In the last lecture we renamed this class to ReservationTests but apparently the renamed refactoring feature in Visual Studio didn't work properly, because the name of the file is still unit test one. So, I'm going to rename this to ReservationTests.cs. With this convention, you can easily find all of the tests for a given class.

Okay, beautiful, now let's write the second unit test. So, I'm going to write a public void method, and by the way, all your test methods should be public void.

So, here we're going to use our convention to call this method, the first part is the method on the test, that is CanBeCancelledBy. The second part is the scenario, so let's say we want to test the scenario where the same user who made the reservation wants to cancel this reservation. So, SameUserCancellingThe Reservation. Now the third part, is the expected behavior, what do we expect this method to return? It should return true, Okay? Now, inside this method we should have three parts arrange, act, and assert. In the last lecture I added comments to separate the arrange, act, and assert parts, but in a real world scenario we don't necessarily have to do this, if your test method is short and clean. So, in the arrange part, I'm going to create a new reservation object. So, reservation is a new reservation, in the acts, I'm going to call reservation, that CanBeCancelledBy now here I want to simulate a scenario where the same user is cancelling this reservation, so I'm going to create a user object here, and first of all set the MadeBy property

to this user. So this user created this reservation before.

Now, I want to pass the same user object as an argument to this method, user, now, we get the result, and store it. And finally make an assertion. So, we assert. That result is true.

So now with these vertical spaces I have added in the code, we can see the Arrange, the Act, and the Assert parts. Back in our test, export window, let's run all the tests, so we have one failing test, and this is because in the last lecture I created a bug in our code, so let me change this here we return true, now back to our test class, this new method I created here is not picked up by a test explorer. We don't see that here.

Why? Because we forgot to decorate it with the TestMethod attribute. So that's why you should decorate it every test method with this attribute, so it can be picked up by the test runner, now save, back in Test Explorer, run all, now we have two passing tests, beautiful, now, let's write the last test, so I'm going to start with test method,

then public void, can CanBeCancelledBy the third scenario is a user trying to cancel someone else's reservation. So, another user CancellingReservation.

What is the expected behavior? This method should return false, so return false. Once again, in the arrange part, I'm going to create

a reservation object, new Reservation and act part we call CanBeCancelledBy, now here we want to deal with two different user objects. So here I can pass the user object, and when initializing the

reservation, I can set MadeBy to a different User object. And finally, we need to store the result here, and Assert that result is false. So IsFalse, result. Now, let's run all the tests. So we have three passing tests, beautiful.