ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

ABSTRACT FACTORY

Học phần: Lập trình hướng đối tượng

Lớp: 19CQ/4

Nhóm: OOP For Life!

Họ và tên các thành viên:

1. **BÙI LÊ TUẤN ANH – 19120163**
2. **NGÔ NHẬT TÂN – 19120128**
3. **PHẠM TIẾN KHẢI – 19120250**

Thành phố Hồ Chí Minh, tháng 05 năm 2021

Nội dung

LỜI NÓI ĐẦU

*Đây là bản báo cáo của học phần **"Lập trình hướng đối tượng"**, giảng dạy tại **Khoa Công nghệ thông tin, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh**. Báo cáo này được thực hiện bởi ba thành viên:*

1. ***Bùi Lê Tuấn Anh – Mã số sinh viên: 19120163***
2. ***Ngô Nhật Tân – Mã số sinh viên: 19120128***
3. ***Phạm Tiến Khải – Mã số sinh viên: 19120250***

*Nội dung chính của bản báo cáo này tập trung phân tích những khía cạnh căn bản về mẫu thiết kế **Abstract Factory** – một trong số 23 mẫu thiết kế của **Lập trình hướng đối tượng**. Trong đó bao gồm một số bài toán thông dụng sử dụng mẫu thiết kế này, cũng như **lợi ích** của việc dùng mẫu thiết kế trong thiết kế và phát triển phần mềm.*

Trong quá trình thực hiện bản báo cáo này, nhóm chúng tôi không tránh khỏi những thiếu sót (do kiến thức còn hạn chế), nhưng chúng tôi đã và đang cố gắng hết sức để hoàn thiện nó tốt nhất trong khả năng của mình. Mong nhận được sự thông cảm từ mọi người.

*Chúng tôi cũng xin chân thành cảm ơn sự hỗ trợ tận tình từ các giảng viên **Khoa Công nghệ Thông tin** nói chung và bộ môn **Công nghệ phần mềm** nói riêng. Chúng tôi cũng xin gửi lời cảm ơn và ghi nhận đối với những lập trình viên trên khắp thế giới đã không ngừng nghỉ làm việc để mang đến những chương trình hữu ích cho người sử dụng mà ở đó chúng tôi tham khảo và sử dụng mã nguồn một cách thận trọng nhất và tôn trọng nguyên tác nhất.*

*Toàn bộ mã nguồn (và cả báo cáo này) cũng sẽ được đăng tải trên **Github** tại **repository** (hiện tại đang ở chế độ riêng tư) mang tên **OOP-For-Life**. Vui lòng gửi thông tin email đăng ký Github qua địa chỉ 19120163@student.hcmus.edu.vn để được hỗ trợ truy cập **(toàn bộ ở nhánh Master)**. Xin trân trọng cảm ơn.*

I. BÀI TOÁN ĐẶT RA

Trước đó, trong phần nghiên cứu về Factory Method, chúng tôi đưa ra một bài toán như sau:

Một công ty ô tô có 3 dòng xe: Xe bán tải, Xe du lịch/Thể thao, Xe mui trần (xe sang). Cần có phần mềm quản lý nhà máy sản xuất của công ty này.

Bài toán này sẽ được nâng cấp với một **yêu cầu đặc biệt** (tình huống giả định thực tế) như sau:

Ngay khi dịch COVID-19 bùng phát tại Việt Nam, để phục vụ tiến trình tiêm vaccine cho người dân tại các địa phương đang có dịch, Bộ Y tế đã giao công ty sản xuất dòng xe chuyên dụng cho Bộ. Trong đó, Bộ đã và đang phê duyệt cho việc sử dụng một số thương hiệu vaccine, mỗi thương hiệu vaccine có 1 loại xe vận chuyển khác nhau, cụ thể như sau:

1. **Vaccine AstraZeneca, do Đại học Oxford (Anh) và công ty AstraZeneca (liên doanh Anh – Thụy Điển) sản xuất.**
2. **Vaccine Nano Covax, do công ty Nanogen (Việt Nam) sản xuất.**
3. **Vaccine Pfizer/BioNTech, do liên doanh hai công ty Pfizer (Mỹ) và BioNTech (Đức) sản xuất.**

II. PHÂN TÍCH BÀI TOÁN

Yêu cầu đặt ra lúc này cho bài toán mới sẽ là:

1. Tiến hành mở rộng phần mềm sản xuất với tốc độ cao nhất, tức là ***không cần phải chỉnh sửa mã nguồn cũ mà vẫn có thể viết mã nguồn mới***, dựa vào mã nguồn cũ. Yêu cầu này được đặt ra nhằm đảm bảo việc sẽ có xe trong thời gian **sớm nhất**, để vaccine được bảo quản và vận chuyển **nhanh nhất** đến các địa phương đang có dịch.
2. ***Tách hoàn toàn*** mảng xe chuyên dụng với mảng xe kinh doanh. Đây là yêu cầu quan trọng, bởi không thể quản lý được việc sản xuất xe của cả hai mảng nếu mã nguồn được đặt chung **tại cùng một vị trí**, rất khó kiểm soát và không đảm bảo an toàn sản xuất.
3. Việc sản xuất của các loại xe này phải tuân thủ ***tính độc lập*** (tức là không ảnh hưởng lẫn nhau giữa các loại xe).
    a. Mỗi loại xe, mỗi dòng xe có các thuộc tính **khác nhau**.
    b. **Không thể** dùng cùng một loại phần mềm của mảng xe kinh doanh cho mảng chuyên dụng
    c. **Không thể** dùng cùng thông tin của dòng xe chở vaccine A cho dòng xe chở vaccine B, vốn có điều kiện bảo quản trên xe khác dòng xe chở vaccine A.
4. Sau khi dịch bệnh được kiểm soát, các bộ ngành khác như **Bộ Công an, Bộ Quốc phòng, Bộ Nội vụ**… khi đặt hàng công ty, phần mềm này phải có khả năng ***tiếp tục mở rộng*** để đáp ứng nhu cầu **cấp bách** của quốc gia mà **không ảnh hưởng** đến dây chuyền sản xuất xe chở vaccine của Bộ Y tế.

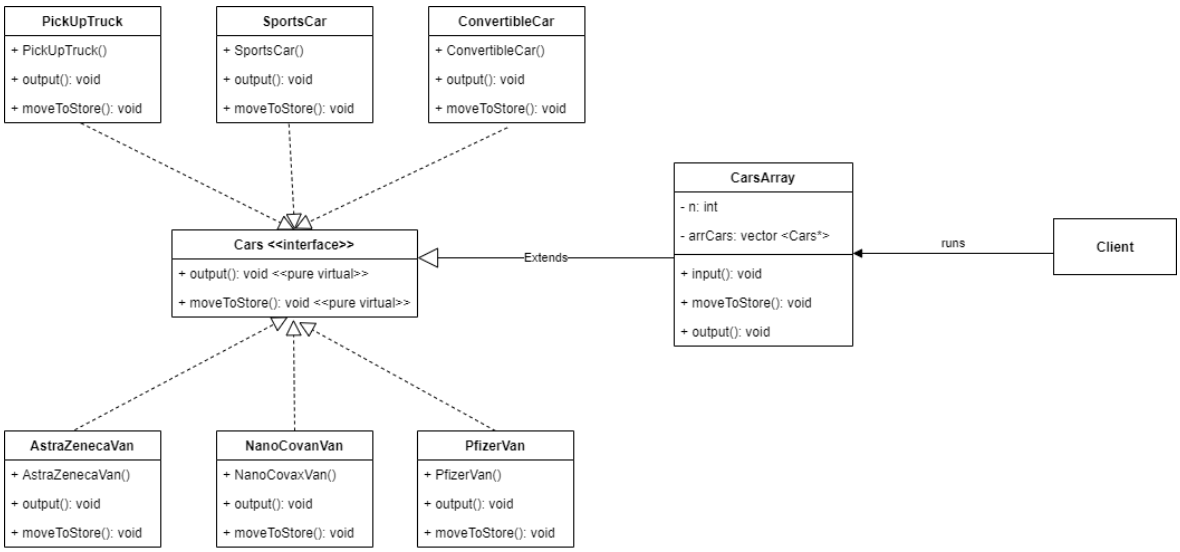
Đây là những yêu cầu hết sức thực tế, đòi hỏi công ty phải có giải pháp để gia tăng khả năng sản xuất, đáp ứng yêu cầu của Chính phủ cũng như người dân.

Ở phần sau của bản báo cáo này, chúng tôi xin giới thiệu một số hướng tiếp cận cho bài toán này cũng như các bài toán khác có tính chất tương tự.

III. MỘT SỐ HƯỚNG TIẾP CẬN TRƯỚC ĐÂY

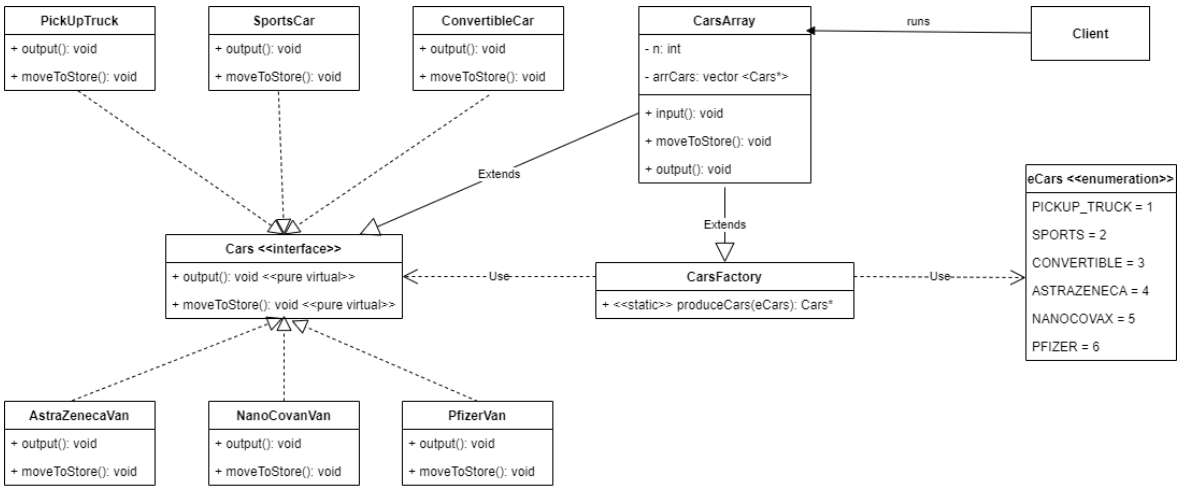
1. Hướng tiếp cận cơ bản - Kế thừa và Hàm dựng

Gom tất cả các loại xe thuộc cùng một dòng về cùng một lớp, tức là: Tất cả các loại xe Bán tải, Thể thao và Mui trần, cùng với ba loại xe chuyên dụng chở vaccine sẽ chỉ kế thừa đúng một tính chất của một xe ô tô bình thường. Mỗi loại xe sẽ có một hàm khởi tạo tương ứng. Sơ đồ lớp tương ứng sẽ đặt ở dưới:



2. Hướng tiếp cận nâng cao – Factory Method

Từ hướng tiếp cận cơ bản ở trên, tiến hành lập một lớp trừu tượng, một "nhà máy" duy nhất để sản xuất tất cả các loại xe từ dòng xe Kinh doanh lẫn Chuyên dụng. Sơ đồ lớp sẽ có sự thay đổi, cụ thể như ở dưới:



3. Nghịch lý từ cả hai hướng tiếp cận

Từ cả hai hướng tiếp cận trên, chúng ta có thể thấy được:

- **Đối với hướng tiếp cận cơ bản**:
  - Bất cập sẽ phát sinh ở việc 3 loại xe **Chuyên dụng** không thể đem ra kinh doanh, nên không thể kế thừa hàm **moveToStore()** (không thể sử dụng được).
  - Giả sử viết thêm hàm **deliverToGov()** thì lúc này, cả ba lớp của dòng xe **Kinh doanh** buộc phải định nghĩa lại hàm này, nếu không thì ba lớp này cũng không thể sử dụng được (***vì là lớp trừu tượng***).
- **Đối với hướng tiếp cận nâng cao**:
  - Dù có một nhà máy chung nhưng lúc này, bắt buộc phải định nghĩa chung một hàm khởi tạo xe cho cả hai dòng xe. Đây là việc hết sức vô lý, bởi vốn dĩ các thông số kỹ thuật cũng như yêu cầu từng dòng xe là hoàn toàn khác nhau.
  - Dòng xe **Kinh doanh** có thể đưa đến cửa hàng nhưng dòng xe **Chuyên dụng** không thể đưa đến cửa hàng (***theo đúng yêu cầu, Bộ Y tế toàn quyền sử dụng để phục vụ cho quốc gia***).
- Cả hai hướng tiếp cận đều gây ra tình trạng nhập nhằng và khó kiểm soát, ***làm sai lệch yêu cầu bài toán về tính độc lập – chữ S và tính – chữ O trong chùm nguyên lý SOLID***.
- Ngoài ra, khi một **Factory Method** trở nên quá lớn (**có quá nhiều xử lý if-else hay switch-case**) thì sẽ phải sử dụng ***nhiều điều kiện logic***, lúc này sẽ rất khó khăn trong việc mở rộng thêm các nhóm đối tượng khác và tạo nhiều product khác nhau.
- ***Không che giấu việc khởi tạo các đối tượng với người dùng***, điều đó dẫn đến việc người dùng có thể tự định nghĩa lại, không lường trước được sai lệch kết quả khi mã nguồn bị chỉnh sửa mà không kiểm soát được.

Với những nghịch lý được nêu ở trên, cần phải tìm ra hướng tiếp cận mới cho bài toán, đảm bảo việc bảo trì, quản lý, kiểm soát việc thiết kế và sử dụng mã nguồn được tốt nhất. Do đó, một mẫu thiết kế mới phải được sử dụng cho bài toán này.

Ở phần sau, chúng tôi sẽ giới thiệu Abstract Factory, một mẫu thiết kế được nâng cấp từ chính mẫu thiết kế Factory Method – một mẫu thiết kế cùng loại, nhằm giúp ích cho việc giải quyết bài toán này.

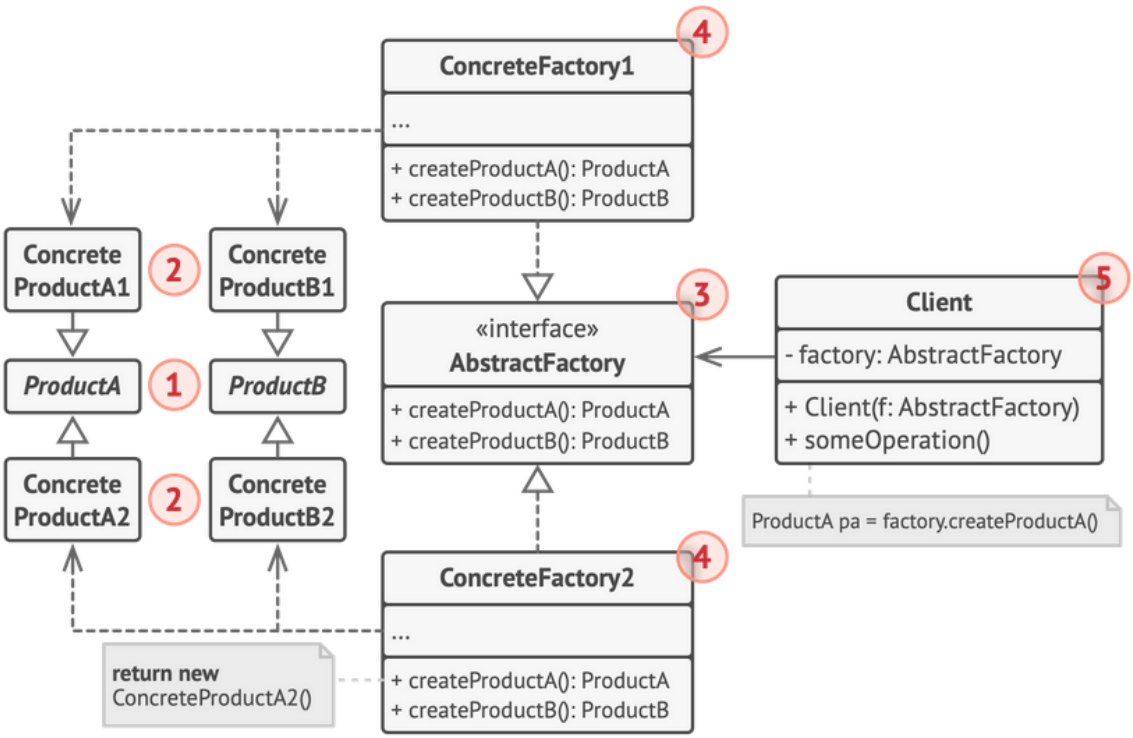
IV. GIỚI THIỆU VỀ MẪU THIẾT KẾ

Để phục vụ tốt nhất cho bài toán này, chúng tôi giới thiệu mẫu thiết kế Abstract Factory (Nhà máy ảo/Nhà máy trừu tượng) – một mẫu thiết kế trong lập trình hướng đối tượng (OOP – Object-Oriented Programming) được nâng cấp từ chính mẫu thiết kế Factory Method (chính là hướng tiếp cận nâng cao trong bài toán, được giới thiệu ở mục III trước đó).

Mẫu thiết kế Abstract Factory được định nghĩa như sau:

Abstract Factory là một mẫu thiết kế trong OOP thuộc nhóm Khởi tạo (Creational), cung cấp một lớp giao diện (interface) có khả năng tạo ra tập hợp các đối tượng liên quan với nhau hoặc phụ thuộc lẫn nhau mà không cần chỉ rõ lớp cụ thể sẽ được tạo ra.

Mẫu thiết kế này được cụ thể hóa bằng sơ đồ lớp (UML) như sau:



Ta có thể cụ thể hóa sơ đồ này thành ba phân vùng: Phân vùng dành cho Sản phẩm (**Product**, ***số 1 và số 2***), phân vùng dành cho nhà máy (**Abstract Factory**, ***số 3 và số 4***) và phân vùng truy cập chính (**Client**, ***số 5***).

1. ***Các lớp Dòng sản phẩm (Product)****: sẽ khai báo và gom nhóm các sản phẩm thuộc cùng một dòng sản phẩm lại với nhau, nhằm thuận lợi hơn cho quá trình quản lý.*
2. ***Các lớp Loại sản phẩm (Concrete Product)****: Kế thừa theo từng dòng sản phẩm và tùy biến cho phù hợp với từng loại sản phẩm khác nhau.*
3. ***Lớp Nhà máy chính (AbstractFactory)****: Nhà máy này có chức năng khai báo các phương thức để khởi tạo từng dòng sản phẩm khác nhau từ các lớp Sản phẩm.*
4. ***Các lớp Nhà máy con (ConcreteFactory)****: Mỗi nhà máy con kế thừa dây chuyền của nhà máy cha (AbstractFactory) và định nghĩa lại sao cho phù hợp với từng dòng sản phẩm mà mình đang sản xuất.*
5. ***Lớp truy cập chính (Client)****: Có thể truy cập vào các nhà máy để xin cấp quyền sản xuất sản phẩm.*

Ở phần V, chúng ta sẽ áp dụng mẫu thiết kế này vào bài toán đặt ra để đánh giá sự hiệu quả và tính ưu việt của mẫu thiết kế này...

V. HƯỚNG TIẾP CẬN SỬ DỤNG MẪU THIẾT KẾ

1. Nhắc lại bài toán:

Ở phần I, chúng ta xét đến bài toán sau:

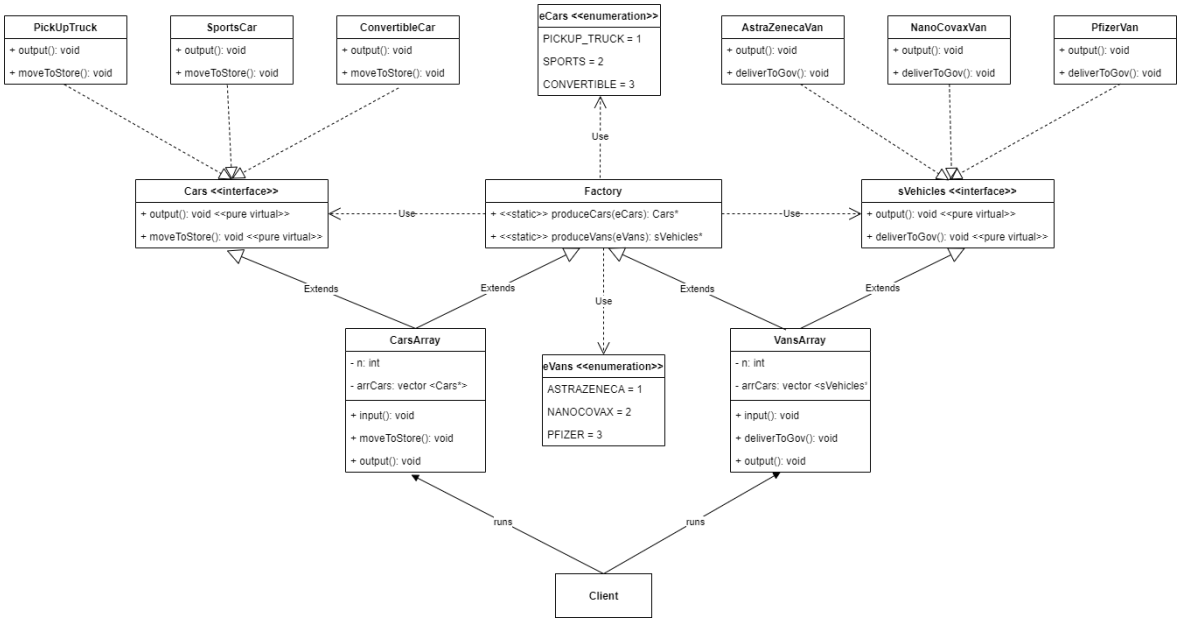
> ***Một công ty ô tô có 3 loại xe thuộc dòng xe Kinh doanh: Xe bán tải, Xe du lịch/Thể thao, Xe mui trần (xe sang).***
>
> ***Ngay khi dịch COVID-19 bùng phát tại Việt Nam, để phục vụ tiến trình tiêm vaccine cho người dân tại các địa phương đang có dịch, Bộ Y tế đã giao công ty sản xuất dòng xe chuyên dụng cho Bộ. Trong đó, Bộ đã và đang phê duyệt cho việc sử dụng một số thương hiệu vaccine, mỗi thương hiệu vaccine có 1 loại xe vận chuyển khác nhau, cụ thể như sau:***
>
> 1. **Vaccine AstraZeneca, do Đại học Oxford (Anh) và công ty AstraZeneca (liên doanh Anh – Thụy Điển) sản xuất.**
> 2. **Vaccine Nano Covax, do công ty Nanogen (Việt Nam) sản xuất.**
> 3. **Vaccine Pfizer/BioNTech, do liên doanh hai công ty Pfizer (Mỹ) và BioNTech (Đức) sản xuất.**
>
> ***Cần có phần mềm quản lý nhà máy sản xuất của công ty này.***

Bài toán này có đầy đủ yếu tố để sử dụng mẫu thiết kế Abstract Factory, gồm có:

- Hai dòng xe: ***Kinh doanh*** và ***Chuyên dụng***. Mỗi dòng xe sẽ tương ứng với một lớp **Dòng sản phẩm (Product)**.
- Dòng xe **Kinh doanh** có 3 loại xe: ***Bán tải, Thể thao*** và ***Mui trần.*** Dòng xe **Chuyên dụng** có 3 loại xe chuyên chở 3 loại vaccine COVID-19: ***AstraZeneca, Nanocovax*** và **Pfizer/BioNTech**. Mỗi loại xe tương ứng với một lớp **Loại sản phẩm (Concrete Product)** ứng với từng dòng xe.
- Cần có phần mềm chung quản lý nhà máy sản xuất **(Abstract Factory)**, nhưng tách riêng hai dòng xe để không bị xung đột giữa hai dòng **(Concrete Factory)**.

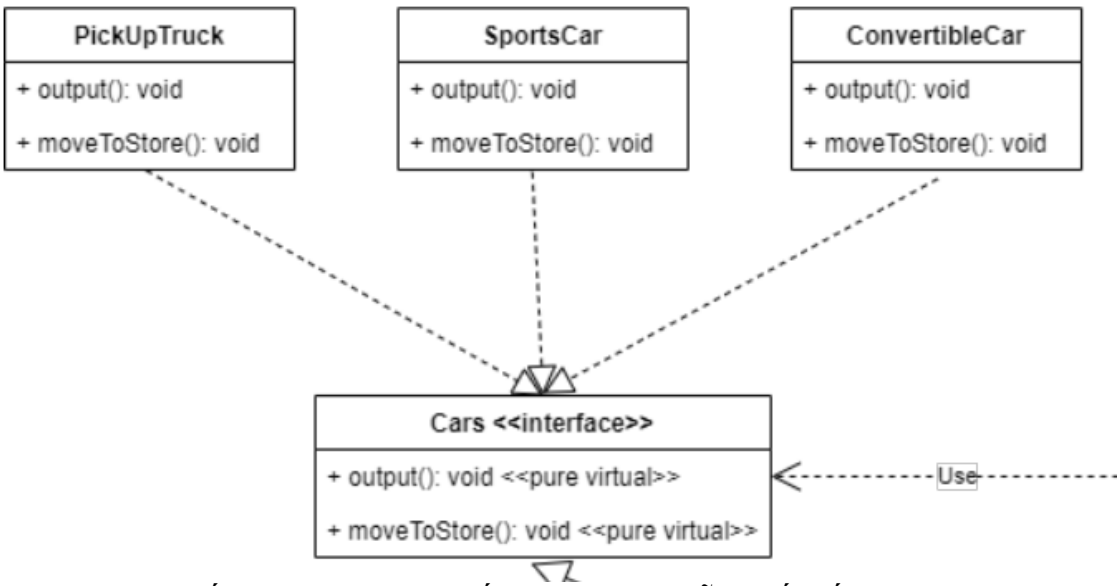
2. Sơ đồ lớp với mẫu thiết kế mới – Phân tích sơ đồ

Chúng ta sẽ vẽ sơ đồ lớp (UML) cho bài toán vừa rồi. Trong đó sẽ có một số điểm khác biệt so với mẫu thiết kế gốc.

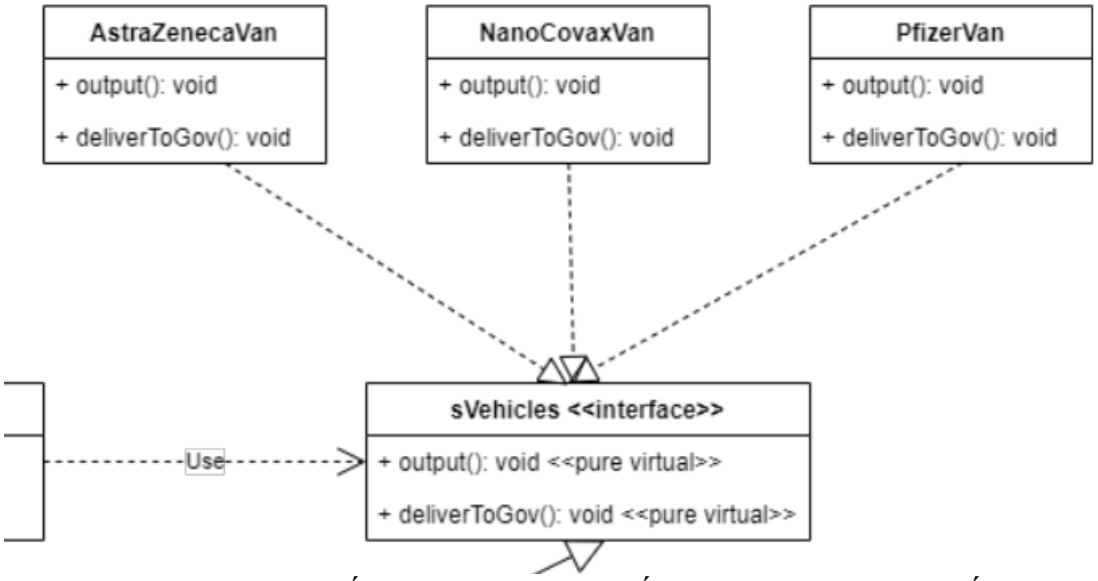


Ta phân tích sơ đồ lớp này thành ba phần:

A. *Phân vùng các lớp **Sản phẩm** (gọi chung các lớp **Product** và **ConcreteProduct**)*



Lớp **Cars** tiếp tục giữ nguyên cấu trúc như ở mẫu thiết kế Factory Method, với ba lớp con kế thừa: ***PickUpTruck, SportsCar*** và ***ConvertibleCar***.

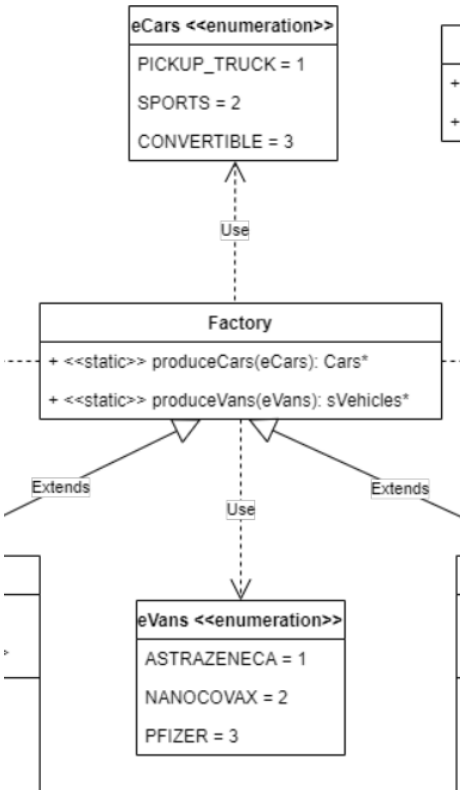


Lớp **sVehicles** xuất hiện, giữ nguyên cấu trúc với ba lớp con kế thừa tương ứng với ba loại vaccine: ***AstraZenecaVan, NanoCovaxVan*** & ***Pfizer Van***.

Tương ứng trên sơ đồ lớp của mẫu thiết kế sẽ là:

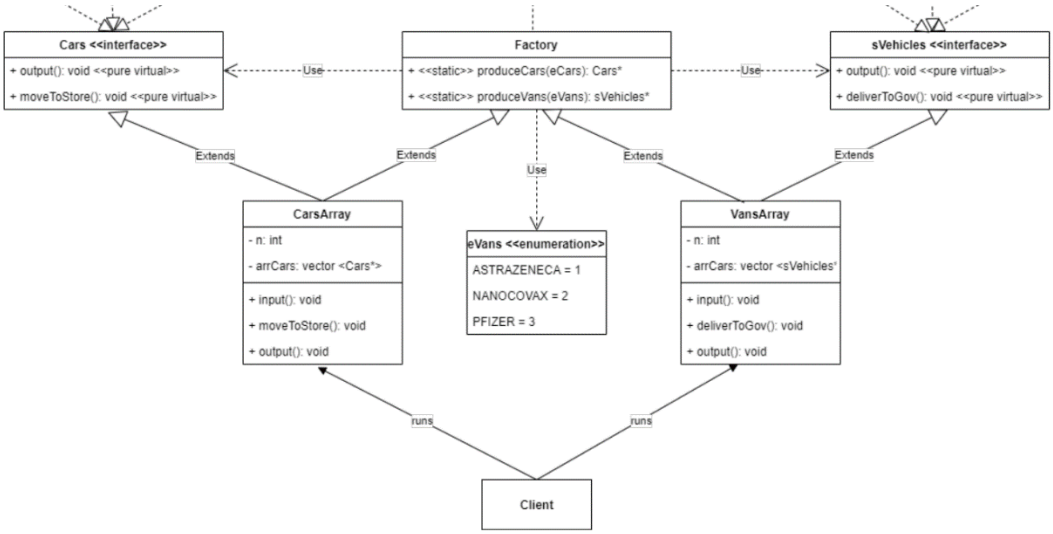
- Lớp **Dòng sản phẩm** (***Product***): Hai lớp **Cars** và **sVehicles**
- Các lớp **Loại sản phẩm** (***ConcreteProduct***): **Cars** chứa ba lớp ***PickUpTruck, SportsCar*** và ***ConvertibleCar***; **sVehicles** chứa ba lớp: ***AstraZenecaVan, NanoCovaxVan*** và ***Pfizer Van***.

B. *Phân vùng lớp* ***Nhà máy*** *(gọi chung các lớp* ***AbstractFactory*** *và* ***ConcreteFactory****)*



- Xuất hiện thêm 1 enum ***eVans***, gom nhóm và đánh số để phục vụ việc lựa chọn xe.
- Xuất hiện thêm 1 hàm tĩnh ***produceVans***, hàm này có cấu trúc tương tự hàm ***produceCars*** đã được cấu hình trước đó, chịu trách nhiệm khởi tạo từng loại xe chuyên dụng cho việc chở vaccine.
- Tương ứng với 2 hàm trong **Factory** chính là 2 **ConcreteFactory**, đây là sự khác biệt so với sơ đồ gốc, giúp giảm số lượng lớp cần phải viết lại, tiết kiệm thời gian và tận dụng hết khả năng của mẫu **Abstract Factory**.

*C. Phân vùng các lớp lưu trữ (gọi chung các lớp **Array**)*

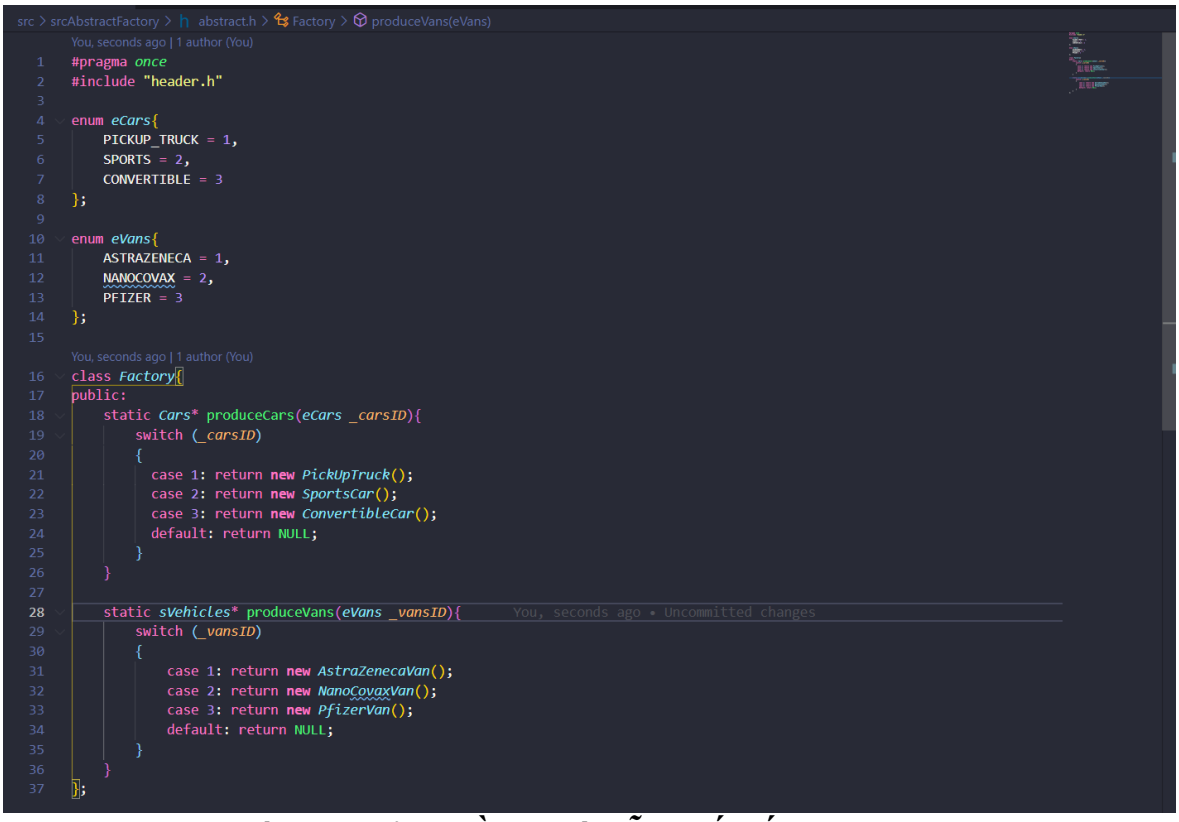


Hai lớp **CarsArray** và **VansArray** có tính chất tương tự nhau, đều là hai lớp quản lý việc sản xuất, tiêu thụ và vận chuyển xe đến phân khu cần thiết, **CarsArray** đến cửa hàng, còn **VansArray** đến khu vực tập kết của Chính phủ.

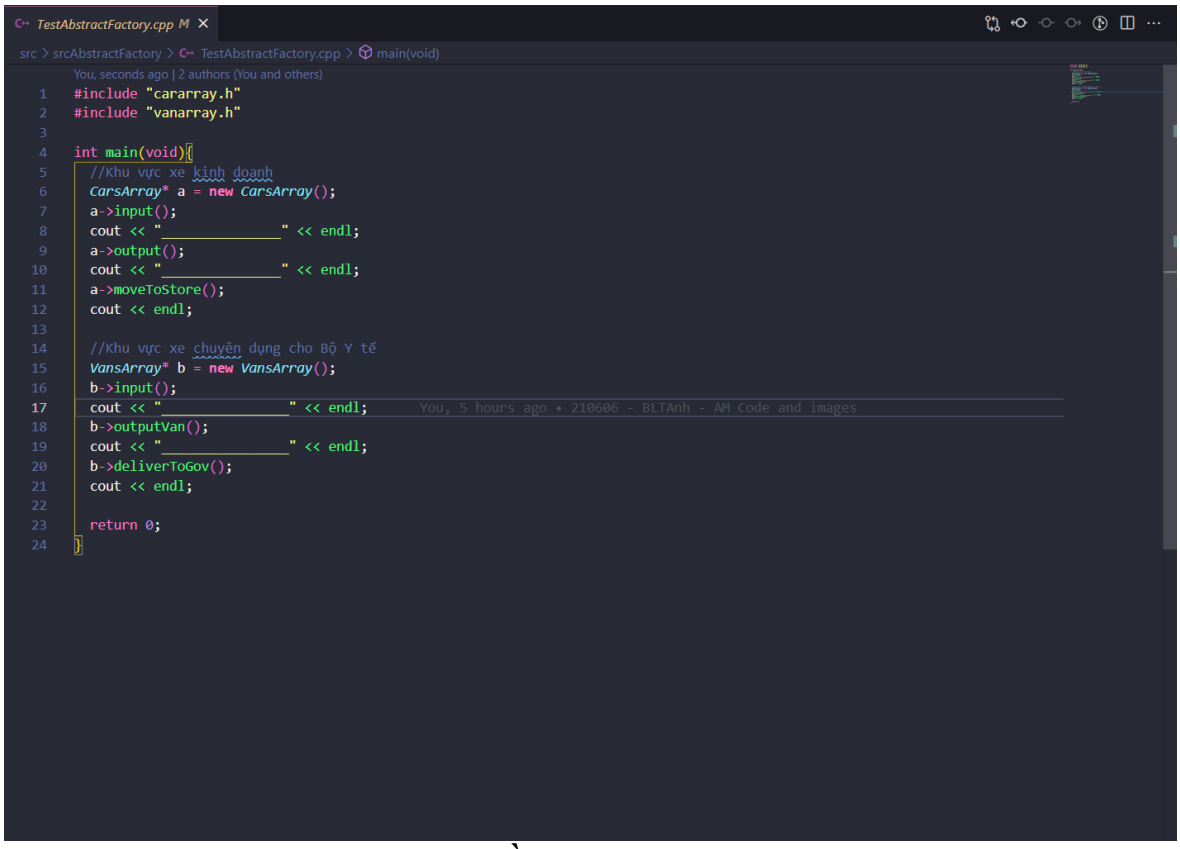
3. Một số đoạn mã nguồn đáng chú ý

```
You, seconds ago | 1 author (You)
class sVehicles{
public:
    virtual void outputVan() = 0;
    virtual void deliverToGov() = 0;
};

You, seconds ago | 1 author (You)
class AstraZenecaVan: public sVehicles{
public:
    void outputVan(){
        cout << "A van for AstraZeneca vaccines is in production!!!" << endl;
    }
    void deliverToGov(){
        cout << "The van for AstraZeneca vaccines has now been delivered to Government for emergency use." << endl;
    }
};

You, seconds ago | 1 author (You)
class NanoCovaxVan: public sVehicles{
public:
    void outputVan(){
        cout << "A van for NanoCovax vaccines is in production!!!" << endl;
    }
    void deliverToGov(){      You, seconds ago • Uncommitted changes
        cout << "The van for NanoCovax vaccines has now been delivered to Government for emergency use." << endl;
    }
};

You, seconds ago | 1 author (You)
class PfizerVan: public sVehicles{
public:
    void outputVan(){
        cout << "A van for Pfizer/BioNTech vaccines is in production!!!" << endl;
    }
    void deliverToGov(){
        cout << "The van for Pfizer/BioNTech vaccines has now been delivered to Government for emergency use." << endl;
    }
};
```

Hình 1: Lớp sVehicles và các lớp kế thừa.



Hình 2: Mã nguồn xử lý mẫu thiết kế Abstract Factory.



Hình 3: Mã nguồn chương trình chính (hàm main).

4. Kiểm thử mã nguồn

Toàn bộ mã nguồn của mẫu thiết kế được chúng tôi nộp kèm theo bản báo cáo này, được trải qua quá trình kiểm thử độc lập giữa các thành viên trong nhóm. Sau đây là kết quả kiểm thử, được chụp lại và ghi nhận trong khi làm báo cáo:

Hình ảnh kiểm thử trên gdbDebugger

Hậu quá trình kiểm thử, chúng tôi tự đánh giá đây là một mẫu thiết kế có khả năng đáp ứng được nhu cầu của bài toán ở mức độ ***KHÁ TỐT (80-85%)****. Tuy nhiên, tỷ lệ này vẫn chưa cao so với kỳ vọng, do mã nguồn đã được mở rộng khá nhiều trong quá trình thiết kế:*

Đặt giả thuyết có thêm những đơn vị cần đặt hàng những dòng xe mới (không liên quan đến những dòng xe trước đó) từ công ty này, lúc này buộc lớp ***Factory*** *sẽ phải tiếp tục mở rộng thêm để đảm bảo phục vụ nhu cầu. Việc này có thể dẫn đến việc mã nguồn khó quản lý, nếu không tách ra riêng biệt. Do đó, một số mẫu thiết kế như* ***Builder*** *hay* ***Prototype*** *có thể giúp xử lý tình huống này. Ở phạm vi giới hạn của bản báo cáo này, chúng tôi chỉ nêu những đánh giá chung nhất và một số gợi ý phát triển mã nguồn theo hướng hoàn thiện hơn, dễ cài đặt và bảo trì hơn.*

VI. MỘT SỐ BÀI TOÁN VÀ VÍ DỤ KHÁC

Một số bài toán sau đây có liên quan đến mẫu thiết kế Abstract Factory, xem như các ví dụ để người đọc tham khảo:

1. Bài toán hiệu sách:

Một hiệu sách có kinh doanh sách thuộc 2 ngôn ngữ Anh, Việt với các thể loại sau:

- **Tiếng Việt**
  - ❖ ***Truyện tranh***: Có thể có nhiều phần
  - ❖ ***Sách văn học***: Có thể có nhiều chương
  - ❖ ***Sách ngoại văn***: Có thể có nhiều chương, nhiều tập
- **Tiếng Anh**
  - ❖ ***Tiểu thuyết***: Có thể có nhiều chương
  - ❖ ***Sách khoa học***: Có thể có nhiều tập, nhiều chương.
  - ❖ ***Sách thiếu nhi***: Có thể có nhiều tập.
- *Mỗi quyển sách có những thông tin: mã, tên, tác giả, năm xuất bản, số trang, thể loại. Vẽ sơ đồ lớp và viết chương trình quản lý hiệu sách này.*

Yêu cầu:

- *Phần mềm tích hợp việc tìm kiếm & thêm sách, phục vụ trao đổi và kinh doanh.*
- *Việc thao tác trên một quyển sách không làm ảnh hưởng đến các quyển sách khác trong hệ thống.*

2. Bài toán quản lý địa chỉ/số điện thoại:

Liên minh Viễn thông thế giới (ITU) cần xây dựng một hệ thống quản lý địa chỉ và mã vùng điện thoại tương ứng với từng quốc gia.

Theo đó, có thêm 3 quốc gia sẽ nhận địa chỉ và đầu số điện thoại bao gồm:

- **Đông Timor (Timor Leste)**
- **Kosovo**
- **Palestine**

Vẽ sơ đồ lớp và viết chương trình quản lý này.

3. Bài toán cửa hàng điện thoại/laptop:

Một cửa hàng điện thoại có kinh doanh điện thoại và laptop của các hãng:

Điện thoại	**Laptop**
❖ Samsung	❖ Dell
❖ Apple	❖ Acer
❖ Xiaomi	❖ MSI

- *Mỗi điện thoại và laptop có mã, tên, năm sản xuất, hãng sản xuất.*
- *Khách hàng đến cửa hàng có thể xem thông tin của tất cả sản phẩm của cửa hàng đang kinh doanh. Vẽ sơ đồ lớp và viết chương trình quản lý cửa hàng trên.*

Yêu cầu: *Việc thêm điện thoại/laptop của 1 hãng mới không ảnh hưởng đến các hãng khác.*

Lưu ý đối với bài 2:

- Việc thiết kế chương trình sẽ dựa vào 2 yếu tố chính: **Địa chỉ** và **mã vùng**.
- Một lớp **Factory** sẽ nhận nhiệm vụ tạo ra cả hai yếu tố này.
- Mỗi quốc gia sẽ tương ứng với một lớp **ConcreteFactory**, kế thừa và định nghĩa lại cho phù hợp.
- Bài toán này có cấu trúc khác biệt hơn so với hai bài 1 và 3. Tuy nhiên vẫn có thể sử dụng mẫu thiết kế **Abstract Factory** để thực hiện bài toán.

KẾT LUẬN

Sau khi phân tích và thiết kế mã nguồn cho phần mềm, có thể nói mẫu thiết kế Abstract Factory mang đến một số ưu và nhược điểm so với những hướng tiếp cận trước đây, cụ thể là:

- **Ưu điểm**:
  - ✓ *Đảm bảo tính tương thích cao giữa các sản phẩm trong cùng một dòng và giữa các sản phẩm với nhau.*
  - ✓ *Tách riêng phần mã nguồn của sản phẩm với mã nguồn chính, giúp dễ dàng quản lý và kiểm soát mã nguồn.*
  - ✓ *Tuân thủ đồng thời hai nguyên lý trong chùm nguyên lý **SOLID** trong OOP:*
    - ***Nguyên lý Đơn trách nhiệm (Single Responsiblity Principle)**: Tách phần mã nguồn tạo sản phẩm với các hàm thủ tục khác, giúp tăng khả năng bảo trì và kiểm soát mã nguồn.*
    - ***Nguyên lý Đóng – mở (Open/Closed Principle)**: Mã nguồn có thể mở rộng với nhiều dòng sản phẩm với nhiều loại sản phẩm khác nhau.*
- **Nhược điểm:** *Mã nguồn dài hơn, phức tạp và mất nhiều thời gian thiết kế hơn.*

*Mẫu thiết kế **Abstract Factory**, tuy chưa phải là giải pháp tốt nhất về mặt thiết kế, nhưng lại là một lựa chọn khá hiệu quả khi cần viết một phần mềm với nhiều lớp, nhiều thuộc tính có liên quan hay phụ thuộc lẫn nhau. Mẫu thiết kế này, cùng với mẫu đơn giản hơn là **Factory Method** và các mẫu phức tạp hơn cùng nhóm **Khởi tạo (Creational)** là **Prototype** hay **Builder**... có thể nói là những mẫu thiết kế dễ xây dựng, dễ tiếp cận nhất đối với các lập trình viên trong **Lập trình hướng đối tượng**.*

TÀI LIỆU THAM KHẢO

1. https://refactoring.guru/design-patterns/abstract-factory
2. https://vi.wikipedia.org/wiki/Abstract_factory ***(Wikipedia Tiếng Việt)***
3. ***Head First Design Patterns: A Brain-Friendly Guide*** (Eric Freeman, Kathy Sierra, Bert Bates,..)
4. https://topdev.vn/blog/gioi-thieu-abstract-factory-pattern/
5. https://en.wikipedia.org/wiki/Abstract_factory_pattern ***(English Wikipedia)***
6. https://refactoring.guru/design-patterns/abstract-factory/cpp/example
7. https://www.stdio.vn/software-architecture/design-pattern-abstract-factory-pattern-1xe8z
8. https://www.codeproject.com/Articles/331304/Understanding-and-Implementing-Abstract-Factory-2
9. Khu vực kiểm thử phần mềm online: https://www.onlinegdb.com/