PRÁCTICA 10 UT7. Creación de una interfaz y pruebas

Vamos a crear una interfaz gráfica en JavaFX para gestionar productos en un stock de una tienda.
La aplicación va a almacenar los datos en una lista en memoria (sin base de datos, usando un arraylist).
Vamos a implementar pruebas unitarias con JUnit y Mockito y a aplicar validaciones de seguridad para evitar datos erróneos o vacíos.

1. Crea un proyecto Maven, añade las dependencias de JavaFX, JUnit y Mockito.
2. Utiliza un diseño MVC similar a este:

```
StockManagementFX
 ├ src/main/java/com/example/stock
 |  ├ MainApp.java         # Clase principal
 |  ├ Product.java         # Modelo de datos
 |  ├ ProductService.java  # Lógica de negocio
 |  ├ ProductController.java # Controlador de la UI
 |  └ ProductView.fxml      # Interfaz en JavaFX
 └ src/test/java/com/example/stock
    ├ ProductServiceTest.java  # Pruebas con JUnit y Mockito
    └ SecurityTests.java       # Validaciones de seguridad
```

3. Product.java debe tener el constructor para al menos nombre, cantidad y precio de un producto y los getters y setters.
4. ProductService.java debe tener la lógica, puedes implementar un ArrayList, asegúrate de validar los campos que pueda introducir el usuario (por ejemplo nombre vacío, cantidad y precio negativos…).
5. ProductView debe ser la vista usando JavaFX. Si no usas JavaFX puedes hacer una interfaz con JFrame pero te recomiendo usar JavaFX y más aún, investigar cómo realizar un .fxml.
6. Añade al menos 5 pruebas con JUnit y Mockito.

7. Añade las validaciones necesarias de seguridad de los datos.
8. Si acabas rápido, puedes implementar el acceso a una base de datos MySQL como en Acceso a Datos (con Hibernate si prefieres) en vez de los datos en memoria.

Rúbrica de corrección

Criterios	2 Puntos	1 Punto	0 Puntos
1. Interfaz	La interfaz es usable, con un diseño organizado, uso adecuado de eventos y validaciones en tiempo real.	La interfaz permite la interacción básica, pero tiene una estructura simple o poco clara.	No hay una interfaz funcional o tiene errores graves que impiden su uso.
2. Almacenamiento en memoria	Implementa correctamente una lista de productos con métodos para agregar, editar y eliminar de forma eficiente.	Los productos se almacenan en memoria, pero la implementación es básica y puede presentar fallos en algunos casos.	No almacena productos correctamente o los datos se pierden inesperadamente.
3. JUnit y Mockito para pruebas unitarias	Incluye JUnit y Mockito, con al menos 5 pruebas unitarias bien estructuradas cubriendo los casos principales.	Contiene algunas pruebas unitarias con JUnit, pero no usa Mockito o faltan casos importantes.	No hay pruebas unitarias o las pruebas no funcionan correctamente.
4. Validaciones de seguridad en los datos	Se validan correctamente nombre, cantidad y precio, evitando valores inválidos y posibles ataques (ej. inyección de código).	Existen validaciones básicas, pero pueden permitir ciertos errores como valores negativos o vacíos.	No hay validaciones o la aplicación permite ingresar datos erróneos sin restricciones.

5. Manejo de errores y robustez	La aplicación maneja errores correctamente con mensajes claros y sin cierres inesperados.	Se manejan algunos errores, pero pueden existir excepciones no controladas que afecten la experiencia.	No hay manejo de errores o la aplicación se cierra abruptamente al ingresar datos incorrectos.