Laboratorio L09
Singleton

Objetivos

- Ver la necesidad y uso del Patrón Singleton

Herramientas a utilizar

- Eclipse

Entregas

Fichero **ZIP** con la **exportación del proyecto** del laboratorio L09. Este fichero debe contener el **código** (fichero exportado de Eclipse) y la **documentación** (fichero **.doc** generado desde Eclipse).

1. Nombre del fichero: *apellidoNombre_L09*.zip
2. La entrega debe ser **individual** y a través de eGela

Tareas a realizar en el laboratorio

Todos los cambios introducidos deben reflejarse en la documentación. Asegúrate de que al finalizar el laboratorio, las clases están totalmente documentadas.

1. Importa el proyecto que contiene los laboratorios L06 y L07 y cambia su nombre a apellidoNombre_L09.
2. Haz lo siguiente en la clase `SuperOnline`:
    a) Crea otro objeto de tipo `Stock` (`stockProba`) y añade en la instancia `stockProba` recién creada un **nuevo producto**. En este momento, los productos en stock pueden estar recogidos en dos instancias diferentes de la clase `Stock`.
    b) Muestra en pantalla la cantidad de productos en stock que tiene el supermercado.
    c) *Reflexión:* ¿tiene sentido que los productos en stock del supermercado estén separados en dos objetos (instancias de `Stock`) distintos? ¿No sería más lógico que todos esos productos estuvieran en el mismo lugar (misma instancia de `Stock`)?

    El **patrón Singleton** nos permite conseguir este objetivo, garantizando que sólo puede haber una instancia de la clase `Stock`.

    La clase `Stock` tendrá que implementarse siguiendo las pautas establecida por el patrón *Singleton*.
3. Cambia la clase `Stock` para que implemente el patrón *Singleton*.
4. Corrige los errores que hayan surgido en la clase `SuperOnline` como consecuencia de los cambios en la clase `Stock`.

5. ¿Qué ocurre ahora con las dos variables de tipo `Stock`? ¿Tienen la misma información? ¿Hemos conseguido nuestro objetivo?
6. Elimina el código del método `main` que está en la clase `SuperOnline`, después crea un método que defina un menú para hacer la gestión de la tienda. Esta clase debe tener los siguientes atributos y métodos estáticos:

   Atributos:

   a) `mStock`: referencia a la **única instancia** de `Stock` que puede haber.

   b) `scanner`: referencia al objeto de tipo Scanner para poder leer la información desde el teclado.

   Métodos:

   a) `main`. El programa principal tendrá que:

   - ✓ Obtener la instancia de `Stock`.
   - ✓ Abrir la comunicación con el teclado para poder realizar las lecturas.
   - ✓ Cargar los productos.
   - ✓ Elaborar un menú que se utilizará para comunicarse con el usuario en diferentes ocasiones. El menú constará de varias opciones. Para facilitar su programación se deben implementar los métodos auxiliares que se describen a continuación:
     1. Escribir en pantalla la información de todos los productos en `Stock`.
     2. Eliminar un producto del `Stock`. Para ello se solicitará al usuario el código del producto a eliminar.
     3. Ordenar los productos del `Stock` según su identificador.
     4. Mostrar la lista de productos «envíables».
     0. Finalizar la aplicación.
   - ✓ Cerrar la comunicación con teclado.

   b) `loadProducts`: método auxiliar y estático que almacena productos en `Stock`. La implementación de este método está en eGela.

   c) `printMenuReadOption`: Escribe en pantalla las opciones del menú del programa principal y lee y traduce la opción elegida por el usuario. Si no existe la opción elegida, muestra un mensaje de error y vuelve a leer la opción hasta que el usuario teclee alguna existente.

   d) `readProductCode`: Pide al usuario el código de un producto, lo lee y devuelve el producto que tiene dicho código. Si no hay ningún producto con ese código, informa al usuario y le pide que teclee otro código. Este proceso se repite hasta que el usuario teclee el código de algún producto existente.

7. Añadir los siguientes métodos en la clase Stock:

   a) `containsProduct`: Dado un producto, devuelve *true* si dicho producto está en `Stock` y *false* en caso contrario.

   b) `obtainProduct`: Dado un código, devuelve el producto del `Stock` que tiene dicho código. Si no hay ningún producto con ese código, devuelve *null*.

c) `updateAmount`: Dado el código de un producto y una cantidad, actualiza la cantidad del producto que tiene que tiene dicho código, asignando a su atributo *contidad* el valor pasado como parámetro. Si no hay ningún producto con el código buscado, escribe un mensaje de error en la salida estándar y pide de nuevo el código el código del producto. La cantidad para hacer la actualización se mantiente.

d) `obtainProductListToOrder`: devuelve la lista de códigos de productos (del ArrayList) con una cantidad de unidades inferior al número recibido por parámetro.

e) `removeAllProductsWith0Units`: Elimina del Stock los productos con 0 unidades.

f) `removeProducts`: recibe como parámetro la condición de retirada (String) y elimina todos los productos que cumplan dicha condición. Este método, además, debe devolver una lista con todos los productos eliminados.

8. Añadir en la clase `SuperOnline`, llamadas para probar todas las opciones del menú previamente descrito.

Ejercicio complementario

9. Una vez probados todos os métodos de la clase descritos en los apartados anteriores, completa las opciones del menú con la posibilidad de añadir un nuevo producto al `Stock`. Para ello implementa el método `addNewProduct`, que lee teclado la información del producto que el usuario desea añadir, crea el producto y lo añade al `Stock`. A la hora de realizar este método, ten en cuenta:

10. Después de implementar y probar todos los métodos previamente indicados, completa nuestro menú añadiéndole una opción más que ofrezca al usuario la posibilidad de añadir un nuevo producto al `Stock`. Para ello, debes incluir la nueva opción en el menú e implementar un método adicional, `addNewProduct`, que, lea del teclado la información del nuevo producto que el usuario quiere añadir, cree el producto correspondiente y lo añada al `Stock`. A la hora de realizar este método, ten en cuenta:

    a) Que los productos tienen algunas características en común, pero otras dependen del tipo del producto que se va a añadir.

    b) Por lo tanto, será necesario preguntar al usuario qué tipo de producto quiere crear y, a continuación, pedirle los valores de los datos en función de su tipo, es decir, tanto de los atributos heredados como de los específicos de la clase de producto a crear.