Daza Corredor Alejandro
Paolo

Patrones Creacionales

Factory Method

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [Christopher Alexander].

1. Introducción

El concepto de patrones creacionales nos dice que los patrones de creación abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica dejando para más tarde la decisión de qué clases crear o cómo crearlas, este concepto nos habla de decidir que clase concreta se instanciara en tiempo de ejecución. Uno de los patrones que nos permite implementar este concepto es el Factory Method, veamos de que se trata.

3. Factory Method

Nombre del patrón:
Factory Method.

Clasificación del patrón:
Creacional.

Intención:
Provee una interfaz para crear un objeto, pero deja a las subclases decidir cual clase instanciar, Factory Method permite a una clase aplazar la instanciación a las subclases.

También conocido como:
kit o toolkit.

Motivación:
Los Frameworks usan clases abstractas para definir y mantener las relaciones entre objetos. Un Framework es a menudo responsable por la creación de objetos.

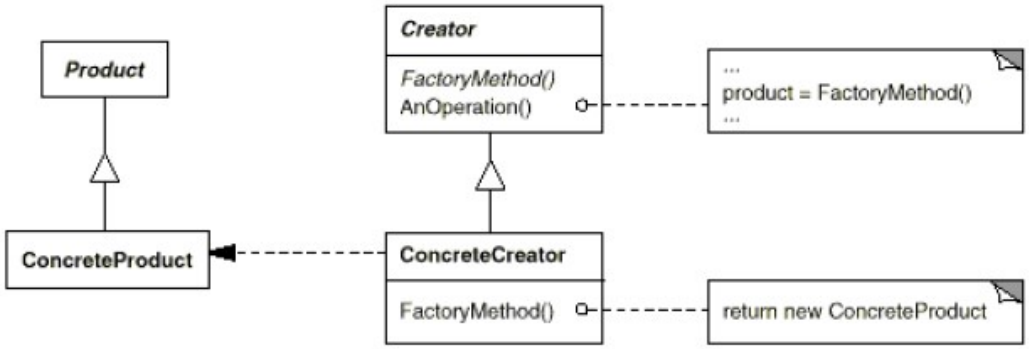
Considere un Framework para aplicaciones que pueden presentar múltiples documentos a los usuarios. las principales abstracciones en este Framework son las clases Application y Document, ambas clases son abstractas y los clientes tienen que dejar a las subclases sus implementaciones específicas de la aplicación.

El patrón Factory Method encapsula el conocimiento de cual subclase de Document crear. La subclase Application redefine la operación abstracta createDocument de la clase Application para retornar la subclase apropiada de Document. En este caso podemos llamar a createDocument un Factory Method porque es responsable de la creación de objetos.

Aplicabilidad:
El patrón Factory Method debe ser usado cuando:

- Una clase no es capaz de anticipar la clase de objetos que esta debe crear.
- Una clase quiere que sus subclases especifiquen los objetos que estos crean.
- Clases que delegan responsabilidad a una de varias clases “ayudantes” y se desea localizar el conocimiento de cual “ayudante” es el delegado.

Estructura:



Participantes:

- Product: define una interfaz para los objetos que Factory Method crea.
- ConcreteProduct: implementa la interfaz de Product.
- Creator: declara el factory method, el cual retorna un objeto del tipo Product. Creator puede también definir una implementación por defecto de factory method que retorne un objeto del tipo ConcreteProduct por defecto. Puede llamar al factory method para crear un objeto de tipo Product.
- ConcreteCreator: sobreescribe el factory method para retornar una instancia de ConcreteProduct.

Colaboraciones:

- Creator depende de sus subclases para definir el factory method y retornar una instancia del ConcreteProduct apropiado.

Consecuencias:

- Elimina la necesidad de construir clases específicas a aplicaciones en el código.
- Una desventaja potencial es que las clases deben tener subclases que permitan crear un producto concreto en particular.
- La creación de objetos en una clase con factory method es más flexible que la creación de objetos directamente.

Implementación:

- Se pueden dar dos variaciones: el primer caso cuando la clase Creator es una clase abstracta y no provee una implementación del factory method que declara; el segundo caso la clase Creator es una clase concreta y provee una implementación por defecto del factory method.
- Otra variación consiste en factory method con parámetros, es decir que se puede tener métodos que puedan retornar varias clases de objetos lo cual decidirán de acuerdo al parámetro recibido.

Código de ejemplo:

```
public abstract class AbstractProduct {
    public abstract String nombreProducto();
}

public class ConcreteProductA extends AbstractProduct{
    @Override
    public String nombreProducto() {
        return "Producto Concreto A";
    }
}

public abstract class AbstractCreator {
```

```
    public abstract AbstractProduct factoryMethod();

    public AbstractProduct construir(){
        AbstractProduct p = factoryMethod();
        return p;
    }
}

public class ConcreteCreatorA extends AbstractCreator{
    @Override
    public AbstractProduct factoryMethod() {
        return new ConcreteProductA();
    }
}

public class Main {
       public static void main(String[] args) {
            AbstractCreator c = new ConcreteCreatorA();
            AbstractProduct p = c.construir();
            System.out.println(p.nombreProducto());
       }
}
```

Usos conocidos:

- Muy usado en toolKits y Frameworks.

Patrones relacionados:

- Las clases de Abstract Factory son frecuente mente implementadas usando métodos de Factory Method.

Referencias bibliográficas

[1] Design Patterns. Elements of Reusable Object-Oriented Software - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley (GoF- Gang of Four)

[2] Desing Patterns Java. James W Cooper. ebook – Addison Wesley.

[3] Patrones de Diseño, Diseño de Software Orientado a Objetos - Joaquin Garcia. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

[4] Patrones de diseño - http://es.kioskea.net/contents/genie-logiciel/design-patterns.php3.

Daza Corredor, Alejandro Paolo
Ingeniero de Sistemas, egresado de la Universidad Distrital Francisco José de Caldas, Docente TCO del proyecto curricular de Ingeniería de Sistemas en el área de programación y estudiante de la especialización de Ingeniería de Software de la Universidad Distrital Francisco José de Caldas.

e-mail: apdaza@gmail.com