



# East West University

## Project Report of

# TESTER SQUAD

**Course Title** : Software Testing & Quality Assurance

**Course Code** : CSE430

**Section** : 02

**Semester** : Spring2024

### Submitted To

**Anika Tabassum**

Lecturer

Department of CSE

East West University

### Submitted By

Student Name	Student ID
Nowshin Tasnia	2020-1-60-197
Krittika Roy	2020-1-60-065
Nabiha Tahsin	2020-2-60-087
Mahbuba Yasmin	2020-2-60-170

**Date Of Submission : 26-05-2024**

# Table of Content

Introduction.....	3
Objective.....	3
Project Scope.....	3
Stakeholders.....	4
Testing Process.....	4
Main Functions And Their Test Cases.....	5
1. Customer - 26 functions.....	5
Table 1: Test Case Table for Customer Class.....	5
2. CustomerManager - 13 functions.....	10
Table 2: Test Case Table for Customer Manager Class.....	10
3. Employee - 11 functions.....	13
Table 3: Test Case Table for EmployeeClass.....	13
4. EmployeeManager - 9 functions.....	15
Table 4: Test Case Table for EmployeeManager Class.....	15
5. Inventory - 15 functions.....	19
Table 5: Test Case Table for Inventory Class.....	19
6. InventoryFileManager - 4 functions.....	24
Table 6: Test Case Table for Inventory File Manager Class.....	24
7. Order - 12 functions.....	25
Table 7: Test Case Table for Order Class.....	25
8. OrderManager - 13 functions.....	27
Table 8: Test Case Table for Order Manager Class.....	27
9. Product - 11 functions.....	29
Table 9: Test Case Table for Product Class.....	29
10. ShoppingCart - 12 functions.....	34
Table 10: Test Case Table for Shopping Cart Class.....	34
11. Supplier - 11 functions.....	35
Table 11: Test Case Table for Supplier Class.....	35
12. Transaction - 14 functions.....	38
Table 12: Test Case Table for Transaction Class.....	38
13. TransactionManager - 11 functions.....	41
Table 13: Test Case Table for Transaction Manager Class.....	41
14. Warehouse - 9 functions.....	44
Table 14: Test Case Table for Warehouse Class.....	44
Expected Outcomes.....	45
Summarization.....	45
Conclusion.....	46

# Introduction

The main focus of the discipline of inventory management is on defining the location and form of stocked products. It is necessary to do so before the normal and scheduled course of production and material stock at various points within a facility or throughout numerous sites of a supply network. The CSE430 Inventory Management System is a Java-based solution that streamlines inventory control for businesses. It offers 14 classes, each focusing on specific aspects of inventory management, including customer details, employee information, product data, and warehouse stock levels. The system facilitates customer orders, tracks transactions, and offers supplier management and data persistence. Its modular design allows for future customization and expansion, enabling businesses to gain real-time inventory visibility, automate tasks, and make data-driven decisions. The CSE430 Inventory Management System relies on thorough testing, with each of its 14 Java classes requiring a dedicated test class built within Eclipse. JUnit, a popular testing framework, integrates with Eclipse, allowing for detailed examination of individual functionalities. Code coverage tools in Eclipse identify the extent of original code testing. Iterative refinement and fixing of test cases ensure a robust inventory management system, fostering confidence in its functionality.

## Objective

The major purpose of creating test cases for the CSE430 Inventory Management System is to validate its functionality and reliability. These test cases cover a wide range of fundamental subjects. At first, the tester certifies that critical functionality such as product addition, search, and update functions properly. Second, data integrity is assessed by observing how the system handles incorrect inputs and prevents unneeded data modification. Third, the test cases look at edge circumstances, such as empty inventories or excessive product numbers, to identify and address any issues. The test cases aim to improve system dependability by detecting problems while monitoring demand. Finally, the user experience is assessed to ensure that information is clearly displayed and error messages are informative. By comprehensively examining these aspects, the test cases will solidify the inventory management system's functionality, reliability, and user-friendliness.

## Project Scope

- 1. Product Management:**
  - 1.1. Add, update, and remove products.
  - 1.2. Search and filter products by ID, name, price range, or other relevant criteria.
- 2. Inventory Management:**
  - 2.1. Track product stock levels.
  - 2.2. Check product availability for a given quantity.
  - 2.3. Identify products falling below a defined stock threshold.
- 3. Inventory Analysis:**
  - 3.1. Calculate total inventory value.
  - 3.2. Calculate average product price.
- 4. Error Handling:**

- 4.1. Handle invalid user input (e.g., negative quantities, non-numeric characters in product names).
- 4.2. Provide informative error messages to guide users.
5. **Assumptions and Exclusions:**
  - 5.1. The system focuses on managing a single inventory location.
  - 5.2. Products are assumed to have unique IDs.
  - 5.3. Discounts or product variants are not included in this scope.
  - 5.4. User authentication or access control mechanisms are not considered.
  - 5.5. Reporting or exporting inventory data is outside the scope.
6. **Testing Strategy:**
  - 6.1. Unit tests will be conducted to verify individual functionalities.
  - 6.2. Integration tests will ensure modules work together seamlessly.
  - 6.3. System tests will evaluate the overall system behavior from a user's perspective.
7. **Deliverables:**
  - 7.1. Functional inventory management system.
  - 7.2. Comprehensive test suite covering identified functionalities.
  - 7.3. Documentation outlining system features, assumptions, and limitations.

## Stakeholders

1. **Project Manager:** Oversees project execution and ensures alignment with business objectives.
2. **Test Lead:** Manages the testing squads and coordinates testing activities.
3. **Developers:** Collaborate with testers to fix identified issues and enhance software quality.
4. **Quality Assurance Team:** Ensures that all testing standards and protocols are maintained.
5. **End Users:** Benefit from improved software quality and reliability.

## Testing Process

We have tested the functionalities of the project, which is called Unit Testing. For Unit Testing we have used Eclipse IDE for Java Developers and for testing purposes the external libraries of JUnit 4 and JUnit 5.



Figure 1: Software and Tools

# Main Functions And Their Test Cases

## 1. Customer - 26 functions

- 1.1. Customer(int id, String name, String email, String address)-----3
- 1.2. purchaseItem(Product product, int quantity)-----5
- 1.3. applyDiscount(double price, int quantity)-----3
- 1.4. returnItem(Product product, int quantity)-----3
- 1.5. getPurchasedItems()---1
- 1.6. getTotalItemsPurchased()-----1
- 1.7. getTotalAmountSpent()-----1
- 1.8. hasPurchasedItem(Product product)-----1
- 1.9. clearAllPurchasedItems()-----1
- 1.10. hasPurchasedMultipleItems()-----1
- 1.11. isFrequentShopper()-----1
- 1.12. calculateAveragePurchaseQuantity()-----1
- 1.13. hasHighSpending()-----1
- 1.14. getId()-----1
- 1.15. setId(int id)-----1
- 1.16. getEmail()-----1
- 1.17. setEmail(String email)-----1
- 1.18. getAddress()-----1
- 1.19. setAddress(String address)-----1
- 1.20. getTotalPurchases()
- 1.21. getBalance()
- 1.22. getStatus()
- 1.23. setStatus(CustomerStatus status)
- 1.24. addBalance(double amount)
- 1.25. removeBalance(double amount)
- 1.26. isActive()

**Table 1: Test Case Table for Customer Class**

Main Class Name: Customer			Test Class Name: TestCustomer	
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
Function name: Customer(int id, String name, String email, String address)				
1	testCustomer()	(2020160065, "Krittika", "k2001roy@gma	(2020160065, "Krittika", "k2001roy@gma	(2020160065, "Krittika", "k2001roy@gmai

		il.com","Dhamrai");	il.com","Dhamrai");	l.com","Dhamrai");
2	testCustomer1()	(2020160197, "Nowshin", "lowshin@gmail.com","Dhaka")	(2020160197, "Nowshin", "lowshin@gmail.com","Dhaka")	(2020160197, "Nowshin", "lowshin@gmail.com","Dhaka")
3	testCustomer2()	(2020160001, "Roy", "roykrittika@gmail.com","Manikganj")	(2020160001, "Roy", "roykrittika@gmail.com","Manikganj")	(2020160001, "Roy", "roykrittika@gmail.com","Manikganj")
4	testInvalidEmailFormat()	(2020160002, "John", "invalid_email", "Dhaka")	invalid_email	invalid_email
5	testEmptyName()	(2020160003, "", "k2001roy@gmail.com", "Dhaka")	""	""
6	testEmptyEmail()	(2020160004, "krittika2", "", "81/A loharpul")	""	""
9	testEmptyAddress()	(2020160004, "krittika2", "k2001roy@gmail.com", "")	""	""
8	testIDUpdate()	(2020160004, "Alice", "alice@gmail.com", "Chittagong")	2020160066	2020160066
9	testEmailUpdate()	(2020160004, "Alice", "alice@gmail.com", "Chittagong")	krittikaroy2020@gmail.com	krittikaroy2020@gmail.com
10	testAddressUpdate()	(2020160004, "Alice",	Comilla	Comilla

		"alice@gmail.com", "Chittagong")		
11	testIdMismatch()	(2020160005, "John", "john@gmail.com", "Sylhet")	2020160197	2020160197
12	testEmailMismatch()	(2020160005, "John", "john@gmail.com", "Sylhet")	k2001roy@gmail.com	k2001roy@gmail.com
13	testAdressMismatch()	(2020160005, "John", "john@gmail.com", "Sylhet")	Dhaka	Dhaka
<b>Function Name: purchaseItem(Product product, int quantity)</b>				
14	testValidPurchase()	(Laptop, 1000)	(Laptop,2)	(Laptop, 2)
15	testExistingProductUpdate()	(Phone, 500)	(Phone, 5)	(Phone, 5)
16	testNewProductPurchase()	(Tablet, 400)	(Tablet,1)	(Tablet,1)
17	testNonExistingProductPurchase()	(Tablet, 400)	(Tablet, 2)	(Tablet, 2)
<b>Function name: applyDiscount(double price, int quantity)</b>				
18	testApplyDiscountLessThan10Items()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 4, 1))	(10002.75, 5)	(10002.75, 5)
19	testApplyDiscountExactly10Items()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 9, 1))	(180049.5, 10)	(180050, 10 )

20	testApplyDiscountMoreThan10Items()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 10, 1))LocalDate.of(2024, 5, 1)	(27007.425, 15)	(27007, 15)
21	testCalculateTotalPriceInCart()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2024, 5, 1))	(10002.75, 5)	(10002.75, 5)
<b>Function name: returnItem(Product product, int quantity)</b>				
22	testAddItemToCart()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 5, 1))	(Headphones, 5)	(Headphones, 5)
23	testReturnItemWithEnoughQuantityInStock()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2024, 5, 1))	(Headphones, 5)	(Headphones, 5)
24	testReturnItemWithExactQuantityInStock()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2024, 5, 1))	0	0
25	testReturnItemWithQuantityMoreThanInStock()	(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2024, 5, 1))	IllegalArgument Exception	IllegalArgumentException
<b>Function name: getPurchasedItems(), getTotalItemsPurchased() and getTotalAmountSpent()</b>				

26	testGetPurchasedItems()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(20 24, 5, 1)); Product(5, "Smartphone", 50000.00, 200, "vivo", LocalDate.of(20 26, 5, 1));	(Headphones, 2) (Smartphone, 1)	(Headphones, 2) (Smartphone, 1)
27	testGetTotalItemsPurchased()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(20 24, 5, 1)); Product(5, "Smartphone", 50000.00, 200, "vivo", LocalDate.of(20 26, 5, 1));	(Headphones, 2) (Smartphone, 1)	TotalItemsPurchased: 3
28	testGetTotalAmountSpent()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(20 24, 5, 1)); Product(5, "Smartphone", 50000.00, 200, "vivo", LocalDate.of(20 26, 5, 1));	(Headphones, 2) (Smartphone, 1) totalAmountSpent: 54011.10	totalAmountSpent: 54011
<b>Function name: getStatus(), addBalance(double amount), removeBalance(double amount) and isActive()</b>				

29	testGetStatus()	Customer(2020160005, "David", "david@gmail.com", "Sylhet")	INACTIVE	INACTIVE
30	testAddBalance()	Customer(2020160005, "David", "david@gmail.com", "Sylhet")	100.0	100
31	testRemoveBalance()	Customer(2020160005, "David", "david@gmail.com", "Sylhet")	50.0	50.0
32	testIsActive()	Customer(2020160005, "David", "david@gmail.com", "Sylhet")	Active	Active
<b>Function name: calculateAveragePurchaseQuantity(), and hasHighSpending()</b>				
33	calculateAveragePurchaseQuantity()	(Headphones, 2) (Smartphone, 1)	AveragePurchaseQuantity: 1.5	AveragePurchaseQuantity: 1.5
34	hasHighSpending()	Product list	False	False

## 2. CustomerManager - 13 functions

- 2.1. CustomerManager()
- 2.2. addCustomer(Customer customer)
- 2.3. removeCustomer(int customerId)
- 2.4. findCustomerById(int customerId)
- 2.5. updateCustomerEmail(int customerId, String newEmail)
- 2.6. updateCustomerAddress(int customerId, String newAddress)
- 2.7. getTotalCustomers()
- 2.8. isPreferredCustomer(int customerId)
- 2.9. calculateTotalCustomerBalance()
- 2.10. findCustomerWithHighestPurchaseAmount()
- 2.11. updateCustomerStatus()
- 2.12. removeInactiveCustomers()
- 2.13. calculateAveragePurchaseAmount()

**Table 2: Test Case Table for Customer Manager Class**

<b>Main Class Name: CustomerManager</b>	<b>Test Class Name: TestCustomerManager</b>
---	---

Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: customerManager()</b>				
1	testCustomerManagerInitialization()	New CustomerManager object	initialized and empty	not null and size is 0
<b>Function name: addCustomer(Customer customer)</b>				
2	testAddCustomer()	(2020160065, "Krittika", "k2001roy@gmail.com", "Dhamrai")	ID: 2020160065	ID: 2020160065
3	testAddCustomer_NewCustomer_AddedSuccessfully()	(2020160065, "Krittika", "k2001roy@gmail.com", "Dhamrai")	ID: 2020160065	(2020160065, "Krittika", "k2001roy@gmail.com", "Dhamrai")
4	testAddCustomerExistingCustomerUpdatedSuccessfully()	(2020160065, "Krittika", "k2001roy@gmail.com", "Dhamrai") (2020160197, "Nowshin", "nowshin@gmail.com", "Dhaka")	ID: 2020160065 is updated	size is 2 and updated customer object is retrieved by ID
5	testAddCustomerNullCustomerThrowsNullPointerException()	Null	Null	Null
<b>Function name: removeCustomer(Customer customer)</b>				
6	testRemoveExistingCustomer()	2020160010	Customer removed from the map by ID	true
7	testRemoveNonExistingCustomer()	2020160010	Customer not found	false

8	testRemoveCustomerWithMultipleCustomers()	2020160065	Customer removed from the map by ID	true
<b>Function name: findCustomerById(int customerId)</b>				
9	testFindExistingCustomer()	2020160010	ID is retrieved	(2020160010, "John Doe", "john@example.com", "Dhaka")
10	testFindNonExistingCustomer()	2020160010	Customer not found	Customer object is null after retrieval
11	testFindCustomerWithMultipleCustomers()	(2020160065 and 2020160197)	respective IDs are retrieved	retrieved by their IDs and match the originals
<b>Function name: updateCustomerEmail(int customerId, String newEmail), updateCustomerAddress(int customerId, String newAddress)</b>				
12	testUpdateCustomerEmail_NonExistingCustomer()	(2020160010) and krittika@gmail.com)	Customer not found	false
13	testUpdateCustomerAddressNonExistingCustomer()	(1, "456 Elm St")	Customer not found	false
<b>Function name: removeInactiveCustomers()</b>				
14	testRemoveInactiveCustomers()	(2020160003, "Roy", "k2001roy@gmail.com", "Dhaka") (2020160004, "Alice", "alice@gmail.com", "Chittagong")	Customer not found	contains only the active customer after removal
15	testSetBalance()	(2020160003, "roy", "k2001roy@g	1000.0	1000

		mail.com", "Dhaka");		
--	--	-------------------------	--	--

### 3. Employee - 11 functions

- 3.1. Employee(int id, String name, String department, String jobTitle, double salary)
- 3.2. getId()
- 3.3. setId(int id)
- 3.4. getName()
- 3.5. setName(String name)
- 3.6. getDepartment()
- 3.7. setDepartment(String department)
- 3.8. getJobTitle()
- 3.9. setJobTitle(String jobTitle)
- 3.10. getSalary()
- 3.11. setSalary(double salary)

**Table 3: Test Case Table for EmployeeClass**

Main Class Name: Employee		Test Class Name: TestEmployee		
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: Employee(int id, String name, String department, String jobTitle, double salary)</b>				
1	testEmployeeConstructor()	(2020160197, "Nowshin", "CSE", "Developer", 50000.0)	(2020160197, "Nowshin", "CSE", "Developer", 50000.0)	(2020160197, "Nowshin", "CSE", "Developer", 50000.0)
2	testEmployeeConstructorWithEmptyName()	""	""	""
3	testEmployeeConstructorWithNegSalary()	-50000.0	-50000.0	-50000.0
<b>Function name: setId(int id)</b>				
4	testSetId()	2024160197	2024160197	2024160197
5	testSetIdNeg()	-1	-1	-1

6	testSetIdZero()	0	0	0
7	testSetIdMultiple()	2020160111, 2020160222	2020160222	2020160222
8	testSetIdMax()	Integer.MAX_V ALUE	Integer.MAX_V ALUE	Integer.MAX_V ALUE
<b>Function name: setName(String name)</b>				
9	testSetName()	Tasnia	Tasnia	Tasnia
10	testSetNameWithSpace()	Nowshin Tasnia	Nowshin Tasnia	Nowshin Tasnia
11	testSetNameMultiple() )	Nowshin, Tasnia	Tasnia	Tasnia
12	testSetNameCheckCase()	boishakhy	Not equal	Not equal
13	testSetNameCheckNull()	Null	Null	Null
<b>Function name: setDepartment(String department)</b>				
14	testSetDepartment()	EEE	EEE	EEE
15	testSetDepartmentMultiple()	EEE, CSE	CSE	CSE
16	testSetDepartmentCheckCase()	eee	Not equal	Not equal
17	testSetDepartmentNull()	Null	Null	Null
<b>Function name: setJobTitle(String jobTitle)</b>				
18	testSetJobTitle()	Tester	Tester	Tester
19	testSetJobTitleMultiple()	Tester, Designer	Designer	Designer
20	testSetJobTitleCheckCase()	designer	Not equal	Not equal
21	testSetJobTitleNull()	Null	Null	Null

Function name: setSalary(double salary)				
22	testSetSalary()	6000.0	6000.0	6000.0
23	testSetSalaryMultiple() ( )	6000.0, 7000.0	7000	7000
24	testSetSalaryNegative() ( )	-7000.0	-7000.0	-7000.0,
25	testSetSalaryMax()	Integer.MAX_V ALUE	Integer.MAX_V ALUE	Integer.MAX_V ALUE
26	testSetSalaryFloat()	70000.550	70000.550	70000.550

#### 4. EmployeeManager - 9 functions

- 4.1. EmployeeManager()
- 4.2. addEmployee(Employee employee)
- 4.3. removeEmployee(int employeeId)
- 4.4. getEmployees()
- 4.5. findEmployeeById(int employeeId)
- 4.6. findEmployeesByDepartment(String department)
- 4.7. calculateTotalSalary()
- 4.8. getEmployeesWithHighestSalary()
- 4.9. findEmployeesByJobTitle(String jobTitle)

**Table 4: Test Case Table for EmployeeManager Class**

Main Class Name: EmployeeManager		Test Class Name: TestEmployeeManager		
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
Function name: Employee(int id, String name, String department, String jobTitle, double salary)				
1	testEmployeeManager()	(2020160197, "Nowshin", "CSE", "Developer", 50000.0)	(2020160197, "Nowshin", "CSE", "Developer", 50000.0)	(2020160197, "Nowshin", "CSE", "Developer", 50000.0)
Function name: addEmployee(Employee employee)				
2	testAddEmployeeValid()	(2020160198, "Krittika", "CSE", "Tester",	True	True

		500000000000.0 )		
3	testAddEmployeeMultiple()	Multiple employee added	Contains	Contains
4	testAddEmployeeOrder()	Multiple employee added	Summoned by get(0), get(1)	Summoned by get(0), get(1)
5	testEmployeeManagerEmpty()	No employee added	Size=0	Size=0
<b>Function name: removeEmployee(int employeeId)</b>				
6	testRemoveEmployeeExisting()	Existing employee removed	Size=0	Size=0
7	testRemoveEmployeeNonExisting()	Remove function over non existing employee	Size = Initial size	Size = Initial size
8	testRemoveEmployeeEmpty()	No employee removed	Size = Initial size	Size = Initial size
9	testRemoveEmployeeMultiple()	Multiple employee added and removed all	Size = 0	Size =0
<b>Function name: getEmployees()</b>				
10	testGetEmployees()	Added 1 employee	Found List Size = 1	Found List Size = 1
11	testGetEmployeesMultipleCalls()	1 employee added in different list	Lists are not same	Lists are not same
<b>Function name: findEmployeeById(int employeeId)</b>				
12	testFindEmployeeById()	Employee1 ID = 2020160166	ID(2020160166) = Employee1	ID(2020160166) = Employee1
13	testFindEmployeeByIdNonExisting()	Non existing ID	Null	Null
14	testFindEmployeeByIdMultipleSameID()	Multiple employee with same ID	Found	Found

15	testFindEmployeeByIdEmpty()	Empty list	Not found	Not found
16	testRemoveEmployee()	Remove employee	Removed	Removed
<b>Function name: findEmployeesByDepartment(String department)</b>				
17	testFindEmployeesByDepartment()	Department = IT	Found	Found
18	testFindEmployeesByDepartmentNonExisting()	Department = CSE	Not found	Not found
19	testFindEmployeesByDepartmentEmpty()	Empty list	Not found	Not found
20	testFindEmployeesByDepartmentCaseInsensitivity()	Searched by small case	Case insensitive	Case insensitive
<b>Function name: calculateTotalSalary()</b>				
21	testCalculateTotalSalary()	6000+7000	13000	13000
22	testCalculateTotalSalaryNegative()	6000+(-7000)	-1000	-1000
23	testCalculateTotalSalaryNoEmployees()	Total salary = 0	Total salary = 0	Total salary = 0
<b>Function name: getEmployeesWithHighestSalary()</b>				
24	testGetEmployeesWithHighestSalary()	Highest salary = 1 employee	Employee_3	Employee_3
25	testGetEmployeesWithHighestSalaryMultiple()	Highest salary = 2 employees	Employee_2, Employee_3	Employee_2, Employee_3
26	testGetEmployeesWithHighestSalaryNegative()	Highest salary = 2 employees	Employee_2, Employee_3	Employee_2, Employee_3
27	testGetEmployeesWithHighestSalaryEmpty()	Highest salary = 0 employee	Size = 0	Size =0
<b>Function name: findEmployeesByJobTitle(String jobTitle)</b>				
28	testFindEmployeesByJobTitle()	Title = Developer	Found 2 employees	Found 2 employees
29	testFindEmployeesByJobTitleNonExisting()	Title = Developer	Not found	Not found

30	testFindEmployeesByJobTitleEmpty()	Empty list	Not found	Not found
31	testFindEmployeesByJobTitleCaseInsensitivity()	Searched by small case	Case insensitive	Case insensitive

## 5. Inventory - 15 functions

- 5.1. Inventory()
- 5.2. addProduct(Product product)
- 5.3. removeProduct(int productId)
- 5.4. getProducts()
- 5.5. findProductById(int productId)
- 5.6. findProductsByName(String name)
- 5.7. updateProduct(int productId, Product updatedProduct)
- 5.8. getProductsByPriceRange(double minPrice, double maxPrice)
- 5.9. getTotalInventoryValue()
- 5.10. checkProductAvailability(int productId, int quantity)
- 5.11. calculateAverageProductPrice()
- 5.12. hasProductType(String type)
- 5.13. updateProductQuantity(int productId, int newQuantity)
- 5.14. removeExpiredProducts()
- 5.15. getProductsBelowThreshold(int threshold)

**Table 5: Test Case Table for Inventory Class**

Main Class Name: Inventory			Test Class Name: TestInventory	
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: Inventory()</b>				
1	testInventoryConstruct or()	Products list	not null and empty	not null and empty
2	testInventoryConstruct orInitialCapacity()	Products list	size is 0	size is 0
3	testInventoryConstruct orNotNull()	Products list	Not null	Not null
<b>Function name: addProduct(Product product)</b>				
4	testAddProduct()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 12, 31))	size is 1 and contains the added product	size is 1 and contains the added product
5	testAddMultipleProducts()	Product(4, "Headphones", 2000.55, 400, "oppo",	size is 2 and contains both the added products	size is 2 and contains both the added products

		LocalDate.of(2023, 12, 31)) Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(2024, 06, 15))		
<b>Function name: removeProduct(Product product)</b>				
6	testRemoveProduct()	Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(2024, 06, 15))	size is 0	size is 0
7	testRemoveNonExistentProduct()	Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(2024, 06, 15))	False	False
8	testRemoveMultipleProducts()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 12, 31)) Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(2024, 06, 15))	size is 0	size is 0
<b>Function name: getProducts()</b>				
9	testGetProducts()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 12, 31)) Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(2024, 06, 15))	containing all products	containing all products

10	testGetProductsEmpty() )	Products list	Empty list	Empty list
11	testGetProductsClone()	ID: 5	ID: 6	ID: 6
<b>Function name:findProductsById(String id)</b>				
12	testFindProductById()	ID: 5	Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(20 24, 06, 15))	Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(202 4, 06, 15))
13	testFindNonExistentPro ductById()	ID: 5	null	null
14	testFindProductByIdE mpty()	Empty inventory	null	null
15	testFindNonExistentPro ductById()	ID: 5	null	null
16	testFindProductByIdE mpty()	Empty inventory	null	null
<b>Function name:findProductsByName(String name)</b>				
17	testFindProductsByNa me()	Headphones, Keyboard	Headphones	Headphones
18	testFindNoProductsBy Name()	Headphones, Keyboard	Empty list	Empty list
19	testFindProductsByNa meEmpty()	Empty	Empty list	Empty list
<b>Function name: updateProduct(int productId, Product updatedProduct)</b>				
20	testUpdateProduct()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 12, 31))	True	True
21	testUpdateProductNotF ound()	Product(4, "Headphones", 2000.55, 400, "oppo", LocalDate.of(2023, 12, 31));	False	False

		Product Product(5, "Keyboard", 500.00, 100, "logitech", LocalDate.of(2024, 06, 15));		
22	testUpdateProductMultipleProducts()	(4, Headphones)	Inventory is updated	Inventory is updated
<b>Function name: getProductsByPriceRange(double minPrice, double maxPrice)</b>				
23	testGetProductsByPriceRange()	PriceRange(12.0, 25.0)	Product(2, "AC", 20.0, 10, "Vision", LocalDate.of(2023, 2, 1))	Product(2, "AC", 20.0, 10, "Vision", LocalDate.of(2023, 2, 1))
24	testGetProductsByPriceRangeNoProducts()	PriceRange(50.0, 60.0)	Empty list	Empty list
25	testGetProductsByPriceRangeMinGreaterThanMax()	PriceRange(max, min) = (30.0, 40.0) 30 < 40	Product(3, "TV", 30.0, 15, "LG", LocalDate.of(2023, 3, 1))	Product(3, "TV", 30.0, 15, "LG", LocalDate.of(2023, 3, 1))
<b>Function name: getTotalInventoryValue()</b>				
26	testGetTotalInventoryValueSingleProduct()	Price : 10.0 Quantity: 5 (“PC”, 5)	TotalInventoryValue: (10.0*5) = 50.0	TotalInventoryValue: (10.0*5) = 50.0
27	testGetTotalInventoryValueMultipleProducts()	Product1: Price : 10.0 Quantity: 5 Product2: Price : 20.0 Quantity: 10 Product3: Price : 30.0 Quantity: 10	TotalInventoryValue: 550.0	TotalInventoryValue: 550.0
28	testGetTotalInventoryValueEmptyInventory()	Empty inventory	0.0	0.0

<b>Function name: checkProductAvailability(int productId, int quantity)</b>				
29	testCheckProductAvailabilityAvailable()	ProductAvailability(1, 3)	True	True
30	testCheckProductAvailabilityNotAvailable()	ProductAvailability(1, 6)	False	False
31	testCheckProductAvailabilityNotInInventory()	ProductAvailability(4, 1)	False	False
<b>Function name: calculateAverageProductPrice()</b>				
32	testCalculateAverageProductPriceWithProducts()	Product1: PC Price : 10.0 Product2: AC Price : 20.0 Product3: TV Price : 30.0	AverageProductPrice: 20.0	AverageProductPrice: 20.0
33	testCalculateAverageProductPriceWithoutProducts	Empty Products	AverageProductPrice: 0.0	AverageProductPrice: 0.0
34	testCalculateAverageProductPriceWithOneProduct()	Product1: PC Price : 10.0	AverageProductPrice: 10.0	AverageProductPrice: 10.0
<b>Function name: hasProductType(String type)</b>				
35	testHasProductTypeTrue()	Product1: PC Product2: AC Product3: TV ProductType: PC	True	True
36	testHasProductTypeFalse()	Product1: PC Product2: AC Product3: TV ProductType: Phone	False	False
37	testHasProductTypeEmptyInventory()	Empty Inventory	False	False
<b>Function name: updateProductQuantity(int productId, int newQuantity)</b>				
38	testUpdateProductQuantityTrue()	ProductQuantity(1, 5)	True	True

39	testUpdateProductQuantityFalse()	ProductQuantity(4, 5)	False	False
40	testUpdateProductQuantityNegativeProductId()	ProductQuantity(-1, 5)	False	False
<b>Function name: removeExpiredProducts()</b>				
41	testRemoveExpiredProducts()	Inventory with expired products	All expired products are removed	All expired products are removed
42	testRemoveExpiredProductsEmptyInventory()	Empty Inventory	Empty list	Empty list
<b>Function name: getProductsBelowThreshold(int threshold)</b>				
43	testGetProductsBelowThresholdNoProducts()	Inventory with all products above the threshold	Empty list	Empty list
44	testGetProductsBelowThresholdEmptyInventory()	Empty Inventory	Empty list	Empty list

## 6. InventoryFileManager - 4 functions

- 6.1. InventoryFileManager(String fileName)
- 6.2. writeInventoryToFile(List<Product> products)
- 6.3. readInventoryFromFile()
- 6.4. backupInventory(String backupFileName)

**Table 6: Test Case Table for Inventory File Manager Class**

<b>Main Class Name: InventoryFileManager</b>		<b>Test Class Name: TestInventoryFileManager</b>		
<b>Test Case ID</b>	<b>Test Case Name</b>	<b>Input Values</b>	<b>Expected Output</b>	<b>Actual Output</b>
<b>Function name: InventoryFileManager(String fileName)</b>				
1	testFileManagerConstructor()	testFile.txt	successfully	not null
<b>Function name: writeInventoryToFile(List&lt;Product&gt; products)</b>				
2	testGetFileName()	testFile.txt	testFile.txt	testFile.txt
3	testGetFileNameWithNullValue()	null	null	null

4	testGetFileNameWithEmptyValue()	""	""	""
<b>Function name: readInventoryFromFile(), backupInventory(String backupFileName)</b>				
5	testSetFileName()	testFile.txt	krittika.txt	krittika.txt
6	testBackupInventory()	filename "testFile.txt" and backup named "testFileBackup.txt"	Backup file created successfully	Backup file exists, is a file, and has the same size

## 7. Order - 12 functions

- 7.1. Order(int orderId, int customerId, String orderDate, String paymentMethod)
- 7.2. getOrderId()
- 7.3. setOrderId(int orderId)
- 7.4. getCustomerId()
- 7.5. setCustomerId(int customerId)
- 7.6. getOrderDate()
- 7.7. setOrderDate(String orderDate)
- 7.8. getPaymentMethod()
- 7.9. setPaymentMethod(String paymentMethod)
- 7.10. getStatus()
- 7.11. setStatus(OrderStatus status)
- 7.12. getTotalPrice()

**Table 7: Test Case Table for Order Class**

Main Class Name: Order			Test Class Name: TestOrder	
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: Order(int orderId, int customerId, String orderDate, String paymentMethod)</b>				
1	getOrderId	123456	123456	123456
2	getCustomerId	12345	12345	12345
3	getOrderDate	2023-07-25	2023-07-25	2023-07-25
4	getPaymentMethod	Visa	Visa	Visa
5	getStatus	OrderStatus.PENDING	OrderStatus.PENDING	OrderStatus.PENDING

6	getTotalPrice	0	0	0
7	setOrderId	54321	54321	54321
8	setCustomerId	54321	54321	54321
9	setOrderDate	2023-08-01	2023-08-01	2023-08-01
10	setPaymentMethod	Amex	Amex	Amex
11	setStatus	OrderStatus.SHIPPED	OrderStatus.SHIPPED	OrderStatus.SHIPPED
12	setTotalPrice	149.99	149.99	149.99
13	getOrderId	13243	13243	13243
14	getCustomerId	23211	23211	23211
15	getOrderDate	2023-07-25	2023-07-25	2023-07-25
16	getPaymentMethod	Amex	Amex	Amex
17	getStatus	OrderStatus.PENDING	OrderStatus.PENDING	OrderStatus.PENDING
18	getTotalPrice	0	0	0
19	setOrderId	54321	54321	54321
20	setCustomerId	12345	12345	12345
21	setOrderDate	2023-08-01	2023-08-01	2023-08-01
22	setPaymentMethod	Visa	Visa	Visa
23	setStatus	OrderStatus.SHIPPED	OrderStatus.SHIPPED	OrderStatus.SHIPPED
24	setTotalPrice	149.99	149.99	149.99
25	getOrderId	34323	34323	34323
26	getCustomerId	42131	42131	42131
27	getOrderDate	2023-08-25	2023-08-25	2023-08-25
28	getPaymentMethod	MasterCard	MasterCard	MasterCard
29	getStatus	OrderStatus.PENDING	OrderStatus.PENDING	OrderStatus.PENDING
30	getTotalPrice	0	0	0
31	setOrderId	98765	98765	98765
32	setCustomerId	54321	54321	54321
33	setOrderDate	2023-09-01	2023-09-01	2023-09-01

34	setPaymentMethod	Amex	Amex	Amex
35	setStatus	OrderStatus.DELIVERED	OrderStatus.DELIVERED	OrderStatus.DELIVERED
36	setTotalPrice	199.99	199.99	199.99

## 8. OrderManager - 13 functions

- 8.1. OrderManager()
- 8.2. addOrder(Order order)
- 8.3. removeOrder(int orderId)
- 8.4. getOrders()
- 8.5. calculateTotalRevenue()
- 8.6. getOrdersByCustomerId(int customerId)
- 8.7. getTotalOrders()
- 8.8. clearAllOrders()
- 8.9. findOrderById(int orderId)
- 8.10. updateOrderStatus(int orderId, OrderStatus newStatus)
- 8.11. getOrdersWithStatus(OrderStatus status)
- 8.12. getOrdersByPaymentMethod(String paymentMethod)
- 8.13. getOrdersByDateRange(String startDate, String endDate)

**Table 8: Test Case Table for Order Manager Class**

Main Class Name: OrderManager			Test Class Name: TestOrderManager	
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
1	testOrderManagerWithOrders1	order1, order2	order1, order2	order1, order2
2	testOrderManagerWithOrders1	order1, order2	order1, order2	order1, order2
3	testOrderManagerWithOrders1	2	2	2
4	testOrderManagerWithOrders1	1	1	1
5	testOrderManagerWithOrders1	2	Order with id 2	Order with id 2
6	testOrderManagerWithOrders1	2, OrderStatus.DELIVERED	Order with status DELIVERED	Order with status DELIVERED
7	testOrderManagerWithOrders1	1002	List of 1 order	List of 1 order
8	testOrderManagerWithOrders1	75	75	75

9	testOrderManagerWithOrders1	OrderStatus.PENDING	Empty list	Empty list
10	testOrderManagerWithOrders1	Credit Card	Empty list	Empty list
11	testOrderManagerWithOrders1	2024-05-01, "2024-05-02"	List of 1 order	List of 1 order
12	testOrderManagerWithOrders1	N/A	0 orders	0 orders
13	testOrderManagerWithOrders2	order1, order2	done	N/A
14	testOrderManagerWithOrders2	N/A	List of 2 orders	List of 2 orders
15	testOrderManagerWithOrders2	N/A	2	2
16	testOrderManagerWithOrders2	101	Order with status Completed	Order with status Completed
17	testOrderManagerWithOrders2	102	Order with id 102	Order with id 102
18	testOrderManagerWithOrders2	102, OrderStatus.DELIVERED	Order with status DELIVERED	Order with status DELIVERED
19	testOrderManagerWithOrders2	2002	List of 1 order	List of 1 order
20	testOrderManagerWithOrders2	OrderList	100	100
21	testOrderManagerWithOrders2	OrderStatus.PENDING	Empty list	Empty list
22	testOrderManagerWithOrders2	Visa	Empty list	Empty list
23	testOrderManagerWithOrders2	2024-06-01, "2024-06-02"	List of 1 order	List of 1 order
24	testOrderManagerWithOrders2	N/A	0 orders	0 orders
25	testOrderManagerWithOrders3	order1, order2	N/A	N/A
26	testOrderManagerWithOrders3	N/A	List of 2 orders	List of 2 orders
27	testOrderManagerWithOrders3	N/A	2	2
28	testOrderManagerWithOrders3	201	N/A	N/A
29	testOrderManagerWithOrders3	202	Order with id 202	Order with id 202

30	testOrderManagerWithOrders3	202, OrderStatus.D ELIVERED	Order with status DELIVERED	Order with status DELIVERED
31	testOrderManagerWithOrders3	3002	List of 1 order	List of 1 order
32	testOrderManagerWithOrders3	N/A	150	150
33	testOrderManagerWithOrders3	OrderStatus.P ENDING	Empty list	Empty list
34	testOrderManagerWithOrders3	AmEx	Empty list	Empty list
35	testOrderManagerWithOrders3	2024-07-01, "2024-07-02"	List of 1 order	List of 1 order
36	testOrderManagerWithOrders3	N/A	0 orders	0 orders

## 9. Product - 11 functions

- 9.1. Product(int id, String name, double price, int quantity, String type, LocalDate expiryDate)
- 9.2. getId()
- 9.3. getName()
- 9.4. getPrice()
- 9.5. getQuantity()
- 9.6. getType()
- 9.7. getExpiryDate()
- 9.8. setPrice(double price)
- 9.9. setQuantity(int quantity)
- 9.10. isExpired()
- 9.11. calculateTotalValue()

**Table 9: Test Case Table for Product Class**

Main Class Name: Product			Test Class Name: TestProduct	
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
Function name: getId()				
1	testGetId_Valid()	Product(1, "Apple", 0.5, 10, "Fruit", LocalDate.of(20 24, 6, 1))	1	1

2	testGetId_Different()	Product(2, "Banana", 0.2, 15, "Fruit", LocalDate.of(2024, 6, 15))	2	2
3	testGetId_Boundary()	Product(0, "Cherry", 0.3, 20, "Fruit", LocalDate.of(2024, 7, 1))	0	0
<b>Function name: getName()</b>				
4	testGetName_Valid()	Product(2, "Banana", 0.2, 15, "Fruit", LocalDate.of(2024, 6, 15))	"Banana"	"Banana"
5	testGetName_Different()	Product(3, "Cherry", 0.3, 20, "Fruit", LocalDate.of(2024, 7, 1))	"Cherry"	"Cherry"
6	testGetName_Empty()	Product(4, "", 1.0, 25, "Fruit", LocalDate.of(2024, 8, 1))	""	""
<b>Function name: getPrice()</b>				
7	testGetPrice_Valid()	Product(3, "Cherry", 0.3, 20, "Fruit", LocalDate.of(2024, 7, 1))	0.3	0.3
8	testGetPrice_Zero()	Product(4, "Date", 0.0, 25, "Fruit", LocalDate.of(2024, 8, 1))	0.0	0.0
9	testGetPrice_High()	Product(5, "Eggplant", 999.99, 5,	999,99	999,99

		"Vegetable", LocalDate.of(20 24, 9, 1))		
<b>Function name: getQuantity()</b>				
10	testGetQuantity_Valid()	Product(4, "Date", 1.0, 25, "Fruit", LocalDate.of(20 24, 8, 1))	25	25
11	testGetQuantity_Zero()	Product(6, "Fig", 2.0, 0, "Fruit", LocalDate.of(20 24, 10, 1))	0	0
12	testGetQuantity_Negative()	Product(7, "Grape", 0.8, -5, "Fruit", LocalDate.of(20 24, 11, 1))	-5	-5
<b>Function name: getType()</b>				
13	testGetType_Valid()	Product(5, "Eggplant", 1.5, 5, "Vegetable", LocalDate.of(20 24, 9, 1))	"Vegetables"	"Vegetables"
14	testGetType_Different()	Product(8, "Honeydew", 3.0, 12, "Fruit", LocalDate.of(20 24, 12, 1))	"Fruit"	"Fruit"
15	testGetType_Empty()	Product(9, "Jackfruit", 1.2, 8, "", LocalDate.of(20 23, 5, 1))	""	""
<b>Function name: getExpiryDate()</b>				

16	testGetExpiryDate_Valid()	Product(6, "Fig", 2.0, 30, "Fruit", LocalDate.of(2024, 10, 1))	LocalDate.of(2024, 10, 1)	LocalDate.of(2024, 10, 1)
17	testGetExpiryDate_Different	Product(7, "Grape", 0.8, 10, "Fruit", LocalDate.of(2024, 11, 1))	LocalDate.of(2024, 11, 1)	LocalDate.of(2024, 11, 1)
18	testGetExpiryDate_Past()	Product(9, "Jackfruit", 1.2, 8, "Fruit", LocalDate.of(2023, 5, 1))	LocalDate.of(2023, 5, 1)	LocalDate.of(2023, 5, 1)
<b>Function name: setPrice(double price)</b>				
19	testSetPrice_Valid()	Product(7, "Grape", 0.8, 10, "Fruit", LocalDate.of(2024, 11, 1))	0.9	0.9
20	testSetPrice_Zero()	Product(8, "Honeydew", 3.0, 12, "Fruit", LocalDate.of(2024, 12, 1))	0.0	0.0
21	testSetPrice_Negative()	Product(10, "Kiwi", 1.1, 7, "Fruit", LocalDate.of(2024, 5, 1))	-1.0	-1.0
<b>Function name: setQuantity(int quantity)</b>				
22	testSetQuantity_Valid()	Product(8, "Honeydew", 3.0, 12, "Fruit",	14	14

		LocalDate.of(2024, 12, 1))		
23	testSetQuantity_Zero()	Product(9, "Jackfruit", 1.2, 8, "Fruit", LocalDate.of(2023, 5, 1))	0	0
24	testSetQuantity_Negative()	Product(11, "Lemon", 0.5, 10, "Fruit", LocalDate.of(2024, 6, 1))	-5	-5
<b>Function name: isExpired()</b>				
25	testIsExpired_True()	Product(9, "Jackfruit", 1.2, 8, "Fruit", LocalDate.of(2023, 5, 1))	true	true
26	testIsExpired_False()	Product(10, "Kiwi", 1.1, 7, "Fruit", LocalDate.of(2024, 5, 1))	false	false
27	testIsExpired_Today()	Product(12, "Mango", 1.0, 5, "Fruit", LocalDate.now())	false	false
<b>Function name: calculateTotalValue()</b>				
28	testCalculateTotalValue_Valid()	Product(11, "Lemon", 0.5, 10, "Fruit", LocalDate.of(2024, 6, 1))	5.0	5.0
29	testCalculateTotalValue_Different()	Product(12, "Mango", 1.0, 5, "Fruit", LocalDate.of(2024, 7, 1))	5.0	5.0

30	testCalculateTotalValue_Zero Quantity()	`Product	-	-
----	--	----------	---	---

## 10. ShoppingCart - 12 functions

- 10.1. ShoppingCart()
- 10.2. addItem(Product product, int quantity)
- 10.3. removeItem(Product product, int quantity)
- 10.4. getItems()
- 10.5. calculateTotalPrice()
- 10.6. clear()
- 10.7. isEmpty()
- 10.8. applyDiscount(double discountPercentage)
- 10.9. getTotalQuantity()
- 10.10. containsProduct(Product product)
- 10.11. getItemsAboveThreshold(int threshold)
- 10.12. updateQuantity(Product product, int newQuantity)

**Table 10: Test Case Table for Shopping Cart Class**

Main Class Name: ShoppingCart		Test Class Name: TestShoppingCart		
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
Function name: getId()				
1	testAddRemoveItem()	Added Apple, Banana, Orange and Removed Banana, Pear	Apple, Orange	Apple, Orange
2	testMultipleItemsInCart()	Added Milk, Bread, Eggs	Milk, Bread, Eggs are in cart	Milk, Bread, Eggs are in cart
3	testHighPriceItems()	Eggs has high price	Eggs	Eggs

## 11. Supplier - 11 functions

- 11.1. Supplier(String name)
- 11.2. addProduct(Product product)
- 11.3. removeProduct(Product product)
- 11.4. updateProductPrice(Product product, double newPrice)
- 11.5. getProductsSupplied()
- 11.6. getTotalProductsSupplied()
- 11.7. clearAllProducts()
- 11.8. getTotalInventoryValue()
- 11.9. hasProduct(Product product)
- 11.10. hasExpiredProducts()
- 11.11. updateProductQuantity(Product product, int newQuantity)

**Table 11: Test Case Table for Supplier Class**

Main Class Name: Supplier		Test Class Name: TestSupplier		
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: Supplier(String name)</b>				
1	testConstructorValidName()	Books	True	True
2	testConstructorSpecialCharacters()	books#CSE_430	True	True
<b>Function name: addProduct(Product product)</b>				
3	testAddProductValid()	Product added	Found	Found
4	testAddProductDuplicate()	Duplicate product added	Found	Found
5	testAddProductDifferentQuantities()	Different quantities added	Found	Found
6	testAddProductModifications()	Modifications added	Found	Found
<b>Function name: removeProduct(Product product)</b>				
7	testRemoveProductExisting()	Removed existing product	Found	Found
8	testRemoveProductNonExisting()	No product removed	Size = 0	Size = 0
9	testRemoveProductDifferentObject()	Different product removed	Found	Found
10	testRemoveProduct_AllAndCheckEmpty()	Remove all products	True	True

Function name: updateProductPrice(Product product, double newPrice)				
11	testUpdateProductPriceIncrease()	Increase price	Price increased	Price increased
12	testUpdateProductPriceDecrease()	Decrease price	Price decreased	Price decreased
13	testUpdateProductPriceNoChange()	No change	No update	No update
14	testUpdateProductPriceMultiple()	Multiple change	Multiple update	Multiple update
Function name: getProductsSupplied()				
15	testGetProductsSupplied()	Products added	Found	Found
16	testGetProductsSupplied2()	Products added	Found	Found
17	testGetProductsSuppliedEmpty()	No products added	Not found	Not found
18	testGetProductsSuppliedModifications()	Modified	Updated	Updated
19	testGetProductsSuppliedMultipleCalls()	Multiple supply	Found	Found
Function name: getTotalProductsSupplied()				
20	testGetTotalProductsSupplied()	1+1	2	2
21	testGetTotalProductsSuppliedEmpty()	No supply	0	0
22	testGetTotalProductsSuppliedAddRemove()	Added and removed	Found	Found
23	testGetTotalProductsSuppliedMultipleCalls()	Multiple supply	Found	Found
24	testGetTotalProductsSuppliedAddingAfterGet()	Added after get	Found	Found
Function name: clearAllProducts()				
25	testClearAllProducts()	All cleared	Not found	Not found
26	testClearAllProductsEmpty()	Nothing cleared	Not found	Not found
27	testClearAllProductsMultipleCalls()	Multiple clear	Cleared	Cleared
28	testClearAllProductsAddingAfterClearing()	Adding after clearing	Cleared	Cleared
29	testClearAllProductsImpactOnOtherMethods()	Impact on hasProduct	No change	No change

Function name: getTotalInventoryValue()				
30	testGetTotalInventoryValueMultipleProducts()	Multiple product	Found	Found
31	testGetTotalInventoryValueSingleProduct()	Single product	Found	Found
32	testGetTotalInventoryValueZero()	Value = 0	0	0
33	testGetTotalInventoryValueEmpty()	Empty value	0	0
34	testGetTotalInventoryValueModifications()	Modified values	Found	Found
Function name: hasProduct(Product product)				
35	testHasProductExisting()	Product exists	True	True
36	testHasProductNonExisting()	Not exists	False	False
37	testHasProductSameDataDifferentObject()	Different product exists	True	True
38	testHasProductEmpty()	Empty product	False	False
Function name: hasExpiredProducts()				
39	testHasExpiredProductsNo()	Expiration check	True	True
40	testHasExpiredProductsYes()	Expiration check	False	False
41	testHasExpiredProductsAllExpired()	Expiration check	True	True
42	testHasExpiredProductsAddRemove()	Expiration check	False	False
Function name: updateProductQuantity(Product product, int newQuantity)				
43	testUpdateProductQuantityIncrease()	Increase quantity	Increased	Increased
44	testUpdateProductQuantityDecrease()	Decrease quantity	Decreased	Decreased
45	testUpdateProductQuantityNoChange()	No quantity change	No change	No change
46	testUpdateProductQuantityMultipleProducts()	Multiple product quantity change	Multiple change	Multiple change

## 12. Transaction - 14 functions

- 12.1. Transaction(String type, double amount)
- 12.2. getDateTIme()
- 12.3. getType()
- 12.4. getAmount()
- 12.5. setAmount(double amount)
- 12.6. isPositiveAmount()
- 12.7. isNegativeAmount()
- 12.8. isOfType(String type)
- 12.9. isRecentTransaction()
- 12.10. isPastTransaction()
- 12.11. exceedsThreshold(double threshold)
- 12.12. isFutureTransaction()
- 12.13. isRefundTransaction()
- 12.14. isSpecificTransaction(String type, double amount)

**Table 12: Test Case Table for Transaction Class**

Main Class Name: Transaction		Test Class Name: TestTransaction		
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: Transaction(String type, double amount)</b>				
1	testTransactionValid()	Deposit = 100	Deposit = 100	Deposit = 100
2	testTransactionEmpty()	No transaction	Null	Null
3	testTransactionLargeAmount()	amount = 1000000.0	amount = 1000000.0	amount = 1000000.0
4	testTransactionMax()	Integer.MAX_VALUE	Integer.MAX_VALUE	Integer.MAX_VALUE
<b>Function name: getDateTIme(), getType()</b>				
5	testGetDateTImeValidTransaction()	Local date time	Not null	Not null
6	testGetTypeValid()	type = "Payment"	type = "Payment"	type = "Payment"
7	testGetTypeEmpty()	Null type	Null type	Null type
8	testGetTypeCaseSensitive()	Mixed case	Case sensitive	Case sensitive

Function name: getAmount()				
9	testGetAmountValid()	amount = 75.50	amount = 75.50	amount = 75.50
10	testGetAmountZero()	amount = 0	amount = 0	amount = 0
11	testGetAmountNegative()	amount = -25.50	amount = -25.50	amount = -25.50
12	testGetAmountLarge()	amount = 10000000.0	amount = 10000000.0	amount = 10000000.0
13	testGetAmountMax()	Integer.MAX_VALUE	Integer.MAX_VALUE	Integer.MAX_VALUE
Function name: setAmount(double amount)				
14	testSetAmountValidPositive()	Positive amount	Found	Found
15	testSetAmountZero()	Amount = 0	Amount = 0	Amount = 0
16	testSetAmountLarge()	amount = 10000000.0	amount = 10000000.0	amount = 10000000.0
17	testSetAmountMax()	Integer.MAX_VALUE	Integer.MAX_VALUE	Integer.MAX_VALUE
Function name: isPositiveAmount(), isNegativeAmount()				
18	testIsPositiveAmount()	Positive	True	True
19	testIsPositiveAmountZero()	Amount = 0	False	False
20	testIsPositiveAmountNegative()	amount = -25.50	False	False
21	testIsNegativeAmount()	amount = -25.50	True	True
22	testIsNegativeAmountZero()	Amount = 0	False	False
Function name: isOfType(String type)				
23	testIsNegativeAmountPositive()	amount = 75.50	False	False
24	testIsOfTypeMatching()	Matching type	True	True

25	testIsOfTypeNonMatching() )	Type non matched	False	False
<b>Function name: isRecentTransaction(), isPastTransaction()</b>				
26	testIsRecentTransaction()	Recent	True	True
27	testIsRecentTransactionFalse() )	Not recent	False	False
28	testIsPastTransaction()	Not recent	True	True
29	testIsPastTransactionFalse() )	Recent	False	False
<b>Function name: exceedsThreshold(double threshold)</b>				
30	testExceedsThreshold()	Exceeds	True	True
31	testExceedsThresholdFalse() )	Not exceeds	False	False
32	testExceedsThresholdZeroThreshold() )	Threshold = 0	Threshold = 0	Threshold = 0
<b>Function name: isFutureTransaction(), isRefundTransaction()</b>				
33	testIsFutureTransaction()	Future	True	True
34	testIsFutureTransactionFalse() )	Not future	False	False
35	testIsRefundTransactionMatching() )	Type = "Refund"	True	True
36	testIsRefundTransactionNonMatching() )	Type = "Withdrawal"	False	False
<b>Function name: isSpecificTransaction(String type, double amount)</b>				
37	testIsSpecificTransactionMatchingTypeAndAmount() )	Type = "Deposit"	True	True
38	testIsSpecificTransactionNonMatchingType() )	Type = "Transfer"	False	False

### 13. TransactionManager - 11 functions

- 13.1. TransactionManager()
- 13.2. addTransaction(Transaction transaction)
- 13.3. removeTransaction(Transaction transaction)
- 13.4. getTransactions()
- 13.5. getTotalRevenue()
- 13.6. getTotalTransactions()
- 13.7. getTransactionsByType(String type)
- 13.8. hasTransaction(Transaction transaction)
- 13.9. clearAllTransactions()
- 13.10. getTotalExpenses()
- 13.11. hasRefundTransactions()

**Table 13: Test Case Table for Transaction Manager Class**

Main Class Name: TransactionManager		Test Class Name: TestTransactionManager		
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
<b>Function name: Transaction(String type, double amount)</b>				
1	testTransactionManagerEmptyList()	Empty list	Null	Null
2	testTransactionManagerListType()	List type = ArrayList	List type = ArrayList	List type = ArrayList
<b>Function name: addTransaction(Transaction transaction)</b>				
3	testAddTransaction()	Transaction added	Found	Found
4	testAddTransactionMultiple()	Added multiple transaction	Found	Found
5	testAddTransactionDuplicate()	Added duplicate transaction	Found	Found
<b>Function name: removeTransaction(Transaction transaction)</b>				
6	testRemoveTransaction()	Removed transaction	Not found	Not found
7	testRemoveTransactionNonExisting()	No transaction removed	True	True

Function name: getTransactions()				
8	testGetTransactions()	Search for transactions	Found	Found
9	testGetTransactionsEmpty()	Searched	Not found	Not found
10	testGetTransactions_Modification()	Modification	True	True
11	testGetTransactionsCopy()	Copied transaction	True	True
Function name: getTotalRevenue()				
12	testGetTotalRevenueNoTransactions()	No Transactions	0	0
13	testGetTotalRevenueNegative()	-500-100	0	0
14	testGetTotalRevenueMixed()	100-25+50	150	150
15	testGetTotalRevenueLarge()	1000000.0	1000000.0	1000000.0
Function name: getTotalTransactions(), getTransactionsByType(String type)				
16	testGetTotalTransactionsEmpty()	Empty	0	0
17	testGetTotalTransactionsMultiple()	Multiple transactions	Found	Found
18	testGetTransactionsMatchingType()	Matching	Found	Found
19	testGetTransactionsNonMatchingType()	Non matching	True	True
Function name: hasTransaction(Transaction transaction)				
20	testHasTransaction()	Check transaction	False	False
21	testHasTransactionNonExisting()	Check transaction	Not found	Not found

22	testHasTransactionByData( )	Check transaction	Found	Found
<b>Function name: clearAllTransactions()</b>				
23	testClearAllTransactions()	Clear transactions	Cleared	Cleared
24	testClearAllTransactionsEmpty()	Empty	0	0
25	testClearAllTransactionsMultiple()	Multiple clear	Cleared	Cleared
26	testClearAllTransactionsStateAfterClearing()	State change after	Cleared	Cleared
27	testClearAllTransactionsMultipleTimes()	Multiple clear	Cleared	Cleared
<b>Function name: getTotalExpenses()</b>				
28	testGetTotalExpensesEmpty()	Empty expenses	0	0
29	testGetTotalExpensesAllPositive()	Positive expenses	True	True
30	testGetTotalExpensesMixed()	Mixed expenses	True	True
31	testGetTotalExpensesLarge()	Large expenses	True	True
32	testGetTotalExpensesZero()	No expenses	0	0
<b>Function name: hasRefundTransactions()</b>				
33	testHasRefundTransactionsEmpty()	Check refund	False	False
34	testHasRefundTransactionsNo()	Check refund	False	False
35	testHasRefundTransactionsYes()	Check refund	True	True

36	testHasRefundTransactions MultipleRefunds()	Check refund	True	True
----	--	--------------	------	------

## 14. Warehouse - 9 functions

- 14.1. Warehouse()
- 14.2. addProduct(Product product, int quantity)
- 14.3. removeProduct(Product product, int quantity)
- 14.4. getAvailableQuantity(Product product)
- 14.5. getInventory()
- 14.6. calculateInventoryValue()
- 14.7. containsExpensiveProducts(double thresholdPrice)
- 14.8. updateProductPrice(int productId, double newPrice)
- 14.9. getTotalProductCategories()

**Table 14: Test Case Table for Warehouse Class**

Main Class Name: Warehouse			Test Class Name: WarehouseTest	
Test Case ID	Test Case Name	Input Values	Expected Output	Actual Output
1	testWarehouseOperations1	(product1, 20)	20	20
2	testWarehouseOperations1	(product2, 15)	15	15
3	testWarehouseOperations1	(product3, 12)	12	12
4	testWarehouseOperations1	product1	20	20
5	testWarehouseOperations1	product2	15	15
6	testWarehouseOperations1	product3	12	12
7	testWarehouseOperations1	(product1, 5)	15	15
8	testWarehouseOperations1	500	TRUE	TRUE
9	testWarehouseOperations1	(product2.getProductId(), 549.99)	549.99	549.99
10	testWarehouseOperations1	-	1	1
11	testWarehouseOperations2	(product1, 30)	30	30
12	testWarehouseOperations2	(product2, 20)	20	20
13	testWarehouseOperations2	(product3, 15)	15	15
14	testWarehouseOperations2	product1	30	30
15	testWarehouseOperations2	product2	20	20
16	testWarehouseOperations2	product3	15	15
17	testWarehouseOperations2	(product1, 10)	20	20
18	testWarehouseOperations2	50	TRUE	TRUE

19	testWarehouseOperations2	(product2.getProductId(), 59.99)	59.99	59.99
20	testWarehouseOperations2	-	1	1
21	testWarehouseOperations3	(product1, 8)	8	8
22	testWarehouseOperations3	(product2, 12)	12	12
23	testWarehouseOperations3	(product3, 6)	6	6
24	testWarehouseOperations3	product1	8	8
25	testWarehouseOperations3	product2	12	12
26	testWarehouseOperations3	product3	6	6
27	testWarehouseOperations3	(product1, 3)	5	5
28	testWarehouseOperations3	(product2, 5)	7	7
29	testWarehouseOperations3	(product3, 4)	2	2
30	testWarehouseOperations3	100	TRUE	TRUE
31	testWarehouseOperations3	(product3.getProductId(), 99.99)	99.99	99.99
32	testWarehouseOperations3	-	1	1

## Expected Outcomes

1. Improved detection and resolution of software defects.
2. Enhanced collaboration and communication within testing teams.
3. Streamlined testing processes leading to faster release cycles.
4. Higher overall software quality and reliability.

## Summarization

Serial No	Class Name	Number of functions it contains	No of test cases we covered
1	Customer	26	34
2	Customer Manager	13	15
3	Employee	11	26
4	Employee Manager	9	31
5	Inventory	15	44
6	Inventory File Manager	4	6

7	Order	12	36
8	Order Manager	13	36
9	Product	11	30
10	Shopping Cart	12	3
11	Supplier	11	46
12	Transaction	14	38
13	Transaction Manager	11	36
14	Warehouse	9	32
<b>Total</b>	<b>14 Classes</b>	<b>171</b>	<b>413</b>

## Conclusion

The CSE430 Inventory Management System is a powerful tool designed to streamline inventory operations with 14 Java classes. It offers functionalities like customer and employee management, product tracking, order processing, and financial transactions. The system's success relies on comprehensive testing, using Eclipse and JUnit as development environments and code coverage analysis. This rigorous testing ensures the system's reliability, delivers real-time inventory visibility, automates tasks, and empowers data-driven decision making. The system can help businesses achieve greater efficiency, reduce costs and enhance customer satisfaction.