Software Testing and Validation

André dos Santos Martins Ribeiro - **104083**
Lourenço Mourão - **102122**
Serhii Ivanenko - **102058**

Date: May 2022

Motivation

Design and implement a test suite given the project specification. The specification models a portal of restaurants that allows clients to buy meals from existing restaurants.

Objective

Design and Develop test cases applying the most appropriate test pattern for each case. The test cases to design are the following:

- **Restaurant** and **ShoppingTray** at the class scope
- **computeDiscount()** of class Rest and **add()** of class Restaurant at method scope
- Implement eight test cases concerning the test suite that tests the class **Restaurant**

Table of Contents

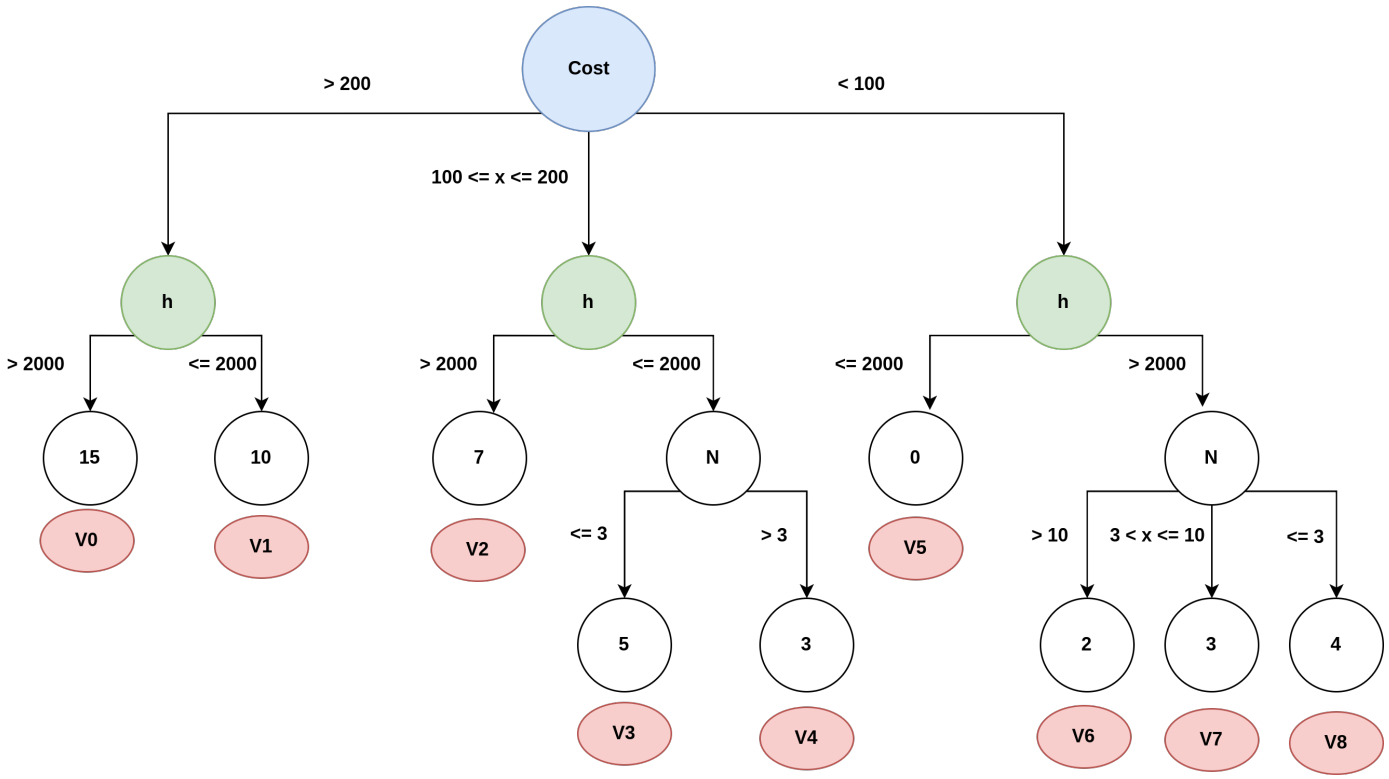
Method Scope Test Suite

Using Method Scope Test Design Pattern we have designed test suites for the methods *computeDiscount()* and *add()* structured in such a way as that it describes which test pattern was chosen, the results of the different stages of the test pattern and a final description of the resulting test cases.

computeDiscount()

Used Test Pattern: Combinational Functional Test

We started with drawing a decision tree that represents all the possible variants of this method. This allows us to identify the different variants that will be used for the next step.



After which, we created a decision table representing the inputs and expected outputs of each variant.

	Input			Output
V0	cost > 200	history > 2000	number - any	15
V1	cost > 200	history <= 2000	number - any	10
V2	100 <= cost <= 200	history > 2000	number - any	7
V3	100 <= cost <= 200	history <= 2000	number <= 3	5
V4	100 <= cost <= 200	history <= 2000	number > 3	3
V5	cost < 100	history <= 2000	number - any	0
V6	cost < 100	history > 2000	number > 10	2
V7	cost < 100	history > 2000	3 < number <= 1	3
V8	cost < 100	history > 2000	number <= 3	4

And defined the domain matrix for each variant representing each boundary points and expected results and if they were accepted or not (in green).

V0			1 (-)	2 (1)	3 (-)	4 (2)
		ON	200			
		OFF		201		
cost	> 200	IN			350	420
		ON			2000	
		OFF				2001
history	> 2000	IN	2005	2500		
number	any	IN	3	4	5	6
Expected result			V2	A, 15	V1	A, 15

V1			1 (-)	2 (3)	3 (4)	4 (-)
		ON	200			
		OFF		201		
cost	> 200	IN			320	480
		ON			2000	
		OFF				2001
history	<= 2000	IN	10	1200		
number	any	IN	5	6	7	8
Expected result			V4	A, 10	A, 10	V0

V2			1 (5)	2 (-)	3 (6)	4 (-)	5 (-)	6 (7)
		ON	200					
cost	<= 200	OFF		201				
		ON			100			
cost	>= 100	OFF				99		
cost		IN					150	180
		ON					2000	
		OFF						2001
history	> 2000	IN	4308	2400	2900	3200		
number	any	IN	1	2	3	4	5	6
Expected result			A, 7	V0	A, 7	V7	V4	A, 7

V3			1 (8)	2 (-)	3 (9)	4 (-)	5 (10)	6 (-)	7 (11)	8 (-)
		ON	200							
cost	<= 200	OFF		201						
		ON			100					
cost	>= 100	OFF				99				
cost		IN					120	130	140	150
		ON					2000			
		OFF						2001		
history	<= 2000	IN	1337	1400	1500	1600			1700	1800
		ON							3	
		OFF								4
number	<= 3	IN	1	1	2	2	1	2		
Expected result			A, 5	V1	A, 5	V5	A, 5	V2	A, 5	V4

V4			1 (12)	2 (-)	3 (13)	4 (-)	5 (14)	6 (-)	7 (-)	8 (15)
		ON	200							
cost	<= 200	OFF		201						
		ON			100					
cost	>= 100	OFF				99				
cost		IN					110	115	155	145
		ON					2000			
		OFF						2001		
history	<= 2000	IN	800	900	1000	1100			1200	1300
		ON							3	
		OFF								4
number	> 3	IN	5	7	8	9	10	11		
Expected result			A, 3	V1	A, 3	V5	A, 3	V2	V3	A, 3

V5			1 (-)	2 (16)	3 (17)	4 (-)
		ON	100			
		OFF		99		
cost	< 100	IN			55	75
		ON			2000	
		OFF				2001
history	<= 2000	IN	1400	880		
number	any	IN	1	2	3	4
Expected result			V3	A, 0	A, 0	V7

V6			1 (-)	2 (18)	3 (-)	4 (19)	5 (-)	6 (20)
		ON	100					
		OFF		99				
cost	< 100	IN			15	30	50	60
		ON			2000			
		OFF				2001		
history	> 2000	IN	2400	2700			3000	3400
		ON					10	
		OFF						11
number	> 10	IN	12	13	14	15		
Expected result			V2	A, 2	V5	A, 2	V7	A, 2

V7			1 (-)	2 (21)	3 (-)	4 (22)	5 (23)	6 (-)	7 (-)	8 (24)
		ON	100							
		OFF		99						
cost	< 100	IN			10	20	30	40	50	60
		ON			2000					
		OFF				2001				
history	> 2000	IN	2250	2360			7800	3220	4100	4200
		ON					10			
number	<= 10	OFF						11		
		ON							3	
number	> 3	OFF								4
number		IN	5	6	7	8				
Expected result			V2	A, 3	V5	A, 3	A, 3	V6	V8	A, 3

V8			1 (-)	2 (25)	3 (-)	4 (26)	5 (27)	6 (-)
		ON	100					
		OFF		99				
cost	< 100	IN			5	15	25	45
		ON			2000			
		OFF				2001		
history	> 2000	IN	3000	3400			4000	5500
		ON					3	
		OFF						4
number	<= 3	IN	1	2	1	2		
Expected result			V2	A, 4	V5	A, 4	A, 4	V7

We have a total of **27 accepted tests** that don't overlap with other variants.

add()

Used Test Pattern: Category Partition Test

We started by creating a list of the functions of that method:

- Add Dish

- Check if Restaurant is full
- Check if Dish is free and we exceed amount of free dishes
- Return if dish was added
- Update price if the Dish already exists
- Check if new DIsh is free and don't add it

Then we identified the inputs and outputs.

Inputs	Outputs
Dish	Returned boolean
Price	Dishes
Dishes	Amount of free dishes
Amount of free dishes	Price
Restaurant is vegetarian?	

And the categories in which each of the inputs can be apart of.

	Category	Choices
Dish	Dish in Restaurant	d1
	Dish not in Restaurant	d2
Price	Free	0
	Not free	price in [1, 15]
	Invalid	> 15
Dishes	m-elems	ds1; m in [6,17]; m=11
	special cases	ds2; full (size=17)
		ds3; min (size=6)
Amt of free dishes	n dishes	n < size/4
	special cases	n = size/4
		n = 0
	Invalid	n > size/4
Rest is veg?	Vegetarian	VERDADEIRO
	Not vegetarian	FALSO

We have identified one logical constraint where:

- Dish is in Restaurant, Restaurant is vegetarian, and Dish is not vegetarian.

And designed the tests cases for each cross-join product of all the choices.

	Input					Output			
TC	Dish	Price	Dishes	Amt of free dishe	Rest is veg?	Return value	Dishes	Amt of free dishes	Price
1	d2=(name:"bacalhau", price:7,veg=false)	7	ds1 (d2 not in ds1)	1	FALSO	VERDADEIRO	ds1 ++ d2	1	7
2	d2=(name:"veg soup", price:5,veg=true)	5	ds1 (d2 not in ds1)	2	VERDADEIRO	VERDADEIRO	ds1 ++ d2	2	5
3	d2=(name:"bacalhau", price:7,veg=false)	7	ds1 (d2 not in ds1)	1	VERDADEIRO	InvalidInvocationException	ds1	1	NO PRICE
4	d2=(name:"veg soup", price:5,veg=true)	5	ds1 (d2 not in ds1)	1	FALSO	VERDADEIRO	ds1 ++ d2	1	5
5	d1=(name:"bacalhau", price:7,veg=false)	10	ds1 (d1 in ds1)	0	FALSO	FALSO	ds1	0	10
6	d1=(name:"veg soup", price:5,veg=true)	7	ds1 (d1 in ds1)	1	VERDADEIRO	FALSO	ds1	1	7
7	d1=(name:"veg soup", price:5,veg=true)	7	ds1 (d1 in ds1)	0	FALSO	FALSO	ds1	0	7
8	d2=(name:"bacalhau", price:7,veg=false)	0	ds1 (d2 not in ds1)	2	FALSO	InvalidInvocationException	ds1	2	NO PRICE
9	d1=(name:"veg soup", price:5,veg=true)	0	ds1 (d1 in ds1)	1	FALSO	FALSO	ds1	2	0
10	d1=(name:"bacalhau", price:10,veg=false)	17	ds1 (d1 in ds1)	2	FALSO	InvalidInvocationException	ds1	2	10
11	d2=(name:"veg soup", price:6,veg=true)	19	ds1 (d2 not in ds1)	1	VERDADEIRO	InvalidInvocationException	ds1	1	NO PRICE
12	d2=(name:"bacalhau", price:8,veg=false)	8	ds2 (d2 not in ds2)	1	FALSO	InvalidInvocationException	ds2	1	NO PRICE
13	d1=(name:"bacalhau", price:8,veg=false)	11	ds2 (d1 in ds2)	2	FALSO	FALSO	ds2	2	11
14	d2=(name:"bacalhau", price:11,veg=false)	11	ds3 (d2 not in ds3)	0	FALSO	VERDADEIRO	ds3 ++ d2	0	11
15	d1=(name:"bacalhau", price:11,veg=false)	12	ds3 (d1 in ds3)	1	FALSO	FALSO	ds3	1	12
16	d1=(name:"veg soup", price:5,veg=true)	0	ds3 (d1 in ds3)	1	VERDADEIRO	InvalidInvocationException	ds3	1	5
17	d1=(name:"veg soup", price:6,veg=true)	0	ds2 (d1 in ds2)	4	VERDADEIRO	InvalidInvocationException	ds2	4	6
18	d1=(name:"bacalhau", price:9,veg=false)	0	ds1 (d1 in ds1, size=8)	2	FALSO	InvalidInvocationException	ds1	2	9
19	d1=(name:"bacalhau", price:12,veg=false)	0	ds1 (d1 in ds1)	0	FALSO	FALSO	ds1	1	0
20	d1=(name:"veg soup", price:6,veg=true)	0	ds2 (d1 in ds2)	3	VERDADEIRO	FALSO	ds2	4	0

We have a total of **20 tests** from the cross-join of all the choices.

Class Scope Test Suite

Using Class Scope Test Design Pattern we have designed test suites for the classes *Restaurant* and *ShoppingTray* structured in such a way as that it describes which test pattern was chosen, the results of the different stages of the test pattern and a final description of the resulting test cases.

Restaurant

Used Test Pattern: Non-modal Class Test

We identified these conditions:

1. Vegetarian Restaurant has only vegetarian Dishes - ForEach d in VegRestaurant: d.isVeg == true
2. num of Dishes in [6,17]
3. Names of the dishes need to be unique - ForEach d1 and d2 in Restaurant: d1.name = d2.name -> d1=d2
4. Amount of free Dishes < 1/4 of number of total Dishes

And created a Domain Matrix for the class Restaurant according to the conditions and domain logic.

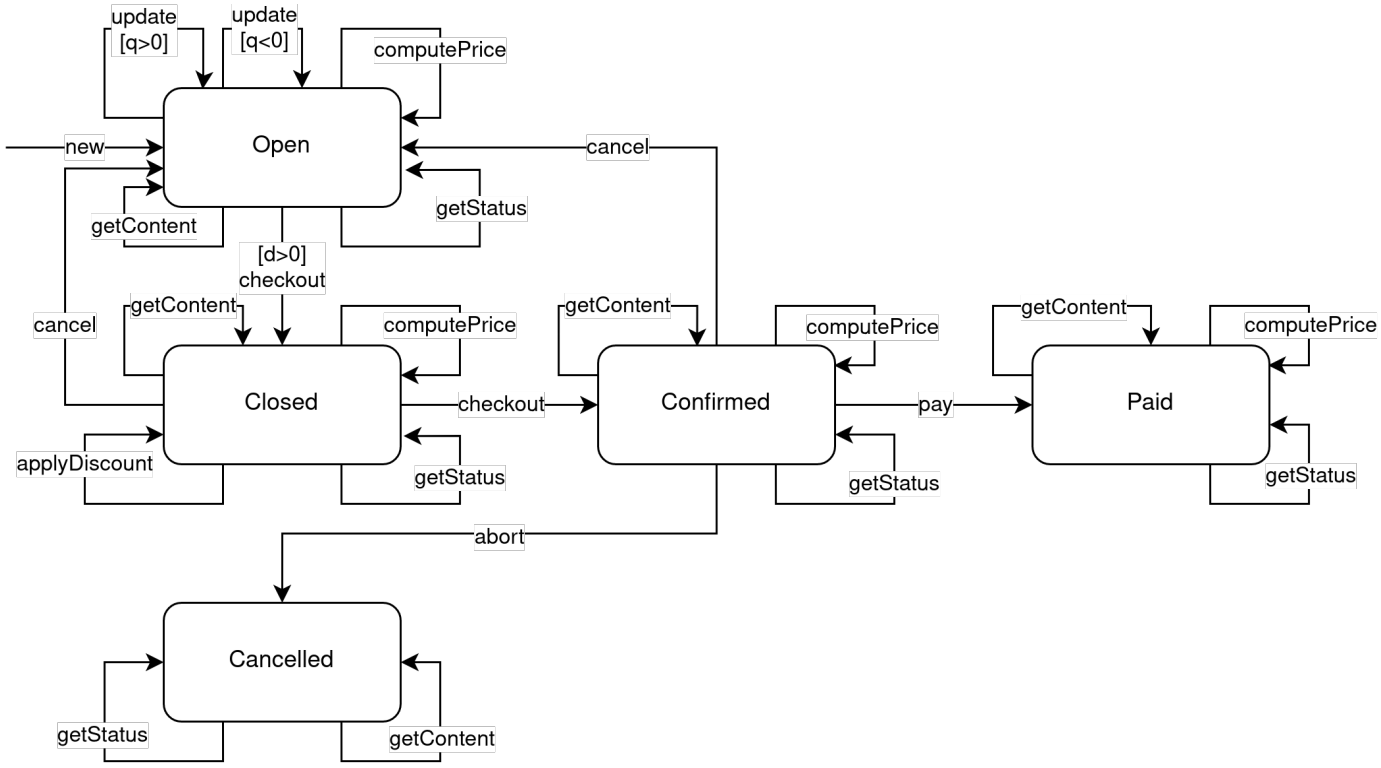
			1	2	3	4	5	6	7	8	9	10
		ON	6									
	>= 6	OFF		5								
		ON			17							
	<= 17	OFF				18						
#dishes	Typical	IN					12	8	9	10	11	12
		ON					3					
	<= 3	OFF						4				
#free	Typical	IN	0	1	2	0			1	2	0	1
		ON							T			
	cond1	OFF								F		
restaurant	Typical	IN	T	T	T	T	T	T			T	T
		ON									T	
	cond2	OFF										F
name	Typical	IN	T	T	T	T	T	T	T	T		
Expected result			A	R	A	R	A	R	A	R	A	R

We have a total of **10 tests** in which 5 are accepted (correct behaviour) and 5 are rejected (and an exception is thrown).

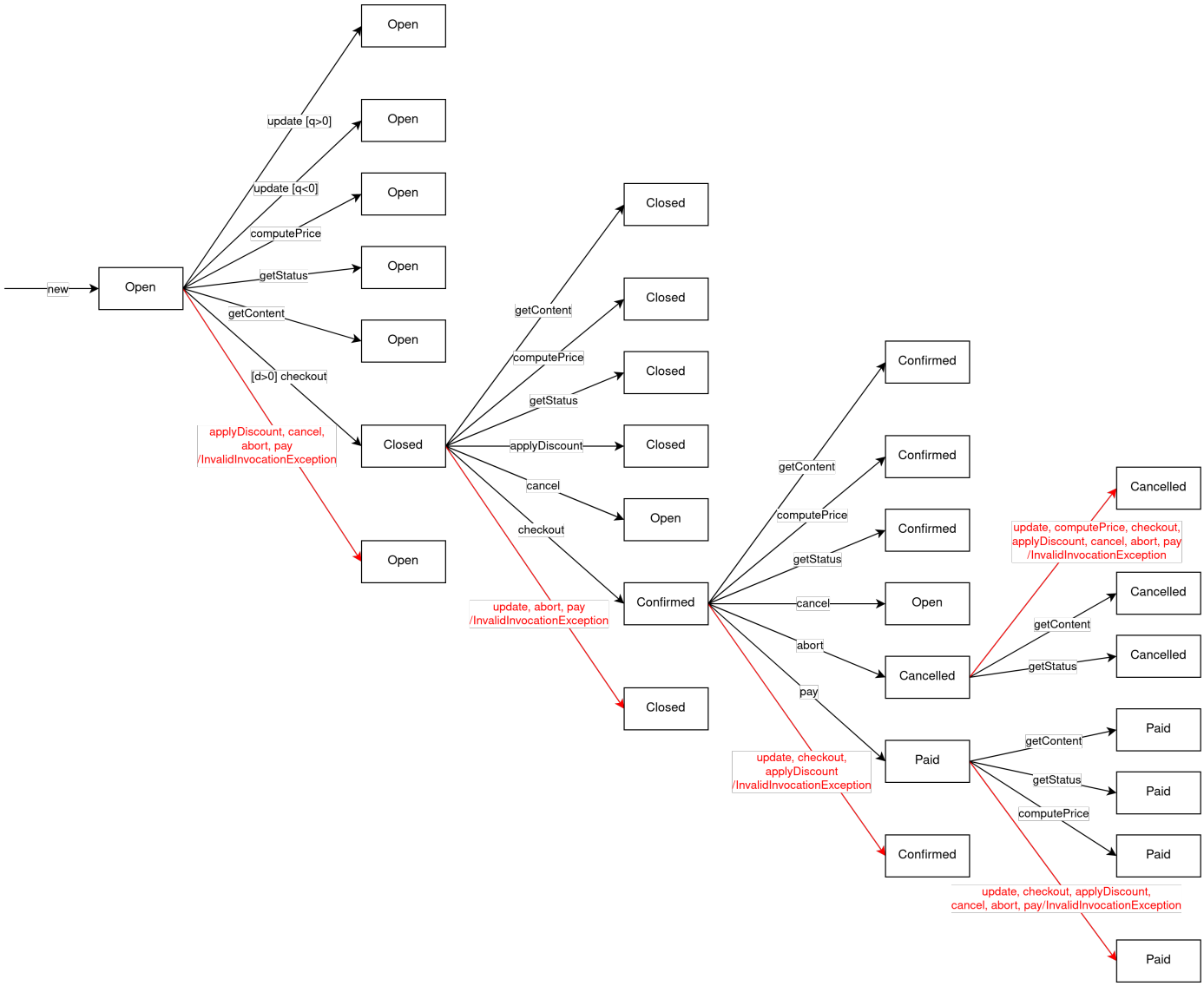
ShoppingTray

Used Test Pattern: Modal Class Test using a Finite State Machine Based Test Approach

We have developed a state machine diagram for the class ShoppingTray in which it represents the state of the class and its possible transitions.



And then created a transition tree to help identify transition paths and in red the possible sneak paths that we want to get caught in a expection.



Then we identified the conditional transition variants:

Step 2	q - quantity of dishes to be added, d - num of dishes in the ShoppingTray		
State	Message	Condition	Next state
Open	update	Post: q > 0	Open
Open	update	Post: q < 0	Open
Open	checkout	Pre: d > 0	Closed

And created a Conformance Test Suite for the correct behavior paths.

Step 4						
Conformance Test Suite						
	Test run / Event path					
Run	Level 1	Level 2	Level 3	Level 4	Level 5	Expected terminal state
1	new					Open
2	new	update [q>0]				Open
3	new	update [q<0]				Open
4	new	computePrice				Open
5	new	getStatus				Open
6	new	getContent				Open
7	new	[d>0] checkout				Closed
8	new	[d>0] checkout	getContent			Closed
9	new	[d>0] checkout	computePrice			Closed
10	new	[d>0] checkout	getStatus			Closed
11	new	[d>0] checkout	applyDiscount			Closed
12	new	[d>0] checkout	cancel			Open
13	new	[d>0] checkout	checkout			Confirmed
14	new	[d>0] checkout	checkout	getContent		Confirmed
15	new	[d>0] checkout	checkout	computePrice		Confirmed
16	new	[d>0] checkout	checkout	getStatus		Confirmed
17	new	[d>0] checkout	checkout	cancel		Open
18	new	[d>0] checkout	checkout	abort		Cancelled
19	new	[d>0] checkout	checkout	abort	getContent	Cancelled
20	new	[d>0] checkout	checkout	abort	getStatus	Cancelled
21	new	[d>0] checkout	checkout	pay		Paid
22	new	[d>0] checkout	checkout	pay	getContent	Paid
23	new	[d>0] checkout	checkout	pay	getStatus	Paid
24	new	[d>0] checkout	checkout	pay	computePrice	Paid

Then we created test data for each path using invariant boundaries.

Step 5		
update in Open		
Condition	ON	OFF
q > 0	0	1
q < 0	0	-1
checkout in Open		
Condition	ON	OFF
d > 0	0	1

And identified the Possible Sneak Paths.

Step 7					
Events\States	Open	Closed	Confirmed	Paid	Cancelled
update	?	PSP	PSP	PSP	PSP
computePrice	Y	Y	Y	Y	PSP
getStatus	Y	Y	Y	Y	Y
getContent	Y	Y	Y	Y	Y
checkout	?	Y	PSP	PSP	PSP
applyDiscount	PSP	Y	PSP	PSP	PSP
cancel	PSP	Y	Y	PSP	PSP
abort	PSP	PSP	Y	PSP	PSP
pay	PSP	PSP	Y	PSP	PSP

And then created a Conformance Test Suite for the PSP.

	Test run / Event path						
Run	Level 1	Level 2	Level 3	Level 4	Level 5	Expected terminal state	Exception
25	new	applyDiscount				Open	Yes
26	new	cancel				Open	Yes
27	new	abort				Open	Yes
28	new	pay				Open	Yes
29	new	[d>0] checkout	update			Closed	Yes
30	new	[d>0] checkout	abort			Closed	Yes
31	new	[d>0] checkout	pay			Closed	Yes
32	new	[d>0] checkout	checkout	update		Confirmed	Yes
33	new	[d>0] checkout	checkout	checkout		Confirmed	Yes
34	new	[d>0] checkout	checkout	applyDiscount		Confirmed	Yes
35	new	[d>0] checkout	checkout	abort	update	Cancelled	Yes
36	new	[d>0] checkout	checkout	abort	computePrice	Cancelled	Yes
37	new	[d>0] checkout	checkout	abort	checkout	Cancelled	Yes
38	new	[d>0] checkout	checkout	abort	applyDiscount	Cancelled	Yes
39	new	[d>0] checkout	checkout	abort	cancel	Cancelled	Yes
40	new	[d>0] checkout	checkout	abort	abort	Cancelled	Yes
41	new	[d>0] checkout	checkout	abort	pay	Cancelled	Yes
42	new	[d>0] checkout	checkout	pay	update	Paid	Yes
43	new	[d>0] checkout	checkout	pay	checkout	Paid	Yes
44	new	[d>0] checkout	checkout	pay	applyDiscount	Paid	Yes
45	new	[d>0] checkout	checkout	pay	cancel	Paid	Yes
46	new	[d>0] checkout	checkout	pay	abort	Paid	Yes
47	new	[d>0] checkout	checkout	pay	pay	Paid	Yes

We have a total of **47 tests** including the possible sneak path tests.

Implementation of Restaurant Class Test Suite

We have chosen to implement 8 test cases that represent the first 8 cases that we described above on the design of the Restaurant class.

```
package restaurant;

import org.testng.annotations.Test;
import org.testng.annotations.DataProvider;

import java.util.Arrays;
import java.util.List;

import static org.testng.Assert.*;

@Test
public class TestRestaurant {

    @DataProvider
    private Object[][] computeValidDataForRestaurant() {
        Dish dish1 = new Dish("abc", "ok", false, 7);
        Dish dish2 = new Dish("bcde", "ko", false, 9);
        Dish dish3 = new Dish("cdefg", "fds", true, 10);
        Dish dish4 = new Dish("ab", "aaaaa", false, 11);
        Dish dish5 = new Dish("jkrig", "m", false, 14);
        Dish dish6 = new Dish("ytong", "odfdgk", false, 4);
        Dish dish7 = new Dish("hokr", "oklkds", true, 6);
        Dish dish8 = new Dish("reren", "okooa", true, 2);
        Dish dish9 = new Dish("salo", "kokos", false, 7);
        Dish dish10 = new Dish("ppfd", "gdfcxx", true, 8);
        Dish dish11 = new Dish("vnmde", "das-fd", false, 0);
        Dish dish12 = new Dish("iddo", "fdscv", false, 13);
        Dish dish13 = new Dish("kllmm", "dfg", false, 11);
        Dish dish14 = new Dish("odas", "dsf ds", true, 8);
        Dish dish15 = new Dish("mmn", "fhgvhs", false, 7);
        Dish dish16 = new Dish("hgfo", "sawdqs", false, 4);
        Dish dish17 = new Dish("hog", "fdsf", true, 0);
        Dish dish18 = new Dish("ncs", "mmnvd", false, 7);
        Dish dish19 = new Dish("test", "ekekks", true, 6);
        Dish dish20 = new Dish("vwwdg", "ekek21", true, 6);
        Dish dish21 = new Dish("yred", "ekek14", true, 4);
        Dish dish22 = new Dish("olfh", "ekek745", true, 3);
        Dish dish23 = new Dish("mnbd", "ekes3", true, 6);
        Dish dish24 = new Dish("qwsdq", "eke24ks", true, 5);
        Dish dish25 = new Dish("zzxc", "eke3kks", true, 9);
        Dish dish26 = new Dish("zztop", "e79kekks", true, 8);
        Dish dish27 = new Dish("dmode", "eke32kks", true, 11);
        Dish dish28 = new Dish("oelzy", "ek0e2kks", true, 13);
        Dish dish29 = new Dish("vwvw", "ek3e9kks", true, 14);
        Dish dish30 = new Dish("acdc", "ek3ek5ks", true, 0);
        Dish dish31 = new Dish("metal", "ek3ek5ks", true, 0);
        return new Object[][] {
                {"Claude Monet", "Str. Shukhevycha", Arrays.asList(dish1, dish2, dish9, dish10,
dish14, dish19), false},
```

```
                {"VegChum", "Av. de Berna", Arrays.asList(dish3, dish7, dish8, dish10, dish14,
dish17, dish20, dish21, dish22, dish23, dish24, dish25, dish26, dish27, dish28, dish29,
dish30), true},
                {"GoVeg", "Iifds", Arrays.asList(dish3, dish7, dish8, dish10, dish14, dish17,
dish22, dish24, dish31, dish28, dish30, dish25), true},
                {"Lisbon", "Tehran", Arrays.asList(dish4, dish5, dish6, dish11, dish12, dish13,
dish15, dish16, dish18), false},
        };
    }

    @Test(dataProvider = "computeValidDataForRestaurant")
    public void testRestaurantAccepted(String name, String address, List<Dish> dishes, boolean
isVegetarian) {
        // Arrange
        Restaurant rest = new Restaurant(name, address, dishes);
        // Act
        rest.setVegetarian(isVegetarian);
        // Assert
        assertEquals(rest.getName(), name);
        assertEquals(rest.getAddress(), address);
        assertEquals(rest.getDishes(), dishes);
        assertEquals(rest.isVegetarian(), isVegetarian);
    }

    @Test
    public void testRestaurantSetVegetarian() {
        // Arrange
        String name = "Bong";
        String address = "Unter den Linden";
        Dish dish1 = new Dish("abc", "ok", true, 7);
        Dish dish2 = new Dish("bcde", "ko", true, 9);
        Dish dish3 = new Dish("cdefg", "fds", true, 10);
        Dish dish4 = new Dish("ab", "aaaaa", false, 11);
        Dish dish5 = new Dish("jkrig", "m", true, 14);
        Dish dish6 = new Dish("ytong", "odfdgk", true, 0);
        Dish dish7 = new Dish("hokr", "oklkds", true, 6);
        Dish dish8 = new Dish("reren", "okooa", true, 0);
        Dish dish9 = new Dish("salo", "kokos", false, 7);
        Dish dish10 = new Dish("ppfd", "gdfcxx", true, 8);
        List<Dish> dishes = Arrays.asList(dish1, dish2, dish3, dish4, dish5, dish6, dish7,
dish8, dish9, dish10);
        // Act
        Restaurant rest = new Restaurant(name, address, dishes);
        assertThrows(InvalidInvocationException.class, () -> rest.setVegetarian(true));
        // Assert
        assertEquals(rest.getName(), name);
        assertFalse(rest.isVegetarian());
        assertEquals(rest.getDishes(), dishes);
        assertEquals(rest.getAddress(), address);
```

```
    }

    @Test
    public void testRestaurantAdd18Dishes() {
        // Arrange
        String name = "Aagog";
        String address = "Rua Augusta";
        Dish dish1 = new Dish("abc", "ok", true, 7);
        Dish dish2 = new Dish("bcde", "ko", true, 9);
        Dish dish3 = new Dish("cdefg", "fds", true, 10);
        Dish dish4 = new Dish("ab", "aaaaa", false, 11);
        Dish dish5 = new Dish("jkrig", "m", true, 14);
        Dish dish6 = new Dish("ytong", "odfdgk", true, 10);
        Dish dish7 = new Dish("hokr", "oklkds", true, 6);
        Dish dish8 = new Dish("reren", "okooa", true, 0);
        Dish dish9 = new Dish("salo", "kokos", false, 7);
        Dish dish10 = new Dish("ppfd", "gdfcxx", true, 8);
        Dish dish11 = new Dish("vnmde", "das-fd", false, 14);
        Dish dish12 = new Dish("iddo", "fdscv", false, 13);
        Dish dish13 = new Dish("kllmm", "dfg", false, 11);
        Dish dish14 = new Dish("odas", "dsf ds", true, 8);
        Dish dish15 = new Dish("mmn", "fhgvhs", false, 7);
        Dish dish16 = new Dish("hgfo", "sawdqs", false, 4);
        Dish dish17 = new Dish("hog", "fdsf", true, 0);
        Dish dish18 = new Dish("ncs", "mmnvd", false, 7);
        List<Dish> dishes = Arrays.asList(dish1, dish2, dish3, dish4, dish5, dish6, dish7,
dish8, dish9, dish10, dish11, dish12, dish13, dish14, dish15, dish16, dish17);
        // Act
        Restaurant rest = new Restaurant(name, address, dishes);
        assertThrows(InvalidInvocationException.class, () -> rest.addDish(dish18, 14));
        // Assert
        assertEquals(rest.getDishes(), dishes);
        assertEquals(rest.getName(), name);
        assertFalse(rest.isVegetarian());
        assertEquals(rest.getAddress(), address);
    }


    @DataProvider
    private Object[][] computeInvalidDataForRestaurant() {
        Dish dish1 = new Dish("abc", "ok", false, 7);
        Dish dish2 = new Dish("bcde", "ko", false, 9);
        Dish dish3 = new Dish("cdefg", "fds", true, 10);
        Dish dish4 = new Dish("ab", "aaaaa", false, 11);
        Dish dish5 = new Dish("jkrig", "m", false, 14);
        Dish dish6 = new Dish("ytong", "odfdgk", false, 0);
        Dish dish7 = new Dish("hokr", "oklkds", true, 6);
        Dish dish8 = new Dish("reren", "okooa", true, 2);
        Dish dish9 = new Dish("salo", "kokos", false, 7);
```

```
        Dish dish10 = new Dish("ppfd", "gdfcxx", true, 8);
        Dish dish11 = new Dish("vnmde", "das-fd", false, 0);
        Dish dish12 = new Dish("iddo", "fdscv", false, 13);
        Dish dish13 = new Dish("kllmm", "dfg", false, 11);
        Dish dish14 = new Dish("odas", "dsf ds", true, 8);
        Dish dish17 = new Dish("hog", "fdsf", true, 0);
        Dish dish18 = new Dish("ncs", "mmnvd", false, 0);
        Dish dish19 = new Dish("test", "ekekks", true, 6);
        Dish dish20 = new Dish("vwwdg", "ekek21", true, 6);
        Dish dish21 = new Dish("yred", "ekek14", true, 4);
        Dish dish22 = new Dish("olfh", "ekek745", true, 3);
        Dish dish23 = new Dish("mnbd", "ekes3", true, 6);
        Dish dish24 = new Dish("qwsdq", "eke24ks", true, 5);
        Dish dish25 = new Dish("zzxc", "eke3kks", true, 9);
        Dish dish26 = new Dish("zztop", "e79kekks", true, 8);
        Dish dish27 = new Dish("dmode", "eke32kks", true, 11);
        Dish dish28 = new Dish("oelzy", "ek0e2kks", true, 13);
        Dish dish29 = new Dish("vwvw", "ek3e9kks", true, 14);
        Dish dish30 = new Dish("acdc", "ek3ek5ks", true, 15);
        Dish dish31 = new Dish("metal", "ek3ek5ks", true, 1);
        return new Object[][] {
                {"Musow", "Str. Levytskoho", Arrays.asList(dish2, dish1, dish9, dish13,
dish17), false},
                {"VegKeg", "Ahahaha", Arrays.asList(dish3, dish7, dish8, dish10, dish14,
dish19, dish20, dish21, dish22, dish23, dish24, dish25, dish26, dish27, dish28, dish29, dish30,
dish31), true},
                {"Toronto", "Av. de Libedrade", Arrays.asList(dish4, dish5, dish17, dish11,
dish12, dish13, dish18, dish6), false},
        };
    }

    @Test(dataProvider = "computeInvalidDataForRestaurant")
    public void testRestaurantRejected(String name, String address, List<Dish> dishes, boolean
isVegetarian) {
        // Assert
        assertThrows(InvalidInvocationException.class, () -> new Restaurant(name, address,
dishes));
    }
}
```