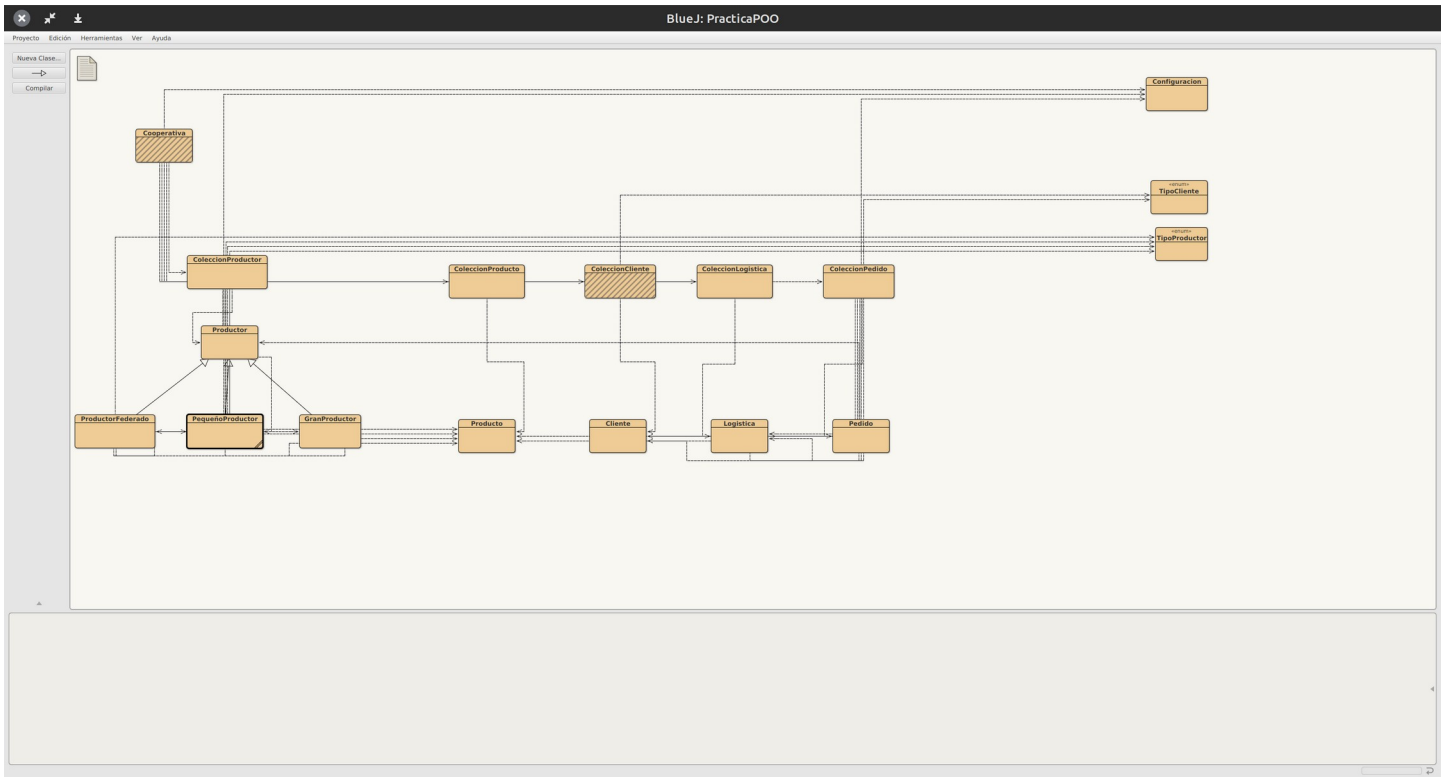
MEMORIA PED POO 2023

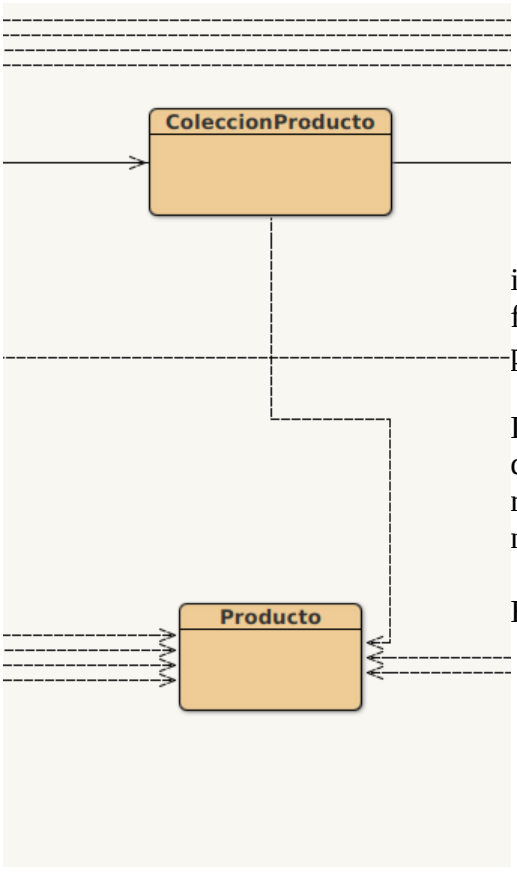
Pablo Ramos Muras



He afrontado esta practica haciendo el siguiente esquema de clases.

Dividí el código en una clase Cooperativa que es la clase principal y 5 grupos de clases, Con sus respectivas colecciones.

1. Producto:

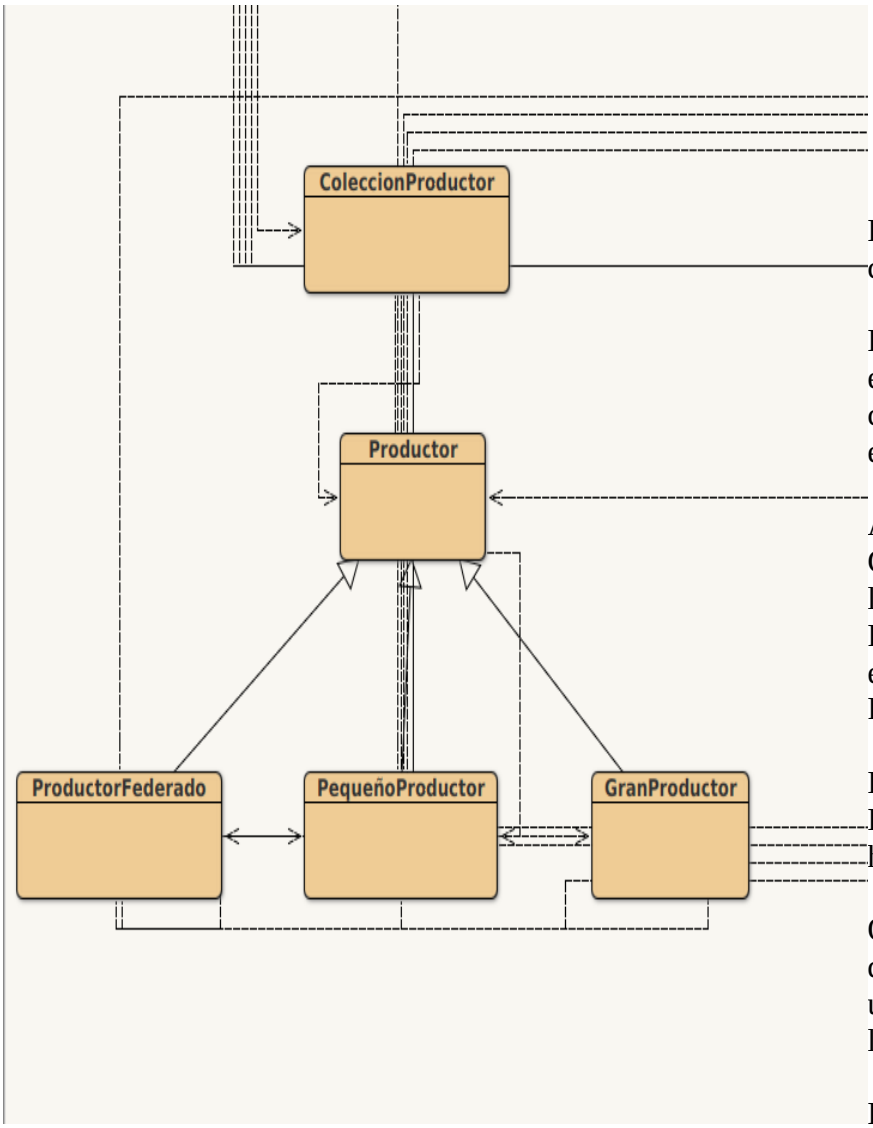


Producto contiene toda la información de producto, incluyendo un HashMap que relacciona los precios con una fecha. Cada vez que este precio se actualiza, se guarda para poder ser visitado más adelante.

La clase ColeccionProducto sirve de Wrapper a un ArrayList de productos, y al mismo tiempo de Factory, teniendo la responsabilidad de la creación y almacenamiento de los nuevos productos.

Está elección de diseño se repetirá con todas las colecciones.

2. Productor:



La clase “Productor” sirve de padre a tres clases, una por cada tipo de Productor.

En su interior contiene todos los campos especificados para el Productor, cambiando el comportamiento para por ejemplo, el ProductorFederado.

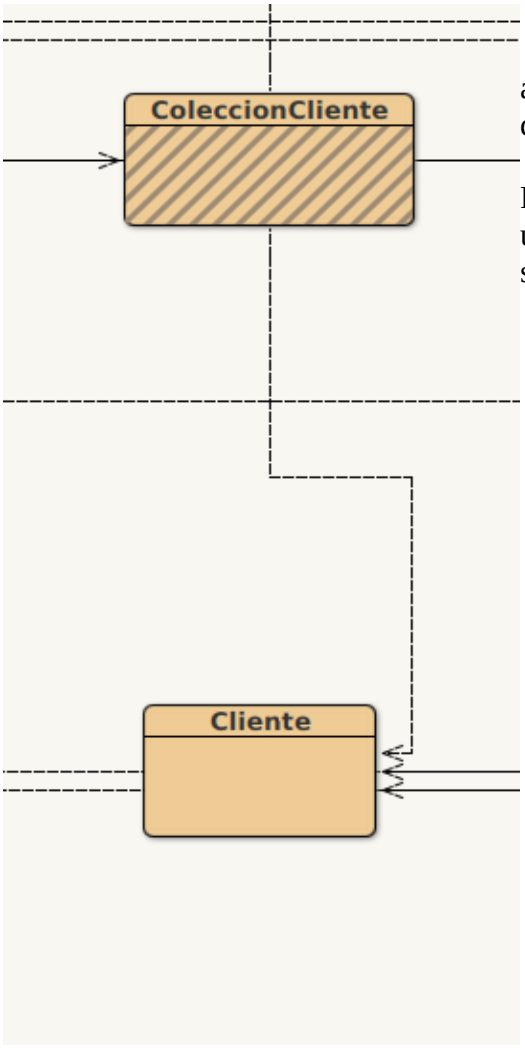
Al igual que con Producto, la clase ColeccionProductor se encarga de la lógica de creación y almacenaje de los Productores, siendo ella la que decide, por ejemplo, si un productor es Grande o Pequeño.

En el interior de productor hay un HashMap que relacciona Producto y hectareas (como un double).

Originalmente, usaba el id como key, ya que todos los objetos de mi practica tienen un IDs único y secuenciales para dentro de la cooperativa.

He decidido que por motivos de facilidad a la hora de obtener los datos, usaría el objeto. Está decisión también se mantiene en la clase pedidos, que se relaccionará con todas las otras clases a traves de la composición.

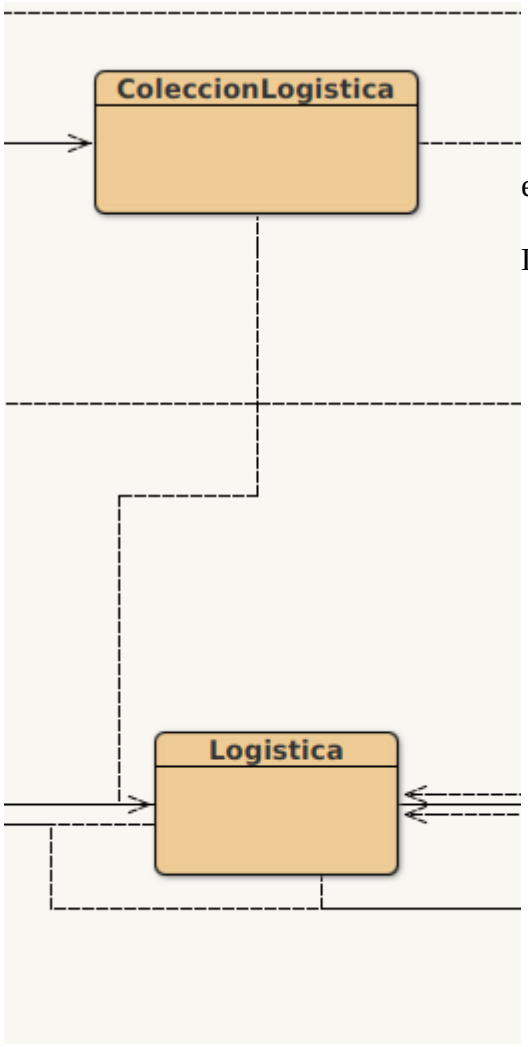
3. Clientes (Distribuidores y Consumidores):



En la clase Clientes he decidido no utilizar la herencia, aunque es una decisión que me ha afectado más adelante, he decidido dejarlo así.

Para diferenciar entre Distribuidores y Consumidores, he usado un tipo Enum, que también utilicé para los Productores, pero solo para filtrado en colección Productor.

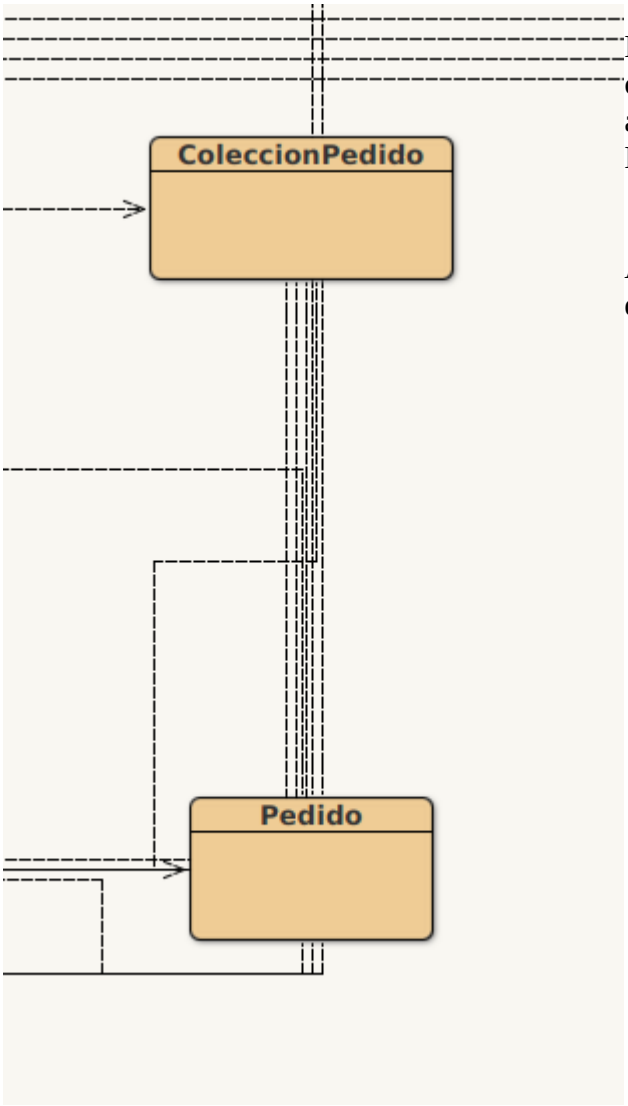
4. Logística:



La logística vuelve a usar el patrón anterior ya especificado en las anteriores clases.

La creación vuelve a ser responsabilidad de la colección.

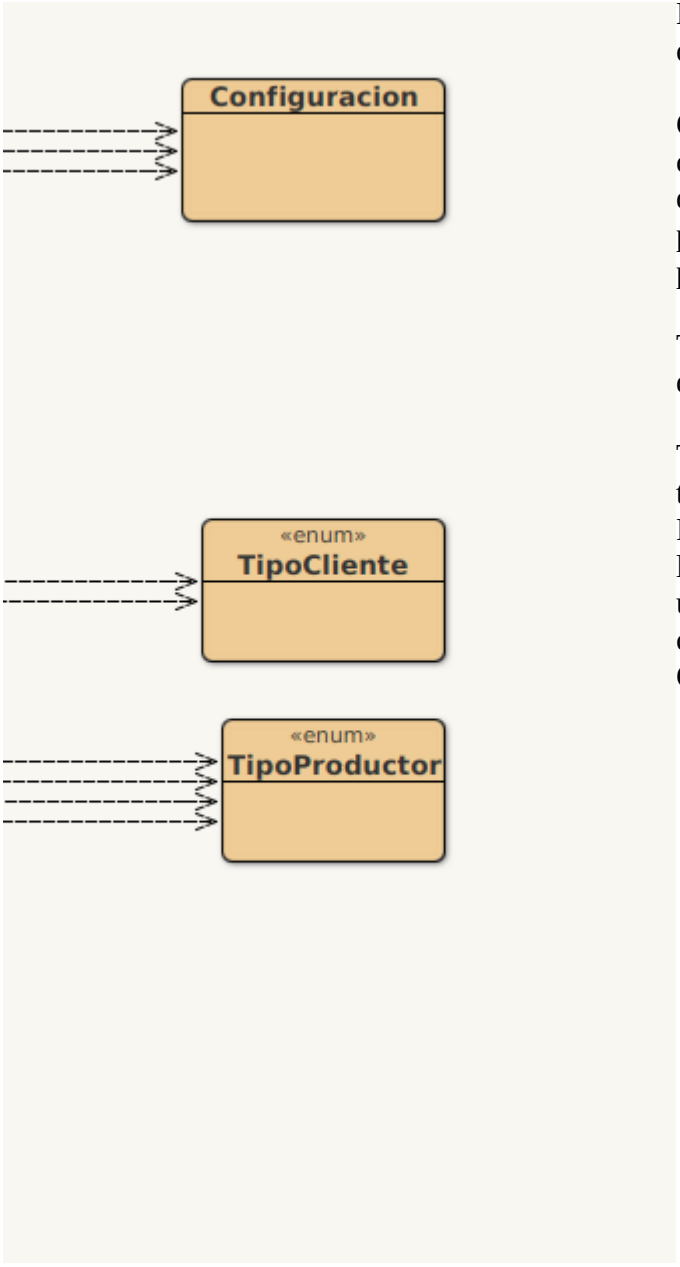
5. Pedidos



La clase pedidos es la más complicada, teniendo por composición instancias de casi todas las clases anteriores, relacionandose de esta forma con Cliente, Logística, Producto y Productores.

A su vez, contiene dos LocalDate, tanto el de la fecha de Creación como el de la fecha de entrega.

6. Externas:



Este grupo de clases solo sirven para completar otras.

CONFIGURACION: Contiene todas las coonstantes para evitar magic numbers, lo he dejado como una clase, pensando, por ejemplo, en poder extraer estas propiedades a un archivo properties.

TipoCliente: Enum que contiene los posibles tipos de clientes. (DISTRIBUIDOR, CONSUMIDOR)

TipoProductor: Enum que contiene los posibles tipos de productor. (GRAN, PEQUEÑO, FEDERADO). A pesar de que para esto se decide la herencia, he decidido implementar un enum para un filtrado, para evitar pasar un String o una Class como parametros en la función en ColecciónProductor.

8. PRINCIPAL

Clase COOPERATIVA. Por composición, se relacciona con todas las clases Colección. Tiene la responsabilidad de mostrar todos los menus y manejar la mayor parte de la información.

7. Notas y conclusiones.

Originalmente, planeaba 3 clases más, que por cuestiones de tiempo no he podido terminar de implementar:

- Actualizador: Clase pensada para hacer MOCK a una API, actualizaría los precios 1 vez por semana, y también calcularía la distancia entre dos puntos A y B. La idea sería que esta clase fácilmente podría ser substituida por otra implementación, usando la inyección de dependencias a la clase cooperativa.
Está clase la sustituí por interacción directa por el usuario.

- Estadísticas: Una clase especializada en la creación y calculo de las estadísticas, fue la última que descarté, ya que seguía en mis planes. Decidí pasar la responsabilidad de esto a las diferentes colecciones, aunque al final han quedado escasas.

- Menú: Clase estática que contendría todos los métodos que manejan el menú principal. Sería el equivalente a la Vista en un Patron MVC (Modelo, Vista, Controlador). Lo he descartado por problemas de tiempo y de dependencias pero me sigue gustando la idea. De tener Vista, el modelo serían las 5 colecciones y el controlador la propia clase cooperativa.

Conclusión:

Me hubiera gustado haber empezado esta práctica antes, ya que muchas ideas se me han quedado en el tintero.