

Praxisbericht:
Entwicklung einer
Testbibliothek mit
einem **Fluent Interface**

Marko Schulz

business@datenreisender.de

<http://datenreisender.de/>

Überblick

- Fluent Interfaces: Was ist das?
- Praxisbericht: Anwendung bei der Entwicklung einer Testbibliothek
- Schlussfolgerungen
- Diskussion!

Ein erstes Beispiel

```
Zeitspanne spanne = new Zeitspanne();  
spanne.setStartStunde(10);  
spanne.setStartMinute(15);  
spanne.setEndeStunde(11);  
spanne.setEndeMinute(45);
```

```
Zeitspanne spanne =  
    Zeitspanne.von("10:15").bis("11:45");
```

```
Zeitspanne spanne =  
    Zeitspanne.von(10, 15).bis(11, 45);
```

Was ist denn
„nicht fluent“?

Was ist ein „klassisches
Interface“?

set

get

do

```
vertrag.setHauptprodukt(produkt)
```

```
vertrag.getHauptprodukt()
```

```
vertrag.kündige(termin)
```

imperativ

objektorientiert-
imperativ

naheliegend?

Trennung von Funktionen und Prozeduren

mitunter gestelzt

mitunter langatmig

Fluent Interface

Eric Evans
+
Martin Fowler

Ein Bruch mit
set/get/do

Verwendung der
Objekte soll
„flüssiger“ sein

„natürlichsprachlicher“

interne DSLs

Fluent Interfaces
verschieben die
Sichtweise

Von „Was bietet ein
Objekt an?“

Zu „Wie arbeitet man
praktisch mit einem
Objekt?“

Beispiel 1

-

<http://martinfowler.com/bliki/FluentInterface.html>

```
private void makeNormal(Customer customer) {  
    Order o1 = new Order();  
    customer.addOrder(o1);  
    OrderLine line1 = new OrderLine(6, Product.find("TAL"));  
    o1.addLine(line1);  
    OrderLine line2 = new OrderLine(5, Product.find("HPK"));  
    o1.addLine(line2);  
    OrderLine line3 = new OrderLine(3, Product.find("LGV"));  
    o1.addLine(line3);  
    line2.setSkippable(true);  
    o1.setRush(true);  
}
```

```
private void makeFluent(Customer customer) {  
    customer.newOrder()  
        .with(6, "TAL")  
        .with(5, "HPK").skippable()  
        .with(3, "LGV")  
        .priorityRush();  
}
```

Beispiel 2

-

Junit 4.4 / Hamcrest

```
assertEquals(10, list.size());
```

```
assertThat(list.size(), is(10));
```

```
assertThat(list, hasItems("a", "b", "c"));
```

```
assertThat(answer, is(not(41)));
```

```
assertThat(food,  
            is(sameInstance(Food.CHEESE))));
```

```
assertThat(string,  
            either(containsString("color"))  
                .or(containsString("colour")));
```

Beispiel 3

-

Joda-Time

```
public boolean isOverdue(DateTime renttime) {  
    Period maxRentPeriod =  
        new Period().withDays(2).withHours(12);  
  
    return renttime.plus(maxRentPeriod).isBeforeNow();  
}
```

Ein Bruch mit set/get/do

```
assertThat(string,  
    either(containsString("color"))  
    .or(containsString("colour")));
```

Verwendung soll „fließender“ sein

```
private void makeFluent(Customer customer) {  
    customer.newOrder()  
        .with(6, "TAL")  
        .with(5, "HPK").skippable()  
        .with(3, "LGV")  
        .priorityRush();  
}
```

Aber auch seltsames

Seltsame Methodennamen

`with(...)`

is(...)

either(...)

or(...)

Was machen diese
Methoden?

Bedeutung ergibt sich
erst aus dem Kontext

getLength()

dagegen, steht für sich

Seltsame Implementierungen

```
assertEquals(10, list.size());
```

```
assertThat(list.size(), is(10));
```

```
assertThat(answer, is(not(41)));
```

Das Fallbeispiel

Projekt bei einem
Kunden für einen
weiteren Kunden

4-10 Leute

heterogeneous Team

mehrere Jahre

Java 1.4

agil?

Software:
Vertragsmanagement
(Prozesse + Daten)

Das Problem

Geschäftsobjekte

Kunde

Vertrag

Vertragsposition

Bestellung

Bestellposition

Produkte

Produktdaten

Laufzettel (Prozessbeschreibung)

Prozessmappe

Viele und große
Geschäftsklassen

Diverse Abhängigkeiten

Diverse
„Entkopplungen“

Scheinentkopplung
per ID statt Referenz

Logische Kopplungen

Kundennummer



Vorgangsnummer

Produkt



Vertragsart

OK im
Produktivcode

Probleme in den Tests

```
public void testProcessOneRoutingSlip() {  
    // SETUP  
    RoutingSlip rs = new RoutingSlip();  
    Contract contract = new Contract();  
    Customer customer = new Customer();  
    Product product = new Product();  
  
    Person contactPerson = new Person();  
    contactPerson.setPhoneCode("030");  
    product.setContactPerson(contactPerson);  
    product.setInstallationAddress(new Address());  
    contract.addProduct(product);  
    customer.addContract(contract);  
    ProcessFolder pf = new ProcessFolder(rs, customer, contract);  
    _dataService.save(pf);  
  
    // RUN  
    _oca.processRoutingSlips();  
  
    // CHECK  
    assertTrue(pf.getRoutingSlip().isTaskDone());  
}
```

Simplest Thing?

Extract!
DRY!

```
public void testProcessFolderIsNotForwarded() {  
    // SETUP  
    MockReceiver testReceiver =  
        new RejectingReceiver();  
    ProcessFolder testProcessFolder =  
        TestFactory.createBlankProcessFolder();  
  
    // RUN  
    _pfds.registerReceiver(testReceiver);  
    _pfds.receive(testProcessFolder);  
  
    // CHECK  
    assertFalse(testReceiver.hasReceived());  
}
```

Testfactory:
Das große
Testobjekterstellungsmonster

10⁷ createXY()-
Methoden in TestFactory

23 Methoden um eine
Bestellung zu erzeugen

19 Methoden um einen
Kunden zu erzeugen

15 Methoden um einen
Vertrag zu erzeugen

```
createFullCustomer() {...}
createFullCustomerWithPredecessorContract() {...}
createIllegalContractCustomer() {...}
createCustomerWithoutLogin() {...}
createCustomerWithoutOrder() {...}
createCustomerWithoutCustomerNumber() {...}

createCustomer2WithoutLogin () {...}
createCustomer2WithoutOrder () {...}
createCustomer2WithoutCustomerNumber () {...}
```

Methodennamen und
Parameter reichen zur
Konfiguration nicht aus

Zu unübersichtlich

Zu unflexibel

Eine neue Lösung
musste her

Testobjekte erzeugen
sollte in den
Unittests ...

... möglichst einfach
sein

... möglichst
aussagekräftig sein

Günstige
Projektsituation
für ein kleines
Experiment

Fluent Interface

Vorgehensweise:
Durch den Bedarf in
den Tests getrieben

Beispiel:

Ich will testen, dass ein
Lieferant ausgewählt
wird

```
order = createOrder()  
        .withDeliveryAddress();
```

```
createOrderProcessFolder()  
    .withOrder(order)  
    .unlockedTask(DETERMINE_DELIVERY_PROVIDER);
```

„Sprache“ wurde
entwickelt, indem sie
gesprochen wurde

Oberste Richtlinie:
Sich der Benutzung der
Schnittstelle
unterzuordnen

Implementierung der
Schnittstelle sehr
strikt untergeordnet

Freiheit in der Formulierung

```
createOrderProcessFolder().with(Order);
```

VS.

```
createOrder().withProcessFolder();
```

Implementierung:
Klimmzüge und
rumtricksen, falls nötig

Besser aufwendigen,
dreckigen Code in der
Testobjekt-Bibliothek
und dafür
aussagestarken Code in
allen Unittests

Beispiel:

Vertragsart, Produkt,
Produkttdaten
koordinieren

Beispiel:

Zirkuläre Referenzen
benutzen

Beispiel:

Method-Chaining ermöglichen

```
createOrder()  
  .overProduct("Käse")  
  .withOrderNumber("ORDER 123_XY")  
  .withoutDeliveryAddress();
```

Beispiel:

Method-Chaining ermöglichen

```
public OrderBuilder overProduct(String p) {  
    _product = p;  
    return this;  
}
```

Implementierungsdetail 1:

Klasse TestObjects als
Objectmother

```
testobjects.createOrderProcessFolder();
```

Implementierungsdetail 2:

Fluent-Builder-Klassen

```
OrderBuilder o = testobjects.createOrder();
```

Testbuilder und
Fachklassen werden
getrennt gehalten

Eigentliche
Fachobjekte baut die
Objectmother

Fluent-Builder-Artikel von Bernd Schiffer:

[http://berndschiffer.blogspot.com/2008/04/
artikel-entwurfsmuster-flexibel.html](http://berndschiffer.blogspot.com/2008/04/artikel-entwurfsmuster-flexibel.html)

Und, wie ist's gelaufen?

Statische Typisierung:
Freund + Feind

Mehr Ausdrucksstärke
durch
viel Proxycode

Klassenhierarchie
vs.
Method-Chaining

Starre Struktur von
Java nicht ideal

Code-completion
hilfreich

Was ist gewichtiger als
die technische Seite?

Der Faktor Mensch!

Fluent Interfaces sind
ungewohnt

Gerade für Javaentwickler

heterogeneous Team!

Zu lange Soloprojekt

Zu wenig
Pairprogramming

Spät Wissenstransfer
auf anderen Kanälen

Leute taten sich
schwer damit, die
Testbibliothek zu
benutzen

Leute taten sich
schwer damit, die
Testbibliothek zu
erweitern

Schwierigkeiten beim Aufbau einer klaren Sprache

`createOrder.withProduct(XY)`

`createOrder.overProduct(XY)`

`createOrder.forProduct(XY)`

Englisch als Hürde?

Hat es sich gelohnt?

Ja, für das anvisierte
Ziel, war das Mittel
geeignet

Ein gewisses Maß an
neuen Techniken ist
hilfreich

Fluent Interfaces sind
ein scharfes Schwert

Abgegrenzter Bereich
hilfreich

Entwurf braucht
Zeit + Erfahrung

... weil man
rumprobieren muss

... weil die
Implementierung nicht
einfach ist

Auch die Benutzung
braucht
Zeit + Erfahrung

... weil die Benutzung
ungewohnt ist

... weil man ein Fluent
Interface auch
missbrauchen kann

```
assertThat(list.size(), is(10));
```

```
assertThat(i,
```

```
    anyOf(equalTo(0), allOf(greaterThan(5), lessThan(10))));
```

```
assertTrue("myNumber should be 0 or between 5 and 10",  
    myNumber == 0 || (myNumber > 5 && myNumber < 10));
```

```
assertThat(myNumber, is(0).or().between(5, 10));
```

Benutzer eines Fluent
Interfaces wird schnell
zu einem Erweiterer

Danke!

Meinungen?

Folien bald unter
<http://datenreisender.de/>