POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

14. *Factory Method*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

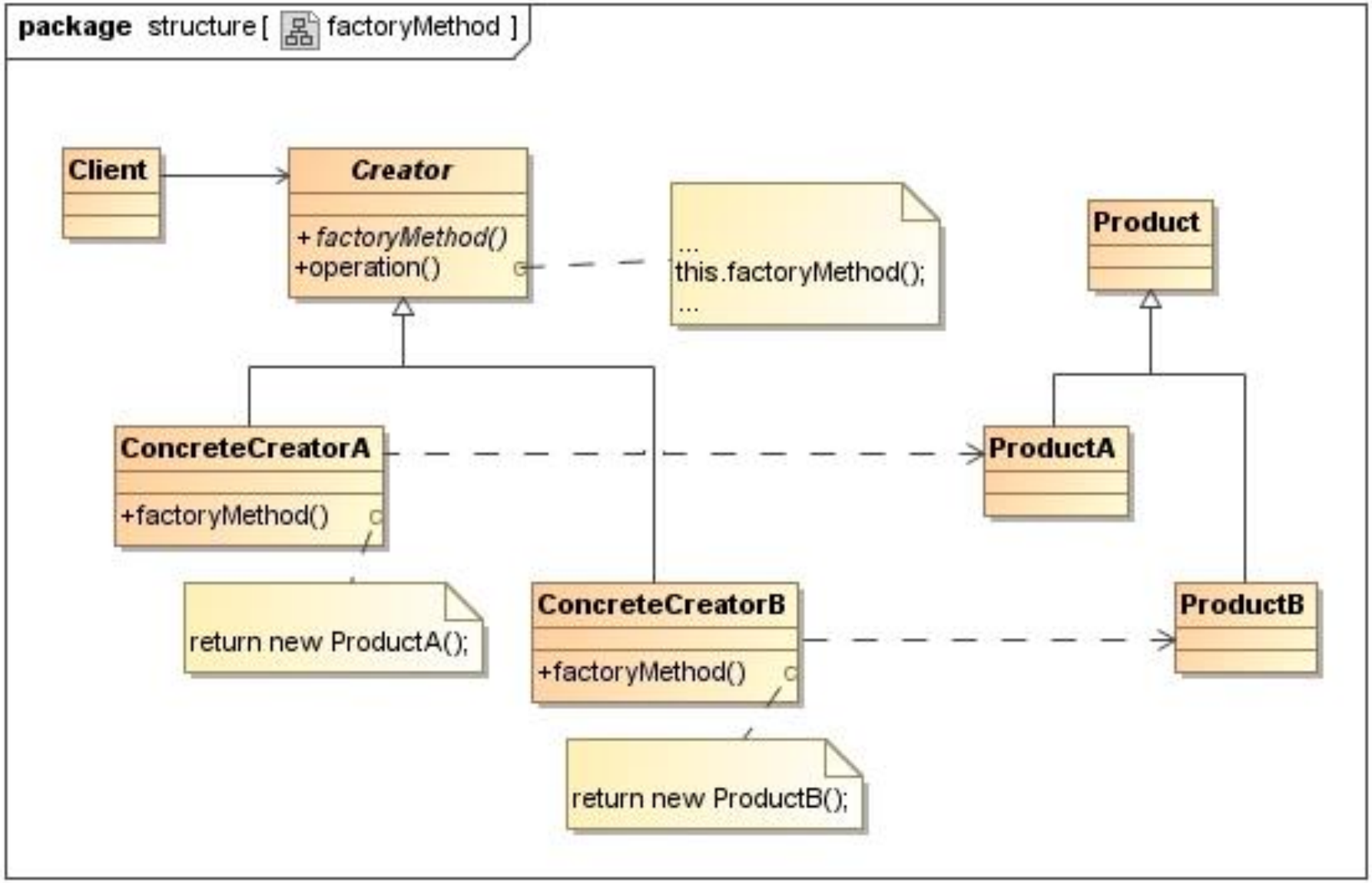
- *La formación de un repartidor con toda su responsabilidad (recoger paquetes, estudiar rutas, entrega de paquetes, …) incluyendo la conducción según los medios de transporte disponibles (p.e. bicicleta, moto, coche, …) se hace muy complicada y tiende a crecer según los nuevos disponibles .*
- *Un Repartidor abstracto define toda su responsabilidad pero no se ciñe a un medio de transporte particular, lo menciona de forma sin detallar. Otros RepartidorEnBicicleta y RepartidorEnFurgoneta redefinen por separado la conducción dependiendo del medio de transporte concreto con el que se completará toda la responsabilidad del Repartidor abstracto*
- *Ninguna explicación tiende a crecer y son más sencillas*

- ***Define una interfaz para crear un objeto pero permite a las subclases qué clase instanciar permitiendo a una clase diferir la instanciación a las subclases***

1. Problema

- Úsese cuando:
  - Una clase no puede prever la clase de objetos que debe crear
  - Una clase quiere que sean sus subclases quienes especifiquen los objetos que ésta crea
  - Las clases delegan la responsabilidad en una de entre varias clases auxiliares y queremos localizar qué subclase de auxiliar concreta es en la que se delega

2. Solución


package structure [factoryMethod]
Client
Creator
+ factoryMethod()
+operation()
...
this.factoryMethod();
...
Product
ConcreteCreatorA
+factoryMethod()
ProductA
return new ProductA();
ConcreteCreatorB
+factoryMethod()
ProductB
return new ProductB();

2. Solución

- Las dos principales variantes del patrón *Factory Method* son (1) cuando la clase Creador es una clase abstracta y no proporciona una implementación para el método de fabricación que declara y (2) cuando el Creador es una clase concreta y proporciona una implementación predeterminada del método de fabricación. También es posible tener una clase abstracta que defina una implementación predeterminada, pero esto es menos común.
  - El primer caso requiere que las subclases definan una implementación porque no hay ningún comportamiento predeterminado razonable. En el segundo caso, el Creador concreto usa el método de fabricación principalmente por flexibilidad. Lo hace siguiendo una regla que dice: "crear objetos en una operación aparte, para que las subclases puedan redefinir el modo en que son creados". Esta regla asegura que los diseñadores de las subclases puedan cambiar la clase de objetos

2. Solución

- Otra variante del patrón permite que los método de fabricación creen varios tipos de productos. El método de fabricación recibe un parámetro que identifica el tipo de objeto a crear. Todos los objetos creados por el método compartirán la interfaz Producto.
- Otro potencial problema de los métodos de fabricación es que pueden obligar a usar la herencia sólo para crear los objetos Producto apropiados. Otra forma de hacer esto es proporcionar una subclase plantilla de Creador que está parametrizada con la clase de Producto.

2. Solución

- Eliminan la necesidad de ligar las clases específicas de la aplicación a nuestro código. El código sólo trata con la interfaz Producto; además, puede funcionar con cualquier clase ProductoConcreto definida por el usuario
- Un inconveniente potencial de los métodos de fabricación es que los clientes pueden tener que heredar de la clase Creador simplemente para crear un determinado objeto ProductoConcreto
- Crear objetos dentro de una clase con un método de fabricación es siempre más flexible que hacerlo directamente. El *Factory Method* les da a las subclases un punto de enganche para porveer una versión extendida de un objeto

3. Consecuencias

- Los clientes pueden encontrar útiles los métodos de fabricación, especialmente en el caso de jerarquías de clases paralelas. Las jerarquías de clases paralelas se producen cuando una clase delega alguna de sus responsabilidades a una clase separada (como las clases de estructuras de datos y la obtención de sus iteradores).

4. Relaciones

- Alternativamente:
  - ***Template Method*** para llamar a los métodos de fabricación desde su interior.
  - ***Prototype*** para crear los productos