

ONLINE LIBRARY MANAGEMENT

IIHT

Time To Complete: 3 hrs

CONTENTS

1 Problem Statement	3
2 Business Requirements:	3
3 Implementation/Functional Requirements	3
3.1 Code Quality/Optimizations	3
3.2 Template Code Structure	4
a. Package: com.onlinelibrarymanagement	4
b. Package: com.onlinelibrarymanagement.model	4
c. Package: com.onlinelibrarymanagement.repository	4
4 Execution Steps to Follow	5

1 PROBLEM STATEMENT

The Online Library Management System empowers users to perform CRUD (Create, Read, Update, Delete) operations and search functionalities in different criterias on books and subjects. Users can create new book entries and subject categories, update existing book and subject information, delete books and subjects that are no longer relevant or available, and retrieve book and subject details for viewing.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	<ol style="list-style-type: none">1. User needs to enter into the application.2. The user should be able to do the particular operations3. The console should display the menu<ol style="list-style-type: none">1) create subject2) update subject3) delete subject4) view all subjects5) create book6) update book7) delete book8) view all books9) search subject by name10) list all issued books11) search book by name12) remove all subjects13) remove all books14) exit

Business Requirements

System Features

The application should provide the following functionalities to the user:

- **Create Subject:** Add a new subject to the system.
- **Update Subject:** Modify the details of an existing subject.

- **Delete Subject:** Remove a subject from the system.
- **View All Subjects:** Display all subjects available.
- **Create Book:** Add a new book to the system.
- **Update Book:** Modify the details of an existing book.
- **Delete Book:** Remove a book from the system.
- **View All Books:** Display all books available.
- **Search Subject by Name:** Search for subjects by their name.
- **List All Issued Books:** List all the books that are currently issued.
- **Search Book by Name:** Search for books by their name.
- **Remove All Subjects:** Delete all subjects from the system.
- **Remove All Books:** Delete all books from the system.

Classes and Method Descriptions

OnlineLibraryApplication Class

displayMenu()

- **Task:** Displays the menu with available operations.
 - **Functionality:** Displays the list of operations such as adding, updating, deleting subjects and books, and searching for them as shown in business requirement above.
 - **Return Value:** void
 - **Explanation:** This method provides the user with options to interact with the system.

searchSubjectByName()

- **Task:** Retrieves all subjects by matching the given name.
 - **Functionality:** Prompts user for entering subject name and then search by calling `subjectDAO.searchByName(name)`.
 - **Return Value:** void
 - **Explanation:** If no subject is found then print `"No subjects found with the given name."` otherwise print each subject on a separate line as `"ID: " + subject.getId() + ", Name: " + subject.getName()`.

addSubject()

- **Task**: Adds a new subject.
 - **Functionality**: Prompts the user to enter the subject's name and calls `subjectDAO.create()` to insert it into the database.
- **Return Value**: void
- **Explanation**: Displays a success message if the subject is added successfully.
- **Message to Display**: "Subject added successfully!"

updateSubject()

- **Task**: Updates an existing subject.
 - **Functionality**: Prompts the user for the subject ID and the new name, and calls `subjectDAO.update()` to save the changes.
- **Return Value**: void
- **Explanation**: If the subject is updated successfully, a success message is displayed otherwise print `"Subject not found with ID: " + id``.
- **Message to Display**: "Subject updated successfully!"

deleteSubject()

- **Task**: Deletes an existing subject.
 - **Functionality**: Prompts the user for the subject ID and calls `subjectDAO.delete()` to remove it from the database.
- **Return Value**: void
- **Explanation**: After deletion, a success message is displayed.
- **Message to Display**: "Subject deleted successfully!"

viewAllSubjects()

- **Task**: Views all subjects.
 - **Functionality**: Retrieves all subjects using `subjectDAO.getAll()` and displays them to the user.
- **Return Value**: void

- **Explanation**: Displays the details of all available subjects.
- **Message to Display**: "----- All Subjects -----" followed by subject details as ``subject.getId() + ": " + subject.getName()``. In case no subjects are found print ``No subjects found.`` •

addBook()

- **Task**: Adds a new book.
- **Functionality**: Prompts the user for the book's name, subject ID, ISBN, and issue status, then calls ``bookDAO.create()`` to insert the book into the database.
- **Return Value**: void
- **Explanation**: Displays a success message if the book is added successfully.
- **Message to Display**: "Book added successfully!".

updateBook()

- **Task**: Updates an existing book.
- **Functionality**: Prompts the user for the book ID and the new details, then calls ``bookDAO.update()`` to update the book in the database.
- **Return Value**: void
- **Explanation**: If the book is found then prompts for name, subject id, isbn and is issued or not and update it and print a success message as ``"Book updated successfully!"`` otherwise print ``"Book not found with ID: " + id``.
- **Message to Display**: "Book updated successfully!".

deleteBook()

- **Task**: Deletes a book.
- **Functionality**: Prompts the user for the book ID and calls ``bookDAO.delete()`` to remove it from the database.
- **Return Value**: void
- **Explanation**: After deletion, a success message is displayed.
- **Message to Display**: "Book deleted successfully!".

viewAllBooks()

- **Task**: Views all books.
 - **Functionality**: Retrieves all books using `bookDAO.getAll()` and displays them to the user.
 - **Return Value**: void
 - **Explanation**: Displays the details of all available books.
 - **Message to Display**: "----- All Books -----" followed by book details as `book.getId() + ": " + book.getName()`.

listAllIssuedBooks()

- **Task**: List down all issued books.
 - **Functionality**: Calls `bookDAO.getAllIssued()` and prints all issued books.
 - **Return Value**: void
 - **Explanation**: If found then displays the list of books as each book should be printed on separate line as `"ID: " + book.getId() + ", Name: " + book.getName() + ", ISBN: " + book.getISBN()`, otherwise print `"No issued books found."`.
 - **Message to Display**: "Issued Books:".

searchBookByName()

- **Task**: Searches for a book by its name.
 - **Functionality**: Prompts the user for a book name and calls `bookDAO.searchByName()` to retrieve matching books from the database.
 - **Return Value**: void
 - **Explanation**: If found then displays the list of books matching the name on separate line as `"ID: " + book.getId() + ", Name: " + book.getName() + ", ISBN: " + book.getISBN()`, otherwise print `"No books found with the given name."`.
 - **Message to Display**: "Books found:".

removeAllSubjects()

- **Task**: Removes all subjects.
 - **Functionality**: Calls `subjectDAO.deleteAll()` to delete all subjects from the database.

- ****Return Value****: void
- ****Explanation****: After deletion, a success message is displayed.
- ****Message to Display****: "All subjects have been removed."

removeAllBooks()

- ****Task****: Removes all books.
- ****Functionality****: Calls `bookDAO.deleteAll()` to delete all books from the database.
- ****Return Value****: void
- ****Explanation****: After deletion, a success message is displayed.
- ****Message to Display****: "All books have been removed."

DAOImpl Classes and Method Descriptions

BookDAOImpl Class

create(Book book)

- ****Task****: Inserts a new book into the database.
- ****Functionality****: Executes an SQL `INSERT` query to add the book to the database.
- ****Return Value****: void
- ****Explanation****: This method inserts the provided `Book` object into the database and returns the book object with the generated ID.

getById(int id)

- ****Task****: Retrieves a book by its ID.
- ****Functionality****: Executes a query to retrieve the book details by ID from the database.
- ****Return Value****: `Book` object representing the book.
- ****Explanation****: This method retrieves the book details from the database using the provided ID.

getAll()

- **Task**: Retrieves all books from the database.
- **Functionality**: Executes a `SELECT` query to retrieve all books and their details.
- **Return Value**: List of `Book` objects.
- **Explanation**: This method retrieves all books available in the database.

update(Book book)

- **Task**: Updates an existing book in the database.
- **Functionality**: Executes an SQL `UPDATE` query to modify the book details.
- **Return Value**: void
- **Explanation**: This method updates the book information in the database.

delete(int id)

- **Task**: Deletes a book from the database.
- **Functionality**: Executes an SQL `DELETE` query to remove the book by its ID.
- **Return Value**: void
- **Explanation**: This method deletes the book from the database using the provided ID.

getAllIssued()

- **Task**: Retrieves all issued books from the database.
- **Functionality**: Executes an SQL query to retrieve all issued books.
- **Return Value**: List of `Book` objects.
- **Explanation**: This method will return all issued books.

searchByName(String name)

- **Task**: Retrieves all books with having name as given from the database.
- **Functionality**: Executes an SQL query to search all books by given matching name.
- **Return Value**: List of `Book` objects.
- **Explanation**: This method will search and return all books matching the given name.

deleteAll()

- **Task**: Deletes all books from the database.
- **Functionality**: Executes an SQL `DELETE` query to remove all books.
- **Return Value**: void
- **Explanation**: This method deletes all books from the database.

SubjectDAOImpl Class

create(Subject subject)

- **Task**: Inserts a new subject into the database.
- **Functionality**: Executes an SQL `INSERT` query to add the subject to the database.
- **Return Value**: void
- **Explanation**: This method inserts the provided `Subject` object into the database and returns the subject object with the generated ID.

getById(int id)

- **Task**: Retrieves a subject by its ID.
- **Functionality**: Executes a query to retrieve the subject details by ID from the database.
- **Return Value**: `Subject` object representing the subject.
- **Explanation**: This method retrieves the subject details from the database using the provided ID.

getAll()

- **Task**: Retrieves all subjects from the database.
- **Functionality**: Executes a `SELECT` query to retrieve all subjects and their details.
- **Return Value**: List of `Subject` objects.
- **Explanation**: This method retrieves all subjects available in the database.

update(Subject subject)

- **Task**: Updates an existing subject in the database.
- **Functionality**: Executes an SQL `UPDATE` query to modify the subject details.
- **Return Value**: void
- **Explanation**: This method updates the subject information in the database.

delete(int id)

- **Task**: Deletes a subject from the database.
- **Functionality**: Executes an SQL `DELETE` query to remove the subject by its ID.
- **Return Value**: void
- **Explanation**: This method deletes the subject from the database using the provided ID.

searchByName(String name)

- **Task**: Retrieves all subjects from the database matching given name.
- **Functionality**: Executes an SQL query to find all subjects matching the given name.
- **Return Value**: List of `Subject` objects.
- **Explanation**: This method retrieves all subjects from the database matching the given name.

deleteAll()

- **Task**: Deletes all subjects from the database.
- **Functionality**: Executes an SQL `DELETE` query to remove all subjects.
- **Return Value**: void
- **Explanation**: This method deletes all subjects from the database

Model Classes with Annotations Guidance

Book Class

The **Book** class represents the book entity in the library system. Below are the field annotations and their descriptions. Please annotate this class appropriately so that it is treated as an entity and a table should be created with the name “books”.

1. **id field (Primary Key)**: The **id** field is expected to be the **primary key** for the **Book** entity. It should be auto-generated by the database using an identity generation strategy.
2. **name field**: This field represents the name of the book and should have **not null** validation with a length of **10 characters**.
3. **subjectId field**: This field should link to column “subject_id” and should be **not null**. It represents a foreign key linking the book to a subject.
4. **ISBN field**: The ISBN number of the book should be **not null** and is used for unique identification of books.
5. **isIssued field**: This boolean field tracks whether the book is issued or available in the library. It is **not null**. And it must have the column name as “is_issued”.

Subject Class

The **Subject** class represents the subject entity in the library system. Below are the field annotations and their descriptions. Please annotate this class appropriately so that it is treated as an entity and a table should be created with the name “subjects”.

1. **id field (Primary Key)**: The **id** field is expected to be the **primary key** for the **Subject** entity. It should be auto-generated by the database using an identity generation strategy.
2. **name field**: This field represents the name of the subject and should have **not null** validation with a length of **10 characters**.

3 IMPLEMENTATION/FUNCTIONAL REQUIREMENTS

3.1 CODE QUALITY/OPTIMIZATIONS

1. Associates should have written clean code that is readable.
2. Associates need to follow SOLID programming principles.

3.2 TEMPLATE CODE STRUCTURE

A. PACKAGE: COM.ONLINELIBRARYMANAGEMENT

Resources

Class/Interface	Description	Status
OnlineLibraryApplication.java(class)	This represents bootstrap class i.e class with Main method, that shall contain all console interaction with the user.	Partially implemented

B. PACKAGE: COM.ONLINELIBRARYMANAGEMENT.MODEL

Resources

Class/Interface	Description	Status
Book.java(class)	This represents entity class for Book	Partially Implemented
Subject.java(class)	This represents entity class for Subject	Partially Implemented

C. PACKAGE: COM.ONLINELIBRARYMANAGEMENT.REPOSITORY

Resources

Class/Interface	Description	Status
BookDao.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
BookDaoImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to be written by test taker	Partially Implemented
SubjectDao.java(interface)	This is an interface containing declaration of DAO method	Already Implemented

SubjectDaoImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to be written by test taker	Partially Implemented

4 EXECUTION STEPS TO FOLLOW

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. To build your project use command:
mvn clean package -Dmaven.test.skip
4. This editor Auto Saves the code.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. Default credentials for MySQL:
 - a. Username: **root**
 - b. Password: **pass@word1**
7. To login to mysql instance: Open new terminal and use following command:
 - a. **sudo systemctl enable mysql**
 - b. **sudo systemctl start mysql**

NOTE: After typing the second sql command (sudo systemctl start mysql), you may encounter a warning message like :

System has not been booted with systemd as init system (PID 1).
Can't operate. Failed to connect to bus: Host is down

>> **Please note that this warning is expected and can be disregarded. Proceed to the next step.**

c. `mysql -u root -p`

The last command will ask for password which is '**pass@word1**'

8. These are time bound assessments. The timer would stop if you logout (Save & Exit) and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

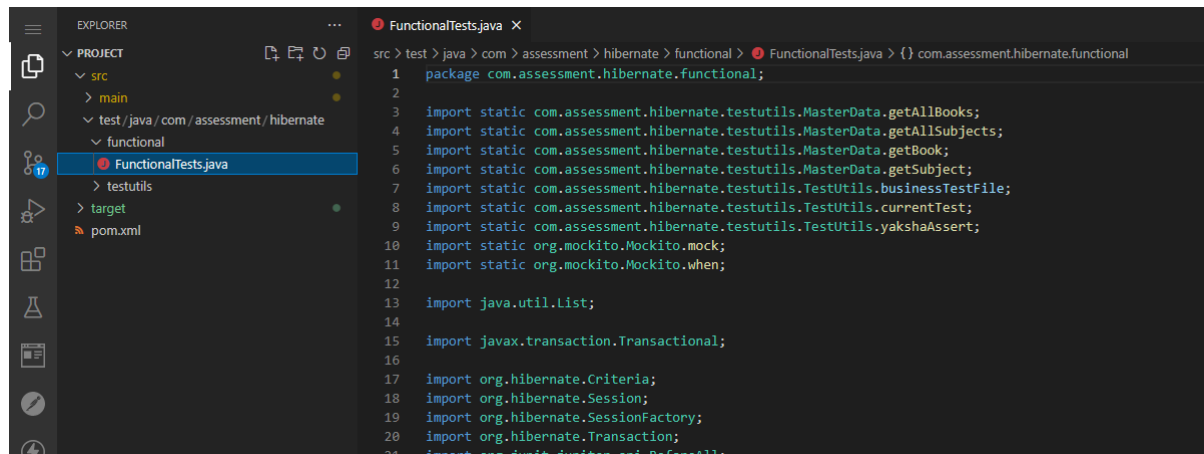
9. To run your project use command:

`mvn clean install exec:java`

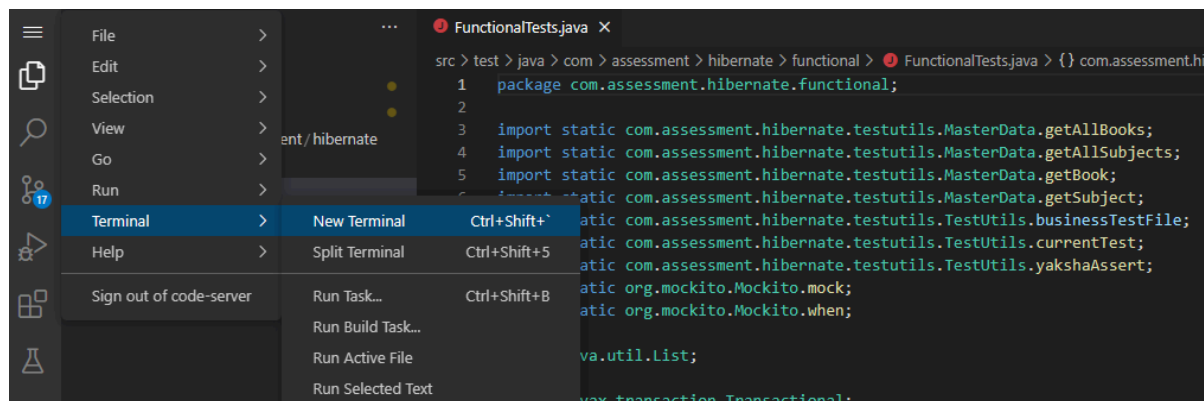
`-Dexec.mainClass="com.onlinelibrarymanagement.OnlineLibraryApplication"`

10. To test your project, use the command

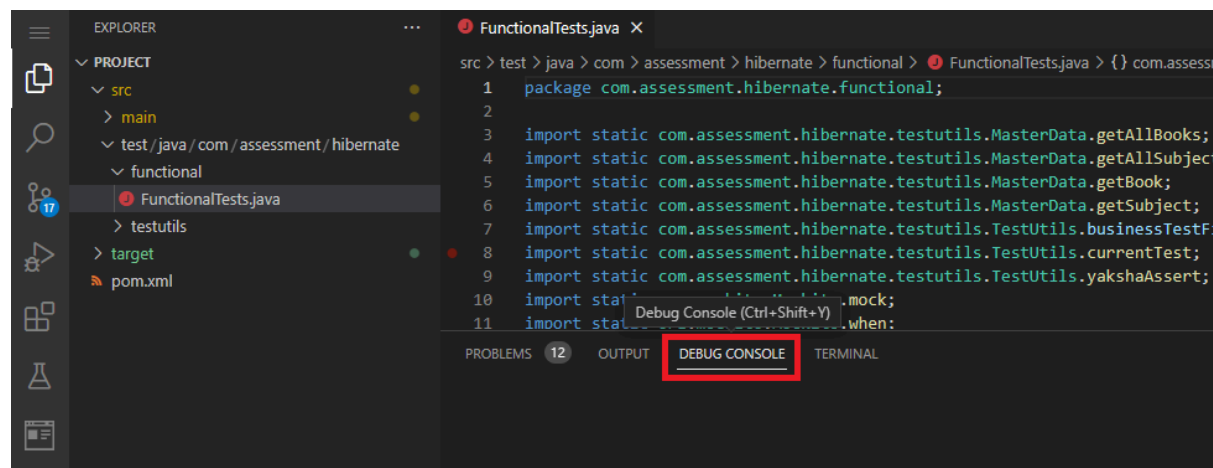
a. Open FunctionalTests.java file in editor



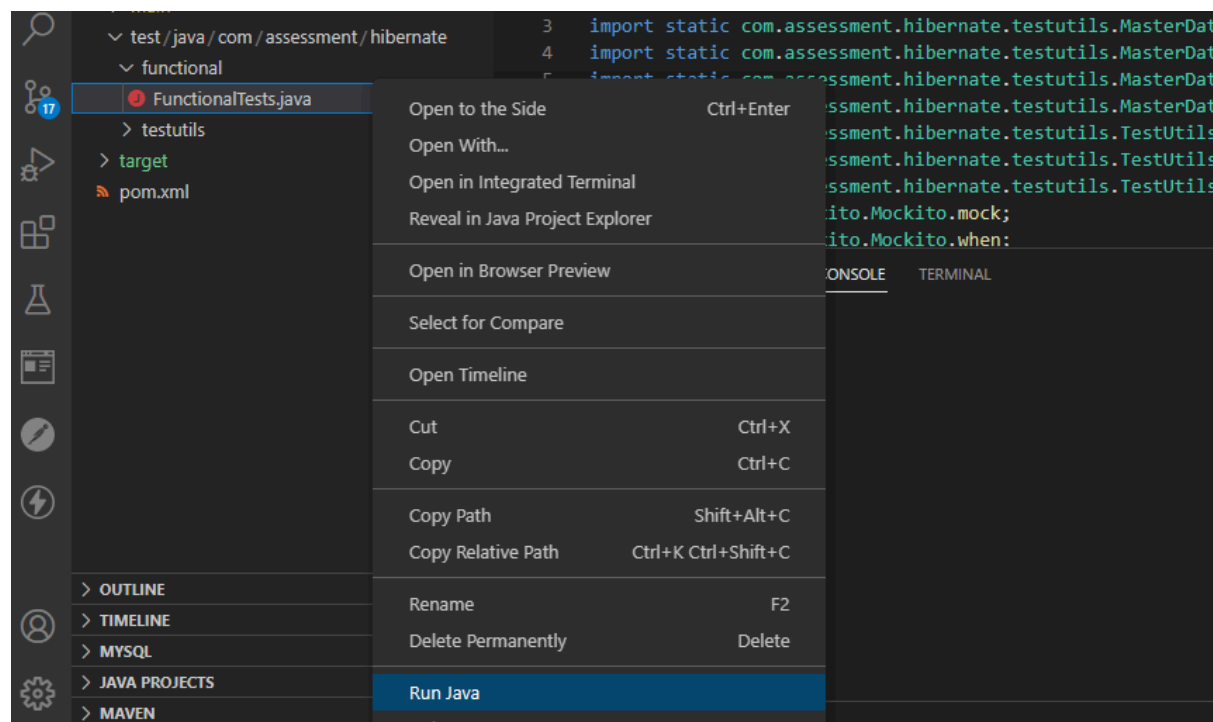
b. Open a new Terminal



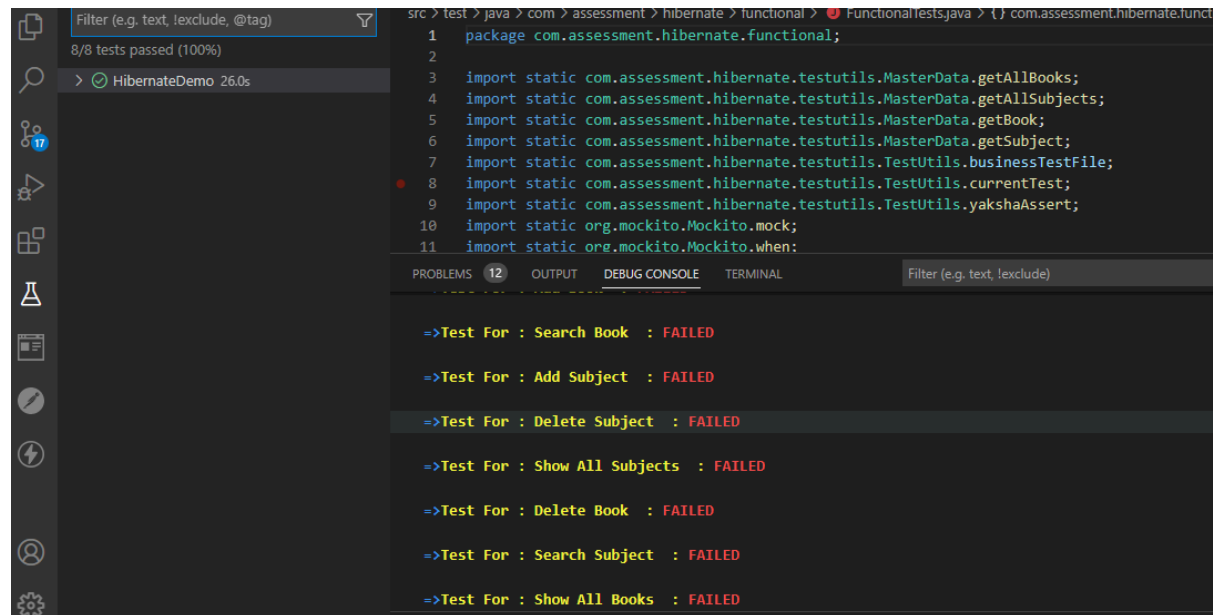
c. Go to Debug Console Tab



d. Right click on `FunctionalTests.java` file and select option Run Java



- e. This will launch the test cases and status of the same can be viewed in
Debug Console



The screenshot shows an IDE interface with a sidebar on the left and a main editor area on the right. The sidebar contains a filter bar with the text "Filter (e.g. text, !exclude, @tag)" and a search icon. Below the filter bar, it says "8/8 tests passed (100%)". The main editor area displays a Java file named "FunctionalTests.java" with the following code:

```
src > test > java > com > assessment > hibernate > functional > FunctionalTests.java > {} com.assessment.hibernate.fun
1 package com.assessment.hibernate.functional;
2
3 import static com.assessment.hibernate.testutils.MasterData.getAllBooks;
4 import static com.assessment.hibernate.testutils.MasterData.getAllSubjects;
5 import static com.assessment.hibernate.testutils.MasterData.getBook;
6 import static com.assessment.hibernate.testutils.MasterData.getSubject;
7 import static com.assessment.hibernate.testutils.TestUtils.businessTestFile;
8 import static com.assessment.hibernate.testutils.TestUtils.currentTest;
9 import static com.assessment.hibernate.testutils.TestUtils.yakshaAssert;
10 import static org.mockito.Mockito.mock;
11 import static org.mockito.Mockito.when;
```

Below the code, there is a tab labeled "DEBUG CONSOLE" with a filter bar. The debug console shows the following output:

```
=>Test For : Search Book : FAILED
=>Test For : Add Subject : FAILED
=>Test For : Delete Subject : FAILED
=>Test For : Show All Subjects : FAILED
=>Test For : Delete Book : FAILED
=>Test For : Search Subject : FAILED
=>Test For : Show All Books : FAILED
```