Creational Pattern

1. Tìm hiểu về mục đích, cách thức triển khai, ưu điểm của:

**Signleton**

Single Pattern là một design pattern mà:

- Đảm bảo rằng một class chỉ có duy nhất một instance

- Và cung cấp một cách toàn cầu để truy cấp tới instance đó.

Mục đích của **Single Pattern** là đảm bảo rằng một lớp chỉ có **duy nhất một thể hiện (instance)** trong suốt vòng đời của ứng dụng và cung cấp một điểm truy cập toàn cục tới thể hiện đó.

**Cách sử dụng:**

* Đặt constructor là private để class khác không thể khởi tạo object của class
* Tạo một biến static private là instance của class đó để đảm bảo rằng nó là duy nhất và chỉ được tạo ra trong class đó thôi
* Tạo một public static method trả về instance vừa khởi tạo bên trên, đây là cách duy nhất để các class khác có thể truy cập vào instance của class này

public class Singleton {

private static Singleton instance;

private Singleton() {

}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

public void showMessage() {

System.out.println("Đây là một đối tượng Singleton.");

}

}

**Ưu điểm:**

* Có thể chắc chắn rằng một lớp chỉ có một instance
* Có khả năng truy cập đến instance từ mọi nơi (global access)
* Đối tượng singleton chỉ được khởi tạo duy nhất một lần khi nó được yêu cầu lần đầu.
* Kiểm soát truy cập đến instance duy nhất
* Giảm namespace bằng cách không tạo ra nhiều đối tượng cho cùng một mục đích

**Nhược điểm:**

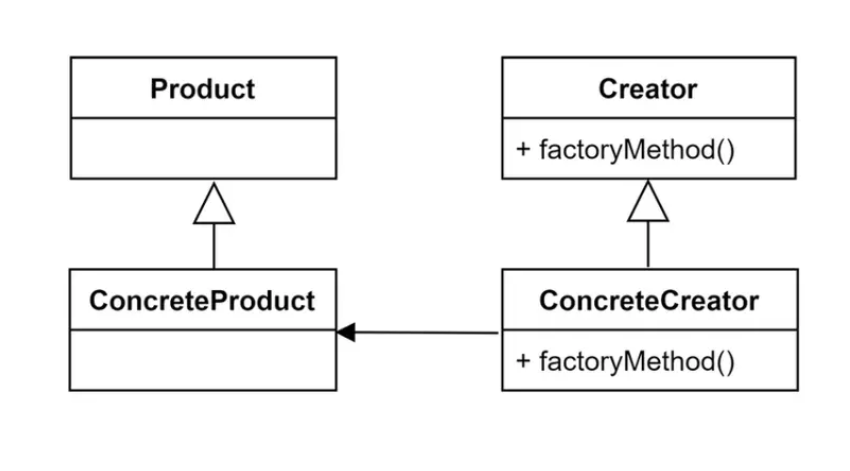
* Vi phạm Single Responsibility Principle. Mẫu này giải quyết hay vấn đề trên cùng một thời điểm.
* Singleton pattern có thể thể hiện thiết kế kém (bad design), chẳng hạn, khi các thành phần của chương trình biết quá nhiều về nhau.
* Có thể sinh ra khó khăn trong việc unit test client code của Singleton bởi nhiều test frameworks dựa vào kế thừa khi sản sinh mock objects.

**Factory method**

**Factory Method** cung cấp một interface, phương thức trong việc tạo nên một đối tượng (object) trong class. Nhưng để cho class con kế thừa của nó có thể ghi đè để chỉ rõ đối tượng (object) nào sẽ được tạo. Factory method giao việc khởi tao một đối tượng (object) cụ thế cho lớp con (subclass)

**Mục đích:**

* Tạo ra một cách khởi tạo object mới thông qua một interface chung
* Che giấu quá trình xử lý logic của phương thức khởi tạo
* Giảm sự phụ thuộc, dễ dàng mở rộng
* Giảm khả năng gây lỗi compile



**Cách thức triển khai**

Factory Method Pattern thường bao gồm 4 thành phần chính:

* **Product:** Một giao diện hoặc lớp trừu tượng định nghĩa các phương thức chung cho các đối tượng được tạo ra.
* **ConcreteProduct:** Các lớp cụ thể triển khai giao diện Product. Đây là các đối tượng thực tế sẽ được tạo ra.
* **Creator:** Một lớp trừu tượng hoặc giao diện khai báo phương thức factory (factoryMethod) để tạo ra các đối tượng Product.
* **ConcreteCreator:** Các lớp cụ thể kế thừa từ Creator và ghi đè phương thức factory để trả về một đối tượng ConcreteProduct cụ thể.

**Factory method được sử dụng khi:**

* Có một super class với nhiều class con và dựa trên đầu vào, chúng ta cần trả về một class con. Mô hình này giúp chúng ta đưa trách nhiệm của việc khởi tạo một lớp từ phía người dùng (client) sang lớp Factory, giúp tiết kiệm tài nguyên hệ thống vì nhờ vào việc tái sử dụng các object đã có thay vì xây dựng lại mỗi phần có thêm product
* Chúng ta không biết sau này sẽ cần đến những lớp con nào nữa. Khi cần mở rộng, hãy tạo ra sub class và implement thêm vào factory method cho việc khởi tạo sub class này.