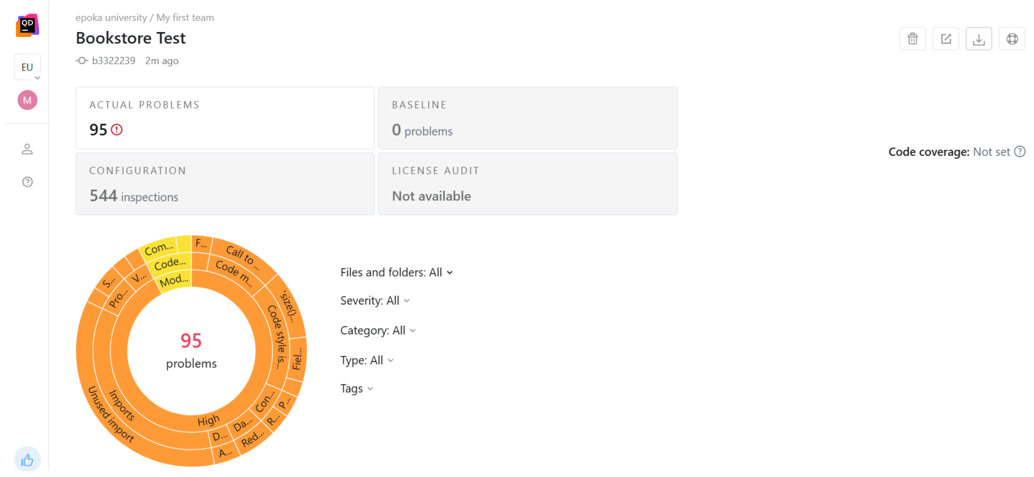
Software Quality Assurance and Testing

Project Name	Bookstore Testing
Members	Saadet Yilmaz
	Selma Aksoy
Github Link	https://github.com/Saadet2805/Bookstore_Testing.git

Part 1: Github Usage and Qodana



epoka university / My first team

Bookstore Test

b3322239 2m ago

ACTUAL PROBLEMS 95	BASELINE 0 problems
CONFIGURATION 544 inspections	LICENSE AUDIT Not available

Code coverage: Not set

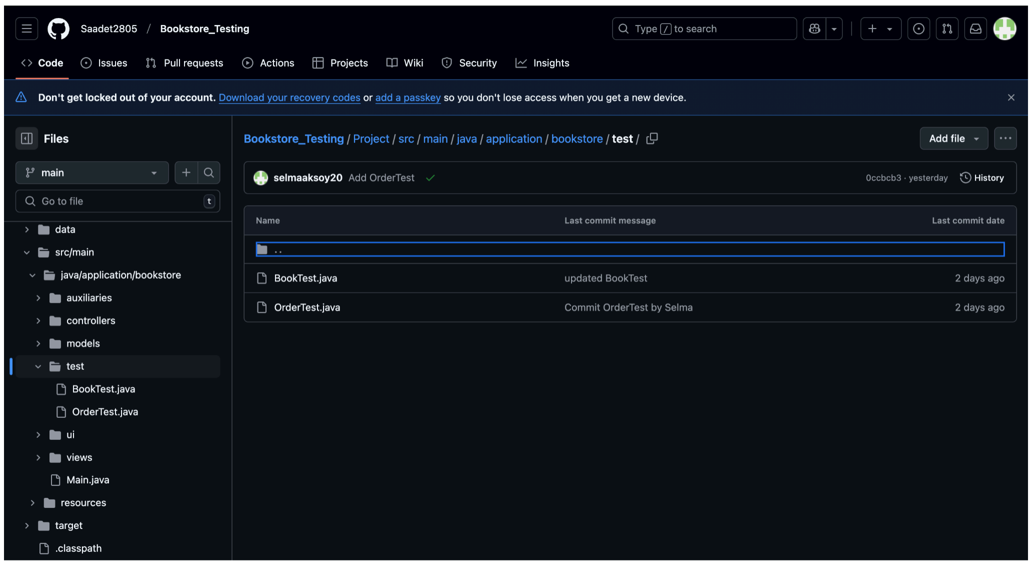
Files and folders: All

Severity: All

Category: All

Type: All

Tags



Saadet2805 / Bookstore_Testing

Type / to search

Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights

Don't get locked out of your account. Download your recovery codes or add a passkey so you don't lose access when you get a new device.

Files

main

Go to file

- data
- src/main
  - java/application/bookstore
    - auxiliaries
    - controllers
    - models
    - test
      - BookTest.java
      - OrderTest.java
    - ui
    - views
    - Main.java
  - resources
- target
- .classpath

Bookstore_Testing / Project / src / main / java / application / bookstore / test /

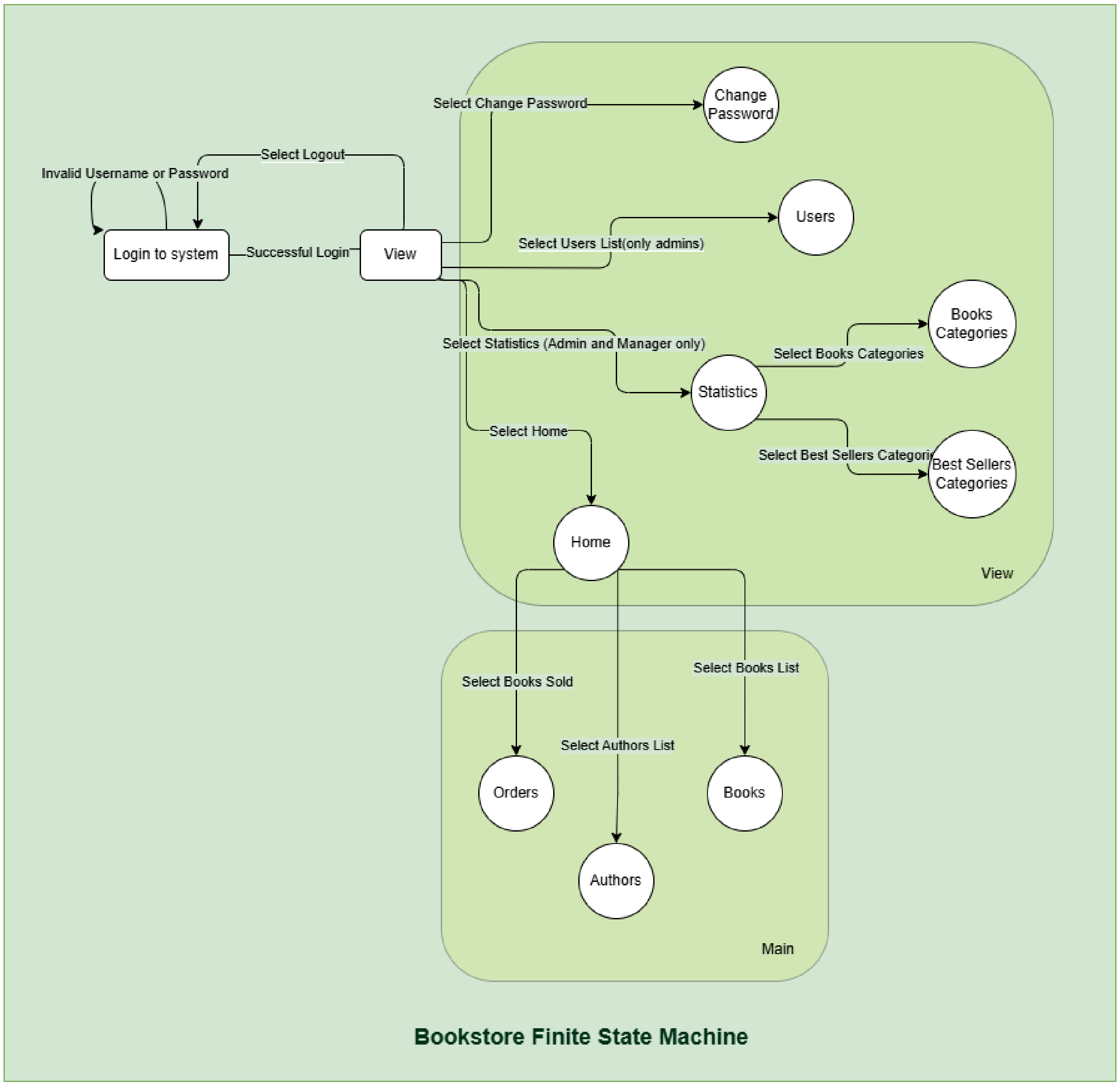
Add file

selmaaksoy20 Add OrderTest ✓ 0ccbcb3 · yesterday History

Name	Last commit message	Last commit date
..		
BookTest.java	updated BookTest	2 days ago
OrderTest.java	Commit OrderTest by Selma	2 days ago

Threads

The finite State Machine



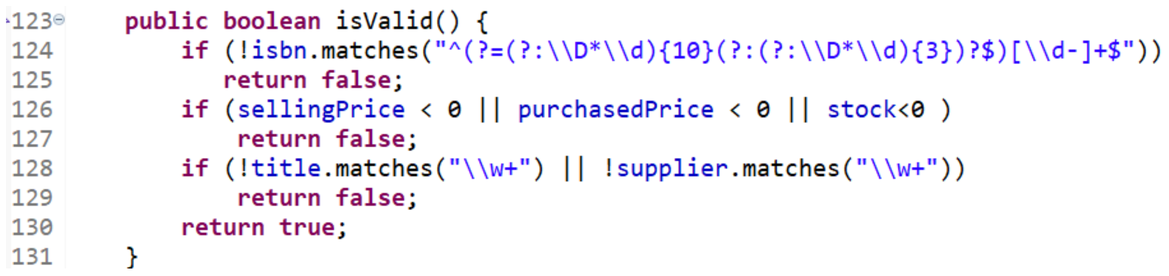
High Level Use Cases

Use Case	Description
Login to system with correct credentials	User enters correct Username and Password and is directed to the Home page (by default)
Login to system with wrong credentials	“Wrong Username or Password” is displayed
Click on Logout on tab bar	The user is redirected to the login view
Change Password	User enters new password and validates
Select Users (only for admin)	Display new tab on tab bar “Users”
Click on Users Tab	View the users list and edit them
Select Book Categories from Statistics	Display new tab on tab bar “Books”
Click on Books tab	A pie chart on book categories is displayed
Select Best Sellers Categories from Statistics	Display a new tab on tab bar “Best Sellers Categories”
Click on Best Sellers Categories tab	A pie chart on best sellers is displayed
Select home	Display new tab on tab bar “Home”
Click on Home tab	Three buttons are displayed: Books List, Authors List and Book Sold (Orders)
Select Books List	Display new tab on tab bar “Books”
Click on Books tab	View the list of books and edit
Select Authors List	Display new tab on tab bar “Authors”
Click on Authors tab	View the list of authors and edit
Select Books Sold	Display new tab on tab bar “Orders”
Click on Orders tab	View the Orders list and edit

Part 2: Testing Analysis

Unit Testing

Method 1 (Saadet Yılmaz):



```java
    public boolean isValid() {
        if (!isbn.matches("^(?=(?:\\D*\\d){10}(?:(?:\\D*\\d){3})?$)[\\d-]+$"))
           return false;
        if (sellingPrice < 0 || purchasedPrice < 0 || stock<0 )
            return false;
        if (!title.matches("\\w+") || !supplier.matches("\\w+"))
            return false;
        return true;
    }
```

Method Explanation:

The isValid() method of the Book class checks whether a book edited is valid or not according to its ISBN, which must be 10 to 13 digits, returns false if any price entered or stock is negative, and checks whether the supplier or the title contains any invalid characters.

Boundary Value Testing

Take the minimum, minimum+1, nominal value, maximum, maximum+1 boundary. For the ISBN, title and supplier, we test the boundaries according to the length (as it is stated in the regex), for the prices and the stock we check negative, minimum and nominal values - no upper boundary is mentioned.

BOUNDARY VALUE ANALYSIS			
	Invalid (min-1)	**Valid (Min, nominal, max)**	**Invalid (max+1)**
ISBN	9	10, 11, 13	14
sellingPrice	-0.01	0, 15	
purchasedPrice	-0.01	0, 10	
Stock	-1	0, 10	
Title	0	1, 11	
Supplier	0	1, 4	

Class Evaluation

Evaluate different combinations of the conditions.

Condition	**TC1**	**TC2**	**TC3**	**TC4**	**TC5**	**TC6**	**TC7**	**TC8**
ISBN is valid	True	False	True	True	False	False	True	False
purchasedPrice ≥ 0	True	True	False	True	False	True	False	False
sellingPrice ≥ 0	True	True	True	True	True	True	True	True
stock ≥ 0	True	True	True	True	True	True	True	True
title matches regex	True	True	True	False	True	False	False	False
supplier matches regex	True	True	True	False	True	False	False	False
Expected Output	True	False	False	False	False	False	False	False

Code Coverage Testing

Statement Coverage Testing

We have to make sure each statement is run at least once. The statements are lines: 125, 127, 129, 130. It is sufficient to test the cases when isbn is false, selling price is negative, title is not valid and when all are valid.

Condition	**TC1**	**TC2**	**TC3**	**TC4**
ISBN matches regex	True	False	True	True
sellingPrice < 0	True	True	False	True
Title matches regex	True	True	True	False
Output	**True**	**False**	**False**	**False**

$$Statement\ Coverage\ =\ \frac{Number\ of\ executed\ statements}{number\ of\ statements}$$

$$Statement\ Coverage\ =\ \frac{4}{4}\ =\ 100\%$$

Branch Coverage

We have to make sure each branch of every if statement is executed at least once. Once when the if statement is true and once when it is false.

Test Cases: All are valid, ISBN is not valid, sellingPrice is not valid, Title is not valid.

In this case, it is the same as the statement coverage. Although the outcome of one condition does not affect the outcomes of the other conditions (they are independent), the function ends with the first encounter of false.

Condition Coverage

- **Case 1:** All inputs satisfy isValid().
- **Case 2:** Invalid ISBN.
- **Case 3:** Negative sellingPrice.
- **Case 4:** Empty title.

There is only one case for the second and third condition as they are short-circuited. Let us calculate the coverage:

$$Condition\ Coverage\ = \frac{number\ of\ executed\ conditions}{number\ of\ conditions}$$

$$Condition\ Coverage\ = \frac{4}{4}\ =\ 100\%$$

MC/DC Coverage Testing

The first condition does not contribute to the MC/DC testing as it does not have a boolean operator that combines more than 1 condition such as || or &&. However, we can do the testing on the second and the third conditions as they contain an or operator.

- **Case 1:** All inputs satisfy isValid().
- **Case 3:** Negative sellingPrice.
- **Case 4:** Negative purchasedPrice.
- **Case 5:** Negative stock.
- **Case 6:** Empty title.
- **Case 7:** Empty supplier.

We see that for the second condition if any **one** of the three (sellingPrice, purchasedPrice, stock) is false, the outcome is directly false, the other two do not have an effect on the output. Same goes for the third condition.

Method 2 (Selma Aksoy):

```
public static float getTotal(float price, int quantity) throws Exception{
    if(price<0) {
        throw new Exception("Price cannot be negative");
    }
    else if(quantity<0) {
        throw new Exception("Quantity cannot be negative");
    }
    float sum=0;
    sum+= quantity * price;
    return sum;
}
```

Method Explanation :

The getTotal method calculates the total sum of the order based on its price and the quantity wanted . It also has conditions to check if the price and quantity are valid inputs(non negative).

- float price : represents the price of a single item
- int quantity : represents the number of items to be purchased
- Exception : method throws exception if price is negative ("Price cannot be negative") and if quantity is negative ("Quantity cannot be negative")

Boundary Value Testing

Tests methods behaviour when the inputs are valid but boundary conditions. First assertion tests the lower boundary where both price and quantity are zero. This case handles the condition where there are no costs or items. The second assertion tests a possible condition where the price and quantity are positive and within a reasonable range, this ensures that the price*quantity calculation performs correctly.

BOUNDARY VALUE ANALYSIS		
	Invalid (min-1)	**Valid (Min, min+1, nominal, max-1, max)**
Price	-1.0	0, 10
Quantity	-1	0, 10

Class Evaluation

The class evaluation tests the functionality of a method that calculates the total cost based on the price and quantity of an item.

- **Valid Cases**: Both price and quantity are positive, price or quantity is zero, and both are zero, ensuring accurate total calculation.

- **Invalid Cases**: Negative price, negative quantity, or both, validating that the method throws exceptions with clear error messages ("Price cannot be negative") and ("Quantoty cannot be negative")

Test Case	Price	Quantity	Expected Behavior	Expected Output
TC1: Valid Inputs - Positive price and quantity	10.0	5	Price and quantity are valid, compute total.	Total = 10 * 5 = 50
TC2: Valid Inputs - Price is zero, quantity positive	0.0	5	Price is zero, total should be zero	Total = 0
TC3: Valid Inputs - Price positive, quantity is zero	10.0	0	Quantity is zero, total should be zero.	Total = 0
TC4: Valid Inputs - Both price and quantity are zero	0.0	0	Both are zero, total should be zero.	Total = 0
TC5: Invalid Inputs - Negative price, positive quantity	-5.0	5	Negative price is invalid, should throw exception.	Exception: "Price cannot be negative"
TC6: Invalid Inputs - Positive price, negative quantity	10.0	-3	Negative quantity is invalid, should throw exception.	Exception: "Quantity cannot be negative"
TC7: Invalid Inputs - Both price and quantity negative	-10.0	-3	Both are negative, should throw exception for negative price first.	Exception: "Price cannot be negative"
TC8: Invalid Inputs - Price is zero, quantity negative	0.0	-3	Negative quantity is invalid, should throw exception.	Exception: "Quantity cannot be negative"

Code Coverage

Statement Coverage

We have to make sure each statement is run at least once:

- **Case 1:** Normal case
- **Case 2:** Negative price
- **Case 3:** Negative quantity

$$Statement\ Coverage\ =\ \frac{number\ of\ executed\ statements}{number\ of\ statements}\ =\ 100\%$$

Branch Coverage

For this method, the key decision points arise from the if and else if conditions that check the validity of the price and quantity parameters. In branch testing we tested all the possible branches/conditions for this method . All cases where price and quantity were valid and invalid were tested.

Price

Condition	Quantity	Test Case	Input	Expected Output
Valid (price >= 0)	Valid (>=0)	TC1	Price = 0.0, quantity = 0, price = 10.0, quantity = 10	0.0 100.0
Invalid (price < 0)	Valid (>=0)	TC2	Price = -1.0, quantity = 10	Exception with message “Price cannot be negative”

Quantity

Condition	Price	Test Case	Input	Expected Output
Valid (quantity >= 0)	Valid (>=0)	TC3	Price = 0.0, quantity = 0, price = 10.0, quantity = 10	0.0 100.0
Invalid (<0)	valid (>=0)	TC4	Price = 10.0, quantity = -5	Exception with message “Quantity cannot be negative”

Condition Coverage

For the condition coverage, we check all condition paths similar to the branch coverage.

MC/DC Coverage Testing

Method used for this testing is the Author.isValid() function.

```
public boolean isValid() {
    return getFirstName().length() > 0 && getLastName().length() > 0;
}
```

Condition	**True**	**False**	**Independently Affects Decision**
getFirstName().length() > 0	Yes (TC1, TC3)	Yes (TC2, TC4)	Yes
getLastName().length() > 0	Yes (TC1, TC2)	Yes (TC3, TC4)	Yes

Author.java - AuthorTest.java

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	toString	Ensure first and last names are concatenated correctly	firstName = "Selma", lastName = "Aksoy"	"SelmaAksoy"	PASS
TC-02	existsInList	Check if author exists in the list	Author(“Selma”, “Aksoy”)in list	true	PASS
TC-03	existsInList	Check if author does not exist in the list	Author(“Saadet”, “Yilmaz”) not in list	false	PASS
TC-04	getFullName	Validate full name format	firstName = "Selma", lastName = "Aksoy"	"Selma Aksoy"	PASS
TC-05	saveInFile	Save author to file	Author("Selma", "Aksoy")	true, Author added to static list	PASS
TC-06	deleteFromFile	Delete author from file	Author(“Selma”, “Aksoy”)exists	true,Author removed from static list	PASS
TC-07	getSearchResults	Search for an author by partial name	searchText = "Selma"	List containing Author(“Selma”, “Aksoy”)	PASS
TC-08	getSearchResults	No matching authors	searchText = "Nonexistent"	Empty list	PASS
TC-09	updateFile	Update file with current list	Author(“Selma”, “Aksoy”)added	True , File updated successfully	PASS

Order.java - OrderTest.java

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	saveInFile()	Test that an order is saved correctly to a file.	Order object with valid data	true	PASS
TC-02	print(PrintWriter writer)	Test if order details are printed correctly to the writer.and positive quantity.	Order object, PrintWriter (mocked)	Correct order details printed (order ID, client name, total, etc.)	PASS
TC-03	isValid()	Test that a valid order with a non-empty client name returns true.	clientName = "John Doe"	true	PASS
TC-04	isValid()	Test that an order with an empty client name returns false.	clientName = ""	false	PASS
TC-05	deleteFromFile()	Test that an order can be deleted from the file correctly.	Order object (saved previously)	true (order deleted)	PASS
TC-06	getTotal	Check if author exists in the list	Author(“Selma”, “Aksoy”)in list	true	PASS
TC-07	getTotal	Check if author does not exist in the list	Author(“Saadet”, “Yilmaz”) not in list	false	PASS

User.java - UserTest.java

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	ifusernameExists()	Verify if the method correctly checks for the existence of a username in the list.	username = "testUser" (added to the list), new User with username "newUser"	True if "testUser" exists,false if "newUser" doesn't exist	PASS
TC-02	getSearchResults()	Verify if the search method returns correct results based on a substring match in the username.	Search for "test" in list containing "Klodjan" and "Nikolin"	Results include both "Klodjan" and "Nikolin", size = 2	PASS
TC-03	getUser()	Verify if the method returns the correct user when it exists in the list.	User "testUser" added to the list, search for "testUser" and "nonUser"	Returns "testUser", null for "nonUser"	PASS
TC-04	isValid()	Verify if the method correctly checks the username and password format.	validUser = "validUser" with password "password123", invalidUser = "invalid user" with password "pass word"	Returns true for validUser, False f or invalidUser	PASS
TC-05	saveInFile()	Verify if the user is correctly saved in the file and added to the users' list.	user = "fileUser" added to the list and saved in file	Returns true for save operation, "fileUser" added to the list	PASS

Role.java - RoleTest.java

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	testRoleAssignment()	Verify that a Role can be assigned to a User and retrieved correctly.	User initialized with Role.Manager, then changed to Role.Librarian	Returns Role.Manager initially, then Role.Librarian after role change	PASS
TC-02	testRoleValues()	Ensure that all defined Role values are present and can be accessed.	Accessing all values of the Role enum	Returns 3 roles: MANAGER , LIBRARIAN , ADMIN	PASS

Category.java - CategoryTest.java

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	testFromStringWithValidCategory	Test converting a valid category string to enum value.	"Adventure"	Category.Adventure	PASS
TC-02	testFromStringWithValidCategory	Test converting a valid category string with case insensitivity.	"romance"	Category.Romance	PASS
TC-03	testFromStringWithInvalidCategory	Test invalid category input results in Category.Unknown.	"InvalidCategory"	Category.Unknown	PASS
TC-04	testFromStringWithNull	Test null input results in Category.Unknown	null	Category.Unknown	PASS

Integration Testing

Book.java - BookTest.java

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	mocktestUpdateFile	Ensure book details are updated when file is writable	Book("1234567890", "The Great Gatsby", 15.0f, 20.0f, Author("F. Scott", "Fitzgerald"), 5, Category.FICTION, "Supplier"), mock file exists and is writable	True, file updated successfully	PASS
TC-02	mocktestDeleteFromFile()	Delete the book from the file	Book("1234567890", "The Great Gatsby", 15.0f, 20.0f, Author("F. Scott", "Fitzgerald"), 5, Category.FICTION, "Supplier"), mock file exists and is writable	True, book deleted from file, book does not exist in list	PASS

Checks how the files are handled when a book information is updated or when a book is deleted from the file by mocking the file.

OrderBookQuantityTesting.java

This test case ensures that when an order is placed, the stock of the book is updated correctly.

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	testAddBooks()	Verify that the stock of the book decreases correctly when an order is processed.	Two books: "Harry Potter" (10 stock) and "Hamlet" (5 stock) with authors and categories.	Books list size should be 2, with correct titles "Harry Potter" and "Hamlet".	PASS

- From **Book** class :
- **getStock()** to update stock

- From **Order** class :
- **setBookSold()** to set linked book
- **getNewBookStock()** to update stock

- Interdependency :
- Because it checks how the **Order** and **Book** classes work together to handle stock updates

OrderUserBookTest.java
This test ensures that order can be created and correctly associate with books and users.

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	testAddBooks()	Verify that books can be added to the list and their details are stored correctly.	Two books: "Harry Potter" and "Hamlet" with their respective authors and categories..	Books list size should be 2, with titles "Harry Potter" and "Hamlet".	PASS
TC-02	testAddUser()	Verify that a user can be added to the user list and their details are stored correctly.	User with username "jkrowling", password "password123", and role ADMIN	Users list size should be 1, with the username "jkrowling".	PASS
TC-03	testCreateOrder()	Ensure an order can be created, linked to a user and book, and its details are stored correctly.	Order for "Harry Potter", username "jkrowling", quantity 1, price 15.0.	Orders list size should be 1, with the client name "jkrowling", ISBN "1234567890", and quantity 1.	PASS
TC-04	testRetrieveOrderDetail	Verify the details of an	Order for "Harry Potter",	Retrieved order: Client	PASS

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
	s()	order, including the client name, book ISBN, quantity, and book details (e.g., title, author).	username "jkrowling", quantity 1, price 15.0.	name "jkrowling", ISBN "1234567890" , quantity 1, book title "Harry Potter", author "JK Rowling".	

- From **Book** Class :
- Constructor **newBook()**
- **saveInFile()** to store book data
- **getBooks()** retrieve all books

- From **User** class:
- Constructor **newUser()**
- **saveInFile** to store user data
- **getUsers** to retrieve all users

- From **Orde**r class :
- Constructor **newOrder()**
- **saveInFile** to store order data
- **getOrders** to retrieve all orders

- Interdependency :
- An order references a **book** (via ISBN) and a **user** (via username)
- **Order.getOrders()** retrieves the order and checks that it correctly links to the corresponding **Book** and **User** data.

System Testing

LoginView.java - LoginViewTest.java
To verify the correctness of user interactions and behaviors in the login view, such as entering credentials, validating them, and responding to user actions like clicking buttons

Test Case ID	Method	Description	Input	Expected Output	Pass/Fail
TC-01	should_display_error_for_invalid_credentials()	Verify that an error message is displayed for invalid login credentials.	Username : invalidUser Password : wrongPassword	Error label should contain an error message.	PASS
TC-02	should_login_with_valid_credentials()	Verify that the user can log in with valid credentials and no error message is displayed.	Username : validUser Password : correctPassword	Error label should be empty, and scene should transition to the next view.	PASS
TC-03	should_close_on_cancel()	Verify that the application closes when the cancel button is clicked.	Click on the cancel button.	The stage should close (no longer visible).	PASS

MainView..java - MainViewTest.java
This test ensures the MenuBar in the MainView is properly initialized and reflects the expected menus, especially for an user.

Test Case ID	Test Case	Method	Description	Expected Output	Pass/Fail
TC-01	testMainViewInitialization	getMenuBar()	Verifies that the MenuBar component is initialized correctly and contains the expected menus, especially for the Admin role.	-Menu Bar is not null. -Menu Bar contains at least one menu. - The first menu's text is “Home”. - A menu with “Statistics” exists.	PASS
TC-02	testTabPaneInitialization	getTabPane()	Retrieves the TabPane from the MainView and validates that it is properly initialized and contains the required tabs.	-TabPane is not null -TabPane contains at least one tab. - The first tab's text is “Home”.	PASS
TC-03	testMenuBarInitialization	getMenuBar()	Confirms that theMenuBar component is present and contains menus for navigation within the application.	-MenuBar is not null -MennuBar contains menus.	PASS

HomeView.java - HomeViewTest.java
This test ensures the MenuBar in the MainView is properly initialized and reflects the expected menus, especially for an user.

Test Case ID	Test Case	Method	Description	Expected Output	Pass/Fail
TC-01	testMainViewInitialization	getMenuBar()	Verifies that the MenuBar component is initialized correctly and contains the expected menus, especially for the Admin role.	-Menu Bar is not null. -Menu Bar contains at least one menu. - The first menu's text is “Home”. - A menu with “Statistics” exists.	PASS
TC-02	testTabPaneInitialization	getTabPane()	Retrieves the TabPane from the MainView and validates that it is properly initialized and contains the required tabs.	-TabPane is not null -TabPane contains at least one tab. - The first tab's text is “Home”.	PASS
TC-03	testMenuBarInitialization	getMenuBar()	Confirms that theMenuBar component is present and contains menus for navigation within the application.	-MenuBar is not null -MennuBar contains menus.	PASS

SearchView..java - SearchViewTest.java

- The tests validate that the SearchView behaves predictably, providing a seamless and intuitive experience.
- Ensures all components adhere to design requirements
- Catches potential issues early

Test Case ID	Test Case	Description	Input	Expected Output	Pass/Fail
TC-01	testSearchFieldInitialization	Verifies that the search field is correctly initialized with the right placeholder text.	none	Search field should be initialized and have the placeholder text "Type something ... ".	PASS
TC-02	testSearchButtonInitialization	Ensures the clear button is correctly initialized, is an instance of ClearButton and has an icon.	none	Clear button should be initialized, be a ClearButton and its graphic should be an ImageView	PASS
TC-03	testSearchPaneInitialization	Verifies that the search pane contains the correct elements in the right order..	none	Search pane should have 3 elements: a Label, a SearchButton, and a ClearButton	PASS
TC-04	testClearButtonFunctionality	Tests if the clear button clears the text in the search field when clicked.	Set text in the search field and click clear button	After clicking the clear button, the text in the search field should be cleared (empty string).	PASS
TC-05	testSearchButtonFunctionality	Tests if clicking the search button retains the current text in the search field.	Set text in the search field and click search button	After clicking the search button, the text in the search field should remain unchanged.	PASS

BookView..java - BookViewTest.java

Test Case ID	Test Case	Method	Description	Expected Output	Pass/Fail
TC-01	Verify "Book List" Button	clickBookListButton()	Simulates clicking on the "Book List" button and verifies the button's text is correct.	Button labeled as "Book List" is present and clickable.	PASS
TC-02	Verify "Author List" Button	clickAuthorListButton()	Simulates clicking on the "Author List" button and verifies the button's text is correct.	Button labeled as "Author List" is present and clickable.	PASS
TC-03	Verify "Books Sold" Button	clickBooksSoldButton()	Simulates clicking on the "Books Sold" button and verifies the button's text is correct.	Button labeled as "Books Sold" is present and clickable.	PASS
TC-04	Check all of the buttons	shouldContainButtons()	(Commented-out test) Verifies that all buttons ("Book List," "Author List," "Books Sold") have correct labels.	All three buttons are correctly labeled and present.	PASS

Conclusion

In conclusion, through unit testing, integration testing and system testing the bookstore program proved to be a reliable and mostly functional system. The majority of the tests passed successfully. Through these tests we were able to systematically evaluate the functionality and overall performance of the system. However, we also faced some errors and difficulties during the testing process. This highlighted potential areas for improvement. We identified and analysed the errors to further enhance the quality of the program.
Overall, the tests we performed not only validated the program's flow but also provided us with insights into real world testing.