MT AG

business by integration

Testen mit Pfefferminzgeschmack

Birgit Kratz

DOAG 2012, Nürnberg, 22.11.2012

MT AG

Daten und Fakten

Als anerkannter Partner der führenden Technologiehersteller verbinden wir die Agilität eines mittelständischen Unternehmens mit der Lösungskompetenz internationaler Beratungshäuser.

Hauptsitz
Ratingen

Niederlassungen
Hamburg, Dortmund, Frankfurt, Luxemburg

Tochtergesellschaften
MT-ifs GmbH, MT-ics GmbH

Beschäftigte (2012)
220 Festangestellte
80 Freiberufler

MT AG

Kurzvorstellung

Wer will Ihnen denn hier was erzählen?

- Birgit Kratz
- Seit 1997 in der IT tätig als Beraterin
- Seit 2009 für die MT AG
- Angefangen mit C, C++
- Später Java, J2EE, PHP, HTML, CSS, …..
- Leidenschaft entdeckt für TDD und Clean Code Prinzipien

MT AG

Agenda

1. Problemstellung
2. Mock Objekte
3. Mockito – kurz & gut
4. Erstellung von Mock Objekten
5. „DI“ mit Mockito
6. Stubbing
7. Argument Matcher
8. Verifizieren
9. Spionieren (Spying)
10. Aufrufparameter aufzeichnen
11. Limitations

Problemstellung

- (Unit-)Tests schreiben JA, aber…
  - Sehr (zeit)aufwändig
  - Sehr komplex, umständlich
  - Abhängigkeiten zu anderen Systemen/Klassen

- Unit-Tests sollen kleine Einheiten (Klassen/Methoden) isoliert von ihrer Umgebung testen
  - Wie erstelle ich diese Isolation, wenn es diese ganzen Abhängigkeiten zu anderen Systemen/Klassen gibt?
  - Die „echten“ Umgebungssysteme und –klassen sollen im Test nicht verwendet werden, da sonst keine Isolation gewährleistet ist.
  - Lösung: „Nachbau“ der Umgebung – YOU ARE KIDDING!

Mock Objekte

- Sind „Dummies“, „Test-Doubles“, die an die Stelle der Systeme/Klassen treten, von der meine zu testende Einheit abhängig ist
- Sie repräsentieren eine definierte Situation, ein definiertes Verhalten
- Sie implementieren die Schnittstelle, über die die zu testende Einheit auf sie zugreift
- Sie stellen sicher, dass Methodenaufrufe mit den geforderten Parametern möglich sind, liefern aber keine „echten“ Daten zurück, sondern zur Testsituation passende Daten
- Bieten damit die Möglichkeit Testszenarien abzubilden
- ABER DAS BEDEUTET JA – für jedes Szenario eventuell ein neues Mock Objekt schreiben zu müssen
- PJUUHHHHHH

Mockito – kurz & gut

„Mockito is a mocking framework that tastes really good. It lets you write beautiful tests with clean & simple API. Mockito doesn't give you hangover because the tests are very readable and they produce clean verification errors." (www.mockito.org)

- wurde inspiriert von EasyMock => leichter Umstieg
- hat eine intuitive API => leichter Einstieg
- hat aber, im Gegensatz zu EasyMock, keinen record/play Modus
- 2 Regeln:
  - Stubbing kommt vor der Ausführung
  - Verifizierung von Interaktionen kommt nach der Ausführung

Mockito – kurz & gut (cont.)

- Features:
  - Leichte Erstellung von Mocks
  - Annotationen
  - Integration mit anderen TestRunnern (z.B. Spring)
  - Methoden-Stubbing, auch void-Methoden und mehrfache Aufrufe derselben Methode
  - Argument Matching
  - Verifizierung von Verhalten
  - Verifizierung von Anzahl und Reihenfolge von Aufrufen
  - Verifizierung von Aufrufen mit bestimmten und unbestimmten Parametern
  - Spying realer Objekte
  - u.v.m

MT AG

Erstellung von Mock Objekten

Ohne Annotationen

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

public class Testklasse {
    MyClass mockedMyClass = mock(MyClass.class);

    @Test
    public void testMethode() {
        ...
    }
}
```

Erstellung von Mock Objekten

Mit Annotationen (1)

```
@RunWith (MockitoJUnitRunner.class)
public class Testklasse {
    @Mock
    MyClass mockedMyClass;

    @Test
    public void testMehtode () {
        ...
    }
}
```

Erstellung von Mock Objekten

Mit Annotationen (2)

```
@RunWith (SpringJUnit4ClassRunner.class)
public class Testklasse {
    @Mock
    MyClass mockedMyClass;

    @Before
    public void init() {
        MockitoAnnotations.initMocks(this);
    }
}
```

„DI“ mit Mockito

```java
public class MyClass {
    @Inject
    private SomeService service;

    public String doSomething() {
        ...
    }
}
```

```java
@RunWith (MockitoJUnitRunner.class)
public class MyClassTest {
    @Mock (name = "service")
    private SomeService serviceMock;

    @InjectMocks
    private MyClass classUnderTest;

    @Test
    public void testDoSomething() {
        ...
    }
```

- Mockito wird nun versuchen @InjectMocks zu instanziieren mit Hilfe von
  1. Constructor injection
  2. Property setter injection (Setter-Methoden)
  3. Field injection
- Im vorliegenden Beispiel wird Field-Injection verwendet.

MT AG

Stubbing

- Legt fest, wie das Mock-Objekt im jeweiligen Test-Kontext reagieren soll
- Einfaches Stubbing:

```
// wenn mock.someMethod() aufgerufen wird, soll das Mock-Objekt
// den Wert 10 zurück liefern.
when(service.someMethod()).thenReturn(10);
```

- Stubbing von void-Methoden mit Exceptions:

```
// wenn mock.someVoidMethod() aufgerufen wird,
// soll eine RuntimeException geworfen werden.
doThrow(new RuntimeException()).when(service).someVoidMethod();
```

- Iterator-Style stubbing:

```
// Der erste Aufruf von mock.someMethod() liefert 1 zurück,
// der zweite 2
// und der dritte Aufruf sowie alle weiteren wirft eine
// RuntimeException (das letzte Stubbing gilt).
when(service.someMethod())
    .thenReturn(1)
    .thenReturn(2)
    .thenThrow(new RuntimeException());
```

noch mehr Stubbing

- Stubbing mit Callbacks (erlaubt Stubbing mit einem generischen Answer interface)

```
when(mock.someMethod(anyString())).thenAnswer(new Answer() {
    Object answer(InvocationOnMock invocation) {
    Object[] args = invocation.getArguments();
    Object mock = invocation.getMock();
        return "called with arguments: " + args;
    }
});
```

- doReturn(), doAnswer(), doNothing(), doCallRealMethod()

```
doThrow(new RuntimeException()).when(mockedList).clear();
```

Argument Matcher

- erlauben flexibles Stubbing und Verifizieren
- es müssen keine realen Werte an die Stub-Methoden übergeben werden
- für (fast) jede Situation gibt es schon fertige Argument Matcher

any(), any(java.lang.Class<T> clazz), anyBoolean(),
anyByte(), anyChar(), anyCollection(), anyCollectionOf(java.lang.Class<T> clazz),
anyDouble(), anyFloat(), anyInt(), anyList(), anyListOf(java.lang.Class<T> clazz),
anyLong(), anyMap(), anyMapOf(java.lang.Class<T> clazz), anyObject(),
anySet(), anySetOf(java.lang.Class<T> clazz), anyShort(),
anyString(), anyVararg()

anyClass(java.lang.Class<T> clazz)

- zusätzlich kann man eigene Matcher mit Hilfe von Hamcrest Matchern erstellen.

MT AG

Verifizieren

- ein Mock Objekt „merkt“ sich alle seine Interaktionen
- dadurch können diese Interaktionen nach der Ausführung der zu testenden Methode(n) verifiziert werden

```
// überprüft, dass mock.someMethod() (genau einmal) aufgerufen wurde
verify(mock).someMethod();
```

- Verifizierung von (max, min) Anzahl von Aufrufen, oder ob eine Aufruf überhaupt erfolgte bzw. erfolgen durfte (nicht gewollte Aufrufe)

```
verify(mock, times(2)).someMethod();     // genau 2 Mal
verify(mock, atLeast(2)).someMethod();   // mindestens 2 Mal
verify(mock, atMost(5)).someMethod();    // nicht mehr als 5 Mal
verify(mock, never()).someMethod();      // niemals, Alias für times(0)
verifyZeroInteractions(mock);            // keinerlei Aufruf am Mock-Objekt
```

Verifizieren

- Auswertung der Reihenfolge von Aufrufen

```
// Mock Objekt dessen Methoden in einer gewissen Reihenfolge
//aufgerufen werden müssen
List listMock = mock(List.class);

listMock.add("was added first");
listMock.add("was added second");

//InOrder verifier
InOrder inOrder = inOrder(listMock);

// Überprüfen dass add zuerst mit "was added first"
// und danach mit "was added second" aufgerufen wurde.
inOrder.verify(singleMock).add("was added first");
inOrder.verify(singleMock).add("was added second");
```

- funktioniert sowohl für einzelne als auch für mehrere Mocks

```
//InOrder verifier für mehrere Mocks
InOrder inOrder = inOrder(mock1, mock2);
```

Spionieren (Spying)

- Spying ermöglicht, ein Objekt nur teilweise zu mocken („partial mocking“)
- Methoden, die einen Stub haben, liefern vordefinierte Ergebnisse
- Bei allen anderen Methoden werden die Originale verwendet
- Erzeugung eines Spy-Objektes:
  - Implizite Verwendung des Default Konstruktor

```
@Spy
Bar spyOnBar;
```

  - expliziter Aufruf eines anderen Konstruktors

```
@Spy
Foo spyOnFoo = new Foo("argument");
```

- Verwendung der doReturn() - Stubbing Familie angeraten

Aufrufparameter Aufzeichnen

- Möglichkeit zum Aufzeichnen von Aufrufparametern zur späteren Auswertung mit Hilfe von „ArgumentCaptors“
  - Beispiel ohne Annotationen

```
ArgumentCaptor<Person> personCaptor = ArgumentCaptor.forClass(Person.class);
verify(mock).doSomething(personCaptor.capture());
assertEquals("Baghira", personCaptor.getValue().getName());
```

  - Beispiel mit Annotationen

```
@Captor
ArgumentCaptor<Person> personCaptor;

@Test
public void shouldDoSomethingUseful() {
    //...
    verify(mock, times(2)).doStuff(personCaptor.capture());
    assertEquals("Peppels", personCaptor.get(0).getName());
    assertEquals("Baghira", personCaptor.get(1).getName());
}
```

Limitierungen

- Mockito hat einige wenige Limitierungen, die hauptsächlich technische Natur sind
- Die Mockito Autoren glauben jedoch, dass die Verwendung von „Hacks“ zur Umgehung dieser Limitierungen förderlich für das Schreiben von schlecht testbarem Code sind.
- Mockito kann folgendes nicht mocken:
  - Konstruktoren
  - finale Klassen
  - Enums
  - finale, statische und private Methoden
  - hashCode() und equals()
- Ausweg: PowerMock oder JMockit

Q & A

MT AG

Besuchen Sie auch unsere weiteren Vorträge auf der DOAG 2012

Dienstag, 12 Uhr, Raum Riga	Dynamisch Unterschiede in Datensätzen auf Feldebene finden by S.O. Kelbert
Dienstag, 13 Uhr, Raum Seoul	Route to ASM by Ernst Leber
Dienstag, 14 Uhr, Raum Stockholm	Automatische Generierung der ETL-Prozesse OWB vs. ODI by Irina Gotlibovych
Dienstag, 15 Uhr, Raum Kopenhagen	Wiederverwendung von bestehendem PL/SQL Code in ADF Anwendungen by Hendrik Gossens
Dienstag, 16 Uhr, Raum Stockholm	„Managed Code“ mit OWB – Methoden und Wege by Bernhard Rosenberger
Mittwoch, 13 Uhr, Raum Riga	Dateizugriff mit new I/O 2 by Wolfgang Nast
Mittwoch, 15 Uhr, Raum Riga	WebServices in Java SE und EE by Wolfgang Nast
Mittwoch, 16 Uhr, Raum Seoul	Das Mysterium OPatch by Volker Mach
Donnerstag, 09 Uhr, Raum Istanbul	Das größte APEX Projekt der Welt @ Union Investment by Niels de Bruijn
Donnerstag, 14 Uhr, Raum Konf. EG	Testen mit Pfefferminzgeschmack by Birgit Kratz
Donnerstag, 15 Uhr, Raum Istanbul	APEX goes UNIT Testing by Oliver Lemm
Donnerstag, 16 Uhr, Raum Oslo	SOA verspielt – rekursive BPEL Prozesse by Guido Neander

MT AG

MT AG

business by integration

Vielen Dank.

MT AG

Balcke-Dürr-Allee 9
40882 Ratingen

Telefon: +49 (0) 21 02 309 61-0
Telefax: +49 (0) 21 02 309 61-10

E-Mail: info@mt-ag.com
www.mt-ag.com