Dev Senior

dev Senior

RETO 3 JAVA

2025

2025

Sistema de Gestión de Biblioteca con Pruebas Unitarias en Java

www.devseniorcode.com

Sistema de Gestión de Biblioteca con Pruebas Unitarias en Java

Introducción

En el desarrollo de software, la implementación de pruebas unitarias es fundamental para garantizar la calidad y fiabilidad del código. En este documento, se abordará la creación de un sistema de gestión de biblioteca en Java, aplicando pruebas unitarias con JUnit 5 y Mockito. Este ejercicio permite comprender la importancia de la separación de responsabilidades en el diseño de software y la utilización de pruebas para validar su correcto funcionamiento.

Objetivos

Desarrollar una aplicación de escritorio que permita:

- Implementar un sistema de gestión de biblioteca en Java que permita la administración de libros, usuarios y préstamos.
- Desarrollar pruebas unitarias con JUnit 5 para validar la funcionalidad de la capa de servicio.
- Utilizar Mockito para simular el comportamiento de los repositorios y asegurar pruebas unitarias efectivas.
- Implementar el manejo de excepciones en Java para capturar y gestionar errores de manera eficiente.
- Fomentar el trabajo en equipo y el uso de herramientas de control de versiones como Git y GitHub.

Beneficios del Ejercicio en el Marco de Pruebas Unitarias en Java

- **Mejora de la calidad del código:** Permite detectar errores antes de la implementación en producción.
- **Facilita el mantenimiento**: Un código con pruebas bien diseñadas es más fácil de modificar y escalar.
- **Promueve el desarrollo basado en pruebas (TDD)**: Ayuda a diseñar software de manera más estructurada y modular.
- **Reduce el tiempo de depuración:** La detección temprana de errores evita la acumulación de fallos.
- **Fomento del trabajo colaborativo**: Permite que los estudiantes trabajen en equipo, organizando sus tareas y usando herramientas profesionales.

Ejemplo de Flujo del Programa

1. **Agregar un Libro**: El usuario introduce los datos de un nuevo libro en la biblioteca.
2. **Consultar un Libro**: Se obtiene información detallada de un libro a partir de su ID.
3. **Registrar un Usuario**: Se añade un nuevo usuario al sistema con su información.
4. **Realizar un Préstamo**: Un usuario solicita un libro, registrándose la fecha de préstamo.
5. **Consultar Préstamos por Usuario**: Se obtiene el historial de préstamos realizados por un usuario.

Requerimientos Funcionales

1. Gestión de Libros
- Agregar un libro con ID, título y autor.
- Obtener un libro mediante su ID.

2. Gestión de Usuarios
- Crear un usuario con ID y nombre.

3. Gestión de Préstamos
- Prestar un libro a un usuario registrando la fecha del préstamo.
- Obtener todos los préstamos de un usuario específico.

Requerimientos de Pruebas Unitarias

- **Pruebas de LibraryService**
  - Validar la adición de un libro.
  - Validar la recuperación de un libro por ID.
  - Validar la creación de un usuario.
  - Validar el préstamo de un libro a un usuario.
  - Validar la consulta de préstamos por usuario.

- **Uso de Mockito**
  - Simular la interacción con BookRepository y LoanRepository.

Manejo de Excepciones y Errores en Java

Para garantizar la estabilidad del sistema, se implementará el manejo de excepciones con try y catch. Algunos casos de uso incluyen:

- **Libro no encontrado:** Lanzar una NoSuchElementException si el libro solicitado no existe en la base de datos.
- **Usuario no encontrado:** Lanzar una IllegalArgumentException si se intenta prestar un libro a un usuario inexistente.
- **Libro ya prestado:** Capturar un posible error si un libro ya ha sido prestado y evitar que se preste nuevamente sin ser devuelto.
- **Errores generales de base de datos:** Uso de SQLException para manejar errores de conexión o consulta a la base de datos.

Metodología para el Reto 3: Sistema de Gestión de Biblioteca con Pruebas Unitarias en Java

1. Trabajo en Parejas

- **Formación de Parejas:** Los estudiantes se organizarán en parejas para realizar el reto. La colaboración será esencial para repartir responsabilidades y abordar problemas de manera conjunta.
- **División de Tareas:** Se recomienda que las parejas distribuyan las tareas de manera equitativa. Un miembro podría encargarse de la lógica del sistema (gestión de libros, usuarios y préstamos), mientras que el otro se enfoque en las pruebas unitarias y validación de errores.

2. Uso de Git y GitHub

- **Creación del Repositorio en GitHub**: Cada pareja creará un repositorio en GitHub para almacenar el código fuente. El repositorio debe ser público para revisión.
- **Uso de Ramas y Control de Versiones:**
- Se recomienda que cada estudiante trabaje en una rama separada.
- Una vez completadas las funcionalidades, deben fusionar las ramas.
- **Commits Regulares:** Los estudiantes deberán realizar commits frecuentes con mensajes claros.

3. Video de Presentación

- **Grabación del Video:** Una vez completado el proyecto, los estudiantes grabarán un video donde presentarán el sistema y los desafíos que enfrentaron.
- **Contenido del Video:**
- Introducción al proyecto, presentación de los desafíos y soluciones.
- Demostración de la ejecución del sistema.
- Participación equilibrada de ambos estudiantes en la presentación.

4. Revisión y Feedback

- **Entrega del Reto:** Los estudiantes compartirán el enlace al repositorio de GitHub y el video de presentación.
- **Feedback:** Los docentes revisarán el código y el video, proporcionando retroalimentación sobre la implementación, el uso de Git y la presentación del proyecto.

5. Evaluación con Rúbrica

Criterio	Descripción	Puntos
Implementación Técnica	Funcionalidad completa del sistema de gestión de biblioteca.	50
Uso de Git y GitHub	Uso adecuado de commits, ramas y documentación en el repositorio.	20
Presentación del video	Explicación clara, participación equitativa y demostración funcional del sistema.	20
Documentación	Código comentado y README con instrucciones claras.	10

Total Máximo: 100 puntos

- **Excelente (90-100 puntos):** Implementación completa y bien estructurada, con pruebas unitarias y un video claro.
- **Bueno (80-89 puntos):** Funcionalidad adecuada con detalles mejorables en pruebas o presentación.