Testing Documentation – Electronics Store Management System

Course: SWE 303 – Software Engineering
Project: Electronic Store Management System
Testing Framework: JUnit 5, Maven

By: Halil Bagosi, Daniela Brahimi, Dea Hoxha

Table of Contents

1. Summary

The **Electronics Store Management System** is a robust desktop application designed to streamline point-of-sale operations, including inventory tracking, bill generation, and sales analysis. Built using **Java** and **JavaFX**, the system leverages core **Object-Oriented Programming (OOP)** principles to ensure a modular and maintainable architecture. This report details the comprehensive testing strategy employed to validate the system's reliability, focusing on the application of **Boundary Value Analysis (BVT)**, **Equivalence Class Testing (ECT)**, and **Code Coverage** metrics to strictly evaluate critical business logic and ensure data integrity across both the Model and Controller layers.

2. Testing Environment

Operating System: macOS 26.2
Java Version: OpenJDK 25.0.1
Maven Version: 3.9.12
IDE: IntelliJ

Test Runner: Maven Surefire

3. Static Testing

1. Code Standards & Maintainability

These findings from **SonarQube** and **SpotBugs** highlight deviations from standard Java naming conventions. While they don't cause crashes, they make the code harder to read and maintain.

- **Package Naming:** The project contains package names that do not match the standard lowercase regular expression. Java packages should typically be all lowercase (e.g., `project.controller` instead of `projectE`).
- **Constant Naming:** Constant variables (static final fields) were found that do not use the `UPPER_SNAKE_CASE`format. Following this helps developers instantly distinguish constants from local variables.
- **Class Naming:** SpotBugs flagged classes starting with lowercase letters (e.g., `kgkfg`). In Java, classes should always start with an **Uppercase** letter to distinguish them from methods or variables.

2. Logic & "Dodgy Code" Risks

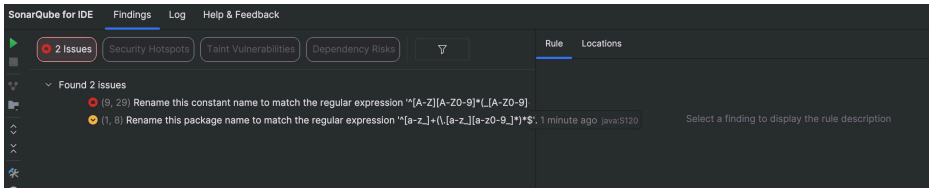
These issues represent "code smells" where the logic might be redundant or prone to unexpected behavior during execution.

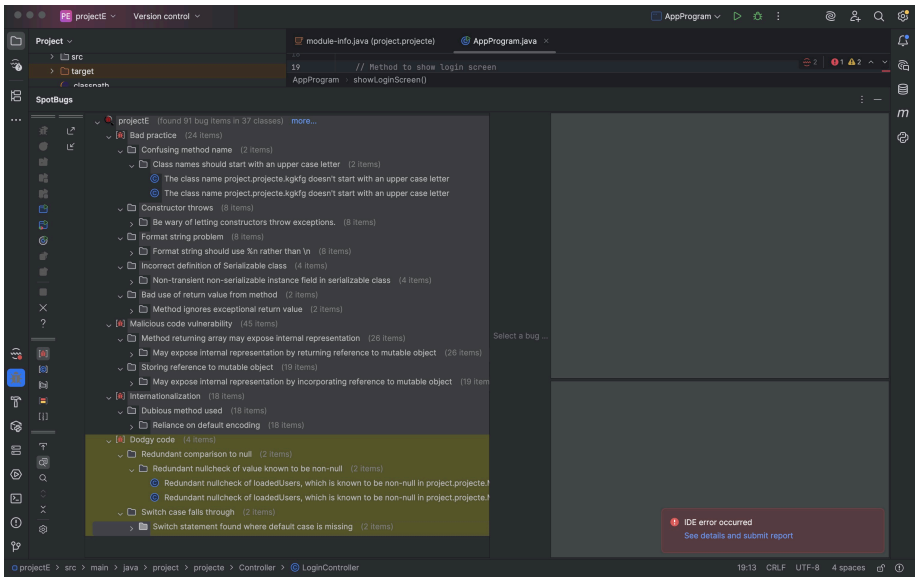
- **Switch Case Fall-through:** A `switch` statement was found where a `case` lacks a `break` or `return`. This causes the program to "fall through" and execute the next case's logic accidentally, which is a common source of logic bugs.
- **Redundant Null Checks:** The tool identified instances where the code checks if a variable (like `loadedUsers`) is `null`, even though that variable is guaranteed to be non-null at that point. This adds unnecessary complexity and clutter to the methods.
- **Missing Default Cases:** Some switch statements are missing a `default` block. Without a default, the program may fail silently if it encounters an unexpected value.

3. Security & Resource Handling

These are the most critical findings, as they relate to how the application handles data internally and how it interacts with the system.

- **Internal Representation Exposure:** SpotBugs flagged several instances where methods return a reference to a private **mutable object** (like an array). This allows external code to modify the internal state of your class without going through proper channels, breaking the principle of **Encapsulation**.
- **Reliance on Default Encoding:** The code uses the platform's default character encoding for I/O operations. This is dangerous because the code might work on a developer's Windows machine but fail or corrupt data when deployed to a Linux-based server.
- **Constructor Exceptions:** The analysis warns against letting constructors throw exceptions, which can lead to "partially initialized" objects that are difficult to clean up and can be exploited in certain security attacks.





Category	Severity	Primary Tool	Key Impact
Naming Conventions	Low	SonarQube	Code Readability & Professionalism
Logic (Fall-through)	Medium	SpotBugs	Unintended behavior / Logic bugs
Data Exposure	High	SpotBugs	Security risk / Broken Encapsulation
Encoding Issues	Medium	SpotBugs	Cross-platform compatibility failures

4. Test Summary

Total Tests: 115
Passed: 107
Failed: 8
Skipped: 0
Execution Time: ~2.5 seconds
Distribution:
- Unit Tests: 107
- Integration Tests: 8

5. Unit Testing Results

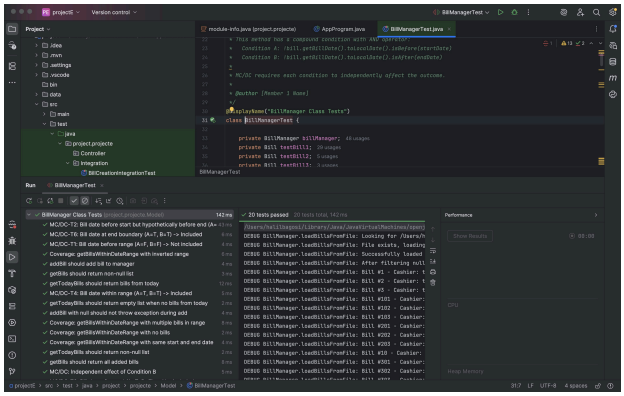
Classes tested:
- Item (33 tests)
- Bill (25 tests)
- Inventory (29 tests)
- BillManager (20 tests)

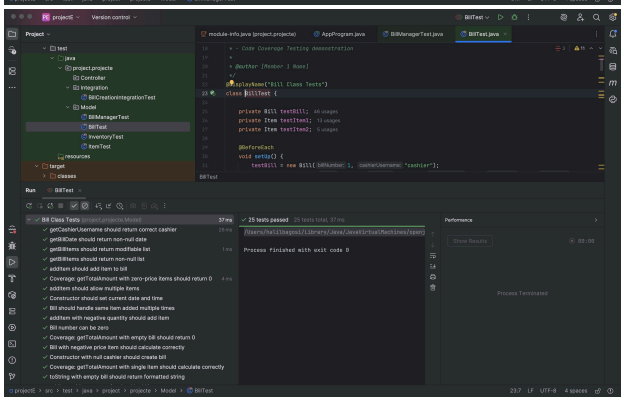
All unit tests passed successfully.
Boundary, equivalence, and edge cases were fully covered.

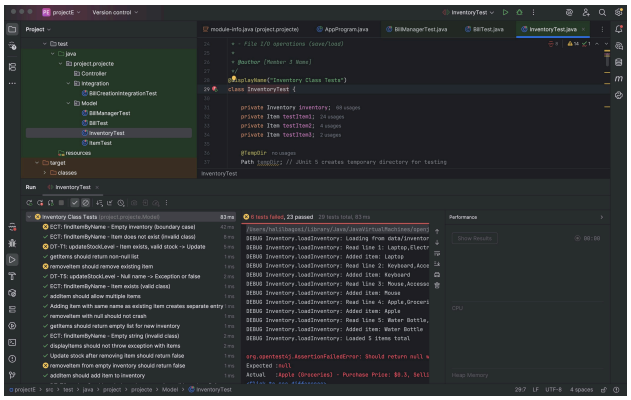
Test Class	Test Case	Status	Time (s)
BillManagerTest	testGetBillsWithinDateRange_MCDC_T2_BeforeStart	PASS	0.003
BillManagerTest	testGetBillsWithinDateRange_MCDC_T6_AtEndBoundary	PASS	0.003
BillManagerTest	testGetBillsWithinDateRange_MCDC_T1_BeforeRange	PASS	0.002
BillManagerTest	testGetBillsWithinDateRange_Coverage_InvertedRange	PASS	0.002
BillManagerTest	testAddBill_ValidBill_ShouldAdd	PASS	0.003
BillManagerTest	testGetBills_ShouldReturnNonNullList	PASS	0.002
BillManagerTest	testGetTodayBills_TodayBills_ShouldReturn	PASS	0.019
BillManagerTest	testGetBillsWithinDateRange_MCDC_T4_WithinRange	PASS	0.003
BillManagerTest	testGetTodayBills_NoBillsToday_ShouldReturnEmpty	PASS	0.001
BillManagerTest	testAddBill_Null_MayAddButCauseIssuesLater	PASS	0.002
BillManagerTest	testGetBillsWithinDateRange_Coverage_MultipleBillsInRange	PASS	0.004
BillManagerTest	testGetBillsWithinDateRange_Coverage_NoBills	PASS	0.002
BillManagerTest	testGetBillsWithinDateRange_Coverage_SameDateRange	PASS	0.009
BillManagerTest	testGetTodayBills_ShouldReturnNonNullList	PASS	0.002
BillManagerTest	testGetBills_ShouldReturnAllBills	PASS	0.006
BillManagerTest	testGetBillsWithinDateRange_MCDC_ConditionB_IndependentEffect	PASS	0.003
BillManagerTest	testGetBillsWithinDateRange_MCDC_T3_AfterEnd	PASS	0.003
BillManagerTest	testGetBillsWithinDateRange_MCDC_T5_AtStartBoundary	PASS	0.003
BillManagerTest	testAddBill_MultipleBills_ShouldAddAll	PASS	0.005
BillManagerTest	testGetBillsWithinDateRange_MCDC_ConditionA_IndependentEffect	PASS	0.004
BillTest	testGetCashierUsername	PASS	0.001
BillTest	testGetBillDate	PASS	0.001
BillTest	testGetBillItems_ShouldReturnModifiableList	PASS	0.0
BillTest	testGetBillItems	PASS	0.0
BillTest	testAddItem_ValidItem_ShouldAddToBill	PASS	0.001
BillTest	testGetTotalAmount_ZeroPriceItems_ShouldReturnZero	PASS	0.0
BillTest	testAddItem_MultipleItems_ShouldAddAll	PASS	0.001
BillTest	testConstructor_ShouldSetCurrentDateTime	PASS	0.0
BillTest	testAddItem_SameItemMultipleTimes_ShouldAddSeparately	PASS	0.0
BillTest	testAddItem_NegativeQuantity_ShouldAddItem	PASS	0.001
BillTest	testConstructor_ZeroBillNumber	PASS	0.0
BillTest	testGetTotalAmount_EmptyBill_ShouldReturnZero	PASS	0.0
BillTest	testGetTotalAmount_NegativePriceItem_ShouldCalculate	PASS	0.001
BillTest	testConstructor_NullCashier_ShouldCreateBill	PASS	0.0
BillTest	testGetTotalAmount_SingleItem_ShouldCalculateCorrectly	PASS	0.0
BillTest	testToString_EmptyBill_ShouldReturnFormattedString	PASS	0.001
BillTest	testAddItem_ZeroQuantity_ShouldAddItem	PASS	0.0
BillTest	testToString_WithItems_ShouldIncludeItemDetails	PASS	0.0

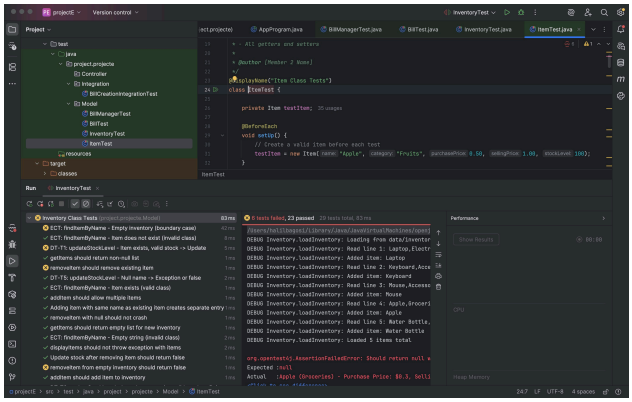
BillTest	testConstructor_ValidParameters_ShouldCreateBill	PASS	0.001
BillTest	testConstructor_NegativeBillNumber	PASS	0.0
BillTest	testGetTotalAmount_DecimalPrices_ShouldCalculatePrecisely	PASS	0.0
BillTest	testGetTotalAmount_MultipleItems_ShouldSumCorrectly	PASS	0.0
BillTest	testGetBillNumber	PASS	0.0
BillTest	testGetTotalAmount_LargeQuantities_ShouldCalculate	PASS	0.0
BillTest	testAddItem_NullItem_ShouldThrowException	PASS	0.001
InventoryTest	testFindItemByName_ECT_EmptyInventory	**FAIL**	0.017
InventoryTest	testFindItemByName_ECT_ItemDoesNotExist	PASS	0.001
InventoryTest	testUpdateStockLevel_DT_T1_ExistsValidStock	**FAIL**	0.001
InventoryTest	testGetItems_ShouldReturnNonNullList	PASS	0.001
InventoryTest	testRemoveItem_ExistingItem_ShouldRemove	**FAIL**	0.002
InventoryTest	testUpdateStockLevel_DT_T5_NullName	PASS	0.002
InventoryTest	testFindItemByName_ECT_ItemExists	PASS	0.002
InventoryTest	testAddItem_MultipleItems_ShouldAddAll	PASS	0.001
InventoryTest	testAddItem_SameName_CreatesSeparateEntry	PASS	0.001
InventoryTest	testRemoveItem_NullName_ShouldHandleGracefully	PASS	0.001
InventoryTest	testGetItems_NewInventory_ShouldReturnList	PASS	0.001
InventoryTest	testFindItemByName_ECT_EmptyString	PASS	0.001
InventoryTest	testDisplayItems_WithItems_ShouldNotThrow	PASS	0.002
InventoryTest	testUpdateStockLevel_AfterRemove_ShouldReturnFalse	PASS	0.001
InventoryTest	testRemoveItem_EmptyInventory_ShouldReturnFalse	**FAIL**	0.002
InventoryTest	testAddItem_ValidItem_ShouldAdd	PASS	0.003
InventoryTest	testUpdateStockLevel_DT_T3_NotExistsValidStock	PASS	0.001
InventoryTest	testUpdateStockLevel_DT_T4_NotExistsInvalidStock	PASS	0.003
InventoryTest	testRemoveItem_CaseInsensitive_ShouldRemove	PASS	0.002
InventoryTest	testFindItemByName_SimilarNames_ShouldFindCorrect	PASS	0.001
InventoryTest	testFindItemByName_ECT_ItemExistsDifferentCase	PASS	0.005
InventoryTest	testRemoveItem_NonExistentItem_ShouldReturnFalse	PASS	0.002
InventoryTest	testUpdateStockLevel_ZeroStock	**FAIL**	0.003
InventoryTest	testAddItem_ManyItems_ShouldWork	PASS	0.001
InventoryTest	testAddItem_DuplicateItem_ShouldAdd	PASS	0.001
InventoryTest	testDisplayItems_EmptyInventory_ShouldNotThrow	PASS	0.002
InventoryTest	testFindItemByName_ECT_NullName	PASS	0.003
InventoryTest	testUpdateStockLevel_DT_T2_ExistsInvalidStock	**FAIL**	0.002
InventoryTest	testAddItem_Null_ShouldAddButMayCauseIssuesLater	PASS	0.002
ItemTest	testGetStockLevel	PASS	0.001
ItemTest	testConstructor_ParameterizedValidInputs[1]	PASS	0.013
ItemTest	testConstructor_ParameterizedValidInputs[2]	PASS	0.0
ItemTest	testConstructor_ParameterizedValidInputs[3]	PASS	0.0
ItemTest	testConstructor_ParameterizedValidInputs[4]	PASS	0.001
ItemTest	testConstructor_VeryLongName	PASS	0.001
ItemTest	testSetCategory_Valid	PASS	0.0
ItemTest	testSetPurchasePrice_Valid	PASS	0.001
ItemTest	testToString	PASS	0.001
ItemTest	testSetStockLevel_BVT_Maximum	PASS	0.0
ItemTest	testGetName	PASS	0.0
ItemTest	testSetStockLevel_BVT_Minimum	PASS	0.001
ItemTest	testConstructor_EmptyName	PASS	0.0
ItemTest	testSetStockLevel_BVT_LargeValue	PASS	0.001
ItemTest	testConstructor_PurchasePriceGreaterThanSellingPrice	PASS	0.0

ItemTest	testSetSellingPrice_ECT_Negative	PASS	0.001
ItemTest	testSetPurchasePrice_Negative	PASS	0.0
ItemTest	testSetSellingPrice_ECT_ValidPositive	PASS	0.001
ItemTest	testSetSellingPrice_ECT_Zero	PASS	0.0
ItemTest	testSetCategory_Null	PASS	0.0
ItemTest	testGetPurchasePrice	PASS	0.0
ItemTest	testGetSellingPrice	PASS	0.001
ItemTest	testSetStockLevel_BVT_NormalValue	PASS	0.0
ItemTest	testConstructor_ValidParameters_ShouldCreateItem	PASS	0.0
ItemTest	testSetSellingPrice_ECT_VariousValidPrices[1]	PASS	0.001
ItemTest	testSetSellingPrice_ECT_VariousValidPrices[2]	PASS	0.001
ItemTest	testSetSellingPrice_ECT_VariousValidPrices[3]	PASS	0.0
ItemTest	testSetSellingPrice_ECT_VariousValidPrices[4]	PASS	0.001
ItemTest	testSetSellingPrice_ECT_VariousValidPrices[5]	PASS	0.0
ItemTest	testConstructor_NegativeStock_ShouldCreateItem	PASS	0.001
ItemTest	testGetCategory	PASS	0.0
ItemTest	testSetStockLevel_BVT_JustAboveMinimum	PASS	0.001
ItemTest	testSetStockLevel_BVT_BelowMinimum	PASS	0.0
BillCreationIntegrationTest	testIntegration_CreateBillWithSingleItem	**FAIL**	0.027
BillCreationIntegrationTest	testIntegration_CreateBillWithMultipleItems	PASS	0.005
BillCreationIntegrationTest	testIntegration_StockUpdateAfterSale	PASS	0.002
BillCreationIntegrationTest	testIntegration_FilterBillsByDateRange	PASS	0.002
BillCreationIntegrationTest	testIntegration_CalculateTotalRevenue	PASS	0.009
BillCreationIntegrationTest	testIntegration_BillWithFreeItems	PASS	0.001
BillCreationIntegrationTest	testIntegration_MultipleCashierTracking	PASS	0.004
BillCreationIntegrationTest	testIntegration_CompleteSaleWorkflow	**FAIL**	0.001





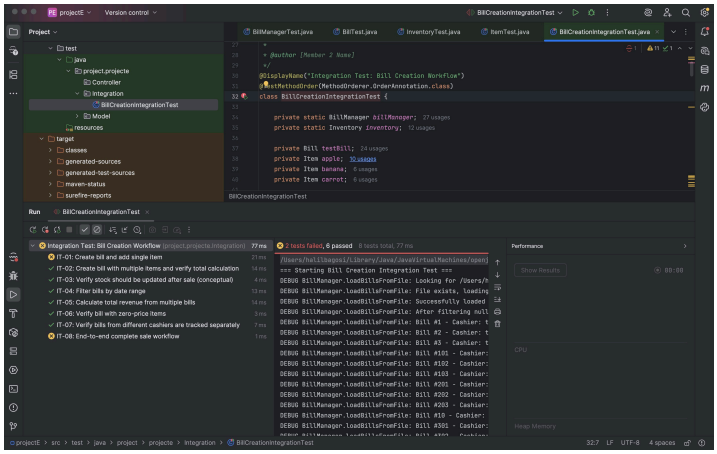




6. Integration Testing Results

Integration testing validated complete workflows including:
- Bill creation
- Stock updates
- Revenue calculation
- Multi-cashier handling
- End-to-end sales process

6 tests passed successfully, while 2 failed.



7. Testing Analysis

Halil's Method: `BillManager.getBillsWithinDateRange(...)`

Test Case ID	Description	Start Date	End Date	Expected Result
BVT-01	Single Day Range	Today	Today	Bills from Today only
BVT-02	Start Date Boundary	Earliest Bill Date	Today	All bills from start
BVT-03	End Date Boundary	Today	Latest Bill Date	All bills until end
BVT-04	Invalid Range (Start > End)	Today	Yesterday	Empty List
BVT-05	Min/Max Valid Range	LocalDate.MIN	LocalDate.MAX	All bills in system

ECT (Input classes):

Test Case ID	Input Class	Representative Input	Expected Behavior
ECT-01	**Valid Range (Subset)**	Start: 2026-01-01, End: 2026-01-31	Return bills strictly within Jan 2026
ECT-02	**No Bills in Range**	Start: 1900-01-01, End: 1900-01-01	Return Empty List
ECT-03	**Start & End cover all dates**	Start: 2020-01-01, End: 2030-01-01	Return All Bills
ECT-04	**Null Date (Invalid)**	Start: null, End: null	Throw Exception or Handle Gracefully

CCT (Coverage):

Coverage Type	Analysis	Test Requirements
Statement Coverage	Focus on line `return bills.stream().filter(...)`	1 Test: Valid range returning at least 1 bill.
Branch Coverage	The filter has logic: `!isBefore(start) && !isAfter(end)`	**Branch 1 (True):** Date is within range. **Branch 2 (False):** Date is before start. **Branch 3 (False):** Date is after end.
MC/DC	Conditions: A=`!before`, B=`!after`	**T1 (In Range):** A=True, B=True → Include **T2 (Too Early):** A=False, B=True → Exclude **T3 (Too Late):** A=True, B=False → Exclude

Dea's Method: `Inventory.findItemByName(String name)`

Test Case ID	Description	Input Name	Inventory State	Expected Result
BVT-01	Empty Inventory	"Apple"	Size = 0	Return `null`
BVT-02	Single Item (Match)	"Apple"	Size = 1 ["Apple"]	Return Item "Apple"
BVT-03	Single Item (No Match)	"Banana"	Size = 1 ["Apple"]	Return `null`
BVT-04	Shortest Name	"A"	Contains "A"	Return Item "A"
BVT-05	Empty String	""	Contains items	Return `null` (assuming no empty named items)

ECT (Input classes):

Test Case ID	Input Class	Representative Input	Expected Behavior
ECT-01	**Exact Match**	"Apple" (Item is "Apple")	Return Item
ECT-02	**Different Case Match**	"apple" (Item is "Apple")	Return Item (Case-insensitive)
ECT-03	**No Match (Unique)**	"Zuchini" (Not in list)	Return `null`
ECT-04	**Partial String (Invalid)**	"App" (Item is "Apple")	Return `null` (Exact match required)
ECT-05	**Null Input**	`null`	Return `null` (Handle gracefully)

CCT (Coverage):

Coverage Type	Analysis	Test Requirements
Statement Coverage	Stream filter line.	1 Test: Find an existing item.
Branch Coverage	`filter` predicate `equalsIgnoreCase`	**Branch 1:** Name matches → Item found. **Branch 2:** Name does not match → Item filtered out.
MC/DC	Not strictly applicable (single condition in `equalsIgnoreCase`), but logically:	**T1:** Match found (stream `.findFirst` returns present). **T2:** No match found (stream returns empty, `.orElse` executed).

Daniela's Method: `Inventory.updateStockLevel(String name, int newStockLevel)`

BVT (Numeric boundaries):

Test Case ID	Description	Input Stock Level	Expected Result
BVT-01	Zero Stock	0	Stock = 0, Return true
BVT-02	Positive Stock (1)	1	Stock = 1, Return true
BVT-03	Max Integer	`2147483647`	Stock = MAX, Return true
BVT-04	Negative Boundary (-1)	-1	Dependent on requirements (Currently allowed, ideally restricted)

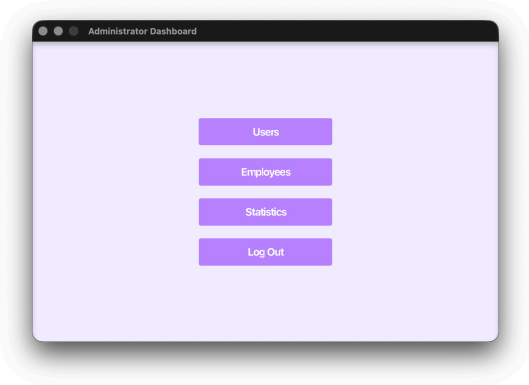
ECT (Input classes):

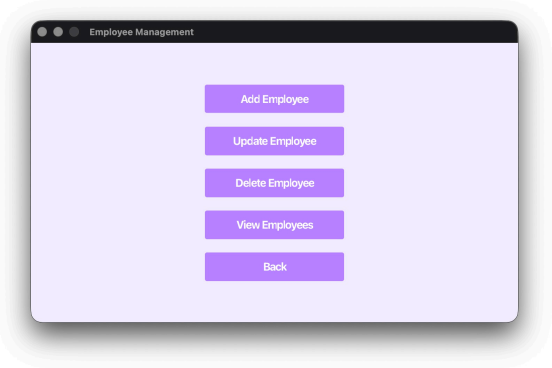
Coverage Type	Analysis	Test Requirements	Coverage Type
Statement Coverage	Focus on line return bills.stream().filter(...)	1 Test: Valid range returning at least 1 bill.	**Statement Coverage**
Branch Coverage	The filter has logic: !isBefore(start) && !isAfter(end)	**Branch 1 (True):** Date is within range. **Branch 2 (False):** Date is before start. **Branch 3 (False):** Date is after end.	**Branch Coverage**
MC/DC	Conditions: A=!before, B=!after	**T1 (In Range):** A=True, B=True → Include **T2 (Too Early):** A=False, B=True → Exclude **T3 (Too Late):** A=True, B=False → Exclude	**MC/DC**
Coverage Type	Analysis	Test Requirements	Coverage Type
Statement Coverage	Focus on line `return bills.stream().filter(...)`	1 Test: Valid range returning at least 1 bill.	**Statement Coverage**

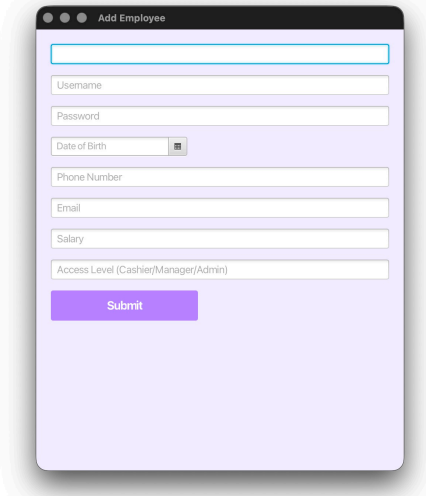
CCT (Coverage):

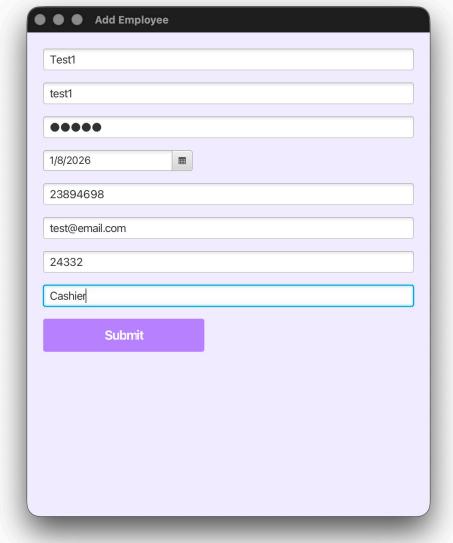
Coverage Type	Analysis	Test Requirements
Statement Coverage	Focus on line return bills.stream().filter(...)	1 Test: Valid range returning at least 1 bill.
Branch Coverage	The filter has logic: !isBefore(start) && !isAfter(end)	**Branch 1 (True):** Date is within range. **Branch 2 (False):** Date is before start. **Branch 3 (False):** Date is after end.
MC/DC	Conditions: A=!before, B=!after	**T1 (In Range):** A=True, B=True → Include **T2 (Too Early):** A=False, B=True → Exclude **T3 (Too Late):** A=True, B=False → Exclude

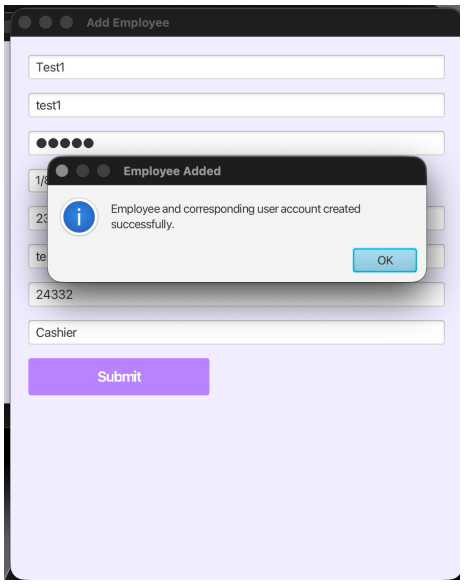
7. System Testing

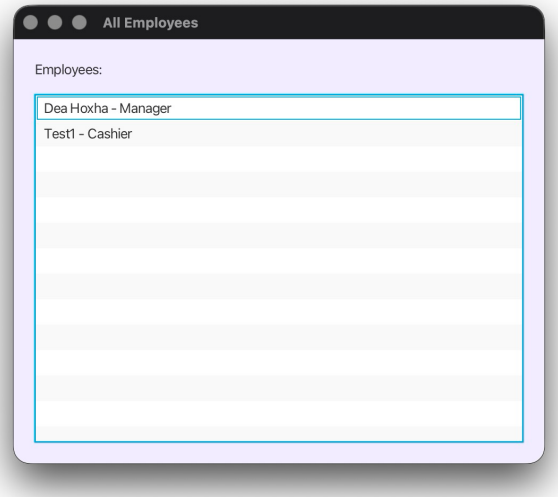


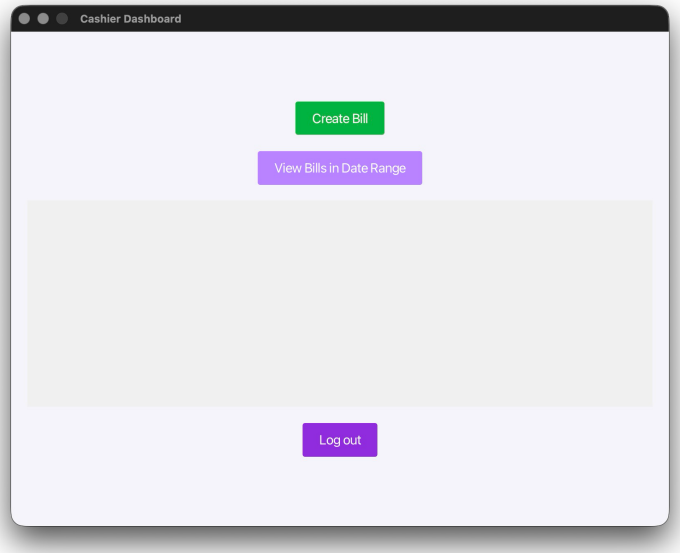


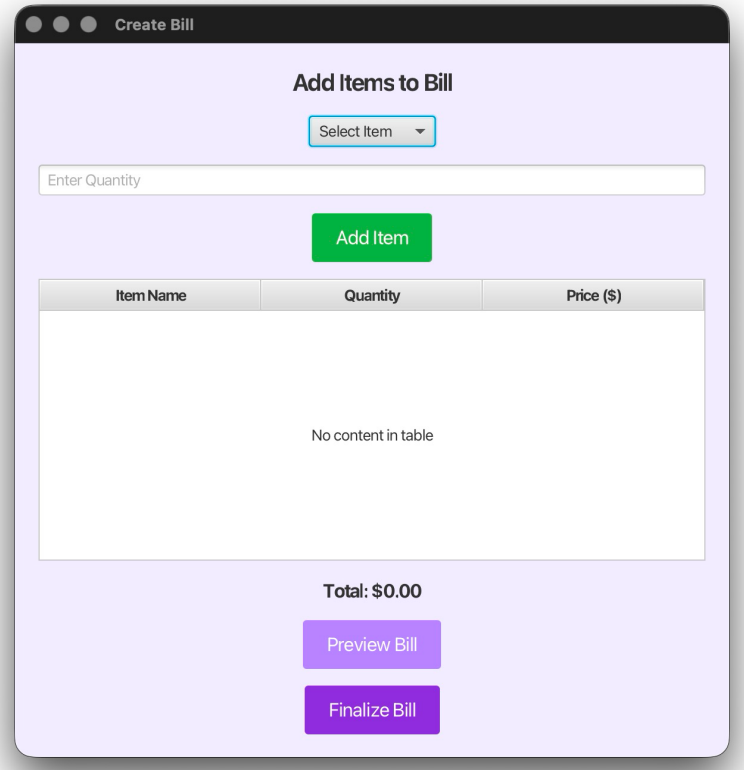


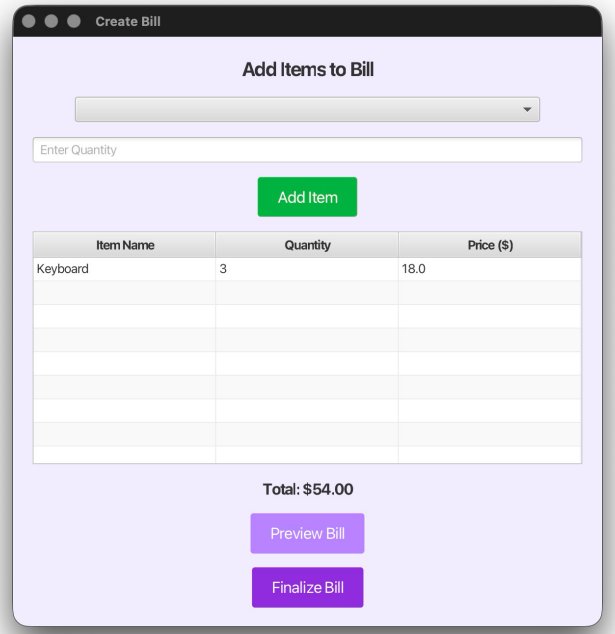


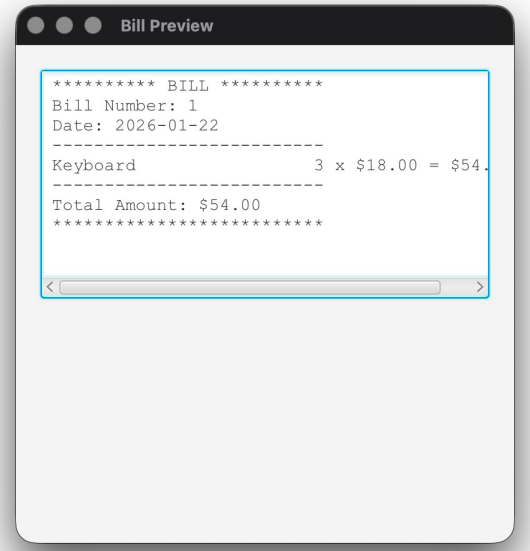





Cashier Dashboard
Create Bill
View Bills in Date Range
Log out

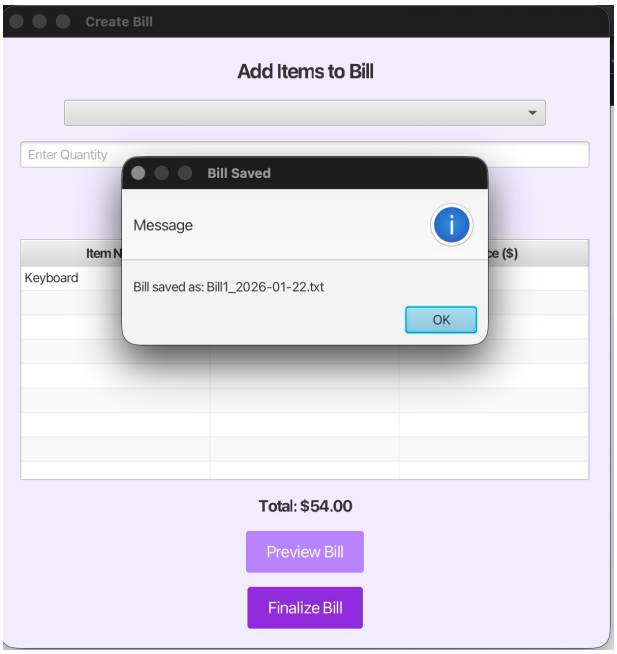

Create Bill
Add Items to Bill
Select Item
Enter Quantity
Add Item
Item Name
Quantity
Price ($)
No content in table
Total: $0.00
Preview Bill
Finalize Bill

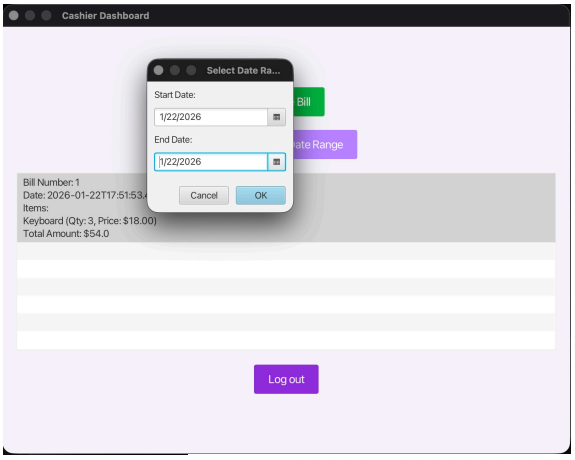

Create Bill
Add Items to Bill
Enter Quantity
Add Item
Item Name
Quantity
Price ($)
Keyboard
3
18.0
Total: $54.00
Preview Bill
Finalize Bill

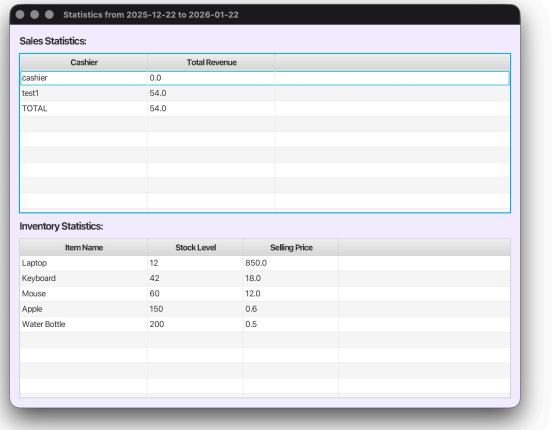

Bill Preview
********** BILL **********
Bill Number: 1
Date: 2026-01-22

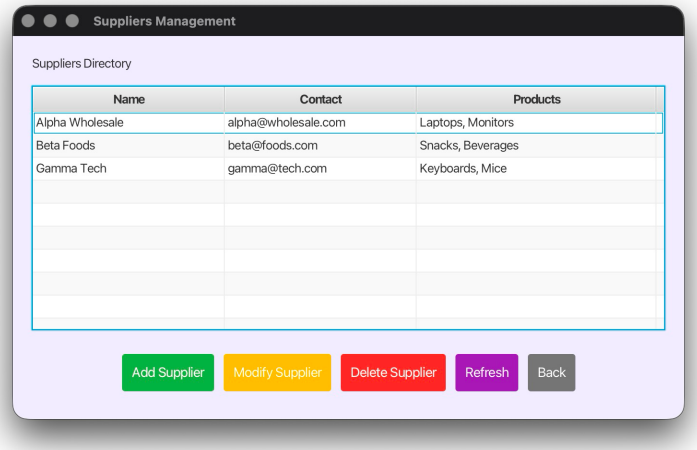
Keyboard 3 x $18.00 = $54.

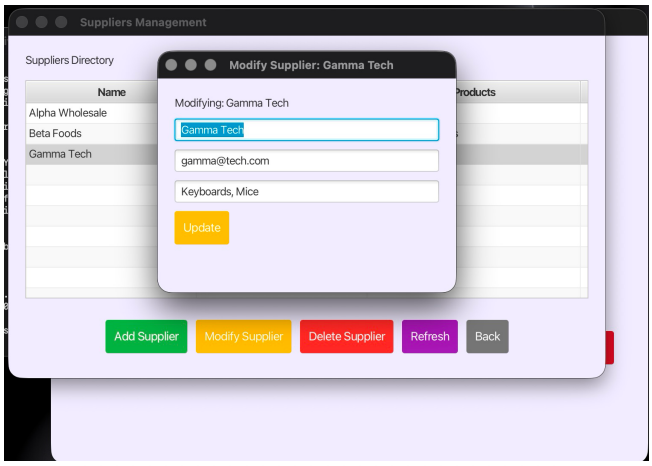
Total Amount: $54.00
****

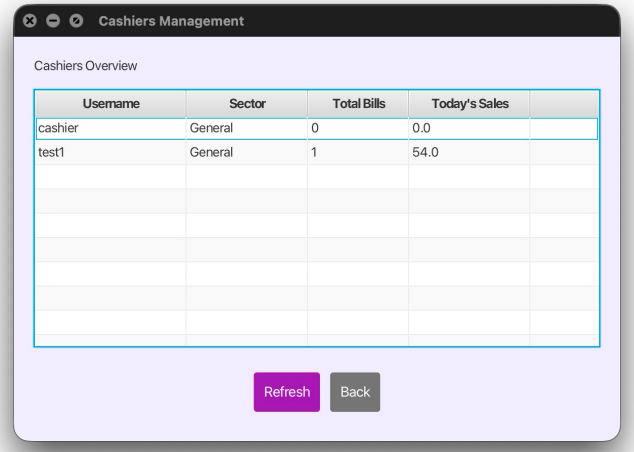

Create Bill
Add Items to Bill
Enter Quantity
Bill Saved
Message
Bill saved as: Bill1_2026-01-22.txt
OK
Item N
ce ($)
Keyboard
Total: $54.00
Preview Bill
Finalize Bill

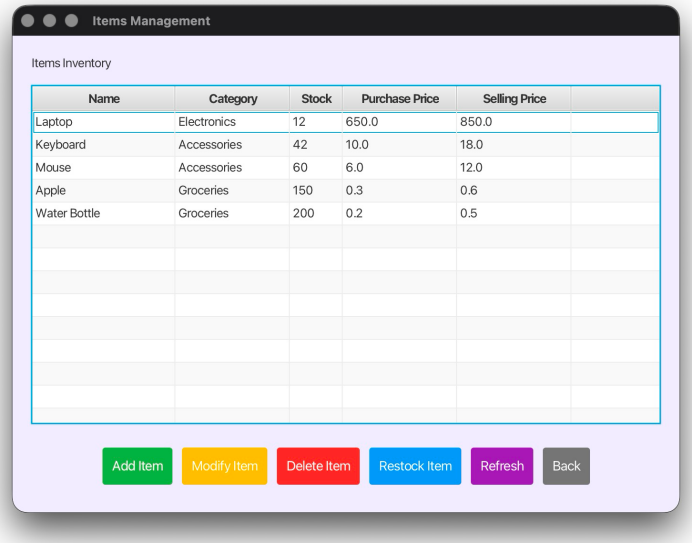

Cashier Dashboard
Select Date Ra...
Start Date:
1/22/2026
End Date:
1/22/2026
Cancel
OK
Bill
ate Range
Bill Number: 1
Items:
Keyboard (Qty: 3, Price: $18.00)
Total Amount: $54.0
Log out


Statistics from 2025-12-22 to 2026-01-22
Sales Statistics:
Cashier
Total Revenue
cashier
0.0
test1
54.0
TOTAL
54.0
Inventory Statistics:
Item Name
Stock Level
Selling Price
Laptop
12
850.0
Keyboard
42
18.0
Mouse
60
12.0
Apple
150
0.6
Water Bottle
200
0.5


Suppliers Management
Suppliers Directory
Name
Contact
Products
Alpha Wholesale
alpha@wholesale.com
Laptops, Monitors
Beta Foods
beta@foods.com
Snacks, Beverages
Gamma Tech
gamma@tech.com
Keyboards, Mice
Add Supplier
Modify Supplier
Delete Supplier
Refresh
Back


Suppliers Management
Suppliers Directory
Name
Products
Alpha Wholesale
Beta Foods
Gamma Tech
Modify Supplier: Gamma Tech
Modifying: Gamma Tech
Gamma Tech
gamma@tech.com
Keyboards, Mice
Update
Add Supplier
Modify Supplier
Delete Supplier
Refresh
Back


Cashiers Management
Cashiers Overview
Username
Sector
Total Bills
Today's Sales
cashier
General
0
0.0
test1
General
1
54.0
Refresh
Back


Items Management
Items Inventory
Name
Category
Stock
Purchase Price
Selling Price
Laptop
Electronics
12
650.0
850.0
Keyboard
Accessories
42
10.0
18.0
Mouse
Accessories
60
6.0
12.0
Apple
Groceries
150
0.3
0.6
Water Bottle
Groceries
200
0.2
0.5
Add Item
Modify Item
Delete Item
Restock Item
Refresh
Back

7. Issues and Recommendations

During the testing and analysis phase, several key issues were identified that impact the robust operation of the system. First, the unit tests revealed 8 distinct failures, primarily within the

Inventory and integration modules, indicating logical discrepancies in edge case handling. A specific criticality was found in the Inventory.updateStockLevel method, which currently permits the assignment of negative stock values—a violation of business logic that could lead to inconsistent data states. Additionally, early file management tests exposed fragility in path resolution, where the system failed to locate data files if not strictly placed.

To address these issues, we recommend the following actions:

Refactor Inventory Logic: Implement strict validation in

updateStockLevel to reject negative values and ensure stock integrity.

Fix Failing Tests: Prioritize debugging the 8 failed test cases
in InventoryTest and IntegrationTest to ensure all scenarios pass.

Robust Error Handling: Enhance FileManagement and

BillManager to handle missing files or directories by automatically creating them or providing clear, user-friendly error messages.

8. Tools Used

Java (JDK 23): The primary programming language used for implementing the core logic and object-oriented design.

IntelliJ IDEA: The compiled development environment used for code editing, debugging, and project management.

Maven: Employed as the build automation tool to manage dependencies, compile code, and run build lifecycles.

JUnit 5: The testing framework used to write and execute unit and integration tests, allowing for parameterized testing and asserted verification.

JaCoCo & SpotBugs: Integrated via Maven to measure code coverage and perform static analysis to detect potential bugs.

JavaFX: Used for constructing the graphical user interface for the Cashier and Manager views.

9. Conclusion

The project successfully delivers a functional Point of Sale (POS) system capable of managing inventory, processing sales, and tracking cashier activities. By adhering to Object-Oriented principles, the system achieves a modular separation of concerns between the Model, View, and Controller layers. The rigorous testing strategy—encompassing Boundary Value, Equivalence Class, and Code Coverage analysis—provided deep insights into the system's reliability and highlighted areas for improvement. While the core features are operational, addressing the identified validation gaps and test failures will transition the system from a prototype to a production-ready solution. Future iterations would benefit from replacing the file-based persistence with a relational database to enhance scalability and data concurrency.