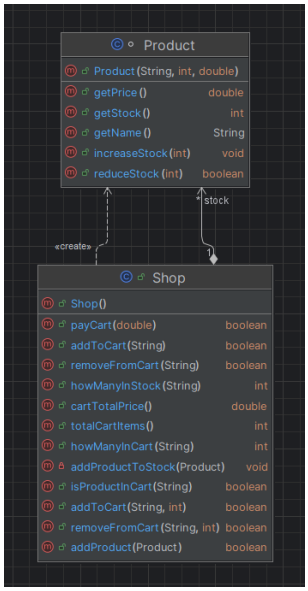
Exercise 1 – Test Case Design



These tests are designed for the Shop class in Exercise 1. The Shop class is responsible for:

- Instantiating a new Shop object
- Adding products to stock and the shopping cart
- Removing products from the shopping cart
- Handling payment of products in the shopping cart

No specification was provided for this exercise.

The Shop class consists of 12 methods to handle a simple shop. In addition, this exercise contains a product class that handles product creation and the reduction and increase of stock.

Black and Whitebox Analysis

Boundary Value Analysis

Method	not allowed	allowed	not allowed	datatype
addProduct (price)	<0.00	<= 0.00		double
addProduct (amount)	<0	0<		int
addToCart	<0	0,<=stock	stock<	int
removeFromCart	<0	0,<=inCart	inCart<	int
payCart	<cartTotalPrice	cartTotalPrice=<		double

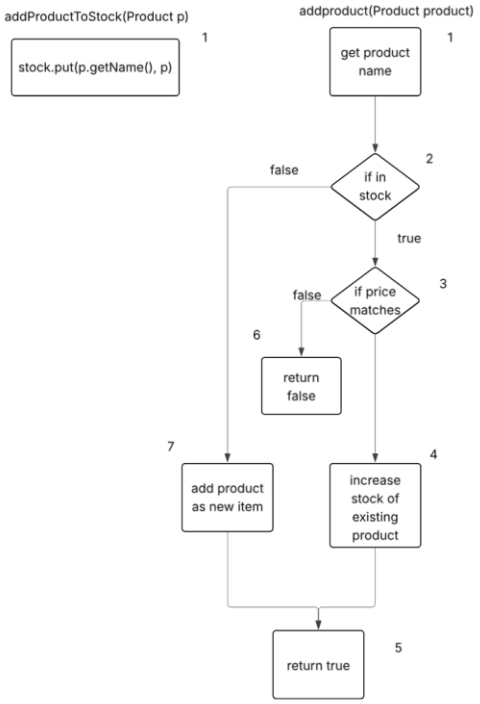
Here is a quick overview of numeric input values in the Shop class. The Product class also handles numeric values, but it is outside the scope of this exercise.

This table also includes a column for the datatypes.

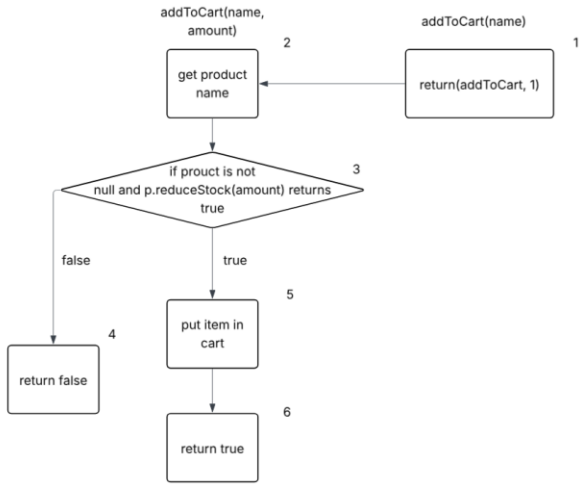
Code Coverage Analysis

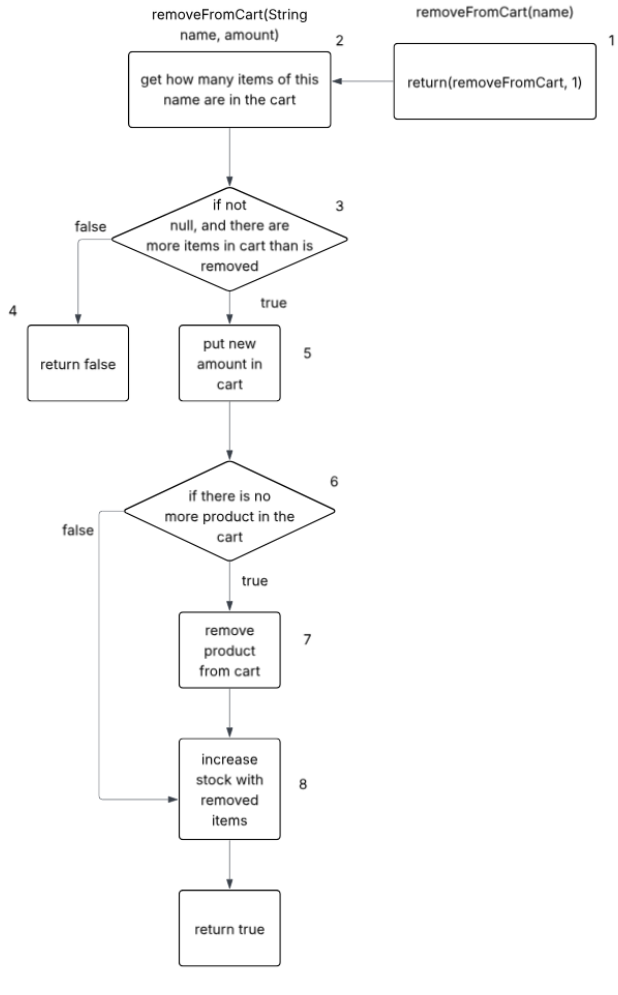
This report includes flowcharts for the central methods in the Shop class, such as adding products to stock, adding products to the cart, removing products from cart and paying for the cart. These charts illustrate the flow of the programs, but do not include a deeper analysis such as DU-pairs and flows to smaller helper methods.

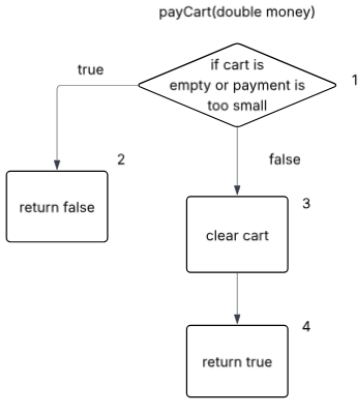
For this task, focus is on the larger methods.


addProductToStock(Product p)
1
stock.put(p.getName(), p)

addproduct(Product product)
1
get product name
2
if in stock
false
true
3
if price matches
false
6
return false
7
add product as new item
4
increase stock of existing product
5
return true


addToCart(name, amount)
2
get product name
addToCart(name)
1
return(addToCart, 1)
3
if prouct is not null and p.reduceStock(amount) returns true
false
true
4
return false
5
put item in cart
6
return true


removeFromCart(String name, amount)
2
get how many items of this name are in the cart
removeFromCart(name)
1
return(removeFromCart, 1)
3
if not null, and there are more items in cart than is removed
false
true
4
return false
5
put new amount in cart
6
if there is no more product in the cart
false
true
7
remove product from cart
8
increase stock with removed items
return true


payCart(double money)
if cart is empty or payment is too small
1
true
2
return false
false
3
clear cart
4
return true

Methods	Paths	Paths	Start nodes	End nodes
addProduct	[1,2,3,4,5],[1,2,7,5][1,2,3,6]	3	1	5,6
addToCart(name)	[1]	1	1	1
addToCart(name, amount)	[2,3,4],[2,3,5,6]	2	2	4,6
removeFromCart(name)	[1]	1	1	1
removeFromCart(name,amount)	[2,3,4],[2,3,5,6,8,9],[2,3,5,6,7,8,9]	3	2	4,9
payCart(money)	[1,2],[1,3,4]	2	1	2,4

These are the possible paths in the methods that are investigated.

Test Cases

These tests ensures decent coverage of the chosen methods in the Shop Class, as they explore each node in the methods at least once.

At this point, the overloading methods addToCart and removeFromCart have been excluded. Also, the [2,3,5,6,8,9] path of removeFromCart has been excluded, as the same nodes are covered by the [2,3,5,6,7,8,9] path.

Code Coverage Test Paths

Test Case	Mehthod	Path	Precondition	Input	Expected Output/State
CC1	addProductToStock	[1]	create product object "Chocolate", 10, 2.50	"Chocholate"	10 chocolates in stock
CC2	addProduct	[1,2,3,4,5]	create new Egg prduct object	object: "Egg", 5, 0.20	15 eggs in stock
CC3	addProduct	[1,2,7,5]	create product object "Raspberries", 10, 3.00	String: "Raspberries"	10 raspberries in stock
CC4	addProduct	[1,2,3,6]	create prioduct object "Egg", 5, 0.19	object: "Egg", 5, 0.19	10 eggs in stock
CC5	addToCart(name,amount)	[2,3,4]		"Egg", 20	false
CC6	addToCart(name,amount)	[2,3,4]		"Pancakes", 5	false
CC7	addToCart(name,amount)	[2,3,5,6]		"Egg", 5	true, stock reduced
CC8	removeFromCart(name,amount)	[2,3,4]	empty cart	"Egg", 5	false
CC9	removeFromCart(name,amount)	[2,3,4]	"Egg", 5 in cart	"Egg", 0	false
CC10	removeFromCart(name,amount)	[2,3,5,6,7,8,9]	"Egg", 5 in cart	"Egg", 5	cart empty, stock: "Egg", 10
CC11	payCart(money)	[1,2]	empty cart	0.20	false
CC12	payCart(money)	[1,2]	"Egg", 5 in cart	0.20	false
CC13	payCart(money)	[1,3,4]	"Egg", 5 in cart	1.00	true, empty cart

Test Case	Description
CC1	Add a new product to the stock
CC2	Increases stock when a new product with same price and name is added
CC3	Add product as a new stock item when name is not found in stock
CC4	Return false is existing product name does not have the same price
CC5	Add more products to cart than is in stock
CC6	Add product that does not exist to cart
CC7	Add existing product that does not exceed stock
CC8	Remove unexisting products from cart
CC9	Remove 0 products from cart
CC10	Remove all products from the cart, update stock with removed items
CC11	Payment when cart is empty
CC12	Paying less than cart has
CC13	Pay the contents of the cart with correct ammount of money

BVA & ISP

TestCase	Method	Precondition	Input	Expected output/state
BVA1	addProduct		"Pancakes", 0, 1.70	true, "Pancakes", 0, 1.70 in stock
BVA2	addProduct		"Pancakes", 1, 1.70	true, "Pancakes", 1, 1.70 in stock
BVA3	addProduct		"Pancakes", -1, 1.70	false
BVA4	addProduct		"Pancakes", 1, 0.00	true, Pancakes, 1, 0.00
BVA5	addProduct		"Pancakes", 1, -0.01	false
BVA6	addToCart	product exists	"Egg", -1	false, empty cart
BVA7	addToCart	product exists	"Egg", 0	false, empty cart
BVA8	addToCart	product exists	"Egg", 1	true, 1 egg in cart
BVA9	addToCart	exceeds stock	"Egg", 11	false, empty cart
BVA10	removeFromCart	5 eggs in cart	"Egg", -1	false, 5 eggs in cart
BVA11	removeFromCart	5 eggs in cart	"Egg", 0	false, 5 eggs in cart
BVA12	removeFromCart	5 eggs in cart	"Egg", 5	true, empty cart
BVA13	removeFromCart	5 eggs in cart	"Egg", 6	false, 5 eggs in cart
BVA14	payCart	2.00 total in cart	1,99	false
BVA15	payCart	2.00 total in cart	2,00	true, empty cart
BVA16	payCart	2.00 total in cart	2,01	true, empty cart

Chosen test cases for Junit testing

Test	Method(s)	Input	Expected Output/State
addValidProductToShop	addProduct	"Pancakes", 0, 1.70	howManyInStock=0
addExistingProductToShopDifferentPrice	addProduct	"Egg", 10, 0.19	false
addExistingProductToShopSamePrice	addProduct	"Egg", 10, 0.20	howManyInStock=20
addValidProductsToCart	addToCart	"Egg", 5	howManyInStock=5, howManyInCart=5
addMoreProductsThanExistsToCart	addToCart	"Egg", 5	false
removeTooManyProductsFromCart	addToCart, removeFromCart	"Egg", 11	false
removeProductsFromCart	addToCart, removeFromCart	Egg, 5	howManyInStock=10, howManyInCart=0
payCorrectAmount	addToCart, payCart	1,00	true, cart is empty
payWrongAmount	addToCart, payCart	0,99	false, 5 eggs in cart
addNewProductNegativeValueNegativePrice	addProduct	"Pancakes", -1, -1.70	false
addNegativeAmountToCart	addToCart	"Egg", -1	false

UseCaseTest	Method(s)	Input	Expected Output/State
buyMultipleProductsIndecisiveCustomer	addToCart, removeFromCart, payCart	"Egg", 4, "Toilet Paper", 20.00	true

- addValidProductToShop
  - This test adds a new product to the shop but initially has zero items in stock.
- addExistingProductToShopDifferentPrice
  - Add a product with the same name as an existing product, but the new product has a different price than the old product. Should return false.
- addExistingProductToShopSamePrice
  - Add a new product with the same name as an existing product, the new product has the same price as the old product. Should increase the stock of pre-existing product.
- addValidProductsToCart
  - When products are added to the cart the stock and cart states should be updated.
- addMoreProductsThanExistsToCart
  - The user should not be able to add more products than the current stock to the cart.
- removeTooManyProductsFromCart
  - The user should not be able to remove more products than they have in the cart.

- removeProductsFromCart
    - When all products are removed from the stock and cart states should be updated accordingly.
- payCorrectAmount
    - A payment equal to the cart value should result in a successful payment and empty the cart.
- payWrongAmount
    - A payment that is less than the cart value should fail, cart should not be emptied.
- addNewProductNegativeValueNegativePrice
    - This test checks whether the system allows for products with negative prices and stocks.
- addNegativeAmountToCart
    - This test checks whether the system allows for a negative item value in the cart.
- butMultipleProuctsIndecisiveCustomer
    - This use case test simulates a scenario where a user adds and removes products before check out. Additionally, this test checks if a payment larger than the cart amount is accepted. It should be checked whether this is allowed in the system, if it checks the account balance or if the system withdraws this amount.

Results

Test	Method(s)	Input	Expected Output/State	Pass/Fail	Post Bugfix
addValidProductToShop	addProduct	"Pancakes", 0, 1.70	howManyInStock=0	pass	pass
addExistingProductToShopDifferentPrice	addProduct	"Egg", 10, 0.19	false	pass	pass
addExistingProductToShopSamePrice	addProduct	"Egg", 10, 0.20	howManyInStock=20	fail	pass
addValidProductsToCart	addToCart	"Egg", 5	howManyInStock=5, howManyInCart=5	fail	pass
addMoreProductsThanExistsToCart	addToCart	"Egg", 5	false	pass	pass
removeTooManyProductsFromCart	addToCart, removeFromCart	"Egg", 11	false	pass	pass
removeProductsFromCart	addToCart, removeFromCart	Egg, 5	howManyInStock=10, howManyInCart=0	fail	pass
payCorrectAmount	addToCart, payCart	1,00	true, cart is empty	fail	pass
payWrongAmount	addToCart, payCart	0,99	false, 5 eggs in cart	fail	pass

UseCaseTest	Method(s)	Input	Expected Output/State	Pass/Fail	Post Bugfix
buyMultipleProductsIndecisiveCustomer	addToCart, removeFromCart, payCart	"Egg", 4, "Toilet Paper", 20.00	true	fail	pass

Tests that need to be checked up	Method(s)	Input	Expected Output/State	Pass/Fail	Post Bugfix
addNewProductNegativeValueNegativePrice	addProduct	"Pancakes", -1, -1.70	false	fail	fail
addNegativeAmountToCart	addToCart	"Egg", -1	false	fail	fail

The bugs and bugfixes are detailed in the bug report, which also includes questions that need to be answered before further development.

It is recommended to write more Junit tests based on the paths and BVA analysis in this report.