John Chau

5/24/2021

SE333

Code Coverage Analysis

a) Every single line of code in the PurchaseDBO, ShoppingCartApi, ShoppingAppApplication, Accounts classes are impossible to reach. With the PurchaseDBO class, the code is never tested because we are using a mock database for testing. Similarly, we are not testing the web interface so of course there is 0% coverage for the ShoppingCartApi in our tests. Also, the ShoppingAppApplication class generates 0% coverage in our tests because we aren’t testing the server. Finally, the Accounts class, we are not testing user accounts so the “code” in there is impossible to reach too.

An example of difficult to reach code is in the TaxCalculator class. Specifically, the construction of a TaxCalculator object is difficult to reach with our equivalence partitioning and boundary tests of the TotalCostCalculator class. Since the equivalence partition and boundary tests never create a TaxCalculator object, this code is never tested. In normal tests, it is also unnecessary to test this anyway.

I would also consider the “get” (and also the setIDNum) methods in the Purchase function to be difficult to reach code because there’s no real reason to actually use them in our tests since we’re making purchases ourselves so we already know these details since we created them. The toString() method for this class is also not really useful in our tests.

b) 80% is not achievable for the classes that currently have 0% code coverage because we never test their functionality.

I don’t think the Purchase class can reach 80% coverage because the get and toString methods have no reason to be called in our tests. Similarly, I don’t think the PurchaseAgent class can reach 80% coverage because the database is not saving purchases properly so one branch of the save function will be missed, and so will all the statements in the getPurchases function.

The PurchaseItem class might reach 80% coverage if the set methods are called although I am skeptical because there are private variables and also get methods that have no reason to be called.

Some of the classes such as Bill, ShoppingCart, TotalCostCalculator, TaxCalculator, already have above 80% coverage.

c) For the Purchase class, I am missing branch coverage for all of the get functions because they’re never called.

For the PurchaseAgent class, I am missing branch coverage for the averagePurchase function.

d) One class we should not consider testing is the TaxCalculator class. This class already had 93% code coverage just from doing the boundary tests and equivalence partitioning tests on the TotalCostCalculator class. The only part that wasn’t tested was the creation of a new TaxCalculator object (with the default constructor since one wasn’t given), but that’s unnecessary since we never need to create a TaxCalculator object for ourselves, and all the logic is already tested.

Similarly, ShippingType is a class we don’t need to test. This class already has 100% coverage from the equivalence partitioning tests for the TotalCostCalculator class.

The Bill class is another class that doesn’t need to be tested. Just by testing the first calculate function of the TotalCostCalculator class, we already reach 91% test coverage for this class. The missing 9% for my tests came from not including the total() function (which is different from the getTotal() function) in my tests but that is redundant since they both do the same thing.

Another class that doesn’t need to be tested is the Accounts class. There’s only one function in it, and that function has no logic so it is irrelevant to the project. It should be removed or added to.

e) The PurchaseDBO, ShoppingCartApi, ShoppingAppApplication should be tested to see if it the database, web interface and the server respectively are running correctly. However, all three of these tests are out of scope for this assignment. We should also test the Accounts class too to see if that works once more code is added in there that does something.