Okay, first thing we need to do to unit test this class is to refactor this code and take out the parts that touch external resources. So here we have three points where we work with external resources. Here's one point where we query our database, here's another point where we touch the file system, and here's the last point where we send an email. So, let's start the first one, UnitOfWork.

In the last section, where we deal with the database, I showed you how to extract the query logic, and encapsulate it inside a repository. So as I told you before, we use the repository pattern to encapsulate querys, in this case we don't have a complex query, we're simply getting all the housekeepers from the database.

So there's really no need to extract a new class called HousekeeperRepository. I would simply extract an interface from this unit of work. So we don't directly work with a concrete unit of work implementation that touches the database. So, let's go to this unit of work class, it's defined here now, alt and enter, refactor, and extract interface. So, IUnitOfWork, and we add this query method here, next, here is our interface, beautiful, now back to our housekeeper helper, look this method is defined as static. So as you learned in the last section, the only way to inject a dependency here is using a parameter in this method, but you already saw the solution in the last section, so in this section you want to use constructor injection. So, I'm going to remove the static modifier, and make this method an instance method. Chances are this is going to break some parts of the code, and we're going to have compilation issues, but let's assume we're not going to have a lot of breaking changes. So, now we have a compilation error because this method is defined in a static class. So we should also remove the static modifier here.

Okay now, we can add a constructor and inject an IUnitOfWorkObject.

So UnitOfWork now here we can use Rider or ReSharper to quickly create and initialize a private field, so we press alt and enter and the first option is introduce the read-only field, underline unit of work. So you can see

Rider created this private field and initialized it using this argument here. With this we no longer need this private static read only field. So delete and now we have a compilation error here because UnitOfWork does not exist so, let's change this to underline UnitOfWork.

Okay, beautiful. So we refactor the first point where we touch an external resource. Now the second point is where we save the statement in the file system. So, back in the solution explorer, I'm going to create a new class, let's call this Statement Generator. Okay. Now, back in our HousekeeperHelper class, let's go to the implementation of SaveStatement method, it's here, so I'm going to make this public, now, we can select this code, cut it, and back to our new class statement generator, paste it here. Let's import the misting (?) statements, beautiful, and finally we need to extract an interface from this class. So, the cursor is on the class name, alt and enter, refactor and extract interface.

Okay, note that here, we don't have any members to select. So, I forgot to remove this static modifier, because we cannot put static members

in an interface, that's why you should avoid using static because you cannot extract from an interface and Mock that interface in your unit tests. So, let's remove this. One more time.

Extract interface.

Okay, now we have this SaveStatement method, let's select it. Next, beautiful, here's our new interface, and finally we need to inject this inside our housekeeper helper class.

So, let's go to the constructor. here is the constructor, let's inject

IStatementGenerator. And once again we can use Rider or ReSharper to quickly create and initialize a private field. So alt and enter, introduce read-only field. Now finally, let's fix this compilation issue. So, we call statementGenerator. SaveStatement. And finally, the last point, that's where to send an email. Right here.

So once again, we need to create a new class, and use this private step method inside that class. So, back in the solution explorer, let's add a new class call this EmailSender.

Now, back here, let's go to the implementation of this method, it's right here, so I'm going to select this, cut and back in the email sender, paste it here.

Now, let's import all the missing references, done, now, you need to make this method public, and remove the static modifier.

And with this we are ready to extract an interface from this class.

So, Refactor, Extract Interface. Let's select this member, next, done, and, finally back in our housekeeperHelper class, we need to inject this inside the constructor.

So, our constructor has two parameters, and if I put the third one, it's not going to fit on the screen. So always pay great attention to formatting your code, in this case I would rather put each parameter on a single line, this way we can clearly see what parameters we have here. So IEmailSender, emailSender, and now let's initialize this private field, done, and the last thing.

We need to fix this compilation issue. And use this object emailSender. Actually, I noticed there is one more point where

we work on an external resource. Look at this line. ExtraMessageBox.Show. I'm not entirely sure what this class is, but looks like it's some kind of helper that displays a message box, so most likely this source code is used in a WPF or a Windows format application. So, when running our unit tests, we don't want to display a message box, that's also considered an external resource we only want to test the logic, we don't want to run the application and see a message box. So, we need to work with an interface here.

Now, let's look at the declaration of this extra message box. I've created that here, so all we need to do is to extract an interface, we don't need to wrap this inside another class like, extra message box helper, there's no need to create an extra class. So, if you have control over an existing class, simply extract an interface from that class if you're using a third party code that doesn't have an interface, then you will have to encapsulate that code inside a class in your application that you can control.

Okay, so, first we need to remove this static modifier, and then extract an interface called IXtraMessageBox. Let's add this method here. Now, we need to inject this inside the constructor. So IXtraMessageBox. Let's just call this messageBox. And create and initialize this private field. And finally, we can replace this with message Box.Show. So with these refactoring, this code is now loosely coupled and testable, so in the next lecture, we're going to write our first unit test for this method.