Mock Objects

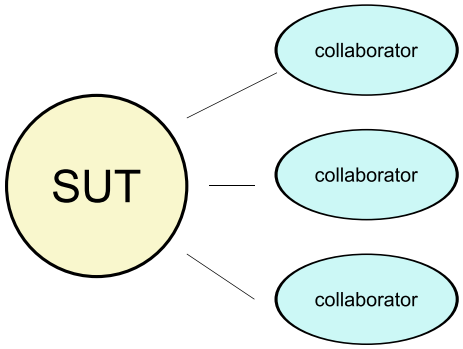
- What are Test Doubles?
- Mock Object Fundamentals

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

System Under Test vs Colloborators

- During unit testing, test-case methods typically verify a single class or object under test (SUT)
- However, many SUTs require other objects with which to collaborate so we can verify correctness



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

State Verification vs Behavioral Verification

- In **state verification** you create a fixture, exercise the SUT, then verify that the fixture's state is as expected
- In **behavior verification**, you create a fixture, exercise the SUT, then verify that the SUT interacted with collaborators as expected

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Test Doubles

- Sometimes it's useful to use a stand-in for an actual collaborator
- Gerard Meszaros refers to these stand-ins as **test doubles** and defines these categories:

Dummy	Dummy objects are passed around but never actually used. Usually they are just used to fill parameter lists.
Fake	Fake objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).
Stub	Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.
Mock	Mocks are pre-programmed with expectations which form a specification of the calls they are expected to receive

---

Gerard Meszaros' book is xUnit Test Patterns: Refactoring Test Code (Addison-Wesley, 2007).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Do We Need Test Doubles?

- The real collaborator runs too slow to allow a test suite to finish quickly
- The real collaborator does not yet exist
- Building the fixture with a real collaborator is tedious and/or time consuming
- Performing exception-handling checking with a real collaborator is difficult or dangerous
- You desire additional instrumentation of the collaborator during testing

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What Are Mock Objects?

- Mocks are different from the other types of test doubles because they allow for **behavior verification**
- In other words, you configure mocks with the expected behavior, then exercise the SUT and verify that the expected colloborations actually occurred
- You can write mock classes by hand or use frameworks such as jMock or EasyMock



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to jMock

- From www.jmock.org: "JMock is a library that supports test-driven development of Java code with mock objects."
- As of this writing, jMock is at version 2.5.



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple jMock Example

```
import org.jmock.integration.junit4.JMock;
import org.jmock.integration.junit4.JUnit4Mockery;
import org.jmock.Expectations;

@RunWith(JMock.class)
class PublisherTest {
    Mockery context = new JUnit4Mockery();

    @Test
    public void oneSubscriberReceivesAMessage() {
        // set up
        final Subscriber subscriber = context.mock(Subscriber.class);

        Publisher publisher = new Publisher();
        publisher.add(subscriber);

        final String message = "message";

        // expectations
        context.checking(new Expectations() {{
            oneOf (subscriber).receive(message);
        }});

        // execute
        publisher.publish(message);
    }
}
```

Create mock

configure expected behavior

exercise SUT

end of method runs verification

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to EasyMock

- From www.easymock.org: "EasyMock provides Mock Objects for interfaces in JUnit tests by generating them on the fly using Java's proxy mechanism. Due to EasyMock's unique style of recording expectations, most refactorings will not affect the Mock Objects. So EasyMock is a perfect fit for Test-Driven Development"
- As of this writing, EasyMock is at version 2.4.



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned

- Why test doubles are necessary
- The differences between types of test doubles
- The fundamentals of mock objects

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum