Assignment 2 Report
Software Verification and Validation
2019–2020

João David
49448

09/06/2020

1 HTML Unit

While defining the HTML Unit tests, some adaptations where made to the JSP files. The $name$ property was added to some HTML elements, in order to be able to get them using HTML Unit's API, those adaptations were made to the following files:

- CustomerInfo.jsp
- SalesInfo.jsp
- ShowSalesDelivery.jsp
- addSaleDelivery.jsp
- SalesDeliveryInfo.jsp

An ID column was added to the addresses table in the CustomerInfo.jsp. This was useful when doing narrative e) because it allows to assert that the just added address is the same that was associated in the sale delivery.

Regarding the tests themselves, a TestUtils class was created with useful methods such as, addCustomer, addSale and addAddress. This helps have a more easy to read code within the test methods, and reduces duplicate code.

In order to check that the report pages have the expected information, the methods previously mentioned return an HtmlPage object, that is then used in the assert functions. Other methods such as getCustomers and getCustomerSales return an HtmlTable instead, for the same reasons.

All the narratives were written inside the same test class, since not all of them have the same set up and tear down rules, and to avoid having several test classes with the same WebClient setup code. The set up and tear down pieces of code where added to the top and bottom of each test method respectively, they were clearly marked with a comment, in order to better differentiate test code from set up/tear down code.

2 DB Setup

Each task from the assignment was defined in a separate test method. The last four extra tests concerning the expected behavior of sales and sale deliveries are in the bottom of the test class, with a comment block describing them. The following two pages also contain that information.

The class DBTest relies on a Utils class DBSetupUtils, that is responsible for inserting some values in the database, so that to test class itself has some information to work with.

2.1 Sales Behavior

The first Junit test verifies that it is not possible to add a sale to an non existent customer.

```
public void extraSaleBehaviour1() throws
   ApplicationException {
  int vat = 503183504;
  assertFalse(hasClient(vat));
  assertThrows(ApplicationException.class, () -> {
    SaleService.INSTANCE.addSale(vat);
  });
}
```

All new sales created for a new customer must have its date the same as of creation, a total of 0.0, an open status and it needs to be associated to the right vat.

```
public void extraSaleBehaviour2() throws
   ApplicationException {
  int vat = 503183504;
  assertFalse(hasClient(vat));
  CustomerService.INSTANCE.addCustomer(vat, "FCUL",
     217500000);
  assertTrue(hasClient(vat));
  SaleService.INSTANCE.addSale(vat);
  List<SaleDTO> sales =
     SaleService.INSTANCE.getSaleByCustomerVat(vat).sales;
  SimpleDateFormat dateFormat = new
     SimpleDateFormat("yyyy-MM-dd");
  for (SaleDTO curr : sales) {
    assertEquals(dateFormat.format(new Date()),
       curr.data.toString());
    assertEquals(new Double(0.0), curr.total);
    assertEquals("O", curr.statusId);
    assertEquals(vat, curr.customerVat);
  }
}
```

2.2 Sale Deliveries Behavior

After a sale has been closed, it should not be possible to add a delivery for that sale

```
public void extraSaleDeliveryBehaviour1() throws
   ApplicationException {
  int vat = 197672337;
  assertTrue(hasClient(vat));
  SaleService.INSTANCE.addSale(vat);
  assertEquals("O",
     SaleService.INSTANCE.getSaleById(1).statusId);
  SaleService.INSTANCE.updateSale(1);
  assertEquals("C",
     SaleService.INSTANCE.getSaleById(1).statusId);
  assertThrows(ApplicationException.class, () -> {
    SaleService.INSTANCE.addSaleDelivery(1, 1);
  });
}
```

After removing a customer, its sale deliveries should be removed as well

```
public void extraSaleDeliveryBehaviour2() throws
   ApplicationException {
  int vat = 197672337;
  assertTrue(hasClient(vat));
  SaleService.INSTANCE.addSaleDelivery(1, 1);
  assertNotEquals(0, SaleService.INSTANCE
       .getSalesDeliveryByVat(vat).sales_delivery.size());
  CustomerService.INSTANCE.removeCustomer(vat);
  assertFalse(hasClient(vat));
  assertEquals(0, SaleService.INSTANCE
    .getSalesDeliveryByVat(vat).sales_delivery.size());
}
```

3 Mockito

It's possible to mock the CustomerRDGW class. Suppose that the application layer is being developed, and the access to the data base is still not set up, or is too slow at the moment, and the programmer does not want to stop working.

In this case, the class that would be using the mock is the CustomerService class, the class uses the CustomerRDGW to insert, update, get and even remove customers from the database, since we would be mocking this behavior, the mock would need to keep track of all customers in memory (instead of going to the DB).

Looking at the original code, the class CustomerRDGW depends on the CustomerFinder class, in order to simplify the mocking process, the only method within the CustomerFinder class would be moved to the CustomerRDGW, this way, the CustomerRDGW would not depend from any other class and all the knowledge about the customers in the system would be in CustomerRDGW.

The mock itself would rely on some data structure to "save" the instances of CustomerRDGWs "inserted" in the database. A good data structure would be a hash map, where the key would be the VAT number (unique number) and the value would the the CustomerRDGW itself, the mock class would have to implement the CustomerRDGWs methods, for instance, the addCustomer would create a new CustomerRDGW and put it in the hashmap using its VAT, and the getCustomer would get it from the hashmap using its VAT number.

4 Bugs found

4.1 Customer Removal

After removing a registered customer from the system, its addresses, sales and sale deliveries were still kept in the database.

4.1.1 Reproduction

1. Create e new customer
2. Add an address
3. Insert a new sale
4. Insert a new sale delivery using the previous two information
5. Remove the customer
6. Use the customer's vat number to search for sales/sale deliveries

4.1.2 Solution

First implement in the Address, Sale and SaleDelivery RDGW classes, the methods responsible for deleting all addresses, sales and sale deliveries, respectively, given a customer VAT number. Then, use them to delete the information in the $removeCustomer$ method in the $CustomerService$ Java class.

4.2 Insert new sales for non-existent customers

The system allows the creation of new sales associated to VAT numbers that do not belong to any customer registered in the system.

4.2.1 Reproduction

1. Click on "Enter new sale"
2. Enter a VAT number that is not registered to any customer

4.2.2 Solution

Within the $addSale$ method from the $SaleService$ class, use the method $getCustomerByVATNumber$ from the $CustomerRDGW$ class, to query the Customer's DB table, using his VAT number. If such customer does not exist, the system should raise an exception.

4.3 Adding an address

The system allows the addition of addresses with invalid/empty parameters. For instance, one could add an address empty address or even an address where the zip code has no numbers, or a locality that is only numbers.

4.3.1 Reproduction

1. Click on "Insert new Address to Customer"
2. Enter a VAT number that is registered to a customer
3. Fill the fields with wrong values, or leave them empty
4. Insert the address

4.3.2 Solution

Before inserting the address into the database, there should be a validation of the fields written by the user. There could also be a regular expression validation in the form itself.

4.4 Adding a sale delivery for a closed sale

The system allows the creation of a Sale Delivery for a Sale that has already been closed.

4.4.1 Reproduction

1. Add a Customer
2. Add an address and sale for that Customer
3. Close the sale previously created
4. Create a Sale Delivery using the address and sale created previously

4.4.2 Solution

Before creating the Sale delivery, the system should verify that the sale chosen is still open. This could be done through the use of SaleRDGW's getSaleById method.

4.5 RDGW missing attribute initialization

The constructor of $SaleDeliveryRowDataGateway$ is not initializing the attribute $address\_id$ and the constructor of $SaleRowDataGateway$ is not initializing the attributes $total$ and $statusId$.

4.5.1 Solution

Using the ResultSet object passed to the constructor, initialize the previously mentioned attributes.

4.6 Creation of unnecessary objects

The RDGW classes have empty constructors, that are being used to create RDGW objects in order to use their instance methods. This is a bad practice, this methods should be re factored to static, and the empty constructors removed. The methods in particular are getters, that return a RDGW object, or a list of RDGW's.

4.6.1 Solution

Remove all RDGW empty constructors, then the Eclipse IDE will signal the places where they were previously used to invoke instance methods, change those methods to static, and then instead of using "new xRDGW().get..." use "xRDGW.get...".