

Design patter factory

Contents

Overiview	2
Mục đích	2
Method factory	3
Sản phẩm	3
Registry Mapping Product	5
Run	6
Abstract Factory	6
Cách tạo	7
Run	9
Chú ý	10
Kết luận	10
Reference:	11

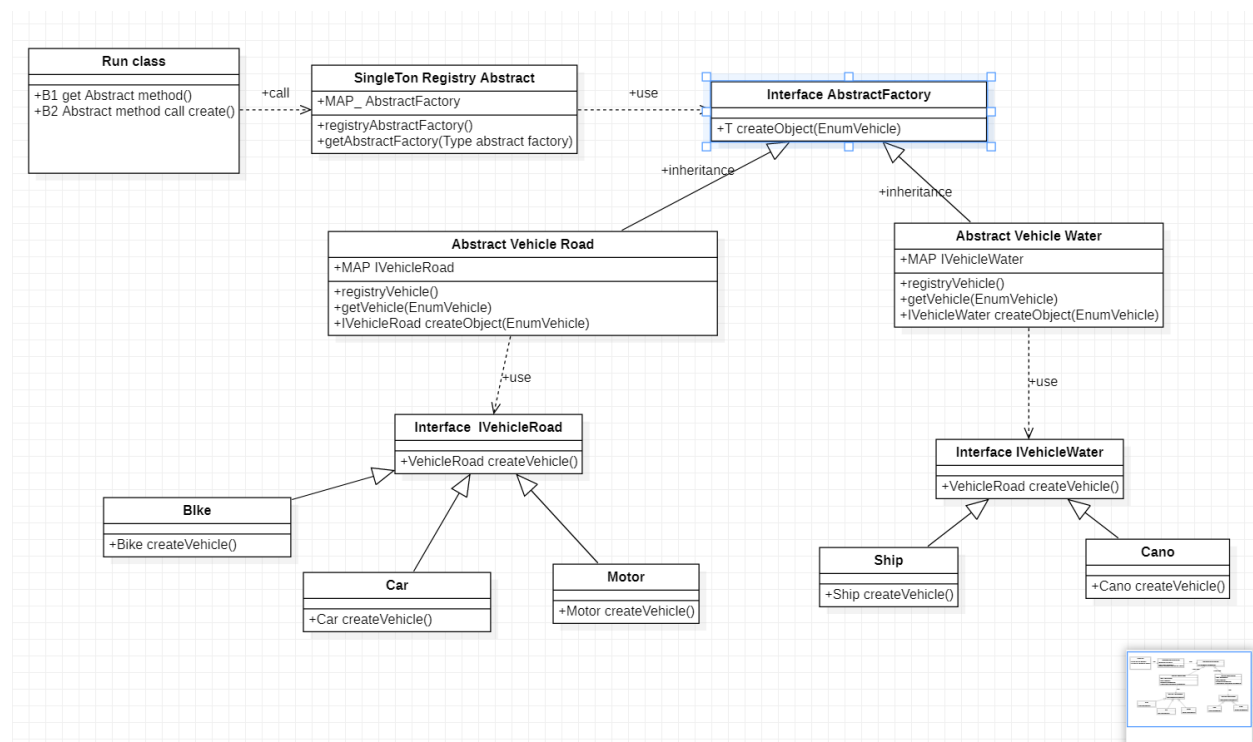
Overview

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 nhắc 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 tiện 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 gộp 2 nhánh lại ta được một hình abstract factory (factory of factory).

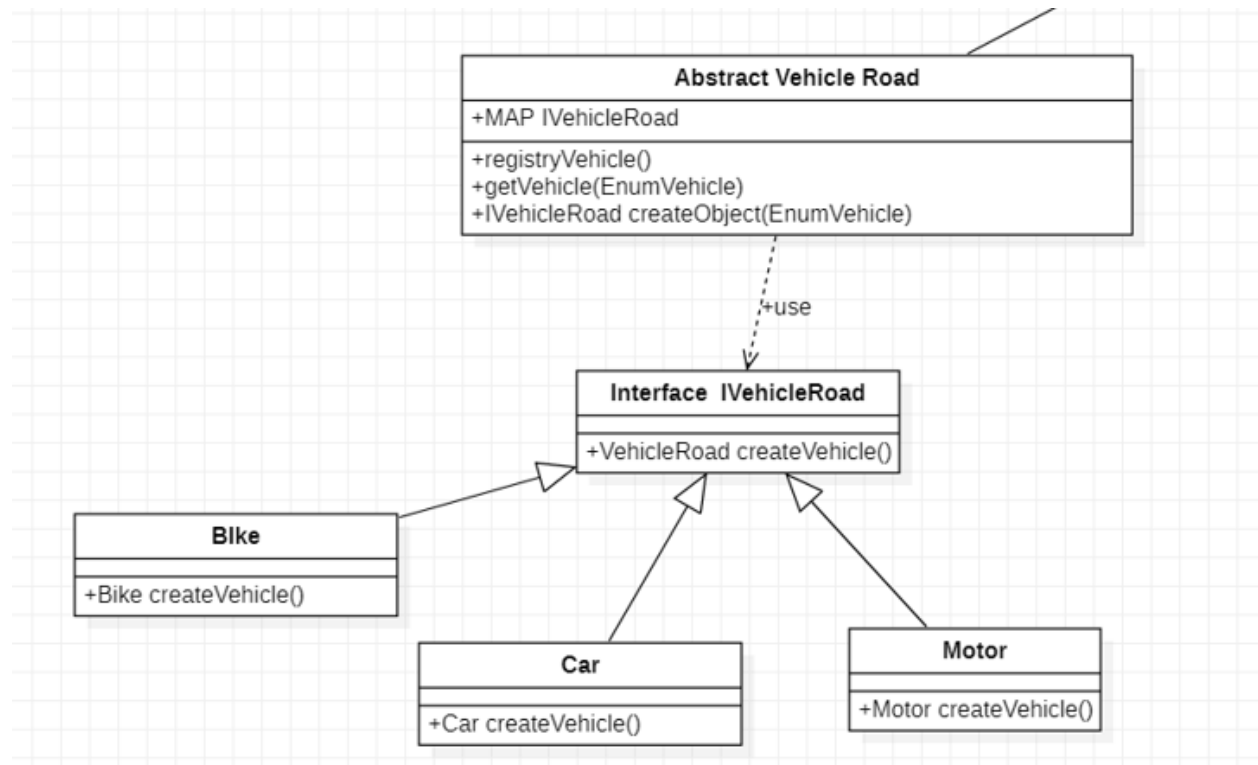


Method factory

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.



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.

```
public interface IvehicleRoad {

    IvehicleRoad createVehicle();

    String getType();

}
```

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.

```
7 public class Bike implements VehicleRoad {
8
9     // registry factory reflection
10    static {
11        RegistryVehicleRoadFactoryReflection.getInstance().addNewVehicle(EnumVehicle.BIKE, Bike.class);
12    }
13
14    // registry factory not reflection
15    static {
16        RegistryFactoryAvoidReflection.getInstance().addNewVehicle(EnumVehicle.BIKE, new Bike());
17    }
18
19    @Override
20    public VehicleRoad createVehicle() {
21        return new Bike();
22    }
23
24
25    @Override
26    public String getType() { return "Bike"; }
27
28 }
29
30
```

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 instance luôn.

```
37
38 // get instance vehicle
39 public VehicleRoad factoryRegistryVehicle(EnumVehicle enumVehicle){
40     try {
41         //Reflection class to instance
42         Class<VehicleRoad> classVehicle = MAP_REGISTRY.get(enumVehicle);
43         Constructor<VehicleRoad> productConstructor = classVehicle.getDeclaredConstructor();
44         return productConstructor.newInstance();
45     }catch( Exception e) {
46         return null;
47     }
48 }
49
```

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

```
26
27 // get instance vehicle
28 public VehicleRoad factoryRegistryVehicle(EnumVehicle enumVehicle) {
29     VehicleRoad classVehicle = MAP_REGISTRY.get(enumVehicle);
30     if (classVehicle != null) {
31         return classVehicle.createVehicle();
32     }
33     return null;
34 }
35
```

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 chứa 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.

```
RegistryVehicleRoadFactoryReflection.java
private static RegistryVehicleRoadFactoryReflection registryFactoryConcrete;

29
30 // map save vehicle
31 private final Map<EnumVehicle, Class<VehicleRoad>> MAP_REGISTRY = new HashMap<>();
32
33 // add new vehicle when new class type vehicle
34 public void addNewVehicle(EnumVehicle enumVehicle, Class classVehicle){
35     MAP_REGISTRY.put(enumVehicle, classVehicle);
36 }
37
38 // get instance vehicle
39 public VehicleRoad factoryRegistryVehicle(EnumVehicle enumVehicle){
40     try {
41         //Reflection class to instance
42         Class<VehicleRoad> classVehicle = MAP_REGISTRY.get(enumVehicle);
43         Constructor<VehicleRoad> productConstructor = classVehicle.getDeclaredConstructor();
44         return productConstructor.newInstance();
45     }catch( Exception e) {
46         return null;
47     }
48 }
49
50 public static synchronized RegistryVehicleRoadFactoryReflection getInstance(){
51     if(registryFactoryConcrete == null){
52         return registryFactoryConcrete = new RegistryVehicleRoadFactoryReflection();
53     }
54     return registryFactoryConcrete;
55 }
56
```

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 sử dụng ta cứ call hàm `factoryRegistryVehicle` (tương đương hàm `get map`)

```
Run.java
22
23 public static void main(String[] args) {
24     // use factory registry Reflection
25     VehicleRoad vehicle2 = RegistryVehicleRoadFactoryReflection.getInstance().factoryRegistryVehicle(EnumVehicle.BIKE);
26     System.out.println(vehicle2.toString());
27
28     VehicleRoad vehicle3 = RegistryVehicleRoadFactoryReflection.getInstance().factoryRegistryVehicle(EnumVehicle.BIKE);
29     System.out.println(vehicle3.toString());
30
31     // use factory registry AvoidReflection
32     VehicleRoad vehicle32 = RegistryFactoryAvoidReflection.getInstance().factoryRegistryVehicle(EnumVehicle.BIKE);
33     System.out.println(vehicle32.toString());
34
35     VehicleRoad vehicle342 = RegistryFactoryAvoidReflection.getInstance().factoryRegistryVehicle(EnumVehicle.BIKE);
36     System.out.println(vehicle342.toString());
37
38 }
39
40 }
```

Run (2) x

D:\Document\openjdk-11\28_windows-x64_bin\jdk-11\bin\java.exe "-javaagent:D:\Tool\IntelliJ IDEA Community Edition 2021.1.2\lib\pattern.creation.factory.fucntionfactoryvehicleroad.model.Bike@39ba5a14;pattern.creation.factory.fucntionfactoryvehicleroad.model.Bike@511baa65;pattern.creation.factory.fucntionfactoryvehicleroad.model.Bike@340f438e;pattern.creation.factory.fucntionfactoryvehicleroad.model.Bike@30c7da1e

Abstract Factory

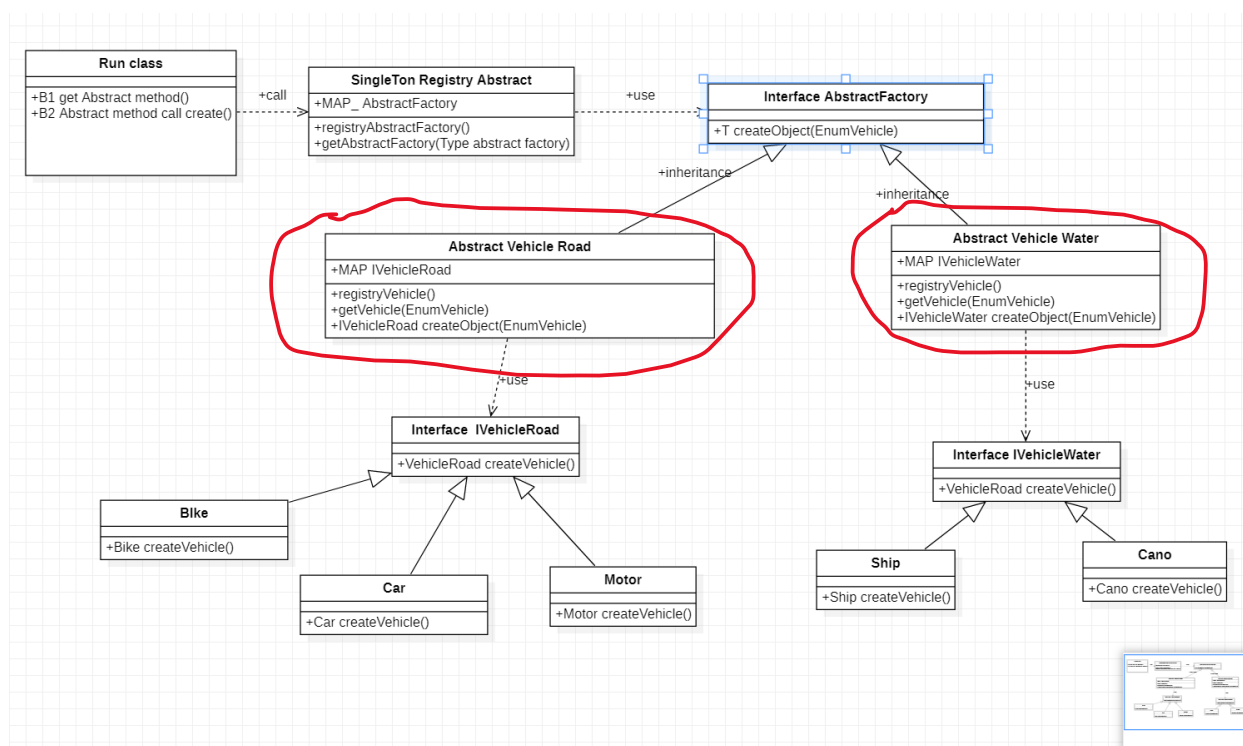
Khi đã tìm hiểu run được factory method thì chúng ta đã đi được một nửa hành trình của abstract factory. Chỉ còn làm **abstract** lên hàm registry map method (của factory method) nữa là được.

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.



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.

```
1 package pattern.creational.factory.astractfactory.main;
2
3 import pattern.creational.factory.fuctionfactoryvehicleroad.model.EnumVehicle;
4
5 public interface AbstractFactory<T> {
6     T createObject(EnumVehicle enumVehicle);
7 }
8
```

+ **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.

```
7 public class RegistryFunctionFactoryNonReflection {
8
9     private static RegistryFunctionFactoryNonReflection registryFunctionFactory;
10
11     private Map<String, AbstractFactory> MAP_ABSTRACT = new HashMap<>();
12
13
14     public void addNewVehicle(String typeVihicle, AbstractFactory classVehicle){
15         MAP_ABSTRACT.put(typeVihicle, classVehicle);
16     }
17
18     public AbstractFactory getFactoryMethod(String enumVehicle){
19         try {
20             return MAP_ABSTRACT.get(enumVehicle);
21         } catch (Exception e) {
22             return null;
23         }
24     }
25
26     public static synchronized RegistryFunctionFactoryNonReflection getInstance(){
27         if(registryFunctionFactory == null){
28             return registryFunctionFactory = new RegistryFunctionFactoryNonReflection();
29         }
30         return registryFunctionFactory;
31     }
32 }
```

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.

```

Bike.java x
3
4 import pattern.creational.factory.functionfactoryvehicleroad.RegistryFactoryAvoidReflection;
5 import pattern.creational.factory.functionfactoryvehicleroad.RegistryVehicleRoadFactoryReflection;
6
7 public class Bike implements VehicleRoad {
8
9     // registry factory reflection
10    static {
11        RegistryVehicleRoadFactoryReflection.getInstance().addNewVehicle(EnumVehicle.BIKE, Bike.class);
12    }
13
14    // registry factory not reflection
15    static {
16        RegistryFactoryAvoidReflection.getInstance().addNewVehicle(EnumVehicle.BIKE, new Bike());
17    }
18

```

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

```

Bike.java x RegistryVehicleRoadFactoryReflection.java x
13 /**
14  * đăng kí với class cách 1
15  * Singleton Class Registration - using reflection
16  * mỗi type sản phẩm thêm mới sẽ tự đăng ký với class này không cần phải edit gì class này khi thêm 1 loại mới
17  */
18 public class RegistryVehicleRoadFactoryReflection implements AbstractFactory<VehicleRoad> {
19
20     // registry factory reflection with abstract factory
21     static {
22         RegistryFunctionFactoryNonReflection.getInstance().addNewVehicle( typeVehicle: "road", getInstance());
23     }
24
25     //private constructor to avoid client applications to use constructor
26     private RegistryVehicleRoadFactoryReflection(){}
27
28     private static RegistryVehicleRoadFactoryReflection registryFactoryConcrete;

```

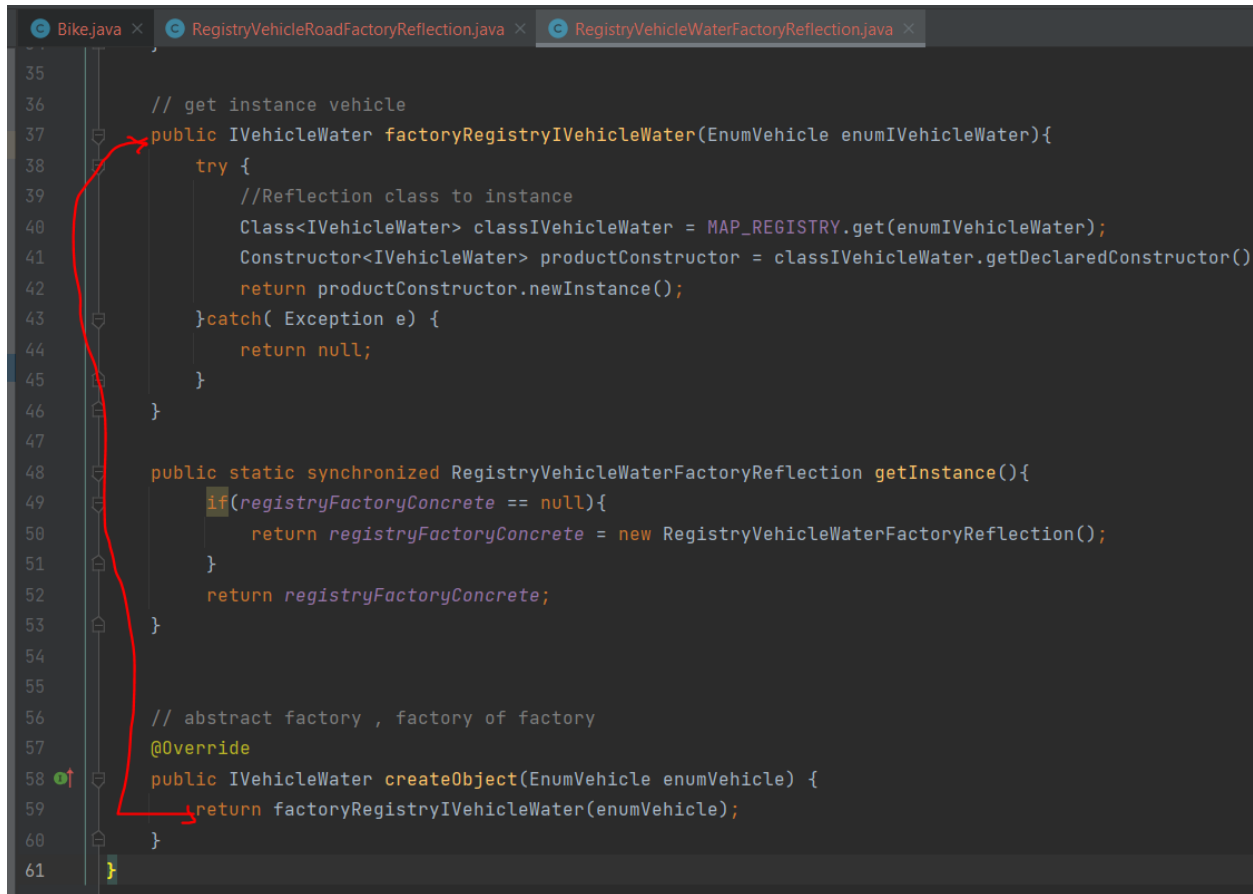
Có được Registry Map method factory đi đăng kí với map của abstract factory. Ví dụ đây là Registry map có type là Water.

```

Bike.java x RegistryVehicleRoadFactoryReflection.java x RegistryVehicleWaterFactoryReflection.java x
1 package pattern.creational.factory.abstractfactory.functionfactoryvehicleground;
2
3 import ...
4
5 public class RegistryVehicleWaterFactoryReflection implements AbstractFactory<IVehicleWater> {
6
7     // registry factory reflection with abstract factory
8     static {
9         RegistryFunctionFactoryNonReflection.getInstance().addNewVehicle( typeVehicle: "water", getInstance());
10    }
11
12    //private constructor to avoid client applications to use constructor
13    private RegistryVehicleWaterFactoryReflection(){}

```


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

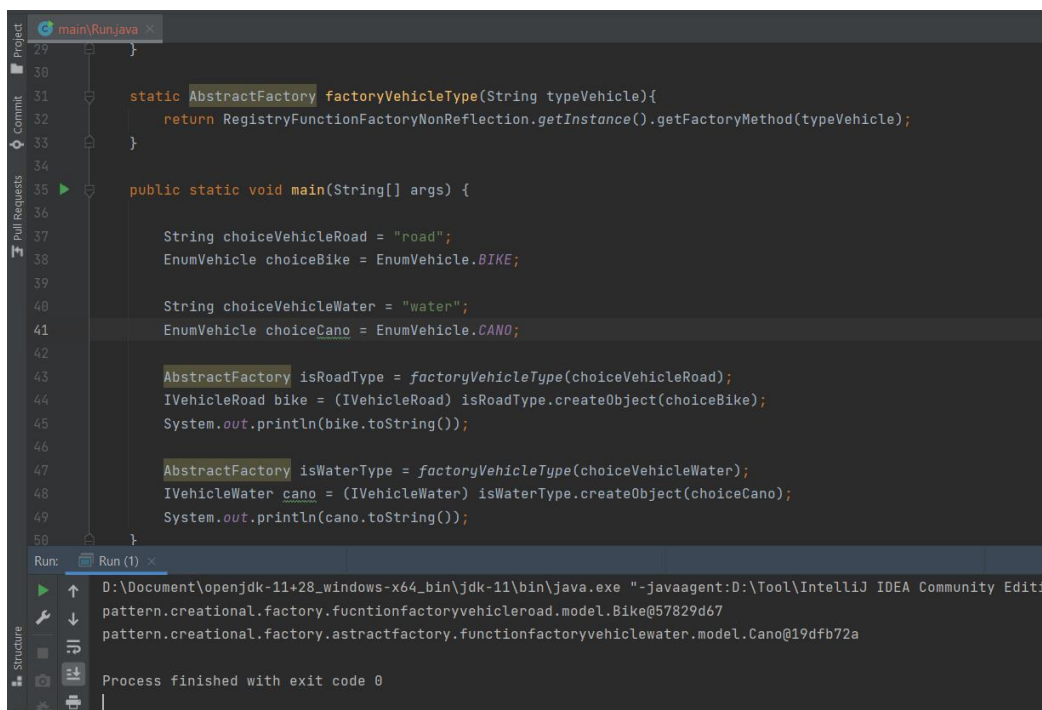


```
35
36 // get instance vehicle
37 public IVehicleWater factoryRegistryIVehicleWater(EnumVehicle enumIVehicleWater){
38     try {
39         //Reflection class to instance
40         Class<IVehicleWater> classIVehicleWater = MAP_REGISTRY.get(enumIVehicleWater);
41         Constructor<IVehicleWater> productConstructor = classIVehicleWater.getDeclaredConstructor()
42         return productConstructor.newInstance();
43     }catch( Exception e) {
44         return null;
45     }
46 }
47
48 public static synchronized RegistryVehicleWaterFactoryReflection getInstance(){
49     if(registryFactoryConcrete == null){
50         return registryFactoryConcrete = new RegistryVehicleWaterFactoryReflection();
51     }
52     return registryFactoryConcrete;
53 }
54
55
56 // abstract factory , factory of factory
57 @Override
58 public IVehicleWater createObject(EnumVehicle enumVehicle) {
59     return factoryRegistryIVehicleWater(enumVehicle);
60 }
61 }
```

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.



```
29
30
31 static AbstractFactory factoryVehicleType(String typeVehicle){
32     return RegistryFunctionFactoryNonReflection.getInstance().getFactoryMethod(typeVehicle);
33 }
34
35 public static void main(String[] args) {
36
37     String choiceVehicleRoad = "road";
38     EnumVehicle choiceBike = EnumVehicle.BIKE;
39
40     String choiceVehicleWater = "water";
41     EnumVehicle choiceCano = EnumVehicle.CANO;
42
43     AbstractFactory isRoadType = factoryVehicleType(choiceVehicleRoad);
44     IVehicleRoad bike = (IVehicleRoad) isRoadType.createObject(choiceBike);
45     System.out.println(bike.toString());
46
47     AbstractFactory isWaterType = factoryVehicleType(choiceVehicleWater);
48     IVehicleWater cano = (IVehicleWater) isWaterType.createObject(choiceCano);
49     System.out.println(cano.toString());
50 }
```

Run: Run (1) ×

D:\Document\openjdk-11+28_windows-x64_bin\jdk-11\bin\java.exe "-javaagent:D:\Tool\IntelliJ IDEA Community Edit\pattern.creationl.factory.fucntionfactoryvehicleroad.model.Bike@57829d67 pattern.creationl.factory.stractfactory.functionfactoryvehicllewater.model.Cano@19dfb72a

Process finished with exit code 0

Chú ý

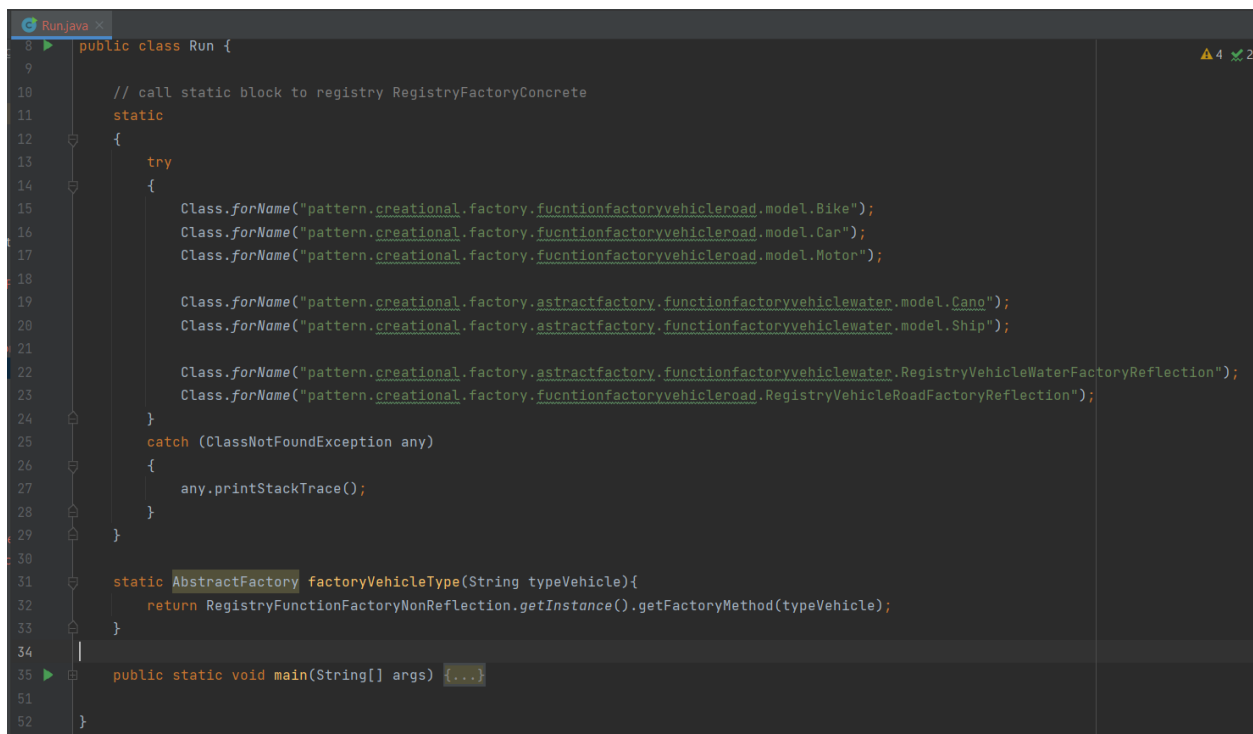
Ở ví dụ này mình sử dụng 3 registry map trong đó có:

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 thức 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.



```
8 public class Run {
9
10     // call static block to registry RegistryFactoryConcrete
11     static
12     {
13         try
14         {
15             Class.forName("pattern.creational.factory.fuctionfactoryvehicleroad.model.Bike");
16             Class.forName("pattern.creational.factory.fuctionfactoryvehicleroad.model.Car");
17             Class.forName("pattern.creational.factory.fuctionfactoryvehicleroad.model.Motor");
18
19             Class.forName("pattern.creational.factory.astractfactory.functionfactoryvehicleground.model.Cano");
20             Class.forName("pattern.creational.factory.astractfactory.functionfactoryvehicleground.model.Ship");
21
22             Class.forName("pattern.creational.factory.astractfactory.functionfactoryvehicleground.RegistryVehicleWaterFactoryReflection");
23             Class.forName("pattern.creational.factory.fuctionfactoryvehicleroad.RegistryVehicleRoadFactoryReflection");
24         }
25         catch (ClassNotFoundException any)
26         {
27             any.printStackTrace();
28         }
29     }
30
31     static AbstractFactory factoryVehicleType(String typeVehicle){
32         return RegistryFunctionFactoryNonReflection.getInstance().getFactoryMethod(typeVehicle);
33     }
34
35     public static void main(String[] args) {...}
36
37 }
```

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 product and function add, getting map return new instance)

Reference:

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

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