Assignment 2

Mestrado em Engenharia Informática
Verificação e Validação de Software
2018/2019

Grupo 6:
Gonçalo Lobo 44870
Nuno Sousa 47164

1. Índice

2. HtmlUnit Tests

As classes *InsertNewAddress.java*, *InsertFirstCustomerAgain.java*, *CreateAndRemoveCustomer.java*, *CreateNewSale.java* inseridas no package *part1* são as classes responsáveis por testar as narrativas (a), (b), (c) e (d) presentes no ponto 1 do enunciado, respetivamente.

(a) Para esta alínea criou-se a classe *InsertNewAddress.java*. Em primeiro lugar é usou-se o método *getNumberRows()* para obter o número inicial de linhas na tabela. De seguida, com o método *insertNewAddressTest()* inseriu-se a nova morada do cliente, testou-se para ver se realmente os parâmetros foram colocados na tabela de moradas e obteve-se o número de linhas após a inserção. Por fim, testou-se este último número para ver se correspondia ao número inicial de linhas na tabela mais um.

(b) Para esta alínea criou-se a classe *InsertFirstCustomerAgain.java*. Primeiro foi usado o método *getFirstCustomer()* para obter o primeiro customer de todos existente na tabela dos customers. Após isso, com o método *insertCustomerAgain()* inseriu-se esse primeiro customer e testou-se para ver se o erro resultante da inserção desse cliente seria igual ao esperado. Essa verificação foi feita através da obtenção do erro da página e a verificação do npc do customer.

(c) Para esta alínea criou-se a classe *CreateAndRemoveCustomer.java*. Em primeiro lugar usou-se o método *insertNewCustomer()* que insere um novo cliente, verifica-se se os parâmetros realmente foram colocados na tabela de clientes e de seguida remove-o, verificando primeiro que os parâmetros anteriormente colocados já não se encontram na tabela dos clientes. Por fim usou-se o método *listAllClients()*, que verifica que os clientes existentes na tabela dos clientes são os que o sistema tinha antes da inserção de um novo cliente.

(d) Para esta alínea criou-se a classe *CreateNewSale.java*. Primeiro, obtém-se o id da última sale do cliente com o método *getLastSaleId().* Depois cria-se uma nova sale para esse mesmo customer e no final usa-se o método *checkNewSaleId()* para fazer as verificações de que a nova sale foi criada, ou seja, que o id da nova sale seja diferente do id da última sale. Por fim, cria-se a nova sale delivery através do método *createNewSaleDelivery()* que, primeiro, obtém o id da última sale delivery e depois cria a sale delivery para a sale em questão. Por fim, usa-se o método *checkNewSaleDelivery()* que obtém o id da nova sale delivery e verifica no final que o id da nova sale delivery é igual ao id da delivery anterior mais um.

3. DBSetup

Para podermos testar todos os casos deste exercício, criámos uma package *part2* na qual estão inseridas todas as classes correspondentes às alíneas indicadas no enunciado. Assim, as alíneas a), b) e d) estão tratadas na classe *CustomersDBTest* e as alíneas c) e e) estão tratadas na classe *SalesDBTest*. De seguida vamos explicar os testes executados nas várias alíneas:

(a) A alínea (a) é concretizada através do método *checkCustomerUpdate()*, no qual primeiro obtém-se o customer com o vat correspondente e depois atualiza-se a informação desse mesmo cliente, mais propriamente o número de telemóvel. Por fim, obtém-se esse cliente novamente e verifica-se que a informação foi guardada.

(b) A alínea (b) é concretizada através do método *deleteAllButOneCustomer()*. Em primeiro lugar é efetuada a remoção de todos os clientes exceto um e depois é verificado que de facto ao remover todos os clientes menos um apenas ficamos com um cliente. Por fim, também são verificados os dados do cliente.

(c) A alínea (c) é concretizada através do método *deleteCustomerAndDeliveries()*. Em primeiro lugar obtém-se um customer, neste caso o primeiro customer existente, e as suas sales deliveries. Após isto, remove-se o cliente, as sales desse mesmo cliente e também as sales deliveries. Por fim, verifica-se que após remover as sales e as sales deliveries estas de facto não existem.

(d) A alínea (d) é concretizada através do método *addDeletedCustomer()*. Em primeiro lugar vamos obter um customer existente através do seu vat. Depois remove-se o cliente, as sales e volta-se a inserir o mesmo customer, com o mesmo vat, designação e número de telemóvel. Por fim, verifica-se que ao obter o cliente pelo vat ele de facto existe e de que foi adicionado sem problema.

(e) A alínea (e) é concretizada através do método *addSaleDelivery()*. Em primeiro lugar vai-se buscar um dos customers, neste caso o primeiro customer existente. De seguida adiciona-se uma sale delivery e verifica-se que ao adicionar uma sale delivery o número de sales delivery é mais um do que o número inicial.

Para além destes testes explicados anteriormente, foram realizados dois testes extra em relação ao comportamento esperado das sales:

1. O primeiro teste adicional é concretizado através do método *addSaleNumber()*. Primeiramente obtém-se as sales iniciais antes de adicionar uma nova sale. De seguida, vai-se buscar um cliente existente e adiciona-se uma nova sale a esse mesmo cliente. Por fim, verifica-se que o número de sales aumentou ao comparar que o número inicial de sales mais um é igual ao número de sales depois de ter adicionado a sale.
2. O segundo teste adicional é concretizado através do método *addSaleVatInvalid()*. Neste adiciona-se uma venda com um vat não existente, de seguida é verificado se é lançada uma exceção. Para que isto acontecesse

foi alterado o método *addSale()* da classe *SaleService* de forma a verificar se o vat é válido e se existe um cliente com este vat, caso contrário é lançada uma exceção de vat inválido.

4. Mockito

A terceira parte do trabalho é em relação à utilização da ferramenta Mockito. Não é possível usar o Mockito uma vez que os serviços que poderiam ser alvo da ferramenta Mockito são enumerados. Em particular, o CustomerService e o SaleService.

5. Alterações no SUT

Nesta secção serão explicadas as alterações que foram feitas no SUT resultantes de erros que foram detetados durante a elaboração dos testes.

1. Ao elaborar o teste *addDeletedCustomer()*, que verifica que ao apagar um cliente é possível adicioná-lo de volta sem lançar exceções, verificámos que quando um cliente era removido não eram removidas as suas sales nem as sales deliveries, como seria de supor. Assim:
    a. Na classe *SaleService.java* criámos os métodos *removeSale(int vat)* e *removeSalesDeliveries(int vat)* que removem as sales e as sales deliveries de um determinado cliente, identificado pelo vat.
    b. Na classe *SaleDeliveryRowDataGateway.java* foi criado o método *removeSalesDelivery()* que é responsável por remover a sale delivery associada ao vat do customer. Para isto também foi criado o remove delivery SQL statement REMOVE_SALE_DELIVERY_BY_VAT.
    c. Finalmente, na classe *SaleRowDataGateway.java* foi criado o método *removeSale()* que remove a sale associada ao vat do customer e para isto também foi criado o remove sale SQL statement REMOVE_SALE_SQL.
2. Ao elaborar o teste extra *addSaleVatInvalid()* verificámos que era possível adicionar uma venda a um cliente inexistente. Para tal:
    a. Alterámos o método *addSale(int customerVat)* da classe *SaleService.java* de forma a verificar se o vat é válido e se existe um cliente com este vat, caso contrário é lançada uma exceção de vat inválido.
3. Em relação aos testes da base de dados:
    a. A data sample insertSaleDeliveries foi criada com uma saleDelivery associada a uma sale e a um customerVat existentes.
    b. Foi criada a operação INSERT_CUSTOMER_SALE_DATA que combina as data samples de insertCustomers, insertSales, insertSaleDeliveries e insertAddresses.
    c. Foi criada uma variável referente ao número inicial de sale deliveries, NUM_INIT_SALES_DELIVERIES, criada na elaboração do teste addSaleDelivery, que verifica que ao adicionar as sales deliveries o seu número total aumenta mais um.