Práctica 4: Implementación del patrón de diseño: Builder para la resolución de un problema

UNAM, Facultad De Ciencias.

Curso: Modelado y Programación
Profesor.Rosa Victoria Villa Padilla
Ayudante. Arturo Lemus Pablo
Ayudante. Fernando López Balcazar
Ayudante. Itzel Azucena Delgado Díaz

Equipo: Prietos en aprietos

SanMartin Macias Juan Daniel No. Cuenta 318181637
López Diego Gabriela No. Cuenta 318243485
Rivera Zavala Javier Alejandro No. Cuenta 311288876

02 de Octubre de 2022.
Ciudad de México.

1. Patrón de diseño Builder

La idea del patrón Builder es la creación de una variedad de objetos complejos teniendo una especie de base estructural de componentes, el objeto componente tiene atributos que ayudan a crear al objeto complejo a partir de ciertas necesidades del programa.
Para poder implementar Builder realizamos lo que se señala a continuación

1. Creamos una clase abstracta o interfaz Builder para la creación de los objetos
2. Creamos una clase ConcreteBuilder que extiende o implementa Builder, que se encarga de reunir a los componentes necesarios para armar el objeto concreto y así construirlo.
3. Creamos una clase Director que funcionaría como main para construir un objeto concreto utilizando ConcreteBuilder.
4. Poduct hace referencia al objeto que construye específicamente la clase ConcreteBuilder
   Es decir,

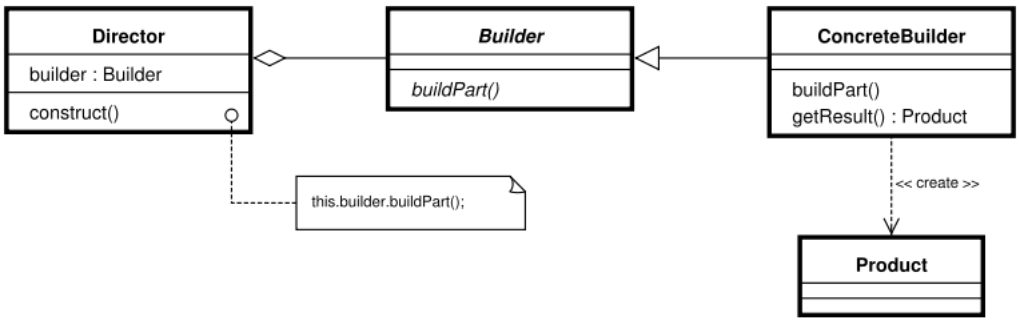

Figura 1: Diagrama UML patron Builder

Desventajas
Con este patrón podemos llegar a crear una cantidad considerable de clases y objetos en el programa, haciendo nuestro código menos manejable y por ende, mas difícil su mantenimiento.

2. Patrón de diseño Abstract Factory

El patrón Abstract Factory se utiliza cuando puede que se agreguen nuevas familias de productos, pero puede resultar contraproducente cuando el número de estas familias crece, puesto que afectaría a todas las familias creadas.

1. Declaramos interfaces de productos abstractos para todos los tipos de productos. Luego hacemos que todas las clases de productos concretos implementen estas interfaces.
2. Declaramos la interfaz de Abstract Factory con un conjunto de métodos de creación para todos los productos abstractos.
3. Implementa un conjunto de clases de fábrica concretas, una para cada variante de producto.
4. Creamos un código de inicialización de fábrica en algún main. Debemos instanciar una de las clases de fábrica concretas. Pasamos este objeto de fábrica a todas las clases que construyen productos.

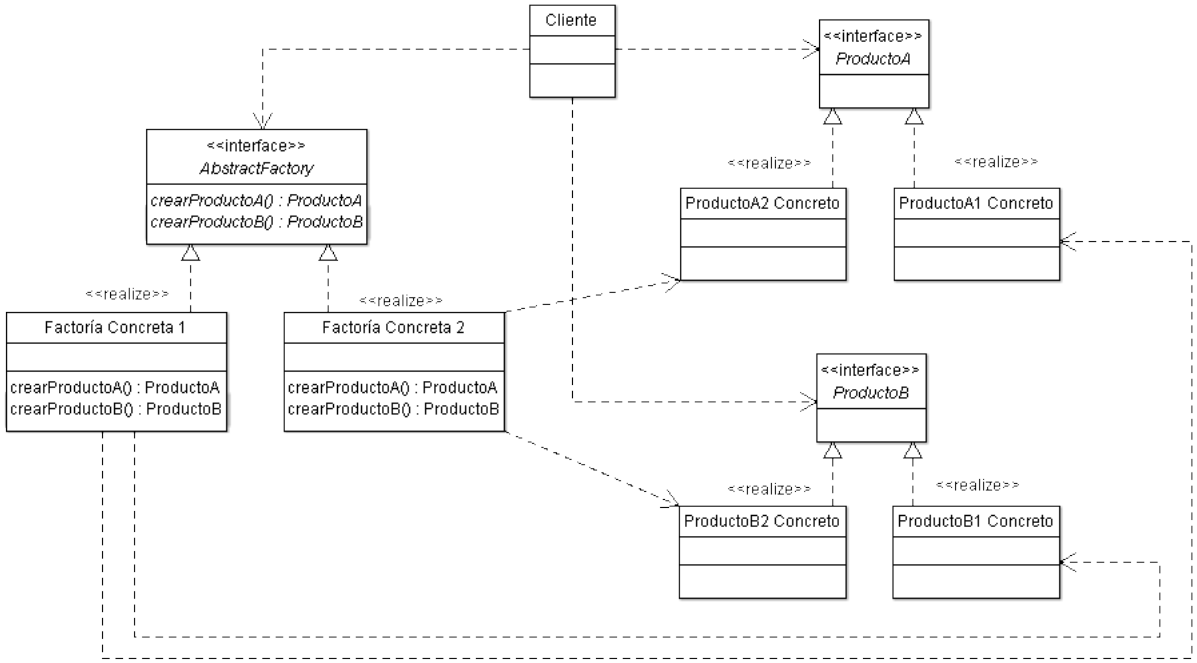

Figura 2: Diagrama UML patron Abstract Factory

Desventajas
Como se puede ver en el UML la cantidad de clases aumenta muchísimo, y gran parte del código en ellas es exactamente el mismo.

3. Patrón de diseño Factory

Al igual que abstract Factory, consiste en utilizar una clase constructora, pero con métodos de comportamiento ya definido, así se verifica que objetos comparten algún comportamiento y se evita hacer tantas clases.

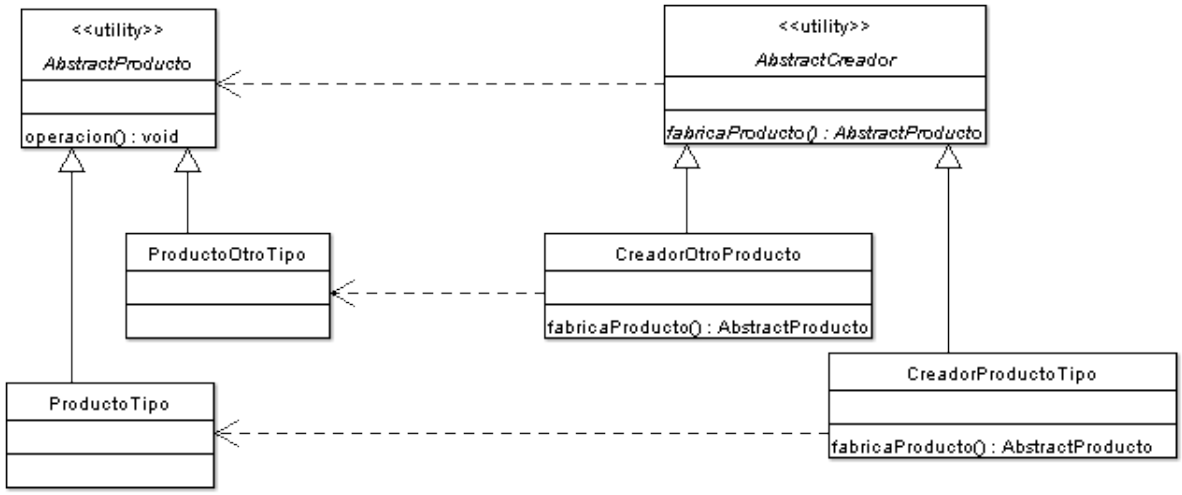

Figura 3: Diagrama UML patron Factory

Desventajas
Una vez que la estructura del algoritmo está establecido y tenemos algunas subclases funcionando es muy complicado modificar el algoritmo, ya que cualquier cambio nos obligaría a modificar todas las subclases, además que conforme empieza a crecer el problema, crece la cantidad de clases necesarias.

Por último,
Para esta práctica decidimos implementar el patrón Builder, ya que es la que menos crece en clases a comparación a los otros dos patrones antes mencionados, y que las mayores ventajas de implementar tanto *Abstract Factory* y *Factory* es que es más faćil el mantenimiento del código, sin embargo en este caso no se necesitaría mantenimiento a futuro para la práctica.
Para compilar la practica 04 escribimos lo siguiente en terminal

javac *.java

Y para ejecutarlo

java ImperioGalactico