Online Faculty Management

IIHT

Time To Complete: 3 hrs

Contents

1 Problem Statement

The Online Faculty Management System enables users to perform CRUD (Create, Read, Update, Delete) operations and search functionalities in different criterias on teachers and departments. Users can create new teacher and department profiles, update existing teacher and department information, delete teachers and departments that are no longer active or needed, and retrieve teacher and department details for viewing.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. User needs to enter into the application. 2. The user should be able to do the particular operations 3. The console should display the menu 1) create department 2) create teacher 3) get department by id 4) get teacher by id 5) get all departments 6) get all teachers 7) update department 8) update teacher 9) delete department 10) delete teacher 11) search teachers by name 12) search teachers by subject 13) search department by name 14) exit

Business Requirements

System Features

The application should provide the following functionalities to the user:

- **Create Department**: Add a new department to the system.
- **Create Teacher**: Add a new teacher to the system, associated with a specific department.

- **Get Department by ID**: Retrieve details of a department by its unique identifier.
- **Get Teacher by ID**: Retrieves details of a teacher by its unique identifier.

- **Get All Departments**: List all departments in the system.
- **Get All Teachers**: List all teachers in the system.

- **Update Department**: Modify the details of an existing department.
- **Update Teacher**: Modify the details of an existing teacher.
- **Delete Department**: Remove a department from the system.
- **Delete Teacher**: Remove a teacher from the system.
- **Search Teachers by Name**: Search for teachers by their name.
- **Search Teachers by Subject**: Search for teachers by their subject.
- **Search Departments by Name**: Search for departments by their name.

Classes and Method Descriptions

OnlineFacultyApplication Class

main(String[] args)

- **Task**: Entry point for the application, handling the execution flow.

- Manages user input and routes it to appropriate methods for CRUD operations.

getUserChoice()

- **Task**: Retrieves the user's choice from the menu.

- **Functionality**: Prompts the user to enter their choice, which corresponds to one of the available operations.
- **Return Value**: int (The user's choice from the menu)
- **Explanation**: The method reads the user’s input from the console and returns the selected option.

printMenu()

- **Task**: Prints the menu for the user to choose from.

- **Functionality**: Displays a list of options, including creating, updating, deleting departments and teachers, and searching for teachers and departments as shown in business requirements above.
- **Return Value**: void
- **Explanation**: The method displays the available options to the user.

createDepartment()

- **Task**: Creates a new department.

- **Functionality**: Prompts the user for the department's name, creates a `Department` object, and calls `departmentDAO.createDepartment()` to save it to the database.
- **Return Value**: void
- **Explanation**: If the department is successfully created, a success message with the department ID is displayed as `"Department created successfully with ID: " + department.getId()` or in case of any error print message as `"Failed to create department."`.

createTeacher()

- **Task**: Creates a new teacher.

- **Functionality**: Prompts the user for the teacher’s name, subject, designation, and department ID, then creates a `Teacher` object and calls `teacherDAO.createTeacher()` to save it.
- **Return Value**: void
- **Explanation**: Displays a success message with the teacher ID if the teacher is successfully created as `"Teacher created successfully with ID: " + teacher.getId()` and incase of any error print message as `"Failed to create teacher."`.

getDepartmentById()

- **Task**: Retrieves department details by ID.

- **Functionality**: Prompts the user for the department ID, retrieves it using `departmentDAO.getDepartmentById()`, and displays the department's details along with its teachers.
- **Return Value**: void
- **Explanation**: If the department exists, its details like id, name and teachers are displayed separately on each line , including the teachers associated with it, otherwise print `"Department with ID " + id + " does not exist."`.

getTeacherById()

- **Task**: Retrieves teacher details by ID.

- **Functionality**: Prompts the user for the teacher ID, retrieves it using `teacherDAO.getTeacherById()`, and displays the teacher's details.
- **Return Value**: void
- **Explanation**: Displays the teacher’s information like id, name, subject and designation separately on each line, otherwise print an error message as `"Teacher with ID " + id + " does not exist."`.

getAllDepartments()

- **Task**: Retrieves and displays all departments.

    - **Functionality**: Calls `departmentDAO.getAllDepartments()` to fetch all departments and their associated teachers from the database.
    - **Return Value**: void
    - **Explanation**: Prints out all departments along with the list of teachers within each department as `"ID: " + department.getId() + ", Name: " + department.getName() + ", Teachers: " + department.getTeachers()`.

getAllTeachers()

- **Task**: Retrieves and displays all teachers.

    - **Functionality**: Calls `teacherDAO.getAllTeachers()` to fetch and display all teachers from the database.
    - **Return Value**: void
    - **Explanation**: Displays all teachers and their relevant details as `"ID: " + teacher.getId() + ", Name: " + teacher.getName() + ", Subject: " + teacher.getSubject() + ", Designation: " + teacher.getDesignation()`.

updateDepartment()

- **Task**: Updates an existing department.

    - **Functionality**: Prompts the user for the department ID and the new department name, then updates the department using `departmentDAO.updateDepartment()`.
    - **Return Value**: void
    - **Explanation**: If department is found then prompts the updated name from user and update it, otherwise, an error message is displayed as `"Department with ID " + id + " does not exist."`.

updateTeacher()

- **Task**: Updates an existing teacher's details.

    - **Functionality**: Prompts the user for the teacher ID and the new details (name, subject, designation, department ID), and updates the teacher’s information using `teacherDAO.updateTeacher()`.
    - **Return Value**: void
    - **Explanation**: If teacher is found by it’s id then prompts for more details like name, subject, designation and department id and update it otherwise, an error message is displayed as `"Teacher with ID " + id + " does not exist."`.

deleteDepartment()

- **Task**: Deletes a department.

    - **Functionality**: Prompts the user for the department ID and calls `departmentDAO.deleteDepartment()` to delete it from the database.
    - **Return Value**: void
    - **Explanation**: A success message is displayed if the department is deleted as `"Department deleted successfully."`.

deleteTeacher()

- **Task**: Deletes a teacher.

    - **Functionality**: Prompts the user for the teacher ID and calls `teacherDAO.deleteTeacher()` to delete the teacher from the database.
    - **Return Value**: void
    - **Explanation**: A success message is displayed if the teacher is deleted as `"Teacher deleted successfully."`.

searchTeachersByName()

- **Task**: Searches for teachers by name.

    - **Functionality**: Prompts the user for the teacher name and calls `teacherDAO.searchTeachersByName()` to retrieve matching teachers from the database.
    - **Return Value**: List of `Teacher` objects.
    - **Explanation**: Displays the list of teachers with the matching name by printing each teacher on a new line otherwise print `"No teachers found with the given name."`.

searchTeachersBySubject()

- **Task**: Searches for teachers by subject.

    - **Functionality**: Prompts the user for the subject and calls `teacherDAO.searchTeachersBySubject()` to retrieve matching teachers from the database.
    - **Return Value**: List of `Teacher` objects.
    - **Explanation**: Displays the list of teachers teaching the specified subject by printing each teacher on a new line otherwise print `"No teachers found with the given subject."`.

searchDepartmentsByName()

- **Task**: Searches for departments by name.

    - **Functionality**: Prompts the user for the department name and calls `departmentDAO.searchDepartmentsByName()` to retrieve matching departments from the database.
    - **Return Value**: List of `Department` objects.

- **Explanation**: Displays the list of departments with the matching name by printing each department on a new line otherwise print message as `"No departments found with the given name."`.

Model Classes with Annotations Guidance

Department Class

The **Department** class represents the department entity in the application. Below are the field annotations and their descriptions:

1. **id field (Primary Key)**: The **id** field is expected to be the **primary key** for the **Department** entity. It should be auto-generated by the database using an identity generation strategy.

2. **name field**: This field represents the department's name and should have **not null** validation with a length between **3** and **10 characters**.

3. **teachers field (One-to-Many Relationship)**: This field defines the one-to-many relationship between the **Department** and **Teacher** entities. Each department can have multiple teachers. The cascade operation should be applied for delete and update operations on the department, ensuring that associated teachers are also updated or deleted. Should be enabled with lazy loading.

Teacher Class

The **Teacher** class represents the teacher entity in the application. Below are the field annotations and their descriptions:

1. **id field (Primary Key)**: The **id** field is expected to be the **primary key** for the **Teacher** entity. It should be auto-generated by the database using an identity generation strategy.

2. **name field**: This field represents the teacher's name and should have **not null** validation with a length between **3** and **10 characters**.

3. **subject field**: This field represents the subject taught by the teacher and should have **not null** validation.

4. **designation field**: This field represents the teacher's designation (e.g., Assistant Professor, Lecturer) and should have **not null** validation.

5. **departmentId field**: This field represents the department the teacher belongs to. It should be **not null**, and represents a foreign key linking the teacher to a department.

DAOImpl Classes and Method Descriptions

DepartmentDAOImpl Class

createDepartment(Department department)

- **Task**: Inserts a new department into the database.

  - **Functionality**: Executes an SQL `INSERT` query to add the department to the database.

  - **Return Value**: Department object (with generated ID).

  - **Explanation**: This method inserts the provided `Department` object into the database and returns the department object with the generated ID.

getDepartmentById(int id)

- **Task**: Retrieves a department by its ID.

  - **Functionality**: Executes a query to retrieve the department details by ID from the database.

  - **Return Value**: `Department` object representing the department.

  - **Explanation**: This method retrieves the department details from the database using the provided ID.

getAllDepartments()

- **Task**: Retrieves all departments from the database.

  - **Functionality**: Executes a `SELECT` query to retrieve all departments and their associated teachers.

  - **Return Value**: List of `Department` objects.

  - **Explanation**: This method retrieves all departments available in the database and their associated teachers.

updateDepartment(Department department)

- **Task**: Updates an existing department in the database.

- **Functionality**: Executes an SQL `UPDATE` query to modify the department details.

- **Return Value**: void

- **Explanation**: This method updates the department information in the database.

deleteDepartment(int id)

- **Task**: Deletes a department from the database.

- **Functionality**: Executes an SQL `DELETE` query to remove the department by its ID.

- **Return Value**: boolean indicating success.

- **Explanation**: This method deletes the department from the database using the provided ID.

searchDepartmentsByName(String name)

- **Task**: Searches for departments by their name.

- **Functionality**: Executes a `SELECT` query to retrieve departments where the name matches the given criteria.

- **Return Value**: List of `Department` objects.

- **Explanation**: This method finds and returns departments whose names match the search string.

TeacherDAOImpl Class

createTeacher(Teacher teacher)

- **Task**: Inserts a new teacher into the database.

- **Functionality**: Executes an SQL `INSERT` query to add the teacher to the database.

- **Return Value**: `Teacher` object (with generated ID).

- **Explanation**: This method inserts the provided `Teacher` object into the database and returns the teacher object with the generated ID.

getTeacherById(int id)

- **Task**: Retrieves a teacher by its ID.

  - **Functionality**: Executes a query to retrieve the teacher details by ID from the database.

  - **Return Value**: `Teacher` object representing the teacher.

  - **Explanation**: This method retrieves the teacher details from the database using the provided ID.

getAllTeachers()

- **Task**: Retrieves all teachers from the database.

  - **Functionality**: Executes a `SELECT` query to retrieve all teachers and their department associations.

  - **Return Value**: List of `Teacher` objects.

  - **Explanation**: This method retrieves all teachers available in the database.

getTeachersByDepartment(int departmentId)

- **Task**: Retrieves all teachers from the database by department id.

  - **Functionality**: Executes a `SELECT` query to retrieve all teachers by given department id.

  - **Return Value**: List of `Teacher` objects.

  - **Explanation**: This method retrieves all teachers in particular department available in the database.

updateTeacher(Teacher teacher)

- **Task**: Updates an existing teacher in the database.

  - **Functionality**: Executes an SQL `UPDATE` query to modify the teacher's details.

  - **Return Value**: void

  - **Explanation**: This method updates the teacher information in the database.

deleteTeacher(int id)

- **Task**: Deletes a teacher from the database.

  - **Functionality**: Executes an SQL `DELETE` query to remove the teacher by its ID.

  - **Return Value**: boolean indicating success.

  - **Explanation**: This method deletes the teacher from the database using the provided ID.

searchTeachersByName(String name)

- **Task**: Searches for teachers by their name.

  - **Functionality**: Executes a `SELECT` query to retrieve teachers whose name matches the given search string.

  - **Return Value**: List of `Teacher` objects.

  - **Explanation**: This method finds and returns teachers whose names match the search string.

searchTeachersBySubject(String subject)

- **Task**: Searches for teachers by their subject.

  - **Functionality**: Executes a `SELECT` query to retrieve teachers whose subject matches the given criteria.

  - **Return Value**: List of `Teacher` objects.

  - **Explanation**: This method finds and returns teachers teaching the specified subject.

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.onlinefacultyapplication

Resources

Class/Interface	Description	Status
OnlineFacultyApplication.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.onlinefacultyapplication.model

Resources

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

c. Package: com.onlinefacultyapplication.repository

Resources

Class/Interface	Description	Status
DepartmentDao.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
DepartmentDaoImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to written by test taker	Partially Implemented
TeacherDao.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
TeacherDaoImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to 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. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
6. 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.

7. Default credentials for MySQL:
   a. Username: root
   b. Password: pass@word1

8. 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

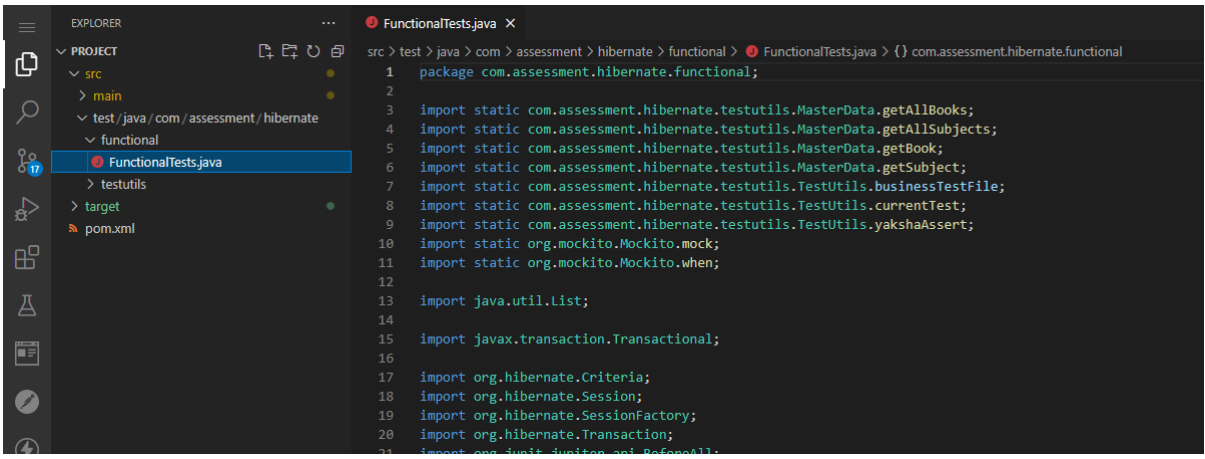
The last command will ask for password which is 'pass@word1'

9. **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.**
10. **To run your project use command:**

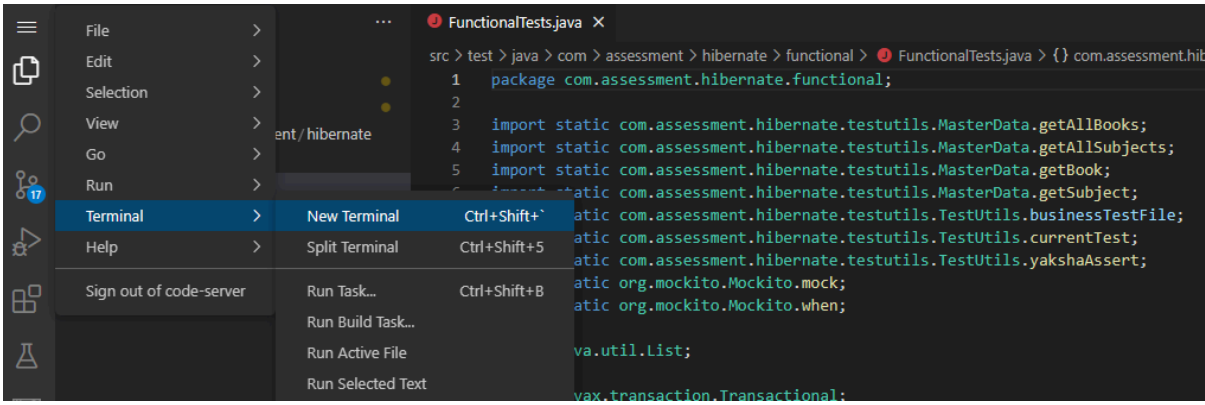
    mvn clean install exec:java -Dexec.mainClass="com.onlinefacultyapplication.OnlineFacultyApplication"

11. **To test your project, use the command**

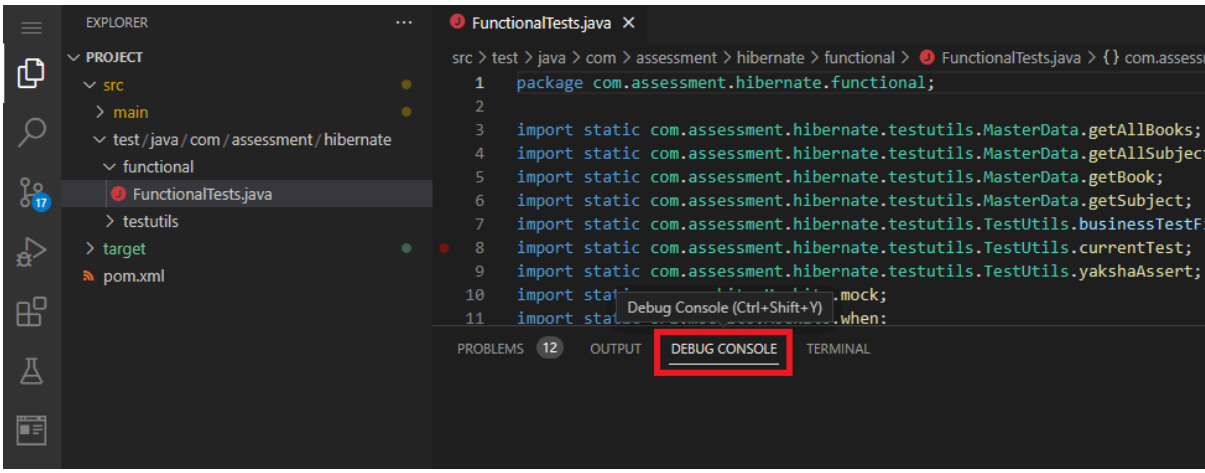
a. **Open FunctionalTests.java file in editor**



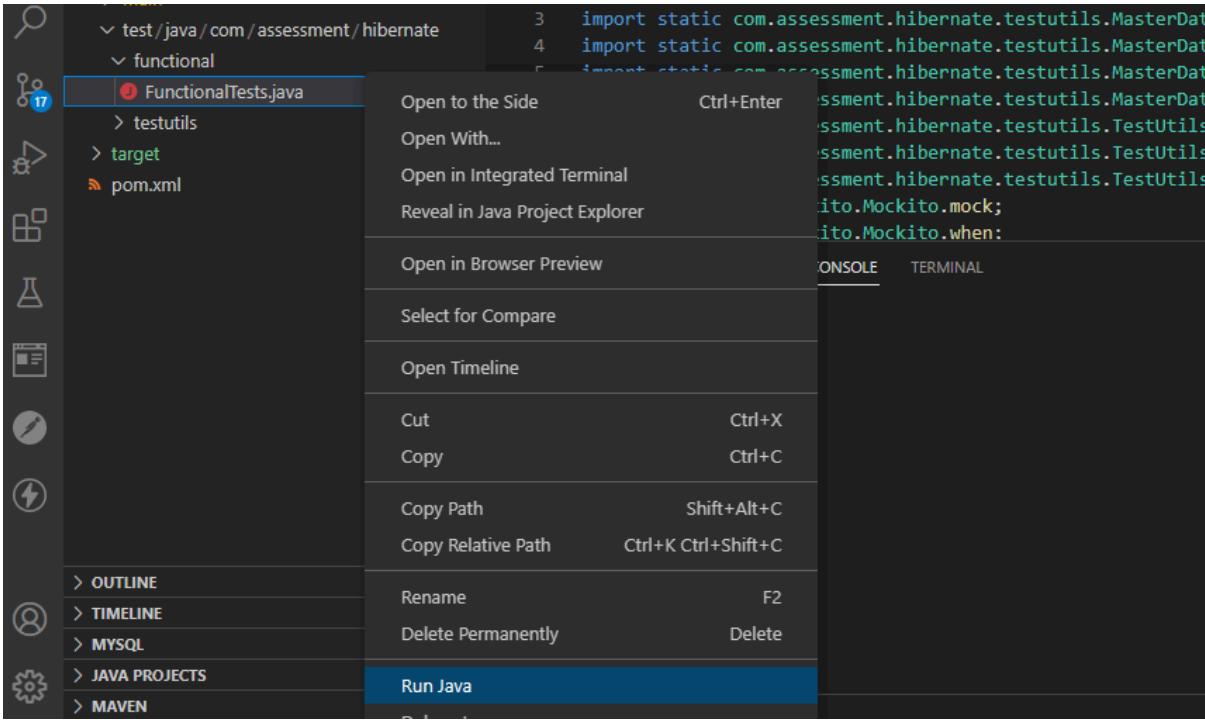
b. **Open a new Terminal**



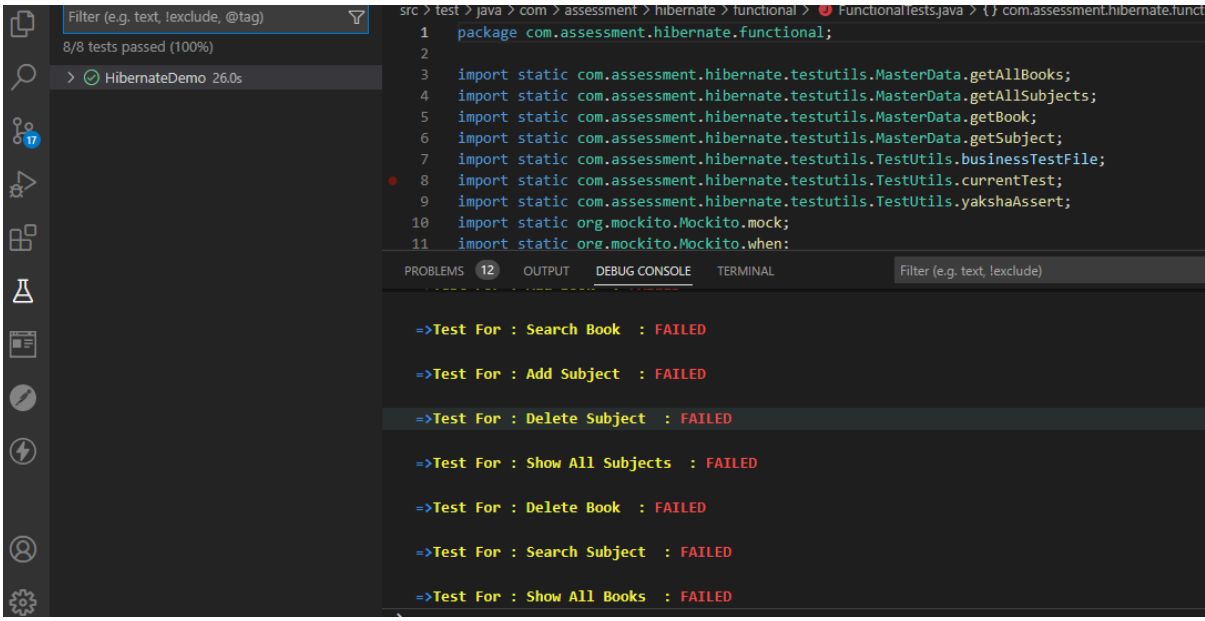
c. Go to Debug Console Tab



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**



12. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**