

**DISEÑO Y**

**MANTENIMIENTO**

**DEL SOFTWARE**

**Proyecto:**

**Gestor de una Lista de la Compra**

Autores:

Sergio Barbero Bascones  
Víctor de Castro Hurtado

Contenido

[Enunciado 3](#_Toc499491746)

[Funcionamiento 4](#_Toc499491747)

[Estructura 5](#_Toc499491748)

[Patrones 6](#_Toc499491749)

[Conclusiones 7](#_Toc499491750)

# Enunciado

El proyecto consistirá en la elaboración de una pequeña aplicación de gestión de una lista de la compra, el lenguaje de implementación será Java.

Requisitos de la aplicación en forma de historias de usuario:

1. Como evaluador quiero acceso al código y su historial para ver la implementación.
2. Como evaluador quiero una memoria para evaluar el uso de buenas prácticas y patrones de diseño.
3. Como evaluador quiero instrucciones para poder construir y lanzar la aplicación.
4. Como cliente quiero poder añadir y quitar productos de la lista.
5. Como cliente quiero poder aumentar y reducir la cantidad de cada producto de la lista.
6. Como cliente quiero poder marcar elementos de la lista como comprados.
7. Como cliente quiero que la lista se mantenga cuando vuelva a abrir la aplicación.
8. Como cliente quiero que al menos se pueda interactuar con un GUI.
9. Como cliente quiero poder marcar productos como favoritos, para que no tenga que volver a escribir su nombre.
10. Como cliente quiero que la aplicación pueda tener nuevas formas de guardar la lista en el futuro.
11. Como cliente quiero que la aplicación pueda tener nuevas formas de interacción en el futuro.

# Funcionamiento

Nuestra aplicación de la lista de la compra funciona de la siguiente manera:

Primero debemos ejecutarlo, ya sea mediante los ejecutables (modo usuario) o desde el editor (modo developer). Para ejecutarlo en este último modo y que no de problemas, debemos poner la variable ‘fsUserVersion’ a false para evitar problemas de rutas con los ficheros y carpetas.

El programa comienza preguntándonos el nombre de la carpeta donde queremos guardar nuestra lista, o leerla si ya existe.

A continuación, nos pedirá el nombre de la lista, para comprobar si existe su correspondiente fichero (en cuyo caso la leerá) o no (la crea vacía).

Después de leer la lista o crearla, llegamos a la parte del menú textual (línea de comandos), el cual nos irá guiando en cada paso que demos, y que cuenta con todas las operaciones posibles sobre la lista y sus productos (añadir nuevo producto; eliminar producto existente; modificar el nombre, cantidad o precio de un producto existente; marcar un producto como favorito o comprado; previsualizar la lista con los productos actuales de la lista; guardar la lista en el fichero; salir sin guardar los cambios de la lista en el fichero).

Para añadir un producto, se le pide al usuario un nombre (el cual se compara para saber si ya existe en la lista), y una cantidad de ese producto.  
En caso de que la cantidad sea 0 o menor que 0, se le preguntará al usuario si quiere guardarla a pesar de ser una cantidad no válida, en cuyo caso se guardará como favorito para que no se elimine.  
El nuevo producto se guarda en la posición de la lista determinada por la ID, que se calcula en función del tamaño de la lista.

Para eliminar un producto, se le pide al usuario la ID de ese producto (la cual se compara para saber si hay un producto con esa ID), en cuyo caso se borra.

Para modificar un producto, le tenemos que especificar la ID de dicho producto, y a continuación nos preguntará qué parámetro queremos modificar, si el nombre, la cantidad de producto o su precio. También tenemos una opción de cancelar modificación, devolviéndonos al menú principal.  
En caso de que el nuevo nombre ya exista, no se cambiará, y para que el precio se elimine del producto debemos escribir cualquier valor negativo, que se representará en nuestra lista como un guión (por defecto).

Las siguientes opciones son marcar el producto como favorito y como comprado. En estos casos se cambia el valor de sus respectivas variables al contrario del que tengan (si la variable comprado estaba a false y se pulsa la opción de comprar, se pondrá a true, y viceversa).   
En caso de que se establezca un producto como comprado y no fuera favorito, se eliminará de la lista.

La opción VIEW nos muestra una preview de nuestra lista (aún sin guardar en el fichero) por pantalla, con todos los campos de los productos, para cerrar la preview basta con pulsar cualquier tecla.

La opción SAVE guarda la lista que hemos estado modificando hasta ahora en el fichero que le hayamos especificado al principio, pero no dejará de ejecutarse, sino que permanecerá en el menú principal por si queremos seguir editando la lista después de guardarla.

Por último, la opción EXIT nos permite dejar de editar la lista y detener la ejecución del programa sin guardar dicha lista en su correspondiente fichero.

# Estructura

Nuestro programa se divide en una clase principal (**Main**), que se encarga de llamar a la interfaz en modo texto que vamos a usar en nuestro caso (UI) que hereda de **Manager**, clase de la que también hereda la interfaz gráfica.

Esta interfaz (**UI**) se encarga de pedir todos los datos que hemos señalado en el apartado anterior para el correcto funcionamiento de la aplicación. Además, indica por pantalla en todo momento lo que se va a realizar o se está realizando, guiando al usuario paso a paso.

Desde este menú en modo texto, y en función de las opciones que introduzca el usuario, se acceden al resto de métodos o clases (tiene algunas llamadas a las clases de Utilities y ProductList). Las opciones BUY, FAV y MOD hacen cambios directamente sobre el producto especificado en la clase ChosenProduct, mientras que ADD, DEL y SAVE interactúan con el objeto Singleton de la clase ProductList. Las opciones VIEW y EXIT no modifican ningún valor, simplemente accede al Singleton del ProductList para mostrar la lista por pantalla (en el caso de VIEW).

La clase **Utilities** es la que más dependencias tiene, ya que se encarga de funcionalidades varias que se usan varias veces (leer un fichero, escribir la lista en otro, establecer el PATH en el que irán los ficheros y directorios, comprobar si un producto existe en una lista de varias formas -por nombre y por Id-, comprobar si existen el directorio y el fichero o crearlos en caso contrario, etc).

A continuación, tenemos una clase **Product**, que se encarga de definir un producto general, con ID y nombre, el cual es necesario para construir nuestro **ChosenProduct**, al que le añadimos una cantidad, de manera que los valores necesarios para que un producto exista en nuestra lista son: una ID (que se genera automáticamente), un nombre (que es único en la lista), y una cantidad. El resto de valores se inicializan por defecto, de manera que podamos cambiarlos más tarde. Estos valores por defecto son:  
Comprado: NO ; Precio: - ; Favorito: NO

En nuestra clase ChosenProduct se realizan las operaciones más básicas sobre un ChosenProduct. A estas funciones son las que se llama la mayoría de las veces desde Utilities o UI.

Al realizar en esta clase las operaciones más básicas, dejando el grueso de la lógica de nuestro programa en otras clases, se facilita la implementación de la **programación funcional**, facilitándonos el trabajo a largo plazo.

La lista propiamente dicha se almacena en **ProductList**, que se ha implementado como un Singleton, para facilitar la implementación y asegurarnos las interacciones sobre la misma lista.

Como elementos extra, hemos decidido crear una serie de ejecutables para facilitarle al usuario su ejecución. Como se especifica en el README.txt del repositorio, dichos ejecutables se encuentran en la carpeta **ShoppingList\_Portable**, y para ejecutarlo sólo hay que ejecutar el .bat si estamos en Windows o el .sh si estamos en Linux.

En la carpeta bin se encuentra el .jar que compone nuestra aplicación ejecutable, y además se encuentra un segundo fichero README.txt por si hay alguna duda.

# Patrones

En cuanto a patrones de diseño utilizados hemos utilizado Singleton para la clase ProductList, así llamando a la clase nos aseguramos de que solamente haya una instancia de la lista, siendo fácil el acceso a esta lista en cualquier parte del código.

Hemos barajado la posibilidad de utilizar el patrón fachada para la lista y sus métodos, pero en este caso no es necesario debido a la baja complejidad del proyecto.

Además, hemos barajado la posibilidad de utilizar el patrón Prototipo para la interfaz. Tenemos una clase que se llama Manager de la cual derivan tanto la Interfaz de texto (UI) como la gráfica (GUI). Pero al no existir nada más que una instancia de cada una no es necesario.

No hemos realizado ningún patrón más a primera vista, aunque hemos simplificado bastante nuestro código intentando usar programación funcional, por ejemplo, al reducir el número de funciones que hacían cosas parecidas, y combinarlas en una sola. Esto se puede ver en las versiones prematuras del código, dónde teníamos varias funciones de la clase UI que le preguntaban al usuario cosas con la siguiente estructura:

String askUserName(){  
print(“Introduce el nombre: ”);  
new Scanner(System.in).next();  
}

De manera que hicimos una función que pidiera los datos con el Scanner, y otra que imprimiera un mensaje genérico pasado por parámetro, quedando la estructura dividida de la siguiente forma:

String getScanner(){  
return new Scanner(System.in).next();  
}

String askUser(String text){  
print(text);  
return getScanner();  
}

Al igual que en el caso anterior, hemos intentado factorizar al máximo el resto de funciones, haciéndolas lo más sencillas posibles, para facilitar su reutilización en la siguiente parte del proyecto.

# Conclusiones

Para la realización del proyecto hemos pensado en la usabilidad como primer factor. Para el usuario tiene que ser fácil manipular su lista.

En cuanto al uso de patrones no nos hemos obsesionado con la utilización de ellos, sino que hemos planificado la codificación del código del modo mas natural, surgiendo los patrones de forma innata.