UNIVERSIDAD DE BURGOS

Diseño y mantenimiento de software

Memoria del proyecto.

2ª Entrega.

Marcos Orive Izarra

Índice

Tabla de contenido

1. Introducción

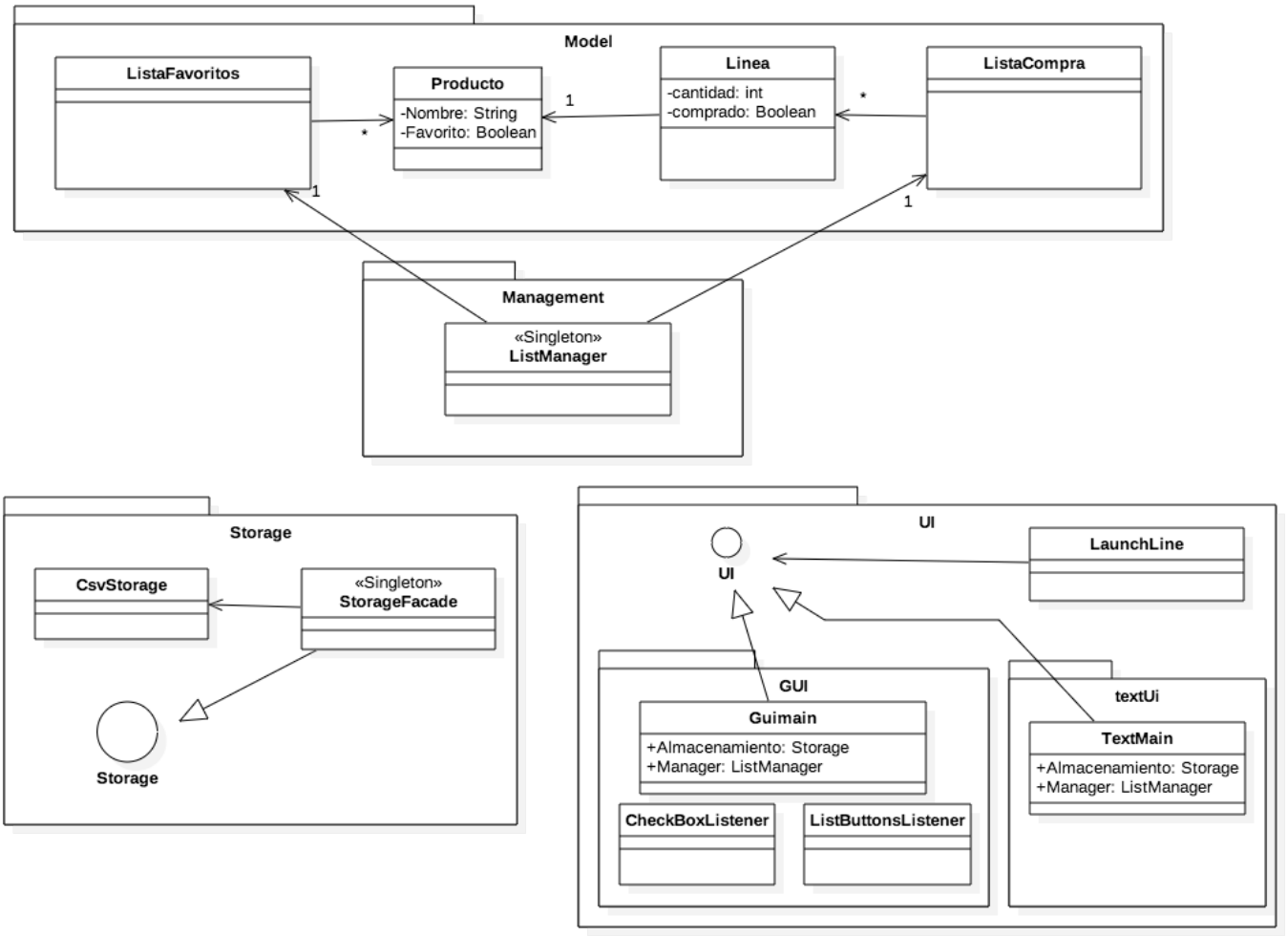
Como parte de la asignatura de Diseño y mantenimiento de software hemos tenido que diseñar una aplicación para gestionar la lista de la compra.

El objetivo de esta memoria es recopilar el uso de buenas prácticas y patrones de diseño a la hora de desarrollar la aplicación pedida, de forma que se facilite la corrección al profesor.

Esta segunda entrega he implementado tanto lo que se requería para la entrega como las mejoras propuestas por el profesor en la corrección de la primera entrega. También he rehecho el diagrama UML que se puede ver en el siguiente punto.

2. Diagrama de clases

En esta segunda entrega he corregido los errores anteriores, añadido y actualizado el nuevo de diseño de algunas partes de la aplicación.



3. Patrones Utilizados

Fachada: Como en la primera entrega, he utilizado el patrón fachada para el almacenamiento y para modelo. Ahora ListManager sí es una fachada, pues he sacado el conocimiento de la lista de favoritos y su uso puede ser completamente opcional. StorageFacade ha sufrido otro cambio importante: Ahora implementa la interfaz UI, más adelante en otro apartado hablaré del porque y las consecuencias de esto.

Singleton: No ha habido cambios significativos en este aspecto desde la primera entrega. ListManager y StorageFacade siguen siendo singleton acompañando su cualidad de Fachada.

Observador: al implementar una interfaz gráfica he creído necesario utilizar el patrón observador para la checkbox y los botones. Para ello he implementado el patrón en dos clases: CheckBoxListener y ListButtonListener, que "escuchan" respectivamente los estados de la Checkbox y de los botones y realizan acciones en consecuencia.

4. Buenas prácticas

Documentación: He completado la documentación del proyecto. Respecto al lenguaje coloquial como por parte del profesor no había problemas he decidido mantenerlo. Igualmente, lo tendré en cuenta para futuros desarrollos.

Métodos cortos: He seguido manteniendo este principio en las partes nuevas desarrolladas para el proyecto. Además he intentado mejorar un poco la clase TextMain y creo que ha quedado algo más clara.

Inversión de dependencias: Lo primero que pensé al desarrollar la aplicación fue el modelo de datos (la lógica del negocio) así que las clases de más alto nivel no dependen de las clases de bajo nivel, si no de las abstracciones.

5. Para añadir sistemas de almacenamiento e Interfaces.

El diseño que he implementado para poder añadir otros sistemas de almacenamiento e interfaces es el siguiente.

Para el almacenamiento he creado la **interfaz Storage**, que contiene los métodos para leer y escribir listas de la compra y favoritos. Lo ideal (y lo que ocurre en mi caso) es que una fachada implemente Storage y complete los métodos necesarios según el sistema que quiera facilitar. De esta forma, para añadir un nuevo sistema de almacenamiento el único cambio que habría que realizar en el código de la aplicación sería instanciar una nueva fachada que implemente UI. Y no habría que cambiar ninguna línea de código más. Por ejemplo:

```
private Storage almacenamiento = StorageFacade.getInstace();
```

Podría cambiarse por

```
private Storage almacenamiento = DataBaseFacade.getInstace();
```

Y no habría que hacer más cambios (aparte de implementar el sistema de almacenamiento nuevo, claro).
De hecho, sería hasta posible reutilizar la misma fachada. Pero personalmente veo más sencillo desarrollar una nueva ya que desarrollas un sistema de almacenamiento nuevo.
Esto logra un desacople entre el sistema de almacenamiento y el resto de la aplicación que simplifica mucho poder añadir nuevos sistemas.

Para añadir una interfaz gráfica nueva he realizado algo parecido. He definido la **interfaz UI**, que simplemente tiene el método execute(). Una nueva interfaz

simplemente debería implementar UI y comenzar su ejecución en el método execute. Ya lo he hecho con mis dos interfaces, TextMain y GuiMain.

Ahora la aplicación se lanza desde la clase LaunchList, que instancia una interfaz según queramos lanzar una u otra (en este caso según los argumentos del main de java, pero podría hacerse de otra forma). En cualquier caso simplemente es necesario desarrollar la interfaz, instanciarla en la clase LaunchList y llamar al método execute().

6. Otras mejoras

- He corregido la clase Textmain para que no sea monoestado y creo que GuiMain la he desarrollado de forma que tampoco lo es.
- He dejado de utilizar toString() para imprimir en las interfaces. Ahora sin las propias interfaces las que se encargan de ellos.