## Mẫu thiết kế ADAPTER

### Bài toán

Giả sử bạn đang phát triển một ứng dụng chỉ có thể làm việc với định dạng đầu ra là JSON, nhưng một thư viện cần dùng chỉ hỗ trợ XML. Bạn có thể tạo một Adapter để chuyển đổi từ XML sang JSON, giúp ứng dụng sử dụng thư viện đó mà không cần sửa đổi code của thư viện.

### Phân tích

Sử dụng Adapter trong ví dụ trên là cần thiết bởi vì nó giải quyết được vấn đề không tương thích giữa hai hệ thống khác nhau mà không cần thay đổi code của bất kỳ hệ thống nào. Adapter giúp kết nối hai interface không tương thích bằng cách tạo ra một lớp trung gian.

### Sơ đồ lớp

A diagram of a software application

Description automatically generated

### các thành phần và mối liên hệ

* IJsonTarget: Interface mà client mong muốn làm việc với.
* JsonAdapter: Lớp kết nối giữa IJsonTarget và XMLAdaptee, thực hiện chuyển đổi interface của XMLAdaptee thành interface mà IJsonTarget yêu cầu.
* XMLAdaptee: Lớp có interface không tương thích với IJsonTarget, nhưng có thể sử dụng được thông qua Adapter.
* Client: Lớp sử dụng IJsonTarget và không biết đến sự hiện diện của XMLAdaptee.

### Giao diện kết quả

A screenshot of a computer

Description automatically generated

### Khi nào sử dụng Adapter Pattern?

* Khi bạn muốn sử dụng một lớp đã có sẵn nhưng interface của lớp đó không tương thích với hệ thống.
* Khi bạn cần kết nối một hệ thống cũ vào một hệ thống mới.

## Mẫu thiết kế FACADE

### Bài toán

Giả sử bạn đang xây dựng một hệ thống máy tính với các thành phần như CPU, Bộ nhớ, và Màn hình. Mỗi thành phần này có thể có các phương thức phức tạp, nhưng bạn chỉ muốn cung cấp một phương thức đơn giản để bật máy tính. Mẫu Facade có thể giúp đơn giản hóa các thao tác này.

### Phân tích

Facade giúp cung cấp một giao diện đơn giản cho một hệ thống phức tạp. Mẫu này ẩn đi sự phức tạp của hệ thống, chỉ giữ lại những chức năng cần thiết, và cung cấp một giao diện dễ sử dụng cho người dùng hoặc các hệ thống khác.

### Sơ đồ lớp

A diagram of a computer component

Description automatically generated

### các thành phần và mối liên hệ

* Các subsystem (CPU, Memory, HardDrive):
  + Các lớp này thực hiện các thao tác phức tạp như tải dữ liệu từ ổ cứng, xử lý và thực thi lệnh. Nếu không có Facade, client sẽ phải gọi từng phương thức của các lớp này, dẫn đến mã nguồn phức tạp và khó bảo trì.
* Facade (ComputerFacade):
  + Lớp ComputerFacade cung cấp một phương thức đơn giản startComputer() mà không yêu cầu client phải biết chi tiết về cách thức hoạt động của từng phần của hệ thống (CPU, bộ nhớ, ổ cứng).

### Giao diện kết quả

A screenshot of a computer

Description automatically generated

### Khi nào sử dụng FACADE?

* Khi muốn giảm độ phức tạp bằng cách cung cấp một giao diện đơn giản cho các hệ thống phức tạp.

## Mẫu thiết kế COMPOSITE

### Bài toán

Một hệ thống thư mục (file system) bao gồm thư mục (folder) và tệp (file). Mỗi thư mục có thể chứa các thư mục con hoặc các tệp. Tất cả đều được xem như "thành phần" (component).

### Phân tích

Composite Pattern sẽ giúp chúng ta:

* Làm việc với tệp và thư mục một cách thống nhất.
* Duyệt qua các thành phần của cây một cách đơn giản.

### Sơ đồ lớp

A diagram of a computer

Description automatically generated

### các thành phần và mối liên hệ

* Component (FileSystem):
  + Abstract class định nghĩa các operation chung cho tất cả các thành phần
* Leaf (File):
  + Đại diện cho các đối tượng không có con
  + Implements các operation của Component
* Composite (Folder):
  + Chứa các thành phần con (có thể là Leaf hoặc Composite khác)
  + Implements các operation của Component
  + Định nghĩa hành vi để quản lý các thành phần con

### Giao diện kết quả

A screenshot of a computer program

Description automatically generated

### Khi nào sử dụng Composite Pattern?

* Khi cần biểu diễn một hệ thống phân cấp (cấu trúc cây), như thư mục, menu, hoặc tổ chức nhân sự.
* Khi muốn xử lý các đối tượng riêng lẻ và nhóm đối tượng theo cách thống nhất.
* Khi hệ thống cần mở rộng linh hoạt mà không ảnh hưởng đến mã client.

## Mẫu thiết kế Chain of Responsibility

### Bài toán

Giả sử công ty có ba cấp bậc nhân viên: Nhân viên (Employee), Trưởng phòng (Manager), và Giám đốc (Director). Mỗi cấp bậc có quyền quyết định liệu một yêu cầu chi tiêu có thể được phê duyệt hay không, hoặc yêu cầu đó cần được chuyển lên cấp cao hơn.

### Phân tích

**Chain of Responsibility** giúp truyền tải một yêu cầu qua một chuỗi các đối tượng để xử lý yêu cầu đó. Mỗi đối tượng trong chuỗi có thể xử lý yêu cầu hoặc truyền nó đến đối tượng tiếp theo trong chuỗi. Mẫu này cho phép xây dựng một chuỗi các đối tượng xử lý có thể quyết định việc xử lý yêu cầu mà không cần phải biết trước ai sẽ xử lý yêu cầu.

### Sơ đồ lớp

A diagram of a company

Description automatically generated

### các thành phần và mối liên hệ

* Approver: Định nghĩa các phương thức chung và chuỗi các đối tượng phê duyệt.
* Employee, Manager, Director: Các lớp này xử lý các yêu cầu cụ thể dựa trên số tiền, và nếu không thể xử lý, chúng sẽ chuyển tiếp yêu cầu cho đối tượng tiếp theo.

### Giao diện kết quả

A screenshot of a computer screen

Description automatically generated

### Khi nào sử dụng Chain of Responsibility?

* Khi gửi yêu cầu qua một chuỗi các đối tượng và không muốn quyết định ai sẽ xử lý yêu cầu đó.
* Khi bạn cần cho phép mở rộng hệ thống một cách dễ dàng bằng cách thêm mới các handler mà không làm thay đổi mã nguồn hiện tại.

## Mẫu thiết kế VISITOR

### Bài toán

Một hệ thống thư mục (file system) bao gồm thư mục (folder) và tệp (file). Mỗi thư mục có thể chứa các thư mục con hoặc các tệp. Chúng ta muốn thêm thao tác tính total size mà không muốn thay đổi cấu trúc của các lớp hiện tại.

### Phân tích

Visitor Pattern sẽ giúp chúng ta:

* Tách rời các thao tác (operations) khỏi cấu trúc đối tượng.
* Dễ dàng thêm các hành vi mới mà không cần sửa đổi các lớp cấu trúc.

### Sơ đồ lớp

A diagram of a computer

Description automatically generated

### các thành phần và mối liên hệ

Visitor :

* Định nghĩa các phương thức để xử lý từng loại phần tử trong cấu trúc cây.
* ConcreteVisitor (Visitor cụ thể):
* Thực hiện các hành vi cụ thể trên các loại phần tử khác nhau.
* Ví dụ: SizeCalculator để tính tổng kích thước, hoặc NamePrinter để liệt kê tên các phần tử.

### Giao diện kết quả

A screenshot of a computer

Description automatically generated

### Khi nào sử dụng VISITOR Pattern?

* Khi bạn cần thêm các phép toán hoặc hành vi mới mà không muốn thay đổi cấu trúc của các lớp hiện tại.
* Khi bạn có một cấu trúc đối tượng phức tạp và cần thực hiện các hoạt động khác nhau trên các đối tượng đó.