# Behavior-Driven Development in Java mit JGiven

Dr. Jan Schäfer

TNG TECHNOLOGY CONSULTING

15. Juni 2016

KARLSRUHER ENTWICKLERTAG

# Warum BDD?

Pet Clinic A Spring Framework Demonstration

Spring

Home | Find owners | Veterinarians | Error

# Welcome

spring by Pivotal.

# Typischer JUnit-Test

```
@Test
public void shouldInsertPetIntoDatabaseAndGenerateId() {
    Owner owner6 = this.clinicService.findOwnerById(6);
    int found = owner6.getPets().size();

    Pet pet = new Pet();
    pet.setName("bowser");
    Collection<PetType> types = this.clinicService.findPetTypes();
    pet.setType(EntityUtils.getById(types, PetType.class, 2));
    pet.setBirthDate(new DateTime());
    owner6.addPet(pet);
    assertThat(owner6.getPets().size()).isEqualTo(found + 1);

    this.clinicService.savePet(pet);
    this.clinicService.saveOwner(owner6);

    owner6 = this.clinicService.findOwnerById(6);
    assertThat(owner6.getPets().size()).isEqualTo(found + 1);
    // checks that id has been generated
    assertThat(pet.getId()).isNotNull();
}
```

Beispiel von github.com/spring-projects/spring-petclinic

# Probleme von JUnit-Tests

- Viele technische Details
- Eigentliche Kern des Tests oft schwer zu erkennen
- Oft Code-Duplizierung
- **Können nur von Entwicklern gelesen werden**

# Verhaltensgetriebene Entwicklung (BDD)

- Verhalten wird in der **Fachsprache** der Anwendung beschrieben
- Entwickler und Fachexperten arbeiten **gemeinsam** an der Spezifikation
- Verhaltensspezifikationen sind **ausführbar**

# Beispiel

```
Scenario: Owners can be found by last name

    Given an owner with last name "Müller"
     When searching for "Müller"
     Then exactly the given owner is found
```

# BDD in Java


*Behaviour Driven Development with elegance and joy*

# Feature-Dateien (Gherkin)

findowners.feature

```
Feature: Finding Owners

  Scenario: Owners can be found by last name

    Given an owner with last name "Müller"
     When searching for "Müller"
     Then exactly the given owner is found
```

# Schritt-Implementierung (Java)

```java
public class CustomerStepdefs {
    @Given("an owner with last name (.*)")
    public void anOwnerWithLastName(String lastName) { ... }

    @When("searching for (.*)")
    public void searchingFor(String lastName) { ... }

    @Then("exactly the given owner is found")
    public void exactlyTheGivenOwnerIsFound() { ... }
}
```

# Probleme

- Feature-Dateien und Code müssen synchron gehalten werden
- Keine Programmiersprache in Feature-Dateien verwendbar
- Eingeschränkter IDE-Support (z.B. Refactoring)

**Hohe Wartungskosten**

# Beobachtungen aus der Praxis

- Niedrige Akzeptanz bei Entwicklern durch hohen Mehraufwand
- Nicht-Entwickler schreiben selten Feature-Dateien selbst
- Entwickler müssen Feature-Dateien warten

# Andere BDD-Frameworks?

- JBehave: Plain Text + Java (wie Cucumber)
- Concordion: HTML + Java
- Fitness: Wiki + Java
- Spock: Groovy
- ScalaTest: Scala
- Jnario: Xtend
- Serenity


Image: http://www.freeimages.com/photo/1207509

# Stack Overflow?

16

Behavior Driven Development is just a technique that can be used without any tools. You can just write tests in BDD style - e.g. start test methods with `should` and introduce some separate feature with this method. `When` and `then` sections can be replaced with just comments, e.g.

```
@Test
public void should_do_something() {
    // given
    Something something = getSomething();

    // when
    something.doSomething();
    // then
    assertSomething();

    // when
    something.doSomethingElse();
    // then
    assertSomethingElse();
}
```

I also think that the most successful BDD frameworks for Java are those that are not written in Java, since the Java language has no such flexibility for DSL (Domain Specific Language) creation that Groovy or Scala has.

and it can be very important for someone.

- scalatest (written with scala) and easyb (written with groovy) both have the same disadvantage as spock. The "... should ..." and "Given...Then" notation. Specifications are in .story files, and the step implementations are in Java classes. This approach work very well as a collaboration and communication tool to define the specs, but would usually be too much overhead for low-level coding.

- Cucumber is a widely-used tool from the Ruby world, which now has a JVM version that works in many languages. It uses a very similar "Given..When..Then" notation. Ports of Cucumber exist for .NET (SpecFlow), Python (Behave and Lettuce), PHP (Behat) and many more languages.

I also think that the most successful BDD frameworks for Java are those that are not written in Java, since the Java language has no such flexibility for DSL (Domain Specific Language) creation that Groovy or Scala has.

share edit flag

edited Feb 18 '14 at 6:25
Meredith
901 • 9 • 33

answered Apr 17 '13 at 9:56

sody
1,660 • 9 • 20

Quelle: http://stackoverflow.com/questions/16036120/ , Autor: http://stackoverflow.com/users/1371775/sody


JGiven

# Ziele

- Entwicklerfreundlich (geringer Wartungsaufwand)
- Lesbare Tests in Given-When-Then-Form (BDD)
- Einfache Wiederverwendung von Test-Code
- Lesbar für Fachexperten

# Demo

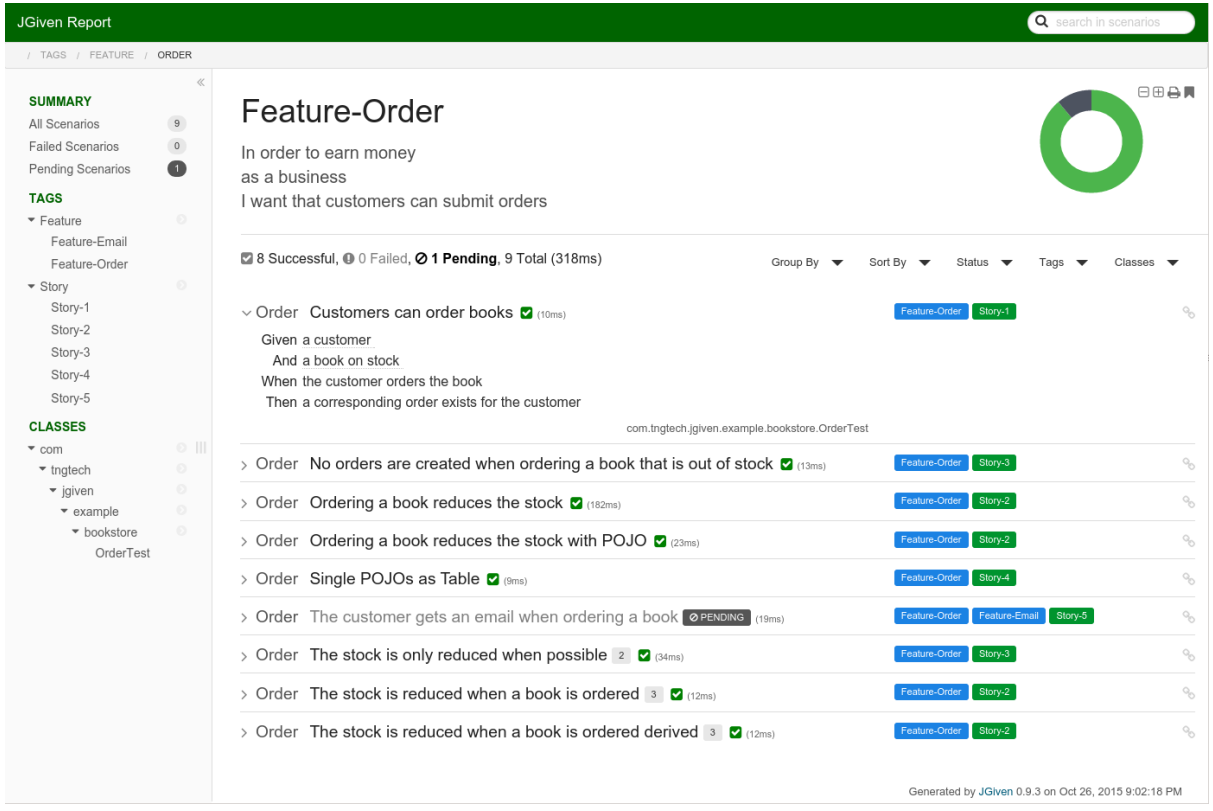
# Szenarien in JGiven

```
import org.junit.Test;
import com.tngtech.jgiven.junit.SimpleScenarioTest;

public class FindOwnerTest extends SimpleScenarioTest<StepDefs> {

    @Test
    public void owners_can_be_found_by_last_name() {

        given().an_owner_with_last_name("Müller");

        when().searching_for("Müller");

        then().exactly_the_given_owner_is_found();
    }
}
```

# Textausgabe

```
Scenario: owners can be found by last name

  Given an owner with last name "Müller"
   When searching for "Müller"
   Then exactly the given owner is found
```

# HTML5-App




- jgiven.org/jgiven-report/html5/ (Local)
- janschaefer.github.io/xke-jgiven/ (Local)
- janschaefer.github.io/etk16-example

# Klassisches BDD

Lesbarer Text ⟷ Code

# JGiven

Lesbarer Code ⟶ Lesbarer Bericht

# Erfahrungen aus der Praxis

- Über 2 Jahre Erfahrung aus einem großen Java-Projekt
- Über 3000 Szenarien
- Lesbarkeit und Wiederverwendung von Test-Code stark verbessert
- Wartungskosten von automatisierten Tests **veringert** (keine harten Zahlen)
- Entwickler und Fachexperten arbeiten **gemeinsam** an Szenarien
- Große Akzeptanz bei Entwicklern
- Leicht von neuen Entwicklern zu erlernen

# Erste Schritte

# JGiven-Abhängigkeit

- com.tngtech.jgiven:jgiven-junit:0.11.4
- oder com.tngtech.jgiven:jgiven-testng:0.11.4
- Lizenz: Apache License v2.0

# ScenarioTest* erweitern

```
import com.tngtech.jgiven.(junit|testng).ScenarioTest;

public class SomeScenarioTest extends ScenarioTest<...> {

}
```

*Oder SimpleScenarioTest

# Stage-Typen hinzufügen

```
import com.tngtech.jgiven.junit.ScenarioTest;

public class SomeScenarioTest
    extends ScenarioTest<MyGivenStage, MyWhenStage, MyThenStage> {

}
```

# Test-Methoden hinzufügen

```
import org.junit.Test;
import com.tngtech.jgiven.junit.ScenarioTest;

public class SomeScenarioTest
    extends ScenarioTest<MyGivenStage, MyWhenStage, MyThenStage> {

    @Test
    public void my_first_scenario() { ... }

}
```

# Schritt-Methoden schreiben

```
import org.junit.Test;
import com.tngtech.jgiven.junit.ScenarioTest;

public class SomeScenarioTest
    extends ScenarioTest<MyGivenStage, MyWhenStage, MyThenStage> {

    @Test
    public void my_first_scenario() {

        given().some_initial_state();
        when().some_action();
        then().the_result_is_correct();

    }
}
```

# Stage-Klassen schreiben

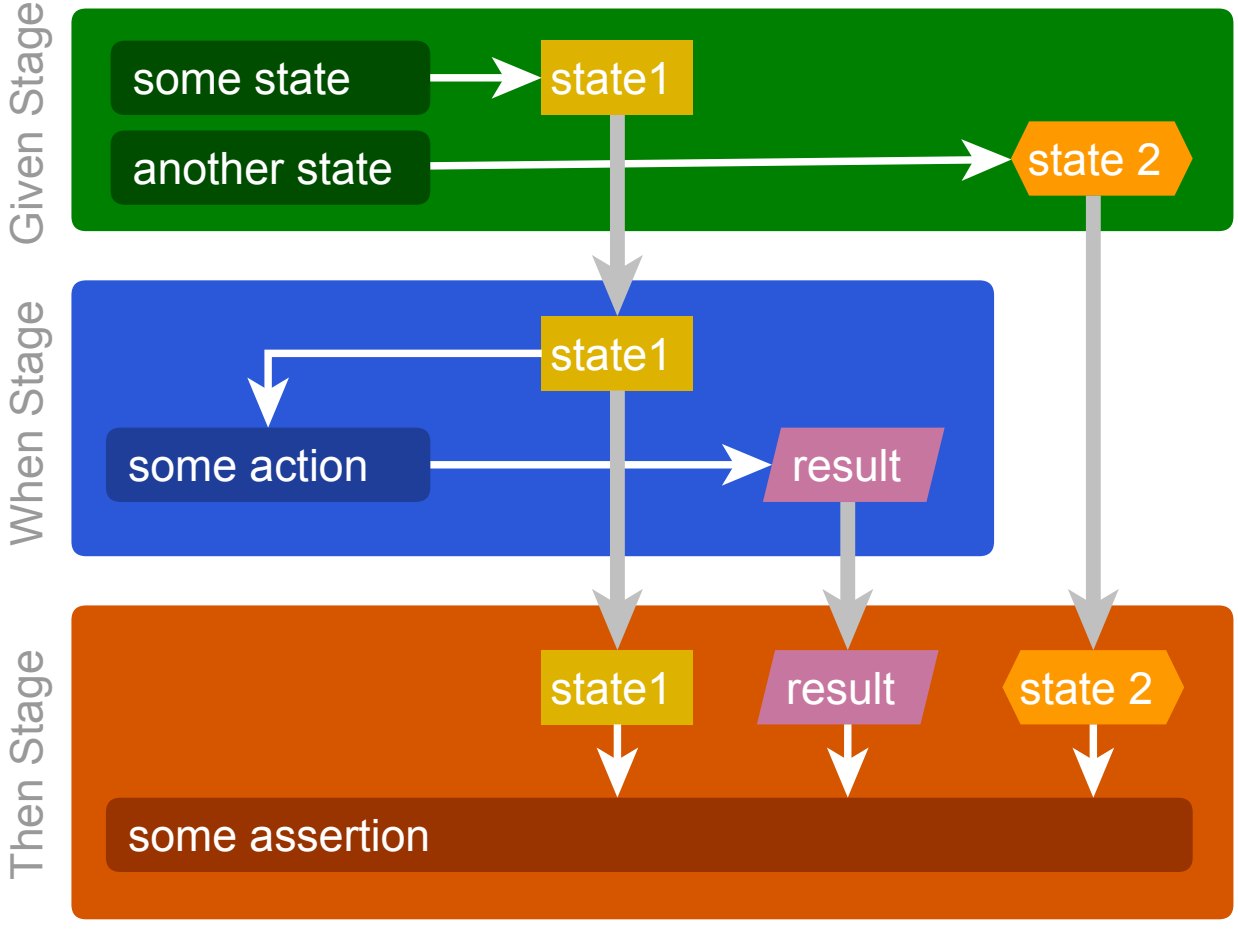
```
import com.tngtech.jgiven.Stage;

public class MyGivenStage extends Stage<MyGivenStage> {

    int state;

    public MyGivenStage some_initial_state() {
        state = 42;
        return this;
    }
}
```

# Stage-Klassen

# Allgemein

- JGiven-Szenarien werden aus **Stage-Klassen** zusammengesetzt
- Stage-Klassen ermöglichen **Modularität** und **Wiederverwendung**

# Zustandstransfer



# Zustandstransfer

- Felder von Stage-Klassen werden mit **@ScenarioState** annotiert
- Werte werden zwischen Stages gelesen und geschrieben
- **@ProvidedScenarioState**, **@ExpectedScenarioState** als Alternative

```
public class MyGivenStage extends Stage<MyGivenStage> {
   @ProvidedScenarioState
   int state;

   public MyGivenStage some_initial_state() {
      state = 42;
      return self();
   }
}

public class MyWhenStage extends Stage<MyWhenStage> {
   @ExpectedScenarioState
   int state;

   @ProvidedScenarioState
   int result;

   public MyWhenStage some_action() {
       result = state * state;
   }
}
```

# Datengetriebene Szenarien

# Parameterisierte Schrittmethoden

```
given().a_customer_with_name( "John" );
```

## Bericht

```
Given a customer with name John
```

# Mitten im Satz?

```
Given there are 5 coffees left
```

## $ to the rescue!

```
given().there_are_$_coffees_left( 5 );
```

# Parameterisierte Szenarien

```
@Test
@DataProvider({
    "1, 0",
    "3, 2",
    "5, 4"})
public void the_stock_is_reduced_when_a_book_is_ordered( int initial,
                                                         int left ) {
    given().a_customer()
        .and().a_book()
        .with().$_items_on_stock( initial );

    when().the_customer_orders_the_book();

    then().there_are_$_items_left_on_stock( left );
}
```

Verwendet den DataProviderRunner (github.com/TNG/junit-dataprovider).
Parameterized Runner und Theories von JUnit sind auch unterstützt.

# Parameterisierte Szenarien

## Textausgabe

```
Scenario: the stock is reduced when a book is ordered

  Given a customer
    And a book
   With <initial> items on stock
   When the customer orders the book
   Then there are <left> items left on stock

 Cases:

   | # | initial | left | Status  |
   +---+---------+------+---------+
   | 1 |       1 |    0 | Success |
   | 2 |       3 |    2 | Success |
   | 3 |       5 |    4 | Success |
```

# Parameterisierte Szenarien

## HTML-Bericht

The stock is reduced when a book is ordered 3 ✅ (0.004s)

Given a customer
And a book
With **<initial>** items on stock
When the customer orders the book
Then there are **<left>** items left on stock

Cases

| # | initial | left | Status |
|---|---|---|---|
| 1 | 1 | 0 | ✅ |
| 2 | 3 | 2 | ✅ |
| 3 | 5 | 4 | ✅ |

# Abgeleitete Parameter

```
@Test
@DataProvider({"1", "3", "5"})
public void the_stock_is_reduced_when_a_book_is_ordered( int initial ) {

    given().a_customer()
           .and().a_book()
           .with().$_items_on_stock( initial );

    when().the_customer_orders_the_book();

    then().there_are_$_items_left_on_stock( initial - 1 );

}
```

# Abgeleitete Parameter

## Textausgabe

```
Scenario: the stock is reduced when a book is ordered

  Given a customer
    And a book
   With <initial> items on stock
   When the customer orders the book
   Then there are <numberOfItems> items left on stock

 Cases:

   | # | initial | numberOfItems | Status  |
   +---+---------+---------------+---------+
   | 1 |       1 |             0 | Success |
   | 2 |       3 |             2 | Success |
   | 3 |       5 |             4 | Success |
```

# Abgeleitete Parameter

## HTML-Bericht

The stock is reduced when a book is ordered 2 3 ✅ (0.005s)

Given a customer
And a book
With **<initial>** items on stock
When the customer orders the book
Then there are **<numberOfItems>** items left on stock

Cases

| # | initial | numberOfItems | Status |
|---|---|---|---|
| 1 | 1 | 0 | ✅ |
| 2 | 3 | 2 | ✅ |
| 3 | 5 | 4 | ✅ |

# Verschiedene Schritte

```
@Test
@DataProvider({ "3, 2, true",
                "0, 0, false" })
public void the_stock_is_only_reduced_when_possible(
          int initial, int left, boolean orderExists) {

    given().a_customer()
        .and().a_book()
        .with().$_items_on_stock( initial );

    when().the_customer_orders_the_book();

    if ( orderExists ) {
        then().a_corresponding_order_exists_for_the_customer();
    } else {
        then().no_corresponding_order_exists_for_the_customer();
    }
}
```

# Verschiedene Schritte

## Textausgabe

```
Scenario: the stock is only reduced when possible

  Case 1: initial = 3, left = 2, orderExists = true
   Given a customer
     And a book
    With 3 items on stock
    When the customer orders the book
    Then there are 2 items left on stock
     And a corresponding order exists for the customer

  Case 2: initial = 0, left = 0, orderExists = false
   Given a customer
     And a book
    With 0 items on stock
    When the customer orders the book
    Then there are 0 items left on stock
     And no corresponding order exists for the customer
```

# Verschiedene Schritte

## HTML-Bericht

The stock is only reduced when possible 2 ✅ (0.011s)

Case 1: initial = 3, left = 2, orderIsDone = true

Given a customer
And a book
With **3** items on stock
When the customer orders the book
Then there are **2** items left on stock
And **a corresponding order exists for the customer**

Case 2: initial = 0, left = 0, orderIsDone = false

Given a customer
And a book
With **0** items on stock
When the customer orders the book
Then there are **0** items left on stock
And **no corresponding order exists for the customer**

# Weitere Features

# Parameter-Formatierung

- Default: toString()
- @Format( MyCustomFormatter.class )
- @Formatf( " -- %s -- " )
- @MyCustomFormatAnnotation

# Beispiel

## @OnOff

```
@Format( value = BooleanFormatter.class, args = { "on", "off" } )
@Retention( RetentionPolicy.RUNTIME )
@interface OnOff {}
```

## Auf Parameter anwenden

```
public SELF the_machine_is_$( @OnOff boolean onOrOff ) { ... }
```

## Schritt benutzen

```
given().the_machine_is_$( false );
```

## Bericht

```
Given the machine is off
```

# Tabellen als Parameter

- @Table um einen Parameter als Tabelle zu markieren
- Muss der letzte Parameter sein
- Muss ein Iterable of Iterable, ein Iterable of POJOs, oder ein POJO sein

# Tabellen als Parameter

## Arrays

```
SELF the_following_books_are_on_stock( @Table String[][] stockTable ) {
   ...
}
```

- Die erste Zeile ist der Tabellenkopf

# Tabellen als Parameter

## Arrays

```
@Test
public void ordering_a_book_reduces_the_stock() {

    given().the_following_books_on_stock(new String[][]{
        {"id", "name", "author", "stock"},
        {"1", "The Hitchhiker's Guide to the Galaxy", "Douglas Adams", "5"},
        {"2", "Lord of the Rings", "John Tolkien", "3"},
    });

    when().a_customer_orders_book("1");

    then().the_stock_looks_as_follows(new String[][]{
        {"id", "name", "author", "stock"},
        {"1", "The Hitchhiker's Guide to the Galaxy", "Douglas Adams", "4"},
        {"2", "Lord of the Rings", "John Tolkien", "3"},
    });

}
```

# Tabellen als Parameter

## Textausgabe

```
Scenario: ordering a book reduces the stock

   Given the following books on stock

     | id | name                                    | author         | stock |
     +----+-----------------------------------------+----------------+-------+
     |  1 | The Hitchhiker's Guide to the Galaxy | Douglas Adams |     5 |
     |  2 | Lord of the Rings                       | John Tolkien  |     3 |

    When a customer orders book 1
    Then the stock looks as follows

     | id | name                                    | author         | stock |
     +----+-----------------------------------------+----------------+-------+
     |  1 | The Hitchhiker's Guide to the Galaxy | Douglas Adams |     4 |
     |  2 | Lord of the Rings                       | John Tolkien  |     3 |
```

# Tabellen als Parameter

## HTML-Bericht

Ordering a book reduces the stock ✅ (0.060s)

Given the following books are on stock

| id | name | author | stock |
|---|---|---|---|
| 1 | The Hitchhiker's Guide to the Galaxy | Douglas Adams | 5 |
| 2 | Lord of the Rings | John Tolkien | 3 |

(0.026s)

When a customer orders book 1

Then the stock looks as follows

| id | name | author | stock |
|---|---|---|---|
| 1 | The Hitchhiker's Guide to the Galaxy | Douglas Adams | 4 |
| 2 | Lord of the Rings | John Tolkien | 3 |

(0.022s)

# Tabellen als Parameter

## Liste von POJOs

- Feldnamen: Kopf
- Feldwerte: Daten

```
SELF the_following_books_are_on_stock( @Table List<BookOnStock> books) {
   ...
}
```

# Tabellen als Parameter

## Einfaches POJO

```
SELF the_following_book(
      @Table(includeFields = {"name", "author", "priceInEurCents"},
             header = VERTICAL) Book book) {
   ...
}
```

## HTML-Bericht

Single POJOs as Table ✅ (0.005s)

Given the following book

| | |
|---|---|
| **name** | Lord of the Rings |
| **author** | John Tolkien |
| **priceInEurCents** | 30 |

# @BeforeScenario und @AfterScenario

```
public class GivenSteps extends Stage<GivenSteps> {

    @ProvidedScenarioState
    File temporaryFolder;

    @BeforeScenario
    void setupTemporaryFolder() {
            temporaryFolder = ...
    }

    @AfterScenario
    void deleteTemporaryFolder() {
        temporaryFolder.delete();
    }
}
```

# @ScenarioRule

```
public class TemporaryFolderRule {
    File temporaryFolder;

    public void before() {
        temporaryFolder = ...
    }

    public void after() {
        temporaryFolder.delete();
    }
}

public class GivenSteps extends Stage<GivenSteps> {
    @ScenarioRule
    TemporaryFolderRule rule = new TemporaryFolderRule();
}
```

# @AfterStage, @BeforeStage

```
public class GivenCustomer extends Stage<GivenSteps> {
    CustomerBuilder builder;

    @ProvidedScenarioState
    Customer customer;

    public void a_customer() {
        builder = new CustomerBuilder();
    }

    public void with_age(int age) {
        builder.withAge(age);
    }

    @AfterStage
    void buildCustomer() {
        customer = builder.build();
    }
}
```

# Tags

```
@Test @FeatureEmail
void the_customer_gets_an_email_when_ordering_a_book() {
    ...
}
```

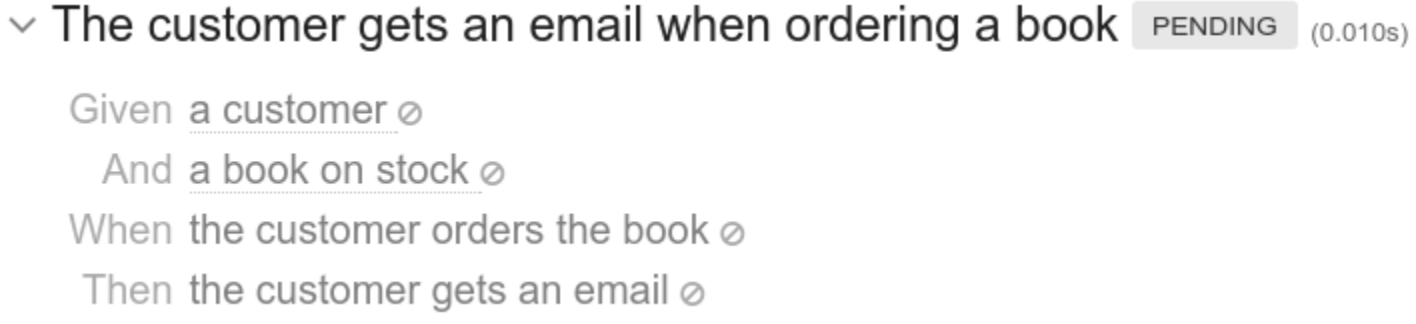
## Mit Werten

```
@Test @Story( "ABC-123" )
void the_customer_gets_an_email_when_ordering_a_book() { ... }
```

# @Pending

- Markiert den ganzen Test oder einzelne Schrittmethoden als noch nicht implementiert
- Schritte werden übersprungen und im Bericht entsprechend markiert

## HTML-Bericht



# @Hidden

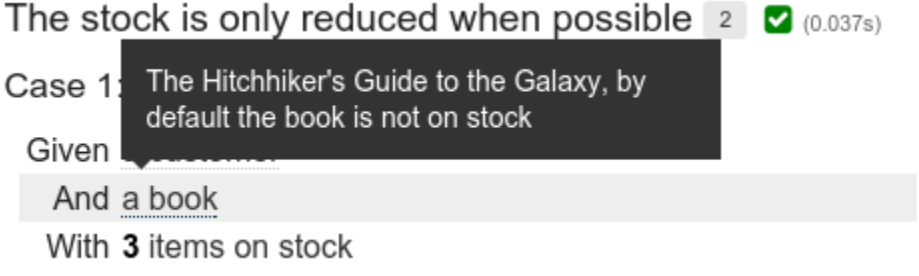
- Markiert Methoden die **nicht** im Bericht erscheinen sollen
- Sinnvoll für Methoden, die technisch gebraucht werden

```
@Hidden
public SELF doSomethingTechnical() { ... }
```

# Erweiterte Schrittbeschreibungen

```
@ExtendedDescription("The Hitchhiker's Guide to the Galaxy, "
                     + "by default the book is not on stock" )
public SELF a_book() { ... }
```

## HTML-Bericht



# Anhänge

```
public class Html5ReportStage {
    @ExpectedScenarioState
    protected CurrentStep currentStep; // provided by JGiven

    protected void takeScreenshot() {
        String base64 = ( (TakesScreenshot) webDriver )
            .getScreenshotAs( OutputType.BASE64 );
        currentStep.addAttachment(
            Attachment.fromBase64( base64, MediaType.PNG )
                      .withTitle( "Screenshot" ) );
    }
}
```

## HTML-Bericht

When the page of scenario 1 is opened
And scenario 1 is expanded (0.034s)
Then an attachment icon exists

# Zusammenfassung

# Vorteile

- ☑ Entwicklerfreundlich
- ☑ Hohe Modularität und Wiederverwendung von Test-Code
- ☑ Reines Java, keine weitere Sprache nötig
- ☑ Sehr leicht zu erlernen
- ☑ Sehr leicht in bestehende Test-Infrastrukturen zu integrieren
- ☑ Lesbare Berichte für Fachexperten

**BDD ohne den Zusatzaufwand!**

# Nachteile

- Fachexperten können keine JGiven-Szenarien schreiben
  - Aber Akzeptanzkriterien können leicht in JGiven-Szenarien übersetzt werden
  - Die generierten Berichte können von Fachexperten gelesen werden

# Danke!

@JanSchfr

jgiven.org

github.com/TNG/JGiven

janschaefer.github.io/etk16-slides

TNG TECHNOLOGY CONSULTING

# Backup

# Warum snake_case?

- Besser lesbar
  - thisCannotBeReadVeryEasilyBecauseItIsCamelCase
  - this_can_be_read_much_better_because_it_is_snake_case
- Wörter können in korrekter Schreibweise geschrieben werden
  - given().an_HTML_page()
- Berichtgenerierung funktioniert nur sinnvoll mit snake_case