Amazon E-Gift Card Application

IIHT

Time To Complete: 3 hrs

Contents

1 Problem Statement

The Amazon E-Gift Card Application provides users with the capability to perform not only CRUD (Create, Read, Update, Delete) operations and search functionalities in different criterias on e-gift cards, payments, and user profiles but also analytical operations like getting suggestion as per user's last purchase, getting % of all redeemed gift cards shared by user, grouping the cards by amount and many more for analysis and 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 new user 2) update a user 3) get details of a user 4) create a new egiftcard 5) update a egiftcard 6) get details of a egiftcard 7) create a new payment 8) update a payment 9) get details of a payment 10) get suggestions for egiftcards for user 11) get shared giftCards by user 12) get redeemed giftCard percentage 13) get giftCards grouped by amount 14) search eGift cards 15) exit

3 Classes & Method Descriptions

EGiftCardApplication Class

1. showOptions()

- **Task**: Displays the menu and takes user input.

- **Functionality**: Shows a list of numbered options 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**: This method displays a console menu and reads the user's input, returning the selected option.

2. createUser()

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

- **Functionality**: Prompts for user's name, age, and gender, creates a `User` object, and calls `userDAO.createUser()` to insert into the database.

- **Return Value**: void

- **Explanation**: Adds a new user and prints 'User created successfully.' upon success.

3. updateUser()

- **Task**: Updates an existing user's information.

- **Functionality**: Prompts for user ID, fetches the user, and allows input of updated values, which are then saved via `userDAO.updateUser()`.

- **Return Value**: void

- **Explanation**: Updates user details in the database and prints 'User updated successfully.' or 'User not found with ID'.

4. deleteUser()

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

- **Functionality**: Prompts for user ID, checks existence, and deletes via `userDAO.deleteUser()`.

- **Return Value**: void

- **Explanation**: Deletes the user and prints 'User deleted successfully.' or 'User not found with ID'.

5. getUserDetails()

- **Task**: Fetches and displays details of a user by ID.

- **Functionality**: Prompts for user ID, retrieves the user using `userDAO.getUserById()`, and displays details.

- **Return Value**: void

- **Explanation**: Displays user details or prints 'User not found with ID' if not found.

6. getAllUsers()

- **Task**: Displays all users in the system.

- **Functionality**: Calls `userDAO.getAllUsers()` to retrieve and print all users.

- **Return Value**: void

- **Explanation**: Prints a list of users or 'No users found.' if list is empty.

7. createEGiftCard()

- **Task**: Creates a new eGift Card.

- **Functionality**: Prompts for card details, creates an `EGiftCard` object, and saves using `giftCardDAO.createEGiftCard()`.

- **Return Value**: void

- **Explanation**: Adds a new gift card and prints 'eGift Card created successfully.'

8. updateEGiftCard()

- **Task**: Updates an existing eGift Card.

- **Functionality**: Prompts for card ID, fetches and updates card details, and saves via `giftCardDAO.updateEGiftCard()`.

- **Return Value**: void

- **Explanation**: Updates card and prints 'eGift Card updated successfully.' or 'eGift Card not found with ID'.

9. deleteEGiftCard()

- **Task**: Deletes an eGift Card.

- **Functionality**: Prompts for card ID, checks if card exists, and deletes using `giftCardDAO.deleteEGiftCard()`.

- **Return Value**: void

- **Explanation**: Deletes the card and prints 'eGift Card deleted successfully.' or 'eGift Card not found with ID'.

10. getEGiftCardDetails()

- **Task**: Displays details of a specific eGift Card by ID.

- **Functionality**: Prompts for card ID and fetches details using `giftCardDAO.getEGiftCardById()`.

- **Return Value**: void

- **Explanation**: Displays card details or prints 'eGift Card not found with ID'.

11. searchEGiftCards()

- **Task**: Searches eGift Cards by name, code, or amount.

- **Functionality**: Prompts for a keyword and searches using `giftCardDAO.searchEGiftCards(keyword)`.

- **Return Value**: void

- **Explanation**: Displays matching cards or prints 'No eGift Cards found.'

12. createPayment()

- **Task**: Creates a new payment record.

- **Functionality**: Prompts for user ID, card ID, and payment method, creates a `Payment` object, and saves using `paymentDAO.createPayment()`.

- **Return Value**: void

- **Explanation**: Adds a new payment and prints 'Payment created successfully.'

13. updatePayment()

- **Task**: Updates an existing payment record.

- **Functionality**: Prompts for payment ID, updates values, and saves using `paymentDAO.updatePayment()`.

- **Return Value**: void

- **Explanation**: Updates payment or prints 'Payment not found with ID'.

14. deletePayment()

- **Task**: Deletes a payment record.

- **Functionality**: Prompts for payment ID, checks if it exists, and deletes using `paymentDAO.deletePayment()`.

- **Return Value**: void

- **Explanation**: Deletes payment and prints 'Payment deleted successfully.' or 'Payment not found with ID'.

15. getPaymentDetails()

- **Task**: Displays payment details by ID.

- **Functionality**: Prompts for payment ID and displays details if exists.

- **Return Value**: void

- **Explanation**: Displays payment info or prints 'Payment not found with ID'.

16. getAllPayments()

- **Task**: Displays all payments in the system.

- **Functionality**: Retrieves all payments using `paymentDAO.getAllPayments()` and displays them.

- **Return Value**: void

- **Explanation**: Prints list of payments or 'No payments found.' if none.

17. getEGiftCardSuggestions()

- **Task**: Fetches gift card suggestions for a user.

- **Functionality**: Prompts for user ID and fetches suggestions using `userDAO.getSuggestionsForUser(userId)`.

- **Return Value**: void

- **Explanation**: Displays suggested cards or prints 'No eGift card suggestions found for user ID'.

18. showGiftCardsGroupedByAmount()

- **Task**: Displays gift cards grouped by their amount.

- **Functionality**: Fetches gift cards using `paymentDAO.getGiftCardsGroupedByAmount()` and groups them by amount.

- **Return Value**: void

- **Explanation**: Displays grouped gift cards or 'No gift cards found.'

19. showSharedGiftCardsByUser()

- **Task**: Displays gift cards shared by a specific user.

- **Functionality**: Prompts for user ID and fetches using `userDAO.getSharedGiftCardsByUser(userId)`.

- **Return Value**: void

- **Explanation**: Displays shared cards or 'No shared gift cards found for user ID'.

20. showRedeemedGiftCardPercentage()

- **Task**: Displays percentage of redeemed cards for a user.

- **Functionality**: Prompts for user ID and calculates using `userDAO.getRedeemedGiftCardPercentage(userId)`.

- **Return Value**: void

- **Explanation**: Prints percentage or 'No redeemed gift cards found for user ID'.

3.1 Model Classes with Annotations Guidance

3.1.1 EGiftCard Class

The **EGiftCard** class represents the gift card entity in the application. Below are the field annotations and their descriptions. This class is annotated as an entity and maps to the table named **"egift_cards"**.

1. **id field (Primary Key)**:
   The **id** field is the **primary key** for the **EGiftCard** entity. It should be auto-generated by the database using the **IDENTITY** strategy.

2. **name field**:
   Represents the **name** of the eGift card. This field is mapped to the **name** column in the database.

3. **code field**:
   Represents the **unique code** associated with the eGift card. It is stored in the **code** column.

4. **amount field**:
   Stores the **monetary value** of the eGift card. It is mapped to the **amount** column.

5. **message field**:
   Represents the **message** attached to the eGift card. Mapped to the **message** column.

6. **isRedeemed field**:
   A **boolean flag** indicating whether the eGift card has been **redeemed** or not. Stored in the **is_redeemed** column.

3.1.2 Payment Class

The **Payment** class represents a payment transaction made by a user for an eGift card. This class is annotated as an entity and maps to the **"payments"** table.

1. **id field (Primary Key)**:
   The **id** field serves as the **primary key** for the **Payment** entity. It is auto-generated by the database using the **IDENTITY** strategy.

2. **userId field**:
   Represents the **ID of the user** who made the payment. This field is mapped to the **user_id** column and establishes a reference to the user.

3. **cardId field**:
   Represents the **ID of the eGift card** for which the payment was made. This field is mapped to the **card_id** column.

4. **paymentMethod field**:
   Indicates the **method of payment** (e.g., credit card, PayPal). Mapped to the **payment_method** column.

3.1.3 User Class

The **User** class represents a user entity in the application. This class is annotated as an entity and maps to the **"users"** table.

1. **id field (Primary Key)**:
   The **id** field serves as the **primary key** for the **User** entity. It is auto-generated by the database using the **IDENTITY** strategy.

2. **name field**:
   Represents the **name** of the user. This field is mapped to the **name** column in the database.

3. **age field**:
   Stores the **age** of the user. Mapped to the **age** column.

4. **gender field**:
   Represents the **gender** of the user. This field is mapped to the **gender** column.

3.2 DAOImpl Classes and Method Descriptions

3.2.1 EGiftCardDAOImpl Class

1. createEGiftCard(EGiftCard giftCard)

- **Task**: Adds a new eGift card to the database.
  - **Functionality**: Executes an SQL `INSERT` query to insert the eGift card details into the `egift_cards` table.
  - **Return Value**: void
  - **Explanation**: This method saves the provided `EGiftCard` object into the database.

2. updateEGiftCard(EGiftCard giftCard)

- **Task**: Updates an existing eGift card in the database.
  - **Functionality**: Executes an SQL `UPDATE` query to update eGift card details in the `egift_cards` table.
  - **Return Value**: void
  - **Explanation**: This method updates the eGift card data based on its ID.

3. getEGiftCardById(int cardId)

- **Task**: Retrieves an eGift card by its ID.
  - **Functionality**: Executes a `SELECT` query to fetch eGift card details by the given ID.
  - **Return Value**: `EGiftCard`
  - **Explanation**: This method returns the eGift card object if it exists in the database, otherwise returns `null`.

4. searchEGiftCards(String keyword)

- **Task**: Searches eGift cards by name, code, or amount.
    - **Functionality**: Executes an SQL `SELECT` query with `LIKE` and `=` conditions to match the keyword with `name`, `code`, or `amount`.
    - **Return Value**: List of `EGiftCard`
    - **Explanation**: Returns a list of matching eGift cards from the database.

5. deleteEGiftCard(EGiftCard giftCard)

- **Task**: Deletes an eGift card by ID.
    - **Functionality**: Executes an SQL `DELETE` query for the eGift card record based on the given ID.
    - **Return Value**: void
    - **Explanation**: This method removes the eGift card from the database.

6. deleteAllEGiftCards()

- **Task**: Deletes all eGift cards from the database.
    - **Functionality**: Executes an SQL `DELETE` query without a `WHERE` clause to clear the table.
    - **Return Value**: void
    - **Explanation**: This method removes all eGift cards from the `egift_cards` table.

7. getAllEGiftCards()

- **Task**: Retrieves all eGift cards.
    - **Functionality**: Executes an SQL `SELECT` query to fetch all rows from the `egift_cards` table.
    - **Return Value**: List of `EGiftCard`
    - **Explanation**: Returns a list containing all eGift card objects present in the database.

3.2.2 PaymentDAOImpl Class

1. createPayment(Payment payment)

- **Task**: Adds a new payment to the database.
    - **Functionality**: Executes an SQL `INSERT` query to insert the payment details into the `payments` table.
    - **Return Value**: void
    - **Explanation**: This method saves the given `Payment` object to the database.

2. updatePayment(Payment payment)

- **Task**: Updates an existing payment record.
  - **Functionality**: Executes an SQL `UPDATE` query to update payment information using the payment's ID.
  - **Return Value**: void
  - **Explanation**: This method updates payment data in the database based on the provided payment ID.

3. getPaymentById(int paymentId)

- **Task**: Fetches a payment by its ID.
  - **Functionality**: Executes an SQL `SELECT` query to retrieve a payment from the `payments` table using the given ID.
  - **Return Value**: `Payment`
  - **Explanation**: Returns a `Payment` object if a matching record exists, otherwise returns `null`.

4. getAllPayments()

- **Task**: Retrieves all payment records from the database.
  - **Functionality**: Executes an SQL `SELECT` query to fetch all payment records from the `payments` table.
  - **Return Value**: List of `Payment`
  - **Explanation**: Returns a list of all `Payment` objects in the database.

5. deletePayment(Payment payment)

- **Task**: Deletes a payment record based on its ID.
  - **Functionality**: Executes an SQL `DELETE` query to remove the payment entry from the `payments` table.
  - **Return Value**: void
  - **Explanation**: This method deletes the specified payment from the database.

6. deleteAllPayments()

- **Task**: Deletes all payments from the database.
  - **Functionality**: Executes an SQL `DELETE` query to remove all entries in the `payments` table.
  - **Return Value**: void
  - **Explanation**: This method clears all data from the `payments` table.

7. getGiftCardsGroupedByAmount()

- **Task**: Retrieves eGift cards ordered and grouped by amount.
  - **Functionality**: Executes an SQL `SELECT` query on the `egift_cards` table ordered by amount, then groups them using a map based on the amount.
  - **Return Value**: List of `EGiftCard`

- **Explanation**: Returns all eGift cards grouped internally by amount, useful for categorization or reporting.

3.2.3 UserDAOImpl Class

1. createUser(User user)

- **Task**: Adds a new user to the database.
  - **Functionality**: Executes an SQL `INSERT` query to store the user's name, age, and gender into the `users` table.
  - **Return Value**: void
  - **Explanation**: This method inserts the given `User` object into the database.

2. updateUser(User user)

- **Task**: Updates an existing user record in the database.
  - **Functionality**: Executes an SQL `UPDATE` query using the user's ID to modify their name, age, and gender.
  - **Return Value**: void
  - **Explanation**: Updates user details such as name, age, and gender in the database.

3. deleteUser(User user)

- **Task**: Deletes a user from the database.
  - **Functionality**: Executes an SQL `DELETE` query to remove a user using their ID.
  - **Return Value**: void
  - **Explanation**: This method removes the specified `User` from the `users` table.

4. deleteAllUsers()

- **Task**: Deletes all user records from the database.
  - **Functionality**: Executes a `DELETE` query without a `WHERE` clause to remove all users.
  - **Return Value**: void
  - **Explanation**: Clears all user records from the `users` table.

5. getUserById(int userId)

- **Task**: Retrieves a user by their ID.
  - **Functionality**: Executes a `SELECT` query to fetch a user record by ID.
  - **Return Value**: `User`
  - **Explanation**: Returns a `User` object if found, otherwise returns `null`.

6. getAllUsers()

- **Task**: Fetches all users from the database.

- **Functionality**: Executes a `SELECT` query to retrieve all records from the `users` table.
- **Return Value**: List of `User`
- **Explanation**: Returns a list containing all `User` objects in the database.

7. getSuggestionsForUser(int userId)

- **Task**: Retrieves suggested eGift cards based on user's recent purchases.
  - **Functionality**: Executes a SQL query joining `egift_cards`, `payments`, and `users`, filtering by the given user ID, sorted by recent purchases, and limited to 3 results.
  - **Return Value**: List of `EGiftCard`
  - **Explanation**: Returns the top 3 most recent eGift cards purchased by the specified user.

8. getSharedGiftCardsByUser(int userId)

- **Task**: Fetches gift cards shared by a specific user.
  - **Functionality**: Executes a SQL join between `egift_cards` and `payments` filtered by `user_id`.
  - **Return Value**: List of `EGiftCard`
  - **Explanation**: Returns all eGift cards associated with the given user ID via payments.

9. getRedeemedGiftCardPercentage(int userId)

- **Task**: Calculates the percentage of redeemed gift cards shared by the user.
  - **Functionality**: Executes a SQL query that counts total and redeemed gift cards associated with the user's payments.
  - **Return Value**: double
  - **Explanation**: Returns the percentage of redeemed eGift cards shared by the user (0.0 if none found).

4 Implementation/Functional Requirements

4.1 Code Quality/Optimizations

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

4.2 Template Code Structure

Package: com.amazonegiftcardapplication

Resources

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

Package: com.amazonegiftcardapplication.model

Resources

Class/Interface	Description	Status
EGiftCard.java(class)	This represents entity class for EGiftCard	Partially Implemented
Payment.java(class)	This represents entity class for Payment	Partially Implemented
User.java(class)	This represents entity class for User	Partially Implemented

Package: com.amazonegiftcardapplication.repository

Resources

Class/Interface	Description	Status
EGiftCardDAO.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
EGiftCardDAOImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to written by test taker	Partially Implemented
PaymentDAO.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
PaymentDAOImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to written by test taker	Partially Implemented
UserDAO.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
UserDAOImpl.java(class)	This is an implementation class for DAO methods. Contains empty	Partially Implemented

	method bodies, where logic needs to written by test taker	

5 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.

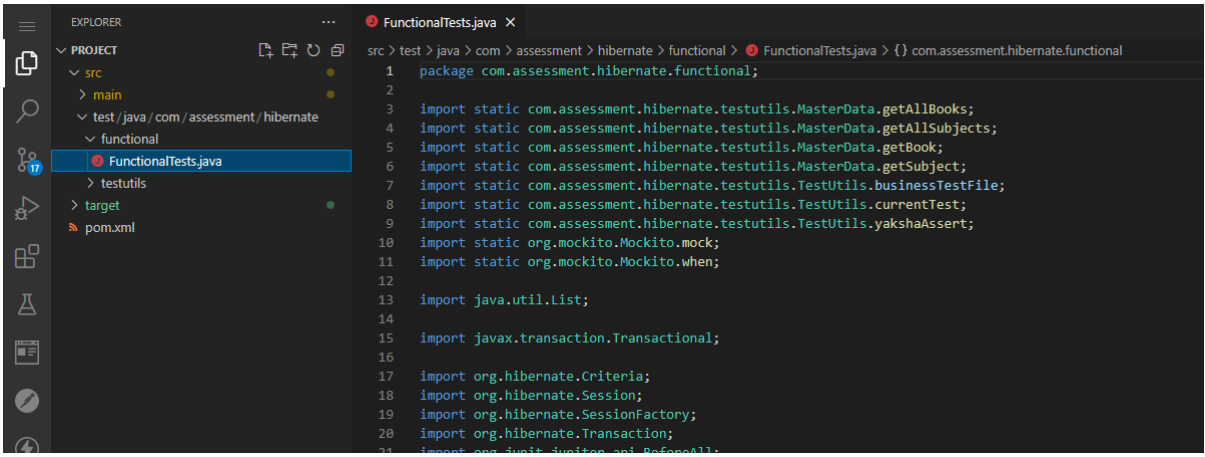
9. **To run your project use command:**

   mvn clean install exec:java

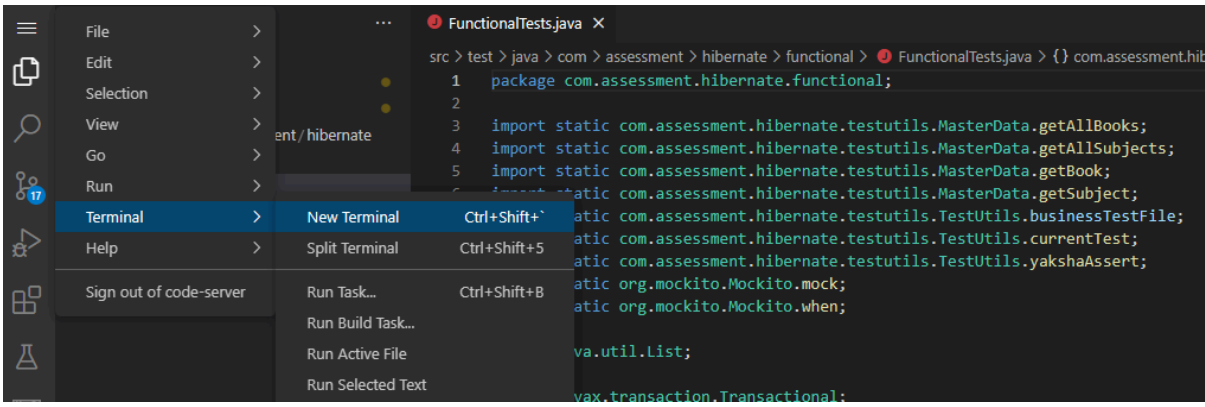
   -Dexec.mainClass="com.amazonegiftcardapplication.EGiftCardApplication"

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

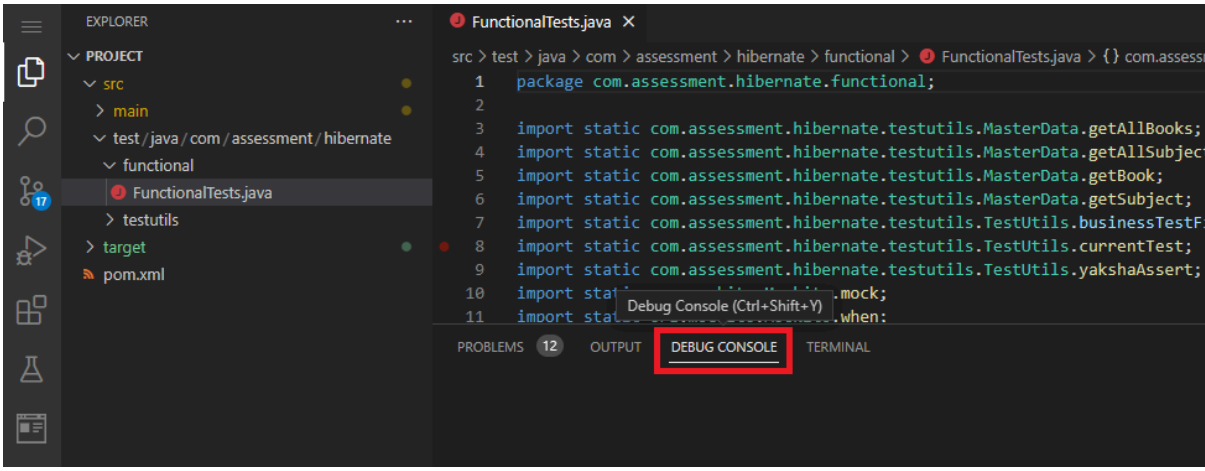
    a. **Open FunctionalTests.java file in editor**



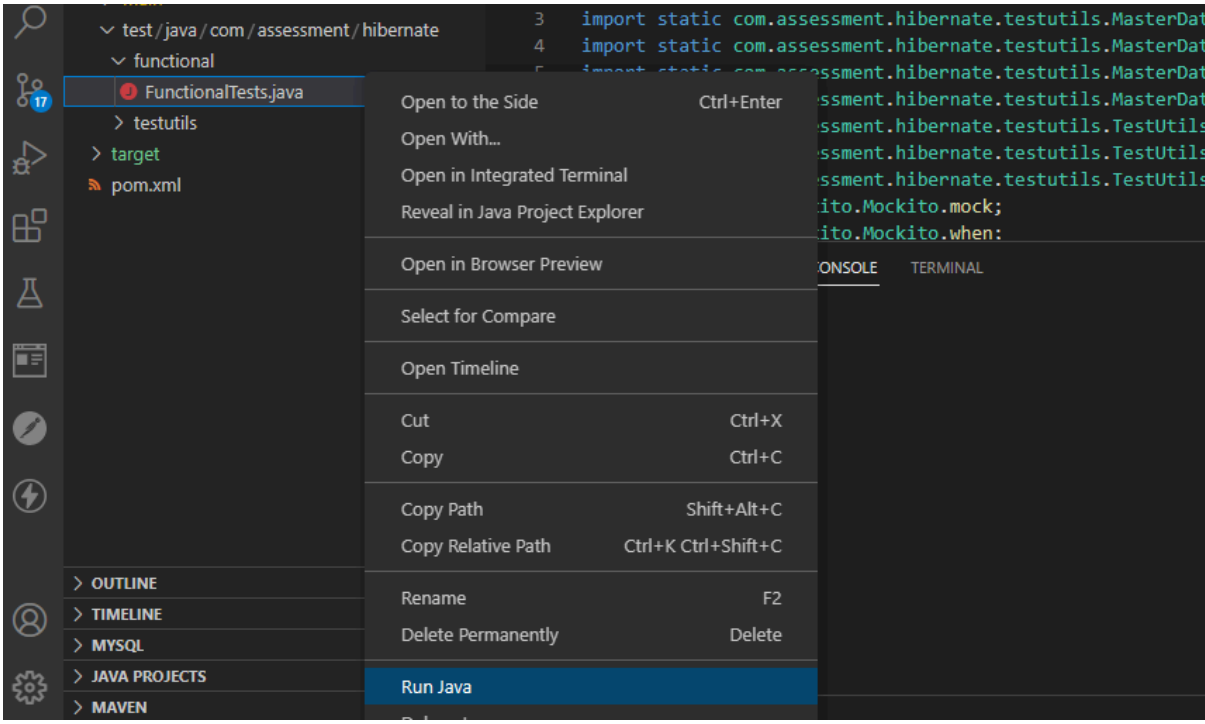
    b. **Open a new Terminal**



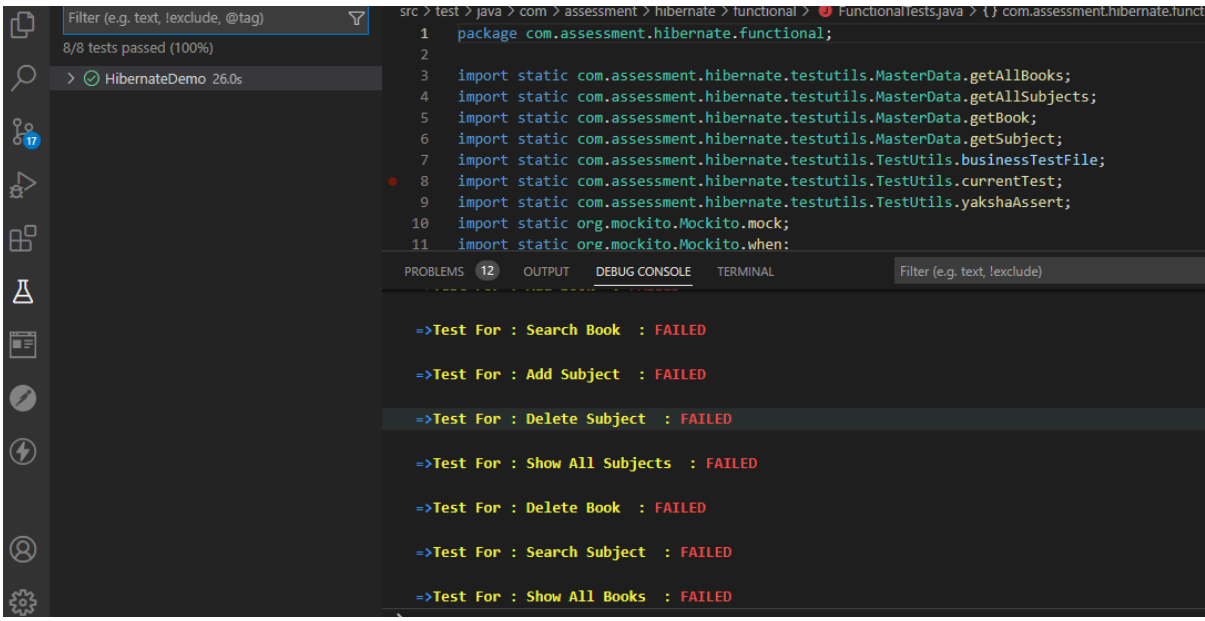
    c. **Go to Debug Console Tab**



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


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
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;
PROBLEMS 12
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, !exclude)
=>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