**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN ĐIỆN TỬ - VIỄN THÔNG  
\*\*\*\*\* 🙠🕮🙢 \*\*\*\*\***



**ĐỒ ÁN I  
Bài tìm hiểu về Desgin Pattern**

|  |  |  |
| --- | --- | --- |
| Giáo viên hướng dẫn | : | PGS.TS.Nguyễn Đức Minh |
| Nhóm 3: |  |  |
| *Đào Đức Dũng   Nguyễn Đức Thái  Nguyễn Văn Đạt* |  |  |
|  |  |  |
|  |  |  |

**Hà Nội, tháng 4-2020**

**Mục lục**

[Phần 1: Mục đích của Design pattern factory 3](#_Toc37494271)

[Phần 2: Ví dụ sử dụng design pattern factory 4](#_Toc37494272)

[Phần 3: Sơ đồ cấu trúc UML cho shape, Edge, ShapeFactory, main() 8](#_Toc37494273)

# 

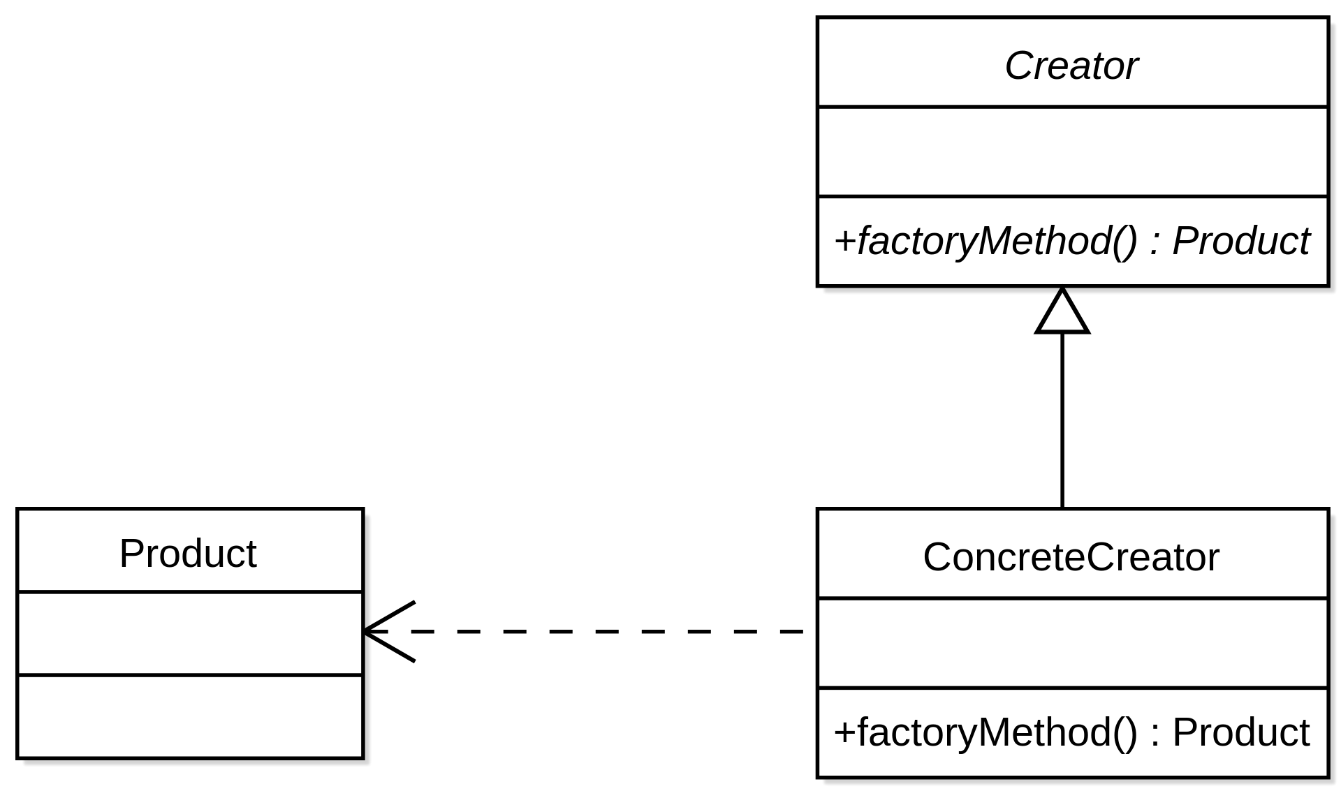
# Phần 1: Mục đích của Design pattern factory

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 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. Method factory thường được áp dụng cho những phương thức mà nhiệm vụ chính của nó là tạo ra đối tượng.

*- Mục đích của Factory method:* Định nghĩa interface để sinh ra đối tượng nhưng cho lớp con quyết định lớp nào được sinh ra đối tượng

- *Cách sử dụng Factory method:* pattern này giúp thay thế việc khởi tạo đối tượng một cách trực tiếp (sử dụng toán tử new) bằng việc goi đến một factory method. Về thực chất, toán tử new vẫn được sử dụng để tạo ra đối tượng mới, nhưng việc thực hiện lời gọi này là do phương thức kia đảm nhận. Các đối tượng mới được tạo ra nhờ factory method còn được gọi là **“product”**. Từ đó chúng ta có thể override factory method này để chỉ rõ đối tượng nào sẽ được khởi tạo.

- *Sơ đồ mô tả cấu trúc UML của Factory method:*



# Phần 2: Ví dụ sử dụng design pattern factory

*Đề bài:* Ứng dụng quản lý logistics mang tên LogisticsApp. Ứng dụng này quản lý vận tải hàng hóa bằng đường bộ thông qua hệ thống xe tải chuyên dụng (Truck) hoặc qua hệ thống tàu chuyên dụng (Ship).

Giải quyết vấn đề:

Đầu tiên, chúng ta nhận thấy rằng cả Truck và Ship đều có những điểm chung sau:

* Cùng là các phương tiện
* Cùng là vận chuyển hàng hóa

Do đó chúng ta sẽ tạo ra interface **Transport,** interface này chứa phương thức **deliver()** bắt buộc bất kì class nào implements từ nó cũng đều phải override lại.

                interface Transport

                {

                      public function deliver(): string;

                }

Tiếp theo chúng ta sẽ tiếp tục tạo ra 2 concrete product class là Truck và Ship. Cả 2 class này đều cùng implements interface **Transport** ở trên, do đó chúng đề phải override lại phương thức **deliver()**:

class Truck implements Transport

{

    public function deliver(): string

    {

*// Truck vận chuyển hàng bên trong các thùng và bằng đường bộ*

          return “Truck: Deliver by land in a box.”;

    }

}

class Ship implements Transport

{

    public function deliver(): string

    {

*// Ship vận chuyển hàng bên trong các container và bằng đường biển*

          return “Ship: Deliver by sea in a container.”;

    }

}

Tiếp đến, chúng ta sẽ tạo ra creator class mang tên **Logistics**, nhiệm vụ chính của class này là cung cấp phương thức abstract mang tên **createTransport** cho phép trả về một đối tượng có kiểu dữ liệu là Transport. Nội dung của phương thức này sẽ do các concrete creator class định nghĩa chi tiết.

{

abstract class Logistics

{

        public function planDelivery(string $transportType): string

        {

                 $transport = $this->createTransport($transportType);

                 echo $transport->deliver();

        }

        abstract public function createTransport(string $transporType): Transport;

}

Chúng ta tiếp tục tạo ra 2 concrete creator class là **RoadLogistics** và **SeaLogistics** với nội dung như sau:

class RoadLogistics extends Logistics

{

      public function createTransport(string $transportType)

      {

                if(strtolower($transportType) === strtolower(“Truck”)) {

                        return new Truck;

                }

      }

}

class SeaLogistics extends Logistics {

      public function createTransport(string $transportType)

      {

                if(strtolower($transportType) === strtolower(“Ship”)) {

                        return new Ship;

                }

    }

}

Sau đó, chạy thử kết quả:

$roadLogistics = new RoadLogistics;

$seaLogistics = new SeaLogistics;

$transport = $roadLogistics->planDelivery(“Truck”);

$transport->delivery();

echo “\n”;

$transport = $seaLogistics->planDelivery(“Ship”);

$transport->delivery();

Kết quả thu được:

Truck: Deliver by land in a box.

Ship: Deliver by sea in a container.

# Phần 3: Sơ đồ cấu trúc UML cho shape, Edge, ShapeFactory, main()

\*Để bên trong folder Code\_4 trên Teams