# Thực hành tuần 07

## Nội dung:

* Thiết kế đối tượng theo tư tưởng Design pattern.
* Áp dụng Bridge pattern vào demo cấu trúc đơn gian quản lý của một hãng hàng không.

## Yêu cầu thực hành:

**Sau một thời gian vận hành của hãng hàng không xyz, thì hệ thống phình to. Dẫn đến mô hình CSDL cũ không còn đáp ứng được yêu cầu thực tế rong quản lý đặt chỗ và quản lý nhân viên.**

|  |  |
| --- | --- |
|  | *Căn cứ vào cấu trúc cơ sở dữ liệu đã cung cấp (cũ). Hãy xây dựng kiến trúc mới ở mức demo cho các module để trình diễn trong buổi đấu thầu:*  **(Code trên lớp)**   * Thiết kế các class ở mức demo cho nhiệm vụ quản lý nhân viên cho mỗi loại nhân (tiếp viên; phi công; an ninh; kĩ thuật mặt đất; không lưu …) theo mẫu thiết kế Pattern – Bridge. * Mỗi nhân viên sẽ có hai thao tác là xác nhận nhận nhiệm vụ và xác nhận hoàn thành nhiệm vụ. |

Hướng dẫn:

## Tổng quan về Design Bridge Pattern:

Bridge Pattern là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern). Mục đích của nó là tách những class có tính trừu tượng (abstraction) ra khỏi các class có tính hiện thực (implementation) của nó. Từ đó có thể dễ dàng chỉnh sửa hoặc thay thế mà không làm ảnh hưởng đến những nơi có gọi/kế thừa từ class ban đầu.

## Khi nào sử dụng Design Bridge Pattern:

Áp dụng trước khi phát triển hệ thống.

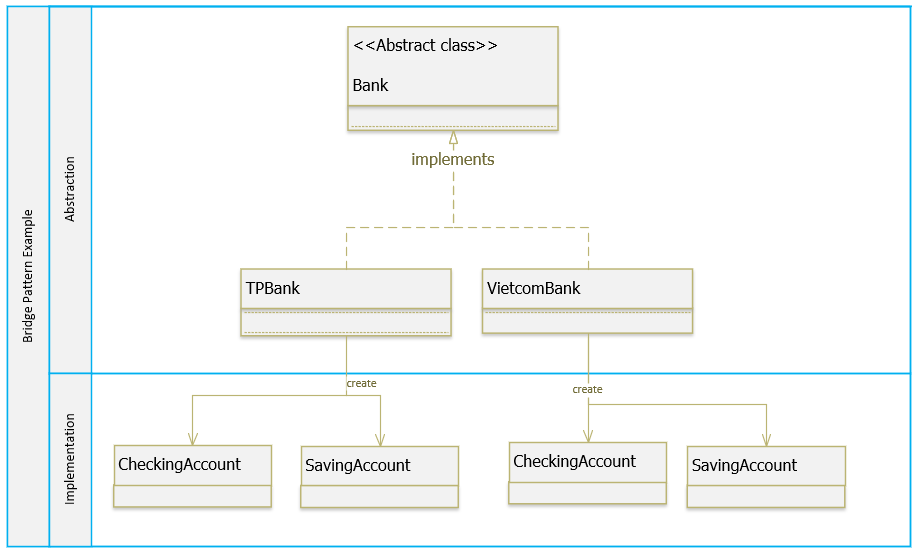
Những module nào có cấu trúc đa hình.

## Các class thành viên trong Design Bridge Pattern:

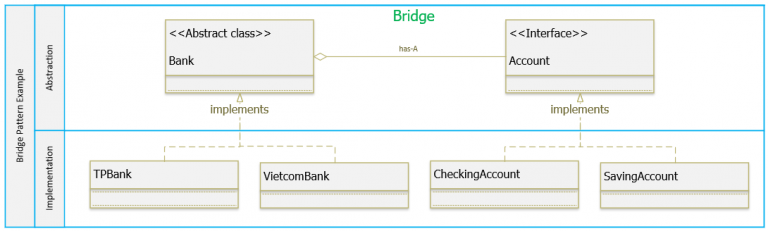
* **Client:** đại diện cho user sử dụng các chức năng thông qua *Abstraction.*
* **Abstraction** : định ra một abstract interface quản lý việc tham chiếu đến đối tượng hiện thực cụ thể (*Implementor*).
* **Refined Abstraction (AbstractionImpl)** : hiện thực (*implement*) các phương thức đã được định nghĩa ra trong Abstraction bằng cách sử dụng một tham chiếu đến một đối tượng của Implementer.
* **Implementor :** định nghĩa các interface cho các lớp hiện thực.
* **ConcreteImplementor** : hiện thực *Implementor interface*.

**Ví dụ:**

Một hệ thống ngân hàng cung cấp các loại tài khoản khác nhau cho khách hàng: *Checking account* và *Saving account*:



Với cách thiết kế trên, thì hệ thống sẽ phình to khi có thêm ngân hàng hoặc loại tài khoản => số lượng class tăng lên rất nhiều. Mà ảnh hưởng đến các phương thức trong class abstrac. Cải tiến với Bridge Pattern, kiến trúc sẽ thay đổi như sau:



Ưu điểm với cấu trúc mới thì khi có thêm một loại tài khoản mới, dev 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, ví dụ như VietinBank; Dev chỉ cần thêm implement mới cho Bank, các thành phần khác không bị ảnh hưởng, mà số lượng class chỉ tăng lên 1.

## Đôi nét về Pactory pattern:

### Mục đích:

* Giúp chúng ta tạo đối tượng mà không để lộ logic tạo đối tượng ở phía người dùng và tham chiếu đến đối tượng mới được tạo ra bằng cách sử dụng một interface chung.
* Tăng tính module hóa cho hệ thống.
* Mở rộng code dễ dàng hơn (khi cần mở rộng, chỉ việc tạo ra sub class và implement thêm vào factory method)
* Quản lý vòng đời của các đối tượng thông qua Class Factory Pattern
* Thống nhất về quy tắc đặt tên(**naming convention**) trong làm việc team.

### Trường hợp dụng pattern:

* Có quá nhiều class đối tượng thuộc cùng một nhóm nhiệm vụ. Mà tại một thời điểm cụ thể ta cần trả về một class đối tượng cụ thể dựa vào input.
* Khi mở rộng hệ thống, mà không thể xác định được sẽ cần cài đặt thêm bao nhiêu lớp con.

**Ex:**

Nếu cần mua xe máy. Thì rõ ràng việc chúng ta đến từng hãng (Sym; Honda; Yamha …) để xem “xe” thì bất tiện hơn việc đến một cửa hàng kinh doanh ủy quyền bán buôn của tất cả các hãng.

**Cài đặt đối tượng:**

Hàm main()???