Software-Architecture — Mock Objects and Mockito

Prof. Dr. Axel Böttcher

3. Oktober 2011
Objectives

- Understand the concept of mocks, stubs and spies
- know when and why to use mock objects
- Apply the mockito library for testing (after lab session).
Consider this sequence diagram for an automated Movie Rental Station (Videothek):
Here, the RentalFrontend-class checks,

1. if a movie may be rented by a given customer (where the simple rule applies: „no restriction“ or „adult-only“).

2. if a copy is available up to the intended date.

```java
import java.util.Date;

public class RentalFrontend {

    private ReservationService reservationService;

    public RentalFrontend(ReservationService reservationService) {
        this.reservationService = reservationService;
    }

    public boolean isRentable(Movie movie, Customer customer, Date date) {
        return movie.isRentableTo(customer) &&
               reservationService.isRentableUntil(movie, date);
    }
}
```
First Step: Test for Movie

The method `isRentableTo` depends on a customer. If we create a concrete `Customer`, we face two main problems:

1. it could be complicated to create a `Customer` object, because there might exist more dependencies
2. the test would depend on the implementation of `isAdult`, which could be a complicated method of its own (depending on the date of birth and the current date etc.).
Mocking

This is where mocking comes into play. Simple implementations can be generated automatically, if there exists an interface (inverted dependencies!):

<table>
<thead>
<tr>
<th>Customer</th>
<th>isAdult(): boolean</th>
</tr>
</thead>
</table>

<<interface>>

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAdult(): boolean</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CustomerImpl</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAdult(): boolean</td>
</tr>
</tbody>
</table>
Libraries for Mocking

There are several alternatives in the Java universe:

- Mockito: www.mockito.org → This will be used here
- EasyMock: www.easymock.org
- Jmock: www.jmock.org

In the literature there is no common understanding of what a mock and a stub is, see e.g. http://xunitpatterns.com/Mocks,Fakes,StubsandDummies.html
Automated Mock Testing with Mockito

- automatically creates mocks from interfaces
- mocks used to verify behavior (correct calls, correct parameters, sufficient number of calls)

Creation of mock objects is simple:

```java
1 import static org.mockito.Mockito.*;
2 Customer customerMock = mock(Customer.class);
```

Afterwards the mock is used in the test. It will record all method calls; they can be checked afterwards.
isRentableTo(customer)
Specifying Return Values (stubbing)

Mocked methods with non-void return type must return a value when called.

```java
when(customerMock.isAdult()).thenReturn(true);
```
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;

public class MovieTest {
    private Movie adultMovie = new MovieImpl("The Silence of the Lambs", true);
    private Customer customerMock;

    @Before
    public void setUp() {
        customerMock = mock(Customer.class);
    }

    @Test
    public void testIsRentableToAdultCustomer() {
        when(customerMock.isAdult()).thenReturn(true);
        assertTrue("Movie " + adultMovie + " must be rentable to adult customer",
                    adultMovie.isRentableTo(customer));
    }
}
Verification of method calls

Stubbing has to be done upfront. It does not verify that the stubbed methods are called. However, your application code should break if not. Sometimes you might want to verify that a method was called afterwards with `verify`:

```java
1  verify(customerMock.isAdult());
```

Calls the `equals`-method on all parameters (if any)!

Test for the RentalFrontEnd

```java
public class RentalFrontEndTest {
    private RentalFrontend rentalFrontend;
    private ReservationService reservationService = mock(ReservationService.class);
    private Movie movie = mock(Movie.class);
    private final Customer customer = mock(Customer.class);
    private Date date = new Date();

    @Test
    public void testSuccessfulReservation() {
        when(movie.isRentableTo(customer)).thenReturn(true);
        when(reservationService.isRentableUntil(movie, date)).thenReturn(true);
        assertTrue("Must be rentable...", rentalFrontend.isRentable(movie, customer, date));
    }
}
```
RentalFrontendTest

RentalFrontend

Movie

Customer

ReservationService

isRentableTo(customer)

isRentableUntil(Movie, Date)

true

true

true

MOCK

MOCK

MOCK

isRentable(movie, date)

isAdult()

isRentable(movie, date)
More Mockito tools

- use InOrder objects to check order between calls.
How does this work?

A proxy mechanism is part of the reflection API: Class `Proxy` provides the following static method:

```java
1  static Object newProxyInstance(ClassLoader loader,
2       Class[] interfaces, InvocationHandler h)
```

Returns an instance of a proxy class for the specified interfaces that dispatches method invocations to the specified invocation handler. `InvocationHandler` is a pretty simple interface:

```java
1  public interface InvocationHandler {
2      Object invoke(Object proxy, Method method, Object[]
3            }
```

Processes a method invocation on a proxy instance and returns the result.

At Mockito’s core you find this generic method:

```java
1  public static <T> T mock(Class<T> classToMock)
```