

**DISEÑO Y**

**MANTENIMIENTO**

**DEL SOFTWARE**

**Proyecto:**

**Gestor de una Lista de la Compra**

**(Segunda Entrega)**

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). De la primera forma tenemos dos ejecutables separados, uno para ejecutarlo en modo GUI interactiva y otro en modo UI con ventana de comandos. De la segunda forma, deberemos cambiar la variable “GUI” de la clase principal (main.java) dependiendo si queremos ejecutarlo en modo GUI interactiva o UI textual.

NOTA: El programa guarda automáticamente la lista que creemos (o leamos) en la carpeta “data”, en el archivo “database.tsv”, ya que está implementado por defecto el sistema de persistencia por fichero. El sistema de persistencia por base de datos no está implementado, aunque se ha dejado indicado en cada método cómo habría que hacerlo, donde irían las conexiones, y cómo se relacionarían los métodos y consultas entre si.

El programa comprueba internamente que existe dicho directorio y fichero (lo mismo pasaría con la base de datos), de manera que crea la lista si no existe, y la lee en caso contrario.  
Después de leer la lista o crearla, tenemos varias opciones que cambian de una interfaz a otra.

En caso de que sea interfaz gráfica:  
Se mostrarán en todo momento los productos que hay en la lista, junto con todos sus atributos (excepto la ID). Al lado tendremos unas checkboxes para seleccionar el producto sobre el que realizar las operaciones, y una pestaña en la parte superior que nos permite hacer una selección múltiple.  
Cuando seleccionemos varios productos, podremos eliminarlos, o marcarlos como favoritos o como comprados (si ya estaban marcados se desmarcarán), pero no podremos modificar varios productos a la vez, para ello tendremos que seleccionar sólo 1 producto, lo que nos abrirá una ventana emergente para poder modificar su nombre, su precio o su cantidad.  
Además, tendremos un botón de añadir un nuevo producto, y de guardar cambios, lo que escribirá los cambios que hayamos realizado en nuestro sistema de persistencia.

NOTA: si cerramos la ventana sin guardar los cambios, se perderán ya que no hay ventana emergente de confirmación.

En caso de que sea interfaz en modo texto:  
Tenemos un primer menú principal, en el que podremos añadir, eliminar, modificar (cambiar nombre, cantidad o precio), marcar como favorito o comprado un producto.

NOTA: A diferencia del modo interfaz gráfica, todos estos cambios se realizarán de uno en uno.

También tendremos un botón de salir, el cual sí que nos preguntará si queremos guardar los cambios que se hayan realizado (sólo cuando se hayan realizado cambios), y un botón de guardar propio, por si queremos guardar los cambios, pero seguir editando la lista. Además, y a diferencia del modo gráfico, tendremos una opción de ayuda, que mostrará ayuda básica sobre cada una de las opciones, desarrolladores y nombre del programa. Se mantiene la opción de la entrega anterior en la que se mostraban todos los productos en la lista con sus características (opción VIEW), para poder ver qué productos podemos modificar o cuales siguen en la lista.

Las opciones de modificación de un producto se mantienen en un menú aparte que se nos muestra cuando decidimos entrar en la opción de modificar el producto. También se puede cerrar dicho menú sin realizar ningún cambio.

# 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 (**UI**) o la interfaz gráfica (**GUI**). La primera de ellas hereda de **UIManager**. Lo hemos realizado de esta manera ya que en un futuro nos puede interesar realizar otras implementaciones, ya sea en modo texto o gráfico, los cuales pueden acceder a métodos comunes del mánager.

La **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. En el caso de la **GUI** es diferente, ya que es más intuitivo, no hace falta avisarle al usuario de lo que está realizando en todo momento, sólo cuando se necesitan ventanas emergentes.

Una vez elegida la interfaz, en ambas se le preguntará al usuario qué sistema de persistencia desea usar, una mediante un mensaje en la consola, y otra mediante una ventana emergente (no se ha implementado esta funcionalidad porque el sistema de persistencia de la base de datos no se ha implementado, simplemente se ha dejado indicado, de modo que no tiene sentido preguntarle al usuario cual prefiere si se va a usar el modo de fichero de todas formas.)  
Cuando se le pregunta al usuario por el sistema de persistencia, se realizan comprobaciones transparentes al usuario para comprobar que existe el fichero/base de datos donde debería estar, y lee los datos que haya guardados. Si no existía el fichero o la base de datos, se creará una nueva, creando una lista vacía.

A partir de este momento se trabajará sobre una lista local de productos (List<Product>), para evitar leer y escribir en el fichero continuamente, o incluso más importante, evitando hacer tantas llamadas a la base de datos, ahorrando tiempo y posibles fallos de conexión.

NOTA: se ha implementado de esta forma ya que no consideramos relevante que se pierdan algunos datos si incrementamos la velocidad de trabajo del usuario lo suficiente. En caso de que se esperen pérdidas de datos más grandes o con sistemas con fallos continuos, se podría llegar a implementar el sistema de persistencia de otra manera, reflejando cada cambio que hagamos en el fichero o la base de datos, y aunque se perdería eficacia, se ganaría en seguridad.

La antigua clase **Utilities** se ha dividido, de manera que todas las operaciones que realizamos sobre la lista con productos ahora se encuentran en **ProductListOperations**, y todo lo relativo a rutas, ficheros, etc se ha movido al correspondiente fichero de persistencia mediante fichero (**FilePersistence**).  
Seguimos teniendo un **GenerlUtilities**, aunque sólo tiene métodos generales, relacionados con el tipo de sistema operativo para construir las rutas de ficheros (usamos en los sistemas de persistencia).

La clase **ChosenProduct** se ha eliminado, ya que estábamos realizando operaciones sobre productos en ella y en la clase **Product**, de manera que todas las operaciones relacionadas se han quedado en esta última clase, dejando dichas operaciones lo más sencillas posibles, trasladando parte de la lógica común a los métodos de otras clases (**ProductList** y **ProductListOperations**).

La lista con productos se sigue almacenando en **ProductList**, a modo de Singleton, para facilitar la implementación y asegurarnos las interacciones sobre la misma lista a lo largo de todo el programa.

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 **ShoppingListUI\_Portable**, en el caso de que se quiera utilizar la interfaz en modo texto, o en la carpeta **ShoppingListGUI\_Portable**, en el caso de que se quiera utilizar la interfaz gráfica, y para ejecutarlo sólo hay que ejecutar el .bat si estamos en Windows o el .sh si estamos en Linux.

# 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 **UIManager** de la cual deriva la Interfaz de texto (**UI**). Pero al no existir nada más que una instancia, no es necesario. Lo mismo ocurre en el caso del sistema de persistencia: tenemos una clase abstracta **PersistenceManager**, de la cual derivan el sistema de persistencia de ficheros (**FilePersistence**) y el sistema de persistencia de la base de datos (**DBPersistence**), y aunque de momento no es necesario, en un futuro se podrían añadir otras formas de persistencia que utilicen el mismo PersistenceManager que estas dos.  
Aunque no nos terminaba de convencer el patrón prototipo, de manera que al final hemos optado por el patrón **plantilla**. El método PersistanceManager es la clase abstracta de la cual heredan los métodos FilePersistence y DBPersistence.

Para crear la interfaz gráfica en un principio creamos la clase **MainWindow**, que heredaba de Application. Esto nos hacía dudar sobre el uso de GUI, ya que había métodos como askUser() que había que redefinir y que no tenían sentido en esa clase, ya que solamente debería pertenecer a interfaz por texto. Al final la solución más acertada ha sido mover todos esos métodos que solo utilizaba la interfaz de texto de UIManager a UI y renombrar la clase MainWindow a **GUI**, eliminando la clase previa GUI.   
Para guardar el estado de la lista desde la interfaz gráfica hemos recurrido a un patrón **adaptador**. De esta manera instanciamos un UIManager en la clase GUI, así podemos escribir en la base de datos.

No hemos realizado ningún patrón más a primera vista, aunque hemos simplificado bastante nuestro código intentando reducir el número de funciones que hacían cosas parecidas, y combinarlas en una sola, haciéndola más genérica. De igual manera que métodos en los que antes teníamos demasiadas líneas de código les hemos reducido todo lo posible (ver **askUserInfo** en la clase UI).

Hemos creído conveniente mencionar el patrón de diseño de **Modelo-Vista-Controlador**, ya que tiene muchas similitudes con nuestra aplicación.

En el apartado del Modelo, tenemos nuestra forma interna de tratar los datos (clase productList, definiendo una lista de productos Lsit<Product>).

En el apartado de vista, tenemos las interfaces de interacción con el usuario (clases UI y GUI como interfaces en modo texto y gráficas).

En el apartado de Controlador, tenemos los dos sistemas de persistencia (filePersistence y DBPersistence).

Además, tenemos un patrón **observable**, ya que la información que se le muestra al usuario se actualiza cada vez que realiza un cambio en el modo gráfico (GUI), y en el modo texto (UI) no se muestra automáticamente, aunque puede utilizar el equivalente del menú principal (VIEW) para ver los cambios.

# Conclusiones

Para la realización del proyecto hemos pensado en la usabilidad como primer factor. Para el usuario tiene que ser fácil manipular su lista y ver los resultados instantáneamente en el caso de la interfaz gráfica.

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 más natural, surgiendo los patrones de forma natural.

Como se pide que se puedan implementar más funcionalidades en un futuro, se han dejado algunos métodos sin completar, como los de la clase **DBPersistence**, que se comunicaría con la base de datos, realizaría la conexión y leería y guardaría los datos.   
No hemos tenido tiempo suficiente, y no consideramos que sea el objetivo de la asignatura el realizar una base de datos completamente funcional, de modo que hemos dejado indicado cómo se implementaría.