

# Schreib' die Tests, die du auch lesen magst!

andrena objects ag

4. Juni 2019  
Entwicklertag Karlsruhe

Claudia Fuhrmann

André Kappes

## Wer von Euch hat...

...schon einmal an einem roten Test gesessen und nicht verstanden, was er eigentlich bewirken soll?

...diesen Test selbst geschrieben?



# Agenda

Code-Typografie

Namensgebung

Lesbare Tests

Übersichtlichkeit

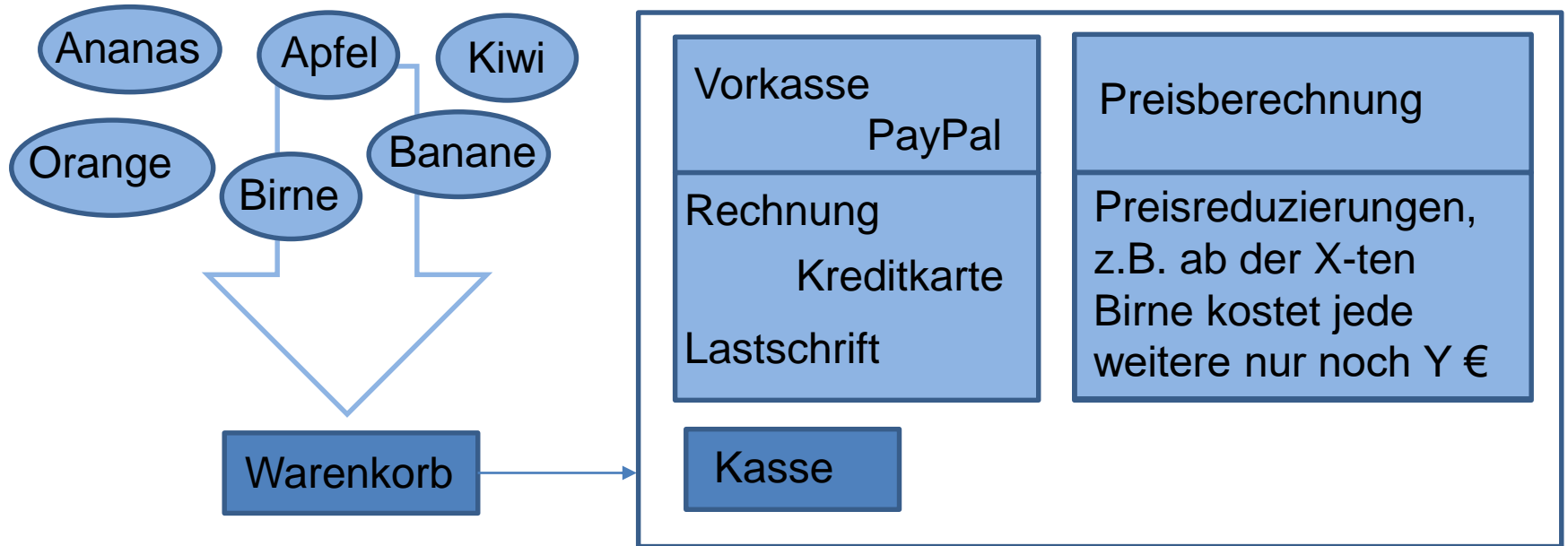
Prägnanz

Ablauflogik



# Das Projekt

# Online - Obstversand



Code-Typografie

Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Ablauflogik



## Regeln aus der Typografie

- Lange Zeilen bzw. Umbrüche vermeiden
- Durch Absätze strukturieren
- AAA-Pattern zur Orientierung:  
Gliederung des Tests in Abschnitten nach  
**Arrange – Act – Assert**





## Code-Typografie

```
@Test
void testAdd() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(ObstTyp.BIRNE,
        PreisStrategyFactory.createNormalPreisStrategy(Money.of(0.70)));
    Warenkorb warenkorb = new Warenkorb(preisRepository);
    warenkorb.add(5, ObstTyp.BIRNE);
    warenkorb.add(3, ObstTyp.BIRNE);
    List<Posten> posten = warenkorb.getPosten();
    assertEquals(ObstTyp.BIRNE, posten.get(0).getTyp());
    assertEquals(8, posten.get(0).getAnzahl());
    assertEquals(Money.of(5.60), posten.get(0).getPreis());
}
```

# Strukturierung mit dem AAA-Pattern

```
@Test
void testAdd() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(ObstTyp.BIRNE,
        PreisStrategyFactory.createNormalPreisStrategy(Money.of(0.70)));

    Warenkorb warenkorb = new Warenkorb(preisRepository);

    warenkorb.add(5, ObstTyp.BIRNE);
    warenkorb.add(3, ObstTyp.BIRNE);
    List<Posten> posten = warenkorb.getPosten();

    assertEquals(ObstTyp.BIRNE, posten.get(0).getTyp());
    assertEquals(8, posten.get(0).getAnzahl());
    assertEquals(Money.of(5.60), posten.get(0).getPreis());
}
```

Arrange

Act

Assert

## Zeilenlängen reduzieren

```
@Test
void testAdd() {
    PreisRepository repository = new PreisRepository();
    repository.save(BIRNE, normalPreis(0.70));
    Warenkorb warenkorb = new Warenkorb(repository);

    warenkorb.add(5, BIRNE);
    warenkorb.add(3, BIRNE);

    Posten soleItem = warenkorb.getPosten().get(0);

    assertEquals(BIRNE, soleItem.getTyp());
    assertEquals(8, soleItem.getAnzahl());
    assertEquals(Money.of(5.60), soleItem.getPreis());
}
```

Static imports  
verwenden

Erklärende Variablen  
einführen

Syntactic sugar-Methoden  
extrahieren

```
private PreisStrategy normalPreis(double value) {
    return PreisStrategyFactory.createNormalPreisStrategy(of(value));
}
```

Code-Typografie



Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Ablauflogik



# Wie soll sie\*denn heißen?

Sprechende Namen finden

\*) Variable, Testmethode

# Was sind schlechte Namen?

Typbezeichnungen an  
Namen

```
class BestellServiceTest {  
    private KundenRepository repositoryMock = mock(KundenRepository.class);  
    private BestellService service = new BestellService(repositoryMock);  
}
```

@Test

```
void berechnePreis() {  
    PreisRepository repository = new PreisRepository();  
    repository.save(BIRNE, reduzierterPreisAb(5, of(10, 15)));  
  
    Warenkorb input = new Warenkorb(repository);  
    input.add(5, BIRNE);  
  
    BestellUebersicht bu = service.getBestellUebersicht(input);  
  
    assertEquals(Money.of(3.30), bu.getTotalpreis());  
}
```

Unverständliche  
Abkürzungen

@Test

```
void someOtherTest() {  
    // uses KundenRepository  
}
```

Generische Namen  
verschleiern den Inhalt.

## Gute Namen

- Erklären den Zweck der Variablen bzw. des Felds in diesem Testfall
- Vermeide generische Namen – bevorzuge spezifische Namen
- Lesbar / aussprechbar
- Je kleiner der Scope, desto kürzer der Name



## Bessere Namen...

```
class BestellServiceTest {  
    private KundenRepository kundenRepository = mock(KundenRepository.class);  
    private BestellService bestellService = new BestellService(kundenRepository);  
  
    @Test  
    void berechnePreis() {  
        PreisRepository preisRepository = new PreisRepository();  
        preisRepository.save(BIRNE, reduzierterPreisAb(5, of(0.70), of(0.50)));  
  
        Warenkorb mit5Birnen = new Warenkorb(preisRepository);  
        mit5Birnen.add(5, BIRNE);  
  
        BestellUebersicht uebersicht = bestellService.getBestellUebersicht(mit5Birnen);  
  
        assertEquals(Money.of(3.30), uebersicht.getGesamtpreis());  
    }  
  
    @Test  
    void someOtherTest() {  
        // uses KundenRepository  
    }  
}
```





# Testmethoden benennen

Namenskonvention von Roy Osherove:

**Function/Feature – Input/State – Outcome/Behaviour**

```
@Test  
void berechnePreis () {  
    ...  
}
```



```
@Test  
void berechnePreis_reduktionAb5Birnen_wirdBeachtet() {  
    ...  
}
```



## Magic Values

```
@Test
void berechnePreis_reduktionAb5Birnen_wirdBeachtet() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(BIRNE, reduzierterPreisAb(5, of(0.70), of(0.50)));

    Warenkorb mit5Birnen = new Warenkorb(preisRepository);
    mit5Birnen.add(5, BIRNE);

    BestellUebersicht uebersicht = bestellService.getBestellUebersicht(mit5Birnen);

    assertEquals(Money.of(3.30), uebersicht.getGesamtpreis());
}
```

Warum 3.30?

## Magic Values

```
@Test
void berechnePreis_reduktionAb5Birnen_wirdBeachtet() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(BIRNE, reduzierterPreisAb(5, of(0.70), of(0.50)));

    Warenkorb mit5Birnen = new Warenkorb(preisRepository);
    mit5Birnen.add(5, BIRNE);

    BestellUebersicht uebersicht = bestellService.getBestellUebersicht(mit5Birnen);

    assertEquals(PREIS_FUER_VIER_BIRNEN.plus(REDUZIERTER_PREIS_FUER_FUENFTE_BIRNE),
                 uebersicht.getGesamtpreis());
}
```

Erklärende Konstanten

## Empfehlungen

- Erklärende Namen für lokale Variablen und Felder wählen
- Testmethode aussagekräftig benennen
- Magic Values vermeiden





Code-Typografie



Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Ablauflogik



# Prägnanz

```
private Posten EINE_ANANAS = Posten.of(1, ANANAS, Money.of(3.00));
private Posten EINE_BANANE = Posten.of(1, BANANE, Money.of(2.00));
private Posten EIN_APFEL = Posten.of(1, APFEL, Money.of(1.50));
private Posten EINE_BIRNE = Posten.of(1, BIRNE, Money.of(1.00));
private Posten EINE_KIWI = Posten.of(1, KIWI, Money.of(3.50));
private Posten EINE_ORANGE = Posten.of(1, ORANGE, Money.of(2.50));
```

```
private PreisRepository preisRepository;
private KundenRepository kundenRepository;
```

```
@BeforeEach
void setUp() {
    preisRepository =
        PreisRepositoryImpl
        kundenRepository
}
}
```

```
@Test
void testWarenkorb_BestellServiceUndZahlungsService() {
    Warenkorb warenkorb = new Warenkorb(preisRepository);
    warenkorb.add(1, ANANAS);
    warenkorb.add(1, BANANE);
    warenkorb.add(1, APFEL);
    warenkorb.add(1, BIRNE);
    warenkorb.add(1, KIWI);
    warenkorb.add(1, ORANGE);
```

```
UUID kundenId = warenkorb.getKundenId();
assertThat(kundenId).isNotNull();
```

Posten im  
Warenkorb

```
List<Posten> posten = warenkorb.getPosten();
assertThat(posten).contains(EINE_ANANAS);
assertThat(posten).contains(EINE_BANANE);
assertThat(posten).contains(EIN_APFEL);
assertThat(posten).contains(EINE_BIRNE);
assertThat(posten).contains(EINE_KIWI);
assertThat(posten).contains(EINE_ORANGE);
```

Bestellübersicht

```
BestellService bestellService = new BestellService(kundenRepository);
BestellUebersicht bestellUebersicht = bestellService.getBestellUebersicht(warenkorb);
```

Viel zu lang und unübersichtlich!

Zahlungsarten  
des  
ZahlungsService

```
assertThat(zahlungsarten).contains(KREDITKARTE);
assertThat(zahlungsarten).contains(LASTSCHRIFT);
assertThat(zahlungsarten).contains(PAYPAL);
assertThat(zahlungsarten).contains(RECHNUNG);
assertThat(zahlungsarten).contains(VORKASSE);
```

KundenId  
wird gesetzt

## Fragen, die ich mir stellen sollte, wenn der Test zu lang ist:

- Kann man den Test aufteilen, weil nicht alles in diesen Test gehört?
- Gehören die einzelnen Asserts in eine andere Testklasse?
- Hat die getestete Klasse zu viele Verantwortlichkeiten?



# Prägnanz

```
@Test
void kundenIdIstNichtNull() {
    Warenkorb warenkorb = new Warenkorb(preisRepository);
    UUID kundenId = warenkorb.getKundenId();
    assertThat(kundenId).isNotNull();
}
```

Kurze Tests

```
@Test
void getPosten() {
    Warenkorb warenkorb = WarenkorbBuilder.get()
        .addAnanas(1)
        .addBananen(1).create();

    List<Posten> posten = warenkorb.getPosten();
    assertThat(posten).containsExactlyInAnyOrder(EINE_ANANAS, EINE_BANANE);
}
```

Wenige Zusicherungen



# Prägnanz

```
public class BestellServiceTest {  
    private BestellService underTest;  
  
    @BeforeEach  
    void setUp() {  
        underTest = new BestellService(new WarenRepository());  
    }  
  
    @Test  
    void getBestellUebersicht() {  
        Warenkorb warenkorb = WarenkorbBuilder.  
            mitWaren(Money.of(2), Money.of(3)).create();  
  
        BestellUebersicht uebersicht = underTest.getBestellUebersicht(warenkorb);  
        assertThat(uebersicht.getPosten()).containsExactlyInAnyOrder(EINE_ANANAS, EINE_BANANE);  
    }  
}
```

Test in die richtige Testklasse  
hinzufügen

# Prägnanz

```
public class ZahlungsServiceTest {  
    private ZahlungsService underTest;  
  
    @BeforeEach  
    void setUp() {  
        underTest = new ZahlungsService(new KundenRepository());  
    }  
  
    @Test  
    void getZahlungsarten() {  
        Warenkorb warenkorb = WarenkorbBuilder.get().addAnanas(1, Money.of(2))  
            .addBananen(1, Money.of(3)).create();  
  
        List<ZahlungsArt> zahlungsarten = underTest.getZahlungsarten(bestellUebersicht);  
        assertThat(zahlungsarten).containsExactlyInAnyOrder(LASTSCHRIFT, LASTSCHRIFT,  
            PAYPAL, RECHNUNG, VORKASSE);  
    }  
}
```

# Empfehlungen

Tests sollten:

- kurz sein
- wenige Zusicherungen haben, am besten nur eine
- in den dazugehörigen Testklassen sein

Wenn dein Test nicht mehr auf deinen Bildschirm passt, ist er definitiv zu lang!



## Sprechende Zusicherungen und Fehlermeldungen

```
@Test
void testZahlungService() {
    BestellUebersicht ueberLimit = mock(BestellUebersicht.class);
    when(ueberLimit.getGesamtpreis()).thenReturn(UEBER_LIMIT_FUER_RECHNUNG);

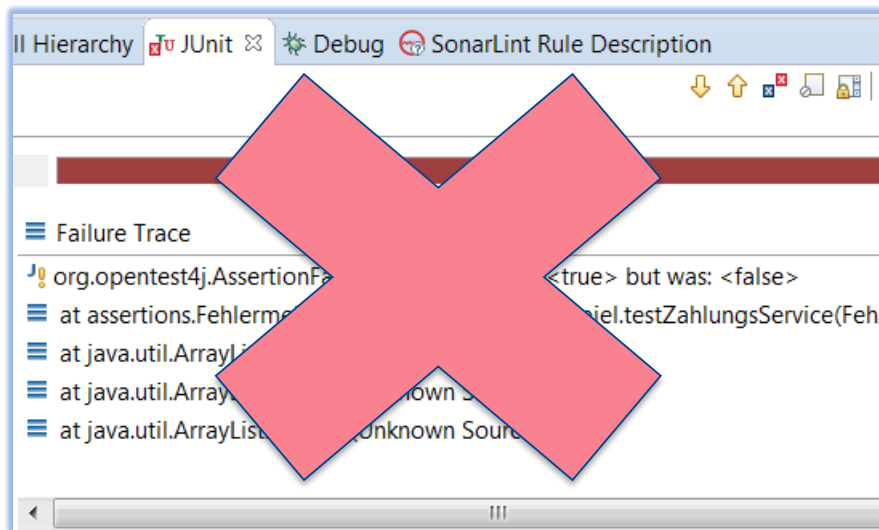
    ZahlungsService zahlungsService = new ZahlungsService(new KundenRepository());
    List<ZahlungsArt> angeboteneZahlungsArten = zahlungsService.getZahlungsarten(ueberLimit);

    assertTrue(!angeboteneZahlungsArten.contains(ZahlungsArt.RECHNUNG));
}
```

Keine Informationen über  
den Grund eines Fehlschlags

Negation

## Sprechende Zusicherungen und Fehlermeldungen



- Einfache Bewertung von Testfehlern, insbesondere bei Auftreten auf dem Build-Server
- Hilfreich bei instabileren Tests, z.B. Integrationstests oder Tests für externe Abhängigkeiten



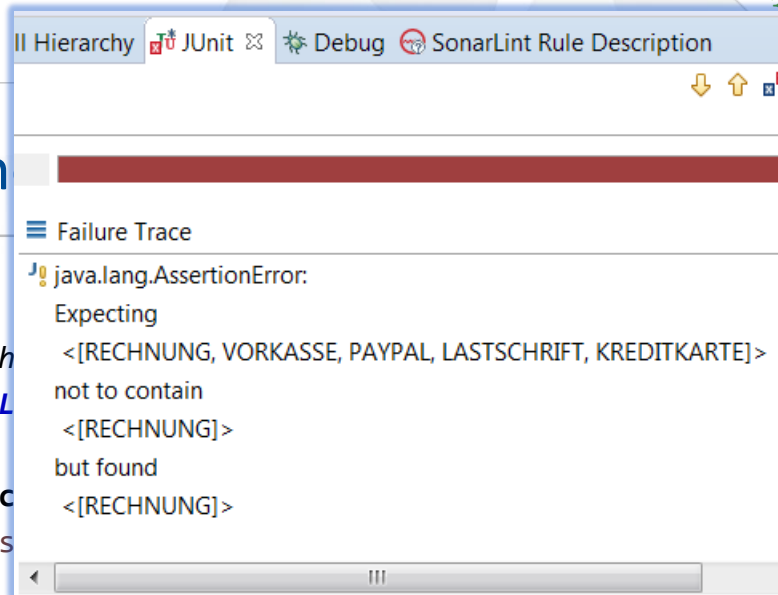
## Matcher für aussagekräftigere Zusich

```
@Test
void testZahlungsService() {
    BestellUebersicht ueberLimit = mock(BestellUebersicht)
    when(ueberLimit.getGesamtpreis()).thenReturn(UEBER_L

    ZahlungsService zahlungsService = new ZahlungsService
    List<ZahlungsArt> angeboteneZahlungsArten = zahlungs

    assertThat(angeboteneZahlungsArten).doesNotContain(ZahlungsArt.RECHNUNG);
}
```

Assertj Matcher



## Empfehlungen

- Zusicherungen möglichst präzise formulieren
- Auf sprechende Fehlermeldung im Testfehlschlags-Fall achten





Code-Typografie



Namensgebung

Lesbare Tests

Übersichtlichkeit



Prägnanz

Ablauflogik





## Arrange

```
private List<ObstTyp> obst = asList(ObstTyp.values());

@Test
void berechnePreis_BefuellterWarenkorb_kostet600() {
    Warenkorb warenkorb = WarenkorbFactory.getWarenkorb();
    for (int i = 0; i < obst.size(); i++) {
        if (i % 2 == 0) {
            warenkorb.add(new Ware(1, obst.get(i)));
        } else {
            warenkorb.add(new Ware(2, obst.get(i)));
        }
    }
    int preis = underTest.berechnePreis();
    assertThat(preis).isEqualTo(600);
}
```

Was wird hier eigentlich gemacht?

# Arrange

```
private static final int PREIS_ANANAS = 170;
private static final int PREIS_BIRNE = 70;

@Test
void berechnePreis_eineAnanasUndZweiBirnen_kosten310() {
    Warenkorb warenkorb = WarenkorbFactory.getWarenkorb();
    warenkorb.add(new Ware(1, ANANAS));
    warenkorb.add(new Ware(2, BIRNE));

    int preis = warenkorb.berechnePreis();

    assertThat(preis).isEqualTo(PREIS_ANANAS + 2 * PREIS_BIRNE);
}
```

Gezielt erstellen, was  
gebraucht wird

## Lösungsansätze

- Erstelle nur, was auch gebraucht wird
- Erstelle für jedes mögliche Szenario einen eigenen Test
- Parametrisierte Tests



# Assert

```
@Test
void getPosten_zweimal5BirnenHinterlegt_10BirnenImWarenkorb() {
    Warenkorb warenkorb = WarenkorbFactory.getWarenkorb();
    warenkorb.add(5, ObstTyp.BIRNE);
    warenkorb.add(5, ObstTyp.APFEL);
    warenkorb.add(5, ObstTyp.BIRNE);

    List<Posten> allPosten = warenkorb.getPosten();

    assertThat(allPosten.size()).isEqualTo(2);
    for (Posten posten : allPosten) {
        if (ObstTyp.BIRNE.equals(posten.getTyp())) {
            assertThat(posten.getAnzahl()).isEqualTo(10);
            break;
        }
    }
}
```

Conditional Verification Logic

# Assert

```
private Posten ZEHN_BIRNEN = Posten.of(10, BIRNE, Money.of(7));  
private Posten FUENF_AEPFEL = Posten.of(5, APFEL, Money.of(4));
```

```
@Test
```

```
void getPosten_zweimal5BirnenHinterlegt_10BirnenImWarenkorb() {  
    Warenkorb warenkorb = WarenkorbFactory.getWarenkorb();  
    warenkorb.add(5, BIRNE);  
    warenkorb.add(5, APFEL);  
    warenkorb.add(5, BIRNE);
```

```
    List<Posten> allPosten = warenkorb.getPosten();
```

```
    assertThat(allPosten).containsExactlyInAnyOrder(ZEHN_BIRNEN, FUENF_AEPFEL);
```

```
}
```

Equality  
Assertion

Auch ein einfaches  
contains möglich

# Empfehlungen und Lösungsansätze

Verwende:

- keine Schleifen (For, while,...)
- keine If-Abfrage
- Equality Assertions
- Customized Assertions





Code-Typografie



Namensgebung

Lesbare Tests

Übersichtlichkeit



Prägnanz



Ablauflogik



# Hin- und Herspringen

Warum? Was gibt es dort?

```
public class BestelluebersichtTest extends WarenkorbTest {  
  
    private Posten EINE_BANANE = Posten.of(1, BANANE, Money.of(2.00));  
    private Posten EINE_ANANAS = Posten.of(1, ANANAS, Money.of(3.00));  
    private Posten EINE_KIWI = Posten.of(1, KIWI, Money.of(4.00));  
  
    private PreisRepository preisRepository = new PreisRepository();  
    private KundenRepository kundenRepository;  
    private BestellService underTest;  
  
    @BeforeEach  
    void setUp() {  
        PreisRepositoryInitializer.create(preisRepository).init();  
        underTest = new BestellService(kundenRepository);  
    }  
}
```

Die Kiwi wird nicht benutzt

Wie sehen die Preise aus?



## Hin- und Herspringen

```
@Test
void getBestelluebersicht_eineAnanasUndEineBanane_Bestelluebersicht()
    Warenkorb warenkorb = createWarenkorbWithAnanasUndBanane(preisRepository);
    Bestelluebersicht bestelluebersicht = underTest.getBestelluebersicht(warenkorb);
    assertBestelluebersicht(bestelluebersicht, warenkorb.getKundenId(), Money.of(5.00),
                            EINE_ANANAS, EINE_BANANE);
}

private void assertBestelluebersicht(Bestelluebersicht bestelluebersicht, UUID kundenId,
                                     Money preis, Posten... posten) {
    assertThat(bestelluebersicht.getPreis()).isEqualTo(preis);
    assertThat(bestelluebersicht.getPosten()).containsExactlyInAnyOrder(posten);
    assertThat(bestelluebersicht.getKundenId()).isEqualTo(kundenId);
}
```

Wo finde ich diese Methode?

Was testet dieses Assert?

# Hin- und Herspringen

```
public class BestellServiceTest {  
    private Money PREIS_EINE_ANANAS = Money.of(3.00);  
    private Money PREIS_EINE_BANANE = Money.of(2.00);  
    private Posten EINE_ANANAS = Posten.of(1, ANANAS, PREIS_EINE_ANANAS);  
    private Posten EINE_BANANE = Posten.of(1, BANANE, PREIS_EINE_BANANE);  
    private Posten EINE_KIWI = Posten.of(1, KIWI, Money.of(4.00));  
  
    private KundenRepository kundenRepository;  
    private BestellService underTest;  
  
    @BeforeEach  
    void setUp() {  
        kundenRepository = new KundenRepository();  
        underTest = new BestellService(kundenRepository);  
    }  
}
```

Toten Code löschen

Keine magische  
Preiserstellung

## Hin- und Herspringen

```
@Test
```

```
void getBestelluebersicht_eineAnanasUndEineBanane() {
```

```
    Warenkorb warenkorb = WarenkorbBuilder.warenkorb().withKundenId(KUNDEN_ID)
        .addAnanas(1, PREIS_EINE_ANANAS)
        .addBananen(1, PREIS_EINE_BANANE).create();
```

```
    Bestelluebersicht bestelluebersicht = underTest.getBestelluebersicht(warenkorb);
```

```
    assertThat(bestelluebersicht)
        .hasGesamtpreis(PREIS_EINE_ANANAS.plus(PREIS_EINE_BANANE))
        .hasPosten(EINE_ANANAS, EINE_BANANE)
        .hasKundenId(KUNDEN_ID);
```

```
}
```

Builder, der den Warenkorb  
samt Preise erstellt

Custom matcher

## Empfehlungen und Lösungsansätze

- Relevantes sichtbar machen, Irrelevantes verbergen
- Nichts Zusätzliches erstellen
- Builder Pattern
- Custom Matchers





Code-Typografie



Namensgebung

**Lesbare Tests**



Übersichtlichkeit



Prägnanz



Ablauflogik



# Zusammenfassung

Kümmere Dich um Eure Tests!

Code-Typografie

Namensgebung

**Lesbare Tests**

Übersichtlichkeit

Prägnanz

Nimm Dein Team mit!

Ablauflogik



# Referenzen

## Bücher

- Roy Osherove, **The Art of Unit Testing** (Manning, 2013)
- Gerard Meszaros, **xUnit Test Patterns: Refactoring Test Code** (Addison Wesley, 2007)  
<http://xunitpatterns.com/>

## Blogs

- Petri Kainulainen, **Writing Clean Tests**  
<https://www.petrikainulainen.net/writing-clean-tests/>
- Thomas Countz, **Essential & Relevant: A Unit Test Balancing Act**  
<https://8thlight.com/blog/thomas-countz/2019/02/19/essential-and-relevant-unit-tests.html>





Vielen Dank!

## Bitte geben Sie uns jetzt Ihr Feedback!

Schreib' die Tests, die du auch lesen magst!

*André Kappes, Claudia Fuhrmann*



### Nächste Vorträge in diesem Raum

**11:45** Müssen wir wirklich schon wieder alles testen? Trotz langlaufender Testsuiten Fehler schnell und zuverlässig aufdecken, *Dr. Sven Amann*

**13:30** Wenn der PO die Akzeptanztests schreibt – ATDD in der Praxis, *Dr. Rolf Schneeweiß, Kathrin Ronellenfitsch*

**14:30** Webservices testen als Mob – Das Warum, das Was und das Wie, *Mario Kühne*