Software Construction

Isolated Testing

Ch. Denzler / M. Kropp
University of Applied Sciences Northwestern Switzerland
Institute for Mobile and Distributed Systems

Learning Target

You

- can explain the importance of isolated testing
- can describe the concepts behind mock testing
- can use mock objects as an efficient way to do Unit Testing

Content

- Introduction to Mock Testing
- Introduction to Mockito

Unit Testing (Reminder!)

- Unit testing is done
  - on each module
  - in isolation
  - to verify the units behavior
- Unit test will
  - establish some kind of artificial environment
  - invoke methods/procedures in the module under test
  - check the results against known/expected values

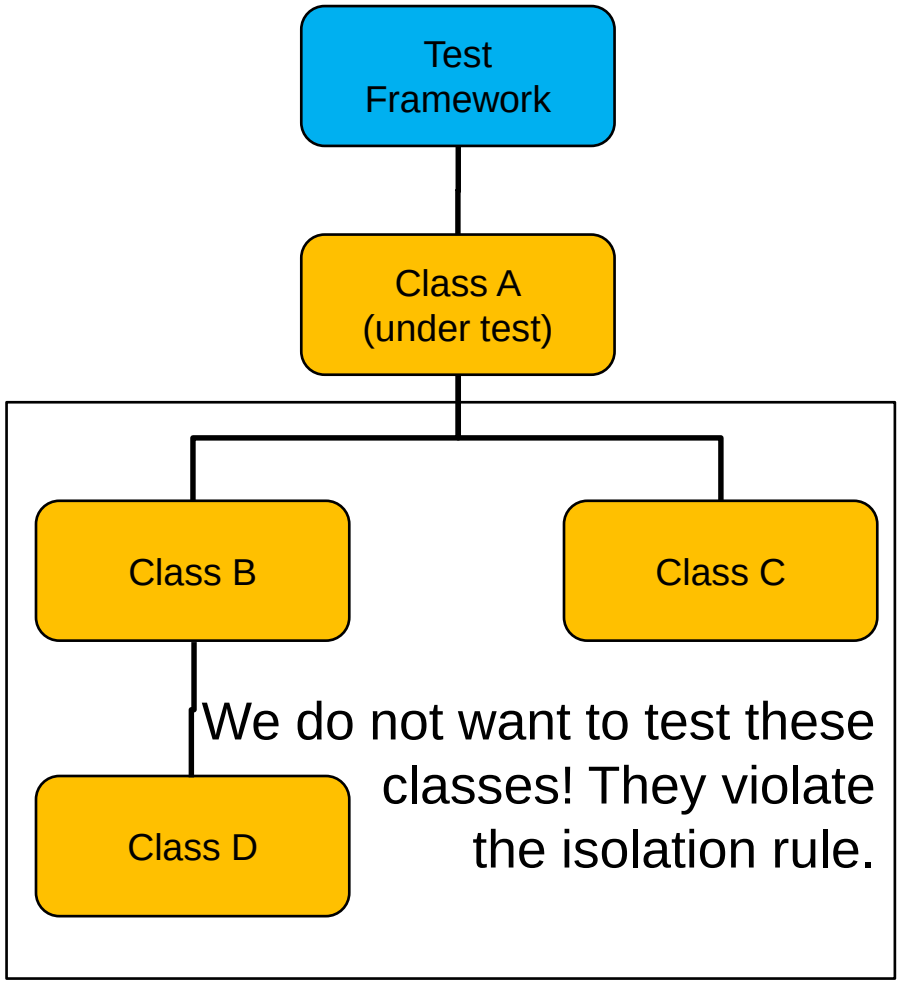
Limits of Unit Testing

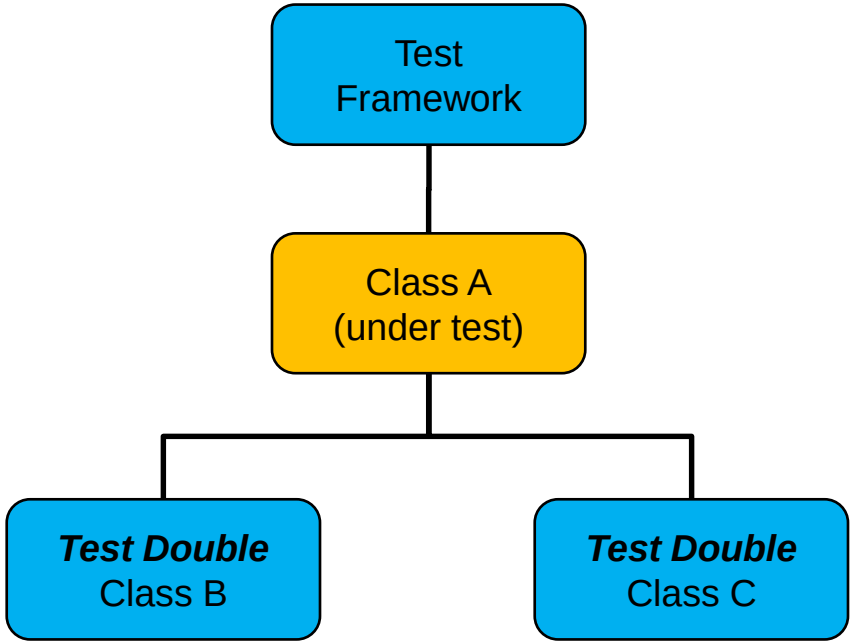
- Some things are difficult to test in isolation
  - Configuration
  - Database access
  - Network access

In general:

How to test a class that depends on other components?

Test Isolation





Test Double

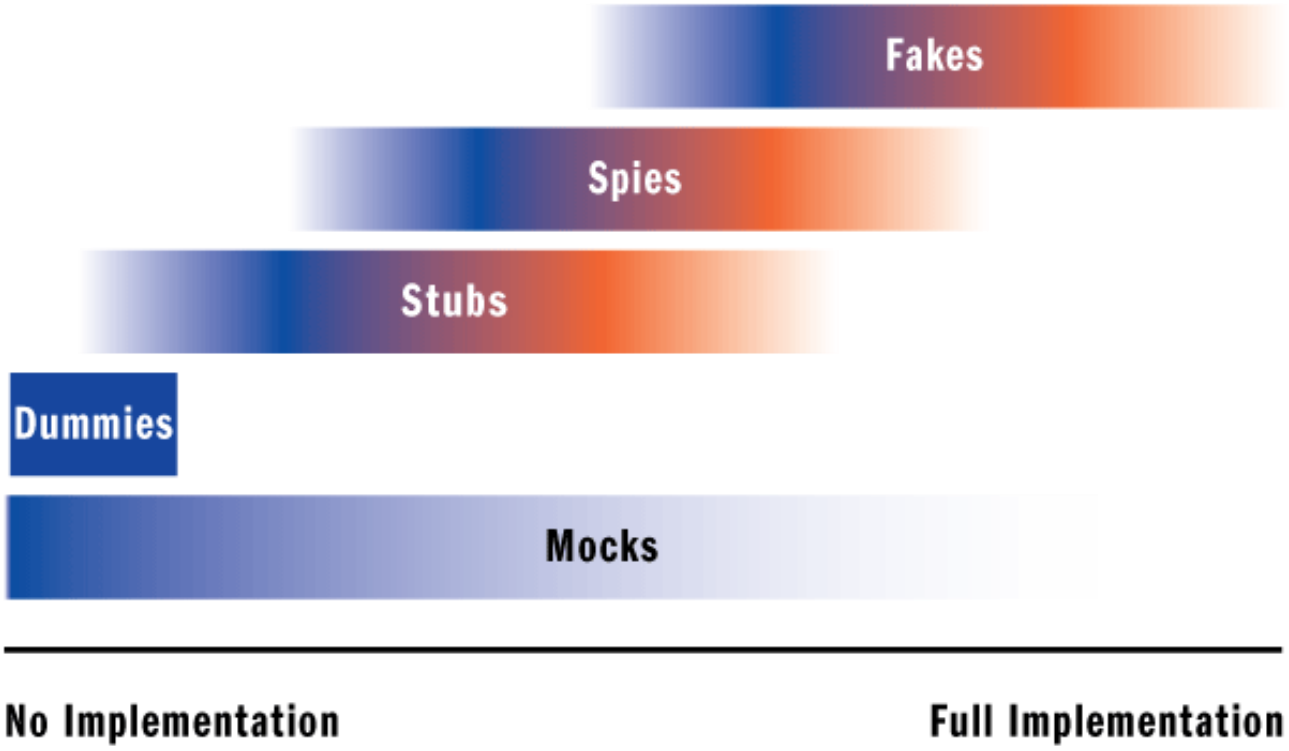
A Test Double is any object or component that is installed in place of the real component for the express purpose of running a test.

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

Test Doubles in Unit Testing

- **Dummy** objects are passed around but never actually used. Usually they are just used to fill parameter lists.
- **Stubs** are minimal implementations of interfaces or base classes. Methods returning void will typically contain no implementation at all, while methods returning values will typically return hard-coded values.
- **Spies** similar to a stub, but a spy will also record which members were invoked so that unit tests can verify that members were invoked as expected.
- **Fakes** contain more complex implementations, typically handling interactions between different members of the type it's inheriting.
- **Mocks** objects pre-programmed with expectations about the methods that will be invoked. The expected arguments for a call and the resulting return values or exceptions

Test Doubles in Unit Testing (1)



Exploring The Continuum Of Test Doubles by Mark Seemann
http://msdn.microsoft.com/en-us/magazine/cc163358.aspx

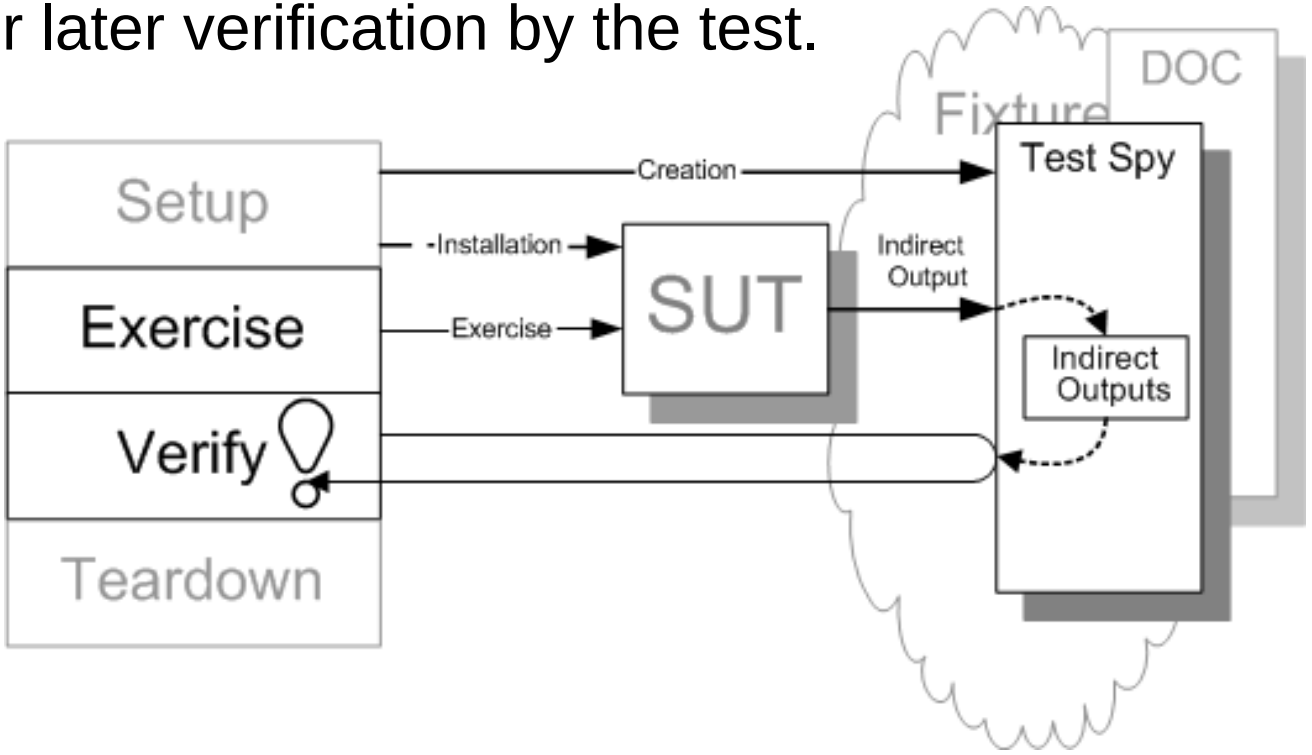
Test Doubles - Stubs

- Stub is a dumb object!
- A Stub is just an object that returns the same value over and over again.
- Allows implementation of tests without production code.
- For example:

```
class EmailStub implements EmailService{
  void sendMail(String adr; String Text) {;}
}
```

Test Doubles – Spies

Use a Test Double to capture the indirect output calls made to another component by the system under test (SUT) for later verification by the test.



- http://xunitpatterns.com/Test%20Spy.html

Test Doubles – Spies

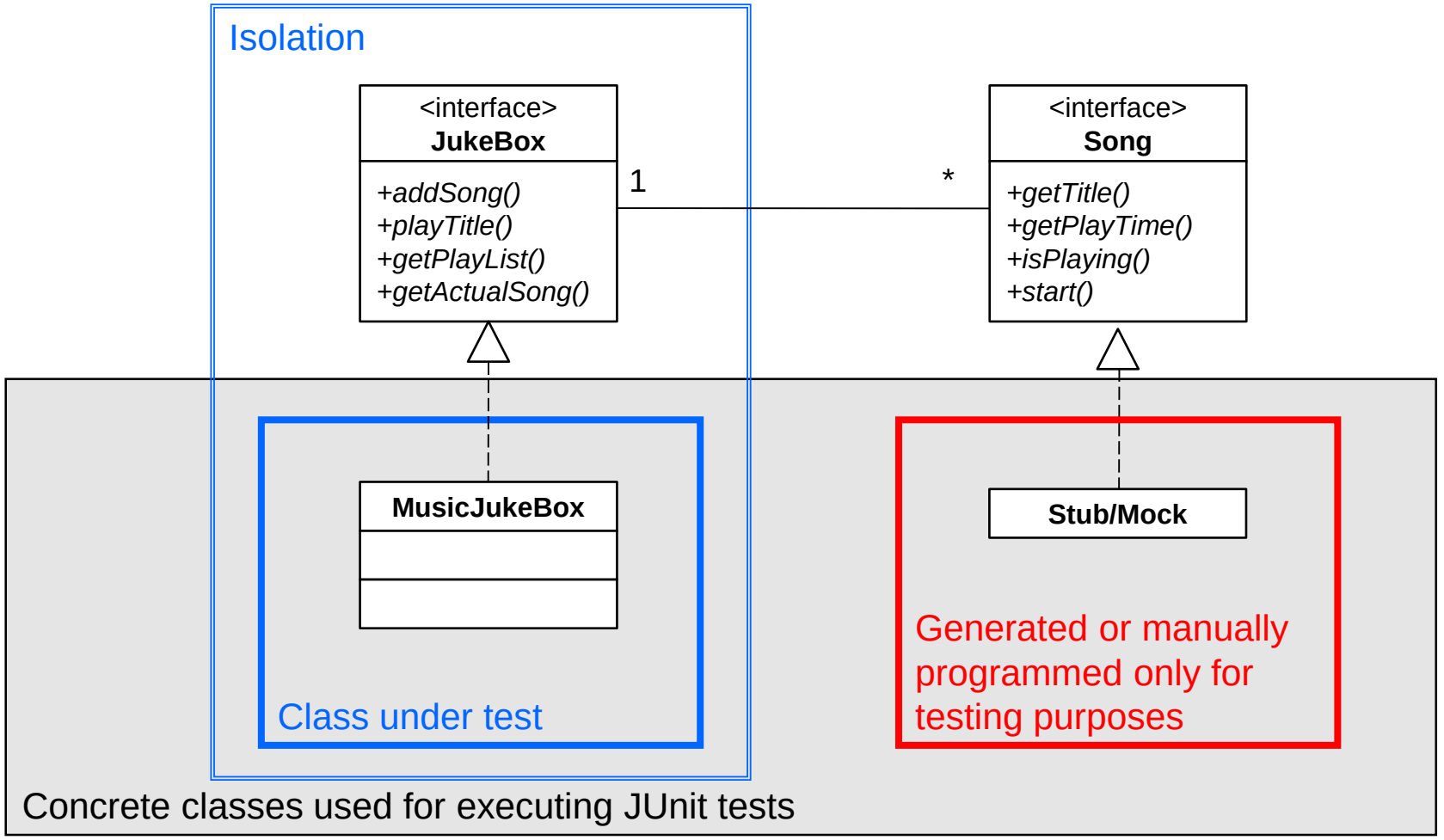
@Spy Annotation – Example (Mockito)

```
@Spy
List<string> listSpy = new ArrayList<string>();

@Test
public void testSpyReturnsRealValues() throws Exception {
 String s = "dobie";
 listSpy.add(new String(s));

 verify(listSpy).add(s);
 assertEquals(1, listSpy.size());
}
</string></string>
```

Test Doubles - Fakes

- Light weight objects only used for testing and not suitable for production!
- They tend to have much more functionality than standard test doubles.
- For Example:
  - In-memory databases
  - Fake Service Layer

Sample application



Exercise: Testing with Stub Objects

Mock Testing

- Mock objects simulate parts of the behaviour of domain objects.
- Classes can be tested in isolation by simulating their collaborators with mock objects.
- Takes classes out of a production environment and puts them in a well defined lab environment.

When To Use Mock Objects

- The real object has non-deterministic behaviour.
- The real object is difficult to set up.
- The real object is slow.
- The real object has a user interface.
- The test needs to ask the real object about how it was used (-> if callback function was called)
- The real object does not yet exist.

from "Pragmatic Unit Testing"

Pros and Cons of Mock Objects

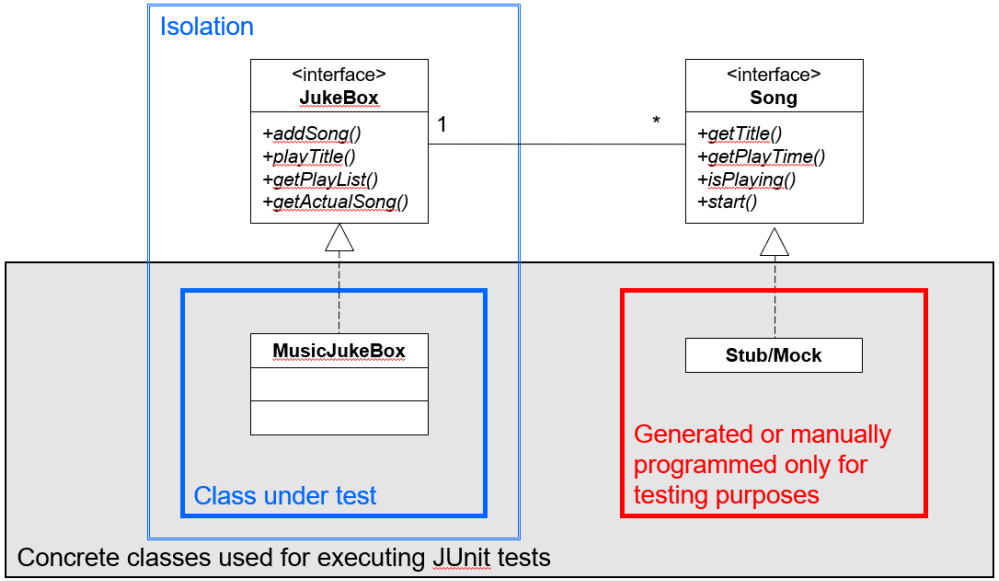
- Pros
  - Emphasize that testing is about isolation
  - Simplifies handling of interfaces with many methods
  - Can enable near-instant testing even of code that uses resource-bound APIs such as JDBC
- Cons
  - Can mirror the implementation too closely, making a test suite fragile
  - Mocking can become complex with APIs like JDBC

Mock Framework - Mockito

- Writing and maintaining mock objects often is a tedious task that may be error prone.
- Mockito is a library that provides an easy way to create and use mock objects.
- Mockito generates mock objects dynamically - no need to write them, and no generated code
- http://site.mockito.org/

Using Mockito

- Remember our sample application
  - We want to test the getActualSong() method of the class MusicBox
- Isolation
  - We need to “mock” the interface Song and emulate its behaviour



Using Mockito

```
import static org.mockito.Mockito.*;
JukeBox box;

@Before void setup() { box = new MusicJukeBox();

@Test void testActualSong() {
  Song mockSong = mock(Song.class);
  when(mockSong.getTitle()).thenReturn("Frozen");
  when(mockSong.isPlaying()).thenReturn(Boolean.TRUE);

  // run the actual tests
  box.addSong(mockSong);
  box.playTitle("Frozen");

  assertEquals("Frozen", box.getActualSong().getTitle());

  verify(mockSong).start();
  verify(mockSong, times(2)).getTitle();
}
```

static import

Create mock object

Specify return value by wrapping call with when() method

Use mock object as normal object

Normal unit tests here

Verfiy behavior of mock object

Did getTitle() get called twice?

The Mocking Usage Pattern

1. **Create** a mock object for the interface or class we would like to simulate
2. **Specify** the expected behavior of the mock
3. **Use** the mock in standard unit testing, as if it was a normal object
4. **Verify** behavior

Create Mock

- Use method `mock(<Class or Interface>)`
- Mock object can now be used

```
import static org.mockito.Mockito.*;

/*
 * create a mock
 */
Song mockSong = mock(Song.class);
```

Specify Method Behavior

- Mock object “learns” how to react on specific input.

```
/*
 * specify return value by wrapping call
 * with when() method and using thenReturn
 * method to provide an actual return value.
 */
when(mockSong.getTitle()).thenReturn("My Song Title");
```

Use Mock Object

- The mock object can be used immediately after its creation.
- Method behavior specification and use of mock object can be in any order (i.e. no need specify everything before first use)

```
/*
 * execute tests
 */
box.addSong(mockSong);
box.playTitle("Frozen");

assertEquals("Frozen", box.getActualSong().getTitle());
```

Verify Behavior

- If we specify behavior, we would like to verify that it is actually used.
- To verify that the specified behavior has been used, we have to call `verify`

```
/*
 * verfiy behavior
 */
verify(mockSong).start();
verify(mockSong, times(2)).getTitle();
```

Mockito Benefits

- No hand-writing of classes for mock objects needed.
- Supports refactoring-safe mock objects: test code will not break at runtime when renaming methods or reordering method parameters
- Supports return values and exceptions.
- Supports order-checking of method calls, for one or more mock objects.
- Supports checking of the number of actual calls of the mock objects

Further Concepts

- Mocks
  - doThrow(<exception object>).when(<mock object>).methodCall()
  - doAnswer(Answer answer) to specify code, that must be called when a method is executed
  - reset() reset the mock object for reuse.
- Argument matchers
  - Uses Hamcrest matchers (from JUnit libs)
  - anyInt, anyString, eq…
- Counting invocations
  - times(count), atLeastOnce(), atLeast(min), atMost(max), never()

Ressources

- Mockito
  http://mockito.org
- Paper "Mocks Aren't Stubs"
  http://www.martinfowler.com/articles/mocksArentStubs.html
- Mockito Refcard
  http://refcardz.dzone.com/refcardz/mockito
- Chapter 6: Using Mock Objects,
  in *Pragmatic Unit Testing*

Exercise: Testing with Mocks