МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ОБОВ'ЯЗКОВЕ ДОМАШНЄ ЗАВДАННЯ

з дисципліни

«Реінжинірінг та верифікація програмного забезпечення»

1 варіант

Виконав: студент групи ІНм-91н

Тарасов О.О.

Перевірив: Москаленко В. В.

СУМИ 2019

Зміст

Завдання 1....3
Текст завдання....3
Класи еквівалентності :....3
Для імені елементу....3
Для ціни....4
Для кількості товару....4
Тип товару....4
Код тестів :....4
AddItemTest....4
CalcDiscountTest....6
Завдання 2....7
Текст завдання....7
Детальний рефакторинг коду....7
Код тестів....8
Висновок....8

Завдання 1

Текст завдання

Дано код програми торгового кошика ShoppingCart.

Необхідно описати класи еквівалентності та здійснити модульне тестування для методу addItem, для якого сформульовано наступні вимоги:

1) назва товару повинна бути не пустою і не перевищувати 32 символи;

2) ціна повинна бути більшою нуля і меншою 1000;

3) кількість товару повинна бути більшою нуля, але не більшою 1000;

4) тип товару повинен бути одним з допустимих типів, описаних константами

Item.Type;

5) при додаванні 100-го товару повинно викинутися виключення

IndexOutOfBoundsException;

6) при неправильному параметрі товару повинно викинутися виключення

IllegalArgumentException.

Необхідно описати класи еквівалентності та розробити параметризовані модульні тести для методу ShoppingCart.calculateDiscount, який розраховує знижку (Discount) за наступними правилами:

1) для REGULAR товаров знижки немає;

2) для SECOND товарів знижка становить 50%, якщо їх купують більше одного;

3) для DISCOUNT товарів знижка становить 10% і по 10% за кожний повний десяток товарів, але не більше 50% сумарно;

4) для SALE товарів знижка становить 90%;

5) крім того за кожну повну сотню товарів додається 10% знижки, але так, щоб сумарна знижка не перевищувала 80%.

Повна ціна розраховується як ціна товару × кількість × знижку.

Класи еквівалентності :

Для імені елементу

N — кількість символів

N = 0 — невалідний пустий рядок

1 <= N <= 32 — валідне ім’я

N > 32 — невалідне ім’я

Для ціни

N — ціна

N <= 0 — невалідна ціна

0 < N < 1000 — валідна ціна

N >= 1000 — невалідна ціна

Для кількості товару

N - к-сть товару

N <=0 — невалідна к-сть

0 <= N <= 1000 — валідна

N > 1000 — невалідна

Тип товару

{Second, Regular, Sales, Discount} — валідний тип

Код тестів :

AddItemTest

```
import com.github.javafaker.Faker;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;
import static odz.ShoppingCart.ShoppingCart.Item.Type.*;
public class AddItemTest {
    Faker faker = new Faker();
    private ShoppingCart cartInstance;
    @Rule
    public ExpectedException thrown = ExpectedException.none();
    @Before
    public void setUp(){
        cartInstance = new ShoppingCart();
    }
    @Test
    public void addItemTitleUnderInvalidEquivalentClass(){
        thrown.expect(IllegalArgumentException.class);
        thrown.expectMessage("Illegal title");
        cartInstance.addItem(faker.lorem().characters(35), 1.0, 1,
                REGULAR);
    }
    @Test
```

```
    public void addItemTitleUpperInvalidEquivalentClass(){
        thrown.expect(IllegalArgumentException.class);
        thrown.expectMessage("Illegal title");
        cartInstance.addItem(faker.lorem().characters(0), 1.0, 1,
                REGULAR);
    }
    @Test
    public void addItemTitleValidEquivalentClass(){
        cartInstance.addItem(faker.lorem().characters(2), 1.0, 1,
                REGULAR);
    }
    @Test
    public void addItemPriceUnderInvalidEquivalentClass(){
        thrown.expect(IllegalArgumentException.class);
        thrown.expectMessage("Illegal price");
        cartInstance.addItem(faker.lorem().characters(15), -1.0, 1,
                REGULAR);
    }
    @Test
    public void addItemPriceUpperInvalidEquivalentClass(){
        thrown.expect(IllegalArgumentException.class);
        thrown.expectMessage("Illegal price");
        cartInstance.addItem(faker.lorem().characters(15), 1001.0, 1,
                REGULAR);
    }
    @Test
    public void addItemPriceValidEquivalentClass(){
        cartInstance.addItem(faker.lorem().characters(15), 2.0, 1,
                REGULAR);
    }
    @Test
    public void addItemQuantityUnderInvalidEquivalentClass(){
        thrown.expect(IllegalArgumentException.class);
        thrown.expectMessage("Illegal quantity");
        cartInstance.addItem(faker.lorem().characters(15), 1.0, -1,
                REGULAR);
    }
    @Test
    public void addItemQuantityUpperInvalidEquivalentClass(){
        thrown.expect(IllegalArgumentException.class);
        thrown.expectMessage("Illegal quantity");
        cartInstance.addItem(faker.lorem().characters(15), 1.0, 1002,
                REGULAR);
    }
    @Test
    public void addItemQuantityValidEquivalentClass(){
        cartInstance.addItem(faker.lorem().characters(15), 1.0, 505,
                REGULAR);
    }
    @Test
    public void addItemSizeValidEquivalentClass(){
        int size = 54;
```

```
        for(int i=0;i<size; ++i){
            cartInstance.addItem(faker.lorem().characters(15), 1.0,
                    550, REGULAR);
        }
    }
    @Test
    public void addItemSizeInvalidEquivalentClass(){
        thrown.expect(IndexOutOfBoundsException.class);
        thrown.expectMessage("No more space in cart");
        int size = 100;
        for(int i=0;i<size; ++i){
            cartInstance.addItem(faker.lorem().characters(15), 1.0,
                    550, REGULAR);
        }
    }
    @Test
    public void addItemTypeValidEquivalentClass(){
        cartInstance.addItem(faker.lorem().characters(15), 1.0, 300,
                SALE);
    }
}
```

CalcDiscountTest

```
import com.github.javafaker.Faker;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import java.util.Arrays;
import java.util.Collection;
import static odz.ShoppingCart.ShoppingCart.Item;
import static odz.ShoppingCart.ShoppingCart.Item.Type.*;
import static org.junit.Assert.*;
@RunWith(Parameterized.class)
public class CalcDiscountTest {
    private int discount;
    private Item currentItem;
    private static final Faker faker = new Faker();
    public CalcDiscountTest(Item currentItem , int discoutPredicted){
        this.currentItem = currentItem;
        discount = discoutPredicted;
    }
    @Parameterized.Parameters
    public static Collection getParametrizedItems(){
        return Arrays.asList(new Object[][]{
                {new Item(faker.lorem().characters(15), 20.0,
                        1, REGULAR ), 0},
                {new Item(faker.lorem().characters(15), 20,
                        1, SECOND), 0},
                {new Item(faker.lorem().characters(15), 20,
                        2, SECOND), 50},
                {new Item(faker.lorem().characters(15), 20,
```

```
                        99, SECOND), 50},
                {new Item(faker.lorem().characters(15), 20,
                        100, SECOND), 60},
                {new Item(faker.lorem().characters(15), 20,
                        101, SECOND), 60},
                {new Item(faker.lorem().characters(15), 20,
                        500, SECOND), 80},
                {new Item(faker.lorem().characters(15), 20,
                        1, DISCOUNT), 10},
                {new Item(faker.lorem().characters(15), 20,
                        9, DISCOUNT), 10},
                {new Item(faker.lorem().characters(15), 20,
                        10, DISCOUNT), 20},
                {new Item(faker.lorem().characters(15), 20,
                        30, DISCOUNT), 40},
                {new Item(faker.lorem().characters(15), 20,
                        90, DISCOUNT), 50},
                {new Item(faker.lorem().characters(15), 20,
                        99, DISCOUNT), 50},
                {new Item(faker.lorem().characters(15), 20,
                        100, DISCOUNT), 60},
                {new Item(faker.lorem().characters(15), 20,
                        500, DISCOUNT), 80},
                {new Item(faker.lorem().characters(15), 20,
                        101, DISCOUNT), 60},
                {new Item(faker.lorem().characters(15), 20,
                        1, SALE), 80},
                {new Item(faker.lorem().characters(15), 20,
                        500, SALE), 80}
        });
    }
    @Test
    public void discountCalculating(){
        assertEquals(discount,
ShoppingCart.calculateDiscount(currentItem));
    }
}
```

Завдання 2

Текст завдання

Сформувати для заданого коду модульні тести (з використання JUnit), покрити ними наступні методи (вимоги описані в коментарях до методів):
ShoppingCart.appendFormatted(StringBuilder,String,int,int)
ShoppingCart.calculateDiscount(ItemType,int)
Провести рефакторинг коду, дописуючи тести в разі необхідності з метою зменшення імовірності появи помилок редагування. Варто покривати тестами метод, перед його розбиванні на складові частини.

Детальний рефакторинг коду

1) Винести клас Item в окремий файл, в даний клас також перемістити типи елементів у вигляді типу перерахування. Згенерувати конструктор для додавання одразу всіх параметрів до об’єкту, також створити get та set методи до всіх полів класу. Оновити код в класі ShopingCart в методі addItem — створення об’єкту.
2) Виділити перевірку валідності елементів доданого item в окремий метод validate
3) Перенести метод calculateDiscount у клас Item та зробити його нестатичним, замінити підрахунки вручну на цей метод у класі ShoppingCart
4) Створити метод calculateTotal у класі Item, замінити підрахунки вручну на цей метод у класі ShoppingCart
5) Створити клас TableFormatter і делегувати йому створення та форматування табличного представлення.

Код тестів

```
import org.junit.Test;

import static odz.ShoppingCart.Item.Type.DISCOUNT;
import static odz.ShoppingCart.Item.Type.SALE;
import static org.junit.Assert.*;

public class TableTest {

    @Test
    public void testTable(){
        String result = "# Title                    Price Quantity
Discount      Total\n" +
"-----------------------------------------------------------------\n" +
                        "1 aaa                      $100.00      230
80%   $4600.00\n" +
                        "2 adlmajsnc               $123.00      130
60%   $6396.00\n" +
                        "3 as.kjcas,kjcbaskjcbk... $550.00      355
80%  $39050.00\n" +
"-----------------------------------------------------------------\n" +
                        "4
$50046.00";
        ShoppingCart cart = new ShoppingCart();
        cart.addItem("aaa", 100.0, 230, SALE);
        cart.addItem("adlmajsnc", 123.0, 130, DISCOUNT);
        cart.addItem("as.kjcas,kjcbaskjcbk,asbckasbc", 550, 355, SALE);
        assertEquals(result, cart.toString());
    }
}
```

Висновок

В ході виконання роботи я навчився проводити рефакторінг та використовувати методику unit тестування для перевірки правильності функціонування методів. Для тестування була використана бібліотеки JUNIT версії 4. Були написані тести, які перевіряють чи функція повертає правильний результат та чи виникає виключення в ході виконання методів. Також були написані параметризовані тести, котрі дозволяють викликати одну функцію і перевірити результат виконання для колекції вхідних параметрів. Ця методологія дозволила провести рефакторінг кодової бази для покращення якості коду.