Tecnológico de Monterrey

Sprint 1 (Java) - Adolfo

Unit Testing

Desarrollo e implantación de sistemas de software (Gpo 104)

Marcela Beatriz De La Rosa Barrios A01637239

Sebastián Denhi Vega Saint Martín A01637397

Ángela Estefanía Aguilar Medina A01637703

Axel Daniel Padilla Reyes A01642700

Diana Nicole Arana Sánchez A01642924

28/03/2025

Unit Testing

1. Revisión del código:

Library Class

La clase `Library` maneja una colección de libros y patronos. Proporciona métodos para agregar libros, eliminarlos, registrar patronos y manejar el préstamo de libros. Los métodos clave que modifican el estado del sistema son:

- `addBook(Book book)` - Añade un libro a la biblioteca.
- `removeBook(Book book)` - Elimina un libro de la biblioteca.
- `checkOutBook(Patron patron, Book book, int daysToDue)` - Realiza el préstamo de un libro a un patrono.
- `returnBook(Patron patron)` - Regresa un libro al inventario.
- `calculateFine(Patron patron)` - Calcula las multas por libros atrasados.
- `listAvailableBooks()` - Devuelve una lista de los libros disponibles.

Book Class

La clase `Book` representa los libros en la biblioteca. Los métodos clave son:

- `checkOut(int daysToDue)` - Marca un libro como prestado y establece la fecha de vencimiento.
- `returnBook()` - Marca un libro como regresado.
- `setDueDate(LocalDate dueDate)` - Establece una fecha de vencimiento para un libro prestado.

Patron Class

La clase `Patron` representa a los usuarios que pueden pedir libros prestados. Los métodos clave son:

- `checkOutBook(Book book)` - El patrono toma un libro prestado.
- `returnBook(Book book)` - El patrono devuelve un libro.

2. Identificación de métodos críticos a probar:

Algunos métodos críticos para probar en la clase `Library` son:

- `addBook(Book book)` - Probar si un libro se puede añadir correctamente, especialmente si ya existe en la lista.
- `checkOutBook(Patron patron, Book book, int daysToDue)` - Asegurar que un libro no se pueda prestar dos veces sin ser regresado.
- `returnBook(Patron patron)` - Asegurar que un libro se pueda devolver correctamente.
- `calculateFine(Patron patron)` - Verificar que las multas se calculen correctamente según los días de retraso.

3. Casos de prueba para cubrir

Feature / Class	Test Case ID	Unit Test Case Name	Test Scenario	Input	Expected outcome	Remarks
Library Class	TC-LIB-001	TestAddBook	Add Book - Valid Book	Add Book with title "1984" and author "George Orwell"	Book should be added to the available books list	Valid test case for adding a new book
Library Class	TC-LIB-002	TestAddDuplicateBook	Add Book - Duplicate Book	Add Book with title "1984" and author "George Orwell" twice	Duplicate books should not be added, only one instance should exist	Test for handling duplicates
Library Class	TC-LIB-003	TestCalculateFineAfterReturn	Calculate Fine After Book Return	Checkout book "Design Patterns" for 2 days, then return after 2 days overdue	No fine should be calculated after returning the book	Verifies fine calculation after return
Library Class	TC-LIB-004	TestNonexistentBookCheckout	Checkout Book - Nonexistent Book	Try to check out a book titled "Nonexistent Book" that isn't in the library	Checkout should fail and return false	Test for nonexistent book checkout
Library Class	TC-LIB-005	TestFineCalculation	Fine Calculation for Overdue Book	Checkout book "Design Patterns" for 2 days, then set it overdue by 5 days	Fine should be correctly calculated as $0.50 per day overdue	Test for correct fine calculation
Library Class	TC-LIB-006	TestListBooksAndPatrons	List Books and Patrons in Library	Add two books and one patron, then list available books and patrons	Should list 2 available books and 1 registered patron	Verifies the listing functionality

4. Resumir el capítulo 5 sobre Mocks y Fragilidad de las Pruebas:

Este capítulo habla sobre el uso de mocks en las pruebas unitarias, que son una herramienta que simulan interacciones entre el sistema que estamos probando (SUT) y sus dependencias, lo que nos deja revisar cómo se comunican. Pero si se usan mocks sin precaución, hacen que las pruebas sean frágiles, es decir, que fallen con cambios en la implementación, lo cual afecta la capacidad de refactorizar sin romper todo.

Diferencia entre Mocks y Stubs

- Mocks: Se usan para emular interacciones de salida, es decir, las llamadas que hace el SUT a sus dependencias para cambiar algo.

- Stubs: Sirven para emular interacciones de entrada, proporcionando datos al SUT, pero no afectan su estado.

Relación entre Mocks y la Fragilidad en las Pruebas

Los mocks pueden hacer que las pruebas sean frágiles si no se usan bien, esto se debe a que dependen demasiado de los detalles internos de implementación, lo cual las hace muy sensibles a cambios. La solución es centrar las pruebas en verificar lo que el sistema hace (comportamiento observable) y no en cómo lo hace (detalles de implementación).

Uso Correcto de Mocks y Stubs

Los mocks deben usarse para verificar interacciones que afectan al mundo exterior, como cambios visibles y los stubs no se deben verificar, ya que solo devuelven datos para el SUT, lo cual puede llevar a pruebas frágiles.

Diseño de una API Bien Definida

Una buena API asegura que el comportamiento observable (lo que el cliente ve) coincida con las operaciones públicas, esto se hace al encapsular los detalles internos del sistema, lo cual hace que el código sea más fácil de mantener y refactorizar, además de mejorar su estabilidad.

Arquitectura Hexagonal

El capítulo introduce la arquitectura hexagonal, que separa la lógica de negocio (dominio) de los servicios externos (como bases de datos o APIs). Los mocks se

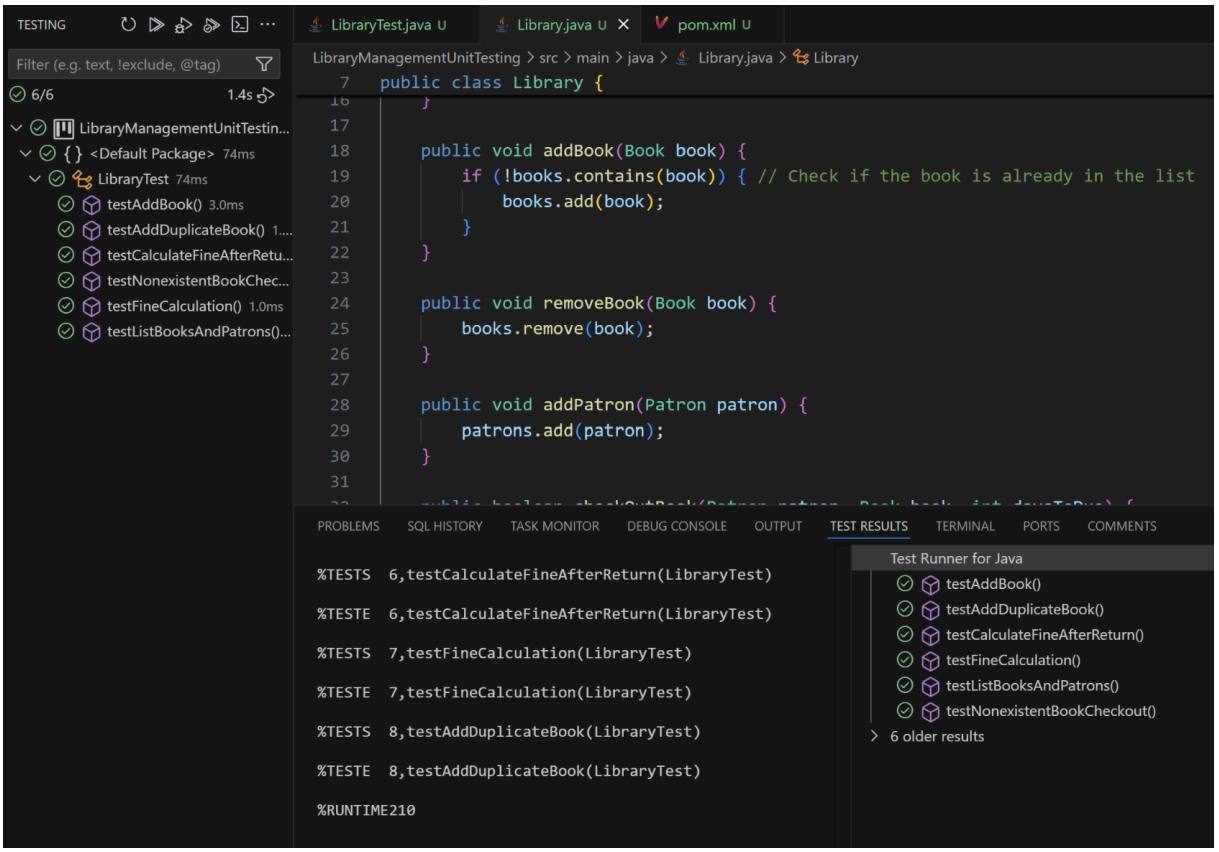
usan principalmente para verificar las interacciones con sistemas externos, no con las comunicaciones internas de la aplicación.

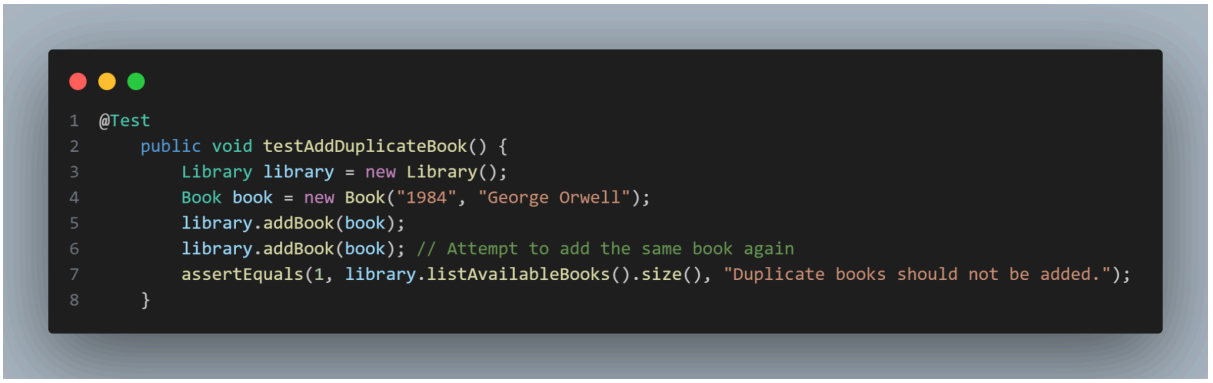
Comunicaciones Intra-sistema vs. Inter-sistema

- Intra-sistema: Son las comunicaciones dentro de la aplicación, entre clases. Estos detalles son internos y no deberían verificarse con mocks.

- Inter-sistema: Son las comunicaciones con sistemas externos. Estas sí forman parte del comportamiento observable y deben verificarse con mocks para asegurar que todo funcione bien con el exterior.

5. Screenshots:





```java
@Test
    public void testFineCalculation() {
        Library library = new Library();
        Patron patron = new Patron("Alice Smith");
        Book book = new Book("Design Patterns", "Erich Gamma");
        library.addBook(book);
        library.addPatron(patron);

        library.checkOutBook(patron, book, 2);
        book.setDueDate(LocalDate.now().minusDays(5)); // Simulate overdue by 5 days

        double fine = library.calculateFine(patron);
        assertEquals(2.5, fine, 0.01, "Fine should be correctly calculated as $0.50 per day overdue.");
    }
```

```java
@Test
    public void testListBooksAndPatrons() {
        Library library = new Library();
        Book book1 = new Book("1984", "George Orwell");
        Book book2 = new Book("Brave New World", "Aldous Huxley");
        Patron patron = new Patron("Alice Smith");

        library.addBook(book1);
        library.addBook(book2);
        library.addPatron(patron);

        assertEquals(2, library.listAvailableBooks().size(), "Should list all available books.");
        assertEquals(1, library.listPatrons().size(), "Should list all registered patrons.");
    }
```

```java
@Test
    public void testNonexistentBookCheckout() {
        Library library = new Library();
        Patron patron = new Patron("Alice Smith");
        Book book = new Book("Nonexistent Book", "Unknown Author");
        library.addPatron(patron);
        boolean result = library.checkOutBook(patron, book, 7); // Attempt to check out a book not in the library
        assertFalse(result, "Should not be able to check out a nonexistent book.");
    }
```