**Phần I , Cấu trúc dữ liệu và giải thuật**

**Bài 1:** Cho 2 mảng đã được sắp xếp. Tìm giao của hai mảng với độ phức tạp là o(mlog(n)) với m và n tương ứng là độ dài của 2 mảng

Kiến thức cần tìm hiểu:

- Độ phức tạp thuật toán

- Tìm kiếm nhị phân

**Bài giải**

**Độ phức tạp thuật toán**:

- Tính toán số phép tính cần thực hiện với input đầu vào là n.

- Độ phức tạp thấp nhất là O(1) (các phép gán, cộng trừ nhân chia, nhập xuất).

- Khi thực hiện 1 vòng for, nếu trong vòng for thực hiện một phép toán với độ phức tạp O(1) thì độ phức tạp của vòng for là O(n).

- Với trường hợp có độ phức tạp là O căn n (ví dụ duyệt từ 2 đến sqrt của n) thì ngta đều gọi chung là O(logN).

- 2 vòng for lồng nhau độ phức tạp là O(m x n).

- O(n ^ 3) rất là tệ

- Ví dụ trong chương trình có 2 đoạn code có độ phức tạp khác nhau, tk A là O(n^2), tk B là O(n) thì độ phức tạp của cả chương trình sẽ lấy theo tk lớn nhất.

- Một số trường hợp đặc biệt: O(2 ^ n) , O(n!)

**Tìm kếm nhị phân**

+ Sử dụng với mảng đã sắp xếp. Nó sẽ so sánh giá trị cần tìm với phần tử giữa mảng, nếu giá trị nhỏ hơn thì sẽ tiếp tục lặp lại bước trên trong nửa khoảng đầu, hoặc ngược lại, lặp như thế cho đến khi tìm ra giá trị.

+ Độ phức tạp O(log n).

**Bài toán tìm giao của hai mảng với độ phức tạp là o(mlog(n))**

**\* Cách thực hiện**

+, Ta sử dụng thuật toán tìm kiếm nhị phân để tìm kiếm một phần tử x có nằng trong mảng B hay không với độ phức tạp là O(log n)

+, Ta sẽ phải thực hiện tìm kiếm cho tất cả các phần tử trong mảng nên độ phức tạp sẽ là O(m(log n)

+, Kết quả của bài toán sẽ là những phần tử của mảng A xuất hiện trong mảng B

**Bài 2**: Một url là url thỏa mãn các yếu tố

 (ví dụ 1 url hợp lệ: https://tiki.vn/dien-thoai-may-tinh-bang/c1789?src=mega-menu):

- Bắt đầu bằng http hoặc https

- Có thể chứa www hoặc không

- Tên miền của url (ví dụ tiki)  chỉ chứa các ký tự la tinh hoa/thường từ a-z và chữ số.

- Phần domain extension (ví dụ .vn) phải chứa dấu chấm ở đầu, thêm sau là các ký tự in thường có độ dài từ 2 đến 6 ký tự.

- Phần path (ví dụ /dien-thoai-may-tinh-bang/c1789?src=mega-menu) không được chứa dấu cách.

Viết đoạn regex để kiểm tra xem một url bất kỳ có là hợp lệ hay không.

**Bài 3:** Tìm vị trí đầu tiên và cuối cùng của một thành phần trong mảng đã sắp xếp

input: một mảng đã được sắp xếp tăng dần và một số

output: vị trí đầu tiên và cuối cùng của số đã cho

nếu không tồn tại thì trả về -1

Ví dụ:

Input: nums = [5,6,7,8,8,15], target = 8

Output: [3,4]

Yêu cầu độ phức tạp: O(log n)

**Bài 4:** Tìm hiểu về Prefix Sum và ứng dụng trong bài toán counting sort

Prefix Sum là một kỹ thuật hữu ích trong việc xử lý các bài toán liên quan đến mảng. Nó giúp tính tổng của bất kỳ dãy con nào của mảng một cách hiệu quả. Ý tưởng cơ bản là tính trước các tổng của các phần tử từ đầu mảng đến một vị trí nhất định, sau đó sử dụng kết quả này để tính tổng của bất kỳ dãy con nào một cách nhanh chóng.

**Cách tính Prefix Sum**

Giả sử ta có một mảng A với n phần tử. Mảng prefixSum được tính như sau:

· prefixSum[0] = A[0]

· prefixSum[i] = prefixSum[i-1] + A[i] với 1 <= i < n

Với mảng prefixSum này, ta có thể tính tổng của dãy con A[l..r] trong O(1) thời gian:

· sum(A[l..r]) = prefixSum[r] - prefixSum[l-1] (nếu l > 0)

· sum(A[0..r]) = prefixSum[r] (nếu l = 0)

**Ứng dụng của Prefix Sum trong Counting Sort**

Counting Sort là một thuật toán sắp xếp không dựa trên so sánh. Thay vào đó, nó sử dụng một mảng đếm để lưu trữ số lần xuất hiện của mỗi giá trị. Thuật toán này hiệu quả khi sắp xếp các số nguyên trong một phạm vi giá trị nhỏ.

**Các bước của Counting Sort**

1. Tìm giá trị lớn nhất và nhỏ nhất trong mảng: Điều này giúp xác định kích thước của mảng đếm count.
2. Khởi tạo mảng đếm count: Mảng này có kích thước bằng giá trị lớn nhất trừ giá trị nhỏ nhất cộng một.
3. Đếm số lần xuất hiện: Duyệt qua mảng đầu vào và tăng giá trị tương ứng trong mảng count.
4. Tính Prefix Sum trên mảng đếm: Đây là bước sử dụng kỹ thuật Prefix Sum. Mảng count sau bước này sẽ cho biết vị trí cuối cùng của mỗi giá trị trong mảng đã sắp xếp.
5. Sắp xếp mảng: Duyệt ngược qua mảng đầu vào và sử dụng mảng count để đặt các giá trị vào vị trí đúng trong mảng kết quả.

**Bài 5:** Cho một chuỗi s và một từ điển dict. Hãy viết một phương thức add các khoảng trắng vào chuỗi s sao cho thành các câu có thể.

Input:

s = "catsanddog"

wordDict = ["cat", "cats", "and", "sand", "dog"]

Output:

[

  "cats and dog",

  "cat sand dog"

]

**Phần 2: Design Pattern**

**Bài 1: Singleton**

**- Đặt vấn đề:**

Khi chúng ta cần giải quết một trong hai vấn đề :

- Đảm bảo một lớp chỉ có một thể hiện duy nhất

- Cung cấp một điểm trủy cập toàn cục cho phiên bản đó

=> sử dụng singleton pattern sẽ là một phương pháp rất tốt

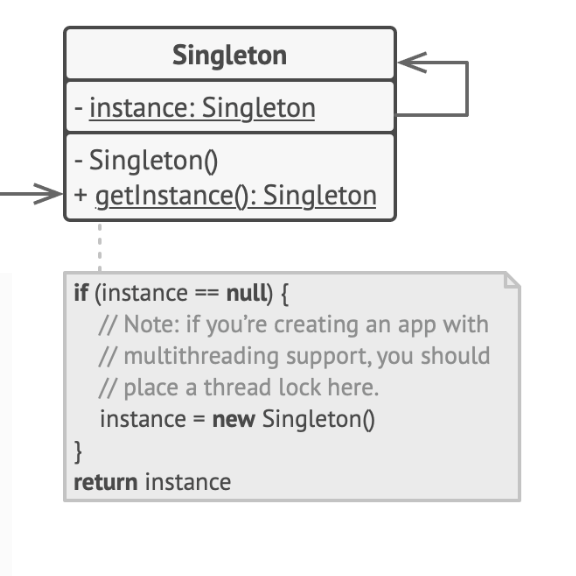
**- Khái niệm**

Singleton là một mẫu thiết kế (design pattern) trong lập trình phần mềm, nó cho phép chỉ tạo ra một instance duy nhất của một lớp và cung cấp một điểm truy cập toàn cục để truy cập vào instance này.

**- Cấu trúc :**

+, Class singleton khai báo một phương thức static getInstance() trả về cùng một instance của chính class đó

+, Hàm tạo của class singleton phải được khai báo private và cách duy nhất để khởi tạo class là gọi đến phương thức getInstance



**- Lợi ích**

+ Đảm bảo rằng chỉ có một instance duy nhất của lớp được tạo ra.

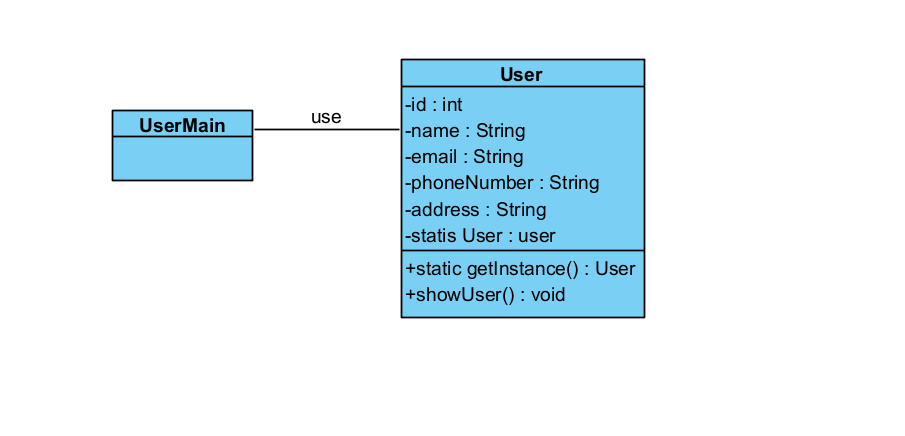
+ Cung cấp một điểm truy cập toàn cục cho instance duy nhất, giúp truy cập dễ dàng từ bất kỳ nơi nào trong ứng dụng.

+ Kiểm soát quá trình tạo instance, đảm bảo rằng chỉ có một instance duy nhất được tạo và sử dụng.

**Ví dụ minh họa**

Bài toán : Chúng ta muốn tạo ra một phiên bản cho một user đang sử dụng hệ thống và sử dụng thông tin đó để hiển thị ở những nơi cần sử dụng trong hệ thống

=> Ta sử dụng design pattern singleton cho bài toán trên

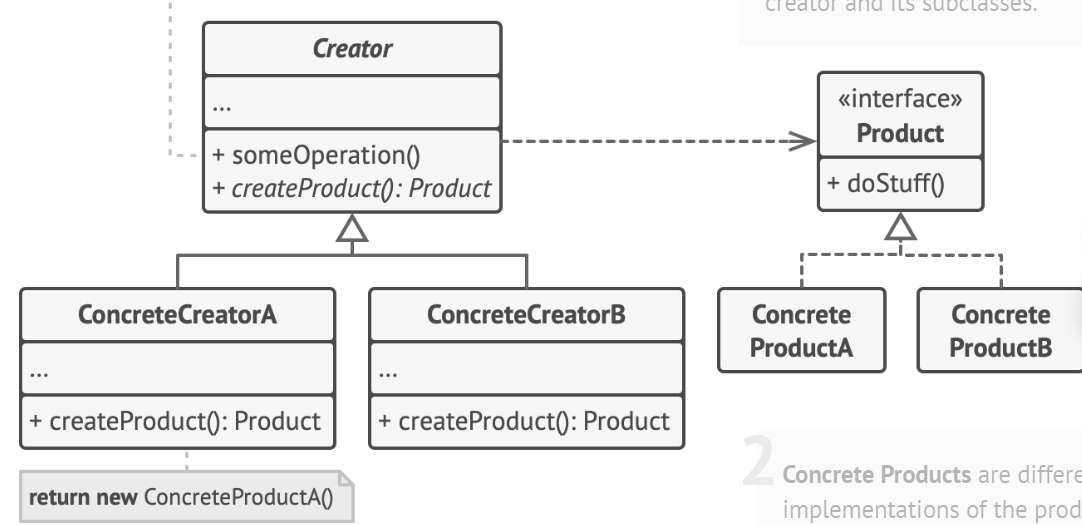


**Bài 2: Factory Pattern**

**- Khái niệm**

Mẫu thiết kế Factory (Factory Pattern) là một mẫu thiết kế creational trong lập trình hướng đối tượng. Nó cung cấp một cách để khởi tạo các đối tượng mà không cần chỉ định chính xác lớp của đối tượng sẽ được tạo.Thay vì client trực tiếp gọi constructor để tạo đối tượng, nó yêu cầu một đối tượng factory để tạo đối tượng thay mặt.

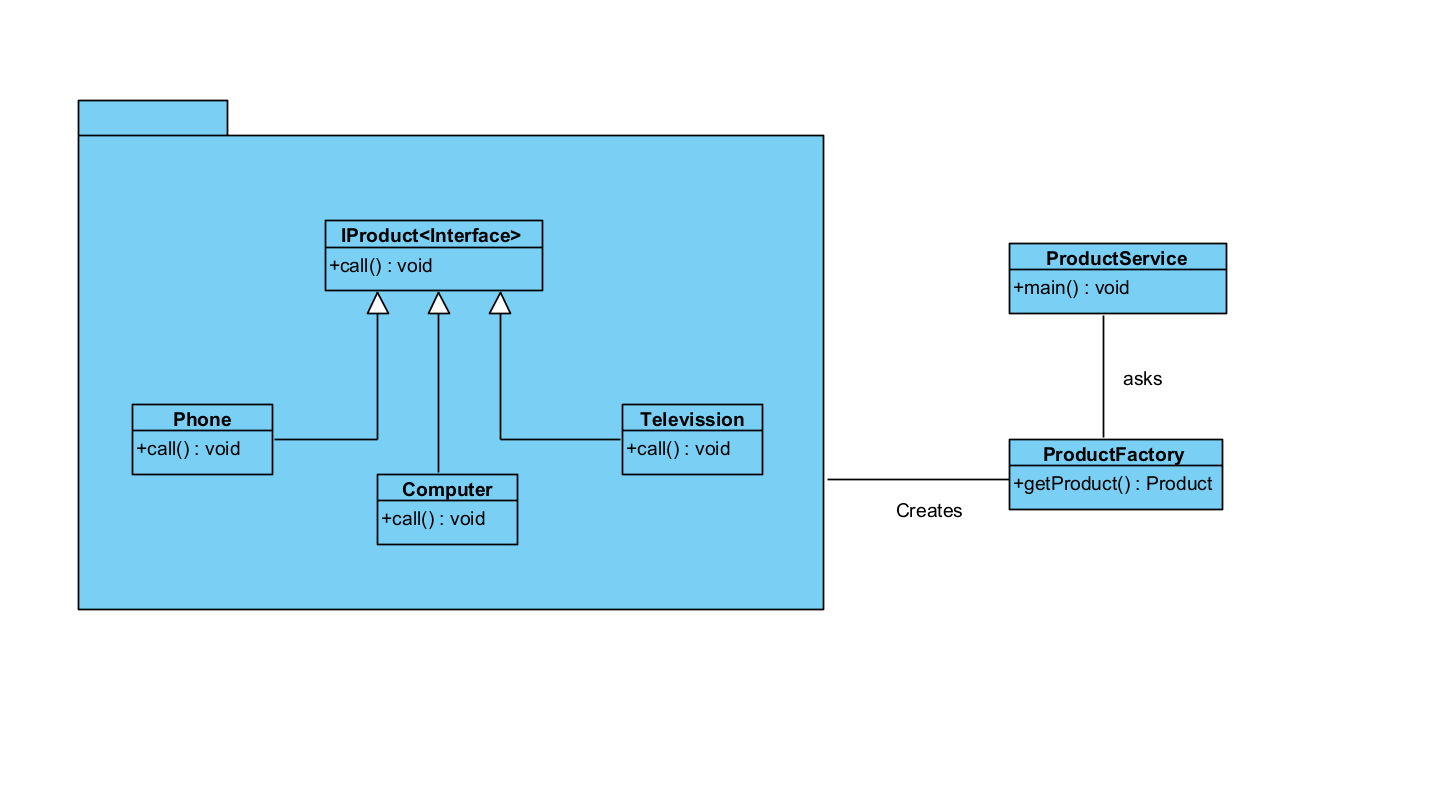
**- Cấu trúc :**



* product khai báo interface chung cho tất cả các đối tượng có thể được tạo ra bởi người tạo và các lớp con của nó.
* product cụ thể là các triển khai khác nhau của inteface product.  
   Lớp Creator khai báo phương thức factory để trả về các đối tượng product mới
* Concrete Creators ghi đè phương thức factory để nó trả về một loại product khác.

**Ví dụ :** Ta có nhiều loại sản phẩm như Phone, Computer, Televission… cần viết các class và khởi tạo các class trên sử dụng Factory Pattern

+, Class diagram mô tả cách hoạt động



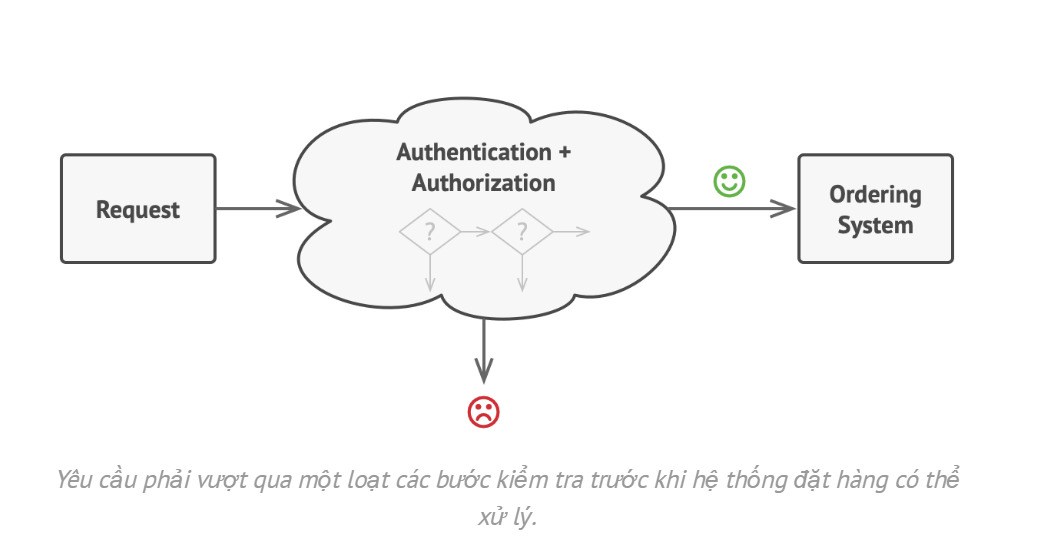
**Bài 3: Chain of Responsibility**

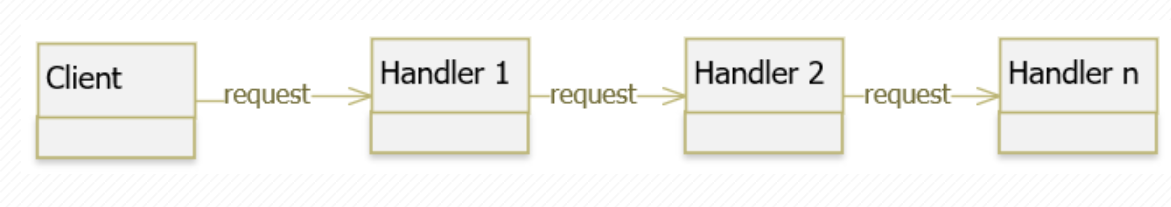
**- Đặt vấn đề :**

Trong một hệ thống Đặt hàng trực tuyến thì chúng ta muốn hạn chế quyền truy cập và chỉ những người dùng vượt qua tất cả các bước kiểm tra thì mới có quyền đặt hàng. Sau một hồi lập kế hoạch chúng ta thấy các bước này phải thực hiện một cách tuần tự. Trong trường hợp các bước kiểm tra đó tăng theo thời gian sẽ khiến cho code của chúng ta bị dối và rất khó để bảo trì một giải pháp tối ưu cho bài toán này là sử dụng Chain of Responsibility Pattern

**- Mô tả**

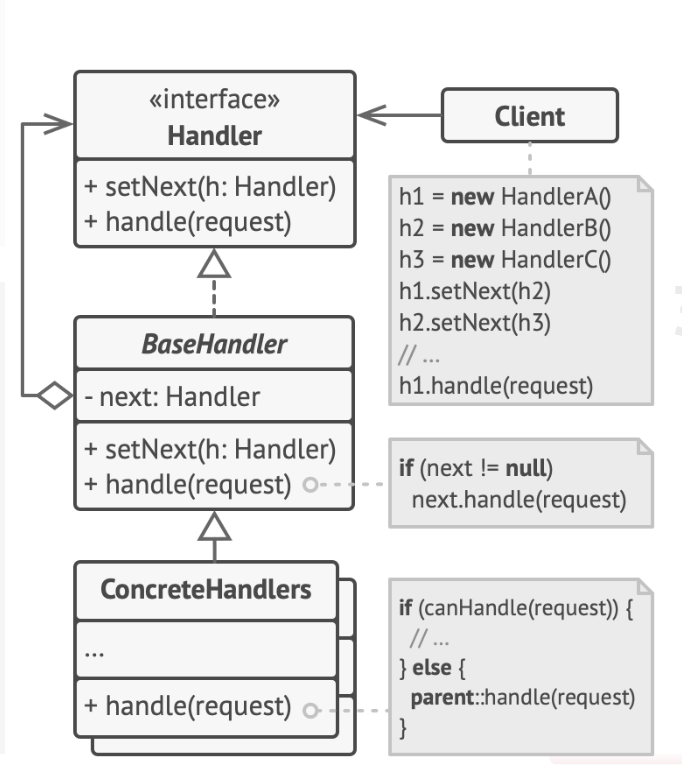
Mẫu thiết kế Chain of Responsibility là một mẫu thiết kế hành vi được sử dụng để xử lý một yêu cầu thông qua một chuỗi các bộ xử lý. Khi nhận được yêu cầu mỗi bộ xử lý có thể xử lý yêu cầu hoặc chuyển nó cho bộ xử lý tiếp theo trong chuỗi.





Chain of Responsibility Pattern hoạt động như một danh sách liên kết ([Linked list](https://gpcoder.com/2555-linkedlist-trong-java/)) với việc [đệ quy](https://gpcoder.com/2327-giai-thuat-de-quy-trong-java/) duyệt qua các phần tử (recursive traversal).

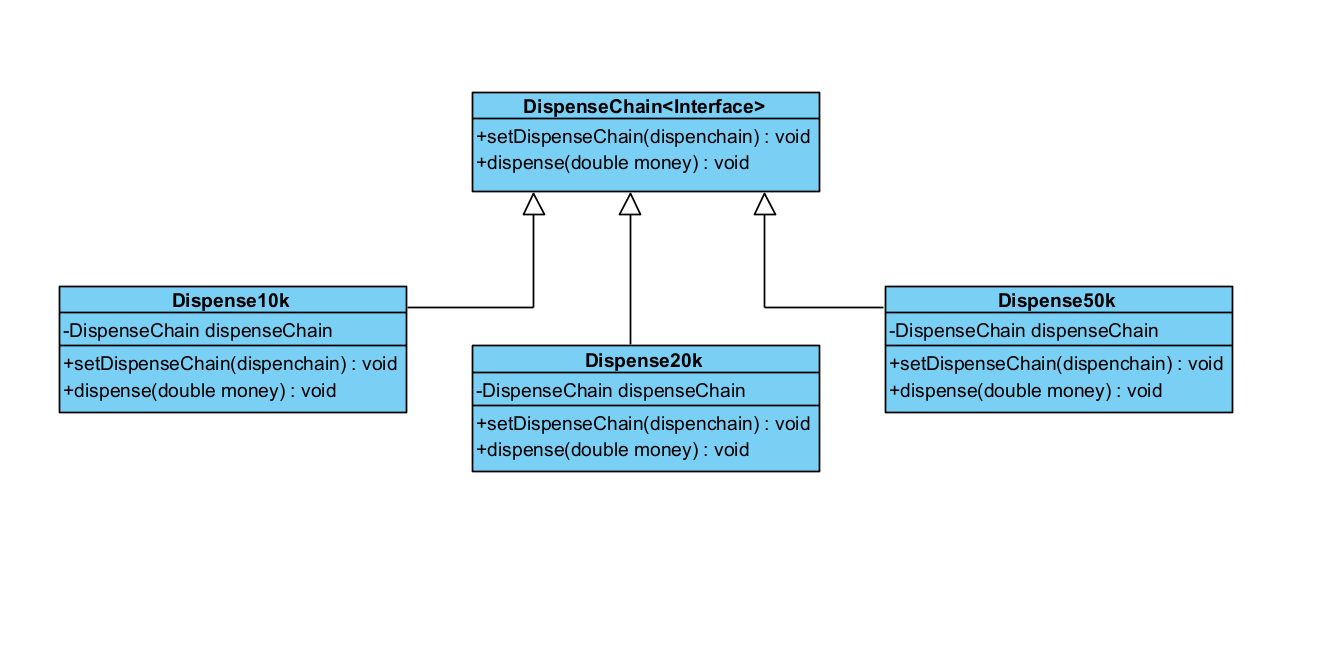
**- Các thành phần trong Chain Of Responsibility**



* Handler : định nghĩa một interface để xử lý yêu cầu
* BaseHandler : Là một lớp tùy chọn ta có thể đặt những đoạn code chung cho tất cả các lớp trong chuỗi xử lý
* ConcreteHandlers : Chứa code thực tế để xử lý các yêu cầu . Khi xử lý yêu cầu thì mỗi trình xử lý phải quyết định xem có cần xử lý cái request đó không hay chuyển nó sang trình xử lý khác
* Client : Tạo ra các yêu cầu và gửi nó đến các đối tượng tiếp nhận

**Ví dụ:** Bài toán rút tiền ở ATM . chúng ta nhập một số tiền cần rút và loại tiền tệ có trong Atm ở các mệnh giá khác nhau như: 10.000, 20.000, 50.000. Hãy in ra hóa đơn rút tiền gồm số lượng các mệnh giá tiền lần lượt đã được rút nếu số tiền không là bội số của 10 thì in ra không hợp lệ.

- Ở bài toán này chúng ta hoàn toàn có thể triển khai một cách rất dễ dàng theo cách thông thường nhưng trong ví dụ này ta sẽ sử dụng cách tiếp cận bằng Chain Of Responsible Pattern



**Bài 4: Adapter Pattern**

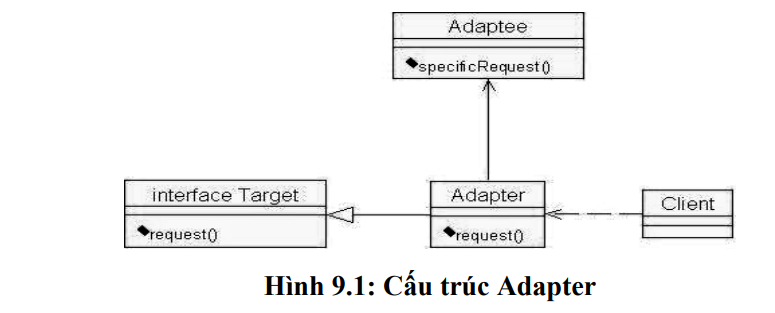
- **Khái niệm**

+ Adapter design pattern hoạt động như một cầu kết nối gữi hai interface hoặc class không tương thích mà nó không thể kết nối trực tiếp

+ Adapter design pattern thường được triển khai ngay sau khi mã ban đầu được viết để kết nối các thành phần không tương thích

+ Adapter pattern thuộc loại structual pattern

**- Cấu trúc**

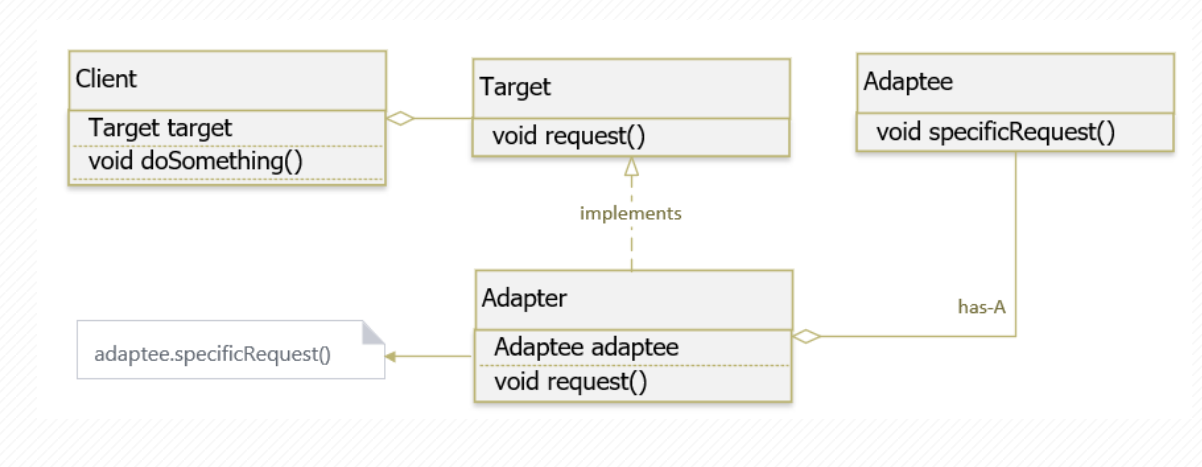


* Target là một Interface xác định các chức năng, yêu cầu mà Client cần sử dụng
* Adaptee là lớp chứa các chức năng mà Target cần sử dụng nhằm tạo ra các chứcnăng mà Target cần cung cấp cho Client.
* Adapter cài đặt phương thức từ Target và sử dụng đối tượng lớp Adaptee, Apdater có nhiệm vụ gắn kết Adaptee vào Target để có được chức năng mà Client mong muốn.

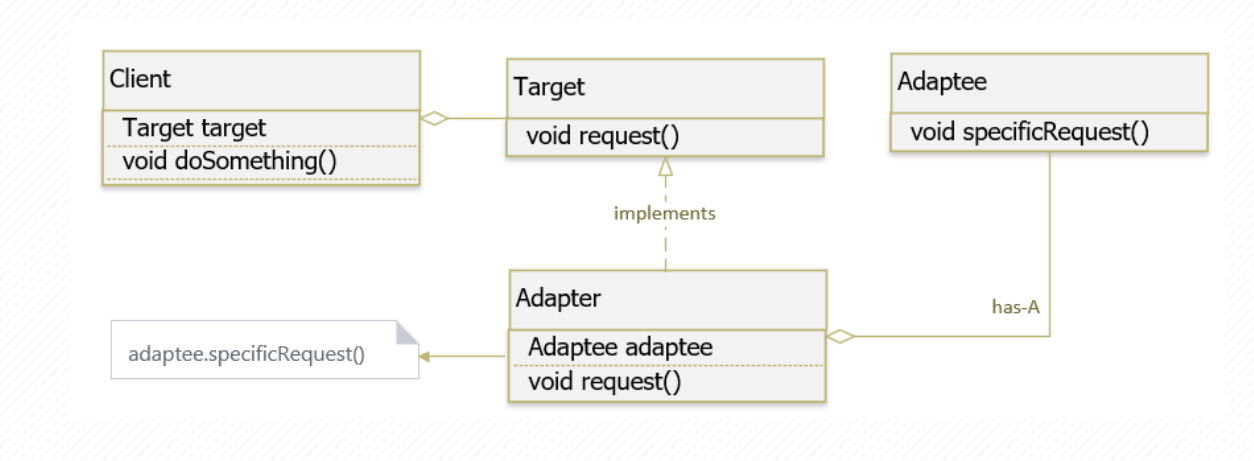
**- Có hai loại Adapter**

**+ Object Adapter - Composition :**

Trong mô hình này một lớp Adapter sẽ tham chiếu đến một hoặc nhiều đối tượng của lớp có sẵn với interface không tương thích adaptee , Adapter sẽ cài đặ lại interface Target

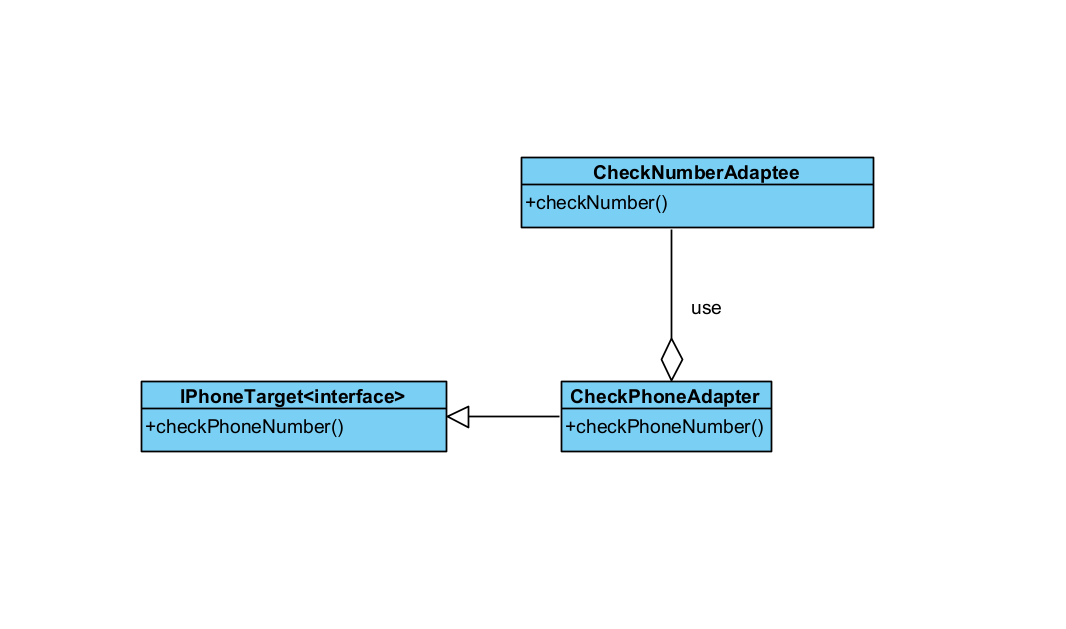


**+ Class Adapter (inheritance)** : trong mô hình này lớp Adapter sẽ kế thừa lớp có sẵn của interface không tương thích Adaptee đồng thời tiến hành cài đặt interface target



**Ví dụ** : Giả sử chúng ta có một lớp CheckNumberAdaptee thực hiện kiểm tra chuỗi ký tự có phải là kiểu số không. Chúng ta muốn kiểm tra xem một chuỗi có phải là số điện thoại hay không chuỗi phải là số có 10 kí tự và bắt đầu bằng (84|0[3|5|7|8|9]) bằng cách sử dụng CheckNumberAdaptee. Để làm điều này, Chúng ta sẽ sử dụng một lớp CheckPhoneAdapter để triển khai interface IPhoneTarget và sử dụng CheckNumberAdaptee để thực hiện kiểm tra số điện thoại.

**+ Object Adapter - composition**



**+ Class Adapter - inheritance**

