Player’s Selection Application

IIHT

Time To Complete: 3 hrs

Contents

1 PROBLEM STATEMENT

The Players Selection Application allows users to perform CRUD (Create, Read, Update, Delete) operations and search functionalities in different criterias on players and scores. Users can create new player profiles, record and update scores for each player, delete player entries, and retrieve player and score 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 a player 2) create a score 3) update player 4) Update score 5) delete player 6) delete score 7) show all players 8) show all scores 9) search players by name 10) search players by domestic team 11) get scores by player ID 12) get average of last three scores for a player 13) exit

Business Requirements

System Features

The application should provide the following functionalities to the user:

- **Create Player**: Add a new player to the system.
- **Create Score**: Add a new score for a player.
- **Update Player**: Modify the details of an existing player.
- **Update Score**: Modify the details of an existing score.
- **Delete Player**: Remove a player from the system.

- **Delete Score**: Remove a score from the system.
- **Get All Players**: List all players in the system.
- **Get All Scores**: List all scores in the system.
- **Search Players by Name**: Search for players by their name.
- **Search Players by Domestic Team**: Search for players by their domestic team.
- **Get Scores by Player ID**: Retrieve scores for a specific player.
- **Get Average of Last Three Scores**: Calculate the average of the last three scores for a player.

Classes and Method Descriptions

PlayersSelectionApplication Class

showOptions()

- **Task**: Show's menu and take user's input.
- **Functionality**: Show the complete menu as per above mentioned business requirement and 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.

addPlayer()

- **Task**: Adds a new player to the system.
- **Functionality**: Prompts the user for the player's name and domestic team, creates a `Player` object, and calls `playerDAO.addPlayer()` to save it to the database.
- **Return Value**: void
- **Explanation**: This method adds a new player to the database and displays a success message as `"Player added successfully."`.

addScore()

- **Task**: Adds a new score for a player.
- **Functionality**: Prompts the user for the player ID and the score, creates a `Score` object, and calls `scoreDAO.addScore()` to save it to the database.
- **Return Value**: void
- **Explanation**: This method adds a score for the given player and displays a success message as `"Score added successfully."`.

updatePlayer()

- **Task**: Updates an existing player.
- **Functionality**: Prompts the user for the player ID and new details (name, domestic team), and calls `playerDAO.updatePlayer()` to update the player's information.

- **Return Value**: void
- **Explanation**: This method updates the player's information in the database if it exists and prints message as `"Player updated successfully."` otherwise it should print `"Player not found!"`.

updateScore()

- **Task**: Updates an existing score.
    - **Functionality**: Prompts the user for the score ID and new score, and calls `scoreDAO.updateScore()` to update the score in the database.
    - **Return Value**: void
    - **Explanation**: This method updates the score information for the specified player and prints message as `"Score updated successfully."` otherwise it should print `"Score not found!"`.

deletePlayer()

- **Task**: Deletes a player from the system.
    - **Functionality**: Prompts the user for the player ID and calls `playerDAO.deletePlayer()` to delete the player from the database.
    - **Return Value**: void
    - **Explanation**: This method deletes the player from the database if it exists otherwise it should print `"Player deleted successfully."`.

deleteScore()

- **Task**: Deletes a score from the system.
    - **Functionality**: Prompts the user for the score ID and calls `scoreDAO.deleteScore()` to delete the score from the database.
    - **Return Value**: void
    - **Explanation**: This method deletes the score from the database if it exists otherwise it should print `"Score not found!"`.

showAllPlayers()

- **Task**: Retrieves all players.
    - **Functionality**: Calls `playerDAO.getAllPlayers()` to fetch all players from the database and displays them.
    - **Return Value**: void
    - **Explanation**: This method retrieves and displays all players in the system. If found then print each player in a separate new line otherwise print `"No players found!"`.

showAllScores()

- **Task**: Retrieves all scores.
    - **Functionality**: Calls `scoreDAO.getAllScores()` to fetch all scores from the database and displays them.
    - **Return Value**: void
    - **Explanation**: This method retrieves and displays all scores in the system and if found print each in a separate new line otherwise print `"No scores found!"`.

searchPlayersByName()

- **Task**: Searches for players by name.
    - **Functionality**: Calls `playerDAO.searchPlayersByName()` to retrieve players by name.
    - **Return Value**: void
    - **Explanation**: This method searches for players whose names match the search string and print each of them in a separate line otherwise print `"No players found matching the search criteria."`.

searchPlayersByDomesticTeam()

- **Task**: Searches for players by domestic team.
    - **Functionality**: Calls `playerDAO.searchPlayersByDomesticTeam()` to retrieve players by their domestic team.
    - **Return Value**: void
    - **Explanation**: This method searches for players belonging to the specified domestic team and if players are found then print each of them in a separate new line otherwise print `"No players found matching the search criteria."`.

getScoresByPlayerId()

- **Task**: Retrieves scores for a specific player.
    - **Functionality**: Calls `scoreDAO.getScoresByPlayerId()` to retrieve all scores for the given player ID.
    - **Return Value**: void
    - **Explanation**: This method retrieves and displays each score associated with the given player in separate new line otherwise print `"No scores found for player ID " + playerId + "."`.

getAverageOfLastThreeScores()

- **Task**: Retrieves the average of the last three scores for a player and prints it.
    - **Functionality**: Calls `scoreDAO.getAverageOfLastThreeScores()` to calculate the average of the last three scores for a given player.
    - **Return Value**: void
    - **Explanation**: This method calculates and displays the average score of the last three scores for the given player as `"Average of last three scores for player ID " + playerId + ": " + average`.

Model Classes with Annotations Guidance

Player Class

The **Player** class represents the player entity in the application. Below are the field annotations and their descriptions. Annotate this class with appropriate annotations that make it an entity and the name of the table should be “Player”.

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

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

3. **domesticTeam field**: This field represents the team that the player belongs to and should have **not null** validation.

4. **average field**: This field represents the player's average score, with a default value of **0** and should be **not null**.

Score Class

The **Score** class represents the score entity in the application. Below are the field annotations and their descriptions. Annotate this class with appropriate annotations that make it an entity and the name of the table should be “Score”.

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

2. **score field**: This field represents the score achieved by the player and should have **not null** validation.

3. **playerId field**: This field represents the ID of the player who achieved the score, and it should be **not null**.

DAOImpl Classes and Method Descriptions

PlayerDAOImpl Class

addPlayer(Player player)

- **Task**: Adds a new player to the database.

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

  - **Return Value**: void

  - **Explanation**: This method inserts the provided \`Player\` object into the database and returns the player object with the generated ID.

updatePlayer(Player player)

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

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

  - **Return Value**: void

  - **Explanation**: This method updates the player's information in the database.

deletePlayer(Player player)

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

  - **Functionality**: Executes an SQL \`DELETE\` query to remove the player by their ID.

  - **Return Value**: void

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

getPlayerById(int id)

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

  - **Functionality**: Executes an SQL \`SELECT\` query to retrieve the player details by ID.

  - **Return Value**: \`Player\` object

- **Explanation**: This method fetches the player’s details using the provided ID.

getAllPlayers()

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

   - **Functionality**: Executes an SQL `SELECT` query to retrieve all players and their details.

   - **Return Value**: List of `Player` objects

   - **Explanation**: This method retrieves all players in the system.

searchPlayersByName(String name)

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

   - **Functionality**: Executes an SQL `SELECT` query to find players with a name matching the provided string.

   - **Return Value**: List of `Player` objects

   - **Explanation**: This method retrieves players whose names match the search criteria.

searchPlayersByDomesticTeam(String domesticTeam)

- **Task**: Searches for players by their domestic team.

   - **Functionality**: Executes an SQL `SELECT` query to find players associated with the given domestic team.

   - **Return Value**: List of `Player` objects

   - **Explanation**: This method retrieves players whose domestic team matches the search criteria.

ScoreDAOImpl Class

addScore(Score score)

- **Task**: Adds a new score for a player.

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

   - **Return Value**: void

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

updateScore(Score score)

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

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

    - **Return Value**: void

    - **Explanation**: This method updates the score information for the specified player.

deleteScore(Score score)

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

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

    - **Return Value**: void

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

getScoreById(int id)

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

    - **Functionality**: Executes an SQL `SELECT` query to retrieve the score details by ID.

    - **Return Value**: `Score` object

    - **Explanation**: This method retrieves the score's details using the provided ID.

getAllScores()

- **Task**: Retrieves all scores.

    - **Functionality**: Executes an SQL `SELECT` query to retrieve all scores and their details.

    - **Return Value**: List of `Score` objects

    - **Explanation**: This method retrieves all scores in the system.

getScoresByPlayerId(int playerId)

- **Task**: Retrieves scores for a specific player.

  - **Functionality**: Executes an SQL `SELECT` query to retrieve all scores for the given player ID.

  - **Return Value**: List of `Score` objects

  - **Explanation**: This method retrieves and displays scores associated with the given player.

getAverageOfLastThreeScores(int playerId)

- **Task**: Retrieves the average of the last three scores for a player.

  - **Functionality**: Executes an SQL query to retrieve the last three scores and calculates their average.

  - **Return Value**: double

  - **Explanation**: This method calculates and returns the average of the last three scores for the given player ID.

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.playersselectionapplication

Resources

Class/Interface	Description	Status
PlayersSelectionApplication .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.playersselectionapplication.model

Resources

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

C. Package: com.playersselectionapplication.repository

Resources

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

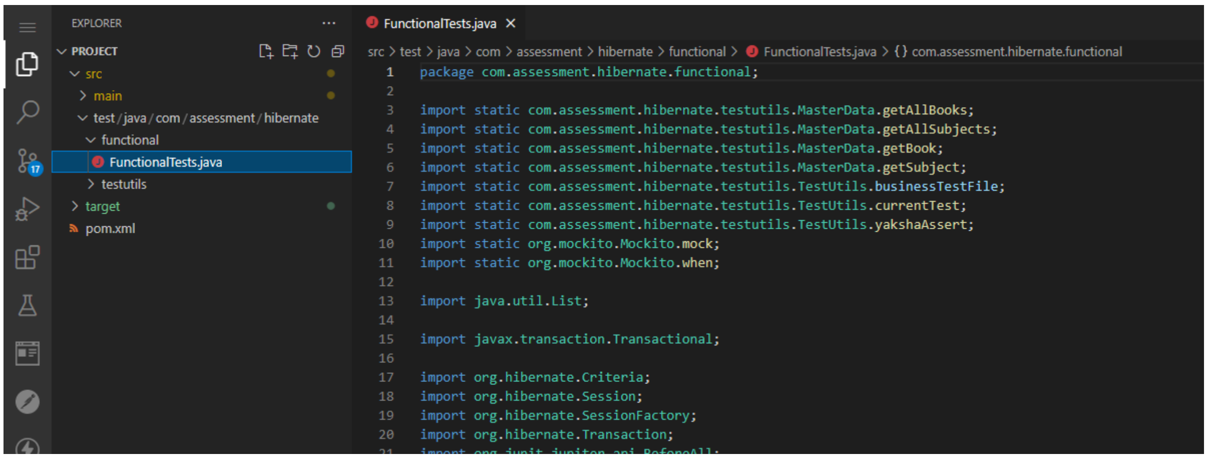
   >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

   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:

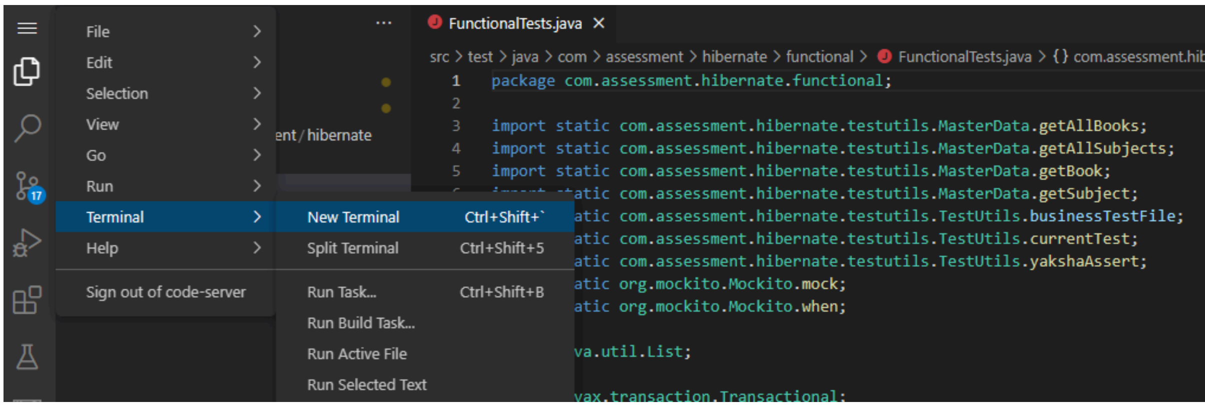
    ```
    mvn clean install exec:java
    -Dexec.mainClass="com.playersselectionapplication.PlayersSelectionApplication"
    ```
11. To test your project, use the command

    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

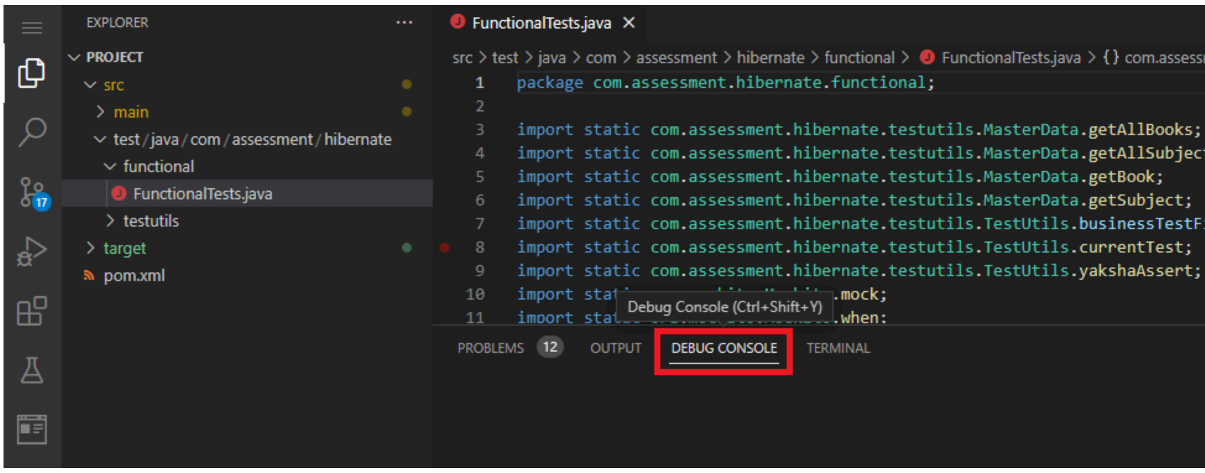
```
package com.assessment.hibernate.functional;

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;
```

b. Open a new Terminal


File
Edit
Selection
View
Go
Run
Terminal
Help
Sign out of code-server
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task... Ctrl+Shift+B
Run Build Task...
Run Active File
Run Selected Text
FunctionalTests.java
src > test > java > com > assessment > hibernate > functional > FunctionalTests.java > {} com.assessment.hib

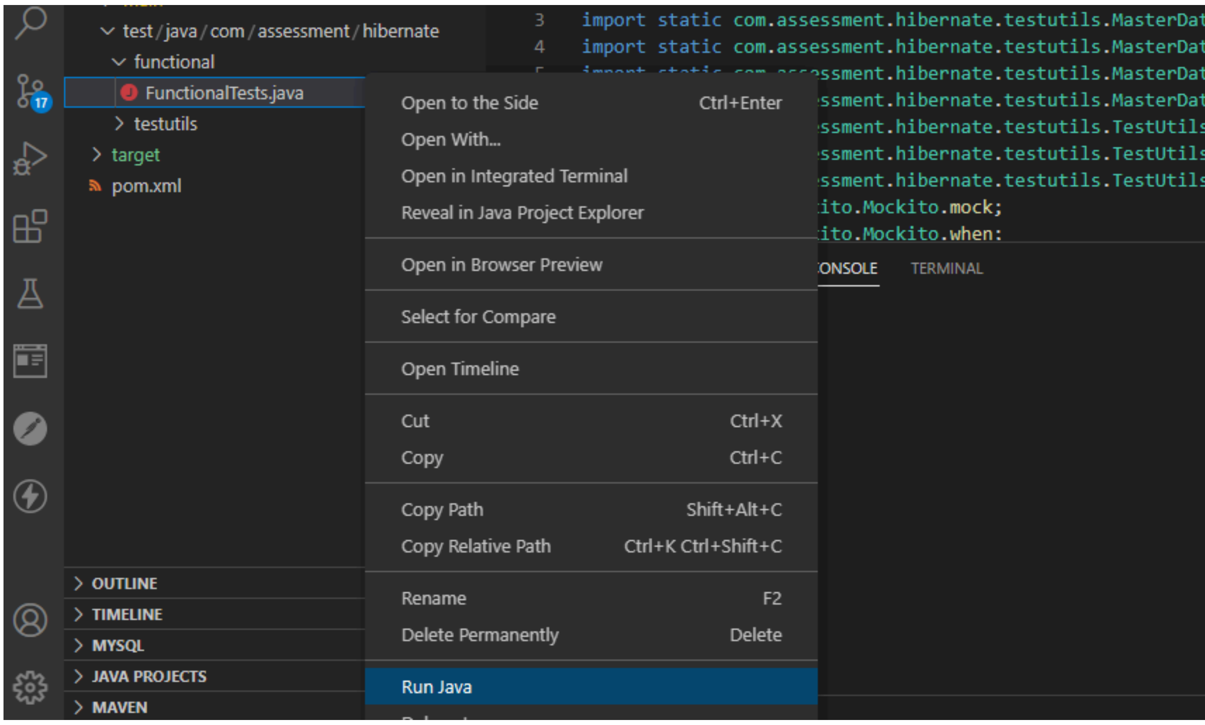
```
package com.assessment.hibernate.functional;

import static com.assessment.hibernate.testutils.MasterData.getAllBooks;
import static com.assessment.hibernate.testutils.MasterData.getAllSubjects;
import static com.assessment.hibernate.testutils.MasterData.getBook;
```

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

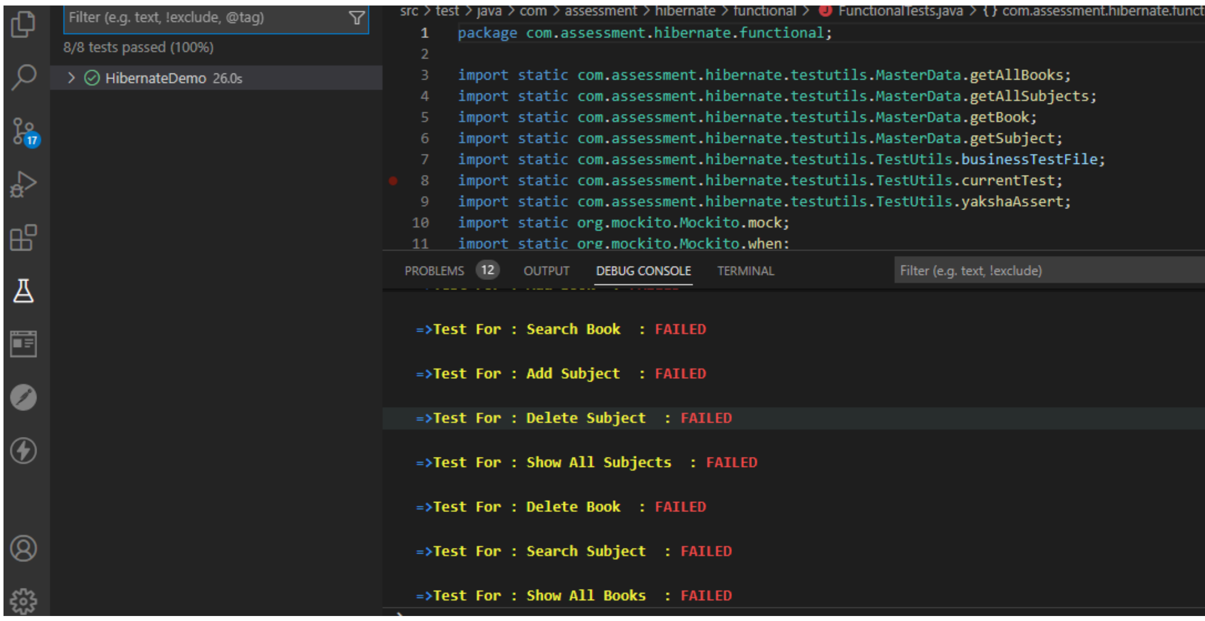
```
package com.assessment.hibernate.functional;

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;
```

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
```

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)

```
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.