Ain Shams University – Faculty of Engineering

Computer Engineering and Software Systems Program

Software Testing, Validation, and Verification

CSE338

Team 17

Name	ID
Youssef Adel Albert	21P0258
Mark Saleh Sobhi	21P0206
Ahmed Tarek Mahmoud	2100561
Omar Khaled Essam	21P0178

GitHub Link : https://github.com/Mark-S2004/E-commerce-System

Introduction

This document serves as a comprehensive guide for testing the codebase of our e-commerce system. While our focus is on ensuring the correctness and functionality of the code through JUnit tests, it's essential to maintain a clear understanding of the system's overall purpose and functionality.

The e-commerce platform features fundamental functionalities tailored to both customers and managers. Users encounter login and account creation options on the homepage, allowing for easy access to personalized services. Managers benefit from intuitive tools for managing the product catalog and adding products. Customers experience a user-friendly product catalog that enables straightforward browsing and searching to locate desired items. The cart and checkout process are designed for convenience, facilitating swift and seamless order placement.

Throughout this testing document, our primary focus is on verifying the correctness of the codebase using JUnit tests. By testing each component, we aim to ensure that the code functions as intended and adheres to established requirements.

Requirement 1

Test ID	Title	Precondition	Steps	Test Data	Expected Results	Status	Actual Results
TC_AccountManagerTest_createCustomerAccount	Create random accounts to make sure that the usernames and passwords are then saved in the customer test accounts hash set		1. User inserts username and password 2. User selects the customer button 3. User press create account button		The usernames and passwords must be saved in the hash sets	Passed	The username and password were added to the hash sets successfully
TC_AccountManagerTest_createManagerAccount	Create random accounts to make sure that the usernames and passwords are then		1. User inserts username and password 2. User selects the manager button 3. User press create account button		The usernames and passwords must be saved in the hash sets	Passed	The username and password were added to the hash sets successfully

	saved in the manager test accounts hash set						
TC_AccountManagerTest_repeatedUsername	Make sure no usernames are repeated	Two same usernames are created	1. Username must be created. 2. Same username created again. 3. Login button pressed		An error message will be shown as the username is created twice.	Passed	"This username has been already used before" error message appeared.
TC_AccountManagerTest_validAuthentication	Valid credentials entered	Username and password must be created	1. Username must be entered. 2. Password must be entered. 3. Login Button pressed	Created accounts	User will be logged in successfully	Passed	User logged in successfully.
TC_AccountManagerTest_invalidAuthentication	Invalid credentials entered	Username and password must be created	1. Username must be entered. 2. Password must be entered. 3. Login Button pressed	Username: "john" Password: "password 123"	User will fail to log in and an error message will appear	Passed	User failed to login and error message appeared

TC_AccountManagerTest_isManager	Checks if the created accounts are manager accounts or non-manager accounts	Account must be created		Created manager account	The manager account must return true for the test case	Passed	The created account returned true as it is a manager account
TC_AccountTest_getUsername	Saves the username that was inserted	Account must be created to make sure the entered username is correct	1. Username must be entered. 2. Password must be entered. 3. Login Button pressed	Username: “john”	The username entered will match that with the test	Passed	Both usernames are matched
TC_AccountTest_getPassword	Saves the password that was inserted	Account must be created to make sure the entered password is correct	1. Username must be entered. 2. Password must be entered. 3. Login Button pressed	Password: “password 123”	The password entered will match that with the test	Passed	Both passwords are matched
TC_ProductTest_setID	Set an ID for a product	You must be on manager account to set products		ID: 2	The ID set must match the ID in the test	Passed	Both are matched
TC_ProductTest_getID	Return the ID of a product	A product must be already inserted		ID: 1	The ID returned must match	Passed	ID returned is same to

					that of the product that was inserted		that was set
TC_ProductTest_setName	Set a name for a product	You must be on manager account to set products		Name: Coca	The name set must match the name in the test	Passed	Both are matched
TC_ProductTest_getName	Return the name of a product	A product must be already inserted		Name: Pepsi	The name returned must match that of the product that was inserted	Passed	Name returned is same to that was set
TC_ProductTest_setPrice	Set a price for a product	You must be on manager account to set products		Price: 4	The price set must match the price in the test	Passed	Both are matched
TC_ProductTest_getPrice	Return the price of a product	A product must be already inserted		Price: 5	The price returned must match that of the product that was inserted	Passed	Price returned is equal to that was set
TC_ProductCatalogTest_getAllProducts	Make sure that the catalog is empty						

TC_ProductCatalogTest_addProduct	Used for adding products to product catalog	Must be a on a manager account to add products	1. Create a manager account 2. Insert product name, ID and price 3. Press add product	Product ID: “001”, “002” Product name: “TestProduct1”, “TestProduct2” Product price: 2, 5	The product will be added successfully to the product catalog	Passed	The product appeared in the product catalog
TC_ProductCatalogTest_addSameItem	Used for adding products to the shopping cart	Must have a customer account	1. Create a customer account 2. Then pick a product from the product catalog 3. Press add item		The product will be added successfully to the shopping cart	Passed	The product appeared in the shopping cart
TC_ProductCatalogTest_removeProduct	Used for removing products from product catalog	Must be a on a manager account and have products added to the product catalog	1. Create a manager account 2. Select the product you want to remove 3. Press remove product	Product ID: “001”, “002” Product name: “TestProduct1”, “TestProduct2”	The product will be removed successfully from the product catalog	Passed	The product removed from product catalog

				Product price: 2, 5			
TC_ProductCatalogTest_ removeSameItem	Used for removing products from the shopping cart	Must have products added in the shopping cart	1. Login with your customer account 2. Then pick a product from the shopping cart 3. Press remove button		The product will be removed successfully from the shopping cart	Passed	The product removed from shopping cart
TC_ProductCatalogTest_ searchProducts	Validate that the products searched for is that same that appeared	Must have a customer account created and logged in	1. Login with your customer account 2. Press on the search bar 3. Enter the name of product		The product name will appear in the dropdown list if available	Passed	The name of the product appeared
TC_ShoppingCartTest_ getItems	Return the items that are in the shopping cart			No items in shopping cart	The test will return nothing as there is no items in the shopping cart	Passed	The shopping cart is empty
TC_ShoppingCartTest_ getTotal	Return the total price of	Items must be added in the shopping cart	1. Login with a customer account	Item1 quantity = 5	The total price must equal both	Passed	Total price are both equal

	items in the shopping cart		2. Add items in the shopping cart 3. Then press on the shopping cart button 4. The price of the items will appear	Item2 quantity = 2	items multiplied by their quantity		
TC_ShoppingCartTest_ addSameItem	Ensures that when you add the same item in the shopping cart its quantity increases	User must be logged in with customer account	1. Login with customer account 2. Choose the item 3. Choose how many items you want 4. Then press add				
TC_ShoppingCartTest_ addDifferentItems	Verifies the behavior of adding different items to the shopping cart	Different products must be added by the manager	1. Login with customer account 2. Choose the item 3. Then press add 4. Choose another item 5. Then press add	Items : Item1 , item2 Quantity: 5 , 2	The cart must maintain the correct quantities and items after each addition.	Passed	Items visible with the correct quantities

TC_ShoppingCartTest_ removeItem	Tests the behavior of removing an item from the shopping cart	Shopping cart must already have added items	1. Select the shopping cart tab 2. Then select the item you want to remove 3. The press remove	Items : Item1 , item2 Quantity: 5 , 2	The cart must be updated correctly and the item selected must be removed	Passed	Shopping cart updated and the selected item was removed
TC_ShoppingCartTest_ clear	Verifies that the all items in the shopping cart will be cleared	Shopping cart must already have added items	1. Select the shopping cart tab 2. Press the clear button		The shopping cart is reset to an empty state.	Passed	The shopping cart is empty.
TC_PlacedOrderTest_ getOrderId	Ensures that the correct order ID is returned	Order must be already placed		Order ID = "123"	Order ID must be returned correctly and same as the test case	Passed	Order ID is the same as the test case
TC_PlacedOrderTest_ setItems	Ensures that the ordered items are updated correctly			Items ID: "P001", "P002" Items name:	Items ordered must be set as the test case	Passed	Items ordered updated successful ly

				"Pepsi", "Coca" Items price: 5 , 4 Items quantity: 2 , 1			
TC_PlacedOrderTest_ getItems	Ensures that the correct ordered items are returned	Order must be already placed		Items ID: "P001", "P002" Items name: "Pepsi", "Coca" Items price: 5 , 4 Items quantity: 2 , 1	Ordered items must be returned correctly and same as the test case	Passed	Ordered items are the same as the test case

TC_OrderManagementTest_testPlaceOrder	Ensures that the order is added successfully to the system and can be retrieved	Payment must be successful for the order to be placed		Order ID: “0001” User Email: “john@example.com”	After payment was successful the order will be placed in the order menu tab	Passed	Order was added successfully to the order menu
TC_OrderManagementTest_testCancelOrder	Ensures that the order is cancelled and deleted from the system	An order must exist so It can be removed		Order ID: “0001” User Email: “john@example.com”	Order placed must be deleted from the order menu	Passed	Order was deleted from order menu

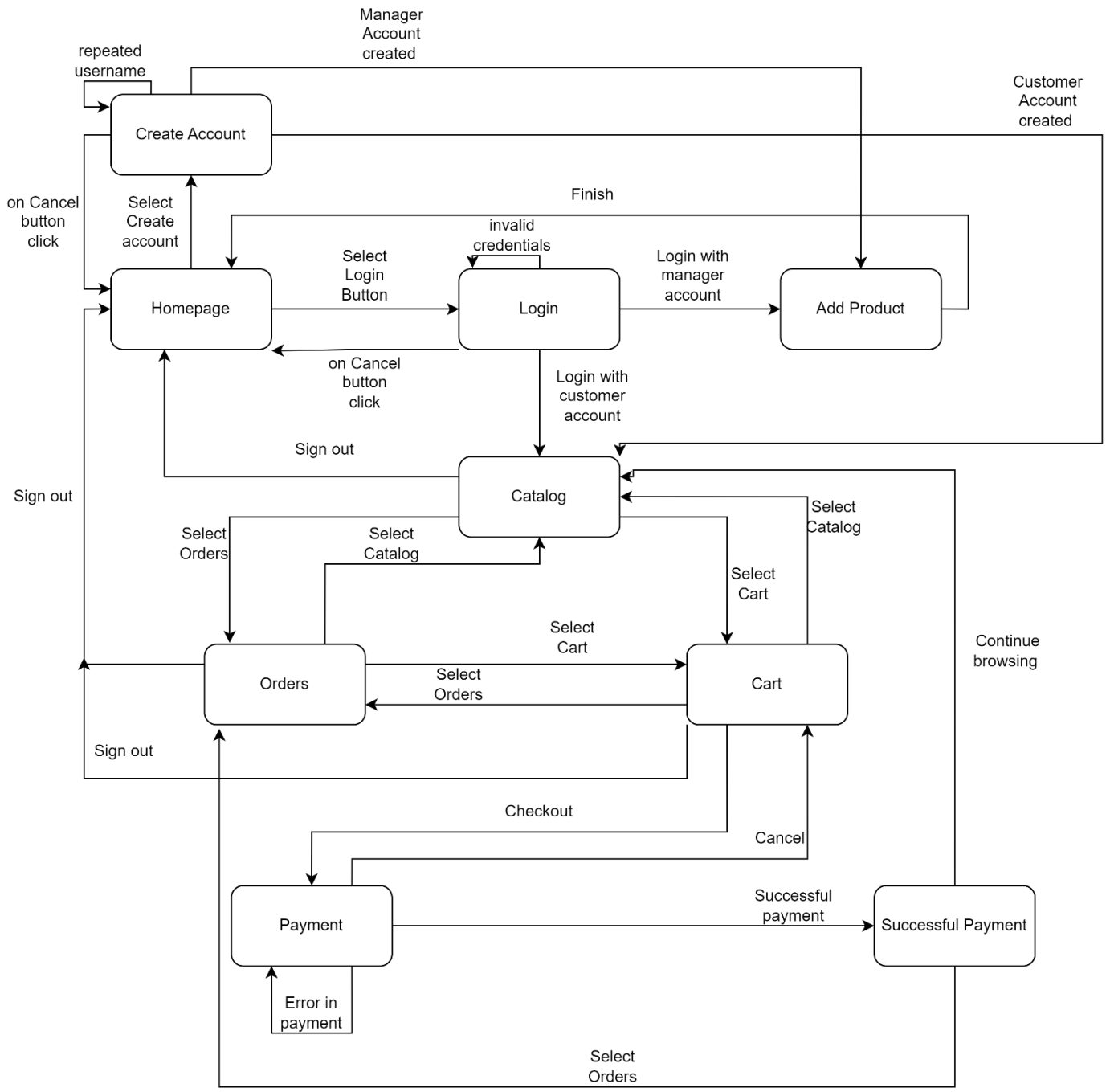
TC_PaymentProcessorTest_testProcessPaymentWithValidAmount	Ensures that the processPayment method behaves as expected when provided with a valid payment amount	Items must be added to the shopping cart so a payment can be made		Amount: 100	The payment will be successful	Passed	Payment successful and the payment successful window is opened and order was placed
TC_PaymentProcessorTest_testProcessPaymentWithZeroAmount	Ensures that the processPayment method behaves as expected when provided with zero amount	Items must be added to the shopping cart so a payment can be made		Amount: 0	The test will return false as there is no amount to pay with	Passed	The test returned false and the order was not placed
TC_PaymentProcessorTest_testProcessPaymentWithNegativeAmount	Ensures that the processPayment method behaves as	Items must be added to the shopping cart so a payment can be made		Amount: -100	The test will return false as there is no amount to pay with	Passed	The test returned false and the order was not placed

	expected when provided with an invalid payment amount						
TC_PaymentProcessorTest_testDisconnectGatewayProcessPayment	Ensures that the processPayment method behaves as expected when the payment gateway is disconnected	Items must be added to the shopping cart so a payment can be made			The test will return false as the payment method is invalid	Passed	The payment Gateway will be disconnected as the payment was unsuccessful
TC_PaymentProcessorTest_connectPaymentGateway	Ensures that the payment gateway is connected	Items must be added to the shopping cart so a payment can be made			The test will return true as the payment gateway is connected	Passed	Payment gateway connected

TC_PaymentProcessorTest_disconnectPaymentGateway	Ensures that the payment gateway is connected	Items must be added to the shopping cart so a payment can be made			The test will return false as the payment gateway is disconnected	Passed	Payment gateway disconnected

Requirement 2

FSM



Transition of states

	Action	Go to
Homepage	Select “Login”	Login
	Select “Create Account”	Create Account
Create Account	Customer Account created	Catalog
	Manager account created	Add product
Login	Login as manager account	Add product
	Login as customer account	Catalog
Catalog	Select “Cart”	Cart
	Select “Orders”	Orders
	Sign out	Homepage
Orders	Select “Cart”	Cart
	Select “Catalog”	Catalog
	Sign out	Homepage
Cart	Select “Catalog”	Catalog
	Select “Orders”	Orders
	Sign out	Homepage
	Checkout	Payment
Payment	Select “Cancel”	Cart
	Successful payment	Successful Payment
	Fail	Payment
Successful Payment	Select “continue browsing”	Catalog
Add product	Select “Finish”	Homepage

Requirement 3

Coverage Report

Current scope: all classes | EcommerceSystem

Coverage Summary for Package: EcommerceSystem

Package	Class, %	Method, %	Branch, %	Line, %
EcommerceSystem	100% (11/11)	95.8% (46/48)	100% (46/46)	98.3% (116/118)

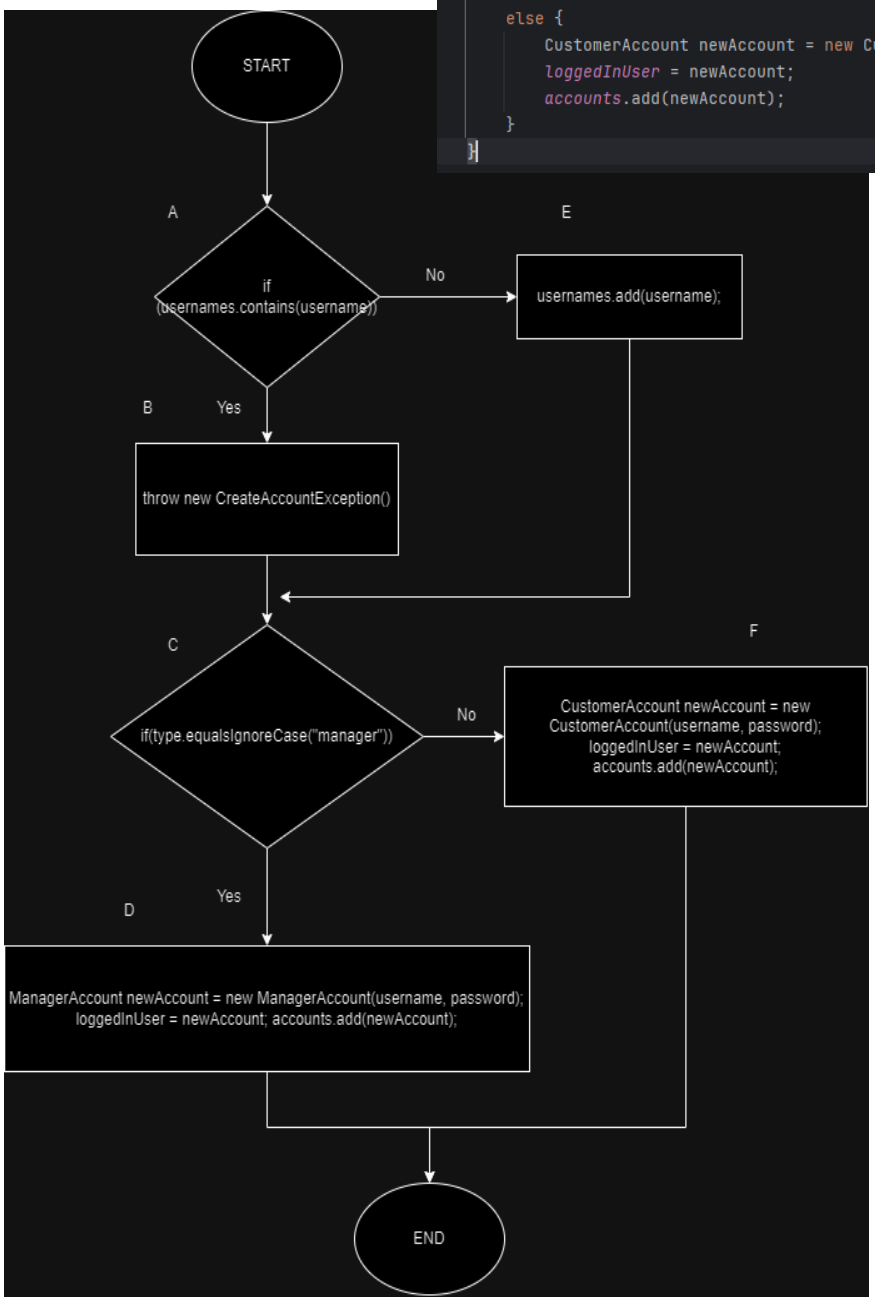
Class	Class, %	Method, %	Branch, %	Line, %
Account	100% (1/1)	100% (3/3)		100% (5/5)
AccountManager	100% (1/1)	85.7% (6/7)	100% (16/16)	95.8% (23/24)
CreateAccountException	100% (1/1)	100% (1/1)		100% (1/1)
CustomerAccount	100% (1/1)	100% (1/1)		100% (3/3)
ManagerAccount	100% (1/1)	100% (1/1)		100% (1/1)
OrderManagement	100% (1/1)	100% (4/4)		100% (6/6)
PaymentProcessor	100% (1/1)	100% (5/5)	100% (4/4)	100% (10/10)
Product	100% (1/1)	100% (8/8)		100% (11/11)
ProductCatalog	100% (1/1)	85.7% (6/7)	100% (22/22)	97.2% (35/36)
ShoppingCart	100% (1/1)	100% (6/6)	100% (4/4)	100% (14/14)
placedOrder	100% (1/1)	100% (5/5)		100% (7/7)

generated on 2024-04-26 21:43

AccountManager

createAccount()

```java
public static void createAccount(String username, String password, String type) throws CreateAccountException {
    if (usernames.contains(username)) {
        throw new CreateAccountException();
    } else {
        usernames.add(username);
    }
    if(type.equalsIgnoreCase( anotherString: "manager")) {
        ManagerAccount newAccount = new ManagerAccount(username, password);
        loggedInUser = newAccount;
        accounts.add(newAccount);
    }
    else {
        CustomerAccount newAccount = new CustomerAccount(username, password);
        loggedInUser = newAccount;
        accounts.add(newAccount);
    }
}
```



Test case 1:

usernames.contains(username) = True

Type.equalsIgnoreCase =manager

Block A will be executed then B, C and D.

Test case 2:

usernames.contains(username) = False

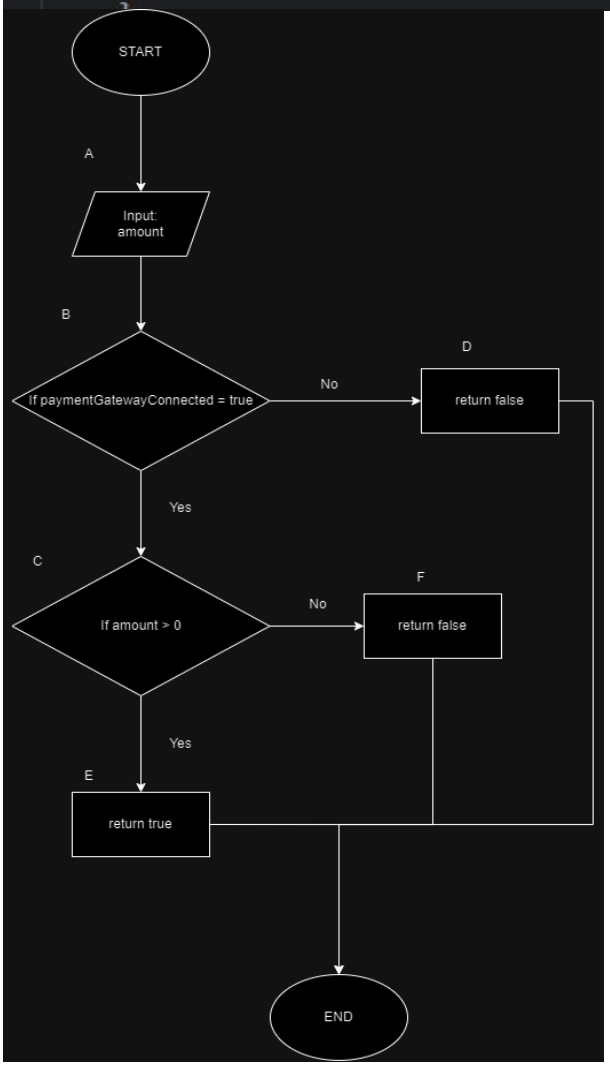
Type.equalsIgnoreCase = customer

Block A will be executed then E, C and F.

PaymentProcessor

processPayment()

```java
public boolean processPayment(double amount) {  1 usage
    if (paymentGatewayConnected) {
        // Simulating payment processing logic
        if (amount > 0) {
            // Payment successful
            return true;
        } else {
            // Payment failed
            return false;
        }
    } else {
        // Payment gateway not connected
        return false;
    }
```



Test case 1:

amount =10

paymentGatewayConnected = true

Block A will be executed then B, C and E.

Test case 2:

amount = 10

paymentGatewayConnected = false

Block A will be executed then B and D.

Test case 3:

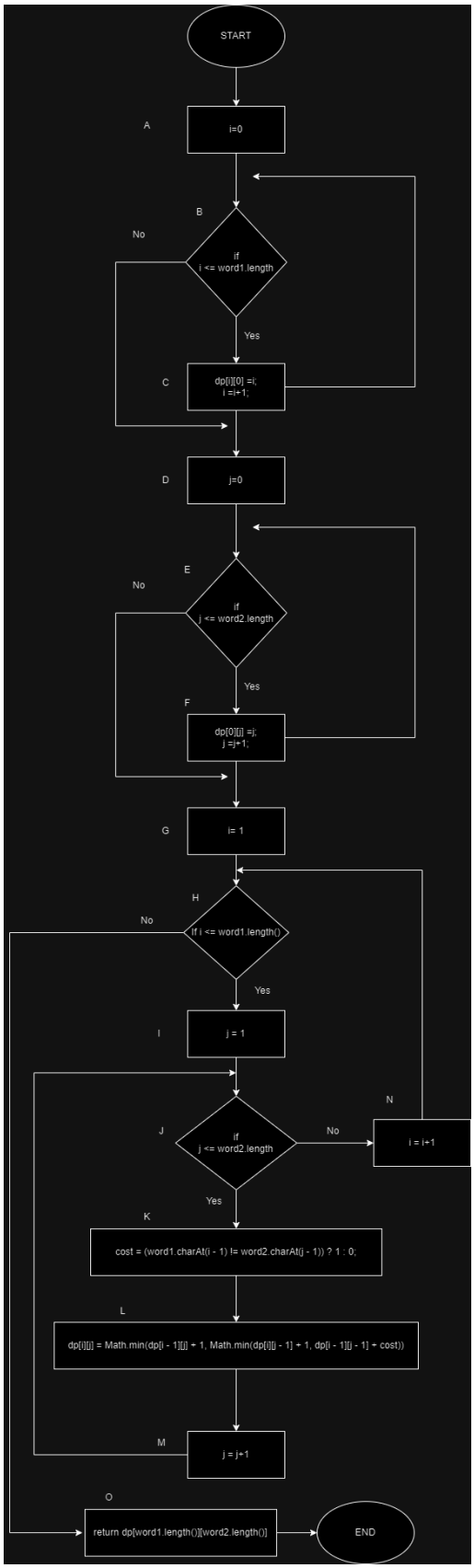
amount = -1

paymentGatewayConnected = true

block A will be executed then B, C and F.

ProductCatalog

calculateLevenshteinDistance()




```java
public static int calculateLevenshteinDistance(String word1, String word2) {  1 usage  Geno212
    int[][] dp = new int[word1.length() + 1][word2.length() + 1];

    for (int i = 0; i <= word1.length(); i++) {
        dp[i][0] = i;
    }

    for (int j = 0; j <= word2.length(); j++) {
        dp[0][j] = j;
    }

    for (int i = 1; i <= word1.length(); i++) {
        for (int j = 1; j <= word2.length(); j++) {
            int cost = (word1.charAt(i - 1) != word2.charAt(j - 1)) ? 1 : 0;
            dp[i][j] = Math.min(dp[i - 1][j] + 1, Math.min(dp[i][j - 1] + 1, dp[i - 1][j - 1] + cost));
        }
    }

    return dp[word1.length()][word2.length()];
}
```

Test case 1:

Word1.length = 0

Word2.length = 5

Block A will be executed then B, C, D, E, F, G, H and O.

Test case 2:

Word1.length = 5

Word2.length = 0

Block A will be executed then B, C, D, E, F, G, H, I, J, N and O.

Test case 3:

Word1.length = 5

Word2.length = 5

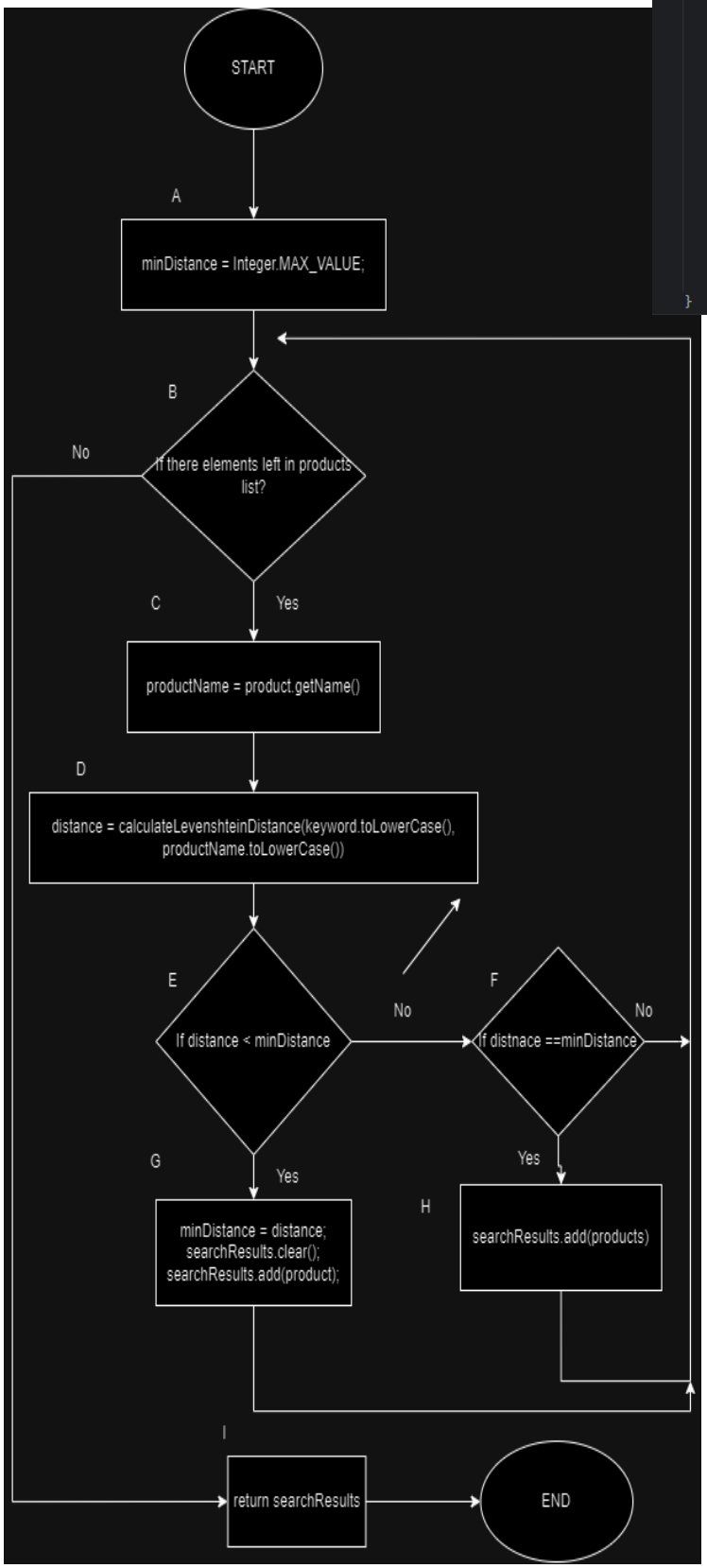
All blocks will be executed.

searchProduct()

```java
public static List<Product> searchProducts(String keyword) {  4 usages  Geno212 +1
    List<Product> searchResults = new ArrayList<>();
    int minDistance = Integer.MAX_VALUE;

    for (Product product : products) {
        String productName = product.getName();
        int distance = calculateLevenshteinDistance(keyword.toLowerCase(), productName.toLowerCase());
        if (distance < minDistance) {
            minDistance = distance;
            searchResults.clear();
            searchResults.add(product);
        } else if (distance == minDistance) {
            searchResults.add(product);
        }
    }

    return searchResults;
}
```



Test case 1:

-If there are no available products in the products list.

-Block A will be executed then B and I.

Test case 2:
-There are products available in the product list and all products distance is smaller than minDistance but not equal to it.
-Distance < minDistance

-Block A will be executed then B, C, D, E, G and I.

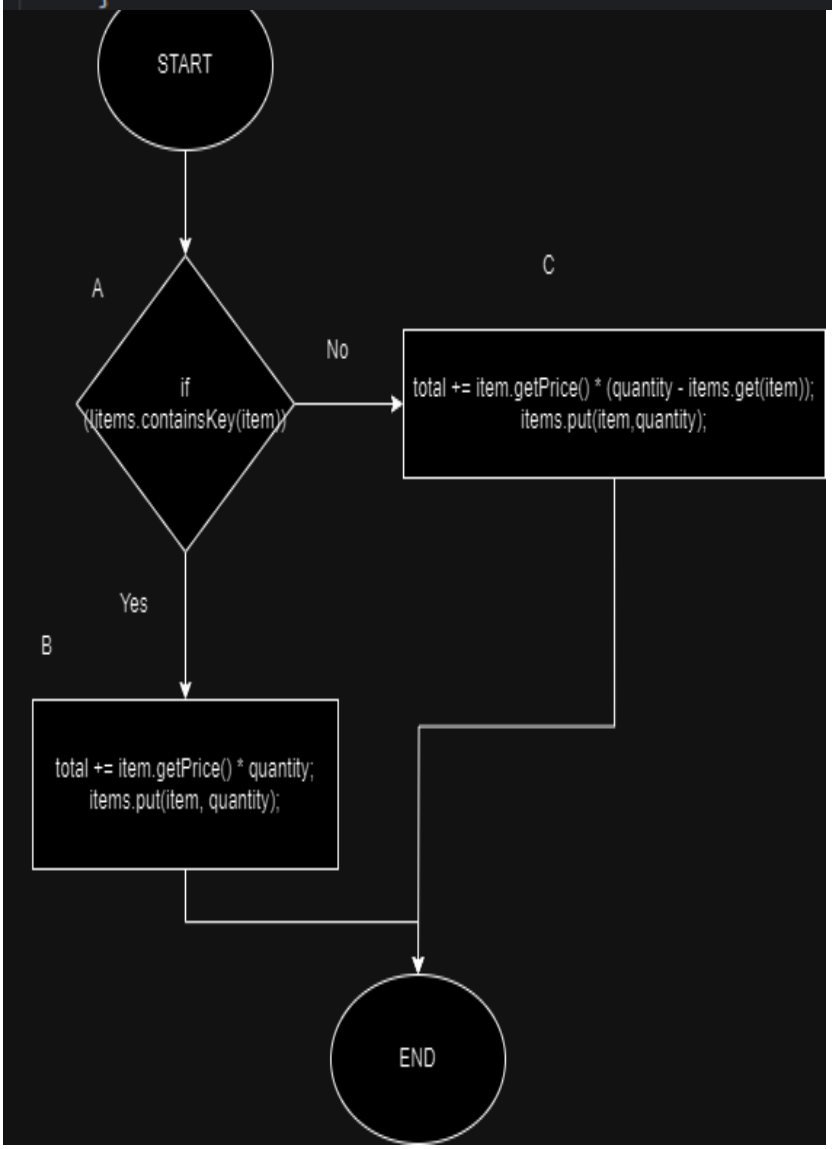
Test case 3:
-There are products available in the product list and there is one product with distance = minDistance.

-All blocks will be executed.

ShoppingCart

addItem()

```java
public void addItem(Product item, Integer quantity) {  9 usages  Mark Saleh
    if (!items.containsKey(item)) {
        total += item.getPrice() * quantity;
        items.put(item, quantity);
    } else {
        total += item.getPrice() * (quantity - items.get(item));
        items.put(item,quantity);
    }
}
```



Test case 1:
items.containsKey(item) = False

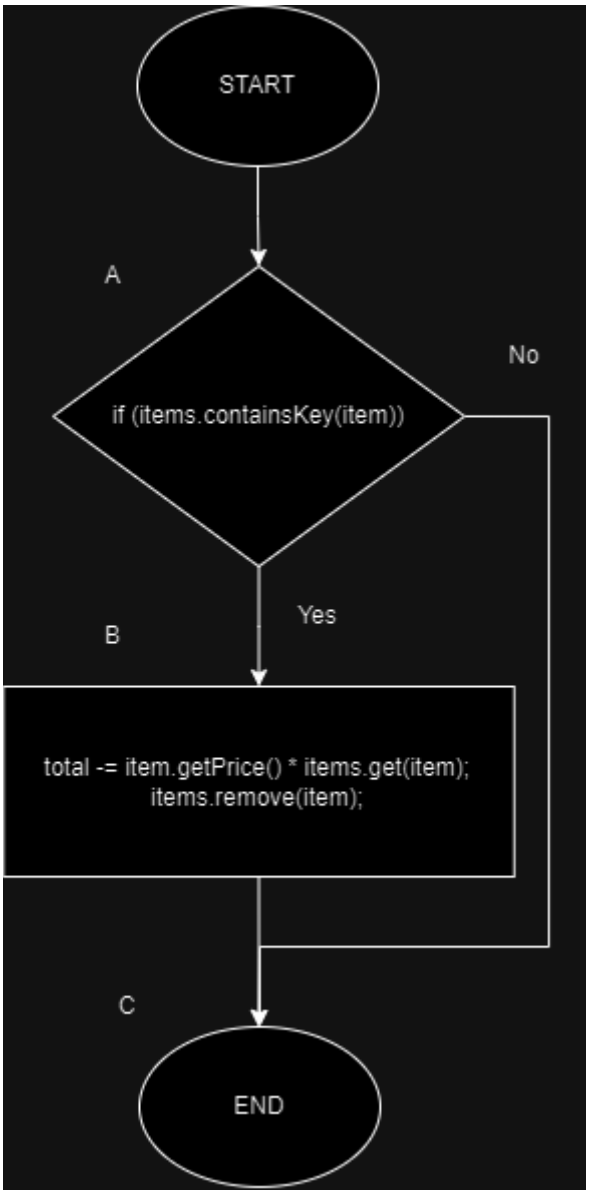
-Block A will be executed then B.

Test casse 2:
items.containsKey(item) = True

-Block A will be executed then C.

removeItem()

```
public void removeItem(Product item) {  3 usages  Mark Saleh
    if (items.containsKey(item)) {
        total -= item.getPrice() * items.get(item);
        items.remove(item);
    }
}
```



Test case 1:

items.containsKey(item) = False

-Block A will be executed then C.

Test case 2:

items.containsKey(item) = True

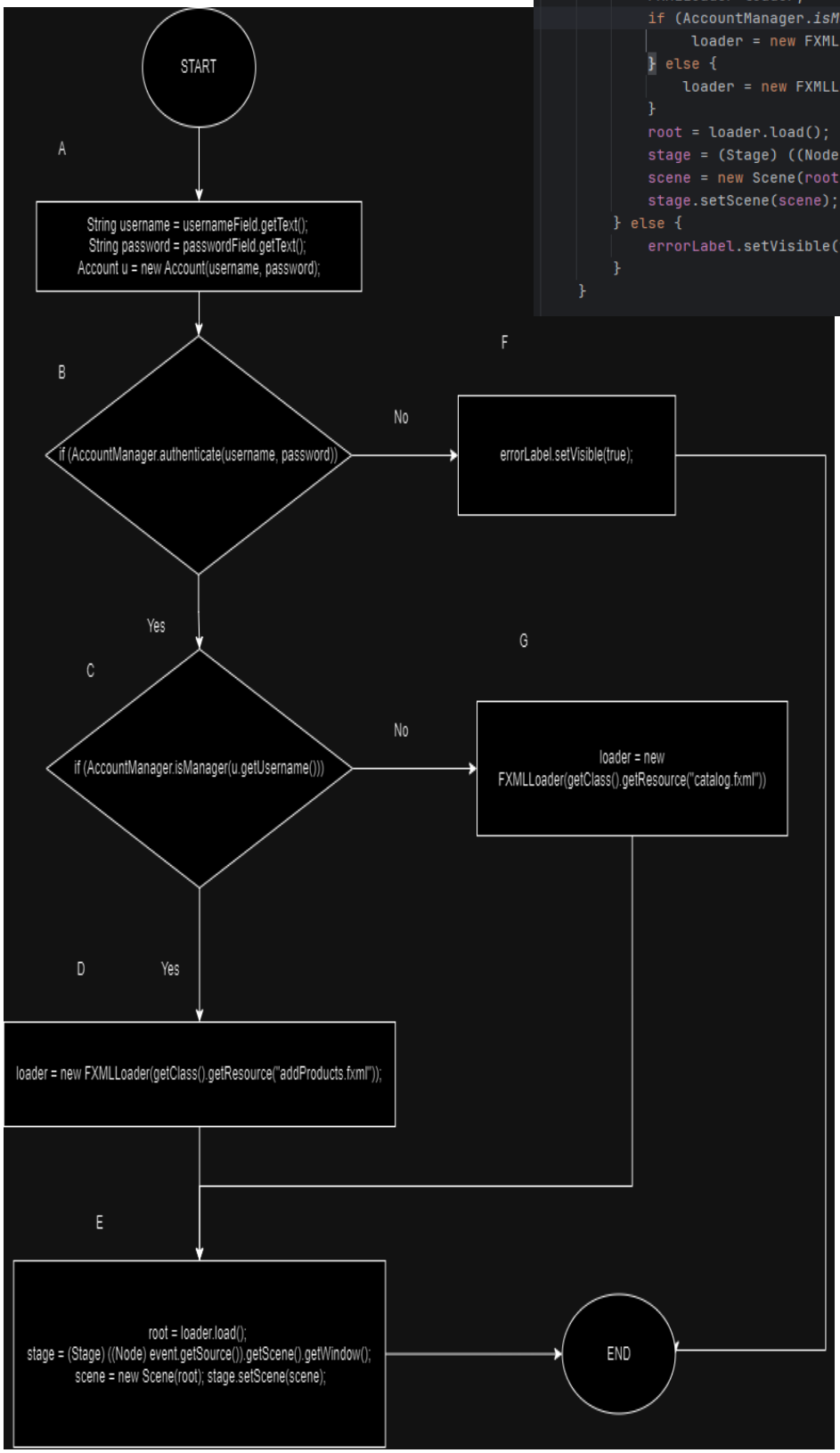
--Block A will be executed then B and C.

LoginController

Login()

```java
private void login(ActionEvent event) throws IOException{
    String username = usernameField.getText();
    String password = passwordField.getText();
    Account u = new Account(username, password);

    if (AccountManager.authenticate(username, password)) {
        FXMLLoader loader;
        if (AccountManager.isManager(u.getUsername())) {
            loader = new FXMLLoader(getClass().getResource( name: "addProducts.fxml"));
        } else {
            loader = new FXMLLoader(getClass().getResource( name: "catalog.fxml"));
        }
        root = loader.load();
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
    } else {
        errorLabel.setVisible(true);
    }
}
```



Test case 1:

AccountManager.*authenticate*(username, password) = True

AccountManager.*isManager*(u.getUsername())

= True

-Block A will be executed then B, C, D and E.

Test case 2:

AccountManager.*authenticate*(username, password) = True

AccountManager.*isManager*(u.getUsername())

= False

-Block A will be executed then B, C, G and E.

Test case 3:

AccountManager.*authenticate*(username, password) = False

Block A will be executed then F.

Requirement 4

Integration of Product class in both ProductCatalog class and ShoppingCart class using catalogController class

1. Test AddProduct Integration:

Create an integration test to verify that adding a product through the AddProductsController interacts correctly with the ProductCatalog and updates the catalog list.

Simulate user input by calling the addProduct() method in the AddProductsController with mock data.

Verify that the product is correctly added to the ProductCatalog and appears in the catalog list.

2. Test RemoveItem Integration:

Create an integration test to verify that removing a product through the AddProductsController interacts correctly with the ProductCatalog and updates the catalog list.

Simulate user selection of a product and call the removeItem() method in the AddProductsController.

Verify that the product is correctly removed from the ProductCatalog and no longer appears in the catalog list.

3. Test ShoppingCart Integration:

Create an integration test to verify that adding and removing items in the ShoppingCart interacts correctly with the ProductCatalog and updates the total price.

Simulate adding and removing items in the shopping cart and verify that the total price is updated accordingly.

Ensure that adding an item not present in the ProductCatalog results in an error or warning.

4. Test GUI Integration:

Create an integration test to verify that the GUI components (catalogController) interact correctly with the backend (ProductCatalog, ShoppingCart) and reflect changes in real-time.

Simulate user actions such as adding or removing products and verify that the GUI updates reflect these changes accurately.

Ensure that error messages are displayed appropriately when input validation fails.

5. Test Exception Handling Integration:

Create an integration test to verify that exception handling in the AddProductsController works correctly when invalid data is entered.

Simulate entering invalid data (e.g., non-numeric price) and verify that the appropriate error message is displayed.

6. Test Navigation Integration:

Create an integration test to verify that navigation between different screens (e.g., from AddProductsController to StartController) works correctly.

Simulate user actions such as clicking buttons to navigate between screens and verify that the correct screen is displayed.

7. Test Multiple Operations Integration:

Create integration tests that combine multiple operations (e.g., adding a product, removing a product, and updating the shopping cart) to ensure that the system behaves correctly under various scenarios.

Test edge cases such as adding and removing multiple items, updating quantities, and handling concurrent user interactions.

Integration of the class CreateAccountException

This class was used in the AccountManager class as it throws exception when creating an account with the same username that was used before and returns an error message stating that the username is already taken by another account. This integration was tested by trying to create more than one account with the same username and it always returned an error message.

Integration of CustomerAccount and ManagerAccount classes in AccountManager class

1. Integration Testing between AccountManager and CustomerAccount:

Test Case 1: Create Customer Account: Verify that when a new customer account is created through AccountManager, it is correctly added to the accounts set and the usernames set.

Call the createAccount method of AccountManager with a unique username, password, and "customer" type.

Assert that the customer account is added to the accounts set and the username is added to the usernames set.

Verify that the loggedInUser is set to the newly created customer account.

Test Case 2: Authenticate Customer: Verify that a customer can authenticate successfully using AccountManager.

Create a customer account using AccountManager.

Call the authenticate method of AccountManager with the created customer's username and password.

Assert that authentication is successful, and the loggedInUser is set to the customer account.

2. Integration Testing between AccountManager and ManagerAccount:

Test Case 3: Create Manager Account: Verify that when a new manager account is created through AccountManager, it is correctly added to the accounts set and the usernames set.

Call the createAccount method of AccountManager with a unique username, password, and "manager" type.

Assert that the manager account is added to the accounts set and the username is added to the usernames set.

Verify that the loggedInUser is set to the newly created manager account.

Test Case 4: Authenticate Manager: Verify that a manager can authenticate successfully using AccountManager.

Create a manager account using AccountManager.

Call the authenticate method of AccountManager with the created manager's username and password.

Assert that authentication is successful, and the loggedInUser is set to the manager account.

Test Case 8: Manager Privileges: Verify that a manager account has the correct privileges.

Create a manager account using ManagerAccount.

Call the isManager method of AccountManager with the created manager's username.

Assert that the method returns true, indicating that the account has manager privileges.

Integration between Product class in placedOrder class and OrderManagement class

1. Integration Testing between OrderManagement and placedOrder:

Test Case 1: Place Order: Verify that orders can be placed successfully and added to the order management system.

Create an instance of OrderManagement.

Create one or more instances of placedOrder representing orders.

Call the placeOrder method of OrderManagement with the created orders.

Assert that the orders are added to the orders list in the OrderManagement object.

Test Case 2: Cancel Order: Verify that orders can be canceled successfully and removed from the order management system.

Create an instance of OrderManagement.

Create one or more instances of placedOrder representing orders.

Place the orders using the placeOrder method.

Call the cancelOrder method of OrderManagement with the placed orders.

Assert that the orders are removed from the orders list in the OrderManagement object.

Test Case 3: Get All Orders: Verify that all placed orders can be retrieved successfully.

Create an instance of OrderManagement.

Create one or more instances of placedOrder representing orders.

Place the orders using the placeOrder method.

Call the getAllOrders method of OrderManagement.

Assert that the returned list contains all the placed orders.

2. Integration Testing between placedOrder and Product:

Test Case 4: Set Items in placedOrder: Verify that items can be set in a placed order.

Create an instance of placedOrder.

Create one or more instances of Product.

Create a map of products and quantities.

Call the setItems method of placedOrder with the created map.

Assert that the items are correctly set in the placed order.

Test Case 5: Get Order ID and Items: Verify that the order ID and items can be retrieved successfully from a placed order.

Create an instance of placedOrder.

Call the getOrderId method to retrieve the order ID.

Call the getItems method to retrieve the items.

Assert that the order ID is correct and the items are retrieved successfully.

3. Integration Testing between OrderManagement and Product:

Test Case 6: Integration of Order and Product: Verify that products associated with placed orders are handled correctly by the order management system.

Create an instance of OrderManagement.

Create one or more instances of Product.

Create a map of products and quantities.

Create an instance of placedOrder and set the items using the created map.

Place the order using the placeOrder method of OrderManagement.

Assert that the order is successfully added to the order management system and the associated products are handled correctly.

Integration between CreateAccountController class and AccountManager class

Test Case 1: Create Account Successfully:

Prepare a test scenario where a user account is created through the GUI interface.

Instantiate a CreateAccountController.

Set up the necessary input fields with valid username, password, and user type.

Simulate the creation of the account by invoking the createAccount method.

Assert that the account is successfully created in the AccountManager.

Test Case 2: Attempt to Create Account with Existing Username:

Prepare a test scenario where a user attempts to create an account with an existing username.

Instantiate a CreateAccountController.

Set up the necessary input fields with a username that already exists.

Simulate the creation of the account by invoking the createAccount method.

Assert that the GUI displays an error message indicating that the username already exists.

Test Case 3: Authentication after Account Creation:

Prepare a test scenario where a user attempts to authenticate after creating an account.

Instantiate an AccountManager.

Set up the necessary input fields with the username and password of the newly created account.

Simulate the authentication process by invoking the authenticate method.

Assert that the authentication is successful and the logged-in user matches the newly created account.

Integration between LoginController class and AccountManager

Test Case 1: Successful Login for Customer:

Prepare a test scenario where a valid customer account exists in the system.

Instantiate a LoginController.

Set up the usernameField and passwordField with valid customer credentials.

Simulate a login attempt by invoking the login method.

Assert that the login is successful, and the correct scene (e.g., catalog view) is loaded.

Test Case 2: Successful Login for Manager:

Prepare a test scenario where a valid manager account exists in the system.

Instantiate a LoginController.

Set up the usernameField and passwordField with valid manager credentials.

Simulate a login attempt by invoking the login method.

Assert that the login is successful, and the correct scene is loaded.

Test Case 3: Unsuccessful Login:

Prepare a test scenario where the entered credentials do not match any existing account.

Instantiate a LoginController.

Set up the usernameField and passwordField with invalid credentials.

Simulate a login attempt by invoking the login method.

Assert that the login fails, and an appropriate error message is displayed.

Test Case 4: Manager Privileges:

Prepare a test scenario where the logged-in user is a manager.

Instantiate a LoginController.

Set up the usernameField and passwordField with valid manager credentials.

Simulate a login attempt by invoking the login method.

Assert that the system correctly identifies the manager account and loads the manager-specific view .

Test Case 5: Account Creation:

Prepare a test scenario where a new account is created through the AccountManager.

Instantiate a LoginController.

Simulate account creation using the createAccount method of AccountManager.

Set up the usernameField and passwordField with the newly created account's credentials.

Simulate a login attempt by invoking the login method.

Assert that the login is successful, and the correct scene is loaded for the newly created account type.

Integration between AddProductsController class, Product class and ProductCatalog class

Test Case 1: Add Product Successfully:

Prepare a test scenario where a product is added through the GUI interface.

Instantiate an AddProductsController.

Set up the necessary input fields with valid product details.

Simulate the addition of the product by invoking the addProduct method.

Assert that the product is successfully added to the product catalog.

Test Case 2: Attempt to Add Duplicate Product:

Prepare a test scenario where a product with the same name as an existing product is added.

Instantiate an AddProductsController.

Set up the necessary input fields with details of a product that already exists.

Simulate the addition of the product by invoking the addProduct method.

Assert that the GUI displays an error message indicating that the product already exists.

Test Case 3: Remove Product:

Prepare a test scenario where a product is removed through the GUI interface.

Instantiate an AddProductsController.

Simulate the selection of a product from the catalog list.

Simulate the removal of the selected product by invoking the removeItem method.

Assert that the product is successfully removed from the product catalog.

Integration Testing between catalogController, ShoppingCart, and ProductCatalog

Test Case 1: Add Item to Cart from Catalog:

Prepare a test scenario where a user selects an item from the catalog and adds it to the shopping cart through the GUI interface.

Instantiate a catalogController.

Set up the necessary input fields by selecting a product from the catalog.

Simulate the action of adding the item to the cart by invoking the addItem method in the ShoppingCart.

Assert that the selected item is successfully added to the shopping cart, and the total price is updated accordingly.

Test Case 2: Remove Item from Cart from Catalog:

Prepare a test scenario where a user removes an item from the shopping cart while browsing the catalog through the GUI interface.

Instantiate a catalogController.

Set up the necessary input fields by selecting a product from the catalog.

Simulate the action of removing the item from the cart by invoking the removeItem method in the ShoppingCart.

Assert that the selected item is successfully removed from the shopping cart, and the total price is updated accordingly.

Test Case 3: Search for Products in Catalog:

Prepare a test scenario where a user searches for products in the catalog using a search keyword through the GUI interface.

Instantiate a catalogController.

Set up the search field with a specific keyword.

Simulate the action of searching for products by invoking the searchProducts method in the ProductCatalog.

Assert that the search results are correctly displayed in the catalog list view.

Test Case 4: Increment and Decrement Quantity in Catalog:

Prepare a test scenario where a user increments and decrements the quantity of an item in the catalog before adding it to the shopping cart through the GUI interface.

Instantiate a catalogController.

Set up the necessary input fields by selecting a product from the catalog.

Simulate the action of incrementing and decrementing the quantity.

Assert that the quantity label is updated accordingly and the decrement button is disabled when the quantity reaches 1.

Test Case 5: Switch to Cart from Catalog:

Prepare a test scenario where a user switches to the shopping cart view from the catalog view through the GUI interface.

Instantiate a catalogController.

Simulate the action of switching to the cart view by invoking the appropriate method in the catalogController.

Assert that the GUI interface navigates to the cart view successfully.

Integration Testing between CartController, ShoppingCart, and AccountManager

Test Case 1: Add Item to Cart:

Prepare a test scenario where a user adds an item to the shopping cart through the GUI interface.

Instantiate a CartController.

Set up the necessary input fields with a product to be added to the cart.

Simulate the addition of the item to the cart by invoking the addItem method in the ShoppingCart.

Assert that the item is successfully added to the cart, and the total price is updated accordingly.

Test Case 2: Remove Item from Cart:

Prepare a test scenario where a user removes an item from the shopping cart through the GUI interface.

Instantiate a CartController.

Set up the necessary input fields with a product to be removed from the cart.

Simulate the removal of the item from the cart by invoking the removeItem method in the ShoppingCart.

Assert that the item is successfully removed from the cart, and the total price is updated accordingly.

Test Case 3: Switch to Payment Page:

Prepare a test scenario where a user switches to the payment page from the shopping cart view through the GUI interface.

Instantiate a CartController.

Simulate the action of switching to the payment page by invoking the appropriate method in the CartController.

Assert that the GUI interface navigates to the payment page successfully.

Test Case 4: Switch to Catalog Page:

Prepare a test scenario where a user switches to the catalog page from the shopping cart view through the GUI interface.

Instantiate a CartController.

Simulate the action of switching to the catalog page by invoking the appropriate method in the CartController.

Assert that the GUI interface navigates to the catalog page successfully.

Test Case 5: Switch to Orders Page:

Prepare a test scenario where a user switches to the orders page from the shopping cart view through the GUI interface.

Instantiate a CartController.

Simulate the action of switching to the orders page by invoking the appropriate method in the CartController.

Assert that the GUI interface navigates to the orders page successfully.

Integration Testing between paymentController, ShoppingCart, OrderManagement, and PaymentProcessor

Test Case 1: Payment Processing:

Prepare a test scenario where a user initiates a payment for the items in the shopping cart through the GUI interface.

Instantiate a paymentController.

Set up the necessary input fields to enter the payment amount.

Simulate the action of processing the payment by invoking the appropriate methods in the PaymentProcessor.

Assert that the payment is processed successfully and the order is placed if the payment is successful.

Verify that the shopping cart is cleared after successful payment.

Test Case 2: Switch to Cart from Payment:

Prepare a test scenario where a user switches to the shopping cart view from the payment section through the GUI interface.

Instantiate a paymentController.

Simulate the action of switching to the shopping cart view by invoking the appropriate method in the paymentController.

Assert that the GUI interface navigates to the shopping cart view successfully.

Test Case 3: Invalid Payment Amount:

Prepare a test scenario where a user enters an invalid payment amount (e.g., negative amount) through the GUI interface.

Instantiate a paymentController.

Set up the input field with an invalid payment amount.

Simulate the action of processing the payment by invoking the appropriate methods in the PaymentProcessor.

Assert that the payment fails and an error message is displayed in the GUI interface.

Test Case 4: Order Placement:

Prepare a test scenario where a user successfully completes the payment, and an order is placed through the GUI interface.

Instantiate a paymentController.

Simulate the action of processing the payment by invoking the appropriate methods in the PaymentProcessor.

Assert that an order is placed in the OrderManagement after successful payment.

Integration between orderManagementController class and OrderManagement class

Test Case 1: View Orders in Order Management:

Prepare a test scenario where a user views the placed orders in the order management section through the GUI interface.

Instantiate an orderManagementController.

Set up the necessary input fields to display the orders.

Simulate the action of loading orders by invoking the appropriate methods in the OrderManagement.

Assert that the orders are correctly displayed in the GUI interface.