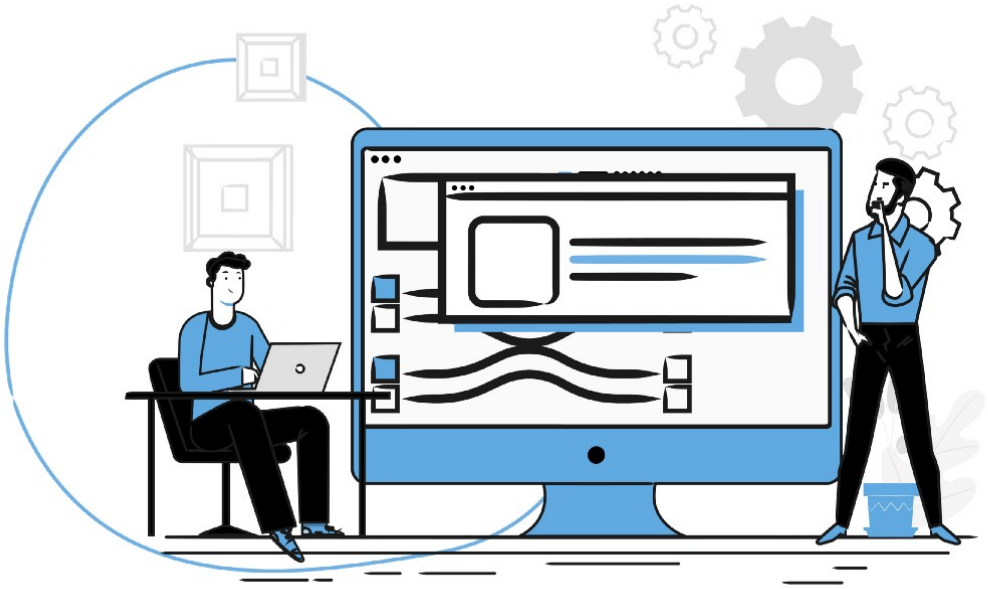
Práctica Patrones de Diseño

Desarrollo de Interfaces

Patrón de Diseño Creacional: Factory



Rafael Jaime Blanco Aranda

ÍNDICE

Definición

Se trata de un patrón de diseño enfocado en la generación de una interfaz que permite crear objetos en una superclase, permitiendo a las subclases alterar el tipo de los objetos creados. Esto se consigue mediante el polimorfismo de la programación orientada a objetos.

Características

- Sigue el principio de responsabilidad única, es decir, permite mover el código de creación de objetos a cualquier lugar del programa, facilitando la encapsulación y el mantenimiento del mismo.

- También sigue el principio de abierto/cerrado, facilitando la incorporación de nuevos tipos de objetos en la aplicación sin la necesidad de descomponer el código existente.

- Evita el acoplamiento entre la clase generadora y el objeto instanciado.

- Requiere un aumento de la complejidad del código en cuanto a número de subclases, pero facilita la escalabilidad, la organización y la limpieza.

Problema que intenta resolver

El problema que intenta resolver el patrón de diseño Factory radica en el acoplamiento que generan los objetos al ser instanciados directamente en la clase que los requieren. Este problema supone fuertes complicaciones en la escalabilidad de la aplicación y genera complejidad innecesaria en el código. La utilización de una interfaz que permite crear objetos en una superclase elimina los problemas de acoplamiento puesto que evita que se genere el tipo concreto del objeto hasta que éste no es instanciado, eliminando posibles relaciones de dependencia y aportando flexibilidad a la aplicación.

¿En qué lenguaje es más utilizado?

Este patrón de diseño se basa en los principios de la programación orientada a objetos (POO), así que los lenguajes en los que lo podemos encontrar son aquellos que sean orientados a objetos como:

- Java
- JavaScript
- TypeScript
- C++
- C#
- Python
- PHP
- Ruby
- Visual Basic
- Perl

Representación UML

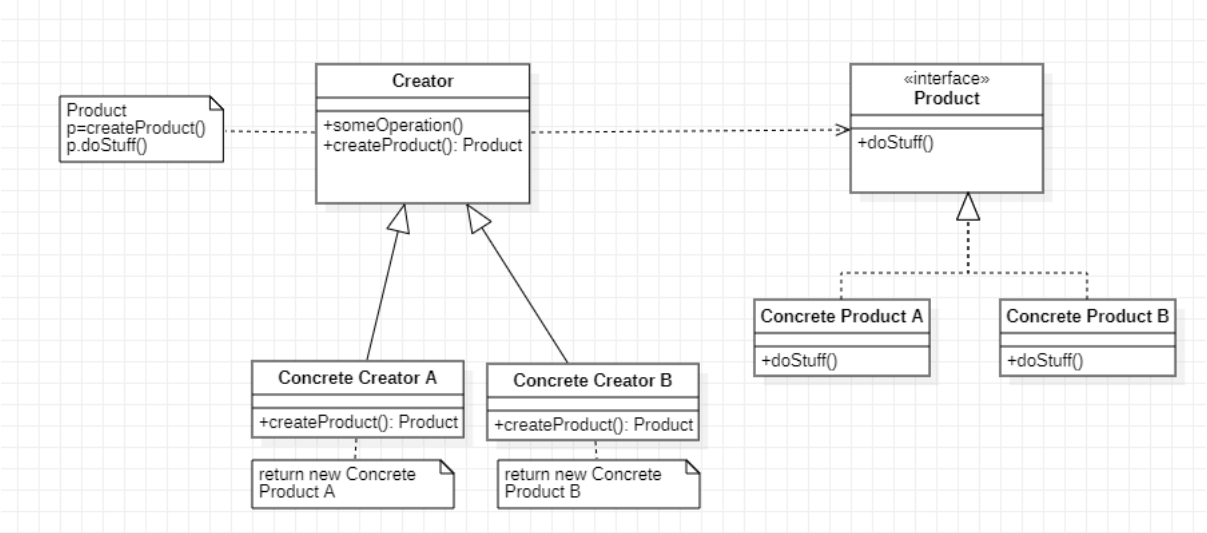


Imagen 1. *Diagrama UML del patrón de diseño Factory.*

Ejemplo: ObradorPatrónFactory

Para poner en práctica este patrón de diseño, se ha diseñado un ejemplo específicamente para esta práctica en el que se pone de manifiesto el funcionamiento y la utilidad de dicho patrón. La aplicación propuesta es un proyecto de java realizado en eclipse donde se intenta recrear el funcionamiento de un obrador, que necesita generar sus dulces para venderlos a los clientes. Para

observar con claridad el ejemplo se recomienda ejecutar la clase *"ObradorPatronFactory"* del proyecto adjunto en la entrega. El funcionamiento del mismo se detalla a continuación:
El proyecto sigue el siguiente diagrama UML acorde con la implementación del patrón Factory.

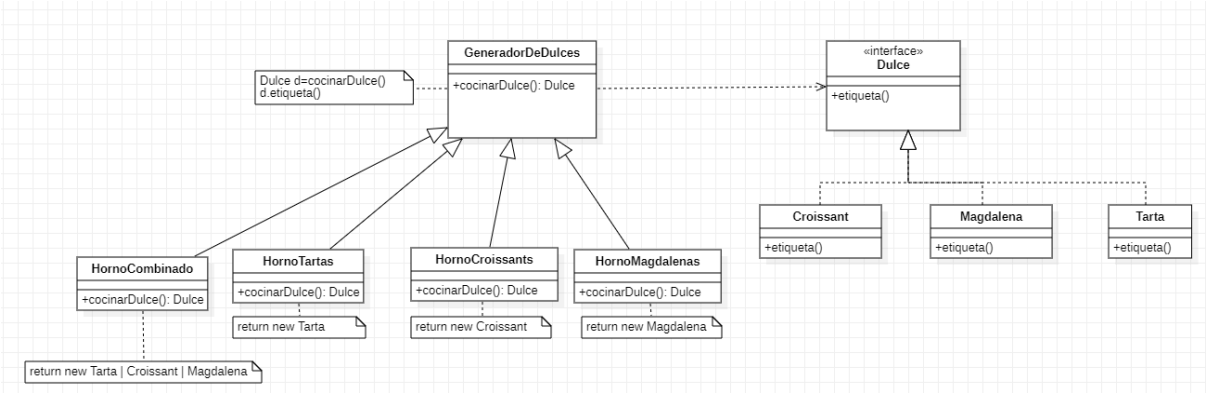


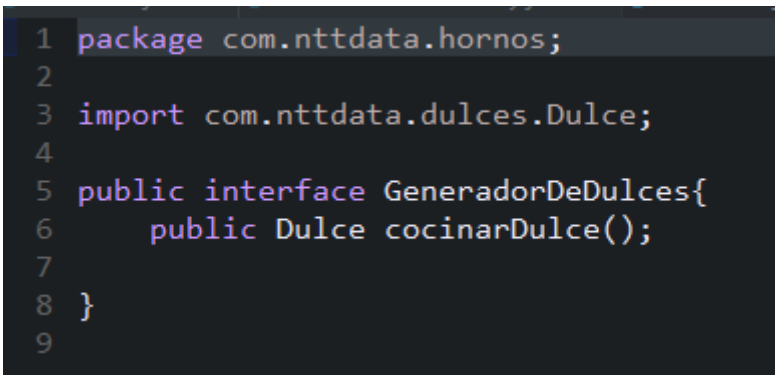
Imagen 2. *Diagrama de clases UML del proyecto de ejemplo en java utilizando el patrón de diseño Factory.*

En primer lugar, existe una interfaz Dulce, que actúa de tipo genérico sobre el que se van a basar los productos del obrador.

```
1 package com.nttdata.dulces;
2
3 public interface Dulce {
4
5     public void etiqueta();
6
7
8 }
```

Imagen 3. *Interfaz general que van a implementar los productos del obrador.*

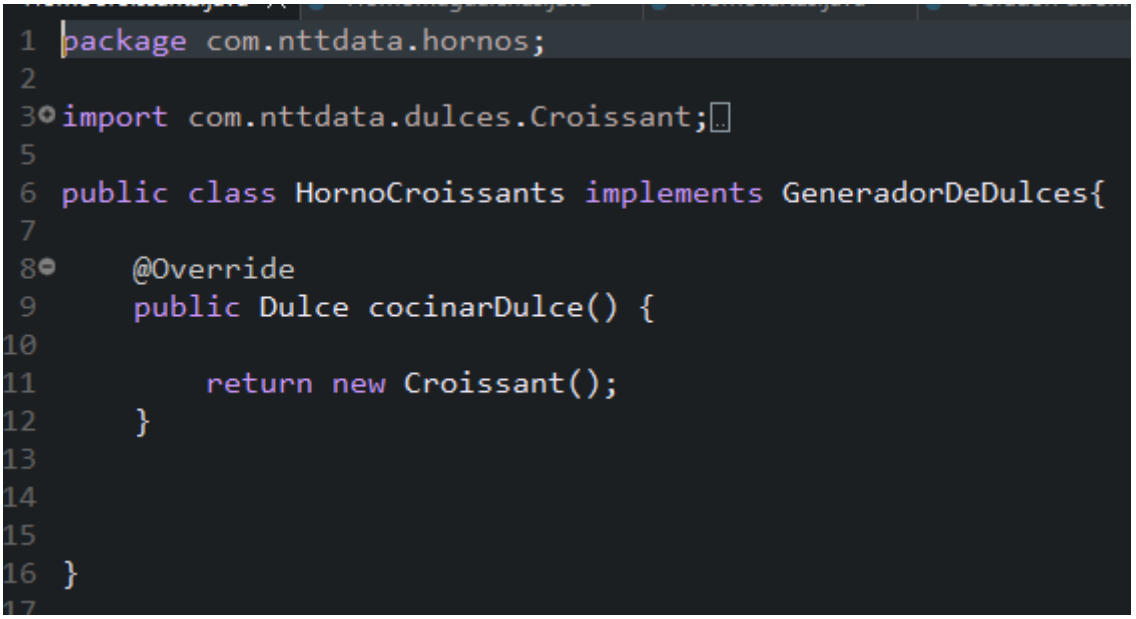
También se puede observar la interfaz *"GeneradorDeDulces"*, que será común en todos los hornos del obrador que van a funcionar como clases generadoras de objetos (dulces).



```java
package com.nttdata.hornos;

import com.nttdata.dulces.Dulce;

public interface GeneradorDeDulces{
    public Dulce cocinarDulce();

}

```

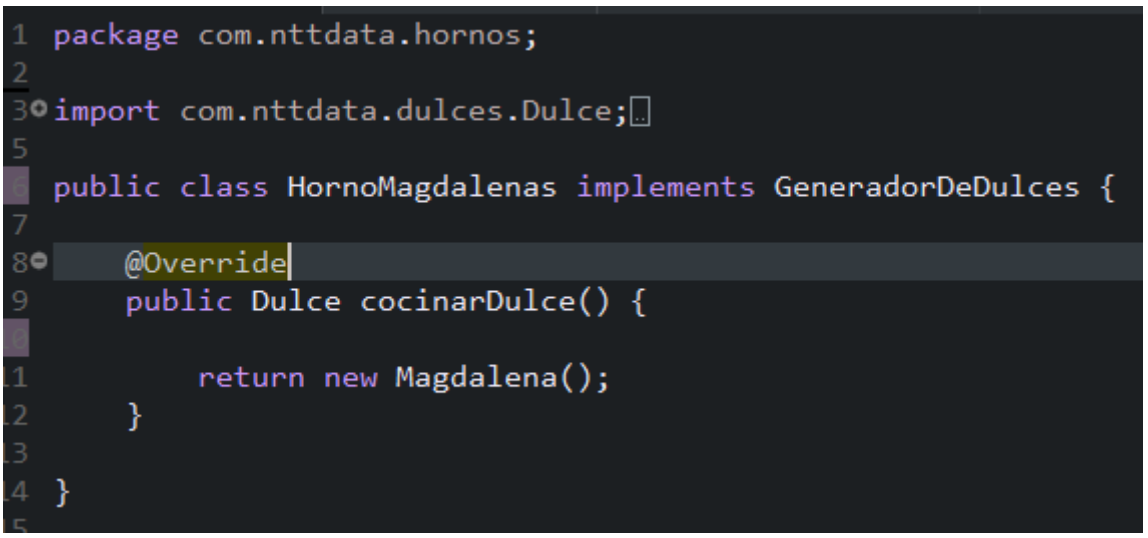
Imagen 4. *Interfaz general que van a implementar los hornos.*

Como se acaba de mencionar, las clases *HornoCroissant*, *"HornoMagdalenas"* y *"HornoTartas"* son clases que implementan la interfaz *"GeneradorDeDulces"* y que van a generar cada una productos de un tipo determinado, correspondiente al objeto que contienen en su nombre.



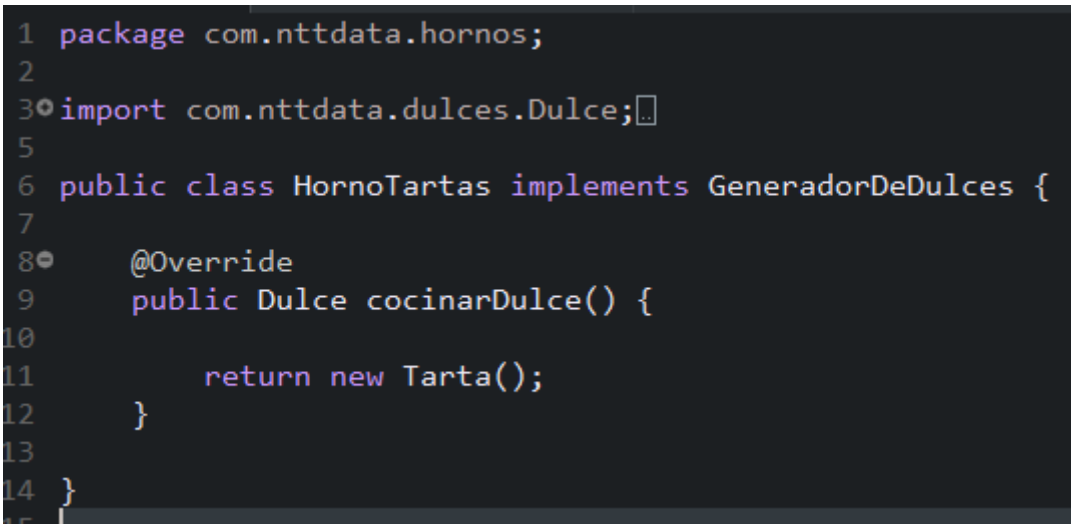
```java
package com.nttdata.hornos;

import com.nttdata.dulces.Croissant;

public class HornoCroissants implements GeneradorDeDulces{

    @Override
    public Dulce cocinarDulce() {

        return new Croissant();
    }



}
```

Imagen 5. *Clase de generadora de objetos de tipo Croissant.*



```java
package com.nttdata.hornos;

import com.nttdata.dulces.Dulce;

public class HornoMagdalenas implements GeneradorDeDulces {

    @Override
    public Dulce cocinarDulce() {

        return new Magdalena();
    }

}
```

Imagen 6. *Clase generadora de objetos de tipo Magdalena.*



```java
package com.nttdata.hornos;

import com.nttdata.dulces.Dulce;

public class HornoTartas implements GeneradorDeDulces {

    @Override
    public Dulce cocinarDulce() {

        return new Tarta();
    }

}
```

Imagen 7. *Clase generadora de objetos de tipo Tarta.*

La clase *"HornoCombinado"* va a instanciar de forma aleatoria cualquiera de los dulces existentes en el obrador.

```java
package com.nttdata.hornos;

import com.nttdata.dulces.Croissant;

public class HornoCombinado implements GeneradorDeDulces{

    @Override
    public Dulce cocinarDulce() {
        int numeroAleatorio=(int)(Math.random()*10);
        Dulce dulce;
        switch(numeroAleatorio) {
        case 0,1,2,3: dulce=new Croissant();
        break;
        case 4,5,6,7: dulce= new Magdalena();
        break;
        case 8,9,10: dulce= new Tarta();
        break;
        default: dulce=null;
        }
        return dulce;
    }

}

```

Imagen 8. *Clase generadora de objetos aleatorios entre los dulces disponibles..*

Por último, se pueden observar las clases *"Croissant"*, *"Magdalena"* y *"Tarta"*, que van a implementar la interfaz *"Dulce"* y serán los diferentes tipos de objetos que se pueden crear en el obrador, cada uno con una implementación diferente del método *"etiqueta()"* originario de la interfaz *"Dulce"*.

```java
package com.nttdata.dulces;

public class Croissant implements Dulce{

    @Override
    public void etiqueta() {
        System.out.println("La etiqueta del Croissant dice:");
        System.out.println("-Cruje al Morderlo");
        System.out.println("-Sabe a Mantequilla");

    }

}

```

Imagen 9. *Clase que representa a los dulces de tipo Croissant.*

```java
package com.nttdata.dulces;

public class Magdalena  implements Dulce{

    @Override
    public void etiqueta() {
        System.out.println("La etiqueta de la Magdalena dice:");
        System.out.println("-Es esponjosa");
        System.out.println("-Como las de La Bella Easo");

    }

}
```

Imagen 10. *Clase que representa a los dulces de tipo Magdalena.*

```java
package com.nttdata.dulces;

public class Tarta implements Dulce{

    @Override
    public void etiqueta() {
        System.out.println("La etiqueta de la porción de Tarta dice:");
        System.out.println("-Es de cumpleaños");
        System.out.println("-Es tarta de queso");

    }

}
```

Imagen 11. *Clase que representa a los dulces de tipo Tarta.*

Al ejecutar el proyecto se instancian las clases generadoras de objetos (hornos, Imagen 12) y se mostrarán por consola (Imágenes 17, 18 y 19) dos situaciones en las que un cliente llega al obrador y pide algún dulce. En estas dos situaciones se crearán los dulces para los clientes (Imágenes 13 y 15) y se mostrarán las etiquetas de los mismos (Imágenes 14 y 16) para asegurar que el proceso se ha realizado correctamente y que la implementación del patrón de diseño ha funcionado a la perfección.

```java
//Inicialización de los generadores de dulces
HornoCroissants hornoCroissants= new HornoCroissants();
HornoMagdalenas hornoMagdalenas= new HornoMagdalenas();
HornoTartas hornoTartas= new HornoTartas();
HornoCombinado hornoGeneral= new HornoCombinado();
```

Imagen 12. *Inicialización de los hornos..*

```
//Generación de los dulces con su horno correspondiente
Dulce dulce1Cliente1=hornoCroissants.cocinarDulce();
Dulce dulce2Cliente1=hornoMagdalenas.cocinarDulce();
Dulce dulce3Cliente1=hornoTartas.cocinarDulce();
```

Imagen 13. *Inicialización de los dulces del Cliente 1.*

```
dulce1Cliente1.etiqueta();
dulce2Cliente1.etiqueta();
dulce3Cliente1.etiqueta();
```

Imagen 14. *Comprobación de la etiqueta de los dulces del Cliente 1.*

```
Dulce dulceCliente2=hornoGeneral.cocinarDulce();
```

Imagen 15. *Inicialización de los dulces del Cliente 2.*

```
dulceCliente2.etiqueta();
```

Imagen 16. *Comprobación de la etiqueta de los dulces del Cliente 2.*

```
---------------------------------------------------------------------------------------------------------------------------------
En función de lo que pida el cliente se hará uso de un horno diferente, aunque todos los hornos son generadores de dulces.
Este polimorfismo nos ofrece claridad y encapsulación del código a la vez que evita problemas de acoplamiento de objetos a lo largo del mismo.
---------------------------------------------------------------------------------------------------------------------------------

Comienza el día en el obrador "PatrónFactory" y se encienden los hornos
```

Imagen 17. *Salida por consola de la ejecución del proyecto que muestra la inicialización de los objetos*

```
Cliente 1
-Buenos días, quisiera un Croissant recién hecho, una Magdalena y un trozo de Tarta
-Por supuesto señor, enseguida se los preparo
-Aquí tiene sus dulces (Le entrega 3 paquetitos)
-Pero, un momento ¿Cómo sé que hay dentro de cada envoltorio?
-Muy fácil, lo que ha salido de nuestro horno de Croissants es un Croissant, lo que ha salido de nuestro horno de Magdalenas,
es una Magdalena, y lo que ha salido de nuestro horno de Tartas, es un pedazo de Tarta. Además, si tiene dudas, puede consultar
su etiqueta

La etiqueta del Croissant dice:
-Cruje al Morderlo
-Sabe a Mantequilla
La etiqueta de la Magdalena dice:
-Es esponjosa
-Como las de La Bella Easo
La etiqueta de la porción de Tarta dice:
-Es de cumpleaños
-Es tarta de queso
```

Imagen 18. *Salida por consola de la ejecución del proyecto que muestra la situación del Cliente 1.*

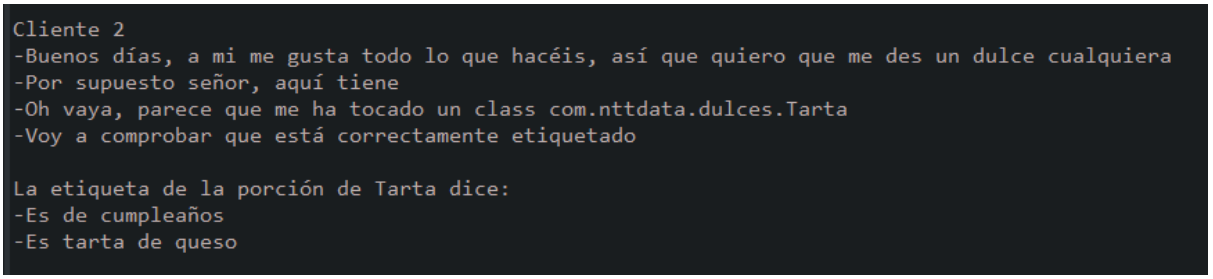


Imagen 19. *Salida por consola de la ejecución del proyecto que muestra la situación del Cliente 2.*

Bibliografía

- https://refactoring.guru/es/design-patterns/factory-method
- https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-factory/
- https://www.flaticon.es/
- https://www.youtube.com/watch?v=lLvYAzXO7Ek