**Adapter Design Pattern**

A picture containing diagram

Description automatically generated

**1. Giới thiệu**

* Adapter (wrapper) là một mẫu thiết kế thuộc nhóm Structural Pattern – những mẫu thiết kế cho việc thiết kế cấu trúc
* Là mẫu thiết kế chuyển đổi khuôn mẫu (interface) của một lớp thành một khuôn mẫu khác mà phía clients muốn. Cho phép 2 interface không liên quan làm việc cùng nhau.
* 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.
* 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
* Diagram, shape, arrow

  Description automatically generated

**2. Mục đích ra đời**

Hãy tưởng tượng rằng chúng ta đang tạo một ứng dụng theo dõi thị trường chứng khoán. Ứng dụng tải xuống dữ liệu kho từ nhiều nguồn ở định dạng XML và sau đó hiển thị các biểu đồ và sơ đồ đẹp mắt cho người dùng.

Một lúc nào đó, ta quyết định cải thiện ứng dụng bằng cách tích hợp thư viện phân tích thông minh của bên thứ 3. Nhưng có một điểm lưu ý: thư viện phân tích chỉ hoạt động với dữ liệu ở định dạng JSON.

Diagram

Description automatically generatedChúng ta có thể thay đổi thư viện để làm việc với XML. Tuy nhiên, điều này có thể ảnh hưởng đến những đoạn code hiện có, khiến cho cách tiếp cận này không thể thực hiện được.

Chúng ta có thể giải quyết vấn đề này bằng cách tạo ra các Adapter để chuyển từ định dạng XML sang JSON cho mỗi lớp của thư viện phân tích. Sau đó, chúng ta điều chỉnh mã của mình để chỉ giao tiếp với thư viện thông qua các Adapter này. Khi một Adapter nhận được yêu cầu, nó sẽ dịch dữ liệu XML sang JSON và chuyển yêu cầu đến các phương thức thích hợp của đối tượng phân tích trong thư viện. A picture containing timeline

Description automatically generatedAdapter không chỉ có thể chuyển đổi dữ liệu thành nhiều định dạng khác nhau mà còn có thể giúp các đối tượng có interface khác nhau collab với nhau.

Đây là cách nó hoạt động:

* Adapter có một interface tương thích với một trong các object hiện có.
* Với việc sử dụng interface này, object hiện có có thể gọi các phương thức của Adapter một cách an toàn.
* Khi được gọi, Adapter sẽ chuyển yêu cầu đến object thứ hai, nhưng theo một định dạng và thứ tự mà object thứ hai mong đợi.

**3. Kiến trúc**

Có hai cách để thực hiện Adapter Pattern dựa theo cách cài đặt (implement) của chúng:

**a. Object Adapter – Composition**

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

Diagram

Description automatically generated (Adaptee/Service), 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. Các thành phần trong mô hình:

* Client là một class chứa business logic của chương trình
* Client interface mô tả một giao thức mà các lớp khác phải tuân theo để có thể collab với client code
* Service: là một class hữu ích (thường là bên thứ 3 hoặc kế thừa). Client không thể sử dụng trực tiếp lớp này vì nó có interface không tương thích.
* Adapter: là một class có thể hoạt động với cả client và service: nó implements client interface, trong khi đóng gói service object. Adapter khi được gọi từ Client thông qua Adapter Interface sẽ chuyển chúng thành các cuộc gọi service object được bao bọc ở định dạng mà nó có thể hiểu được.

**b. Class Adapter – Inheritance**

Trong mô hình này, một lớp mới (Adapter) sẽ kế thừa lớp có sẵn với interface không tương thích (Adaptee/Service), 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.

Diagram

Description automatically generatedCác thành phần:

* Class Adapter: không cần phải bọc bất kỳ object nào vì nó kế thừa các hành vi từ client và service. Adaptation xảy ra trong các phương thức bị ghi đè. Kết quả của Adapter có thể được sử dụng thay cho một client class hiện có

**So sánh Class Adapter và 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 (chứa trong) 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.
* Object Adapter sẽ tốt hơn vì nó sử dụng Composition để giữ một thể hiện của Adaptee, cho phép một Adapter hoạt động với nhiều Adaptee nếu cần thiết.

**4. Ưu & nhược điểm**

**Ưu điểm**

* Single Responsibility Principle: Có thể tách interface hoặc các đoạn code chuyển đổi dữ liệu khỏi logic nghiệp vụ chính của chương trình
* Open/Closed Principle: Giúp code không bị ảnh hưởng từ các thay đổi hoặc các lần cập nhật phiên bản mới từ API hoặc dịch vụ từ bên thứ ba (thay đổi tên hàm, tên lớp,…)

**Nhược điểm**

* Độ phức tạp tổng thể của mã tăng lên vì bạn cần giới thiệu/khai báo một tập hợp các interface và lớp mới. Đôi khi, việc thay đổi lớp service sao cho phù hợp với phần còn lại của mã của bạn sẽ đơn giản hơn.

**5. Khi nào thì sử dụng**

Adapter được sử dụng khi:

* Muốn sử dụng một số class có sẵn nhưng interface của nó không tương thích với code hiện tại
* Muốn sử dụng lại một số subclass hiện có thiếu một số chức năng và không thể thêm vào lớp cha
* Adapter thường được sử dụng trong môi trường lập trình nơi các thành phần mới hoặc ứng dụng mới cần được tích hợp và hoạt động cùng với các thành phần hiện có.

## Tài liệu tham khảo

[1] Refactoring.Guru. <https://refactoring.guru/design-patterns>

[2] Design Patterns for Dummies, Steve Holzner, PhD

[3] Head First, Eric Freeman

[4] Gang of Four Design Patterns 4.0

[5] Dive into Design Pattern