# Laboratorio L10

## Excepciones

## Objetivos

- Tratamiento de excepciones en Java: capturar y elevar.
- Continuar practicando la documentación de programas utilizando JavaDoc.
- Procesamiento de ficheros de texto.
- Trabajar con el depurador.

## Entregas:

Fichero ZIP con la exportación del laboratorio L10. Este fichero debe contener el código JAVA y la documentación generadas.

Nomenclatura:

1. Nombre del archivo: ***apellidoNombre_L10.zip***
2. La entrega debe ser ***individual*** y realizarse a través de eGela
3. Se evaluará el laboratorio comprobando su funcionamiento. EL PROGRAMA DEBE FUNCIONAR. Se entregará un pequeño manual de ejecución si se considera necesario.

## Tareas a realizar

Todos los cambios realizados deben quedar reflejados en la documentación. Al finalizar el laboratorio, asegúrate de que la clase `Product`, sus clases hijas y la clase `Stock` están completamente documentadas.

Ten en cuenta que el método `main` no debe lanzar excepciones, por lo que todas las excepciones que lleguen o se produzcan en el `main` deben capturarse y tratarse adecuadamente. Además, la captura de una excepción nunca debe impedir la ejecución del resto de las sentencias de `main`.

1. Importa el proyecto del laboratorio L09 y cambia su nombre a *apellidoNombre_L10*.

2. Excepciones predefinidas:
   - Modifica o crea el método *set* para el atributo *amount* en la clase `Product` (`setAmount`): este método lanzará *InvalidAttributeValueException*[1] si la cantidad que se pretende asignar a dicho atributo es negativa. Para lanzar esta excepción se debe utilizar el constructor de la clase `Exception` que tiene un parámetro de tipo String con el mensaje de error adecuado a dicha excepción.
   - El método `updateAmount` de la clase `Stock` no gestionará la excepción *InvalidAttributeValueException* que podría producirse al llamarse al método `setAmount` utilizado para su implementación, por lo que debe propagar dicha excepción.
   - Añade una llamada al método `updateAmount` en el menú de la clase `SuperOnline` (si aún no lo has hecho).
   - En el método `readProductCode` de la clase `SuperOnline`, captura y trata la excepción *InputMismatchException* que se puede elevar en el método `nextInt`. En el tratamiento de esta excepción debes asegurarte de que se lea un número entero.

[1] Está definida en el paquete `javax.management`

- Cambia el método `addProduct` de la clase `Stock`: ahora si el producto a añadir ya existe (es decir, si ya hay otro producto con el mismo código) el método debe elevar *InstanceAlreadyExistsException*[2], en lugar de devolver el valor booleano que devolvía hasta ahora. Para elevar esta excepción se debe utilizar el constructor de la clase de `Excepción` que tiene un parámetro de tipo `String`, que contiene un mensaje adecuado a la excepción que se ha producido.
- Corrige los errores de compilación del método `main` implementado en la clase SuperOnline:
  - En el método `loadProducts`: captura la excepción y muestra un mensaje. Haz dos versiones:
    - Incluye todas las llamadas al método `addProduct` dentro del mismo try-catch. ¿Qué pasa?
    - Incluye cada llamada al método `addProduct` en un bloque `try-cath` diferente. ¿Qué pasa ahora? ¿Ha sido el mismo comportamiento en ambos casos?

3. Excepciones no predefinidas:
   - Define la excepción *UnknownCodeException* con dos constructores, uno sin parámetros y el otro con un parámetro de tipo `String`. Modifica los métodos `removeProduct` y `updateAmount` de la clase `Stock` para que lancen esta excepción si el el código de producto que reciben como parámetro no está inventariado.
   - Corrige los errores de compilación existentes en la clase `SuperOnline` mediante un tratamiento adecuado de las excepciones. En todos los casos se debe escribir el mensaje asociado a la excepción como tratamiento a la misma.

[2] Está definida en el paquete `javax.management`