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:

A)

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
	    0. exit

2) Business Requirements

a) 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.

3) Classes and Method Descriptions

a) PlayersSelectionApplication Class

i)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.

ii)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."`.

iii)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."`.

iv)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!"`.

v)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!"`.

vi)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."`.

vii)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!"`.

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.

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.

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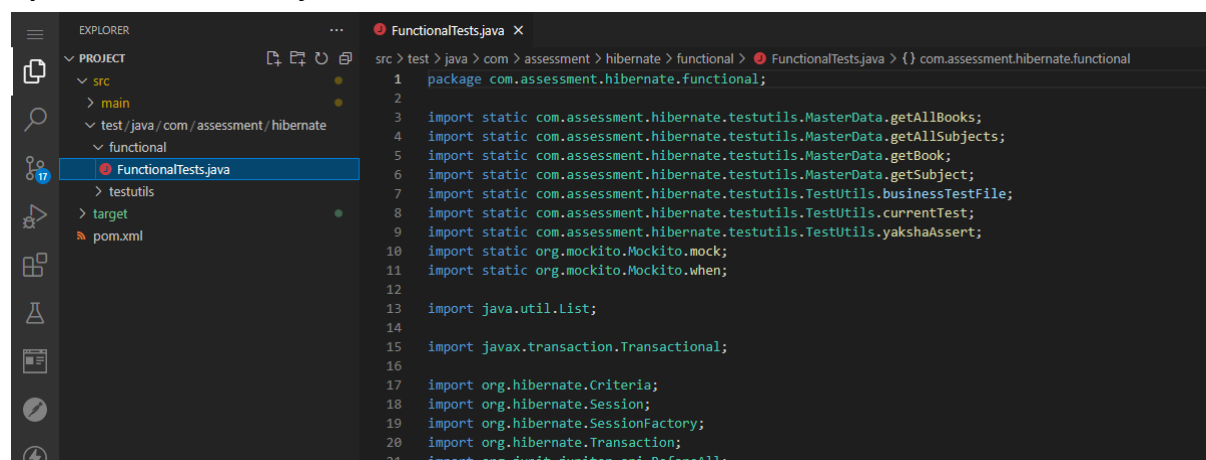
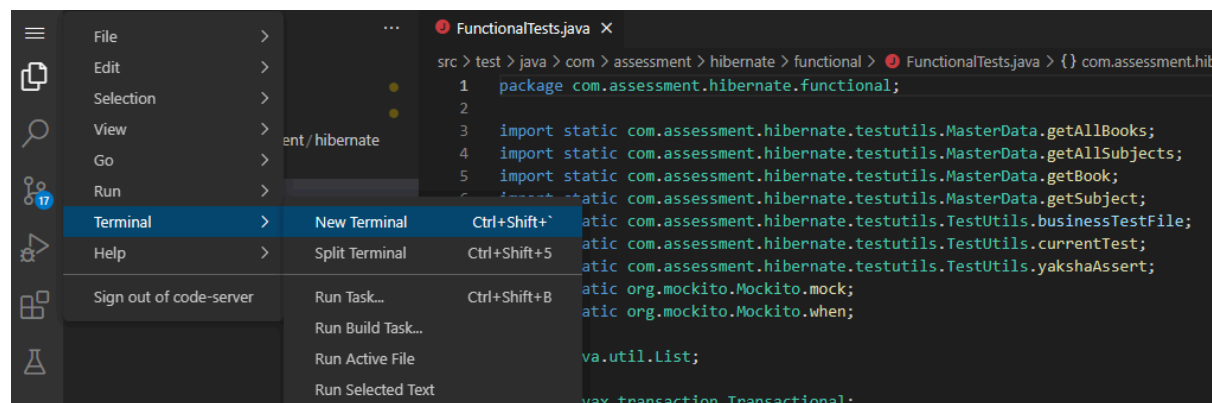9. **To run your project use command:**

   mvn clean install exec:java

   -Dexec.mainClass="com.playersselectionapplication.PlayersSelectionApplication"
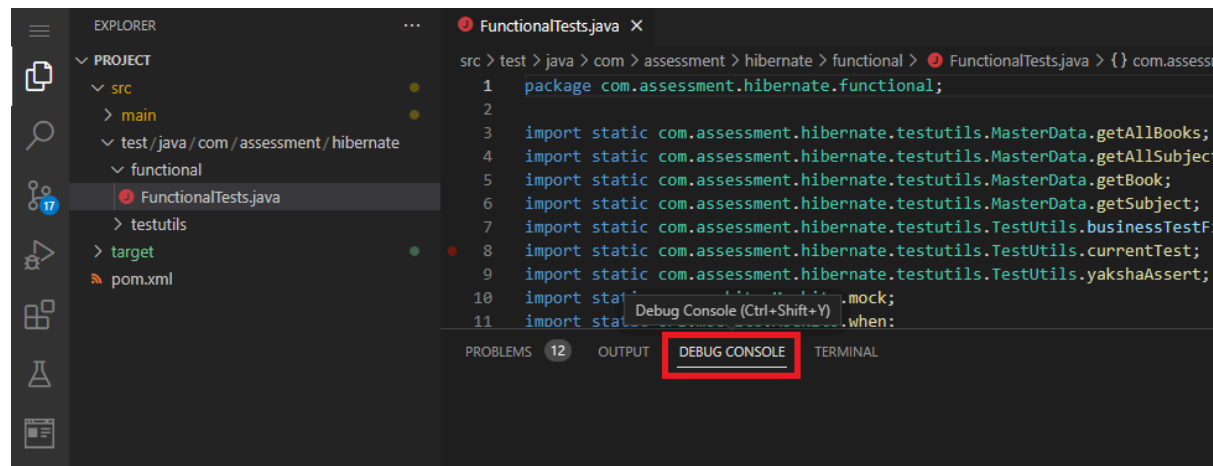
10. **To test your project, use the command**

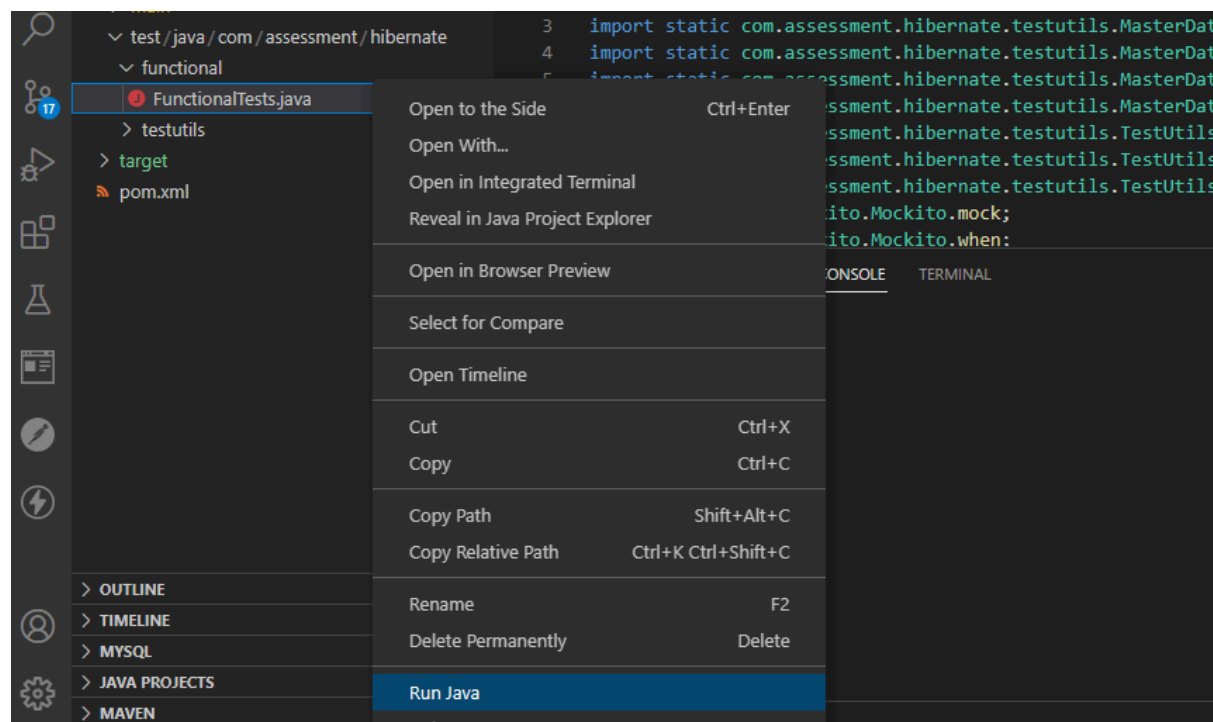   a. **Open FunctionalTests.java file in editor**

   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