Mateo Calderón Rincón

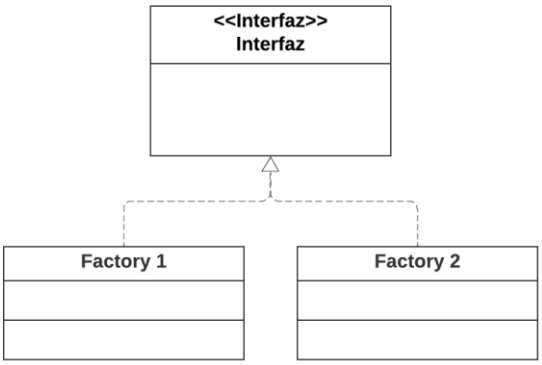
Taller 5

- Información general del proyecto: para qué sirve, cuál es la estructura general del diseño, qué grandes retos de diseño enfrenta (i.e. ¿qué es lo difícil?). Deben incluir la URL para consultar el proyecto.
  El proyecto está encaminado en una fabrica de comida rápida, que se encarga de la producción de hamburguesas y Nuggets de distintos tipos para sus usuarios.
  El diseño tiene varias estructuras interfaces (define los métodos para las principales acciones que se realizan con las hamburguesas y Nuggets, porque define los métodos para crearlos, empacarlos y freírlos), clases de implementación (crea hamburguesas y Nuggets de proteína animal y vegetal) y clases de hamburguesas y Nuggets (implementan las interfaces y las usan para representar hamburguesas de carne, Nuggets de pollo, hamburguesa vegetariana y Nuggets vegetarianos).
  Es importante detallar que hay dos fabricas distintas, una para la comida rápida habitual y la otra para la vegetariana, lo que ayuda a que luego se pueda implementar otra para otros tipos como hamburguesa de pollo, o algún otro tipo por el estilo.
  El principal reto en este proyecto (a pesar de que es pequeño) era poder simplificar la mayor parte del código, con el objetivo de simplificar la comprensión de este, y que cuando se quisiera modificar alguna parte de este, se pueda hacer sin tener que afectar todo lo demás o repetir cosas que ya se habían hecho antes.
  Link del Proyecto Total: https://github.com/kan01234/design-patterns.git (se usó la parte del Abstract Factory, del main y de las hamburguesas-nuggets)
  Link de la parte que se uso: https://github.com/kan01234/design-patterns/tree/master/abstract-factory-pattern/src/main/java/com/dotterbear/abztract/factory/pattern

- Información y estructura del fragmento del proyecto donde aparece el patrón. No se limite únicamente a los elementos que hacen parte del patrón: para que tenga sentido su uso, probablemente va a tener que incluir elementos cercanos que sirvan para contextualizarlo.
  En la mayor parte del proyecto se implementa el patrón, como se mencionaba en el anterior punto, la estructura del proyecto esta encaminado directamente al patrón, ya que implementa las interfaces, las fabricas y las clases especificas (más adelante se mencionará porque esto es importante en este patrón). Si queremos detallar un poco mas la estructura donde se usa el patrón. Estas clases del proyecto se agrupan así:
  Interfaces: FastFoodFactory, Hamburguer, Nugget
  Fabricas: NormalFastFoodFactory, VegetarianFastFoodFactory

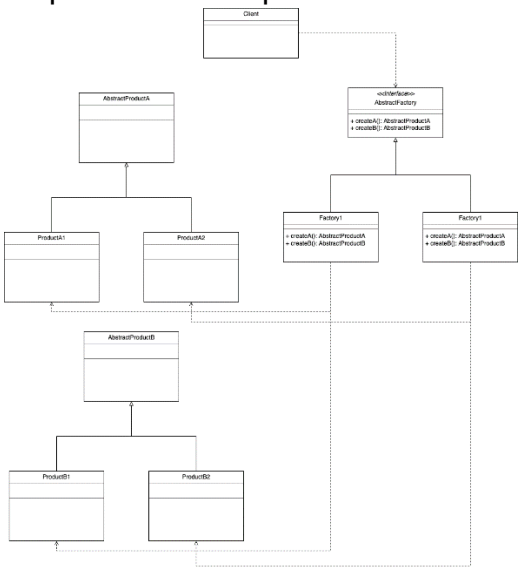
Clases especificas: BeefHamburguer, ChickenNugget, FakeMeatBurger, FakeMeatNugget.
En donde es fácil identificar cuáles son las interfaces ya que en el código lo menciona textualmente, por otro lado en las fábricas dice cuál de las interfaces implementa.

- Información general sobre el patrón: qué patrón es y para qué se usa usualmente.
  El patrón Abstract Factory permite trabajar con varias familias de productos relacionados sin depender de las clases concretas de esos productos. Proporciona una interfaz común, llamada fábrica abstracta, para crear diferentes tipos de productos. Luego, se implementan fábricas concretas que heredan de la fábrica abstracta y se encargan de crear productos específicos dentro de cada familia. Esto permite intercambiar fácilmente entre diferentes familias de productos sin modificar el código que los utiliza. Es posible identificar muchos ejemplos en donde se pueden usar este tipo de patrón, pero mencionaremos dos en concreto; una tienda de muebles que vende sillas, mesas y sofás, pero en diferentes estilos, clásico y moderno, para no tener que crear todos los productos con sus variantes, podemos usar este patrón; otro caso en donde se puede usar es en aquellos videojuegos donde nos dan la opción de jugar con los gráficos clásicos o con los modernos, la jugabilidad del juego no debe cambiar, debe ser exactamente igual para aquellos jueguen con los gráficos modernos o con los antiguos, para no tener que usar dos códigos distintos para cada tipo de gráfico, usamos el patrón de Abstract Factory.



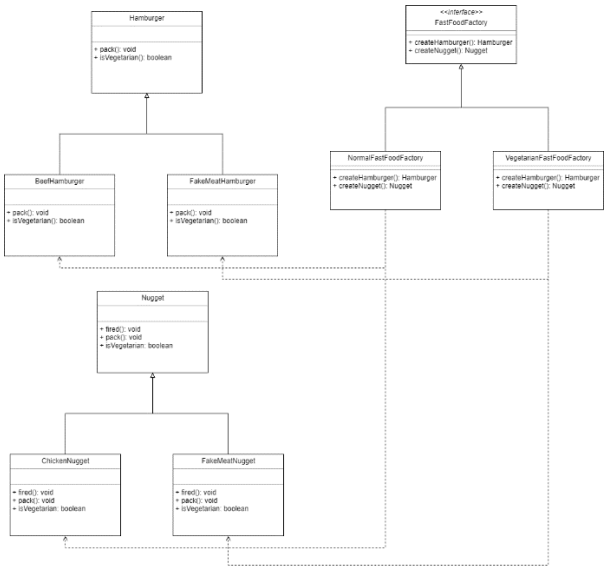
Ahora es importante entender como funciona el patrón, para ello creamos las interfaces para cada producto, independientemente de las variantes que este pueda llegar a tener, por lo que todas las variantes del mismo objeto se mueven a una clase mayor. Además, se debe declarar la fabrica abstracta, que es una interfaz que crea los productos que hacen parte de esa familia. Luego, para crear esas variantes, se crean interfaces basadas en la

FabricaAbstracta, que devuelve cosas de un tipo particular, así es posible cambiar el tipo de fabrica que pasamos en el código al usuario, para que reciba lo que hace sin importar sus variantes.



Este patrón también nos ayuda a tener la certeza de que los productos que se obtienen de una fábrica son compatibles entre sí. Además, evitamos un acoplamiento fuerte entre productos concretos y el código cliente, de igual forma mantiene el principio de abierto cerrado y de responsabilidad única. A pesar de que como se menciono antes ofrece varias ventajas, también puede llegar a complicar tener contras, ya que puede ser que el código se complique más de lo que debería, porque se introducen muchas interfaces (dependiendo lo que haga el programa)

- <u>Información del patrón aplicado al proyecto: explicar cómo se está utilizando el patrón dentro del proyecto.</u>
  Realmente el proyecto esta basado en este patrón, desde las interfaces hasta las fábricas, la manera en como mejor se puede entender esto es mirando el diagrama de clases que explica la estructura del patrón, y ver que realmente esta muy relacionado en la forma como escrito el proyecto.



- ¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?
  Como mencionaba anteriormente, este proyecto está basado en el patrón de Abstract Factory, pensando en usar una estructura extendible en donde se puedan agregar diferentes fabricas para tener otro tipo de productos, y de esa manera que el código se vea más simplificado y que en caso de que se requiera hacer cualquier modificación se pueda hacer sin mayor problema, como que el programa se pueda ajustar a las preferencias del usuario. Esto sin mencionar que este patrón bien aplicado otorga muchos beneficios (como se mencionaron antes), abstracción, desacoplamiento, flexibilidad, extensibilidad, open/closed, entre otros.

- ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?
  También como se mencionaba en la descripción del patrón este patrón tiene algunas desventajas, en el caso de este proyecto el número de clases, en este proyecto se usan 9 clases, cuando en si lo que hace no es tan complejo, aunque en cierto modo se entiende, ya que si se desea extender el código será más fácil con la estructura actual. También puede pasar que aumente el acoplamiento, lo que haría que una modificación en alguna clase podría afectar el resto del código, quizás por el estado actual del proyecto si se presenta puede ser más fácil de identificar.

- ¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?

Siento que se pudo haber solucionado con herencia, aunque en caso de que se deseara usar otro tipo de patrón, seguramente el patrón más cercano seria Factory Method, que comparte características muy similares al Abstract Factory, cuya principal diferencia es que en este ultimo se crean familias de productos relacionados, en este se usa para crear una única variante del producto, lo cual sería útil si solo se creara hamburguesas y nuggets de una sola variante. Por lo que este otro método, lo simplificaría, pero también dificultaría la extensión de este.

- Referencias

https://refactoring.guru/es/design-patterns/abstract-factory
https://www.youtube.com/watch?v=CVIpjFJN17U
https://reactiveprogramming.io/blog/es/patrones-de-diseno/abstract-factory
https://informaticapc.com/patrones-de-diseno/abstract-factory.php
https://lineadecodigo.com/patrones/patron-abstract-factory/