SOFTWARE TESTING AND VALIDATION 2019/2020

INSTITUTO SUPERIOR TÉCNICO

PostOffice

PROJECT REPORT

GRUPO *5*

Raquel Pereira 76171

Pedro Rodrigues 8354

April 10, 2020

Índice

Method Scope Test 3
computeCreditBill () 3
addNewProduct (Product p) 6
Class Scope Test 7
PostOffice class 7
Invoice class 9
TestNG Test Cases 14
PostOffice class 14

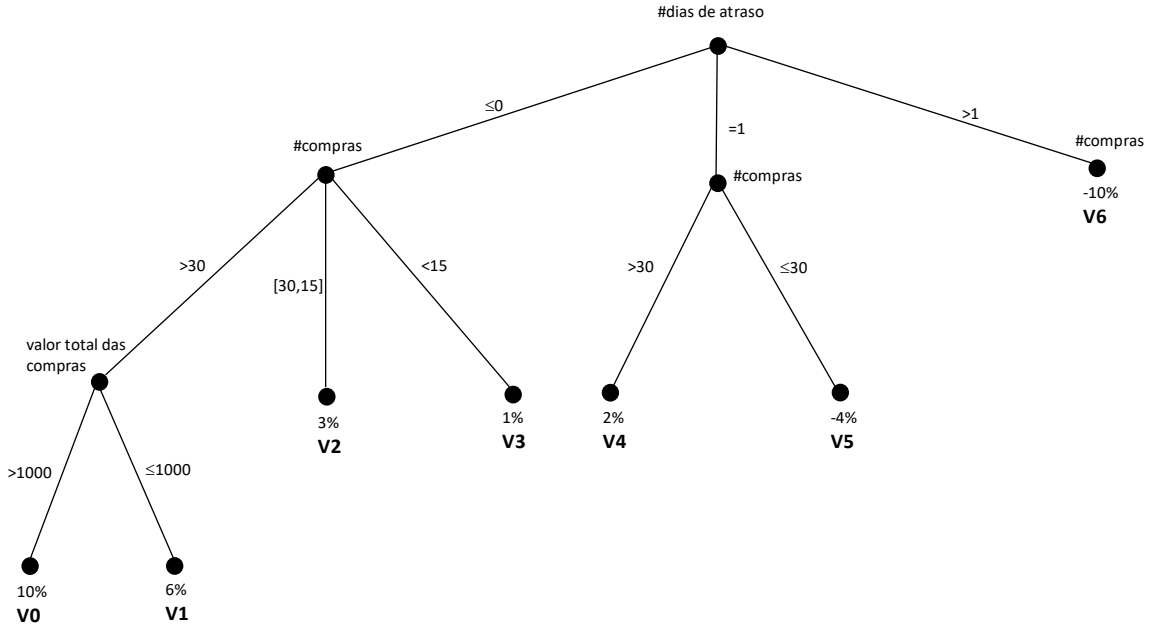
Method Scope Test

computeCreditBill ()

This method calculates the value that a client has to pay from a purchase made by credit. The value to pay depends on multiple reasons: the number of purchases and the total value of the purchases already done by this client, and if the payment is being done until the deadline or after.

Test Pattern - Combinational Functional Test

1. Decision Tree



Decision tree that describes all the combinations of the discount value

V0 → #dias de atraso ≤ 0 ∧ #compras > 30 ∧ valor total de compras > 1000	10% desconto
V1 → #dias de atraso ≤ 0 ∧ #compras > 30 ∧ valor total de compras ≤ 1000	6% desconto
V2 → #dias de atraso ≤ 0 ∧ 15 ≤ #compras ≤ 30	3% desconto
V3 → #dias de atraso ≤ 0 ∧ #compras < 15	1% desconto
V4 → #dias de atraso = 1 ∧ #compras > 30	2% desconto
V5 → #dias de atraso = 1 ∧ #compras ≤ 30	-4% desconto
V6 → #dias de atraso > 1	-10% desconto

2. Domain Analysis

V0			Test Cases					
var	condition		1	-	-	2	-	3
#dias de atraso	≤0	ON	0					
		OFF		1				
		IN			-3	-5	-6	-10
#compras	>30	ON			30			
		OFF				31		
		IN	35	40			50	100
valor total de compras	>1000	ON					1000	
		OFF						1001
		IN	2000	1500	1050	2050		
Expected Results			10%	V4	V2	10%	V1	10%

V1			Test Cases					
var	condition		1	-	-	2	3	-
#dias de atraso	≤0	ON	0					
		OFF		1				
		IN			-3	-5	-6	-10
#compras	>30	ON			30			
		OFF				31		
		IN	35	40			50	100
valor total de compras	≤1000	ON					1000	
		OFF						1001
		IN	900	500	20	50		
Expected Results			6%	V4	V2	6%	6%	V0

V2			Test Cases					
var	condition		1	-	2	-	3	-
#dias de atraso	≤0	ON	0					
		OFF		1				
		IN			-3	-5	-10	-12
#compras	≥15	ON			15			
		OFF				14		
	≤30	ON					30	
		OFF						31
	IN		20	17				
Expected Results			3%	V5	3%	V3	3%	V0/V1

V3			Test Cases			
var	condition		1	-	-	2
#dias de atraso	≤0	ON	0			
		OFF		1		
		IN			-2	-5
#compras	<15	ON			15	
		OFF				14
		IN	10	5		
Expected Results			1%	V5	V2	1%

V4			Test Cases				
var	condition		1	-	-	-	2
#dias de atraso	=1	ON	1				
		OFF		2			
		OFF			0		
		IN				1	1
#compras	>30	ON				30	
		OFF					31
		IN	35	40	50		
Expected Results			2%	V6	V0/V1	V5	2%

V5			Test Cases				
var	condition		1	-	-	2	-
#dias de atraso	=1	ON	1				
		OFF		2			
		OFF			0		
		IN				1	1
#compras	≤30	ON				30	
		OFF					31
		IN	25	20	15		
Expected Results			-4%	V6	V2	-4%	V0/V1

V6			Test Cases	
var	condition		-	1
#dias de atraso	>1	ON	1	
		OFF		2
		IN		
Expected Results			V4/V5	-10%

addNewProduct (Product p)

Adds a new product to a Post Office if all conditions are met. If the Post Office does not have the Product already and if the number of products in the Post Office is less than the max.

Test Pattern – Category Partition Test

1. Functions

Primary Function:

- Add a Product if the p.name is unique.

2. Input/output Parameters

Inputs:

- product name;
- products.

Outputs:

- return value;
- products.

3. Categories and Choices

Parameter	Category	Choices
product name	unique	name
	repeated	namei
	invalid	Null [error]
products	with elements	1 < size < max
	empty	size = 0
	singleton	Size =1

4. Constraints

It is impossible to have a repeated product when products size is 0 or 1.

5. Test Cases

	Choices		Expected Result	
TC	product name	products	return value	products
1	name	1 < size < max	true	size+1
2	name	size = 1	true	size = 2
3	name	size = 0	true	size = 1
4	namei	{namei, ..., ln}, n < max	false	size
5	namei	{namei}	false	{namei}
6	Product null	size = 1	false	Size = 1

Class Scope Test

PostOffice class

This class represents a post office that is responsible for managing its own products. The post office has a maximum number of products that it can store and needs to have at least 1 product stored. Each product stored in the post office has a unique name, a description, a price, a critical value, and a quantity.

Test Pattern – Non-modal Class Test

1. Class invariant

PostOffice Variables	
Variable	**Type**
maxNumberOfProducts	int
products	List<Product>
getNumberOfProducts ()	int

Domain restrictions:

$2 \leq maxNumberOfProducts \leq 20$ &&
$for\ each\ p\ in\ products, \quad p.price \geq 0$ &&
$0 \leq getNumberOfProducts() \leq maxNumberOfProducts$ &&
$for\ each\ p\ in\ products, \quad 0 \leq p.getCurrentQuantity() \leq getNumberOfProducts()$ &&
$for\ any\ p1, p2\ in\ products, \quad p1.name = p2.name \rightarrow p1 = p2$

2. On points and Off points

Condition	On Point	Off Point
$maxNumberOfProducts \geq 2$	2	1
$maxNumberOfProducts \leq 20$	20	21
$p.price \geq 0$	0	-1
$p.name\ is\ unique$	True	False
$getNumberOfProducts() \geq 0$	0	-1
$getNumberOfProducts() \leq max$	10*	11*
$p.getCurrentQuantity() \geq 0$	0	-1
$p.getCurrentQuantity() \leq n$	14**	15**

* consider max = 10
** consider n = getNumberOfProducts () = 14

3. Domain matrix

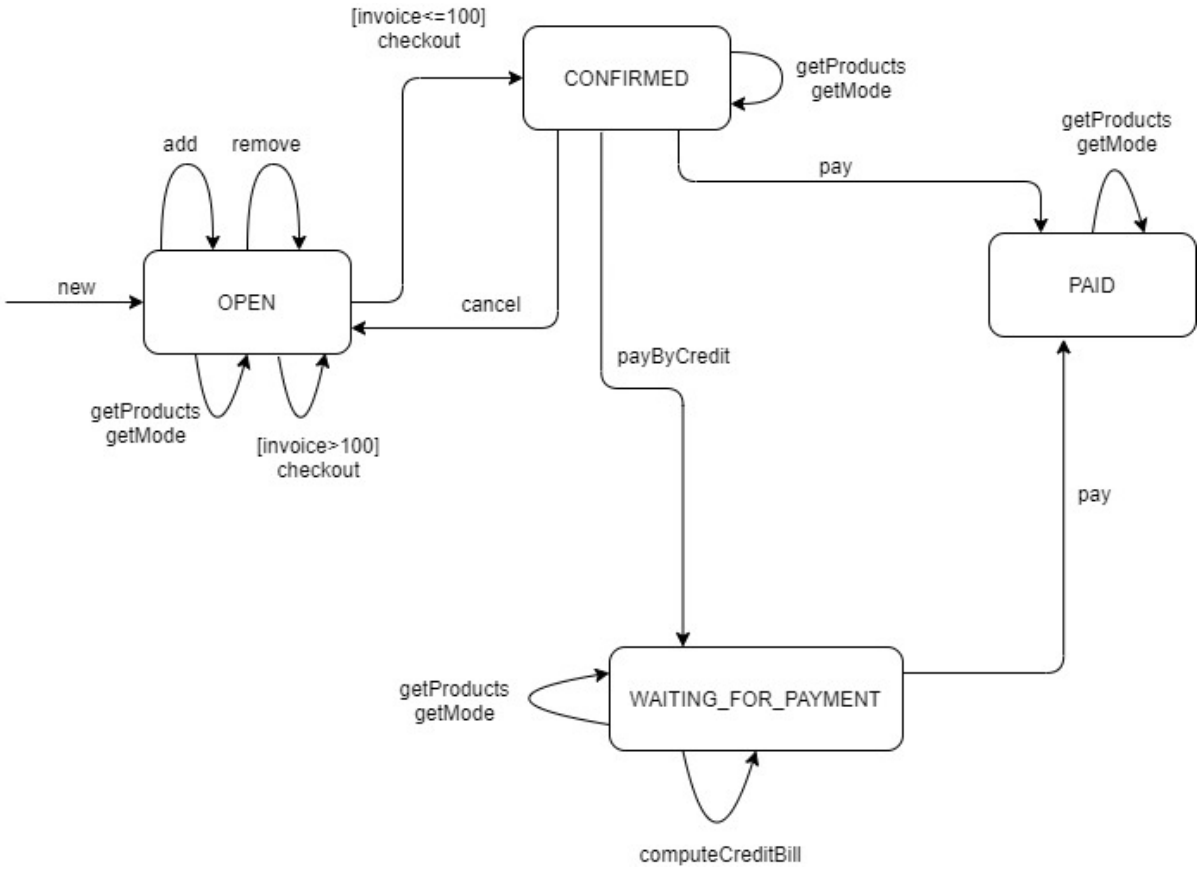
Constraint		Test Cases															
Variable	Condition	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
maxNumberOfProducts	>= 2 ON	2															
	>= 2 OFF		1														
	<= 20 ON			20													
	<= 20 OFF				21												
	IN					3	4	10	10	5	6	8	10	15	16	17	18
getNumberOfProducts()	>= 0 ON					0											
	>= 0 OFF						-1										
	<= max ON							10									
	<= max OFF								11								
	IN	1	2	3	4					7	8	9	10	14	14	15	16
p.price	>= 0 ON									0							
	>= 0 OFF										-1						
	IN	5	10	15	20	25	30	35	40			45	50	55	60	65	70
p.getCurrentQuantity()	>=0 ON											0					
	>=0 OFF												-1				
	<=n ON													14			
	<=n OFF														15		
	IN	1	2	3	4	0	6	8	9	5	6					8	9
p.name	is unique ON															T	
	is unique OFF																F
	IN	T	T	T	T	T	T	T	T	T	T	T	T	T	T		
Expected Result		✓	X	✓	X	✓	X	✓	X	✓	X	✓	X	✓	X	✓	X

Invoice class

This class represents an invoice that resulted from a service provided to a client. Whenever a client wants to buy a product in the post office a new instance of Invoice is created. When an invoice is created, it is OPEN. When the invoice is OPEN, the client can add, and remove products. When the client's checkouts the purchase: if the value of the invoice is less or equal to 100 the invoice is now CONFIRMED. Otherwise the invoice continues OPEN, because it's not allowed to get invoices with a total bigger than 100. When the invoice is CONFIRMED it can now be paid or cancelled. If the invoice is cancelled, then it comes back to OPEN. The invoice can be paid fully or by credit. If its paid fully, the invoice is PAID. Otherwise the invoice is WAITING_FOR_PAYMENT and when paid goes to PAID.

Test Pattern – Modal Class Test

1. State Model

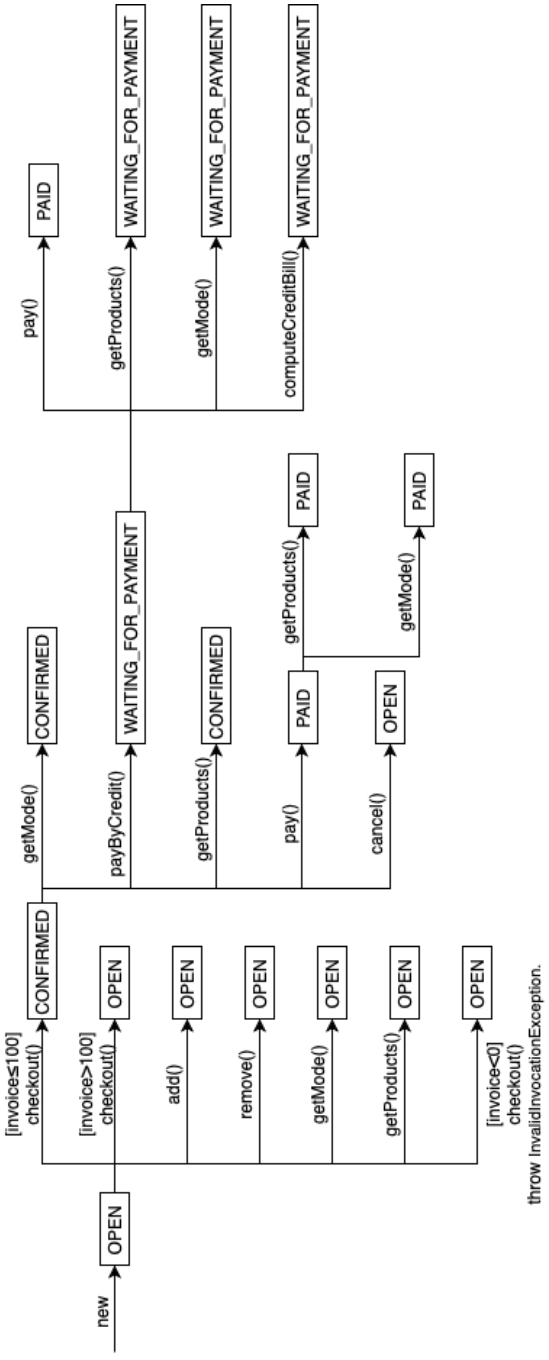


2. Conditional transition variants

State	Transition	Condition	Next State
OPEN	Checkout	Invoice<=100	CONFIRMED
OPEN	Checkout	Invoice>100	OPEN
OPEN	Checkout	Invoice<0 *	OPEN

*InvalidInvocationException is thrown

3. Transition Tree



4. Conformance Test Suite

TC	Level 1	Level 2	Level 3	Level 4	Expected State	Exception
1	new	-	-	-	OPEN	no
2	new	[$invoice \leq 100$] checkout	-	-	CONFIRMED	no
3	new	[$invoice > 100$] checkout	-	-	OPEN	no
4	new	add	-	-	OPEN	no
5	new	remove	-	-	OPEN	no
6	new	getMode	-	-	OPEN	no
7	new	getProducts	-	-	OPEN	no
8	new	[$invoice < 0$] checkout	-	-	OPEN	yes
9	new	[$invoice \leq 100$] checkout	getMode	-	CONFIRMED	no
10	new	[$invoice \leq 100$] checkout	payByCredit	-	WAITING_FOR_PAYMENT	no
11	new	[$invoice \leq 100$] checkout	getProducts	-	CONFIRMED	no
12	new	[$invoice \leq 100$] checkout	pay	-	PAID	no
13	new	[$invoice \leq 100$] checkout	cancel	-	OPEN	no
14	new	[$invoice \leq 100$] checkout	payByCredit	pay	PAID	no
15	new	[$invoice \leq 100$] checkout	payByCredit	getProducts	WAITING_FOR_PAYMENT	no
16	new	[$invoice \leq 100$] checkout	payByCredit	getMode	WAITING_FOR_PAYMENT	no
17	new	[$invoice \leq 100$] checkout	payByCredit	computeCreditBill	WAITING_FOR_PAYMENT	no
18	new	[$invoice \leq 100$] checkout	pay	getProducts	PAID	no
19	new	[$invoice \leq 100$] checkout	pay	getMode	PAID	no

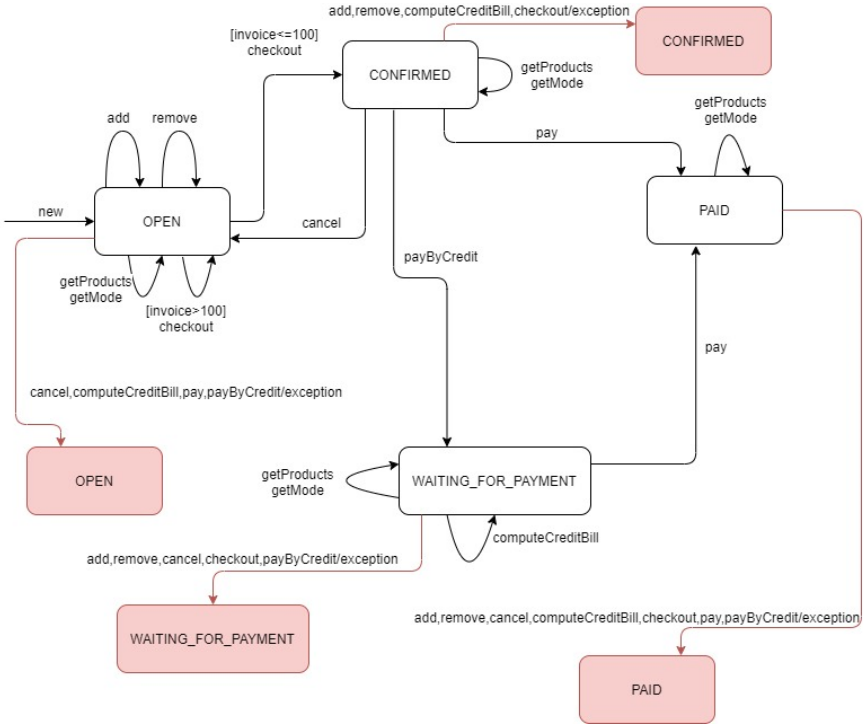
5. Test data for each expanded condition path using Invariant Boundaries

OPEN - checkout	ON	OFF
$invoice \leq 100$	100	101
$invoice > 100$	100	101
$invoice < 0$	0	-1

6. Sneak Path Transition Table

Events	States			
	OPEN	CONFIRMED	WAITING_FOR_PAYMENT	PAID
add	✓	PSP	PSP	PSP
remove	✓	PSP	PSP	PSP
cancel	PSP	✓	PSP	PSP
computeCreditBill	PSP	PSP	✓	PSP
checkout	✓	PSP	PSP	PSP
getProducts	✓	✓	✓	✓
getMode	✓	✓	✓	✓
pay	PSP	✓	✓	PSP
payByCredit	PSP	✓	PSP	PSP

7. Sneak Path Test Suite



TC	Level 1	Level 2	Level 3	Level 4	Expected State	Exception
1	new	cancel	-	-	OPEN	yes
2	new	computeCreditBill	-	-	OPEN	yes
3	new	Pay	-	-	OPEN	yes
4	new	payByCredit	-	-	OPEN	yes
5	new	[*invoice* ≤ 100] checkout	add	-	CONFIRMED	yes
6	new	[*invoice* ≤ 100] checkout	remove	-	CONFIRMED	yes
7	new	[*invoice* ≤ 100] checkout	computeCredit Bill	-	CONFIRMED	yes
8	new	[*invoice* ≤ 100] checkout	checkout	-	CONFIRMED	yes
9	new	[*invoice* ≤ 100] checkout	payByCredit	add	WAITING_FOR_PAYMENT	yes
10	new	[*invoice* ≤ 100] checkout	payByCredit	remove	WAITING_FOR_PAYMENT	yes
11	new	[*invoice* ≤ 100] checkout	payByCredit	cancel	WAITING_FOR_PAYMENT	yes
12	new	[*invoice* ≤ 100] checkout	payByCredit	checkout	WAITING_FOR_PAYMENT	yes
13	new	[*invoice* ≤ 100] checkout	payByCredit	payByCredit	WAITING_FOR_PAYMENT	yes
14	new	[*invoice* ≤ 100] checkout	pay	add	PAID	yes
15	new	[*invoice* ≤ 100] checkout	pay	remove	PAID	yes
16	new	[*invoice* ≤ 100] checkout	pay	cancel	PAID	yes
17	new	[*invoice* ≤ 100] checkout	pay	computeCredit Bill	PAID	yes
18	new	[*invoice* ≤ 100] checkout	pay	checkout	PAID	yes
19	new	[*invoice* ≤ 100] checkout	pay	pay	PAID	yes
20	new	[*invoice* ≤ 100] checkout	pay	payByCredit	PAID	yes

TestNG Test Cases

PostOffice class

```
import org.testng.annotations.*;
import static org.testng.Assert.*;

public class PostOfficeTest {
    private List<Product> products_unique0 = new List<Product>();
    private List<Product> products_unique1 = new List<Product>();
    private List<Product> products_unique2 = new List<Product>();
    private List<Product> products_unique3 = new List<Product>();
    private List<Product> products_unique4 = new List<Product>();
    private List<Product> products_unique7 = new List<Product>();
    private List<Product> products_unique7b = new List<Product>();
    private Product product1 =  new Product("p1","produto",5,5);
    private Product product2 =  new Product("p2","produto",10,10);
    private Product product3 =  new Product("p3","produto",15,15);
    private Product product4 =  new Product("p4","produto",15,15);
    private Product product7 =  new Product("p7","produto",40,40);

    @BeforeMethod
    private void setUp(){
        product1.store(1);
        products_unique1.add(product1);

        product2.store(2);
        products_unique2.add(product2);

        product3.stor(3);
        products_unique3.add(product3);

        product4.store(4);
        products_unique4.add(product4);

        product7.store(8);
        products_unique7.add(product7);
        products_unique7.add(product2);

        products_unique7b.add(product7);
        products_unique7b.add(product1);

    }

    //TC1 (2,1,5,1,T)
    @Test()
    public void test1aProductsMaxEquals2(){
        //Arrange
        PostOffice po = new PostOffice(2,products_unique1);

        //Assert
        assertEquals(po.getProducts().size(), 1);
        assertEquals(po.getMaxNumberOfProducts(), 2);
        assertEquals(po.getNumberOfProducts(), 1);
    }
```

```
//TC1 (2,1,5,1,T)
@Test()
public void test1bProductsMaxEquals2(){
    //Arrange
    PostOffice po = new PostOffice(10,products_unique1);
    po.setMaxNumberOfProducts(2);


    //Assert
    assertEquals(po.getProducts().size(), 1);
    assertEquals(po.getMaxNumberOfProducts(), 2);
    assertEquals(po.getNumberOfProducts(), 1);
}

//TC2 (1,2,10,2,T)
@Test()
public void test2aProductsMaxEquals1(){
 //Arrange
 PostOffice po = new PostOffice(10,products_unique2);
 po.setMaxNumberOfProducts(1);


 //Assert
 assertEquals(po.getProducts().size(), 1);
 assertEquals(po.getMaxNumberOfProducts(), 2);
 assertEquals(po.getNumberOfProducts(), 1);
}
//TC2 (1,2,10,2,T)
@Test(expectedExceptions = Invalidoperation.class)
public void test2abProductsMaxEquals1(){
    PostOffice po = new PostOffice(1,products_unique2);
}

//TC3 (20,3,15,3,T)
@Test()
public void test3aProductsMaxEquals20(){
    //Arrange
    PostOffice po = new PostOffice(20,products_unique3);

    //Assert
    assertEquals(po.getProducts().size(), 1);
    assertEquals(po.getMaxNumberOfProducts(), 20);
    assertEquals(po.getNumberOfProducts(), 3);
}

//TC3 (20,3,15,3,T)
@Test()
public void test3bProductsMaxEquals20(){
    //Arrange
    PostOffice po = new PostOffice(10,products_unique3);
    po.setMaxNumberOfProducts(20);

    //Assert
    assertEquals(po.getProducts().size(), 1);
    assertEquals(po.getMaxNumberOfProducts(), 20);
    assertEquals(po.getNumberOfProducts(), 3);
}
```

```java
//TC4 (21,4,20,4,T)
@Test(expectedExceptions = Invalidoperation.class)
public void test4aProductsMaxEquals21(){
    PostOffice po = new PostOffice(21,products_unique4);
}

//TC4 (21,4,20,4,T)
@Test()
public void test4bProductsMaxEquals21(){
    //Arrange
    PostOffice po = new PostOffice(10,products_unique4);
    po.setMaxNumberOfProducts(21);

    //Assert
    assertEquals(po.getProducts().size(), 1);
    assertEquals(po.getMaxNumberOfProducts(), 10);
    assertEquals(po.getNumberOfProducts(), 4);
}

//TC5 (3,0,25,0,T)
@Test()
public void test5aNumberOfProducts0(){
    //Arrange
    PostOffice po = new PostOffice(3,products_unique0);

    //Assert
    assertEquals(po.getProducts().size(), 0);
    assertEquals(po.getMaxNumberOfProducts(), 3);
    assertEquals(po.getNumberOfProducts(), 0);
}

//TC5 (3,0,25,0,T)
@Test()
public void test5bNumberOfProducts0(){
    //Arrange
    PostOffice po = new PostOffice(3,products_unique1);
    po.update("p1",0);

    //Assert
    assertEquals(po.getProducts().size(), 1);
    assertEquals(po.getMaxNumberOfProducts(), 3);
    assertEquals(po.getNumberOfProducts(), 0);
}

//TC5 (3,0,25,0,T)
@Test()
public void test5cNumberOfProducts0(){
    //Arrange
    PostOffice po = new PostOffice(3,products_unique1);
    po.update("p1",0);
    po.removeProduct("p1");

    //Assert
    assertEquals(po.getProducts().size(), 0);
    assertEquals(po.getMaxNumberOfProducts(), 3);
    assertEquals(po.getNumberOfProducts(), 0);
}
```

```java
//TC7 (10,10,35,8,T)
@Test()
public void test7aNumberOfProductsEqualsMax(){
    //Arrange
    PostOffice po = new PostOffice(10,products_unique7);

    //Assert
    assertEquals(po.getProducts().size(), 2);
    assertEquals(po.getMaxNumberOfProducts(), 10);
    assertEquals(po.getNumberOfProducts(), 10);
}

//TC7 (10,10,35,8,T)
@Test()
public void test7bNumberOfProductsEqualsMax(){
    //Arrange
    PostOffice po = new PostOffice(10,products_unique7b);
    po.update("p1",2);

    //Assert
    assertEquals(po.getProducts().size(), 2);
    assertEquals(po.getMaxNumberOfProducts(), 10);
    assertEquals(po.getNumberOfProducts(), 10);
}

//TC7 (10,10,35,8,T)
@Test()
public void test7cNumberOfProductsEqualsMax(){
    //Arrange
    PostOffice po = new PostOffice(15,products_unique7);
    setMaxNumberOfProducts(10);

    //Assert
    assertEquals(po.getProducts().size(), 2);
    assertEquals(po.getMaxNumberOfProducts(), 10);
    assertEquals(po.getNumberOfProducts(), 10);
}

//TC7 (10,10,35,8,T)
@Test()
public void test7dNumberOfProductsEqualsMax(){
    //Arrange
    PostOffice po = new PostOffice(15,products_unique7b);
    setMaxNumberOfProducts(10);
    po.update("p1",2);

    //Assert
    assertEquals(po.getProducts().size(), 2);
    assertEquals(po.getMaxNumberOfProducts(), 10);
    assertEquals(po.getNumberOfProducts(), 10);
}
```

```
    //TC7 (1O,1O,35,8,T)
    @Test()
    public void test7eNumberOfProductsEqualsMax(){
        //Arrange
        PostOffice po = new PostOffice(15,products_unique7b);
        po.update("p1",2);
        setMaxNumberOfProducts(10);

        //Assert
        assertEquals(po.getProducts().size(), 2);
        assertEquals(po.getMaxNumberOfProducts(), 10);
        assertEquals(po.getNumberOfProducts(), 10);
    }
}
```