**White-Box Testing and TDD**

Homework #1

**COMP474: SOFTWARE ENGINEERING**

Presented by:

**Syeda Tashnuva Nowreen**

**Sudeshna Bhattacharyya**

Loyola University Chicago

Table of Contents

What is Test Driven Development? 3

White-Box Testing: 4

Code Coverage Criteria 4

Advantages 4

Disadvantages 5

Introduction 6

Description of the Project 6

Testing: 7

Flow Chart: 14

Branch Coverage Testing 15

Conditional Coverage Testing 15

Loop Coverage Testing 18

Code Description: 19

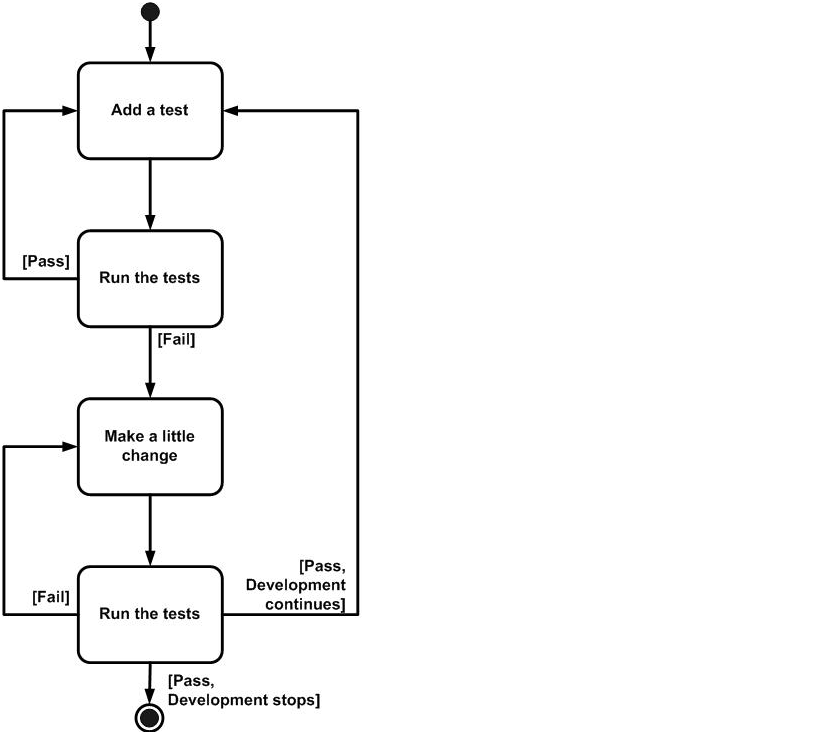
Class Customer: 21

Class MockDatabase: 21

# What is Test Driven Development?

Test-driven development is a method of software testing in which the functionality is tested first and then it is implemented. The concept here is to get something working first and perfect it later on. After each test refactoring is done and the process is repeated until the desired result is reached.

**Steps:**

****

# White-Box Testing:

White Box testing is a method of software testing that tests the internal structure of an application. In a white box testing the internal perspective of the system, the internal perspective of the system as well as the programing skills are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. White Box testing can be applied to Unit, Integration and system levels of the software testing process.

## Code Coverage Criteria

* Control flow testing
* Data flow testing
* Branch testing
* Statement coverage
* Decision coverage
* Modified condition
* Prime path Testing
* Path testing

## Advantages

1. Side effects of having the knowledge of the source code is beneficial to thorough testing.
2. Optimization of code by revealing hidden errors and being able to remove these possible defects.
3. Gives the programmer introspection because developers carefully describe any new implementation.
4. Provides traceability of tests from the source, allowing future changes to the software to be easily captured in changes to the tests.
5. White box tests are easy to automate.
6. White box testing gives clear, engineering-based, rules for when to stop testing.

## Disadvantages

Although white-box testing has great advantages, it is not perfect and contains some disadvantages:

1. White-box testing brings complexity to testing because the tester must have knowledge of the program, including being a programmer. White-box testing requires a programmer with a high level of knowledge due to the complexity of the level of testing that needs to be done.
2. On some occasions, it is not realistic to be able to test every single existing condition of the application and some conditions will be untested.
3. The tests focus on the software as it exists, and missing functionality may not be discovered.

# Introduction

Here for the purpose of this Project we were given specific set of requirements for an organization’s billing process with which we have to come up with some billing solutions for an organization. To deal with the requirements we created some classes and some methods that helped us to arrive to a result that we wanted to. Below mentioned are the description of the classes that had different methods in them to calculate the specifications according to the requirements.

We have developed an inventory and purchasing system for a local retailer. We have followed test-drive development (TTP) principal and done JUnit testing that provide statement, branch, conditional, and loop coverage. That also covers equivalence partitioning (EP) and boundary value analysis (BVA) techniques. We made a mock database that populates mock objects.

# Description of the Project

**Object-Oriented (OO) Language**: Java

**Tools Used**: Eclipse IDE, Android Developers Tool, JUnit, Astah (for class diagram)

**Code Repository:** [**https://github.com/snowreen/white-box-testing**](https://github.com/snowreen/white-box-testing)

**How to import the Code to IDE:** Here we used Eclipse as IDE. At first, From “File” select “New” -> “Java Project”, then uncheck “use default location” box. Now click on “Browse” beside “Location” text box, and select “white-box-testing” folder inside the unzipped file that is provided with this project submission. Then click “Open”. Now you should have this project imported in your Eclipse. (Screenshot attached with the project submission).

**How to Run the Code:** Right click on the **ShoppingCartTest.java** class inside “src/test/java/com/whitebox/testing” and run it as JUnit (Screenshot attached with the project submission).

**Bugs Uncovered during TDD:**

At first, we wrote down all the criteria needs to be covered. Then we started writing tests and implemented the method at the same time. As we progress, we were keeping track of logics that we cover in our testing. But still we have faced some difficulties as some bugs show up.

1. At first we missed the condition where list of product ids can be null or its size can be 0. When we were passing null value, it was throwing NullPointerException. To fix it, we added a check in the beginning, which will not allow any null or 0 size list to be going further into the code.
2. Another bug we figured out was miscalculating the round-off price. Instead of rounding the price to the nearest cent, we were just showing one digit after decimal point. We fixed that by using “Math.round(value\*100.0)/100.0”.
3. Another critical bug we found, we were not properly using if-else-if statement. We checked if the number of products is equal to 10 or greater than 10, and then used only “else” to do the other logic which was specified for greater than 5 but less than 10. But this is completely wrong, and we figured this out during writing tests. We fixed it by using “if-else-if” statement.

# Testing:

We tested the application in different circumstances with different specification and instances. We did perform 33 Tests to make sure that we have covered some amount of combinations possible and give the maximum coverage. The testes are as followed:

1. **+testCalcPurchasePriceWhenItemCountExactTenForGeneralCustomerTaxExempt() : void**

Test for cart contains exact 10 items, the general tax exempt customers gets a 10% discount and no tax applied

1. **+testCalcPurchasePriceWhenItemCountExactTenForGeneralCustomerNonTaxExempt() : void**

Test for cart contains exact 10 items, the general non tax exempt customers gets a 10% discount and 4.5% tax applied

1. **+testCalcPurchasePriceWhenItemCountExactTenForMemberCustomerTaxExempt() : void**

Test for cart contains exact 10 items, the member tax exempt customers gets a 10% discount for number of items and then additional 10% discount for being member of discount shopping club and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountExactTenForMemberCustomerNonTaxExempt(): void**

Test for cart contains exact 10 items, the member non tax exempt customers gets a 10% discount for number of items and then additional 10% discount for being member of discount shopping club and 4.5% tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanTenForGeneralCustomerTaxExempt() : void**

Test for cart contains more than 10 items, the general tax exempt customers gets a 10% discount and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanTenForGeneralCustomerNonTaxExempt() : void**

Test for cart contains more than 10 items, the general non tax exempt customers gets a 10% discount and 4.5% tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanTenForMemberCustomerTaxExempt() : void**

Test for cart contains more than 10 items, the member tax exempt customers gets a 10% discount for number of items and then additional 10% discount for being member of discount shopping club and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanTenForMemberCustomerNonTaxExempt() : void**

Test for cart contains more than 10 items, the member non tax exempt customers gets a 10% discount for number of items and then additional 10% discount for being member of discount shopping club and 4.5% tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanFiveButLessThanTenForGeneralCustomerTaxExempt(): void**

Test for cart contains more than 5 but less than 10 items, the general tax exempt customers gets a 5% discount and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanFiveButLessThanTenForGeneralCustomerNonTaxExempt(): void**

Test for cart contains more than 5 but less than 10 items, the general non tax exempt customers gets a 5% discount and 4.5% tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanFiveButLessThanTenForMemberCustomerTaxExempt(): void**

Test for cart contains more than 5 but less than 10 items, the member tax exempt customers gets a 5% discount and then additional 10% discount for being member of discount shopping club and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanFiveButLessThanTenForMemberCustomerNonTaxExempt(): void**

Test for cart contains more than 5 but less than 10 items, the member non tax exempt customers gets a 5% discount and then additional 10% discount for being member of discount shopping club and 4.5% tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanFifty() : void**

Test for cart contains more than 50 items, should throw IllegalArgumentException.

1. **+testCalcPurchasePriceWhenNoItem(): void**

Test for cart contains no item, should return 0.

1. **+testCalcPurchasePriceWhenProductIdListIsNull(): void**

When list product id is null, should return 0.

1. **+testCalcPurchasePriceForCartWithTenItems(): void**

Test for net total without tax and without discount applied for cart with 10 items.

1. **+testCalcPurchasePriceForCartWithMoreThanFiveButLessThanTenItems(): void**

Test for net total without tax and without discount applied for cart between 5 and 10 items.

1. **+testCalcTaxAmtForCartWithTenItemsForTaxExemptCustomer(): void**

Test for tax amount for cart with 10 items for tax exempt customer.

1. **+testCalcTaxAmtForCartWithMoreThanFiveButLessThanTenItemsForTaxExemptCustomer(): void**

Test for tax amount for cart with more than 5 but less than 10 items for tax exempt customer.

1. **+testCalcTaxAmtForCartWithTenItemsForNonTaxExemptCustomer(): void**

Test for tax amount for cart with 10 items for non tax exempt customer.

1. **+testCalcTaxAmtForCartWithMoreThanFiveButLessThanTenItemsForNonTaxExemptCustomer(): void**

Test for tax amount for cart with more than 5 but less than 10 items for non tax exempt customer.

1. **+testCalcTaxAmtForCartWithTenItemsForMemberTaxExemptCustomer(): void**

Test for tax amount for cart with 10 items for member tax exempt customer.

1. **+testCalcTaxAmtForCartWithMoreThanFiveButLessThanTenItemsForMemberTaxExemptCustomer(): void**

Test for tax amount for cart with more than 5 but less than 10 items for member tax exempt customer.

1. **+testCalcTaxAmtForCartWithTenItemsForMemberNonTaxExemptCustomer(): void**

Test for tax amount for cart with more than 5 but less than 10 items for member non tax exempt customer.

1. **+testCalcTaxAmtForCartWithMoreThanFiveButLessThanTenItemsForMemberNonTaxExemptCustomer(): void**

Test for tax amount for cart with more than 5 but less than 10 items for member non tax exempt customer.

1. **+testCalcDiscountAmtForCartWithTenItemsForGeneralCustomer(): void**

Test for discount amount for cart with 10 items for general customer.

1. **+testCalcDiscountAmtForCartWithMoreThanFiveButLessThanTenItemsForGeneralCustomer(): void**

Test for discount amount for cart with more than 5 but less than 10 items for general customer.

1. **+ testCalcDiscountAmtForCartWithTenItemsForMemberCustomer(): void**

Test for discount amount for cart with 10 items for member customer.

1. **+testCalcDiscountAmtForCartWithMoreThanFiveButLessThanTenItemsForMemberCustomer(): void**

Test for discount amount for cart with more than 5 but less than 10 items for member customer.

1. **+testCalcPurchasePriceWhenItemCountMoreThanZeroButLessThanFiveForGeneralCustomerTaxExempt(): void**

Test for cart contains more than 0 but less than 5 items, the general tax exempt customers gets no discount and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanZeroButLessThanFiveForGeneralCustomerNonTaxExempt(): void**

Test for cart contains more than 0 but less than 5 items, the general non tax exempt customers gets 4.5% tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanZeroButLessThanFiveForMemberCustomerTaxExempt(): void**

Test for cart contains more than 0 but less than 5 items, the member tax exempt customers gets 10% discount for being member of discount shopping club and no tax applied.

1. **+testCalcPurchasePriceWhenItemCountMoreThanZeroButLessThanFiveForMemberCustomerNonTaxExempt(): void**

Test for cart contains more than 0 but less than 5 items, the member non tax exempt customers gets then additional 10% discount for being member of discount shopping club and 4.5% tax applied.

All of above tests passed successfully.

We have tried both Partition testing and Guideline testing.

Partition Testing: We have created partition in all possible input values with common characteristics that should be processed in the same way. And did one tests from within each of these groups with different combination of member/not, tax exempt/not.

When **numOfItems=10:**

**numOfItems=10; Not member, Not Tax exempt**

**numOfItems=10; Not member, Tax exempt**

**numOfItems=10; member, Not Tax exempt**

**numOfItems=10; Member, Tax exempt**

When **10=**<**numOfItems<50:**

**numOfItems=12; Not member, Not Tax exempt**

**numOfItems=12; Not member, Tax exempt**

**numOfItems=12; member, Not Tax exempt**

**numOfItems=12; Member, Tax exempt**

When **5**<**numOfItems<10:**

**numOfItems=7; Not member, Not Tax exempt**

**numOfItems=7; Not member, Tax exempt**

**numOfItems=7; member, Not Tax exempt**

**numOfItems=7; Member, Tax exempt**

When **0**<**numOfItems<5:**

**numOfItems=5; Not member, Not Tax exempt**

**numOfItems=5; Not member, Tax exempt**

**numOfItems=5; member, Not Tax exempt**

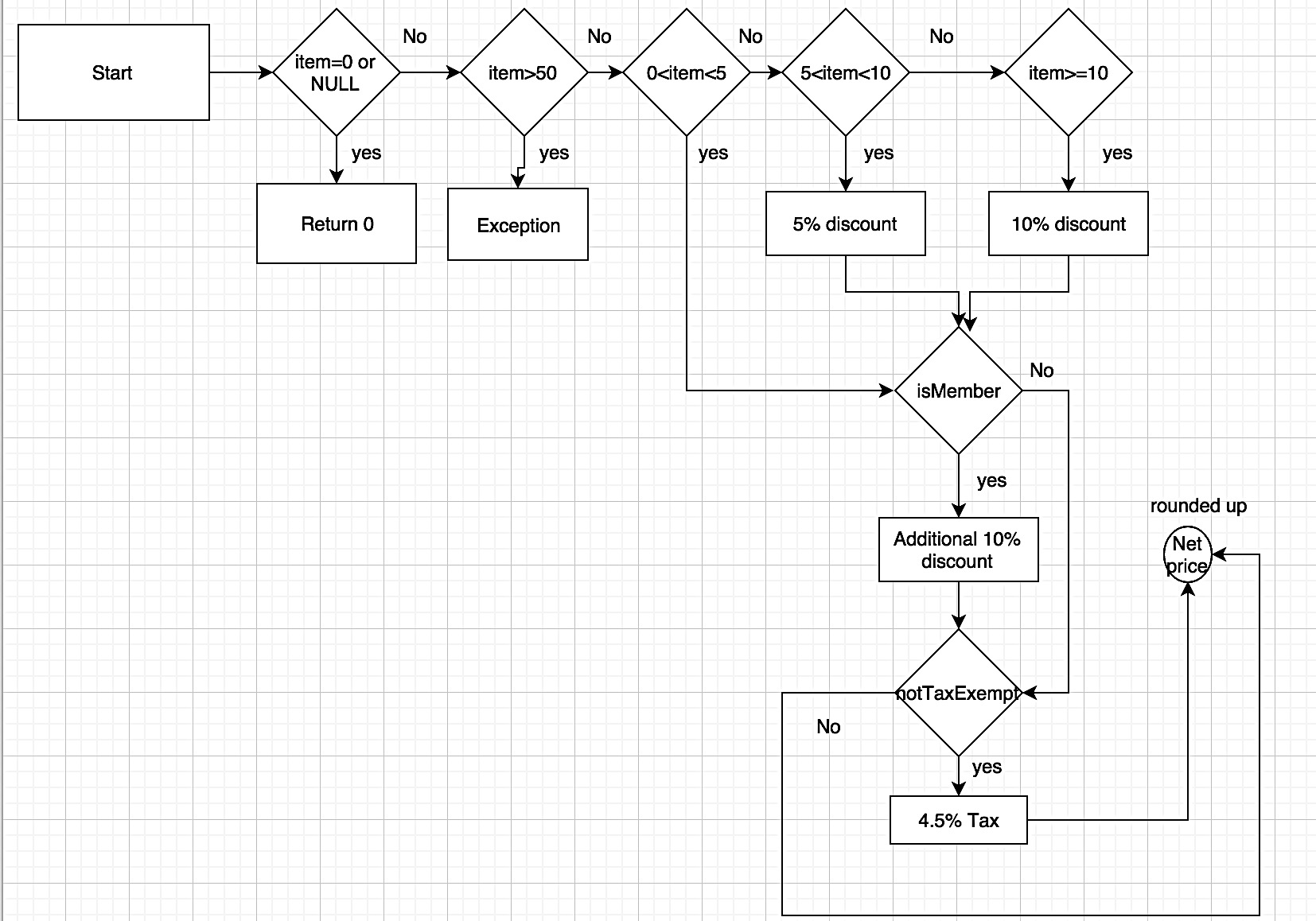
**numOfItems=5; Member, Tax exempt**

Guideline Testing: We have also tested for null lists or empty lists to avoid the common programming errors.

**numOfItems=NULL**  
**numOfItems=0**  
**numOfItems=60**

For extra coverage we have also tested each of our calculated price individually (with/without discount and tax).

## Flow Chart:

****

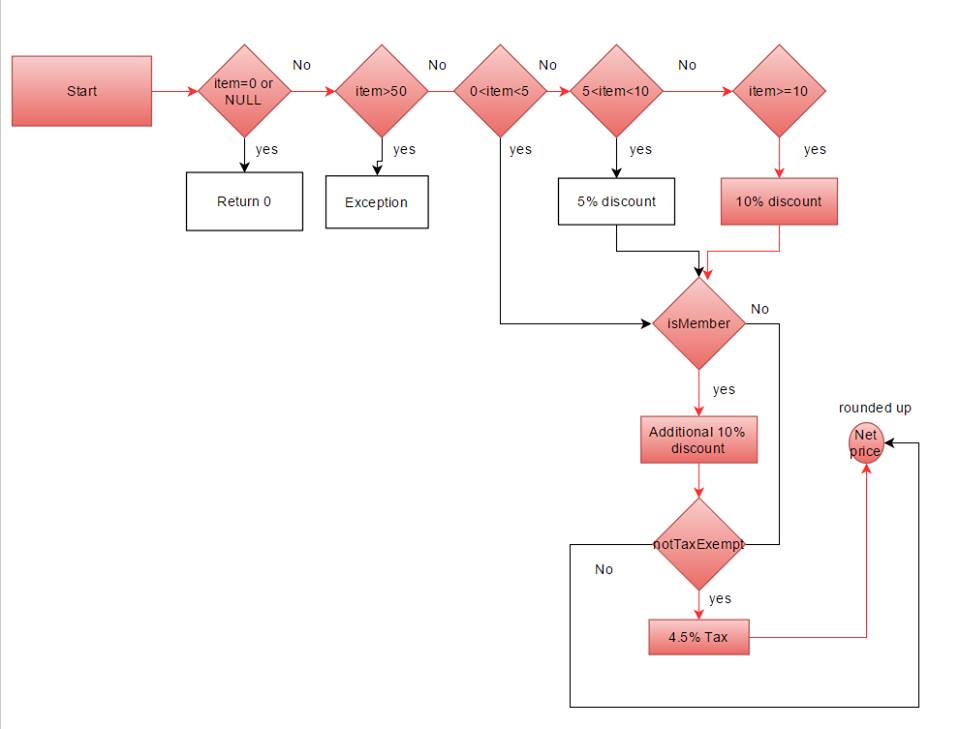
Our tests covered **Branch Coverage, Loop Coverage, Conditional Coverage testing**.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Test Example | Result | Coverage |
| **Branch Coverage** | **numOfItems=12; member, Not Tax exempt** | Passed | 80% |
| **Loop Coverage** | numOfItems=0 | Passed | 20% |
| **Conditional Coverage** | All possible logics | All Passed | 100% |

## Branch Coverage Testing

For, **numOfItems=12; member, Not Tax exempt** the flow chart will be as below:

This Branch Coverage testing will give 80% coverage.

****

## Conditional Coverage Testing

Which gives almost 100% coverage

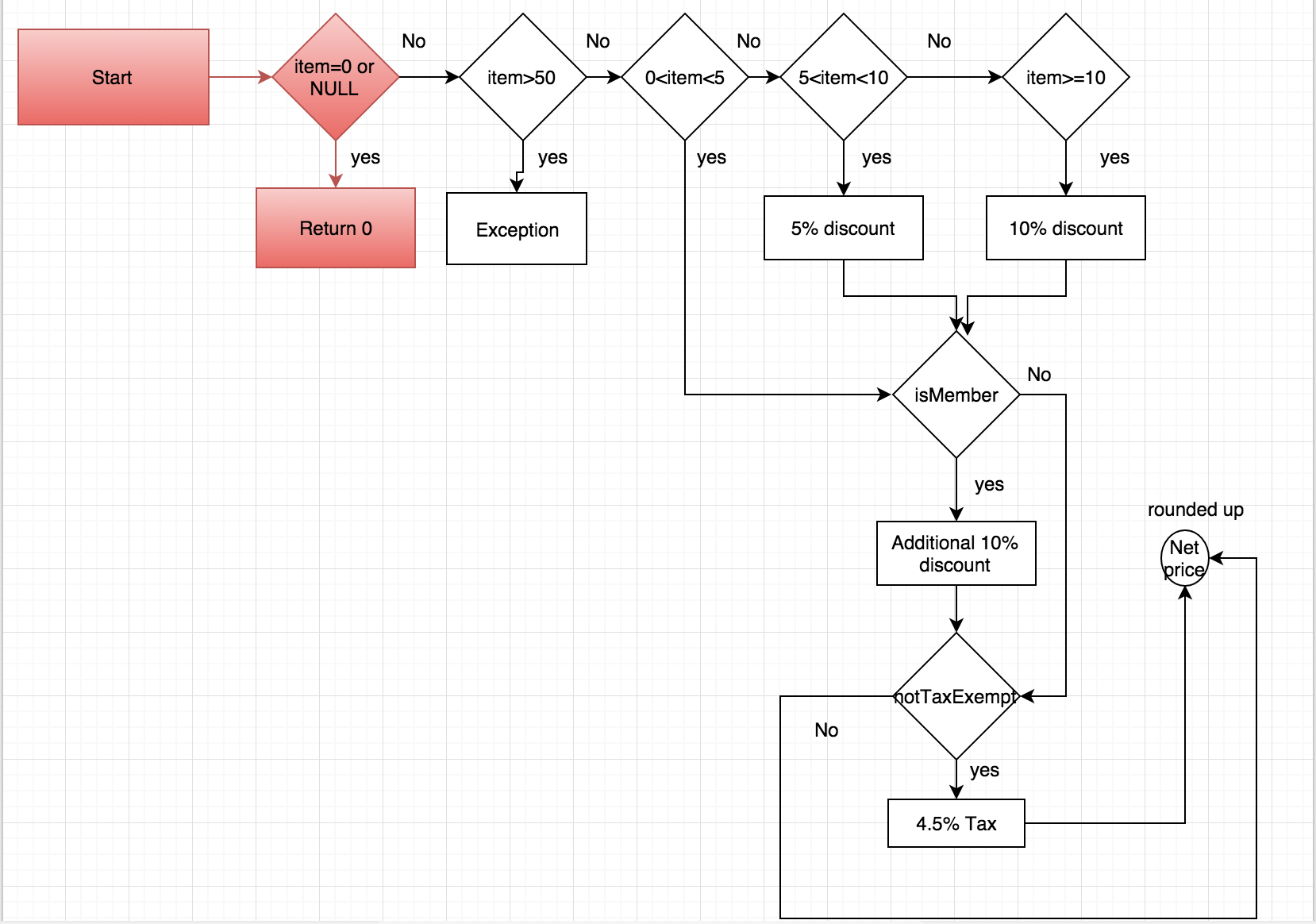
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test # | numOfItems= null OR numOfItems = 0 | numOfItems >50 | numOfItems>0 AND numOfItems<5 | numOfItems>5 AND numOfItems<10 (gets 5% Discount) | numOfItems=> 10  (Gets 10% Discount) | Member’s additional 10% discount | Not Tax Exempt |
| 1 numOfItems=0 | T | - | - | - | - | - | - |
| 2.numOfItems=0 | T | - | - | - | - | - | - |
| 3. numOfItems=60 | F | T | - | - | - | - | - |
| 4. numOfItems=10;  Not member, Not Tax exempt | F | F | F | F | T | F | T |
| 5. numOfItems=10;  Not member, Tax exempt | F | F | F | F | T | F | F |
| 6.numOfItems=10;  member, Not Tax exempt | F | F | F | F | T | T | T |
| 7. numOfItems=10;  Member, Tax exempt | F | F | F | F | T | T | F |
| 8.numOfItems=12;  Not member, Not Tax exempt | F | F | F | F | T | F | T |
| 9. numOfItems=12;  Not member, Tax exempt | F | F | F | F | T | F | F |
| 10.numOfItems=12;  member, Not Tax exempt | F | F | F | F | T | T | T |
| 11. numOfItems=12;  Member, Tax exempt | F | F | F | F | T | T | F |
| 12. numOfItems=7;  Not member, Not Tax exempt | F | F | F | T | F | F | T |
| 13. numOfItems=7;  Not member, Tax exempt | F | F | F | T | F | F | F |
| 14.numOfItems=7;  member, Not Tax exempt | F | F | F | T | F | T | T |
| 15. numOfItems=7;  Member, Tax exempt | F | F | F | T | F | T | F |
| 16. numOfItems=4;  Not member, Not Tax exempt | F | F | T | F | F | F | T |
| 17. numOfItems=4;  Not member, Tax exempt | F | F | T | F | F | F | F |
| 18.numOfItems=4;  member, Not Tax exempt | F | F | T | F | F | T | T |
| 19. numOfItems=4;  Member, Tax exempt | F | F | T | F | F | T | F |

## 

## Loop Coverage Testing

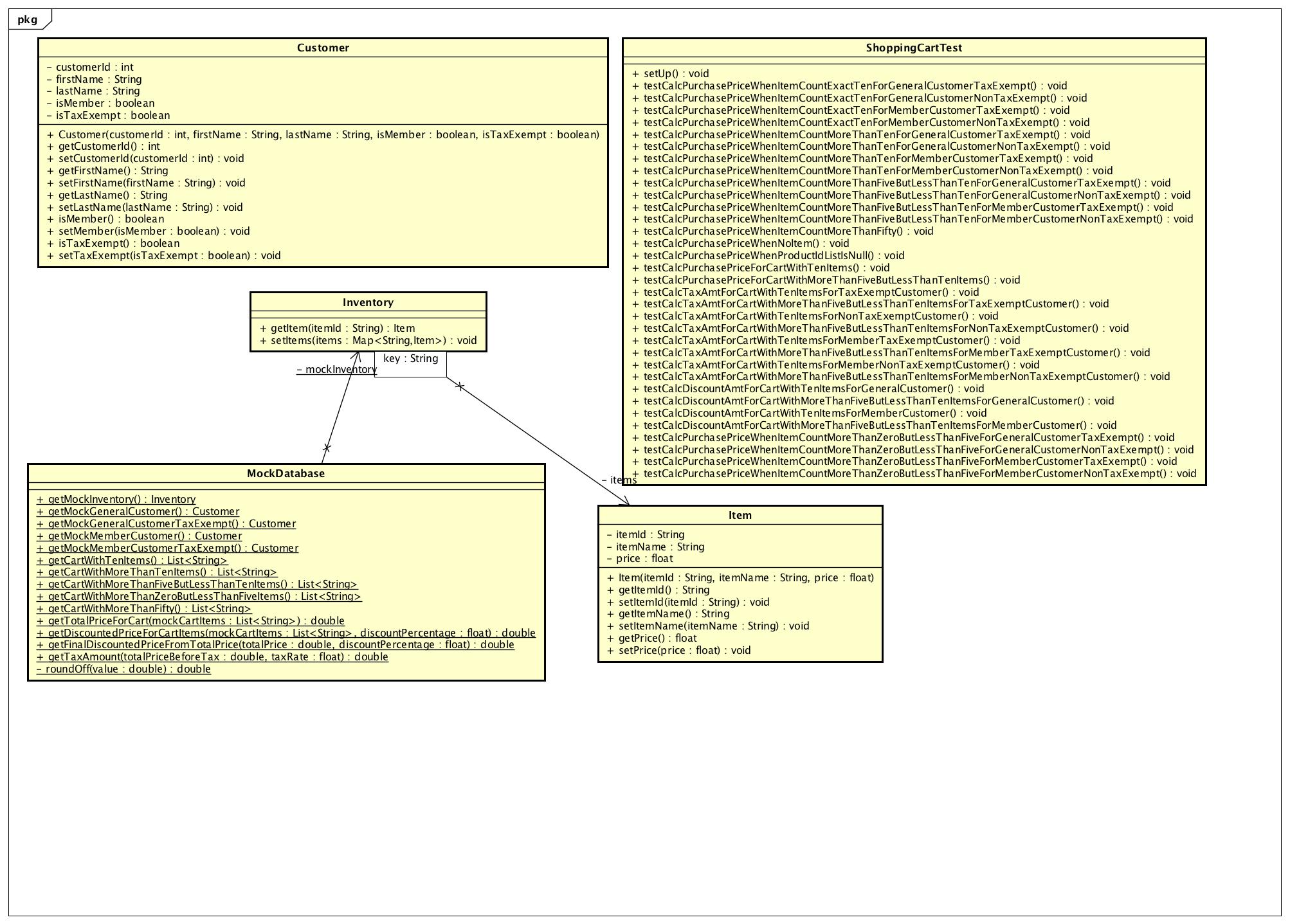
For, **numOfItems=0,** **member, Not Tax exempt** the flow chart will be as below:

This Branch Coverage testing will give 20% coverage.

****

# Code Description:

Class Diagram**:** (clear picture attached separately)

****

## Class Customer:

We created a class called Customer, which will contain all the details of the customer in it. It has 5 basic private attributes to store all the details about the customer.

**- customerId** : int

**- firstName** : string

**- lastName** : string

**- isMember** : Boolean

**- isTaxExempt** : Boolean

Class Inventory:In **Inventory** class we have used Map to store mock item where item name is key and item object is the value. We also have 2 public methods

1. **+ getItem(itemId : String) : Item**

This will return the item id of the specific item.

**2. + setItem(items : Map<String, Item>) : void**

Class Item:In Item class we described all the details about the item. This class has 3 private attributes

**- itemId : string**

**- itemName : string**

**- price : float**

Also has 1 constructor and 6 getter and setter.

+**item (itemId, itemname and price)**

## Class MockDatabase:

This class is to create mock data that will be used to test different criteria. This object is a mock inventory and we are populating with the mock items as well as mock customers. Also calculating mock items price with different type of discounts and tax.

Shopping Cart:  
If we imagine ourselves walking up to store, we all get a shopping cart that contains items that we picked up from the store and needs to be billed. Here too we are doing the same thing. We are getting the items from the inventory and then calculating the individual prices of them using their item id. Then we are getting the total price after adding all the items in the cart. Now at this stage there are certain things that we have taken care here. We made sure that the cart should not have more than 50 items at any point of time as per the requirement. Now, we are calculating the total discount here. If the cart has items between 5-9 a 5 % discount will be honored on the overall purchase, where as if the cart contains more than 10 items then an overall 10 % discount is applied on the entire purchase. On top of that we have also calculated the member’s discount that is an additional 10% on the already discounted price. This member’s discount is valid even if the shopping cart contains less than 5 items. If the member is a tax-exempt, we can simply get the total amount from here by just nearing off to the nearest $0.01 at this point of time. If not then we will add a flat 4.5% on the entire purchase and get the tax amount. Then we get a total amount by adding the Total purchase before tax with the sales tax and get the total amount. Now we are all set to get 4 major requirements that we need subtotal amount, discount, sales tax, and total amount. At this point of time we also generated a bill by printing all the four required specifications. Lastly we calculated the total tax and rounded it to the nearest $0.01.