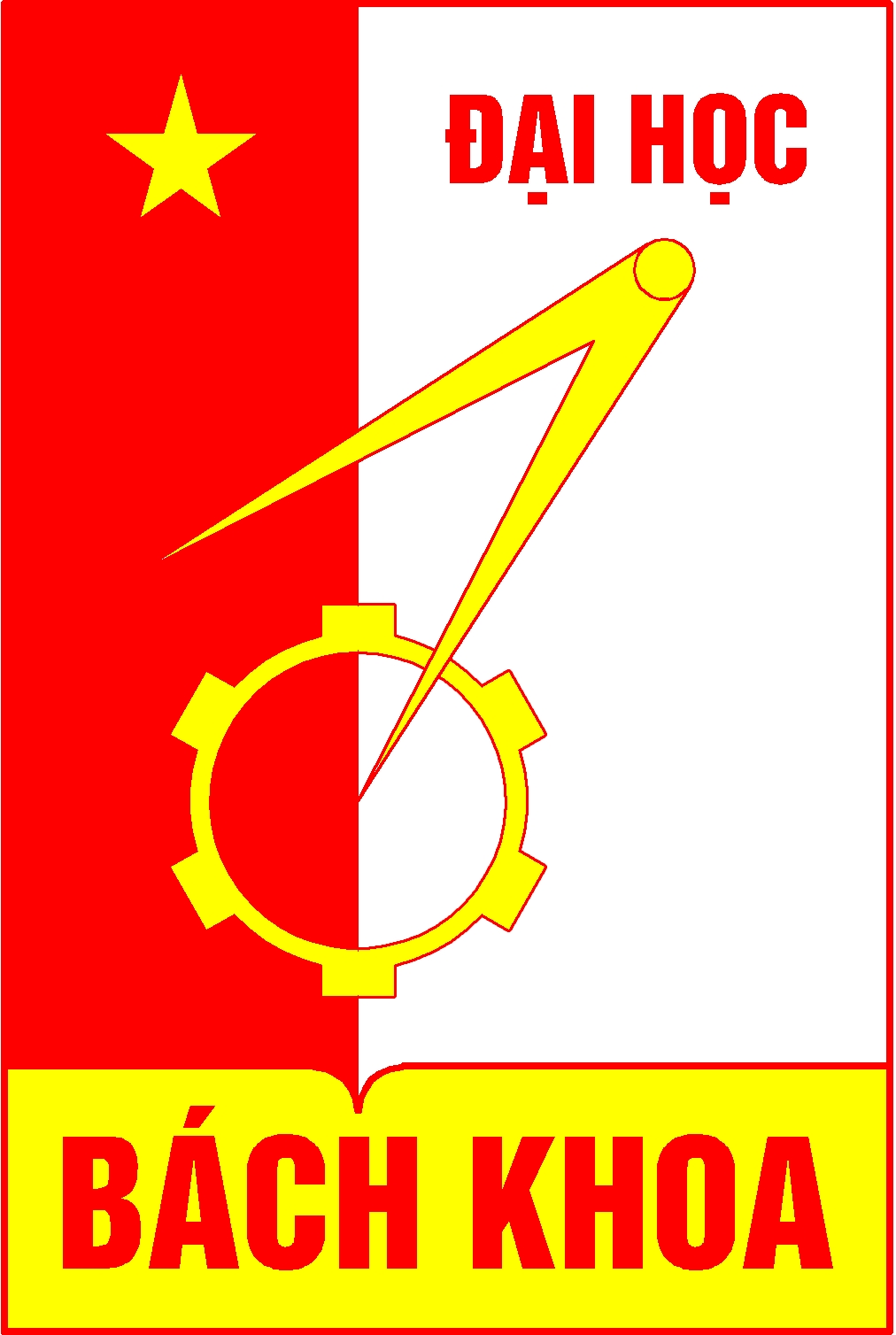
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**



***BÀI TẬP TUẦN***

**TÌM HIỂU MẪU THIẾT KẾ FACTORY METHOD**

|  |  |  |
| --- | --- | --- |
| **Giảng viên hướng dẫn:** | TS. Nguyễn Thị Thu Trang |  |
| **Nhóm:** | 5 |  |
| **Sinh viên thực hiện:** | Nguyễn Văn Đạt | 20154484 |

***Hà Nội, tháng 12 năm 2018***

Mục lục

[1. Giới thiệu 3](#__RefHeading___Toc230_428640332)

[2. Định nghĩa 3](#__RefHeading___Toc232_428640332)

[3. Bản chất của Factory method 3](#__RefHeading___Toc234_428640332)

[4. Lợi ích của việc dùng Factory method 3](#__RefHeading___Toc236_428640332)

[5. Khi nào dùng Factory method 3](#__RefHeading___Toc460_428640332)

[6. Ví dụ 4](#__RefHeading___Toc240_428640332)

[7. Tài liệu tham khảo 7](#__RefHeading___Toc242_428640332)

# Giới thiệu

Factory method là một pattern cho việc khởi tạo đối tượng(thuộc nhóm creational patterns). Pattern này được sinh ra nhằm mục đích khởi tạo đối tượng mà bản thân muốn che giấu class nào được khởi tạo. Pattern này được sử dụng khá phổ biến đồng thời nó cũng không khó khăn để hiểu.

# Định nghĩa

Factory method, đầy đủ là Factory method pattern, là thiết kế mẫu hướng đối tượng trong việc thiết kế phần mềm cho máy tính, nhằm giải quyết vấn đề tạo một đối tượng mà không cần thiết chỉ ra một cách chính xác lớp nào sẽ được tạo. Factory method giải quyết vấn đề này bằng cách định nghĩa một phương thức cho việc tạo đối tượng, và các lớp con thừa kế có thể override để chỉ rõ đối tượng nào sẽ được tạo. Nói chung, "factory method" thường được áp dụng cho những phương thức mà nhiệm vụ chính của nó là tạo ra đối tượng. (Nguồn: Wikipedia )

# Bản chất của Factory method

Về cơ bản thì ta sẽ định nghĩa một **interface** hoặc **Abstract** **class** , các class con sẽ implements nó. Tiếp đến mình sẽ tạo một class được xem là FactoryClass, bên trong FactoryClass này mình sẽ có một phương thức giúp chúng ta khởi tạo các Object chúng ta cần.

# Lợi ích của việc dùng Factory method

Factory pattern đưa ra 1 ý tưởng mới cho việc khởi tạo các instance phù hợp với mỗi request từ phía Client. Sử dụng Factory pattern sẽ có những ưu điểm sau:

* Tạo ra 1 cách mới trong việc khởi tạo cá Object thông qua 1 interface chung.
* Khởi tạo các Objects mà che giấu đi xử lí logic của việc khởi tạo đấy.
* Giảm sự phụ thuộc giữa các module, các logic với các class cụ thể, mà chỉ phụ thuộc vào interface hoặc abstract class.

# Khi nào dùng Factory method

Dựa vào lợi ích của việc sử dụng Factory pattern mà ta sẽ dùng chúng với một số mục đích sau:

* Tạo ra 1 cách mới trong việc khởi tạo Object.
* Che giấu xử lý logic của việc khởi tạo.
* Giảm sự phụ thuộc.

Khi sử dụng Factory method việc hiển nhiên các tạo một object là một việc hoàn toàn mới bằng cách sử dụng interface và các class khởi tạo implements interface đó. Tiếp đến thứ 2 là việc che dấu thông tin, thì trong trường hợp mà bạn muốn viết một thư viện để người khác sử dụng thì factory method là hoàn toàn hợp lý khi các class khởi tạo sẽ implements interface mà bạn tạo sẵn để sử dụng và các phương thức trong đó mà không quan tâm và không biết quá trình khởi tạo sẽ như thế nào. Cuối cùng là về tính phụ thuộc thì rõ ràng factory method đã làm giảm đáng kể tính phụ thuộc giữa bài toán và cách xử lý như nếu bạn không biết chắc được số lượng đối tượng của bài toán bạn đưa ra đã đủ hay chưa? thì factory method hoàn toàn có thể dễ dàng hỗ trợ bạn mở rộng vì phần khởi tạo phía trên không cần phải sửa hay thêm bớt gì thêm mà chỉ cần bạn chỉnh sửa ở các phần phía dưới. Các class bên dưới implements interface khởi tạo ở phía trên.

# Ví dụ

* Về code thì ta lấy ví dụ chọn một cái điện thoại theo các hãng khác nhau và khởi tạo nó:
* Đầu tiên ta định nghĩa một interface Phone để các class sau có thể implements:

public interface Phone {

public void showInfo();

}

* Tiếp theo là khai báo các Class như SamsungPhone, ApplePhone, NokiaPhone

public class SamsungPhone implements Phone {

@Override

public void showInfo() {

System.out.printf("Đây là điện thoại Samsung");

}

}

public class ApplePhone implements Phone {

@Override

public void showInfo() {

System.out.printf("Đây là điện thoại Apple");

}

}

public class NokiaPhone implements Phone {

@Override

public void showInfo() {

System.out.printf("Đây là điện thoại Nokia");

}

}

public enum PhoneType {

SAMSUNG, NOKIA, APPLE

}

* Tiếp nữa là tạo FactoryClass có tên là PhoneFactory

public class PhoneFactory {

public Phone getPhone(PhoneType phoneType) {

Phone phoneCreated = null;

switch (phoneType) {

case SAMSUNG:

phoneCreated = new SamsungPhone();

break;

case APPLE:

phoneCreated = new ApplePhone();

break;

case NOKIA:

phoneCreated = new NokiaPhone();

break;

}

return phoneCreated;

}

}

* Cuối cùng là chạy demo

public class RunDemo {

public static void main(String[] args) {

PhoneFactory phoneFactory = new PhoneFactory();

Phone phone = phoneFactory.getPhone(PhoneType.SAMSUNG);

phone.showInfo();

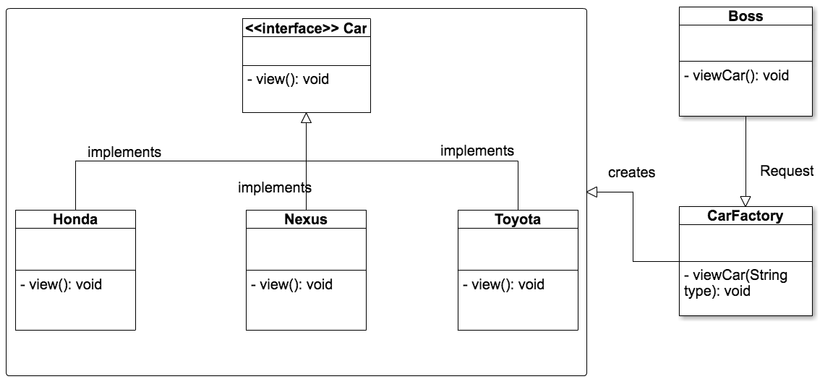
}

}

Và kết quả có được là:

Samsung phone

* Ví dụ tiếp là về biểu đồ các class

 Ở hình trên ta có thể thấy có 3 Class Honda, Nexus và Toyota đều được implements từ Class Car. Cả 3 class này đều là con của class Car nên nó thừa hưởng tất cả các tính chất của Car và phải override lại phương thức view() của Car. Tuy nó có chung phương thức nhưng chúng ta có thể định nghĩa khác nhau cho nó. Hiểu đơn giản là bạn đến cửa hàng Honda để bạn xem xe của hãng Honda chứ không phải xem của Nexus hay Toyota. Và 2 cửa hàng kia cũng vậy. Cho nên Chúng ta phải viết nội dung của phương thức view() khác nhau cho từng Class con.

Tưởng tượng class Boss chính là bạn. Bạn muốn đi xem các hãng ô tô khác nhau mà không muốn đi tới từng cửa hàng khác nhau. Vậy thì phải làm thế nào? Đừng lo, đã có CarFactory. CarFactory đơn giản giống như là 1 cửa hàng có bán nhiều loại xe, bạn chỉ việc tới đó và xem các loại ô tô khác nhau mà không cần phải đi lại nhiều.

# Tài liệu tham khảo

* https://viblo.asia/p/design-pattern-factory-pattern-part-2-XQZGxZqjkwA
* https://viblo.asia/p/factory-method-pattern-trong-java-4dbZNoNQlYM
* https://vi.wikipedia.org/wiki/Factory\_method