Alright, this test is so bulky, we need to refactor it and make it cleaner and more maintainable. Look at these few lines, initialization

of the unit of work, creating these mock objects, and initializing the service. These are the lines that we need in every test, and we don't want to repeat them over and over, so let's move all these lines inside the SetUp method.

So, let's create the SetUp method, public void SetUp.

Now, move all these few lines right here. So, with this we end up with a test method that is only two lines of code. Now finally, we need to fix these compilation issues. So here's our service, we need to

promote this to a private field, so we can reuse it in different methods.

So, alt and enter, refactor, Introduce Field.

Now, I want to make the name shorter, instead of houseKeeperService, I just want to use, service, because it's obvious we're testing the houseKeeperService class here. Okay? Now, we fixed this issue. Done. The second one is our statement generator.

Once again, we need to promote this to a private field. So alt and enter, refactor, Introduce Field. I'm going to keep this name as is, statementGenerator okay? And, finally let's fix this issue, beautiful.

Now in the future videos, as we write more unit tests, we're going to need more these objects as well. So, let's quickly promote this to private fields. Time, Introduce Field, and the last time. Okay, also, I don't like that we have repeated this DateTime in two places, this is making the test a little bit noisy and also I don't want the reader of this code to assume that this particular date is a special date In the domain of this application. This is just a simple value. The actual date doesn't matter. So I'm going to extract this into a private field as well. One more time alt and enter, refactor, introduce field, now Rider is asking to replace both occurrences, or only this one. We want to replace both occurrences. So, it's going to create a private field, called statementDate.

So this is where statementDate is initialized we don't want to repeat this in every test, so we're going to delete this line, we can either put it inside the SetUp method but this method is already a little bit too busy, so I would rather to initialize it directly here. Because DateTime objects are immutable, so it's not possible that somewhere in this class we're going to modify this DateTime object once we create it, it's not mutable or changeable, so we set this directly to a new DateTime.

Now, back to our test method, look, its even cleaner.

Also, it's better to replace these magic values, with a reference to a private field. Because the reader of this code has to scroll up and down

to figure out what is one, what is b, this is where we have defined this housekeeper object, I would rather to extract this into a private field so we can reference it in multiple tests. So, let's select this line and extract this into a private field.

We're going to call this Housekeeper.

Okay? So, we want to keep this inside the test method because I want to make sure before each test, we're going to use a new fresh housekeeper object, in other words I don't want one test impacting the state of this class, I want to make sure that every test starts on a clean state.

However, I don't like the organization of this code, because here we're creating this unitOfWork, and then we move on and create a housekeeper object, and finally. we come back to the unitOfWork again. It's better to create the houseKeeper object first, let's put this here, now we create a unitOfWork, and then program it, so there is no distraction in our code.

Now, finally, I'm going to replace one with houseKeeper.Oid and b with houseKeeper. to full name. Now we can see this line is going beyond the screen, so I want to reformat it like this. That's better. If we're going any further, let's run the test one more time to make sure we have not broken anything. Okay, here's our test it passed, beautiful, next we'll create another interaction test.