Testen mit JMockIt

Dipl.-Informationswirt André Janus
André Janus - IT Consulting

JUG Karlsruhe

jug-ka.de | twitter.com/@jugka

Agenda

- Testen & Unit-Tests
- Mock-Objekte
- JMockIt
  - Core
  - Annotations
  - Expectations

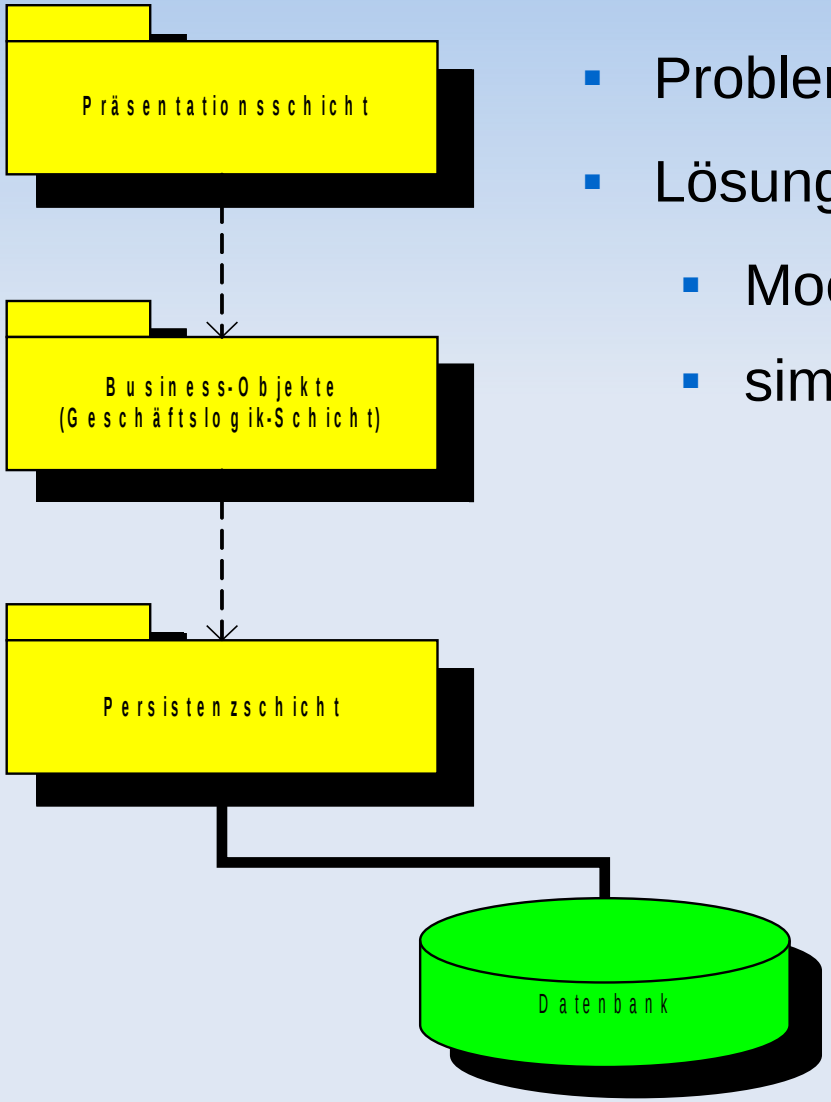
Warum überhaupt testen?

Unit-Test vs. Integrations-Test

- Unit-Test
  - Funktionalität einer Code-Unit (isoliert vom Gesamtsystem!)
  - Unit = Komponente, Klasse, Methode, ...
  - Test: erwartete_Ausgabe ?= Unit(Eingabe)

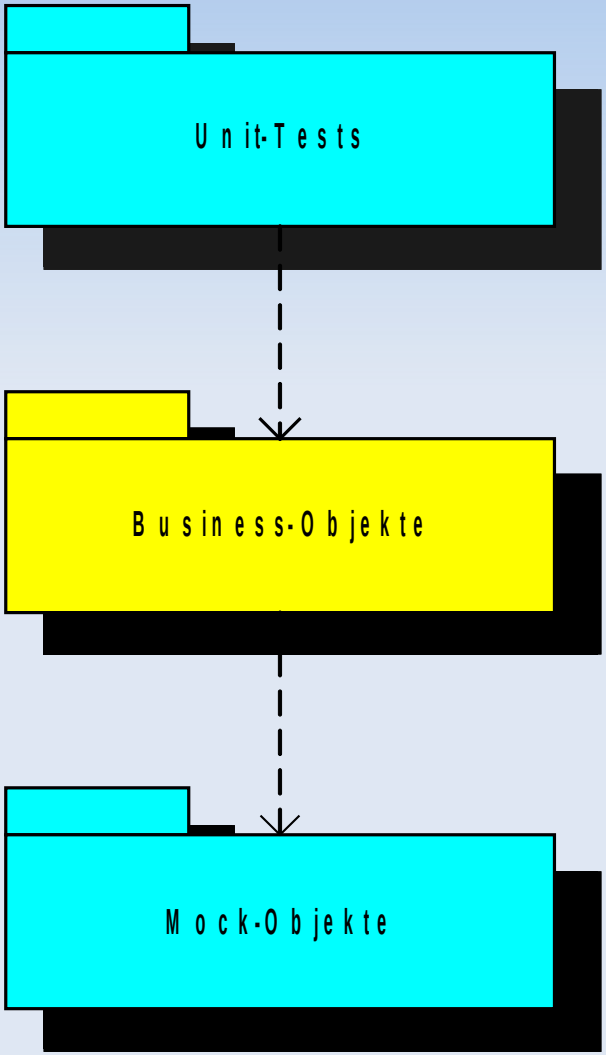
- Integrationstest
  - Gesamtfunktionalität von gekoppelten Komponenten
  - Gesamtsysteme oder abgeschlossene Subsysteme

Probleme bei der Testentwicklung



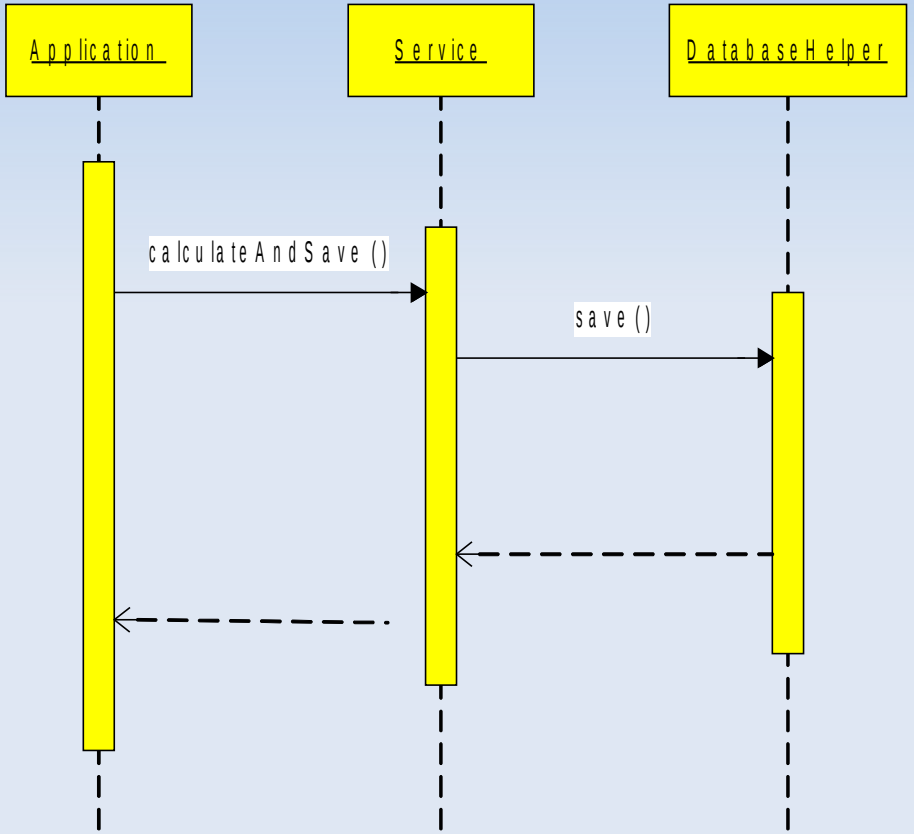
- Problem: Abhängigkeit
- Lösung: Mock-Objekte
  - Mock = Attrappe, Nachahmung, Fälschung
  - simulieren abhängige Komponenten

Verwendung von Mock-Objekten



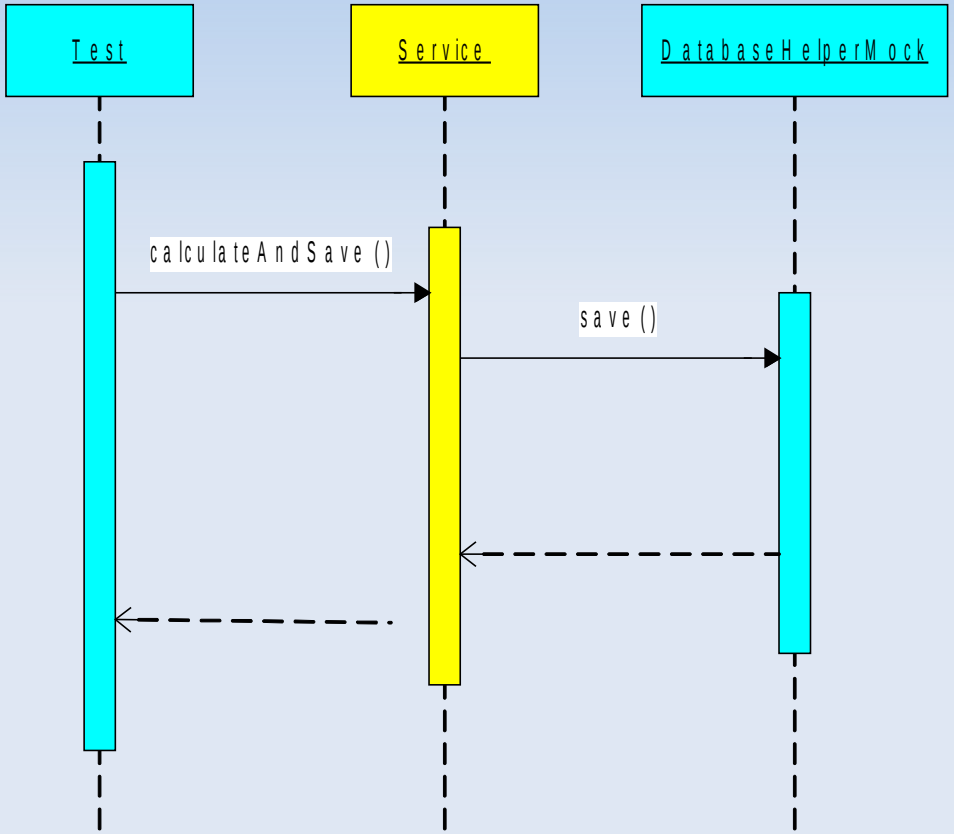
- Mock-Objekte erzeugen
  - Manuell
  - Über Framework
- Probleme:
  - Kapselung
  - Dependency Injection ?
  - Testbarkeit beeinflußt Design

Beispiel: Test eines Service



- Zu mocken: DatabaseHelper.save()
- DatabaseHelper
  - Initialisierung im Konstruktor des Service
  - kein Setter

JMockIt Core



- Redefine
  - *redefineMethods(Clazz.class, Mock.class)*
- Restore
  - *restoreAllOriginalDefinitions()*
  - *restoreOriginalDefinition(Clazz.class)*
- Sukzessive Ersetzung
- DataBaseHelperMock
  - Kein Interface
  - Keine Vererbung

JMockIt Core

```
public class ServiceTest
{
    public void testCalculateAndSave()
    {
        Mockit.redefineMethods(DatabaseHelper.class, DatabaseHelperMock.class);

        int input1 = 6;
        int input2 = 7;

        assertEquals(42, Service.getInstance().calculateAndSave(input1, input2));
    }

    public class DatabaseHelperMock
    {
        public boolean save()
        {
            return true;
        }
    }
```

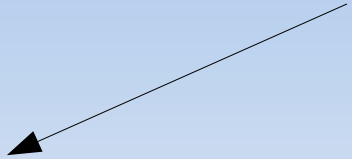
JMockIt Annotations

```
public class ServiceTest
{
    public void testCalculateAndSave()
    {
        Mockit.setUpMocks(DatabaseHelperMock.class);

        int input1 = 6;
        int input2 = 7;

        assertEquals(42, Service.getInstance().calculateAndSave(input1, input2));
    }

    @MockClass(realClass = DatabaseHelper.class)
    public class DatabaseHelperMock
    {
        @Mock(invocations = 1)
        public boolean save()
        {
            return true;
        }
  }
```

Verifizierung von Methodenaufrufen bzw. deren Anzahl

JMockIt Expectations

```
public void testCalculation()
{
    new Expectations(true) {
        protected DatabaseHelper helper;
        {
            helper.getCalculation(6,7);
            returns(42);
        }
    };

    assertEquals(42, Service.getInstance().getCalculation(6,7));
}
```



“Aufnahme/Wiedergabe”

weitere Informationen

- Weitere JMockIt Features
  - JMockIt Coverage
  - JMockIt Hibernate Emulation
  - NEU: JMockIt Verifications

- Entwicklung im Rahmen von Google Code (früher: Java.net)
- Aktuelle Version: 0.998
- Open Source (MIT License)

Links und Referenzen

- http://jmockit.dev.java.net/
- http://code.google.com/p/jmockit/
- http://de.wikipedia.org/wiki/MIT-Lizenz

Vergleich Mock- Frameworks

- EasyMock
  - Klasse und Mock: gemeinsames Interface = Design-Abhängigkeit
  - zu mockende Klasse wird per **Reflection** ersetzt
  - Nicht „mockbar“: statische, finale und private Methoden

- JMockIt
  - keine Design-Abhängigkeit!
  - zu mockende Klasse wird per **Instrumentation** ersetzt
  - VM-Parameter: *–javaagent:jmockit.jar*