Test Cases Documentation

1. Test Case: **testBookConstructor**

Objective

To verify that the parameterized constructor of the **Book** class initializes all attributes correctly.

Test Steps

1. Create a `Book` object using the parameterized constructor.

2. Pass values for **bookID**, **title**, **author**, **isbn**, **genre**, and **availability**.

3. Use assertions to verify that the getters return the same values that were passed during construction.

Expected Output

- bookID: 1

- titl`: "Title"

- author: "Author"

- isbn: "ISBN123"

- genre: "Genre"

- availability "Available"

Sample Assertion

assertEquals(1, book.getBookID());

assertEquals("Title", book.getTitle());

assertEquals("Author", book.getAuthor());

assertEquals("ISBN123", book.getIsbn());

assertEquals("Genre", book.getGenre());

assertEquals("Available", book.getAvailability())

2. Test Case: testDefaultConstructorAndSetters

Objective

To verify that the no-argument constructor initializes a **Book** object with default values and that the setters correctly update the attributes.

Test Steps

1. Create a **Book** object using the no-argument constructor.

2. Update all attributes using the setters.

3. Use assertions to verify that the getters return the updated values.

Expected Output

- bookID: 2

- title: "Another Title"

- author: "Another Author"

- isbn: "ISBN456"

- genre: "Another Genre"

- availability: "Unavailable"

Sample Assertion

assertEquals(2, book.getBookID());

assertEquals("Another Title", book.getTitle());

assertEquals("Another Author", book.getAuthor());

assertEquals("ISBN456", book.getIsbn());

assertEquals("Another Genre", book.getGenre());

assertEquals("Unavailable", book.getAvailability());

3. Test Case: testNullValues

Objective

To verify that the **Book** class handles **null** values correctly when they are set for certain attributes.

Test Step

1. Create a **Book** object using the no-argument constructor.

2. Set **title** and **author** attributes to **null** using the setters.

3. Use assertions to verify that the getters return **null** for these attributes.

Expected Output

- title: null

- author: null

Sample Assertion

assertNull(book.getTitle());

assertNull(book.getAuthor());

4. Test Case: testEdgeCases

Objective

To verify that the **Book** class can handle edge cases, such as extremely long input values for attributes.

Test Steps

1. Create a string with 1000 repeated characters (e.g., "A").

2. Create a `Book` object using the no-argument constructor.

3. Set the `title` attribute to the long string using the setter.

4. Use assertions to verify that the getter returns the exact long string.

Expected Output

- title: A string of 1000 "A" characters.

Sample Assertion

assertEquals(longTitle, book.getTitle());

General Notes

These test cases cover the following:

- Constructor behavior (default and parameterized).

- Getter and setter functionality.

- Handling of null values.

- Edge case scenarios.

Test Cases Documentation for Transaction Class

1. Test Case: testParameterizedConstructor

Objective

To verify that the parameterized constructor of the `Transaction` class initializes all attributes correctly.

Test Steps

1. Create a **Transaction** object using the parameterized constructor.

2. Pass values for **transactionId**, **bookId**, **patronId**, **issueDate**, **dueDate**, and **returnDate**

3. Use assertions to verify that the getters return the same values that were passed during construction.

Expected Output

- transactionId: 101

- bookId: 202

- patronId: 303

- issueDate: 2024-12-01

- dueDate: 2024-12-15

- returnDate: 2024-12-10

Sample Assertion

assertEquals(transactionId, transaction.getTransactionid());

assertEquals(bookId, transaction.getBookid());

assertEquals(patronId, transaction.getPatronid());

assertEquals(issueDate, transaction.getIssuedate());

assertEquals(dueDate, transaction.getDuedate());

assertEquals(returnDate, transaction.getReturndate());

2. Test Case: testSettersAndGetters

Objective

To verify that the setters update the attributes of the **Transaction** class correctly and that the getters return the updated values.

Test Steps

1. Create a **Transaction** object with default values.

2. Update all attributes using the setters.

3. Use assertions to verify that the getters return the updated values.

Expected Output

- transactionId: 111

- bookId: 222

- patronId: 333

- issueDate: 2024-11-01

- dueDate: 2024-11-15

- returnDate: 2024-11-10

Sample Assertion

assertEquals(111, transaction.getTransactionid());

assertEquals(222, transaction.getBookid());

assertEquals(333, transaction.getPatronid());

assertEquals(issueDate, transaction.getIssuedate());

assertEquals(dueDate, transaction.getDuedate());

assertEquals(returnDate, transaction.getReturndate());

3. Test Case: testNullValues

Objective

To verify that the Transaction class handles null values correctly when they are set for date attributes.

Test Steps

1. Create a **Transaction** object with **null** values for **issueDate**, **dueDate**, and **returnDate**.

2. Use assertions to verify that the getters return null for these attributes.

Expected Output

- **issueDate**: null

- **dueDate**: null

- returnDate: null

Sample Assertion

assertNull(transaction.getIssuedate());

assertNull(transaction.getDuedate());

assertNull(transaction.getReturndate());

4. Test Case: testEdgeCases

Objective

To verify that the **Transaction** class can handle edge cases, such as extreme date values.

Test Steps

1. Create a **Transaction** object with edge case date values (e.g., far future dates).

2. Use assertions to verify that the getters return the exact extreme date values.

Expected Output

- issueDate: 3000-01-01

- dueDate: 3000-01-15

- returnDate: 3000-01-10

Sample Assertion

assertEquals(issueDate, transaction.getIssuedate());

assertEquals(dueDate, transaction.getDuedate());

assertEquals(returnDate, transaction.getReturndate());

General Notes

1. These test cases cover the following:

- Constructor behavior (default and parameterized).

- Getter and setter functionality.

- Handling of null values.

- Edge case scenarios with extreme date values.

2. The **Transaction** class should ensure data integrity and properly handle invalid or extreme inputs without crashing.

Future Enhancements

1. Add tests for invalid inputs, such as:

- Negative or zero transactionId, bookId, or patronId.

- Invalid date ranges (e.g., returnDate before issueDate).

2. Consider integration tests for database interactions where **Transaction** objects are saved or retrieved.

DbUtil Test Case Documentation

Overview

This document provides detailed documentation for the test cases of the DbUtil class in the Library Management System project. The test cases focus on verifying the database connection functionality.

Test Cases

1. Database Connection Successful Scenario

- Test Method: testConnectDBSuccess()

- Purpose: Verify successful database connection

- Test Steps:

1. Mock the **DriverManager.getConnection()** method to return a mock connection

2. Mock **JOptionPane.showMessageDialog()** to simulate success message

3. Call **DbUtil.connectDB()** method

- Expected Outcomes:

- Connection object should not be null

- getConnection() method called exactly once with correct parameters

- Success message dialog shown once

- Verification Techniques:

- assertNotNull() for connection object

- Mockito verification of method invocations

2. Database Connection Failure Scenario

- Test Method: testConnectDBFailure()

- Purpose: Verify handling of database connection failure

- Test Steps:

1. Mock the DriverManager.getConnection() method to throw a RuntimeException

2. Mock JOptionPane.showMessageDialog() to simulate error message

3. Call DbUtil.connectDB() method

- Expected Outcomes:

- Connection object should be null

- getConnection() method called exactly once with correct parameters

- Error message dialog shown once with specific error message

- Verification Techniques:

- assertNull() for connection object

- Mockito verification of method invocations and error message

Test Setup

- setUp() Method:

- Creates static mocks for DriverManager and JOptionPane

- Ensures a clean test environment before each test method

Mocking Strategies

- Used MockedStatic to mock static method calls

- Simulated both success and failure scenarios

- Verified method invocations and parameters