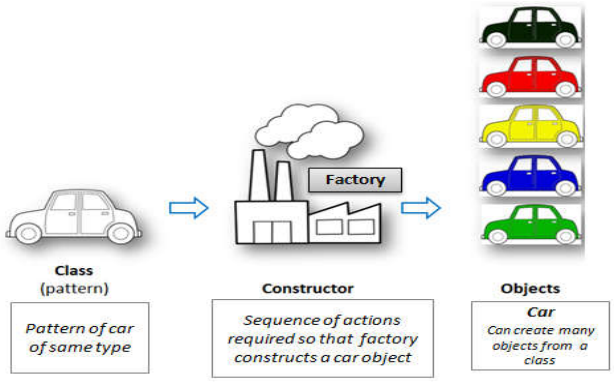
[5] Constructor pattern là một phương thức đặt biệt được sử dụng để tạo một đối tượng mới được tạo sau khi bộ nhớ đã được cấp phát cho nó.



Hình 4. Constructor Pattern [6]

2. Ví dụ

Chúng ta có một mẫu Car và muốn sản xuất mẫu đó với nhiều màu sắc khác nhau.



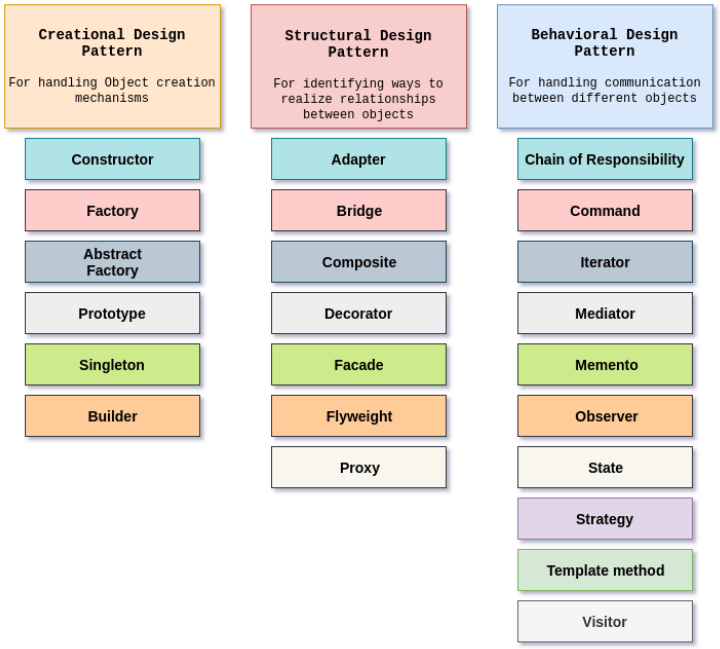
Hình 5. Constuctor Clas Car Color [3]

III. Adapter Pattern

1. Design Pattern là gì?

Design Pattern là một kĩ thuật lập trình hướng đối tượng (OOP) khá quan trọng đối với mọi lập trình viên muốn giỏi đều phải biết. Design Pattern cung cấp cho bạn các "mẫu thiết kế", giải pháp cho các vấn đề thường gặp trong lập trình.

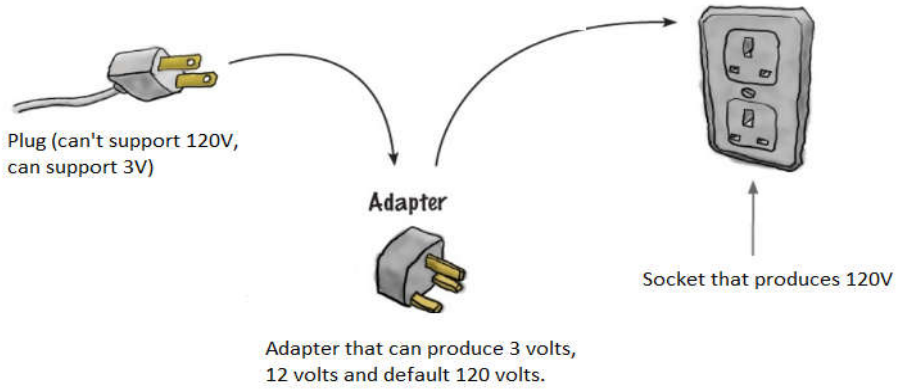
Design Pattern hiện có 23 mẫu thiết kế được định nghĩa trong cuốn "Design patterns Elements of Reusable Object Oriented Software" và được chia làm 3 nhóm là Creational Pattern (nhóm khởi tạo), Structural Pattern (nhóm cấu trúc) và Behaviral Pattern (nhóm tương tác/ hành vi).



Hình 6. Design Pattern

2. Adapter Pattern là gì?

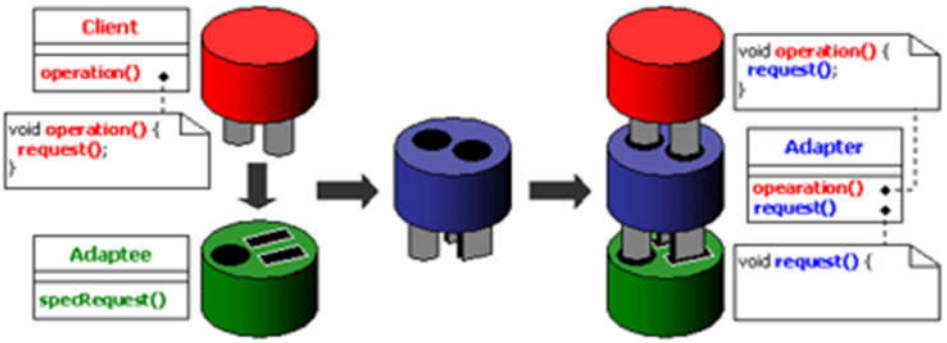
Adapter Pattern (Người chuyển đổi) là một trong những Pattern thuộc nhóm cấu trúc (**Structural Pattern**). Adapter Pattern cho phép các **inteface** (giao diện) không liên quan tới nhau có thể làm việc cùng nhau. Đối tượng giúp kết nối các interface gọi là **Adapter**.



Hình 7. Adapter

Adapter Pattern giữ vai trò trung gian giữa hai lớp, chuyển đổi interface của một hay nhiều lớp có sẵn thành một interface khác, thích hợp cho lớp đang viết. Điều này cho phép các lớp có các interface khác nhau có thể dễ dàng giao tiếp tốt với nhau

thông qua interface trung gian, không cần thay đổi code của lớp có sẵn cũng như lớp đang viết.



Hình 8. Adapter Pattern

Adapter Pattern còn gọi là **Wrapper Pattern** do cung cấp một interface "bọc ngoài" tương thích cho một hệ thống có sẵn, có dữ liệu và hành vi phù hợp nhưng có interface không tương thích với lớp đang viết.

3. Thành Phần

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

- **Adaptee**: định nghĩa interface không tương thích, cần được tích hợp vào.
- **Adapter**: lớp tích hợp, giúp interface không tương thích tích hợp được với interface đang làm việc. Thực hiện việc chuyển đổi interface cho Adaptee và kết nối Adaptee với Client.
- **Target**: một interface chứa các chức năng được sử dụng bởi Client (domain specific).
- **Client:** lớp sử dụng các đối tượng có interface Target.

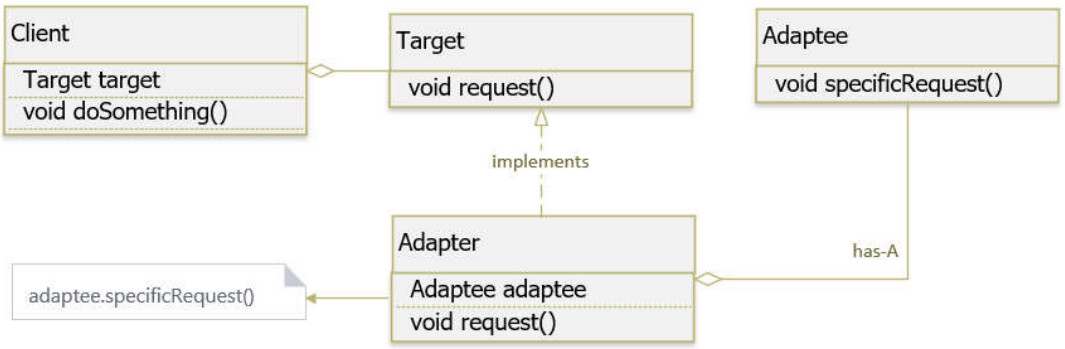
4. Phân loại

Có hai cách để thực hiện Adapter Pattern dựa theo cách cài đặt (implement) của chúng là **Object Adapter – Composition** (Tổng hợp) và **Class Adapter – Inheritance** (Kế thừa).

4.1. Object Adapter – Composition

Trong mô hình Object Adapter, một lớp mới (Adapter) sẽ tham chiếu đến một (hoặc nhiều) đối tượng của lớp có sẵn với interface không tương thích (Adaptee),

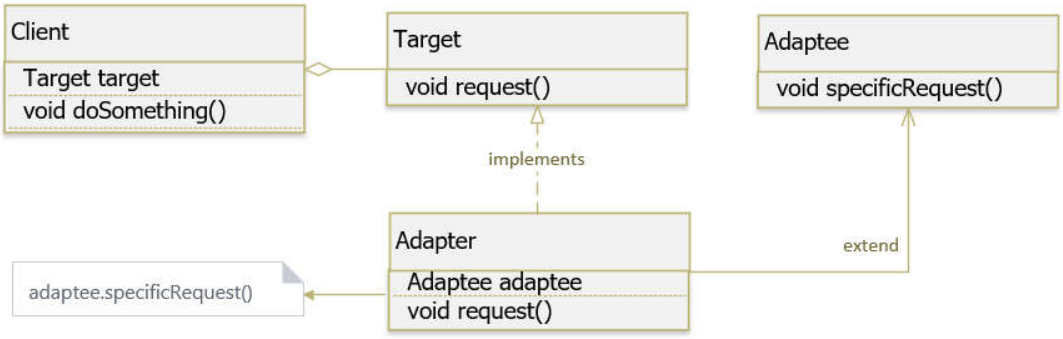
đồng thời cài đặt interface mà người dùng mong muốn (Target). Trong lớp mới này, khi cài đặt các phương thức của interface người dùng mong muốn, sẽ gọi phương thức cần thiết thông qua đối tượng thuộc lớp có interface không tương thích.



Hình 9. Object Adapter

4.2. Class Adapter – Inheritance

Trong mô hình Class Adapter ,một lớp mới (Adapter) sẽ kế thừa lớp có sẵn với interface không tương thích (Adaptee), đồng thời cài đặt interface mà người dùng mong muốn (Target). Trong lớp mới, khi cài đặt các phương thức của interface người dùng mong muốn, phương thức này sẽ gọi các phương thức cần thiết mà nó thừa kế được từ lớp có interface không tương thích.



Hình 10. Class Adapter

4.3. So sánh Class Adapter với Object Adapter

Sự khác biệt chính là Class Adapter sử dụng Inheritance (kế thừa) để kết nối Adapter và Adaptee trong khi Object Adapter sử dụng Composition (tổng hợp) để kết nối Adapter và Adaptee.

Trong cách tiếp cận Class Adapter, nếu một Adaptee là một class và không phải là một interface thì Adapter sẽ là một lớp con của Adaptee. Do đó, nó sẽ không phục vụ tất cả các lớp con khác theo cùng một cách vì Adapter là một lớp phụ cụ thể của Adaptee.

5. Ưu, khuyết điểm của Adapter Pattern

5.1. Ưu điểm

- Việc sử dụng Adapter Pattern đem lại các lợi ích sau:
- Cho phép nhiều đối tượng có interface giao tiếp khác nhau có thể tương tác và giao tiếp với nhau.
- Tăng khả năng sử dụng lại thư viện với interface không thay đổi do không có mã nguồn.

5.2. Khuyết điểm

- Bên cạnh những lợi ích trên, nó cũng nó một số khuyết điểm nhỏ sau:
- Tất cả các yêu cầu được chuyển tiếp, do đó làm tăng thêm một ít chi phí.
- Đôi khi có quá nhiều Adapter được thiết kế trong một chuỗi Adapter (chain) trước khi đến được yêu cầu thực sự.

6. Sử dụng Adapter Pattern khi nào?

Có thể dùng Adapter Pattern trong những trường hợp sau:

Adapter Pattern giúp nhiều lớp có thể làm việc với nhau dễ dàng mà bình thường không thể. Một trường hợp thường gặp phải và có thể áp dụng Adapter Pattern là khi không thể kế thừa lớp A, nhưng muốn một lớp B có những xử lý tương tự như lớp A. Khi đó chúng ta có thể cài đặt B theo Object Adapter, các xử lý của B sẽ gọi những xử lý của A khi cần.

Khi muốn sử dụng một lớp đã tồn tại trước đó nhưng interface sử dụng không phù hợp như mong muốn.

Khi muốn tạo ra những lớp có khả năng sử dụng lại, chúng phối hợp với các lớp không liên quan hay những lớp không thể đoán trước được và những lớp này không có những interface tương thích.

Cần phải có sự chuyển đổi interface từ nhiều nguồn khác nhau.

Khi cần đảm bảo nguyên tắc **Open/ Close** trong một ứng dụng.

7. Các pattern liên quan với Adapter Pattern [4]

Bridge Pattern: có cấu trúc tương tự nhưng mục tiêu khác (tách một giao diện khỏi phần cài đặt).

Decorator Pattern: cung cấp thêm chức năng nhưng không làm thay đổi giao diện, ở đây một Adapter sẽ phối hợp hai đối tượng khác nhau.

Proxy Pattern: định nghĩa một giao diện đại diện cho các đối tượng khác mà không làm thay đổi giao diện của các đối tượng được đại diện, điều này thực hiện được nhờ các Adapter.

8. Một số ví dụ thực tế có sử dụng

Ứng dụng trong các thư viện dựng sẵn của java:[7]

Nhìn đoạn code mẫu dưới là đoạn code bình thường khi nhận input từ màn hình console

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter String");
String s = br.readLine();
System.out.print("Enter input: " + s);
```

Hình 11 Đoạn code nhập input string

Ta biết rằng : System.in là một thể hiện (instance) của Inputstream

Và input stream này đọc datas từ console bằng bytes stream

Trong khi ngược lại, BufferedReader đọc một charater stream

Vấn đề là:

System.in cung cấp các bytes stream trong khi BuffReader lại là nhận các charater stream. Vậy làm cách nào chúng làm việc với nhau như đoạn code trên.

Đó chính là lí do chúng ta thấy có sự xuất hiện của InputStreamReader:

- InputStreamer chính xác là một cầu nối từ bytes stream sang character stream
- InputStreamer sẽ đọc dữ liệu bytes và decode chúng theo dữ liệu characters bằng cách sử dụng một bộ charset cụ thể.

Cuối cùng thì chúng có thể hiểu lẫn nhau và làm việc.

Cũng tương tự ngoài ra còn có các thư viện sau sử dụng Adapter pattern:

java.util.Arrays#asList()

java.io.OutputStreamWriter(OutputStream)

javax.xml.bind.annotation.adapters.XmlAdapter#marshal() and #unmarshal()

IV. Tổng kết

1. Những gì học được

Qua quá trình chuẩn bị, soạn thảo và thiết kế demo, thành viên ở nhóm em đã học được nhiều thứ:

- Cách xây dựng pattern
- Các thành phần pattern
- Những pattern liên quan
- Cách ứng dụng pattern vào bảng demo
- Các thành viên nhóm hoạt động có tổ chức và hiệu quả

2. Khó khăn

- Do trong quá trình tự tìm hiểu nên nhóm em vẫn chưa có sự kiểm chứng là cách hiểu của mình đã hoàn toàn chính xác hay chưa
- Phần UI demo có phần đơn giản vẫn chưa bắt mắt

TÀI LIỆU THAM KHẢO

[1] Jay Chow (2019). Top 10 Javascript Pattern mọi Lập trình viên hàng đầu yêu thích. Lấy từ https://niithanoi.edu.vn/javascript-pattern.html. Xem ngày 14/11/2019.

[2] The Constructor Pattern. Lấy từ https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch09s01.html. Xem ngày 14/11/2019.

[3] GP Coder (2018), Hướng dẫn Java Design Pattern – Adapter. Lấy từ https://gpcoder.com/4483-huong-dan-java-design-pattern-adapter/ Xem ngày 14/11/2019.

[4] Dương Thiên Tứ, Adapter Pattern (mẫu thiết kế tiếp hợp). Lấy từ http://aptech.fpt.edu.vn/chitiet.php?id=388. Xem ngày 15/11/2019.

[5] Refactoring.Guru. Lấy từ https://refactoring.guru/design-patterns/singleton?fbclid=IwAR0uDwGNrWEfeQtscgU5f4eCp8c6R3553KpcloTr-p186kmd2CBe0hFlVjo Xem ngày 15/11/2019.

[6] Topdev. Lấy từ https://topdev.vn/blog/singleton-pattern-la-gi/ Xem ngày 15/11/5019.

[7] Lokesh Gupta- Pradip- Bala- Vinodkumar, Adapter Design Pattern in Java. Lấy từ https://howtodoinjava.com/design-patterns/structural/adapter-design-pattern-in-java/