Machuanin, Elias Ariel
DNI: 37.469.526
Fecha: 25/03/2020

Formato de diseño del patrón Factory

Nombre: Factory Pattern (Patron Fabrica).

Definición:

Factory Pattern es un patrón de diseño de creación que proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crearán.

Problema/s:

Creación de ciertos objetos demasiado compleja o enrevesada. Cuando se desconoce el tipo de objeto que se necesita instanciar, hasta el momento de ejecutar el programa.

Motivación:

Aplicación de gestión de logística: La primera versión de la aplicación solo puede manejar el transporte en camiones, por lo que la mayor parte de su código vive dentro de la clase de Camiones.
Después de un tiempo, la aplicación se vuelve bastante popular. Cada día recibe docenas de solicitudes de compañías de transporte marítimo para incorporar la logística marítima en la aplicación.
En la actualidad, la mayor parte de su código está acoplado a la clase Camion. Agregar Barcos a la aplicación requeriría de realizar cambios en toda la base de código. Además, si más tarde se decide agregar otro tipo de transporte a la aplicación, es probable que se deba realizar todos estos cambios nuevamente.
El resultado será un código bastante desagradable, plagado de condicionales que cambian el comportamiento de la aplicación según la clase de objetos de transporte.

Solución:

El patrón Método de fábrica sugiere que reemplace las llamadas de construcción de objetos directos (utilizando el nuevo operador) con llamadas a un método de fábrica especial. No se preocupe: los objetos aún se crean a través del nuevo operador, pero se llama desde el

método de fábrica. Los objetos devueltos por un método de fábrica a menudo se denominan "productos".
A primera vista, este cambio puede parecer inútil: se acaba de mover la llamada del constructor de una parte del programa a otra. Sin embargo, ahora se puede anular el método de fábrica en una subclase y cambiar la clase de productos que crea el método.
Sin embargo, existe una ligera limitación, las subclases pueden devolver diferentes tipos de productos solo si estos productos tienen una clase base o interfaz común. Además, el método de fábrica en la clase base debe tener su tipo de retorno declarado como esta interfaz.
Por ejemplo, las clases Truck y Ship deben implementar la interfaz de transporte, que declara un método llamado entrega. Cada clase implementa este método de manera diferente: los camiones entregan la carga por tierra, los barcos entregan la carga por mar. El método de fábrica en la clase LogísticaTerrestre devuelve objetos camión, mientras que el método de fábrica en la clase LogísticaAcuatica devuelve barcos.
Mientras todas las clases de productos implementen una interfaz común, puede pasar sus objetos al código del cliente sin romperlo.
El código que usa el método de fábrica (a menudo llamado código de cliente) no ve una diferencia entre los productos reales devueltos por varias subclases. El cliente trata todos los productos como transporte abstracto. El cliente sabe que se supone que todos los objetos de transporte tienen el método de entrega, pero exactamente cómo funciona no es importante para el cliente.

Objetivo: Mirando la declaración del problema, el objetivo debería ser:

1. Que el cliente no debe ser consciente de la creación de instancias del objeto.

2. El cliente debe acceder a los objetos a través de una interfaz común.

Implementación:

1. Hacer que todos los productos sigan la misma interfaz. Esta interfaz debe declarar métodos que tengan sentido en cada producto.

2. Agregar un método de fábrica vacío dentro de la clase de creador. El tipo de retorno del método debe coincidir con la interfaz de producto común.

3. En el código del creador, hallar todas las referencias a los constructores de productos. Uno por uno, reemplazarlos con llamadas al método de fábrica, mientras se extrae el código de creación del producto en el método de fábrica.
   Es posible que se deba agregar un parámetro temporal al método de fábrica para controlar el tipo de producto devuelto.
   En este punto, el código del método de fábrica puede parecer bastante feo. Puede tener un operador switch grande que elija qué clase de producto crear. Pero no se preocupe, lo arreglaremos lo suficientemente pronto.

4. Ahora, cree un conjunto de subclases de creadores para cada tipo de producto enumerado en el método de fábrica. Anule el método de fábrica en las subclases y extraiga los bits apropiados de código de construcción del método base.

5. Si hay demasiados tipos de productos y no tiene sentido crear subclases para todos ellos, puede reutilizar el parámetro de control de la clase base en las subclases.
   Por ejemplo, imagine que tiene la siguiente jerarquía de clases: la clase de correo base con un par de subclases: CorreoAereo y CorreoTierra ; Las clases de transporte son avión, camión y tren. Mientras que la clase CorreoAereo solo usa objetos avion, CorreoTierra puede funcionar con objetos camión y tren. Puede crear una nueva subclase (por ejemplo, CorreoTren) para manejar ambos casos, pero hay otra opción. El código del cliente puede pasar un argumento al método de fábrica de la clase CorreoTierra para controlar qué producto desea recibir.

6. Si, después de todas las extracciones, el método de fábrica base se ha quedado vacío, puede hacerlo abstracto. Si queda algo, puede convertirse en un comportamiento predeterminado del método.

Ventajas:

1. Evita el acoplamiento estrecho entre el creador y los productos concretos.

2. Principio de responsabilidad única. Puede mover el código de creación del producto a un lugar en el programa, lo que facilita el soporte del código.

3. Principio abierto / cerrado. Puede introducir nuevos tipos de productos en el programa sin romper el código existente.

Desventajas:

1. El código puede volverse más complicado ya que necesita introducir muchas subclases nuevas para implementar el patrón. El mejor de los casos es cuando introduce el patrón en una jerarquía existente de clases de creadores.

Patrones relacionados:

Muchos diseños comienzan usando el Patrón Factory (menos complicado y más personalizable a través de subclases) y evolucionan hacia Abstract Factory, Prototype o Builder (más flexible, pero más complicado).

Las clases Abstract Factory a menudo se basan en un conjunto de métodos Factory, pero también puede usar Prototype para componer los métodos en estas clases.

Puede usar el Patrón Factory junto con Iterator para permitir que las subclases de colección devuelvan diferentes tipos de iteradores que son compatibles con las colecciones.

El prototipo no se basa en la herencia, por lo que no tiene sus inconvenientes. Por otro lado, Prototype requiere una inicialización complicada del objeto clonado. El Método de fábrica se basa en la herencia, pero no requiere un paso de inicialización.

Factory Method es una especialización de Template Method. Al mismo tiempo, un Método de Fábrica puede servir como un paso en un Método de Plantilla grande.

Fuente: *https://refactoring.guru/design-patterns/factory-method*