Laboratorio S7
Excepciones

Objetivos

- Conocer el tratamiento de excepciones en Java: definición, captura y elevación.
- Seguir practicando con la documentación en JavaDoc
- Trabajar con la Entrada/Salida de ficheros
- Trabajar con el Debugger

Herramientas a utilizar

- Eclipse
- JavaDoc

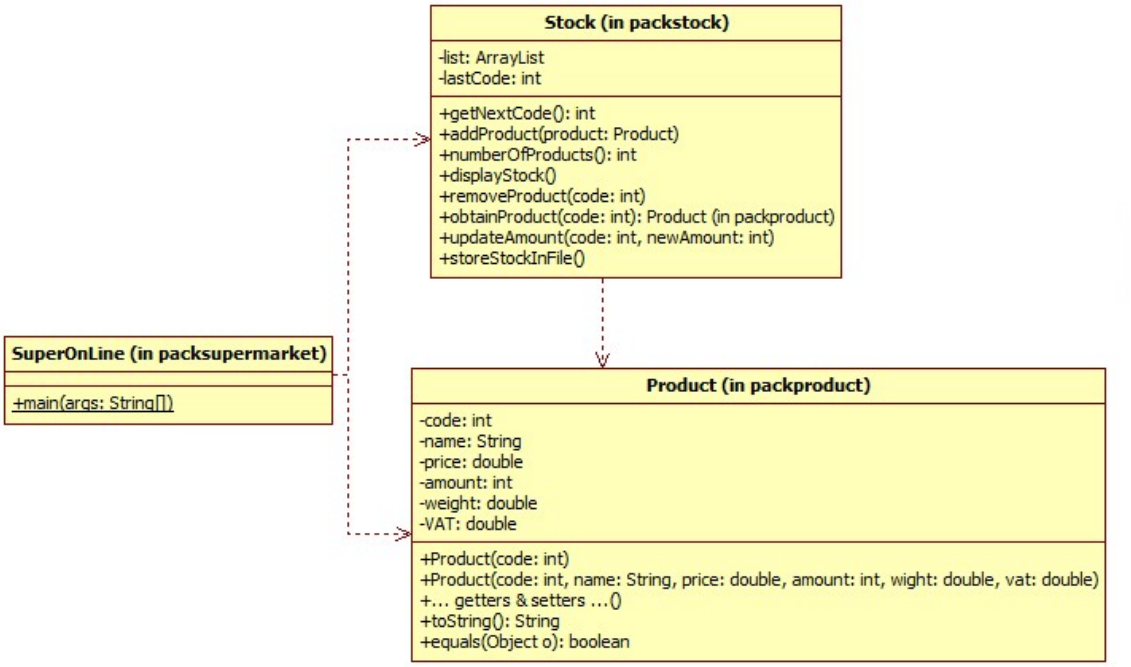
Entregable

Se debe entregar un fichero **ZIP** con los **Proyectos Exportados** del laboratorio S7.

- Nombre del fichero: *apellido_nombre*.zip.
- La **entrega** es **individual** y se subirá a eGela
- **Fecha límite de entrega**: Viernes 12 de marzo a las 23:55

Contexto

En este laboratorio trabajaremos con una aplicación que gestiona los productos y el Stock de una tienda virtual.
El proyecto tiene dos clases principales: **`Product`** y **`Stock`**. El siguiente diagrama de clases UML muestra las clases correspondientes a este laboratorio incluyendo el paquete en el que se definen.
El **`Stock`** tiene una lista que contiene los **`Product`** disponibles en un momento concreto y el **`Product`** toda la información relevante sobre cada producto. El código de **`Product`** (*code*) es único para cada producto. Para ello la clase Stock tiene un atributo *lastCode* que representa el último código utilizado.



Tareas a realizar en el laboratorio

En todos los cambios que se realicen, debe quedar la documentación totalmente actualizada. Asegúrate que las clases **`Product`** y **`Stock`** están totalmente documentadas al finalizar el laboratorio.

1. Descarga el proyecto *SupermarketStudents.zip* de eGela e impórtalo en Eclipse. No te preocupes si te da errores ya que falta por implementar código que de coherencia a toda la aplicación.
2. Vamos a realizar algunos cambios en las clases, verás que en Eclipse te aparecerán como tareas a realizar porque están marcados con comentarios especiales en el código:*//TODO …*. Para verlos debes seleccionar en el menú superior *Window → Show View → Tasks.* Cada vez que termines una tarea, borra el término TODO para que desaparezca de la lista de tareas. A continuación, se describen esos cambios.
3. Implementa de manera automática para la clase **Product** las siguientes operaciones. Para ello utiliza los menús *Source → Generate constructor using Fields ...* y *Source → Generate getters and setters ...*
    - Constructoras: Si no se dice nada sobre el VAT su valor será 0.21 y el pasado como parámetro en caso contrario
        - Una que reciba solo el código
        - Una que reciba un valor por cada parámetro
    - Métodos *getter* y *setter* de todos los atributos.
4. Sobrescribe el método toString para las clases **Product** y **Stock**
    - El método `toString` de la clase **`Product`** devolverá un String con los valores de los atributos de los productos separados por espacios en este orden: code, name, Price, amount, weight y VAT.
    - El método `toString` de la clase **`Stock`** devolverá un String que tiene una línea para cada producto.
5. Dado que la clase **`Stock`** contiene un ArrayList de **`Product`**, y que se utilizan sus métodos, sobrescribe el método `equals` en la clase **`Product`** para que todo funcione adecuadamente. Dos **Product** son iguales si tienen el mismo código.

6. Excepciones no predefinidas en la clase `Product`

- Define la clase excepción **`NegativeAmountException`** como clase independiente en el paquete `packproduct.` Se definirán dos constructoras: una sin parámetros y la otra con un parámetro String para el mensaje:
- Modifica la implementación del método *`setAmount`* considerando ahora que se eleve la excepción **`NegativeAmountException`** si el número dado es negativo. Se debe usar la constructora con el parámetro String, pasando un mensaje apropiado.

7. Excepciones no predefinidas en la clase `Stock` y el método `main` de la clase `SuperOnline`:

- El método **`updateAmount`** no trata la excepción **`NegativeAmountException`** que puede lanzar la llamada al método *setAmount* y, por lo tanto, debe indicar que corre el riesgo de que se propague ese error.
- Define ahora dentro de la clase `Stock` la excepción **`UnknowCodeException.`** Reconsidera los métodos `removeProduct` y `updateAmount` en aquellos casos que reciben como parámetro un código de producto que no está en el inventario. Para indicar que un producto no está en el inventario, en lugar de imprimir mensajes (suprime esas sentencias) eleva la excepción.
- Corrige los errores de compilación del método *`main`* de la clase **`SuperOnline.`** Para ello haz un tratamiento adecuado de las excepciones. Ten en cuenta que el método *main* no debe elevar ninguna excepción, por lo que todas las excepciones en el *main* se deben capturar y mostrar un mensaje. Además, la captura de una excepción no debe impedir la ejecución de las siguientes sentencias del *`main`*.

8. Excepciones predefinidas en la clase `Stock` y el método `main` de la clase `SuperOnline`:

- Agrega en la clase **`Stock`** el método *`storeStockInFile`*: guarda el stock en un archivo, utilizando el método *`toString`* en la clase **`Stock`** (un producto por línea). Captura adecuadamente la excepción que se puede elevar al utilizar la clase FileWriter y trátala mediante un mensaje de error apropiado.
- Actualiza el *`main`* de la clase **`SuperOnline`**: Añade 3 productos nuevos, muestra el stock por pantalla, elimina un producto, volver a mostrar el stock y finalmente guardar el stock en un fichero. El fichero debe contener algo similar a lo siguiente:

```
1001 banana 2.3 50 1.0 0.04
1002 gel_ducha 1.58 100 1.2 0.21
1003 alitas_de_pollo 2.13 40 0.5 0.1
```

- Analiza dónde se genera el fichero y comprueba que el fichero se genere correctamente.

Tareas complementarias

9. Implementa en la clase **`Stock`** el método *obtainProduct*, que devolverá el producto cuyo código es dado por parámetro. Si no está el producto de código dado, se lanzará la excepción **`UnknowCodeException`**.

10. Crea una carpeta data en tu proyecto (a la misma altura que la carpeta src) e intenta que el método *`storeStockInFile`* genere el fichero en dicha carpeta.