**Tuần 2**

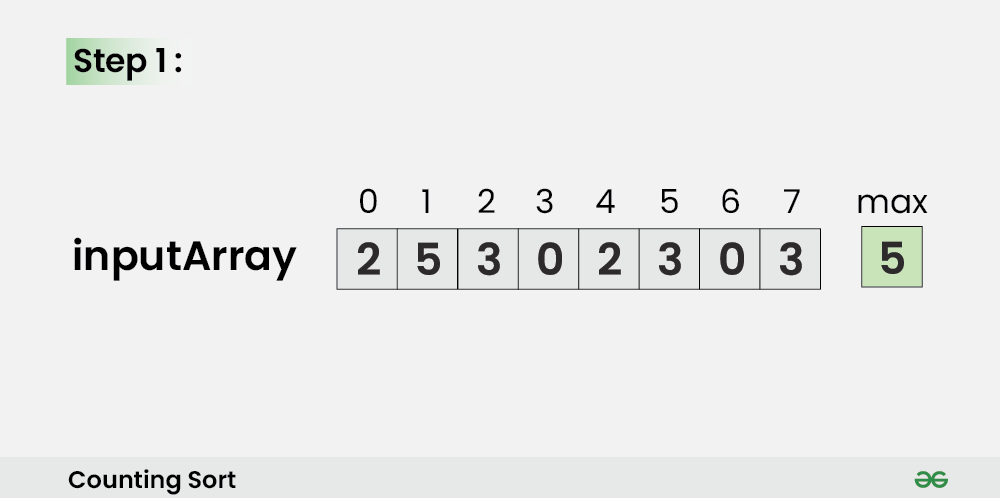
***Bài 1: Tìm hiểu về Prefix Sum và ứng dụng trong bài toán Counting Sort.***

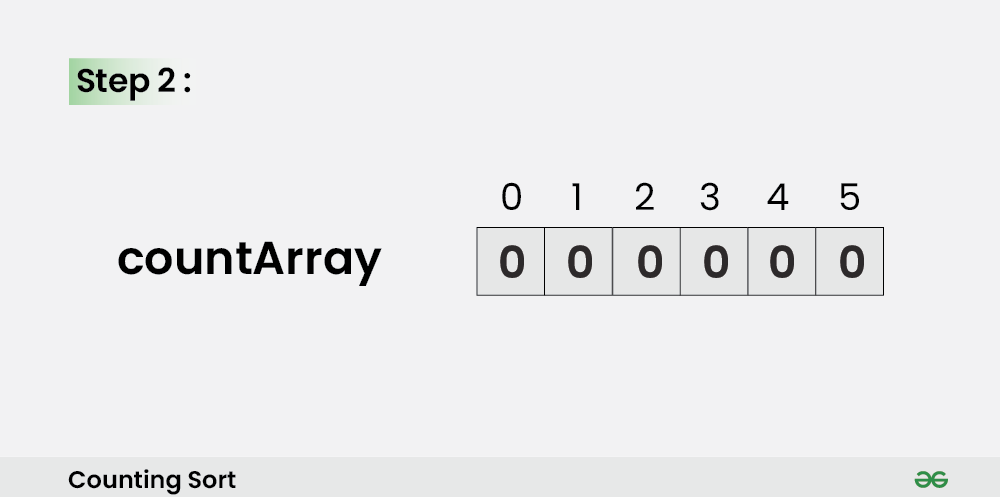
* Prefix sum:
  + Mảng cộng dồn (tổng tiền tố): là một cách sử dụng mảng linh hoạt để tính toán giá trị từ vị trí gốc đến vị trí yêu cầu.
  + Bài toán: giả sử ta có một mảng arr với kích thước N, ta muốn tính giá trị: arr[a] + arr[a+1] + … + arr[b] với a, b thỏa mãn . Chúng ta xét ví dụ với N = 6:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index i | 1 | 2 | 3 | 4 | 5 | 6 |
| arr[i] | 1 | 6 | 4 | 2 | 5 | 3 |

* + Vấn đề: với bài toán trên, tính tổng giá trị của mảng từ a đến b chỉ là một yêu cầu với độ phức tạp thuật toán là O(N) nếu chúng ta xét duyệt tuần tự mảng. Tuy nhiên, trong nhiều trường hợp, bài toán chúng ta có không chỉ có 1 yêu cầu, khi vào trường hợp tệ nhất, số yêu cầu là n, lúc này độ phức tạp thuật toán có thể lên tới O(N2), rất có thể xảy ra lỗi “exceed the time limit”.
  + Thay vào đó, chúng ta có thể sử dung Prefix sum để tính toán các yêu cầu tính tổng các phần tử của mảng như trên. Để triển khai, ta tạo mảng prefix[] với N + 1 phần tử và các phần tử có giá trị ban đầu là 0.
  + Sau đó, ta tính giá trị của mảng prefix bằng công thức sau:
  + Về cơ bản, công thức trên có nghĩa là tại mỗi vị trí k, mảng prefix sẽ lưu giá trị tổng của tất cả các phần tử trong mảng arr[] ban đầu từ 1 đến k. Khi đó, độ phức tạp thuật toán chỉ còn O(N):
  + Ta được kết quả như sau:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Index i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| prefix[i] | 0 | 1 | 7 | 11 | 13 | 18 | 21 |

* + Giờ đây, để tính tổng các phần từ từ a đến b của mảng arr[], ta chỉ cần áp dụng công thức sau:
  + Hay nói cách khác:
  + Với cách làm trên, độ phức tạp thuật toán để tính tổng các phần tử trong mảng arr[] khi đã có mảng prefix[] chỉ là O(1), với N các yêu cầu tính tổng, ta có độ phức tạp là O(N), kết hợp với độ phức tạp khi tạo ra mảng prefix[] là O(N), ta vẫn sẽ có được độ phức tạp chung của cả bài là O(N).
* Bài toán Counting Sort
  + Ý tưởng: sử dụng prefix sum để tính toán vị trí được sắp xếp của các phần tử trong mảng đã cho.
  + Các bước hoạt động:
    - Bước 1: Tìm giá trị lớn nhất (max) của mảng đã cho.
    - Bước 2: Tạo mảng đếm số lần xuất hiện của các phần tử với độ dài max + 1 với tất cả các phần tử bằng 0.



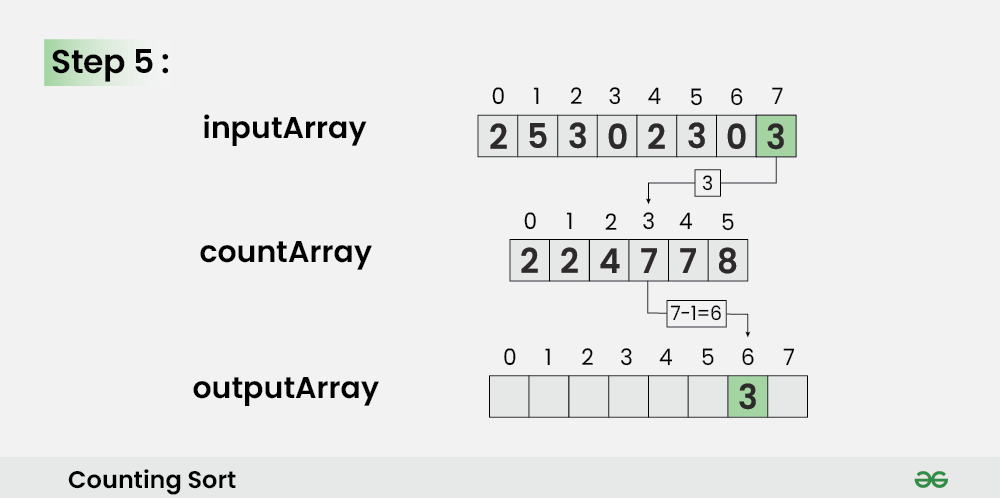
* + - Bước 3: Đếm số lần xuất hiện của từng phần tử trong mảng input vào lưu vào mảng count.



* + - Bước 4: Sử dụng prefix sum để tính tổng các phần tử trước đó.



* + - Bước 5: Xét duyệt mảng input từ cuối mảng lên đầu mảng. Với giá trị của phần tử đang xét sẽ là chỉ số index của mảng count, lấy giá trị tại phần tử index của mảng count trừ đi 1, ta sẽ được vị trí index của mảng output tương ứng. Lần lượt cho đến khi ta được mảng ouput hoàn chỉnh.



* + Triển khai:
    - Dựa theo các bước hoạt động của counting sort, ta triển khai hàm như sau:

A computer screen shot of a program

Description automatically generated

* + - Hàm trả về mảng result tương ứng với mảng output.
    - Bước 1: tìm giá trị lớn nhất

A black screen with white text

Description automatically generated

* + - Bước 2: tạo mảng count



* + - Bước 3: đếm số lần xuất hiện

A black background with white text

Description automatically generated

* + - Bước 4: tính prefix sum

A black background with white text

Description automatically generated

* + - Bước 5: hoàn chỉnh mảng result

A black screen with white text

Description automatically generated

* + Kết quả:
    - Với mảng đầu vào là [2, 5, 3, 0, 2, 3, 0, 3], ta có kết quả như sau:

A computer screen with text

Description automatically generated

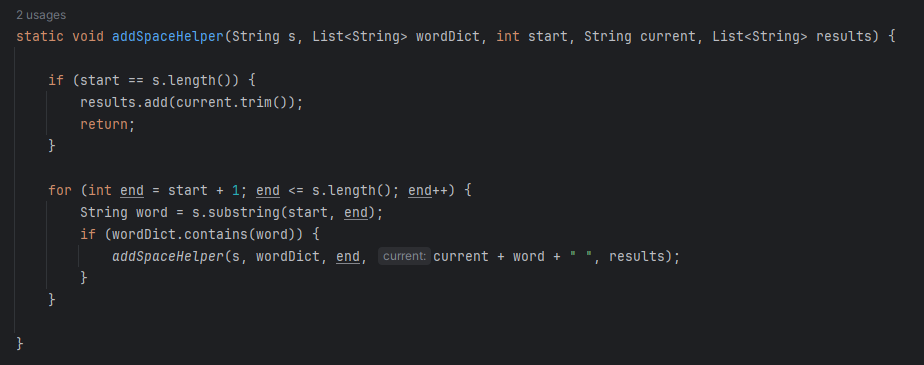
A screen shot of a computer

Description automatically generated

***Bài 2: Cho một chuỗi và một từ điển dict. Hãy viết một phương thức add các khoảng trắng vào chuỗi s sao cho thành các câu có thể.***

A group of black text

Description automatically generated

* Ý tưởng:
  + Sử dụng phương pháp đệ quy: tách chuỗi s thành từng chuỗi nhỏ, nếu có chuỗi nào hợp lệ, xét tiếp chuỗi đằng sau tương tự.
* Triển khai:
  + Viết hàm đệ quy với các tham số đầu vào là: chuỗi cần xét, từ điển, điểm bắt đầu, chuỗi hiện tại, kết quả. Điểm neo của hàm là khi điểm bắt đầu trùng với điểm cuối cùng của chuỗi cần xét.
  + Sử dụng cắt chuỗi lần lượt từ điểm bắt đầu đến khi hết chiều dài của chuỗi cần xét.
  + Code:
  + Viết hàm addSpace với kiểu trả về là List<String>, dùng để khởi tạo các tham số truyền vào. A screen shot of a computer program

    Description automatically generated
* Kết quả:
  + Với input như đề bài, ta có kết quả như sau:

A computer screen shot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

***Bài 3: Tìm hiểu về một số design pattern. Mô tả và implement lại chúng (phân tích so sánh dựa trên 1 case bài toán thực tế)***

* ***Singleton***
* ***Factory***
* ***Chain of Responsibility***
* ***Adapter Pattern***
* **Singleton Pattern**
  + Khái niệm:
    - Là 1 trong 5 design pattern của nhóm Creational Design Pattern: Factory Method, Abstract Factory, Builder, Prototype, Singleton.
    - Đảm bảo chỉ duy nhất 1 thể hiện (instance) được tạo ra và nó sẽ cung cấp một method để có thể truy xuất được instance đó mọi lúc mọi nơi trong chương trình.
  + Nguyên tắc triển khai (Implement):
    - private constructor để hạn chế truy cập từ class bên ngoài.
    - Đặt private static final variable đảm bảo biến chỉ được khởi tạo trong class.
    - Có một method public static để return instance được khởi tạo ở trên.
  + Các cách implement:
    - A computer screen shot of a program code

      Description automatically generatedEager initialization: khởi tạo ngay khi được gọi đến.
      * Ưu điểm: cài đặt nhanh, dễ dàng.
      * Nhược điểm: instance được khởi tạo có thể không dùng đến.
    - A screen shot of a computer program

      Description automatically generatedStatic block initialization: đưa khởi tạo vào khối static để xử lý các ngoại lệ.
    - A computer screen shot of text

      Description automatically generatedLazy initialization: chỉ khi nào getInstance() được gọi thì constructor mới được khởi tạo.
      * Ưu điểm: giải quyết được vấn đề của Eager inititalization.
      * Nhược điểm: nếu nhiều lần gọi hàm getInstance() cùng lúc thì sẽ có nhiều instance được tạo ra.
    - Thread safe: gọi phương thức synchronized của hàm getInstance() để tại một thời điểm chỉ có 1 luồng có thể truy cập vào hàm getInstance() và chỉ có 1 instance được A screen shot of a computer program

      Description automatically generatedtạo ra.
      * Ưu điểm: giải quyết được vấn đề của Lazy initialization.
      * Nhược điểm: phương thức synchronized chạy chậm và tốn hiệu năng. Không thể xử lý được những việc ngay trước và sau khi tạo instance và phải chờ.
    - A screen shot of a computer program

      Description automatically generatedDouble check locking: kiểm tra 2 lần sự tồn tại của instance. Các task có thể được giải quyết trước và sau khi kiểm tra lần 2.
      * Ưu điểm: giải quyết được vấn đề của Thread safe.
    - A computer screen shot of a program code

      Description automatically generatedBill Pugh Singleton: tạo ra một static nested class như 1 helper để tách biệt chức năng cho 1 class function rõ ràng hơn.
      * Ưu điểm: hiệu suất tốt, tránh được lỗi cơ chế tạo instance trong multi-thread.
    - Reflection: phá vỡ cấu trúc Single pattern – cho phép truy cập được vào các thành phần của class như constructor, method, field, … ngay cả khi chúng được khai báo là private.
      * A screen shot of a computer program

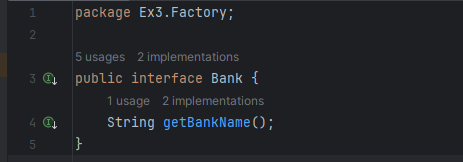
        Description automatically generatedCode:
      * A screen shot of a computer program

        Description automatically generatedOutput:
      * Như vậy, vẫn có 2 instance khác nhau được tạo ra. Điều này làm phá vỡ cấu trúc Singleton Pattern.
    - Enum: hàm constructor của enum là lazy, nghĩa là khi được sử dụng mới chạy hàm khởi tạo và nó chỉ chạy duy nhất một lần. A screen shot of a computer

      Description automatically generated
      * Ưu điểm: có thể giải quyết được vấn đề serialization/deserialization khi so sánh với static block initialization.
      * Nhược điểm: không thể extends từ 1 lớp.
    - Serialization and Singleton: implement interface Serialization trong lớp Singleton để lưu trữ trạng thái của nó trong file hệ thống và truy xuất lại nó sau.
      * A computer screen shot of text

        Description automatically generatedCode:
      * A computer screen shot of a program code

        Description automatically generatedKhi test chương trình serialize/deserialize:
      * Ta được kết quả:
      * A screen shot of a computer code

        Description automatically generatedTheo kết quả trên, ta thấy deserialize đối tượng của SerializedSingleton của 2 đối tượng khác nhau nhưng nó không xảy ra khi dùng enum.
* **Factory Method Pattern**
  + Khái niệm:
    - Là một trong những Pattern thuộc nhóm Creational Design Pattern.
    - Giúp quản lý và trả về các đối tượng theo yêu cầu, giúp khởi tạo đối tượng linh hoạt hơn.
    - Đối tượng được tạo sẽ không để lộ logic tạo cho người dùng và sẽ được tham chiếu đến bằng cách sử dụng một interface chung.
    - Được sử dụng khi có một class cha (super-class) với nhiều class con (sub-class), dựa trên đầu vào và phải trả về 1 trong những class con đó.
  + Triển khai:
    - Các thành phần cơ bản:
      * Super Class: một super class trong Factory Pattern có thể là một interface, abstract class hay một class thông thường.
      * Sub Classes: các sub class sẽ implement các phương thức của super class theo nghiệp vụ riêng của nó.
      * Factory Class: một class chịu trách nhiệm khởi tạo các đối tượng sub class dựa theo tham số đầu vào. Factory class sử dụng if-else hoặc switch-case để xác định class con đầu ra.
    - Bài toán: Thiết kế một API để client có thể sử dụng dịch vụ của một ngân hàng bất kỳ. Hiện tại, phía client chỉ đang sử dụng service của 2 ngân hàng là VietcomBank và TPBank. Tuy nhiên để dễ mở rộng sau này và không cần phải thay đổi code phía client khi sử dụng thêm dịch vụ của ngân hàng khác. Để đáp ứng yêu cầu này, ta sử dụng Factory Method Pattern.
    - Super Class: 
    - Sub Classes: A screen shot of a computer code

      Description automatically generatedA computer screen shot of code

      Description automatically generated
    - Bank Type: A computer screen shot of a black background

      Description automatically generated
    - A computer screen shot of a code

      Description automatically generatedFactory Class:
    - Client: A computer screen shot of a computer code

      Description automatically generated
    - Phía Client chỉ cần gọi duy nhất một phương thức BankFactory.getBank() là có thể sử dụng được service của một ngân hàng bất kỳ.
    - Khi hệ thống muốn cung cấp thêm dịch vụ của ngân hàng khác chỉ cần tạo class mới implement từ interface Bank, thêm vào logic khởi tạo Bank trong Factory và sẽ không làm ảnh hưởng đến code ở phía Client.
* **Chain of Responsibility Pattern**
  + Khái niệm:
    - Là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern).
    - Cho phép một đối tượng được gửi một yêu cầu nhưng không biết đối tượng nào sẽ nhận và xử lý nó. Điều này được thực hiện bằng cách kết nối các đối tượng nhận yêu cầu thành một chuỗi (chain) và gửi yêu cầu theo chuỗi đó cho đến khi có một đối tượng xử lý nó.
    - Hoạt động như một danh sách liên kết (Linked List) với việc đệ quy duyệt qua các phần tử (recursive traversal).
  + Các thành phần cơ bản:
    - Handler: định nghĩa 1 interface để xử lý các yêu cầu. Gán giá trị cho đối tượng successor (không bắt buộc).
    - ConcreteHandler: xử lý yêu cầu. Có thể truy cập đối tượng successor (thuộc class Handler). Nếu đối tượng ConcreteHandler không thể xử lý được yêu cầu, nó sẽ gửi lời yêu cầu cho successor của nó.
    - Client: tạo ra các yêu cầu và yêu cầu đó sẽ được gửi đến các đối tượng tiếp nhận.
  + Hoạt động:
    - Client gửi một yêu cầu để được xử lý, yêu cầu đó được gửi đến chuỗi (chain) các trình xử lý (handlers), đó là các lớp mở rộng lớp Handler. Mỗi Handler trong chuỗi lần lượt cố gắng xử lý yêu cầu nhận được từ Client. Nếu trình xử lý đầu tiên (ConcreteHandler) có thể xử lý nó, thì yêu cầu sẽ được xử lý. Nếu không được xử lý thì sẽ gửi đến trình xử lý tiếp theo trong chuỗi (ConcreteHandler +1).
  + Triển khai:
    - Bài toán: tạo ra một ứng dụng Logger xử lý các yêu cầu theo các mức độ (level).
    - Logger: một abstract class Handler, cho phép thực hiện một chain logger dựa vào giá trị LogLevel ứng với từng Handler. Nếu mức độ lỗi (severity) lớn hơn với LogLevel mà nó có thể handle thì sẽ thực hiện writeMessage(), A screenshot of a computer program

      Description automatically generatedđồng thời gọi Handler kế tiếp nếu có.
    - A screen shot of a computer program

      Description automatically generatedConsoleLogger, FileLogger, EmailLogger: đây là các ConcreteLogger, nó xác định LogLevel mà nó có thể xử lý, và cài đặt phương thức writeMessage() cho riêng nó.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* + - A screen shot of a computer program

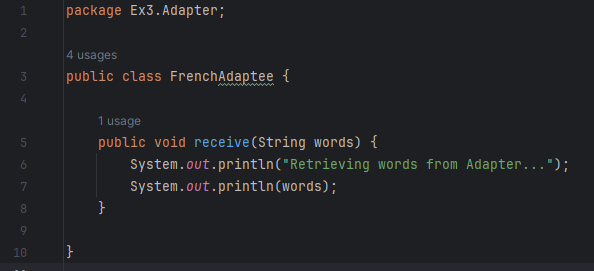
      Description automatically generatedLogLevel: là một enum (constant), dùng để xác định các mức độ ghi log.
    - A computer screen shot of a program code

      Description automatically generatedAppLogger: là một lớp tiện ích, tạo chuỗi Handler để xử lý ghi log. Ở đây ta thấy level của ConsoleLogger là Debug, level của FileLogger là Error và level của EmailLogger là Fatal.
    - A computer screen shot of text

      Description automatically generatedClient: sử dụng logger để ghi log.
  + Output: A screenshot of a computer program

    Description automatically generated
    - Kết quả trên cho thấy:
      * Ở Info level và Debug level: ConsoleLogger có thể xử lý được vì ConsoleLogger có level là Debug.
      * Ở Error level: vì ConsoleLogger không thể handle được nên in ra message “Error message” và chuyển cho FileLogger xử lý.
      * Ở Fatal level: FileLogger không xử lý được vì FileLogger chỉ ở level Error nên sẽ chuyển cho EmailLogger và in ra Fatal message, ConsoleLogger cũng tương tự và in ra Fatal message. EmailLogger vì ở level Fatal nên có thể handle được.
* **Adapter Pattern**
  + Khái niệm:
    - Là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern).
    - Cho phép các interface (giao diện) không liên quan tới nhau có thể làm việc cùng nhau. Đối tượng giúp kết nối các interface gọi là Adapter.
    - Còn được 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.
  + Thành phần:
    - Adaptee: định nghĩa interface không tương thích, cần 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 implement:
    - A diagram of a computer

      Description automatically generatedObject Adapter – Composition (chứa trong): 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). Cách này cho phép Adapter hoạt động với nhiều Adaptee nếu cần thiết.
    - Class Adapter – Inheritance (kế thừa): Adapter kế thừa lớp có sẵn với interface không tương thích (Adaptee), đồng thời cài đặt interface mà người dùng mong muốn (Target). Trong trường hợp này, Adapter sẽ là một lớp con của Adaptee và Adapter sẽ là một lớp phụ cụ thể của Adaptee.
  + A diagram of a computer

    Description automatically generatedTriển khai:
    - Bài toán: ứng dụng Translation – một người Việt muốn trao đổi với một người Pháp nhưng không biết tiếng Pháp nên cần phải có một người chuyển đổi từ ngôn ngữ tiếng Việt sang ngôn ngữ tiếng Pháp.
    - FrenchAdaptee: đây là interface hoặc class được người Pháp sử dụng để nhận message được chuyển đổi từ thông dịch viên (Translator).
    - A screen shot of a computer program

      Description automatically generatedTranslatorAdapter: thông dịch viên (Translator) sẽ là Adapter, nhận message tiếng Việt từ Client và chuyển đổi nó sang tiếng Pháp trước khi gửi cho người Pháp.
    - VietnameseTarget: đây là nội dung message được Client cung cấp cho thông dịch viên (TranslatorAdapter). A screen shot of a computer program

      Description automatically generated
    - A screen shot of a computer

      Description automatically generatedClient: người Việt sẽ là Client trong ví dụ này vì người này cần gửi một số message cho người Pháp.
  + Output: A screen shot of a computer

    Description automatically generated