

**Mục lục**

[1. Bài toán đặt ra. 3](#_Toc74164559)

[2. Giải quyết bài toán theo cách đơn thuần. 3](#_Toc74164560)

[3. Tổng quan về Adapter Pattern. 5](#_Toc74164561)

[4. Kiến trúc Adapter Pattern. 6](#_Toc74164562)

[5. Phân loại. 8](#_Toc74164563)

[6. Cài đặt Adapter Pattern. 9](#_Toc74164564)

[7. Ưu điểm và nhược điểm của Adapter Pattern. 10](#_Toc74164565)

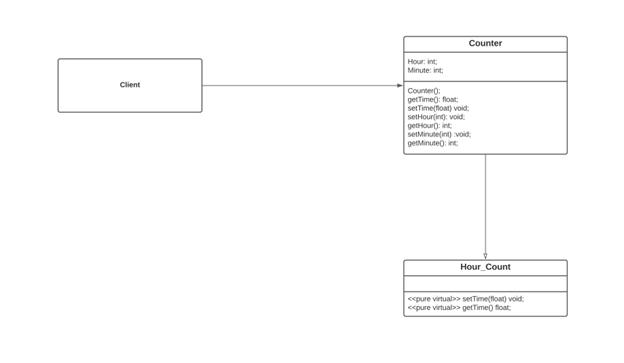
[8. Một số bài toán. 10](#_Toc74164566)

# 1. Bài toán đặt ra.

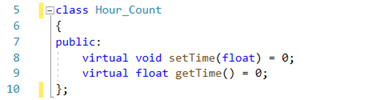
- Giả sử trong 1 tiệm net chỉ gồm có 2 thành phần chính là UI (User Interface) và 1 thành phần Back-end quản lý các người dùng mua thời gian và đưa dữ liệu lên cho UI hiển thị. Khi vào mua thời gian dùng máy bạn sẽ mua với đơn vị là giờ, phút (ví dụ: 1 giờ 30 phút).

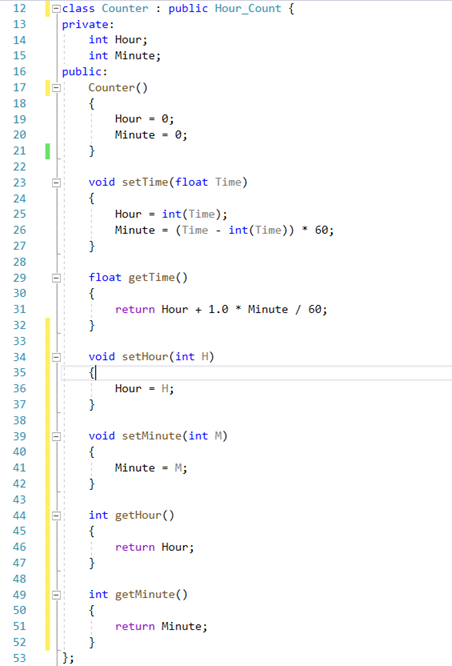
- Đột nhiên vào 1 ngày nào đó phía Back-end muốn sử dụng đơn vị là giờ với phần thập phân (ví dụ: 1.5 giờ) để thuận tiện và nhanh gọn hơn cho việc quản lý. Lúc này chúng ta sẽ thấy được interfaces ban đầu và lúc sau sẽ khác nhau (khác về function name, về danh sách parameters, kiểu dữ liệu của parameters). Vì thế, muốn sử dụng lại interface cũ thì chúng ta đang đứng trước nguy cơ phải sửa code.

# 2. Giải quyết bài toán theo cách đơn thuần.



*Class diagram của bài toán*





*Source code của bài toán*

- Chúng ta sẽ sửa lại class Counter kế thừa từ interface mà người bảo trì mong muốn trong khi đó vẫn giữ các phương thức giao diện mà người dùng mong muốn nhưng cũng đồng thời trích xuất ra dữ liệu mà người bảo trì cần.

- Tuy nhiên, như thế sẽ vi phạm nguyên tắc O trong SOLID là đã sửa đổi class có sẵn. Hơn nữa làm cho class Counter mới phải xử lý nhiều chức năng, điều này cũng đã vi phạm tới nguyên tắc S trong SOLID. Điều này sẽ ảnh hưởng tới việc bảo trì, mở rộng và tái sử dụng sau này của code.

- Cũng chính vì thế, ta có 1 phương pháp tối ưu hơn vừa giúp thõa mãn mong muốn của phía bên người dùng và bảo trì mà không vi phạm 1 nguyên tắc nào như trên. Đó là áp dụng adapter pattern vào giải quyết vấn đề.

# 3. Tổng quan về Adapter Pattern.

- Adapter Pattern chuyển đổi giao diện của một lớp thành một giao diện khác mà client mong đợi. Adapter cho phép các lớp hoạt động cùng nhau mà bình thường là không thể bởi vì sự không tương thích về interface.

- Adapter Pattern là một những mẫu thiết kế thuộc nhóm Structural design patterns.

- Adapter giải quyết các vấn đề như:

+ Làm cách nào để sử dụng lại một lớp không có giao diện mà máy khách yêu cầu?

+ Làm thế nào để các lớp có giao diện không tương thích có thể hoạt động cùng nhau?

+ Làm thế nào để có thể cung cấp một giao diện thay thế cho một lớp?

# 4. Kiến trúc Adapter Pattern.

- Các thành phần cơ bản của một Adapter Pattern:

+ Adaptee: định nghĩa interface không tương thích, cần được tích hợp vào.

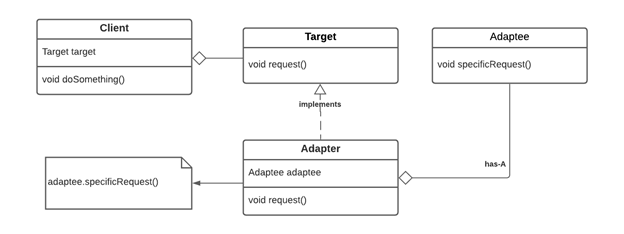
+ Adapter: lớp tích hợp, giúp interface không tương thích tích hợp được với interface đang làm việc. Thực hiện việc chuyển đổi interface cho Adaptee và kết nối Adaptee với Client.

+ Target: một interface chứa các chức năng được sử dụng bởi Client (domain specific).

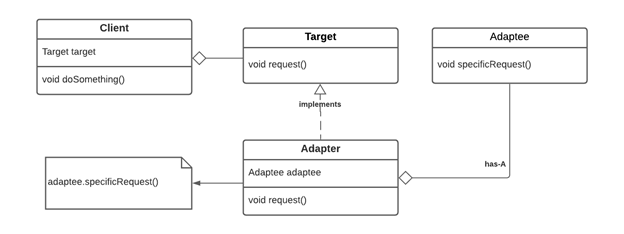
+ Client: lớp sử dụng các đối tượng có interface Target.

- Cách thực hiện:

+ Object Adapter – Composition (Chứa trong): 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 (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.



+ Class Adapter – Inheritance (Kế thừa): trong mô hình này, 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.



# 5. Phân loại.

- Dựa trên kiến trúc ta sẽ có 2 loại adapter pattern:

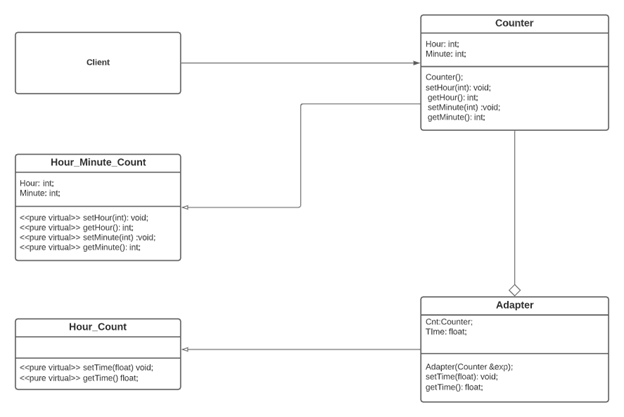
+ Object adapter sử dụng Composition.

+ Class adapter sử dụng Inheritance.

* So sánh Object Adapter và Class Adapter:

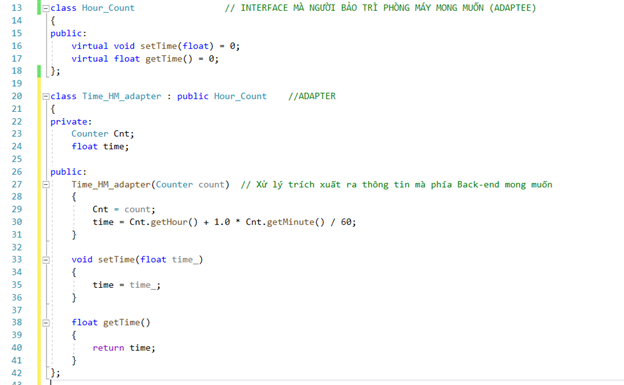
|  |  |
| --- | --- |
| Object Adapter | Class Adapter |
| Sử dụng Composition nên cho phép Adapter hoạt động với nhiều Adaptee. | Chỉ có thể điều chỉnh một lớp adaptee (adapter sẽ kế thừa lớp adaptee). |
| Sử dụng các hành vi có sẵn ở adaptee. | Có thể override hành vi của adaptee nếu cần. |
| Viết ít code để ủy quyền cho adaptee. | Viết nhiều code để override. |
| Sử dụng linh hoạt, tiện lợi cho dữ liệu lớn. | Thích hợp sử dụng cho đối tượng nhỏ. |

# 6. Cài đặt Adapter Pattern.



*Class diagram của bài toán áp dụng Adapter*

* Một lớp adapter sẽ kế thừa từ lớp Hour\_Count (Lớp của nhà bảo trì) để đưa dữ liệu trong giao diện người dùng chuyển hóa thành dữ liệu mà nhà bảo trì cần.



*Code* *của bài toán áp dụng Adapter*

# 7. Ưu điểm và nhược điểm của Adapter Pattern.

- 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.

+ Làm việc với adapter class thay vì sửa đổi bên trong adaptee class đã có sẵn, thuận tiện cho việc mở rộng (Open/closed principle).

- Bên cạnh những lợi ích trên, nó cũng có một số khuyết điểm nhỏ sau:

+ Trong quá trình cài đặt Adapter Pattern có thể dẫn đến trường hợp tồn tại một chuỗi các Adapter dẫn đến việc gia 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.

+ Việc sử dụng Adapter Pattern có thể khiến cho code của chương trình trở nên nhiều hơn và phức tạp hơn. Cần nhiều sự kết hợp từ lớp Client, Adapter, Adaptee cho việc chuyển đổi giao diện trên 1 module nào đó.

# 8. Một số bài toán.

a/ Chuyển đổi ngôn ngữ.

* Đề bài: Một người Việt muốn trao đổi với một người Anh. Tuy nhiên, 2 người này không biết ngôn ngữ của nhau nên cần phải có một người để chuyển đổi từ ngôn ngữ tiếng Việt sang ngôn ngữ tiếng Anh.

- Mô hình hoá với Adapter Pattern:

+ Client: người Việt sẽ là Client trong ví dụ này, vì anh ta cần gửi một số message cho người Anh.

+ Target: đây là nội dung message được Client cung cấp cho thông dịch viên (Translator / Adapter).

+ Adapter: thông dịch viên (Translator) sẽ là Adapter, nhận message tiếng Việt từ Client và chuyển đổi nó sang tiếng Anh trước khi gửi cho người Anh.

+ Adaptee: đây là interface hoặc class được người Anh sử dụng để nhận message được chuyển đổi từ thông dịch viên (Translator).

b/ Chuyển đổi đơn vị.

- Đề bài: bạn cần phát triển ứng dụng để trả về nhiệt độ không khí (độ C). Hiện tại có dịch vụ ở công ty A có thể đo nhiệt không khí, tuy nhiên đơn vị chuẩn được đơn vị này sử dụng là độ F. Do vậy, ta cần tạo ra một thành phần trung gian cho phép chuyển từ đơn vị độ F sang đơn vị độ C.

- Mô hình hóa với Adapter Pattern:

+ Client: công ty A là client vì nó sẽ cung cấp nhiệt độ đo được cho mình.

+ Target: là nhiệt độ mà client cung cấp cho bộ chuyển đổi (Adapter).

+ Adapter: nhận nhiệt độ từ Client và chuyển đổi nó sang đơn vị độ C trước khi đưa cho chúng ta.

+ Adaptee: đây là interface hoặc class được chúng ta sử dụng để nhận nhiệt độ được chuyển đổi từ bộ chuyển đổi (Adapter).

c/ Chuyển đổi tiền tệ.

* Đề bài: Giả sử trong 1 game mua bán hàng, người dùng sử dụng tiền đô và tiền vàng để mua hàng thông dụng và kim cương dành cho những khách hàng có nộp tiền. Tuy nhiên, vào 1 hôm nọ, đội ngũ phát triển trò chơi nhận thấy rằng nếu tích hợp cả vàng và tiền đô thì trò chơi sẽ trở nên nhẹ, dễ bảo trì và phát triển hơn nhiều. Tuy nhiên, theo 1 sự khảo sát thì các khách hàng vẫn muốn sử dụng tiền đô và tiền vàng hơn thay vì tích hợp cả 2. Vì vậy, ta cần phần trung giang để quy đổi từ tiền đô sang tiền vàng.
* Mô hình hóa với Adapter Pattern:
* Người dùng là Client.
* Target: Tiền đô, Tiền vàng.
* Adapter: Nhận lượng tiền đô và tiền vàng từ người dùng và đổi chúng thành tiền vàng cho người phát triển game.
* Adaptee: Đây là Interface mà nhà phát triển game dùng tiền vàng để thực hiện các chức năng khác của trò chơi.

*\*Các bài tập Adapter của các thành viên nhóm:*

**1/ Trần Bảo Tín**

*Đề:* Trong trò chơi plants vs zombies, chúng ta có peashooter (Loại chỉ bắn đạn thường) nhưng để tiêu diệt những tên zombies băng giá chúng ta cần loại có thể bắn được đạn lửa. Mọi chuyện sẽ dễ dàng nếu chúng ta có thể mua được fire-peashooter. Nhưng trong trò chơi lại không tồn tại loại này. Vậy phải làm thế nào để có thể bắn đạn lửa?

*Giải quyết:* sử dụng Adapter Design Pattern.

-Client: Người chơi

-Target: Peashooter.

-Adapter: Torchwood (Đóng vai trò như ngọn đuốc, khi đạn bắn qua ngọn đuốc này đạn bình thường sẽ trở thành đạn lửa)

-Adaptee: Fire-peashooter

**2/ Võ Ngọc Tín**

*Đề*: Trong siêu thị, ở các gian hàng trò chơi. Ban đầu chúng ta phải sử dụng tiền mặt để có thể chơi được cái trò chơi này. Tuy nhiên, sau khi triển khai, phía bên siêu thị đã quyết định sẽ chỉ sử dụng 1 loại tiền xu để dễ quản lý nguồn ra nguồn vào hơn. Mọi chuyện sẽ ổn nếu khách hàng cũng sử dụng tiền xu này cho cuộc sống thường nhật nhưng trong thực tế chúng ta không sử dụng tiền xu nữa. Vậy phải làm thế nào?

*Giải quyết:* sử dụng Adapter Design Pattern.

-Client: Khách hàng.

-Target: Tiền mặt.

-Adapter: Quầy mua xu, ở đây khách hàng sẽ sử dụng tiền mặt để mua số lượng xu mong muốn.

-Adaptee: Tiền xu.

**3/ Nguyễn Phước Vinh**

*Đề:* Trong 1 phần mềm vẽ hình chữ nhật nằm ngang, người dùng đưa vào tham số giao điểm 2 đường chéo, chiều dài và chiều rộng để vẽ hình chữ nhật đó. Tuy nhiên, người dùng lại yêu cầu muốn vẽ cả đường tròn ngoại tiếp của hình chữ nhật đó. Vậy bên cạnh cách thông thường là sẽ thêm lớp Circle mới vào chương trình chúng ta sẽ sử dụng Adapter Design Pattern.

*Giải quyết*: sử dụng Adapter Design Pattern.

-Client: người muốn vẽ hình tròn ngoại tiếp.

-Target: Interface của hình chữ nhật.

-Adapter: thực hiện tính bán kính hình tròn dựa trên chiều dài, chiều rộng của Adaptee và phương thức vẽ hình tròn ngoại tiếp theo yêu cầu.

-Adaptee: Interface của đường tròn ngoại tiếp.