**Hands On Lab 1**

**Mocking Void Methods With EasyMock**

## Mocking a Void Method with EasyMock

### ****1. Overview****

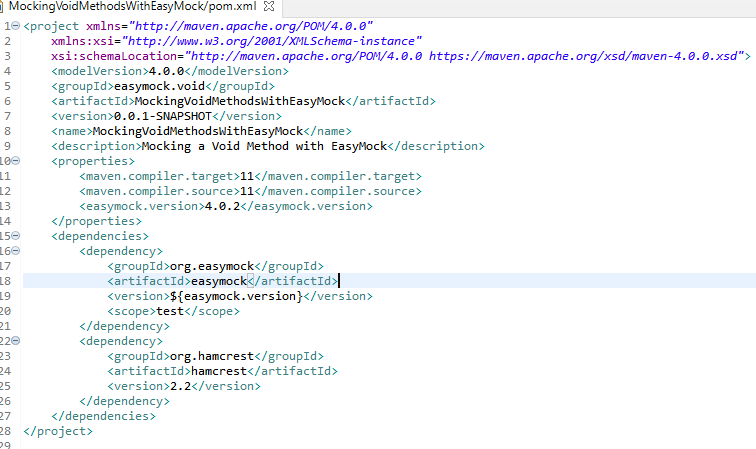
Mocking frameworks are used to mock interaction with dependencies so as to test our classes in isolation. Typically, we mock the dependencies to return the various possible values. This way, we can ensure our class can handle each of those values.

But, sometimes we might have to mock dependency methods that do not return anything.

In this lab, ****we will see when and how to mock***void***methods using EasyMock.****

### ****2. Maven Dependency****

First, let’s add [the EasyMock dependency](https://search.maven.org/search?q=g:org.easymock AND a:easymock) to our *pom.xml*:



You will also need the hamcrest dependency.

### ****3. When to Mock a***void***Method****

When we test classes with dependencies, we would normally want to cover all values returned by the dependency. But sometimes, the dependency methods do not return a value. So, ****if nothing is returned, why would we want to mock a***void***method?****

Even though void methods do not return a value, ****they might have side-effects.**** An example of this is the *Session.save()* method. When we save a new entity, the *save()* method generates an id and sets it on the entity passed.

For this reason, we have to mock the void method to simulate the various processing outcomes.

Another time the mocking might come in handy is when testing exceptions thrown by the void method.

### ****4. How to Mock a***void***Method****

Now, let’s see how we can mock a void method using EasyMock.

Let’s suppose, we have to mock the void method of a *WeatherService* class that takes a location and sets the minimum and maximum temperature:

**public** **interface** **WeatherService** {

**void** **populateTemperature**(Location location);

}

#### ****4.1. Creating the Mock Object****

Let’s start by creating a mock for the *WeatherService*:

@Mock

**private** WeatherService mockWeatherService;

Here, we’ve done this using the EasyMock annotation *@Mock*. But, we can do this using the *EasyMock.mock()* method as well.

Next, we’ll record the expected interactions with the mock by calling *populateTemperature()*:

mockWeatherService.populateTemperature(EasyMock.anyObject(Location.class));

Now, if we don’t want to simulate the processing of this method, this call itself is sufficient to mock the method.

#### ****4.2. Throwing an Exception****

First, let’s take the case where we want to test whether our class can handle ****exceptions thrown by the void method****. For this, we’ll have to mock the method in such a way that it throws these exceptions.

In our example, the method throws *ServiceUnavailableException*:

EasyMock.expectLastCall().andThrow(**new** ServiceUnavailableException());

As seen above, this involves simply calling the *andThrow(Throwable)* method.

#### ****4.3. Simulating Method Behavior****

As mentioned earlier, we might sometimes need to ****simulate the behavior of the void method.****

In our case, this would involve populating the minimum and maximum temperatures of the locations passed:

EasyMock.expectLastCall()

.andAnswer(() -> {

Location passedLocation = (Location) EasyMock.getCurrentArguments()[0];

passedLocation.setMaximumTemparature(**new** BigDecimal(MAX\_TEMP));

passedLocation.setMinimumTemperature(**new** BigDecimal(MAX\_TEMP - 10));

**return** **null**;

});

Here, we’ve used the *andAnswer(IAnswer)* method to define the behavior of the *populateTemperature()* method when called. Then, we’ve used the *EasyMock.getCurrentArguments()* method – that returns the arguments passed to the mock method – to modify the locations passed.

Note that we have ****returned***null***at the end.**** This is because we are mocking a void method.

It’s also worth noting that this approach is not restricted to mocking void methods only. We can also use it for methods that return a value. There, it comes in handy when we want to mock the method to return values based on the arguments passed.

#### ****4.4. Replaying the Mocked Method****

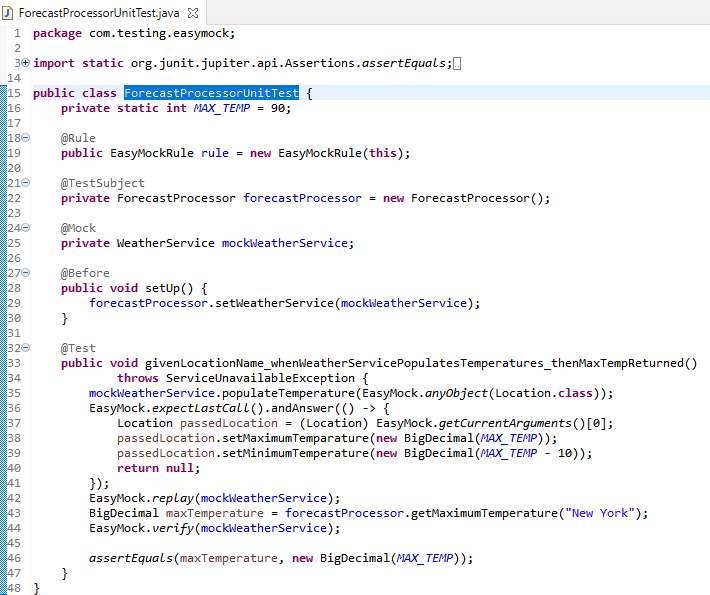
Lastly, we’ll use the *EasyMock.replay()* method to change the mock to “replay” mode, so that the recorded actions can be replayed when called:

EasyMock.replay(mockWeatherService);

Consequently, when we call the test method, the custom behavior defined should be executed.

### ****Instructions:****

* Create a simple maven project (without archetype).
* Pull the provided source files into the src/main/java folder.
* Edit the pom.xml as in the screenshot above.
* Add a test class for ForecastProcessor called ForecastProcessorUnitTest.
* Edit the test class as in the below screenshot.



* Run the test. It should be green.

### ****Conclusion****

In this lab, we saw how to mock void methods using EasyMock.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*