Design patter factory

Contents

[Overiview 2](#_Toc82288455)

[Mục đích 2](#_Toc82288456)

[Method factory 3](#_Toc82288457)

[Sản phẩm 3](#_Toc82288458)

[Registry Mapping Product 5](#_Toc82288459)

[Run 6](#_Toc82288460)

[Abstract Factory 6](#_Toc82288461)

[Cách tạo 7](#_Toc82288462)

[Run 9](#_Toc82288463)

[Chú ý 10](#_Toc82288464)

[Kết luận 10](#_Toc82288465)

[Reference: 11](#_Toc82288466)

# Overiview

Factory design pattern trở thành pattern phổ biến trong ngôn ngữ hiện đại giống như Java và c#. Nó có nhiều biến thể và cách triển khai. Nếu bạn tìm kiếm trên google sẽ ra 2 loại phổ biến là Method và abstract factory.

# Mục đích

Factory là một trong những kiểu pattern tạo ra đối tượng giống như singleton, hay các pattern tạo ra object, factory tạo ra object từ những thông tin ban đầu đưa vào như là kiểu của các object, từ những common interface. Thay vì tạo nó trực tiếp từ new operator một cách cứng nhắn và không mở rộng mô hình được thì factory tạo ra **khả năng mở rộng cao không cần sửa code khi thêm 1 type object mới**.

Ví dụ thực tế: Một công ty có 2 nhánh sản xuất sản phẩm, nhánh thứ nhất chuyên sản xuất những phương tiên giao thông đường bộ, và những nhánh chuyên sản xuất phương tiện giao thông đường thủy, ở đây ta sử dụng mô hình method factory cho mỗi nhánh và gọp 2 nhánh lại ta được một hình abstract factory (factory of factory).

Diagram

Description automatically generated

# Method factory

Ta quan tâm 2 thứ:

+ Cấu trúc lưu trữ sản phẩm, khi thêm mới thì như thế nào.

+ Mapping để get instance khi nhận thông tin từ bên ngoài (ví dụ như type)

Ở đây có 2 nhánh sản xuất có cấu trúc giống nhau, nên ta sẽ đi chi tiết 1 nhánh.

Diagram

Description automatically generated

## Sản phẩm

**interface** **IvehicleRoad**: mô tả phương tiện đường bộ có thể dùng abstract class cũng được.

Trong IvehicleRoad có phương thức createObject sẽ là nơi create new object. Mỗi sản phẩm thêm mới vào sẽ implement interface này và Override method createVehicle để create đúng instance của mình.

Graphical user interface, text, application, chat or text message

Description automatically generated

**3 class implement interface trên**. Cả 3 class đều có cấu trúc tương tự nhau. ở đây bạn sẽ thấy 2 block static (vì sao dung static ở đây mình giải thích ở cuối bài phân [chú ý]). 2 block này có cùng ý nghĩa đều đăng kí this class vào map chứa tất cả các product trong nhánh VehicleRoad. Cụ thể là class Bike này được đăng kí vào quá trình sản xuất của nhà máy này qua 2 block này. Việc sử dụng block đầu ta dùng kĩ thuật reflection còn block 2 là non reflection. Nó chút về reflection ở đây ta có className rồi từ className ta dùng reflection tạo ra instance đơn giản là vậy. Còn block sau dùng non reflection là tạo sẵn instance đưa vào map luôn.

Text

Description automatically generated

Khi sử dụng reflection registry map thì sẽ có lợi như thế này. không cần phải có method createVehicle lúc reflection thì nó sẽ tạo ra new instace luôn.

Text

Description automatically generated

Khi sử dụng non reflection thì map phải chứ instance của class khi cần thì phải call new instance hoặc sử dụng function createVehicle ở trên để tao new instance chứ không dùng instance lưu trong map, vì nếu dùng như vậy chỉ sử dụng 1 instance, không có instance khác.

Text

Description automatically generated

## Registry Mapping Product

Đây là 1 class singleton chỉ tồn 1 instance của chu trình sống. Trong class này quan tâm 3 thứ

+ Một Map chứ all product.

+ hàm addNewProduct: được sử dụng ở 2 block static trên trong các class product.

+ hàm getProduct as same factoryRegistryVehicle khi nhận some information: được call khi user muốn get instance.

Text

Description automatically generated

Một khi class này được tạo ra rồi thì sẽ không thay đổi gì nữa khi có **thêm 1 sản phẩm mới**, mọi thứ được thực hiện 100% ở class product thôi. **Không change** gì ở trong class này.

## Run

Khi sư dụng ta cứ call hàm factoryRegistryVehicle (tương đương hàm get map)

Text

Description automatically generated

# Abstract Factory

Khi đã tìm hiểu run được factory method thì chúng ta đã đi được một nửa hành trình của abstract factory. Chỉ còn làm **abstract** lên hàm registry map method (của factory method) nữa là được.

Lấy ví dụ trước ta có 2 factory method: Một là VehicleRoad và Hai là VehicleWater.

**VehicleRoad** có class singleton RegistryVehicleRoadFactoryReflection chứa map all product của VehicleRoad.

**VehicleWater** có class singleton RegistryVehicleWaterFactoryReflection chứa map all product của VehicleWater.

Nhìn lại cấu trúc: ta đã tạo được 2 registryMap là 2 vòng màu đỏ.

Diagram

Description automatically generated



## Cách tạo

Để hình có thể **hình thành abstract factory** từ các method factory ta sẽ làm **2 điều**:

+ **Create interface** hoặc abstract class chứa function createObject. Những registry map của method factory phải implement interface này, để create object theo đúng type của nó.

Text

Description automatically generated

+ **Create 1 singleTon class** Registry map của abstract factory có map chứa all các registry của factory method.

RegistryMapAbstract -> map chứa all registry method factory.

Text

Description automatically generated

Để đăng kí được vào RegistryMap abstract factory thì các method factory phải có các block static như kiểu mà các product phải làm để đăng kí vào map của factory method giống như diễn tả bên dưới:

Product Bike -> đăng kí với (static block) -> RegistryMap (factory method) -> đăng kí với(static block) -> RegistryMap abstract factory.

Class product Bike đăng kí với RegistryMap (factory method): chỉ 1 block static thôi vì đây mình demo 2 các sử dụng(reflection vs non reflection) nên xài 2 block static.

Text

Description automatically generated

Có được Registry Map method factory đi đăng kí với map của abstract factory. Ví dụ đây là Registry map có type là road.

Text

Description automatically generated

Có được Registry Map method factory đi đăng kí với map của abstract factory. Ví dụ đây là Registry map có type là Water.

Text

Description automatically generated

2 Registry map của method factory sẽ implement hàm create instance. Nó cũng call hàm get instace của map registry map hiện tại thôi.

Text

Description automatically generated

## Run

Hàm factoryVehicleType để get type của AbstractFactory (có 2 registry map method factory implement).

Khi get được registry map theo method factory nào thì cứ truyền information vào và get instance tương ứng.

Text

Description automatically generated

# Chú ý

Ở ví dụ này mình sử dụng 3 registry map trong đó có:

2 Registrymap để chứa các product của vehicle. (RegistryVehicleRoadFactoryReflection, RegistryVehicleWaterFactoryReflection).

1 RegistryMap chứa các registrymap của method factory (2 cái registry ở trên đó). (RegistryFunctionFactoryNonReflection).

Tất cả đều dùng phương thứ là **static block** để tự đăng kí mình với map cách này hay là chúng ta không cần phải chỉnh sửa class registry map khi có thêm product muốn thêm vào nhưng cái bất tiện là mỗi lần run phải **call những class chứa static block** giống như vậy:

Đây là hàm run của abstract factory phải call hết tất cả các class có chứa static block để nó run static block, nếu không sử dụng cách này thì hệ thống sẽ báo **nullPointer** lí do là trong các registrymap không có 1 product nào.

Text

Description automatically generated

Nếu mà không muốn call các forName và static block, thì chỉ còn có cách là **sửa registry map** mỗi lần thêm 1 product hoặc type nào mới vào ứng dụng, thì phải push cứng vào map thôi.

# Kết luận

Cốt lõi của factory này quay quanh việc.

Sản phẩm (Interface include some classes product).

Map (include all interface prodcut and function add, getting map return new instance)

# Reference:

<https://www.oodesign.com/>

<https://www.baeldung.com/java-abstract-factory-pattern>