Amazon Kindle Analyser

IIHT

Time To Complete: 3 hrs

Contents

1 Problem Statement

The Amazon Kindle Analyzer Application allows users to perform not only CRUD (Create, Read, Update, Delete) operations and search functionalities in different criterias on books, shelves, user profiles, and user ratings but also provides the analytical operations like getting suggested shelves as per user's genre, get list of all trending books, get list of all purchased books 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 book 5) update a book 6) get details of a book 7) create a new shelf 8) update a shelf 9) get details of a shelf 10) create user rating 11) search for a book 12) get books suggestions by genre 13) get insights of most read books by age 14) get book suggestions by rating 15) show trending books 16) get list of purchased books 17) get percentage of books purchased after reading their sample 18) exit

3 Implementation/Functional Requirements

AmazonKindleAnalyserApplication 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(Scanner scanner)

- Task: Creates a new user in the system.

- Functionality: Prompts the user to input personal details and favorite genre/writer, then saves the user using userDAO.create().

- Return Value: void

- Explanation: Prints 'User created successfully!' after storing the user.

3. updateUser(Scanner scanner)

- Task: Updates user information.

- Functionality: Takes the user ID, fetches the record, collects updated data, and calls userDAO.update().

- Return Value: void

- Explanation: Prints 'User updated successfully!' if update is successful or 'User not found.' if user doesn’t exist.

4. getUserDetails(Scanner scanner)

- Task: Fetches user details by ID.

- Functionality: Prompts for user ID and retrieves the user using userDAO.getById().

- Return Value: void

- Explanation: Prints user details or 'User not found.'

5. createBook(Scanner scanner)

- Task: Creates a new book entry.

- Functionality: Collects book metadata from the user and inserts it into the database via bookDAO.create().

- Return Value: void

- Explanation: Prints 'Book created successfully!'

6. updateBook(Scanner scanner)

- Task: Updates book details.

- Functionality: Takes book ID, verifies existence, and updates the record via bookDAO.update().

- Return Value: void

- Explanation: Prints 'Book updated successfully!' or 'Book not found.'

7. getBookDetails(Scanner scanner)

- Task: Retrieves book details by ID.

- Functionality: Fetches book info from the database using bookDAO.getById().

- Return Value: void

- Explanation: Prints book details or 'Book not found.'

8. createShelf(Scanner scanner)

- Task: Creates a new shelf entry.

- Functionality: Captures shelf info including user ID, book ID, description, and purchase status, then calls shelfDAO.create().

- Return Value: void

- Explanation: Prints 'Shelf created successfully!'

9. updateShelf(Scanner scanner)

- Task: Updates shelf data.

- Functionality: Fetches shelf by ID, allows updates, and saves changes via shelfDAO.update().

- Return Value: void

- Explanation: Prints 'Shelf updated successfully!' or 'Shelf not found.'

10. getShelfDetails(Scanner scanner)

- Task: Displays shelf details.

- Functionality: Retrieves shelf by ID using shelfDAO.getById().

- Return Value: void

- Explanation: Prints shelf details or 'Shelf not found.'

11. createUserRating(Scanner scanner)

- Task: Creates a new user rating for a book.

- Functionality: Accepts user ID, book ID, and rating input and stores it using userRatingDAO.create().

- Return Value: void

- Explanation: Prints 'User rating created successfully!'

12. searchBooks(Scanner scanner)

- Task: Searches books by keyword.

- Functionality: Prompts for a keyword and retrieves books by name, writer, or genre via bookDAO.search().

- Return Value: void

- Explanation: Displays search results.

13. getBookSuggestionsByGenre(Scanner scanner)

- Task: Recommends books based on user’s favorite genre.

- Functionality: Uses shelfDAO.getSuggestionsByGenre(userId) to fetch relevant books.

- Return Value: void

- Explanation: Displays book suggestions by genre.

14. getMostReadBooksByAge(Scanner scanner)

- Task: Finds books popular among users of a certain age.

- Functionality: Takes an age input and calls bookDAO.getMostReadBooksByAge().

- Return Value: void

- Explanation: Prints the list of most read books for the given age.

15. getBookSuggestionsByRating(Scanner scanner)

- Task: Suggests books based on user ratings.

  - Functionality: Uses shelfDAO.getSuggestionsByRating(userId) to fetch highly rated books.

  - Return Value: void

  - Explanation: Displays recommended books by rating.

16. showTrendingBooks()

- Task: Displays trending books.

  - Functionality: Fetches trending books via bookDAO.getTrendingBooks().

  - Return Value: void

  - Explanation: Prints list of trending books.

17. showPurchasedBooks()

- Task: Lists all purchased books.

  - Functionality: Retrieves purchased books via bookDAO.getPurchasedBooks().

  - Return Value: void

  - Explanation: Displays purchased book details.

18. showPurchasePercentage()

- Task: Calculates and displays sample-to-purchase conversion percentage.

  - Functionality: Gets total unique books and those purchased after reading a sample, then calculates percentage.

  - Return Value: void

  - Explanation: Prints percentage or 'No books found.' if total count is zero.

3.1 Model Classes with Annotations Guidance

3.1.1 Book Class

The **Book** class represents the book 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 `"books"`.

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

2. **releaseYear field:**
   This field represents the year the book was released. It is mapped to the `release_year` column in the database.

3. **sampleAvailable field:**
   A boolean flag indicating whether a sample of the book is available for the user to read. It is stored in the `sample_available` column.

3.1.2 Shelf Class

The **Shelf** class represents the shelf 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 `"shelves"`.

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

2. **userId field:**
   Represents the ID of the user who owns this shelf entry. It is mapped to the `user_id` column in the database.

3. **bookId field:**
   Denotes the ID of the book associated with the shelf. This field is stored in the `book_id` column.

4. **description field:**
   A textual description associated with this shelf entry. It is mapped to the `description` column.

5. **isPurchased field:**
   A boolean flag indicating whether the book on the shelf has been purchased. Stored in the `is_purchased` column.

3.1.3 User Class

The **User** class represents the user 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 `"users"`.

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. **favouriteGenre field:**
   Indicates the user's favorite genre. It is mapped to the `favourite_genre` column in the database.

3. **favouriteWriter field:**
   Represents the user's favorite writer. This value is stored in the `favourite_writer` column.

3.1.4 UserRating Class

The **UserRating** class represents the rating provided by a user for a specific book. This class is annotated as an entity and maps to the table named `"user_ratings"`.

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

2. **userId field:**
   Represents the ID of the user who gave the rating. This field is mapped to the `user_id` column and serves as a foreign key referencing the `users` table.

3. **bookId field:**
   Indicates the ID of the book that is being rated. This field is mapped to the `book_id` column and serves as a foreign key referencing the `books` table.

3.2 DAOImpl Classes and Method Descriptions

3.2.1 BookDAOImpl Class

1. create(Book book)

Task: Adds a new book to the database.

Functionality: Executes an SQL INSERT query to insert book details into the books table.

Return Value: void

Explanation: This method saves the provided Book object into the database and sets the generated ID back to the object.

2. update(Book book)

Task: Updates an existing book record in the database.

Functionality: Executes an SQL UPDATE query to modify the book details based on the book ID.

Return Value: void

Explanation: Updates the fields of the existing book record in the books table.

3. getById(int id)

Task: Retrieves a book record by its ID.

Functionality: Executes an SQL SELECT query to fetch book details with the specified ID.

Return Value: Book

Explanation: Returns a Book object if found, otherwise returns null.

4. search(String keyword)

Task: Searches for books by name, writer, or genre.

Functionality: Executes an SQL SELECT query with LIKE conditions to match the keyword with name, writer, or genre.

Return Value: List<Book>

Explanation: Returns a list of books that match the search keyword criteria.

5. getTrendingBooks()

Task: Retrieves the most recent books by release year.

Functionality: Executes an SQL SELECT query to fetch books ordered by release_year in descending order, limited to 10.

Return Value: List<Book>

Explanation: Returns the 10 most recently released books from the books table.

6. getPurchasedBooks()

Task: Retrieves books that have been marked as purchased.

Functionality: Executes an SQL SELECT query joining books and shelves where is_purchased = true.

Return Value: List<Book>

Explanation: Returns a list of books that users have marked as purchased on their shelves.

7. getPurchasePercentage()

Task: Calculates the percentage of purchased books.

Functionality: Executes SQL queries to count total and purchased shelves, then computes the percentage.

Return Value: float

Explanation: Returns the percentage of shelves that contain purchased books.

8. getMostReadBooksByAge(int age)

Task: Fetches the most read books by users of a specific age.

Functionality: Executes an SQL SELECT query joining books, user_ratings, and users, grouped by book ID and filtered by user age.

Return Value: List<Book>

Explanation: Returns a list of books most frequently rated by users of the given age.

9. getUniqueBookCount()

Task: Counts the number of unique books.

Functionality: Executes an SQL SELECT COUNT(DISTINCT id) query on the books table.

Return Value: int

Explanation: Returns the total number of unique book entries in the database.

10. deleteAllBooks()

Task: Deletes all book records from the database.

Functionality: Executes an SQL DELETE query to remove all rows from the books table.

Return Value: void

Explanation: Completely clears the books table.

11. getAllBooks()

Task: Retrieves all books from the database.

Functionality: Executes an SQL SELECT * query on the books table.

Return Value: List<Book>

Explanation: Returns a list of all books currently stored in the database.

3.2.2 ShelfDAOImpl Class

1. create(Shelf shelf)

Task: Adds a new shelf entry into the database.

Functionality: Executes an INSERT SQL query into the shelves table.

Return Value: void

Explanation: Persists a new Shelf object into the database and assigns the generated ID.

2. update(Shelf shelf)

Task: Updates an existing shelf record.

Functionality: Executes an UPDATE SQL query for the shelves table based on shelf ID.

Return Value: void

Explanation: Modifies shelf details such as user ID, description, and purchase status.

3. getById(int id)

Task: Fetches a shelf record using its ID.

Functionality: Performs a SELECT SQL query using the given shelf ID.

Return Value: Shelf

Explanation: Returns the corresponding Shelf object if found, otherwise returns null.

4. getSuggestionsByGenre(int userId)

Task: Provides book suggestions for a user based on their favorite genre.

Functionality: SELECT query joining books and users tables using genre filter.

Return Value: List<Book>

Explanation: Fetches books from the database that match the user's favorite genre.

5. getSuggestionsByRating(int rating)

Task: Fetches books with user ratings equal to or greater than the provided value.

Functionality: Executes a JOIN query between books and user_ratings filtering by rating.

Return Value: List<Book>

Explanation: Returns books rated highly by users.

6. getMostReadBooksByAge(int age)

Task: Returns a list of most read books for a given age group.

Functionality: Joins books, users, and user_ratings to find the most read books grouped by book ID.

Return Value: List<Book>

Explanation: Books frequently rated by users of the given age.

7. getPurchasedBookCount()

Task: Returns the number of distinct purchased books.

Functionality: Performs a COUNT query on shelves where is_purchased is true.

Return Value: int

Explanation: Helps in calculating purchase metrics such as conversion rate.

8. deleteAllShelves()

Task: Deletes all shelf entries from the database.

Functionality: Executes a DELETE SQL query on the shelves table.

Return Value: void

Explanation: Removes all shelves records, useful during reset or cleanup.

9. getAllShelves()

Task: Retrieves all shelf records from the database.

Functionality: Executes a SELECT * SQL query on the shelves table.

Return Value: List<Shelf>

Explanation: Returns all Shelf entries for administrative or analytical purposes.

3.2.3 UserDAOImpl Class

1. create(User user)

Task: Adds a new user to the database.

Functionality: Executes an SQL INSERT query to store the user details in the users table.

Return Value: void

Explanation: This method inserts a new User object into the database and sets the generated ID.

2. update(User user)

Task: Updates an existing user in the database.

Functionality: Executes an SQL UPDATE query to modify user details in the users table.

Return Value: void

Explanation: This method updates the user's data in the database based on the provided ID.

3. getById(int id)

Task: Retrieves a user by ID.

Functionality: Executes a SELECT query to fetch user details using the given ID.

Return Value: User

Explanation: Returns the User object corresponding to the specified ID, or null if not found.

4. deleteAllUsers()

Task: Deletes all users from the database.

Functionality: Executes a DELETE query on the users table.

Return Value: void

Explanation: Removes all user records from the database.

5. getAllUsers()

Task: Retrieves all users from the database.

Functionality: Executes a SELECT query to fetch all user records.

Return Value: List<User>

Explanation: Returns a list of all users stored in the database.

3.2.4 UserRatingDAOImpl Class

1. create(UserRating userRating)

Task: Adds a new user rating to the database.

Functionality: Executes an SQL INSERT query to insert a user_id, book_id, and rating into the user_ratings table.

Return Value: void

Explanation: This method saves a new UserRating object in the database.

2. deleteAllUserRatings()

Task: Deletes all user ratings from the database.

Functionality: Executes a DELETE SQL query on the user_ratings table.

Return Value: void

Explanation: This method removes all existing user ratings from the database.

3. getById(int id)

Task: Retrieves a specific user rating by its ID.

Functionality: Executes a SELECT query using the provided ID.

Return Value: UserRating

Explanation: Returns a UserRating object that matches the given ID, or null if not found.

4. update(UserRating userRating)

Task: Updates a user rating in the database.

Functionality: Executes an UPDATE query using the ID of the UserRating to modify the data.

Return Value: void

Explanation: This method updates the details of a user rating (user_id, book_id, rating).

5. delete(UserRating userRating)

Task: Deletes a user rating from the database.

Functionality: Executes a DELETE SQL query using the ID from the given UserRating object.

Return Value: void

Explanation: This method deletes a specific user rating identified by its ID.

6. getAllUserRatings()

Task: Retrieves all user ratings from the database.

Functionality: Executes a SELECT query on the user_ratings table.

Return Value: List<UserRating>

Explanation: Returns a list of all UserRating entries from the database.

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

a. Package: com.amazonkindleanalyserapplication

Resources

Class/Interface	Description	Status
AmazonKindleAnalyserApplication.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.amazonkindleanalyserapplication.model

Resources

Class/Interface	Description	Status
Book.java(class)	This represents entity class for Book	Partially Implemented
Shelf.java(class)	This represents entity class for Shelf	Partially Implemented
User.java(class)	This represents entity class for User	Partially Implemented
UserRating.java(class)	This represents entity class for UserRating	Partially Implemented

c. Package: com.amazonkindleanalyserapplication.repository

Resources

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

	method bodies, where logic needs to written by test taker	
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 method bodies, where logic needs to written by test taker	Partially Implemented
UserRatingDAO.java(interface)	This is an interface containing declaration of DAO method	Already Implemented
UserRatingDAOImpl.java(class)	This is an implementation class for DAO methods. Contains empty method bodies, where logic needs to written by test taker	Partially Implemented

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

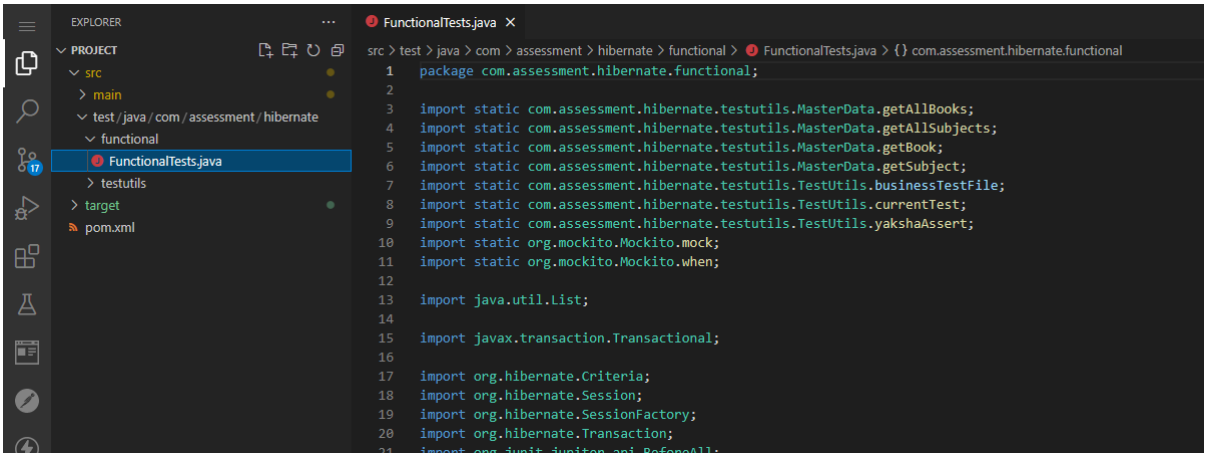
The last command will ask for password which is 'pass@word1'

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:

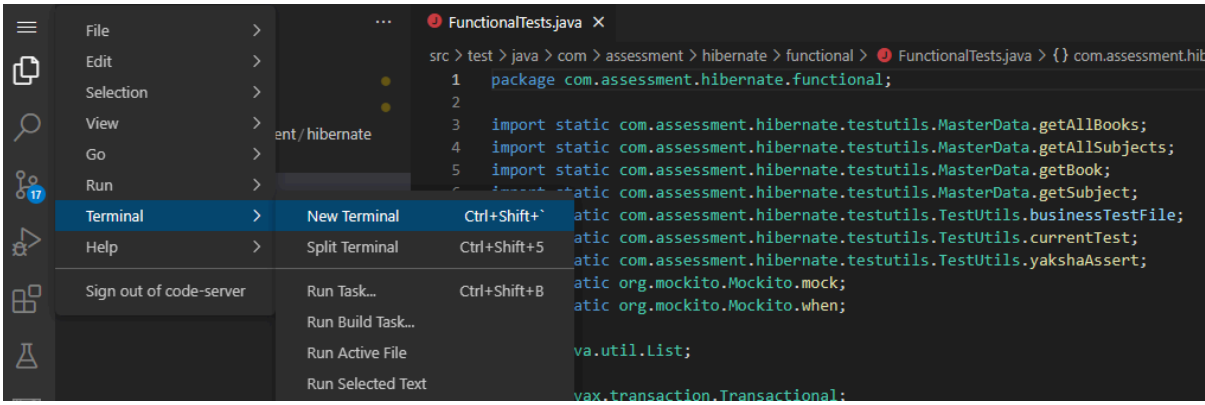
   mvn clean install exec:java -Dexec.mainClass="com.amazonkindleanalyserapplication.AmazonKindleAnalyserApplication"

10. To test your project, use the command

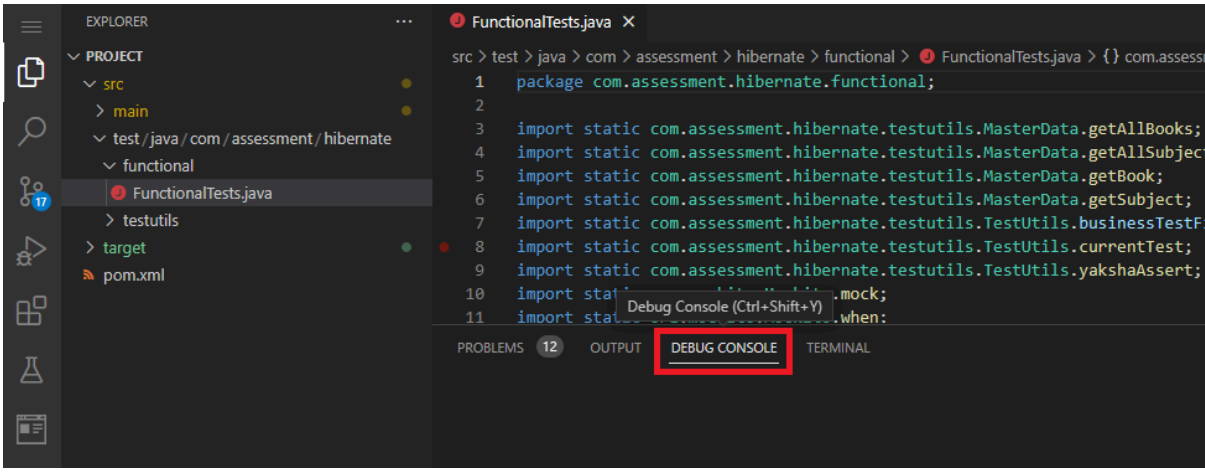
a. Open FunctionalTests.java file in editor



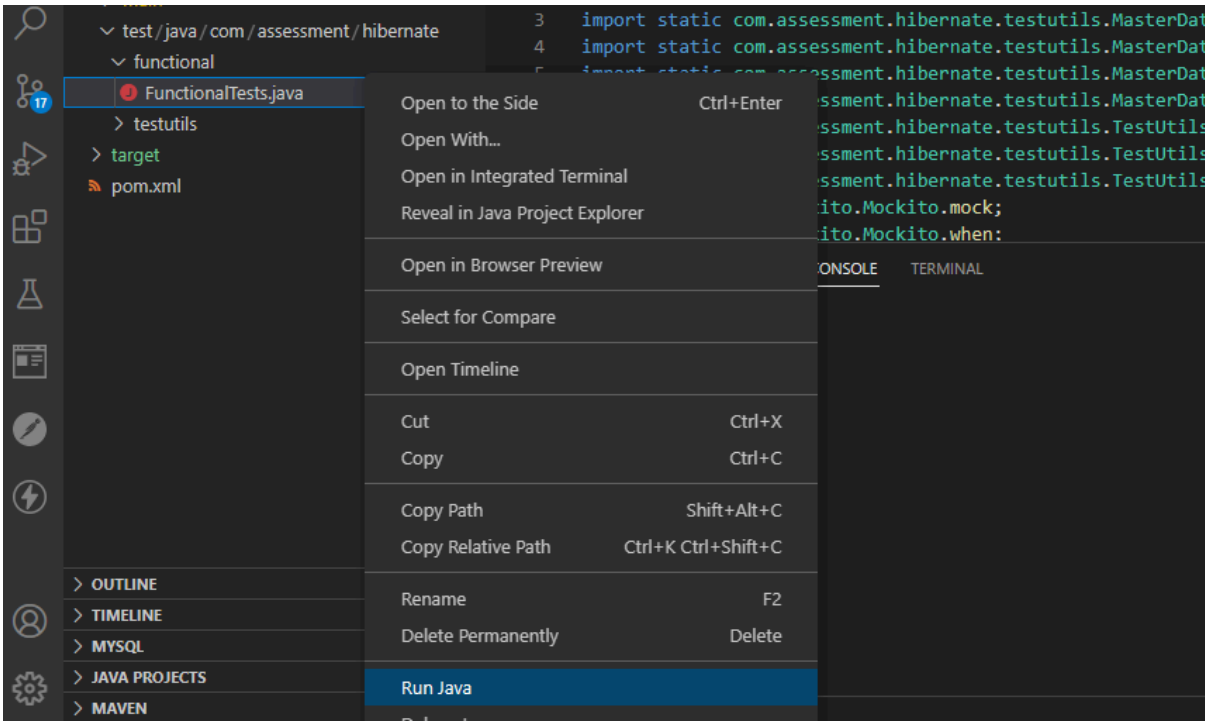
b. Open a new Terminal



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