**Nhóm 3**

**Danh sách câu hỏi của các nhóm về hai mẫu thiết kế Adapter và Bridge**

Thành viên nhóm:

| STT | MSSV | Họ và tên | Lớp | Vai trò |
| --- | --- | --- | --- | --- |
| **1** | 2014478 | Nguyễn Xuân Phát | CTK44B |  |
| **2** | 2011402 | Dương Mỹ Lộc | CTK44B | ***Trưởng nhóm*** |
| **3** | 2011401 | Trần Bảo Lâm | CTK44A |  |
| **4** | 2014446 | Trần Diệu Đông | CTK44B |  |
| **5** | 2011416 | Phan Minh Nhân | CTK44A |  |
| **6** | 2014508 | Nguyễn Trường Vũ | CTK44B |  |

**Nhóm 1: Giữa Class Adapter và Object Adapter thì khi nào sử dụng loại nào?**

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.

Cụ thể:

* Object adapter

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/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.

* Class adapter (chỉ áp dụng được với các ngôn ngữ hỗ trợ đa kế thừa như C++, Python,...)

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.

=> Việc sử dụng 1 trong 2 loại cấu trúc này sẽ tùy thuộc vào từng dự án với các yêu cầu và chức năng cụ thể.

**Nhóm 2: So sánh 2 design pattern?**

Về Adapter: giữ vai trò làm trung gian giữa 2 lớp, chuyển đổi interface của của một hay nhiều lớp có sẵn sang một interface khác thích hợp. Cho phép các lớp không tương thích có thể tương tác được với nhau mà không cần sửa đổi cấu trúc code.

Về Bridge: Để có thể dễ dàng chỉnh sửa hoặc thay thế mà không làm ảnh hưởng đến những nơi sử dụng lớp khởi tạo ban đầu. Bridge được dùng để:

* Tách các ràng buộc giữa Abstraction và Implementation, để có thể sử dụng độc lập với nhau
* Để tách một lớp có các quan hệ gần gũi với nhau thành hai hệ thống phân cấp lớp riêng biệt là: Abstraction và Implementation

**Nhóm 4: Adapter chỉ có thể bọc được 1 or nhiều Interface???**

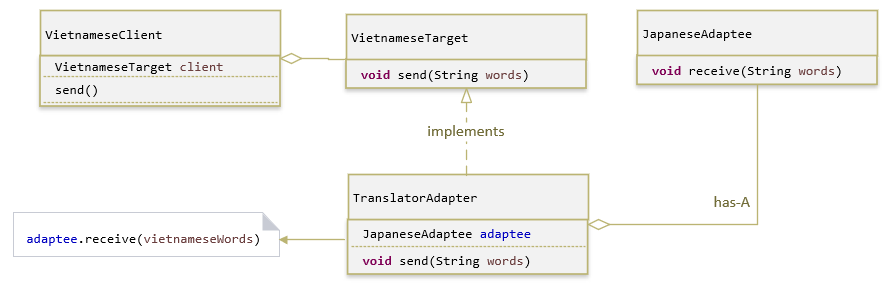
* Được. Chỉ cần truyền các Interface thông qua hàm Constructor() của Adapter.

**Nhóm 5: Adapter pattern có những ứng dụng nào trong thực tế?**

VD: Một người Việt muốn trao đổi với một người Nhật. 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 Nhật.

Chúng ta sẽ mô hình hóa trường hợp này với Adapter Pattern như sau:

* Client: người Việt sẽ là Client trong ví dụ này,vì anh ta cần gửi một số message cho người Nhật.
* 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 Nhật trước khi gửi cho người Nhật.
* Adaptee: đây là interface hoặc class được người Nhật sử dụng để nhận message được chuyển đổi từ thông dịch viên (Translator).

****

**Nhóm 6: Adapter có thể tạo ra nhiều lớp hay không?**

Có thể tạo ra nhiều Adapter trong một dự án. Khi một dự án cần sử dụng nhiều thư viện hoặc các dịch vụ từ bên thứ ba nhưng kiến trúc hiện tại không thể áp dụng một cách trực tiếp thì có thể tạo ra nhiều Adapter để sử dụng các thư viện, dịch vụ đó. Tuy nhiên nếu tạo ra quá nhiều Adapter thì hệ thống sẽ trở nên phức tạp, do đó nếu cần sử dụng nhiều thư viện hoặc dịch vụ bên thứ ba mà hệ thống đã quá cũ, không phù hợp để sử dụng thì nên chỉnh sửa lại hệ thống cho phù hợp.

**Nhóm 7: Giữa 2 design patterns thì loại nào được sử dụng nhiều hơn**

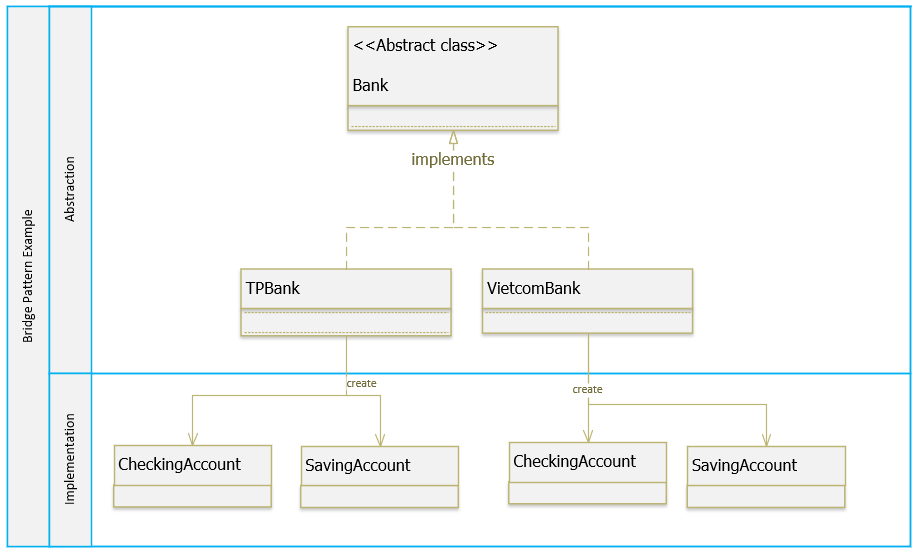
Theo thống kê thì Adapter pattern được sử dụng rộng rãi hơn trong lập trình so với Bridge pattern. Nguyên nhân là do số lượng các thư viện, framework hỗ trợ lập trình, xây dựng các chức năng ngày một nhiều và đa dạng, đồng thời nhiều công nghệ mới xuất hiện. Vì vậy các dự án cũ khi muốn sử dụng các thư viện, framework này mà không muốn thay đổi quá nhiều về mặt cấu trúc của dự án thường sẽ sử dụng mẫu Adapter này để mở rộng, phát triển thêm các chức năng mới.

Nhóm 8: Không đặt câu hỏi

Nhóm 9: Không đặt câu hỏi

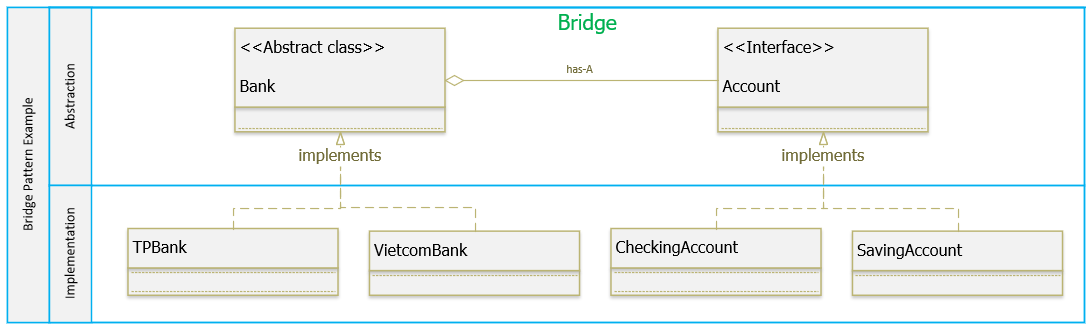
**Nhóm 10: Ngoài ví dụ về hình học thì còn áp dụng gì khác không?**

VD: Một hệ thống ngân hàng cung cấp các loại tài khoản khác nhau cho khách hàng, chẳng hạn: Checking account và Saving account. Chúng ta có sơ đồ như sau



Với cách thiết kế như vậy, khi hệ thống cần cung cấp thêm một loại tài khoản khác, chúng ta phải tạo class mới cho tất cả các ngân hàng, số lượng class tăng lên rất nhiều.

Bây giờ, chúng ta sẽ sử dụng Bridge Pattern để tái cấu trúc lại hệ thống trên như sau:



Với cấu trúc mới như vậy, khi có thêm một loại tài khoản mới, chúng ta đơn chỉ việc thêm vào một implement mới cho Account, các thành phần khác của Bank không bị ảnh hưởng. Hoặc cần thêm một ngân hàng mới, chẳng hạn VietinBank chúng ta chỉ cần thêm implement mới cho Bank, các thành phần khác cũng không bị ảnh hưởng và số lượng class chỉ tăng lên 1.

**Nhóm 11:**

1. **Adapter và Bridge ( so sánh ) ? Sử dụng loại nào?**

Việc sử dụng 1 trong 2 loại pattern này sẽ tùy thuộc vào từng dự án với các yêu cầu và chức năng cụ thể.

1. **Bridge pattern có kế thừa gì từ Adapter trong việc tạo cầu nối giữa các lớp (class) hay không?**

Các mẫu bộ điều hợp và cầu có một số thuộc tính phổ biến. Cả hai thúc đẩy tính linh hoạt bằng cách cung cấp một mức độ gián tiếp cho một đối tượng khác. Cả hai đều liên quan đến các yêu cầu chuyển tiếp đến đối tượng này từ một giao diện khác ngoài chính nó.

Sự khác biệt chính giữa các mẫu này nằm ở ý định của chúng. Bộ điều hợp tập trung vào việc giải quyết sự không tương thích giữa hai giao diện hiện có. Nó không tập trung vào cách các giao diện đó được thực hiện, cũng không xem xét cách chúng có thể phát triển độc lập. Đó là một cách để làm cho hai lớp được thiết kế độc lập hoạt động cùng nhau mà không cần thực hiện lại cái này hay cái kia. Bridge, mặt khác, bắc cầu một sự trừu tượng và các triển khai (có khả năng nhiều) của nó. Nó cung cấp một giao diện ổn định cho khách hàng ngay cả khi nó cho phép bạn thay đổi các lớp thực hiện nó. Nó cũng phù hợp với các triển khai mới khi hệ thống phát triển.

Do những khác biệt này, bộ điều hợp và cầu thường được sử dụng tại các điểm khác nhau trong vòng đời phần mềm. Một bộ điều hợp thường trở nên cần thiết khi bạn phát hiện ra rằng hai lớp không tương thích nên hoạt động cùng nhau, nói chung để tránh sao chép mã. Các khớp nối là không lường trước được. Ngược lại, người dùng cây cầu hiểu về phía trước rằng sự trừu tượng phải có một số triển khai và cả hai có thể phát triển độc lập. Mẫu bộ điều hợp làm cho mọi thứ hoạt động sau khi chúng được thiết kế; Cầu làm cho họ làm việc trước khi họ. Điều đó không có nghĩa là bộ điều hợp bằng cách nào đó kém hơn cây cầu; Mỗi mẫu chỉ đơn thuần giải quyết một vấn đề khác nhau.

**Nhóm 12:**

1. **Có những loại Adapter nào? Mỗi loại đó được sử dụng trong trường hợp nào?**

Có 2 cách để triển khai Design Pattern Adapter là Object / Class Adapter

| 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. |
| --- | --- |

1. **Cần lưu ý gì khi sử dụng Adapter pattern để đảm bảo tính linh hoạt và hiệu quả của hệ thống?**

\*\*\* Bên cạnh những lợi ích của Design Adapter Pattern thì chúng ta cần lưu ý:

Tất cả các yêu cầu đều cần được chuyển tiếp, do đó sẽ làm tăng thêm một ít chi phí. Vì vậy cần sử dụng Adapter phù hợp không nên lạm dụng

Nhóm 13: Không đặt câu hỏi