***BỘ GIÁO DỤC VÀ ĐÀO TẠO***

***TRƯỜNG ĐẠI HỌC GIA ĐỊNH***

***KHOA CÔNG NGHỆ THÔNG TIN***

***ĐỒ ÁN MÔN HỌC***

***LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG NÂNG CAO***

**Giảng viên hướng dẫn: LÊ HUỲNH PHƯỚC**

**Sinh viên thực hiện: PHẠM HUỲNH TÍN 2004110021 – K14DCPM02**

**LÊ VIỆT ĐỨC 2004110028 - K14DCPM01**

**PHAN HUY NGỌ 2004110022 - K14DCPM02**

***🙜 Hk3 năm học 2021-2022🙞***

**MỤC LỤC**

[**LỜI CẢM ƠN** 3](#_Toc112357937)

[***I .ADAPTER PATTERN*** 4](#_Toc112357938)

[**1.** **Giới thiệu** 4](#_Toc112357939)

[**2. Mục đích ra đời** 4](#_Toc112357940)

[**3. Kiến trúc** 6](#_Toc112357941)

[**a. Object Adapter – Composition** 6](#_Toc112357942)

[**b. Class Adapter – Inheritance** 7](#_Toc112357943)

[\***So sánh Class Adapter và Object Adapter** 7](#_Toc112357944)

[**4.Ưu điểm và nhược điểm của Adapter Pattern** 8](#_Toc112357945)

[**a. Ưu điểm** 8](#_Toc112357946)

[**b. Nhược điểm** 8](#_Toc112357947)

[**5. Trường hợp Adapter Pattern được sử dụng** 8](#_Toc112357948)

[**6. Soure code minh họa ( AdapterDemo)** 8](#_Toc112357949)

[**7. Chương trình cụ thể áp dụng mẫu** (**ProgramApdater)** 14](#_Toc112357950)

[***II. FACADE PATTERN*** 15](#_Toc112357951)

[**1. Giới thiệu** 15](#_Toc112357952)

[**2. Mục đích ra đời** 15](#_Toc112357953)

[**a. Problem** 15](#_Toc112357954)

[**b. Solution** 16](#_Toc112357955)

[**3.Kiến trúc** 18](#_Toc112357956)

[**4. Ưu điểm và nhược điểm** 19](#_Toc112357957)

[**a. Ưu điểm** 19](#_Toc112357958)

[**b. Nhược điểm** 19](#_Toc112357959)

[**5. Trường hợp Façade Pattern được sử dụng** 19](#_Toc112357960)

[**6. Soure code minh họa và chương trình áp dụng mẫu** 20](#_Toc112357961)

# **LỜI CẢM ƠN**

Em xin gửi lời cảm ơn chân thành đến **thầy Lê Huỳnh Phước**, người đã tận tình chỉ bảo, hướng dẫn nhóm em hoàn thành đề tài một cách tốt đẹp. Và em cũng thành thật cảm ơn quý thầy cô trong bộ môn LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG NÂNG CAO cùng quý thầy cô khoa CNTT cũng như quý thầy cô trong thư viện khoa và trường đã tạo điều kiện thuận lợi nhất cho em trong việc tìm kiếm tài liệu tham khảo, nghiên cứu, học hỏi.

Em xin chân thành ghi nhận những ý kiến đóng góp quý báu của quý thầy cô cùng các bạn vì nhờ đó mà em được mở mang kiến thức và thêm nhiều bài học bổ ích. Bên cạnh những kết quả đạt được trong quá trình làm đề tài nhưng khó tránh khỏi những thiếu sót. Kính mong quý thầy cô cùng các bạn đóng góp thêm ý kiến để đề tài được trọn vẹn hơn.

Sau cùng em xin kính chúc tất cả quý thầy cô được dồi vào sức khỏe để dìu dắt đàn em mai sau ngày một tiến bộ hơn. Kính chúc quý thầy cô an khang thịnh vượng, tràn ngập niềm hạnh phúc.

***Sinh viên thực hiện***

***Phạm Huỳnh Tín***

***Lê Việt Đức***

***Phan Huy Ngọ***

***I .ADAPTER PATTERN***

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 khuôn mẫu 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 generated**

- Chúng 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 generated**

- Adapter 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**

### **a. Object Adapter – Composition**

Diagram

Description automatically generated

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**

Diagram

Description automatically generated

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.

Các 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 điểm và nhược điểm của Adapter Pattern**

### **a. 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,…)

### **b. Nhược điểm**

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

## **5. Trường hợp Adapter Pattern được 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.

## **6. Soure code minh họa ( AdapterDemo)**

- InterfaceAdapterDemo: + Duck:

Graphical user interface, text

Description automatically generated

+ Turkey:

Graphical user interface, text

Description automatically generated

- MainTestDriveDemo:

Graphical user interface, text, application

Description automatically generated

- ObjectAdapterDemo:

+ DuckAdapter:

Graphical user interface, text, application

Description automatically generated

+ MalldardDuck:

Graphical user interface, text, application, email

Description automatically generated

+ TurkeyAdapter:

Text

Description automatically generated

+WildTurkey:

Graphical user interface, text, application, email

Description automatically generated

## **7. Chương trình cụ thể áp dụng mẫu** (**ProgramApdater)**

- Form: + LoginForm:

Text

Description automatically generated with medium confidence

- Main: + Main Form:

Graphical user interface, text, application

Description automatically generated

# ***II. FACADE PATTERN***

## **1. Giới thiệu**

* Facade là một mẫu thiết kế thuộc nhóm cấu trúc (Structural Pattern).
* Facade Pattern cung cấp cho chúng ta một giao diện chung đơn giản thay cho một nhóm các giao diện có trong một hệ thống con (subsystem). Facade Pattern định nghĩa một giao diện ở cấp độ cao hơn để giúp cho người dùng có thể dễ dàng sử dụng hệ thống con này.
* Facade Pattern cho phép các đối tượng truy cập trực tiếp giao diện chung này để giao tiếp với các giao diện có trong hệ thống con. Mục tiêu là che giấu các hoạt động phức tạp bên trong hệ thống con, làm cho hệ thống con dễ sử dụng hơn.
* Tần suất sử dụng: khá cao

## **2. Mục đích ra đời**

### **a. Problem**

Khi ta chúng ta có chuỗi các hành động được thực hiện theo thứ tự, và các hành động này lại được yêu cầu ở nhiều nơi trong phạm vi ứng dụng của chúng ta (ví dụ khi làm việc với một số lượng lớn các đối tượng trong hệ thống hay một thư viện phức tạp, chúng ta cần phải tự khởi tạo tất cả đối tượng này, thường xuyên theo dõi các thay đổi của nó, thứ tự logic cũng phải xử lý chính xác với bên thứ 3. Từ đó logic nghiệp vụ của các lớp trở nên gắn kết chặt chẽ với thư viện thử 3)

🡺 Vậy mỗi lúc bạn cần dùng đến nó bạn lại phải copy/paste hoặc viết lại đoạn code đó vào những nơi cần sử dụng trong ứng dụng. Điều này có vẻ cũng giúp ta nhanh hơn, nhưng nếu chúng ta nhận ra cần phải thay đổi lại cấu trúc và hoặc một đoạn code nào đó trong hầu hết chuỗi hành động đó, vậy chúng ta phải xử lý như thế nào?

Shape, square

Description automatically generated

Đây chính là mấu chốt của vấn đề, chúng ta sẽ phải đi lục lại đoạn code đó ở tất cả các nơi, rồi lại sửa nó. Điều này quá tốn thời gian và hơn nữa lại còn mất đi sự kiểm soát các đoạn code của chúng ta và trong quá trình đó còn có nguy cơ phát sinh lỗi. Vậy giải pháp là gì?

🡺 Phương pháp Solution ra đời

### **b. Solution**

Chúng ta cần thiết kế một Facade, và trong đó phương thức của Facade sẽ xử lý các đoạn code được sử dụng với nhiều, lặp lại. Với Facade, chúng ta sẽ chỉ cần gọi Facade để thực thi các hành động dựa trên các parameters được cung cấp.

🡺 Bây giờ nếu chúng ta cần bất kỳ thay đổi nào trong quá trình trên, công việc sẽ đơn giản hơn rất nhiều, chỉ cần thay đổi các xử lý trong façade, mọi thứ sẽ được đồng bộ.

Diagram, shape

Description automatically generated

## **3.Kiến trúc**

Diagram

Description automatically generated

Các thành phần trong mô hình:

* Facade*:* Facade nắm rõ được hệ thống con nào đảm nhiệm việc đáp ứng yêu cầu của client, nó sẽ chuyển yêu cầu của client đến các đối tượng hệ thống con tương ứng.
* Addition Facade*:* có thể được tạo ra để tránh việc làm phức tạp một facade. Có thể được sử dụng bởi cả client và facade.
* Complex Subsystems*:* Bao gồm nhiều object khác nhau, được cài đặt các chức năng của hệ thống con, xử lý công việc được gọi bởi Facade. Các lớp này không cần biết Facade và không tham chiếu đến nó.
* Client*:* Đối tượng sử dụng Facade để tương tác với các subsystem thay vì gọi subsystem trực tiếp

Các đối tượng Facade thường là Singleton bởi vì chỉ cần duy nhất một đối tượng Facade.

## **4. Ưu điểm và nhược điểm**

### **a. Ưu điểm**

* Ta có thể tách mã nguồn của mình ra khỏi sự phức tạp của hệ thống con
* Hệ thống tích hợp thông qua Facade sẽ đơn giản hơn vì chỉ cần tương tác với Facade thay vì hàng loạt đối tượng khác.
* Tăng khả năng độc lập và khả chuyển, giảm sự phụ thuộc.
* Có thể đóng gói nhiều hàm được thiết kế không tốt bằng 1 hàm có thiết kế tốt hơn.

### **b. Nhược điểm**

* Class Facade của bạn có thể trở lên quá lớn, làm quá nhiều nhiệm vụ với nhiều hàm chức năng trong nó.
* Dễ bị phá vỡ các quy tắc trong SOLID.
* Việc sử dụng Facade cho các hệ thống đơn giản, ko quá phức tạp trở nên dư thừa.

## **5. Trường hợp Façade Pattern được sử dụng**

Dưới đây là một số trường hợp mà khi gặp sẽ phải cân nhắc sử dụng Facade pattern:

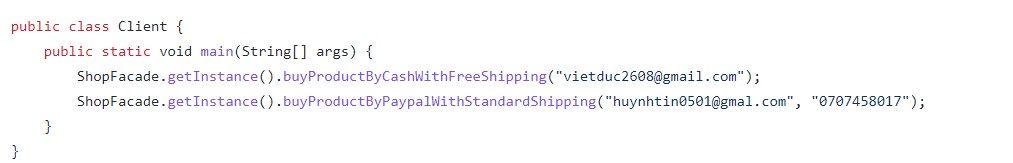
* *Muốn gom nhóm chức năng lại để Client dễ sử dụng.* Khi hệ thống có rất nhiều lớp làm người sử dụng rất khó để có thể hiểu được quy trình xử lý của chương trình. Và khi có rất nhiều hệ thống con mà mỗi hệ thống con đó lại có những giao diện riêng lẻ của nó nên rất khó cho việc sử dụng phối hợp. Khi đó có thể sử dụng Facade Pattern để tạo ra một giao diện đơn giản cho người sử dụng một hệ thống phức tạp.
* *Giảm sự phụ thuộc.* Khi bạn muốn phân lớp các hệ thống con. Dùng Façade Pattern để định nghĩa cổng giao tiếp chung cho mỗi hệ thống con, do đó giúp giảm sự phụ thuộc của các hệ thống con vì các hệ thống này chỉ giao tiếp với nhau thông qua các cổng giao diện chung đó.
* *Tăng khả năng độc lập và khả chuyển*
* *Khi người sử dụng phụ thuộc nhiều vào các lớp cài đặt.* Việc áp dụng Façade Pattern sẽ tách biệt hệ thống con của người dùng và các hệ thống con khác, do đó tăng khả năng độc lập và khả chuyển của hệ thống con, dễ chuyển đổi nâng cấp trong tương lai.
* *Đóng gói nhiều chức năng, che giấu thuật toán phức tạp.*
* *Cần một interface không rắc rối mà dễ sử dụng.*

Ví dụ: Khi bạn gọi điện đến shop để đặt hàng. Khi đó tổng đài sẽ là Facade của tất cả dịch vụ và phòng ban của shop. Hệ thống sẽ cung cấp cho bạn một giao diện đơn giản qua điện thoại để đặt hàng, thanh toán, giao hàng hay nhiều dịch vụ khác nhau.

Diagram

Description automatically generated

## **6. Soure code minh họa và chương trình áp dụng mẫu**

- Façade:  
+ Client:  


+ ShopFacade:

A picture containing text

Description automatically generated

Text

Description automatically generated

- Subsystem:  
+ AccountService:   
Text

Description automatically generated

+ **EmailService:**

Text

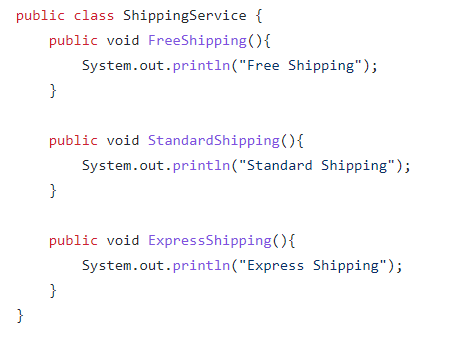
Description automatically generated

+**PaymentService:**

Graphical user interface, text, application

Description automatically generated

+**ShippingService:**



+ **SmsService:**Graphical user interface, text

Description automatically generated with medium confidence