Práctica 3

Ana Isabel Mena Meseguer
José Luis Parra Azor
Elena Vallejo Ruiz
Alejandro Ruiz Salazar

UNIVERSIDAD DE GRANADA

Índice

1 Introducción

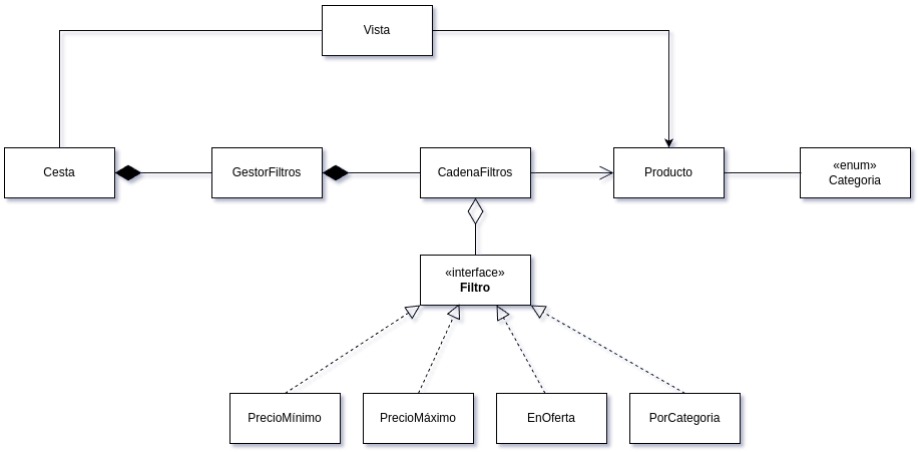
Para esta práctica no hemos utilizado nuestro ejercicio 3 de la práctica anterior, ya que no veiamos una forma clara de como realizar una interfaz para esa idea, asi que hemos tenido que realizar otra.

Nuestra aplicación se basa en la simulación de una interfaz de compra de una tienda, donde podemos visualizar los productos y añadir la cantidad que deseemos a una cesta. Además, podemos filtrar los productos de la tienda según distintos criterios, como la categoria de los productos, por precio mínimo y máximo, o si los productos están en oferta o no.

2 UML

Hemos implementado el patrón de diseño Filtros de Intercepción. Este patrón utiliza una estructura de Tubería y Filtro para procesar datos secuencialmente, lo que permite el pre-procesamiento y post-procesamiento de peticiones y respuestas. Además, mejora la reutilización del código al permitir la combinación de filtros.

Este es nuestro UML:



3 Clases

3.1 Producto

En esta clase creamos los atributos que va a tener nuestro producto, como el nombre, la categoria, el precio y un booleano que dice si el producto está en oferta.

3.1.1 Enum Categoria

Definimos las siguientes categorias que puede tener el producto: cosmeticos, alimentacion, ropa, ninguna.

3.1.2 productos.json

Este json sirve para inicializar una serie de productos que almacenaremos en una lista para utilizarla en nuestra aplicación.

3.2 Cesta

En esta clase implementamos el código de nuestra cesta, que actúa como el cliente en nuestro patrón, es por esto que tiene un GestorFiltros y un Map que contiene los productos que se han incluido en la cesta y el número de veces que se ha introducido a la cesta.

La función AdquirirProducto nos permite incluir un nuevo producto al Map, en el caso de que se haya introducido previamente (es decir, que contenga la key del producto), se aumentará el entero, en caso contrario simplemente se añadirá al map con un 1.

Además de esto la cesta nos devolverá el precio total de la cesta con precioTotal().

3.3 Filtro

Esta clase abstracta nos sirve para implementar los filtros de los que va a disponer nuestra aplicación, por lo que solo dispone del método abstracto "filtrar", que deben implementar los demás filtros para saber cómo deben filtrarse los productos según cada filtro.

3.3.1 EnOferta

Este filtro comprueba toda la lista de productos, y devuelve una lista nueva con los productos que tengan el booleano "descuento" a true. Para ello utiliza el metodo getDescuento() de la clase Producto.

3.3.2 PorCategoria

Este filtro comprueba que la categoría por la que se quiere filtrar no es "ninguna", que es lo que utilizamos para mostrar toda la lista de productos completa. Si es distinto de "ninguna", usa el metodo getCategoria() sobre cada producto de la lista para comprobar si la categoría coincide con la que busca, y la inserta en la lista de productosFiltrados, para después devolver dicha lista (siempre que la categoría a filtrar no sea "ninguna").

3.3.3 PrecioMaximo

Este filtro permite realizar una búsqueda de los productos con un precio igual o inferior al introducido. Para ello, recorre la lista de productos que se le pasa por parámetro, y si su precio es menor o igual que el precio máximo establecido, añade el producto a la lista de productosFiltrados que va a devolver el método.

3.3.4 PrecioMinimo

Este filtro permite realizar una búsqueda de los productos con un precio igual o superior al introducido. Para ello, recorre la lista de productos que se le pasa por parámetro, y si su precio es mayor o igual que el precio mínimo establecido, añade el producto a la lista de productosFiltrados que va a devolver el método.

3.4 FiltroStruct

Esta clase nos sirve para almacenar valores como si fuera un struct (ya que dart no soporta este tipo de dato) que más tarde usaremos para aplicar los filtros de búsqueda seleccionados por el usuario.

3.5 CadenaFiltro

Esta clase actúa como un contenedor para los filtros que se aplicarán a una lista de productos.
- List<Filtro>filtros: Almacena los filtros que se aplicarán a los productos.
- List<Producto>productos;: La lista de productos sobre la que se aplicarán los filtros.
El método filtrar aplica los filtros secuencialmente a la lista de productos y devuelve la lista filtrada. El método addFiltro añade un filtro a la cadena de filtros.

En resumen, CadenaFiltros nos permite almacenar los filtros que se aplicarán a una lista de productos.

3.6 GestorFiltros

Esta clase se encarga de gestionar los filtros que se van a añadir a nuestra aplicación. Contiene un atributo de la clase CadenaFiltros, y el constructor por parámetros inicializa ese atributo con una lista de productos pasada por parámetros. Tiene un método addFiltro que a su vez utiliza el método addFiltro de cadenaFiltros, con un filtro como parámetro. También tiene un método filtrar que llama al método filtrar de cadenaFiltros pasándole por parámetro un atributo del tipo FiltroStruct (explicado anteriormente).

3.7 Utils

Esta clase nos sirve para simplificar el main, que inicializa el gestor añadiéndole todos los filtros y leer el .json, para ello abre el archivo y recorre línea por línea

introduciendo los productos a un vector que luego devolverá.

3.8 Main

Creamos una interfaz que tiene en la parte superior la selección de filtros, debajo muestra un listado de los productos (tras aplicar los filtros correspondientes), desde el cual podemos añadir o eliminar productos a nuestra cesta.

En la parte inferior, tenemos nuestra cesta, en la que salen todos los productos que hemos seleccionado y el precio total de todo lo que llevamos. Además, podemos eliminar productos o añadir más, si así lo deseamos.

El sistema es escalable, ya que solo haría falta ir añadiendo o eliminando los productos al archivo .json según vaya siendo necesario. Además, se podría crear una base de datos, un sistema de pagos...

Para poder realizar la estructura de la interfaz, hemos hecho una combinación de Rows y Columns, dejando espacio entre los diferentes items con SizedBoxes. Para introducir valores o elegir entre los disponibles, hemos hecho uso de DropDownButtons, TextFields, ElevatedButtons...

- Selección de categoría: hemos usado un DropDownButton, que al pulsarlo despliega un menú con todas las categorías disponibles para filtrar.
- Precio mínimo: aquí usamos un TextField donde introducimos el precio mínimo.
- Precio máximo: de la misma forma, se usa un TextField para filtrar el precio máximo.
- Con descuento, aplicar y reiniciar filtros: para estos tres campos hemos usado un ElevatedButton, que es simplemente un botón que realiza una acción al pulsarse.
- Visualizado de productos: tanto para productos filtrados como los de nuestra cesta, usamos un ListView, que nos permite mostrar el contenido de un vector de productos.
- Botones para añadir o eliminar productos: para todos ellos hemos usado IconButtons, que permiten realizar una acción tras ser pulsados (similar a los ElevatedButton) y asignar una imagen al botón (papelera, carrito de la compra...).

Figure 1: Ejemplo con la cesta vacía

Figure 2: Ejemplo tras introducir algunos productos