# Online Food Order Application

IIHT

Time To Complete: 3 hrs

## Contents

## 1 Problem Statement

The Online Food Order Application allows users to perform CRUD (Create, Read, Update, Delete) operations and search functionalities in different criterias on dishes and restaurants. Users can create new dishes and restaurants, update existing dish and restaurant information, delete dishes and restaurants that are no longer available or relevant, and retrieve dish and restaurant details for viewing.

## 2 Business Requirements:

| | |
|---|---|
| Screen Name | Console input screen |
| Problem Statement | 1. User needs to enter into the application.<br>2. The user should be able to do the particular operations<br>3. The console should display the menu<br>1) create restaurant<br>2) create dish<br>3) update restaurant<br>4) update dish<br>5) delete restaurant<br>6) delete dish<br>7) get restaurant by id<br>8) get all restaurant<br>9) get all dishes<br>10) search restaurants by name<br>11) search restaurants by location<br>12) search restaurants by dishname<br>13) search dishes by ingredients<br>14) remove all dishes<br>15) delete all restaurants<br>16) exit |

# Business Requirements

## System Features

The application should provide the following functionalities to the user:

- Create Restaurant: Add a new restaurant to the system.
- Create Dish: Add a new dish to the system, associated with a specific restaurant.
- Update Restaurant: Modify the details of an existing restaurant.
- Update Dish: Modify the details of an existing dish.
- Delete Restaurant: Remove a restaurant from the system.
- Delete Dish: Remove a dish from the system.
- Get Restaurant by ID: Retrieve details of a restaurant by its unique identifier.
- Get All Restaurants: List all restaurants in the system.
- Get All Dishes: List all dishes in the system.
- Search Restaurants by Name: Search for restaurants by their name.
- Search Restaurants by Location: Search for restaurants by their location.
- Search Restaurants by Dish Name: Search for restaurants serving a specific dish.
- Search Dishes by Ingredients: Search for dishes containing specific ingredients.
- Delete All Dishes: Remove all dishes from the system.
- Delete All Restaurants: Remove all restaurants from the system.

# Classes and Method Descriptions

## OnlineFoodOrderApplication Class

- Entry point for the application, handling the execution flow.

  - Prompts users with a menu to perform CRUD operations on restaurants and dishes.

  - You can define all other methods present in this class as:

## getAllDishes()

- **Task**: Retrieves and displays all dishes in the system.

  - **Functionality**: This method calls the `dishDAO.getAllDishes()` method to retrieve a list of all dishes available in the database. It checks if any dishes are available and prints them to the console.
  - **Return Value**: void
  - **Explanation**: If the list of dishes is empty, it should print "No dishes found.". Otherwise, it prints each dish in a new line.

## printMenu()

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

  - **Functionality**: This method prints a list of options for the user to choose from, including operations like creating, updating, deleting restaurants and dishes as mentioned in business requirements above.
  - **Return Value**: void
  - **Explanation**: The menu options are printed to the console to help the user navigate through the system.

## getUserChoice()

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

  - **Functionality**: This method prompts the user to enter their choice, which is an integer corresponding to one of the menu options.
  - **Return Value**: int (The user's choice from the menu)
  - **Explanation**: The method reads the user's input from the console and returns it as an integer to determine which operation to perform.

## createRestaurant()

- **Task**: Creates a new restaurant in the system.

    - **Functionality**: This method prompts the user for the restaurant's name and address, then creates a `Restaurant` object and calls `restaurantDAO.createRestaurant()` to save it in the database.
    - **Return Value**: void
    - **Explanation**: If the restaurant is created successfully, a message is displayed with the restaurant's details as `"Restaurant created successfully: " + restaurant`.

## createDish()

- **Task**: Creates a new dish and associates it with a restaurant.

    - **Functionality**: This method prompts the user for the dish's name, ingredients, and associated restaurant ID. It checks if the restaurant exists, then creates a `Dish` object and calls `dishDAO.createDish()` to add it to the database.
    - **Return Value**: void
    - **Explanation**: If the restaurant ID exists, the dish is created and added to the system; otherwise, an error message is displayed as `"Restaurant not found with ID: " + restaurantId`.

## updateRestaurant()

- **Task**: Updates the details of an existing restaurant.

    - **Functionality**: This method prompts the user to enter the restaurant ID, then retrieves the details from user and updates the restaurant's details like name and address, followed by calling `restaurantDAO.updateRestaurant()` to save the changes.
    - **Return Value**: void
    - **Explanation**: If the restaurant exists, it is updated with the new information; if not, a message is displayed indicating no such restaurant was found as `"Restaurant not found with ID: " + updateRestaurantId`.

## updateDish()

- **Task**: Updates the details of an existing dish.

- **Functionality**: This method prompts the user to enter the dish ID, retrieves the dish using `dishDAO.getDishById()`, and asks user to write details to update like name and ingredients and then updates its details. It then calls `dishDAO.updateDish()` to save the updated dish.
- **Return Value**: void
- **Explanation**: If the dish exists, it is updated; if not, a message is displayed indicating no such dish was found as `"Dish not found with ID: " + updateDishId`.

## deleteRestaurant()

- **Task**: Deletes a restaurant from the system.

- **Functionality**: This method prompts the user to enter the restaurant ID, and calls `restaurantDAO.deleteRestaurant()` to remove the restaurant from the database.
- **Return Value**: void
- **Explanation**: After deletion, a success message is displayed with the restaurant ID as `"Restaurant deleted successfully with ID: " + deleteRestaurantId`.

## deleteDish()

- **Task**: Deletes a dish from the system.

- **Functionality**: This method prompts the user to enter the dish ID, and calls `dishDAO.deleteDish()` to remove the dish from the database.
- **Return Value**: void
- **Explanation**: After deletion, a success message is displayed with the dish ID as `"Dish deleted successfully with ID: " + deleteDishId`.

## getRestaurantById()

- **Task**: Retrieves a restaurant by its unique ID.

- **Functionality**: This method prompts the user to enter the restaurant ID, and calls `restaurantDAO.getRestaurantById()` to retrieve and display the restaurant’s details.
- **Return Value**: void
- **Explanation**: If the restaurant is found, its details are displayed as `"Retrieved Restaurant: " + retrievedRestaurant`; if not, an error message is shown as `"Restaurant not found with ID: " + retrieveRestaurantId`.

## getAllRestaurants()

- **Task**: Retrieves and displays all restaurants in the system.

  - **Functionality**: This method calls `restaurantDAO.getAllRestaurants()` to fetch and display all restaurants from the database.
  - **Return Value**: void
  - **Explanation**: Displays the details of all the restaurants available in the system as `"All Restaurants: " + allRestaurants`.

## searchRestaurantsByName()

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

  - **Functionality**: This method prompts the user for a restaurant name, then searches for matching restaurants using `restaurantDAO.searchRestaurantsByName()`.
  - **Return Value**: void
  - **Explanation**: Displays matching restaurants or a message if no matches are found as `"No restaurants found with the given name."`.

## searchRestaurantsByLocation()

- **Task**: Searches for restaurants by their location.

  - **Functionality**: This method prompts the user for a location, then searches for matching restaurants using `restaurantDAO.searchRestaurantsByLocation()`.
  - **Return Value**: void
  - **Explanation**: Displays matching restaurants or a message if no matches are found as `"No restaurants found in the given location."`.

## searchRestaurantsByDishName()

- **Task**: Searches for restaurants serving a specific dish.

  - **Functionality**: This method prompts the user for a dish name, then searches for matching restaurants using `restaurantDAO.searchRestaurantsByDishName()`.
  - **Return Value**: void
  - **Explanation**: Displays matching restaurants or a message if no matches are found as `"No restaurants found with the given dish name."`.

## searchDishesByIngredients()

- **Task**: Searches for dishes by their ingredients.

- **Functionality**: This method prompts the user for ingredients, then searches for matching dishes using `dishDAO.searchDishesByIngredients()`.
- **Return Value**: void
- **Explanation**: Displays matching dishes or a message if no matches are found as `"No dishes found with the given ingredients."`.

## removeAllDishes()

- **Task**: Removes all dishes from the system.

- **Functionality**: This method calls `dishDAO.removeAllDishes()` to delete all dishes from the database.
- **Return Value**: void
- **Explanation**: Displays a success message after removing all dishes as `"All dishes have been removed from the database."`.

## deleteAllRestaurants()

- **Task**: Removes all restaurants from the system.

- **Functionality**: This method calls `restaurantDAO.deleteAllRestaurants()` to delete all restaurants from the database.
- **Return Value**: void
- **Explanation**: Displays a success message after removing all restaurants as `"All restaurants have been deleted from the database."`.

## Dish Class

### Field Annotations for Dish

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

2. **name field**:
- **Not Null**: This field must not be null.
- **Length Validation**: The **name** should have a length between 3 and 10 characters.

3. **ingredients field**:
- **Not Null**: This field must not be null.
- **Length Validation**: The **ingredients** should have a length between 3 and 100 characters.

4. **restaurantId field**:
- **Not Null**: This field must not be null and links the dish to a specific restaurant.

## Restaurant Class

### Field Annotations for Restaurant

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

2. **name field**:
- **Not Null**: This field must not be null.
- **Length Validation**: The **name** should have a length between 3 and 20 characters.

3. **address field**:
- **Not Null**: This field must not be null.
- **Length Validation**: The **address** should have a length between 3 and 10 characters.

4. **dishes field (One-to-Many Relationship)**:
- Defines a one-to-many relationship between **Restaurant** and **Dish**.
- **Cascade**: If a restaurant is deleted, its associated dishes should also be deleted.
- **Lazy Loading**: Dishes are loaded only when accessed.

# DAO Classes and Method Descriptions

## DishDAOImpl Class

### createDish(Dish dish)

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

    - **Functionality**: Executes an SQL `INSERT` query to add the dish to the database.
    - **Return Value**: void
    - **Explanation**: This method inserts the provided `Dish` object into the database by executing an SQL insert query.

### getDishById(int id)

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

    - **Functionality**: Executes a query to retrieve the dish details by ID from the database.
    - **Return Value**: `Dish` object representing the dish.
    - **Explanation**: This method fetches the details of a dish using the provided `id` by querying the database.

### getAllDishes()

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

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

### updateDish(Dish dish)

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

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

### deleteDish(int id)

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

  - **Functionality**: Executes an SQL `DELETE` query to remove the dish by its ID.
  - **Return Value**: boolean indicating success.
  - **Explanation**: This method deletes the dish from the database using the provided `id`.

### searchDishesByIngredients(String ingredients)

- **Task**: Searches for dishes containing specific ingredients.

  - **Functionality**: Executes a `SELECT` query to search dishes by ingredients.
  - **Return Value**: List of `Dish` objects.
  - **Explanation**: This method fetches dishes that contain the specified `ingredients`.

### removeAllDishes()

- **Task**: Deletes all dishes from the database.

  - **Functionality**: Executes a `DELETE` query to remove all dishes.
  - **Return Value**: void
  - **Explanation**: This method removes all dishes from the database.

## RestaurantDAOImpl Class

### createRestaurant(Restaurant restaurant)

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

  - **Functionality**: Executes an SQL `INSERT` query to add the restaurant to the database.
  - **Return Value**: void
  - **Explanation**: This method inserts the provided `Restaurant` object into the database.

### getRestaurantById(int id)

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

  - **Functionality**: Executes an SQL query to retrieve the restaurant and its

associated dishes by ID.
- **Return Value**: `Restaurant` object.
- **Explanation**: This method fetches the restaurant details using the provided `id`.

## getAllRestaurants()

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

- **Functionality**: Executes a `SELECT` query to retrieve all restaurants and their associated dishes.
- **Return Value**: List of `Restaurant` objects.
- **Explanation**: This method retrieves all restaurant entries from the database.

## updateRestaurant(Restaurant restaurant)

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

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

## deleteRestaurant(int id)

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

- **Functionality**: Executes an SQL `DELETE` query to remove the restaurant by its ID.
- **Return Value**: boolean indicating success.
- **Explanation**: This method deletes the restaurant from the database using the provided `id`.

## searchRestaurantsByName(String name)

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

- **Functionality**: Executes a `SELECT` query to search for restaurants by name.
- **Return Value**: List of `Restaurant` objects.
- **Explanation**: This method retrieves all restaurants matching the `name` provided.

### searchRestaurantsByLocation(String location)

- **Task**: Searches for restaurants by their location.

    - **Functionality**: Executes a query to search for restaurants by location.
    - **Return Value**: List of `Restaurant` objects.
    - **Explanation**: This method retrieves all restaurants located in the specified `location`.

### searchRestaurantsByDishName(String dishName)

- **Task**: Searches for restaurants serving a specific dish.

    - **Functionality**: Executes a `JOIN` query between `restaurant` and `dish` tables to find restaurants serving the dish.
    - **Return Value**: List of `Restaurant` objects.
    - **Explanation**: This method finds restaurants that serve dishes matching the `dishName`.

### deleteAllRestaurants()

- **Task**: Deletes all restaurants from the database.

    - **Functionality**: Executes a `DELETE` query to remove all restaurants.
    - **Return Value**: void
    - **Explanation**: This method deletes all restaurants from the database.

## 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.onlinefoodorderapplication

**Resources**

| Class/Interface | Description | Status |
|---|---|---|

| | | |
|---|---|---|
| OnlineFoodOrderApplication.java(class) | This represents bootstrap class i.e class with Main method, that shall contain all console interaction with user. | Partially Implemented |

### b. PACKAGE: COM.ONLINEFOODORDERAPPLICATION.MODEL

Resources

| Class/Interface | Description | Status |
|---|---|---|
| Dish.java(class) | This represents entity class for Dish | Partially Implemented |
| Restaurant.java(class) | This represents entity class for Restaurant | Partially Implemented |

### c. PACKAGE: COM.ONLINEFOODORDERAPPLICATION.REPOSITORY

Resources

| Class/Interface | Description | Status |
|---|---|---|
| DishDao.java(interface) | This is an interface containing declaration of DAO method | Already Implemented |
| DishDaoImpl.java(class) | This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to written by test taker | Partially Implemented |
| RestaurantDao.java(interface) | This is an interface containing declaration of DAO method | Already Implemented |
| RestaurantDaoImpl.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. a. sudo systemctl enable mysql

b. sudo systemctl start mysql

   NOTE: After typing any of the above commands you might encounter any warnings.

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

c. mysql -u root -p

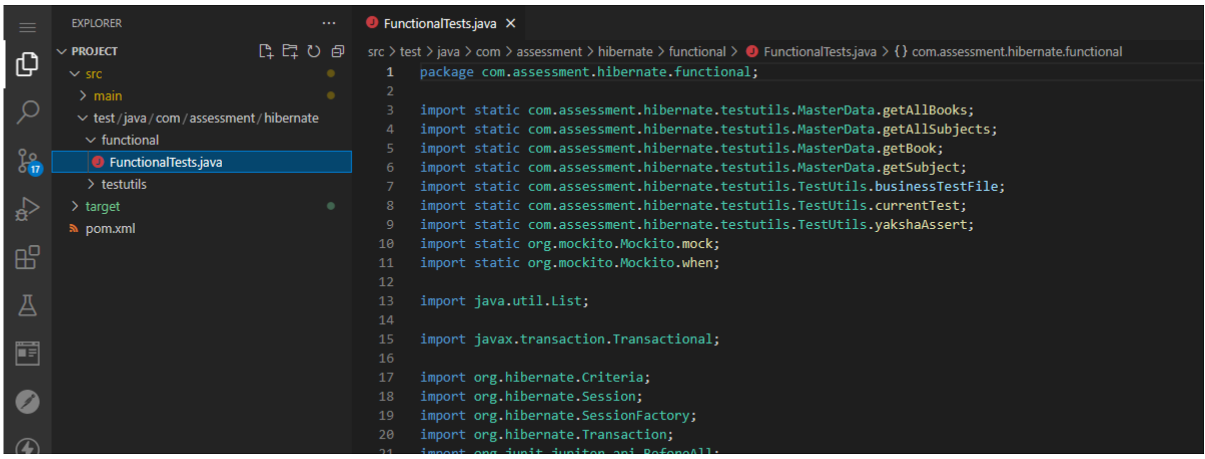
   The last command will ask for password which is 'pass@word1'

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.onlinefoodorderapplication.OnlineFoodOrderApplication"
   ```

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

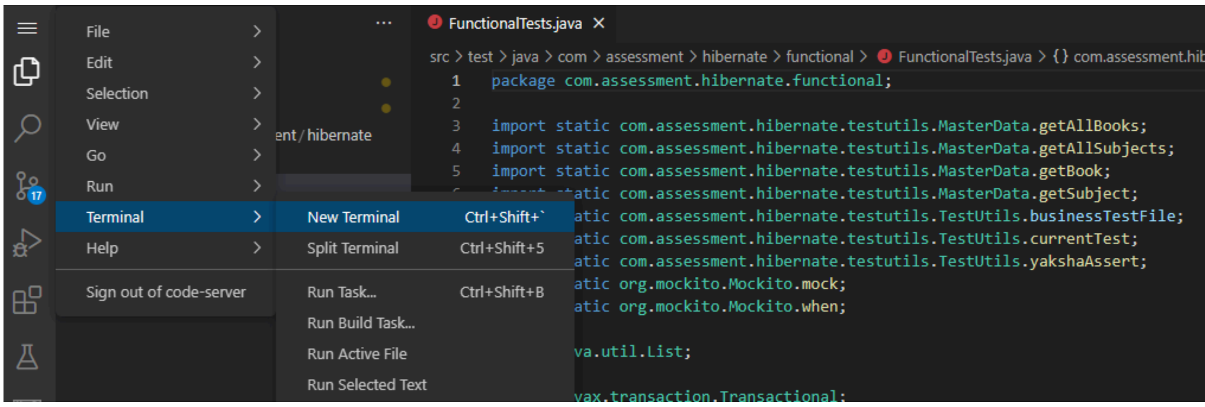
a. **Open FunctionalTests.java file in editor**



EXPLORER
PROJECT
src
main
test / java / com / assessment / hibernate
functional
FunctionalTests.java
testutils
target
pom.xml
FunctionalTests.java
src > test > java > com > assessment > hibernate > functional > FunctionalTests.java > {} com.assessment.hibernate.functional

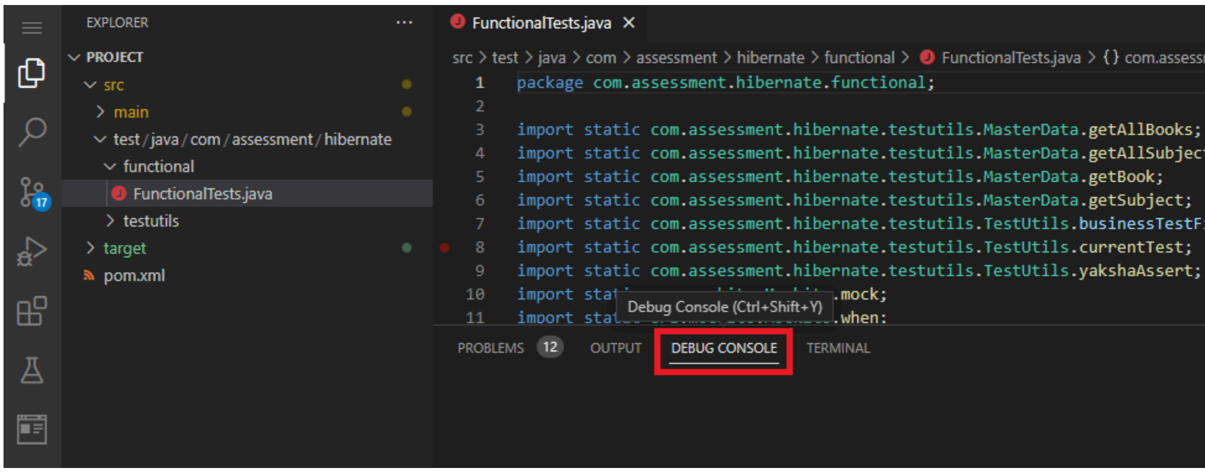
```java
package com.assessment.hibernate.functional;

import static com.assessment.hibernate.testutils.MasterData.getAllBooks;
import static com.assessment.hibernate.testutils.MasterData.getAllSubjects;
import static com.assessment.hibernate.testutils.MasterData.getBook;
import static com.assessment.hibernate.testutils.MasterData.getSubject;
import static com.assessment.hibernate.testutils.TestUtils.businessTestFile;
import static com.assessment.hibernate.testutils.TestUtils.currentTest;
import static com.assessment.hibernate.testutils.TestUtils.yakshaAssert;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import java.util.List;

import javax.transaction.Transactional;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.junit.jupiter.api.BeforeAll;
```

b. **Open a new Terminal**



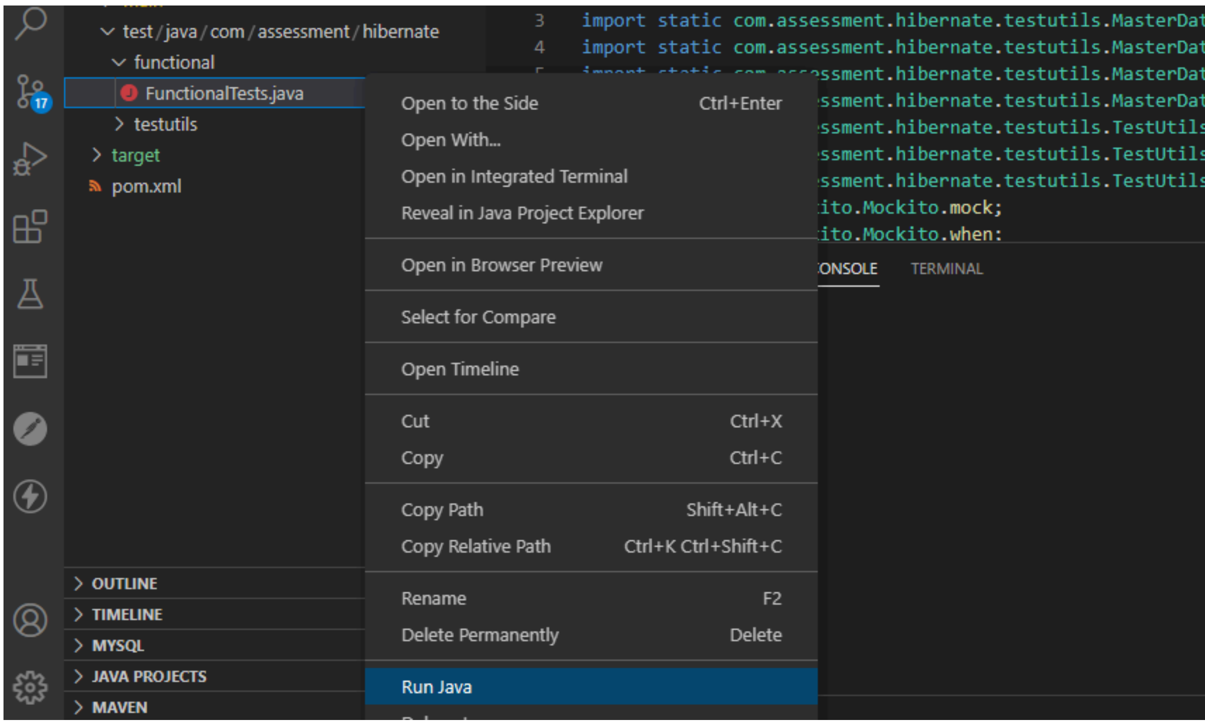
```java
package com.assessment.hibernate.functional;

import static com.assessment.hibernate.testutils.MasterData.getAllBooks;
import static com.assessment.hibernate.testutils.MasterData.getAllSubjects;
import static com.assessment.hibernate.testutils.MasterData.getBook;
atic com.assessment.hibernate.testutils.MasterData.getSubject;
atic com.assessment.hibernate.testutils.TestUtils.businessTestFile;
atic com.assessment.hibernate.testutils.TestUtils.currentTest;
atic com.assessment.hibernate.testutils.TestUtils.yakshaAssert;
atic org.mockito.Mockito.mock;
atic org.mockito.Mockito.when;

va.util.List;

vax.transaction.Transactional;
```

c. **Go to Debug Console Tab**


EXPLORER
PROJECT
src
main
test / java / com / assessment / hibernate
functional
FunctionalTests.java
testutils
target
pom.xml
FunctionalTests.java
src > test > java > com > assessment > hibernate > functional > FunctionalTests.java > {} com.assess
Debug Console (Ctrl+Shift+Y)
PROBLEMS 12
OUTPUT
DEBUG CONSOLE
TERMINAL

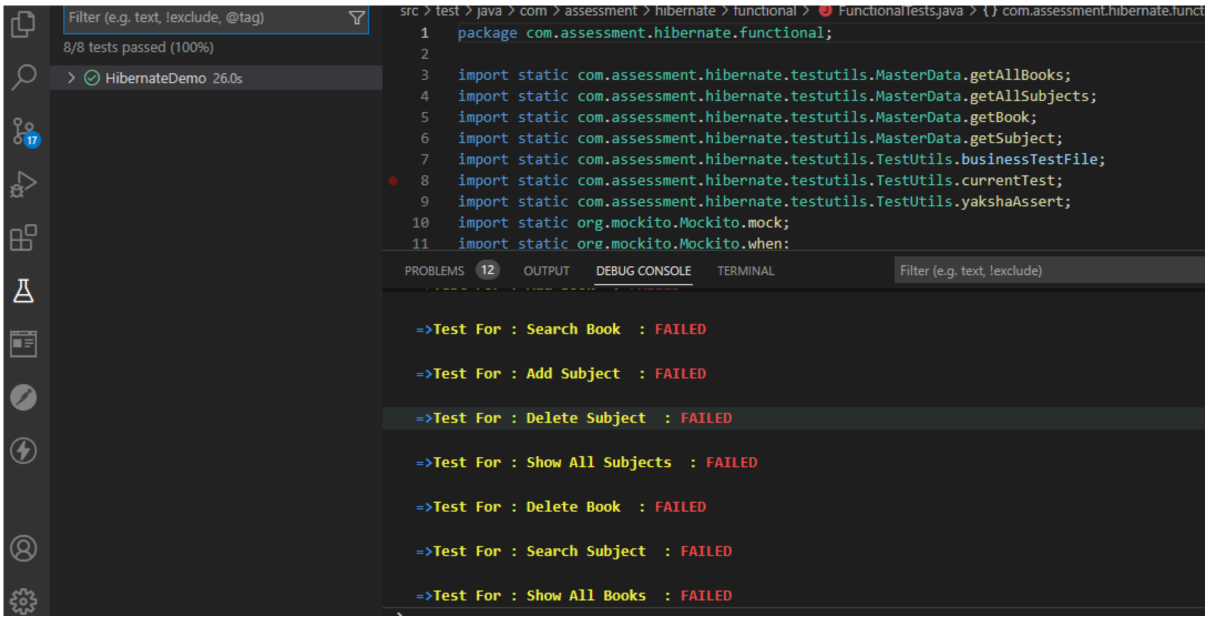
```
package com.assessment.hibernate.functional;

import static com.assessment.hibernate.testutils.MasterData.getAllBooks;
import static com.assessment.hibernate.testutils.MasterData.getAllSubjec
import static com.assessment.hibernate.testutils.MasterData.getBook;
import static com.assessment.hibernate.testutils.MasterData.getSubject;
import static com.assessment.hibernate.testutils.TestUtils.businessTestF
import static com.assessment.hibernate.testutils.TestUtils.currentTest;
import static com.assessment.hibernate.testutils.TestUtils.yakshaAssert;
import stat                                        .mock;
import stat                                        .when;
```

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



```
import static com.assessment.hibernate.testutils.MasterDat
import static com.assessment.hibernate.testutils.MasterDat
import static com.assessment.hibernate.testutils.MasterDat
                 ssment.hibernate.testutils.MasterDat
                 ssment.hibernate.testutils.TestUtils
                 ssment.hibernate.testutils.TestUtils
                 ssment.hibernate.testutils.TestUtils
                 ito.Mockito.mock;
                 ito.Mockito.when;
```

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


Filter (e.g. text, !exclude, @tag)
8/8 tests passed (100%)
HibernateDemo 26.0s
src > test > java > com > assessment > hibernate > functional > FunctionalTests.java > { } com.assessment.hibernate.funct
PROBLEMS 12
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, !exclude)

```java
package com.assessment.hibernate.functional;

import static com.assessment.hibernate.testutils.MasterData.getAllBooks;
import static com.assessment.hibernate.testutils.MasterData.getAllSubjects;
import static com.assessment.hibernate.testutils.MasterData.getBook;
import static com.assessment.hibernate.testutils.MasterData.getSubject;
import static com.assessment.hibernate.testutils.TestUtils.businessTestFile;
import static com.assessment.hibernate.testutils.TestUtils.currentTest;
import static com.assessment.hibernate.testutils.TestUtils.yakshaAssert;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;
```

```
=>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
```

**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.**