Verifying What Methods Are Called Using EasyMock

Nicolae Caprarescu
FULL-STACK SOFTWARE DEVELOPMENT CONSULTANT
www.properjava.com

Overview

Testing state vs. testing behavior

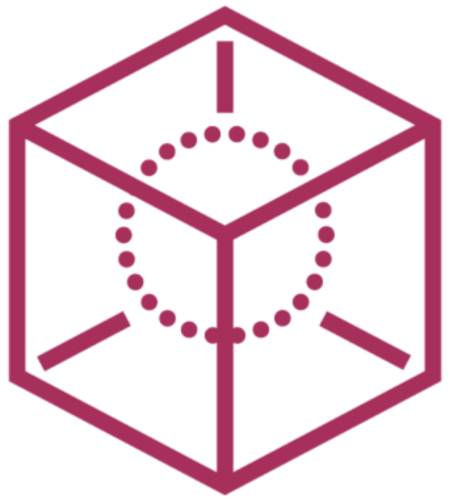
Verifying mocks

Verifying the unexpected

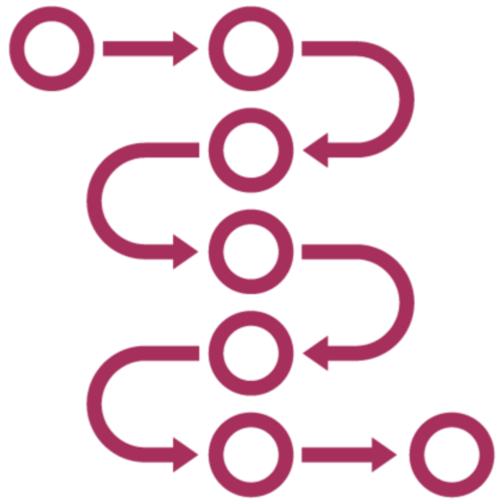
Method stubbing without verification

Verifying order of calls

Testing State vs. Testing Behavior



State-based testing



Behavior-based testing
(the focus of this course)

Testing State

Verifying that the unit-under-test returns the correct result

You examine the state of the unit-under-test once the functionality has been exercised

Unit testing phases: setup, exercise, verify, teardown

Classical, Detroit

Typical State-testing Pseudocode

```
collab = new Collaborator()

uut = new UnitUnderTest(collab)

result = uut.exercise()

assertSomething(result)

teardown()
```

- Setup: non-mock collaborator object(s)
- Setup: unit-under-test
- Exercise
- Verify
- Teardown: doesn’t have to be explicit

Testing Behavior

Verifying that the unit-under-test calls certain methods correctly

The checks are also carried out at verification stage. Often used in combination with state-based testing

Mocks always use this

Mockist, London

Typical Behavior-testing Pseudocode

```
mockCollab = mock(Collaborator.class)

mockCollab.expectation()

uut = new UnitUnderTest(mockCollab)

result = uut.exercise()

verify(mockCollab)
teardown()
```

- Setup: mock collaborator object(s)
- Setup: expectations
- Setup: unit-under-test
- Exercise
- Verify
- Teardown: doesn’t have to be explicit

Testing Behavior with EasyMock

The verify(…) method checks that the expected behavior has actually materialized during the actual test execution

If the expected behavior of a mock wasn't used during the actual test's execution, a test **without** verify(…) won't catch this

A strict mock will catch any actual behavior we didn't expect well before the call to verify(…) though, but that's the opposite of catching the lack of the actual behavior that was expected

Demo

Let's add verification to our previous tests

Relaxing Method Call Counts

The times(…) method can be used to specify the expected number of calls of a mocked method

Omitting this, as we've done so far, implies exactly one expected call

Additional methods include: atLeastOnce() and anyTimes()

Demo

Relaxing method call expectations

Verifying the Unexpected?

The opposite of verifying that **expected** behavior has **actually** materialized during test execution: verifying that no **unexpected** behavior happened during the **actual** test execution

As seen during demos, strict mocks offer this out-of-the-box

Nice mocks don't, and you'd need to explicitly check it yourself

Should I Be Verifying the Unexpected?

My advice: do this on a very exceptional basis, only when the context really requires it

Why? It's obviously impossible to list all the unexpected things to explicitly verify for anyway

In addition, in state-based testing, you generally verify that the state had the correct value change for a given property: you don't verify all other properties. The same point can be made here, for behavior: you care about some behavior, not about all possible other behaviors.

So What Happens if I Verify the Unexpected?

It may seem like you’re adding credibility to your tests

In reality, you’re just making your tests overspecified

Which makes them less readable and more difficult to maintain

Verifying the Unexpected with EasyMock

Nice mocks

Must be carried out explicitly

EasyMock’s design makes it difficult to implement this, but it’s possible

Strict mocks

Already carried out implicitly because exceptions are thrown by mocks for any unexpected calls

Demo

Verifying the unexpected

Method Stubbing without Verification

Occasionally, we want to configure expectations so that mocks respond to some method calls, but we <u>don’t want to verify</u> them against the actual test execution

expect(…) + and<u>Stub</u>Return(…) instead of expect(…) + andReturn(…)

andStubThrow(…), andStubAnswer(…)

```
expect(mock.mockedMethod(
                    …params…
         )).andReturn(result);
```

◀ Configuring a mocked method expectation in the normal way

```
expect(mock.mockedMethod(
                    …params…
         )).andStubReturn(result);
```

◀ Configuring a mocked method expectation when you don't care about verifying it

Verifying Order of Method Calls

On a single mock:

- Simple
- Just need to create a mock of type MockType.STRICT, either using EasyMock.strictMock(...) or @Mock(MockType.STRICT)

Among multiple mocks:

- More complicated
- So we'll cover it in a demo

Demo

Verifying order of calls among multiple mocks

Summary

Compared behavior-based testing, the focus of this course, with state-based testing

Demonstrated how to verify that the expected behavior actually happened during test execution

Demonstrated how to relax method expectations

Discussed and demonstrated verifying the unexpected

Demonstrated how to stub methods when we don't care about verifying them

Finally, we demonstrated how to verify order of calls