agentes

Die NullPointerException-Falle

If we were able to understand it,
we wouldn't call it code!

(aus http://www.michael-puff.de/Ablage/Signaturen.php)

5. Juli 2007
Java Forum Stuttgart

Oliver.Boehm@agentes.de

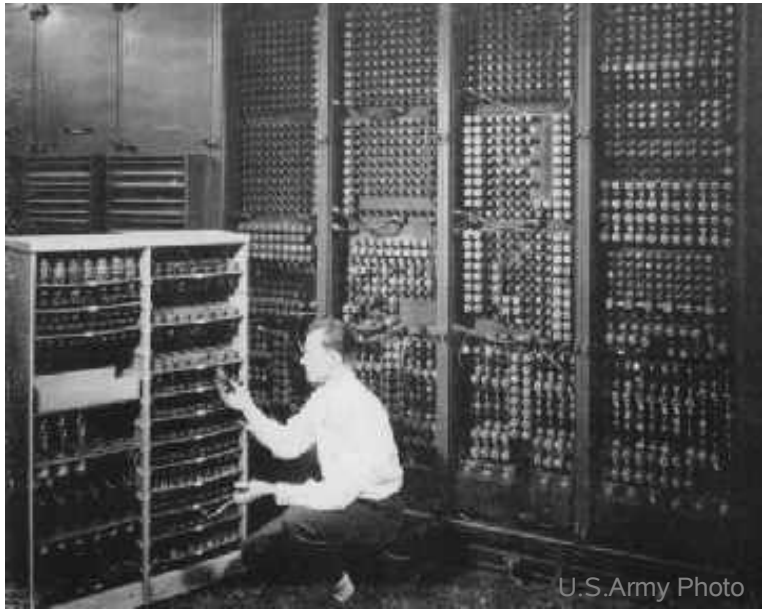
agentes AG 2006

- **Aktionäre**
  - Pironet NDH AG (Hauptaktionär)
  - Vorstände & Manager
- **Standorte**
  - Stuttgart (Firmensitz), Kassel, Hamburg, München
- **140 Mitarbeiter**
- **Ergebnis**
  - 2004: Umsatz 8,6 Mio EUR, EBIT 834 TEUR
  - 2005: Umsatz 8,7 Mio EUR, EBIT ca. 850 TEUR
  - 2006: Umsatz 9,7 Mio EUR (HGB)
- **Kernkompetenzen**
  - Softwareentwicklung für / Wartung von komplexen Verfahren und Banksystemen
  - Softwarelösungen zur Vertriebsoptimierung

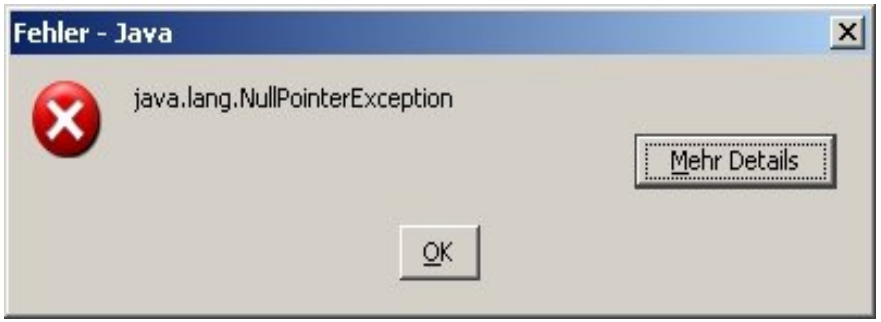
Inhalt

- Das Problem
- Die Idee
- Ein kleiner Abstecher in AOP
- Die Realisierung
- PatternTesting
- Diskussion

Das Problem



Demo

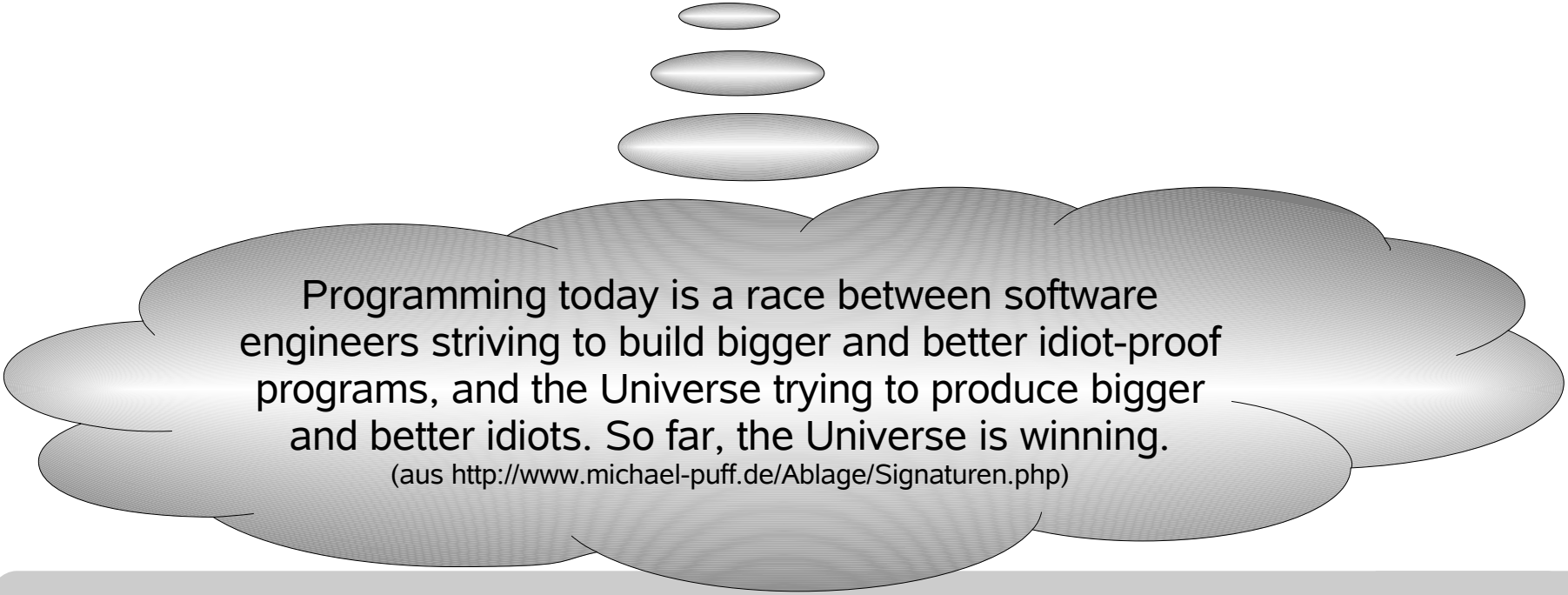


Was sind NullPointerExceptions (NPE)?

- **normal?**
  - http://www.gourmondo.de/hitlist.jsp (-> gibt's nemmer)
  - http://www.sisby.de/sisby/base/de/Suche/GewImmoBoerse/ergSuche/index
  - http://www.learn-line.nrw.de/wettbewerbe/details.jsp?id=109
  - http://www.casinoeuro.com/de/flash/fullflash.jsp?game=
  - s. Google: *"java.lang.NullPointerException" filetype:jsp*
- **ärgerlich!**
- **Zeichen unzureichender Tests**
- **gefährlich für die Anwendung**
- **schlecht für das Image**

Wodurch entstehen NPEs?

- **durch Sorglosigkeit**
- **durch geänderte Rahmenbedingungen**
  - andere Umgebung (Produktion)
  - geänderte Anforderungen
- **durch unzureichende Tests**



Die Idee

Wie lassen sich NPEs vermeiden?

agentes

Wehret den Anfängen:

- (null) als Argument ist verboten
- (null) als Rückgabewert wird verboten
- Ausnahmen werden dokumentiert

Weitere Maßnahmen:

1) Tests
2) Tests
3) Tests

mögliche Realisierung

- **Abfragen / Asserts von Hand ergänzen**
  - viel zu aufwändig
- **(C-)Preprozessor und Makros**
  - nicht wirklich
- **Code über Classloader erweitern**
  - könnte klappen
  - aber: Classloader-Probleme können sehr, sehr frustrierend sein
- **über AOP / AspectJ ?**
  - mal sehen

Ein kleiner Abstecher in AOP

Beispiel: Bankkonto

fachliche Concerns

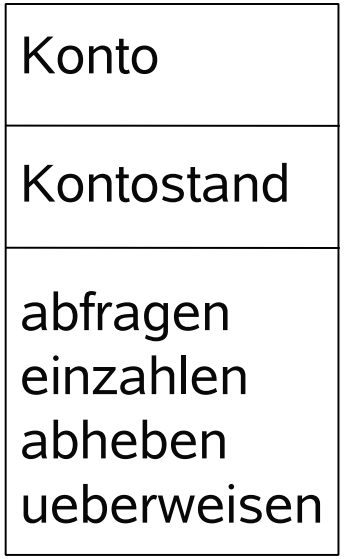
- Einzahlung
- Auszahlung
- Überweisung
- Bonitätsprüfung
- ...

nichtfachliche Concerns

- Logging
- Performance
- Authorisierung
- Sicherheit
- ...

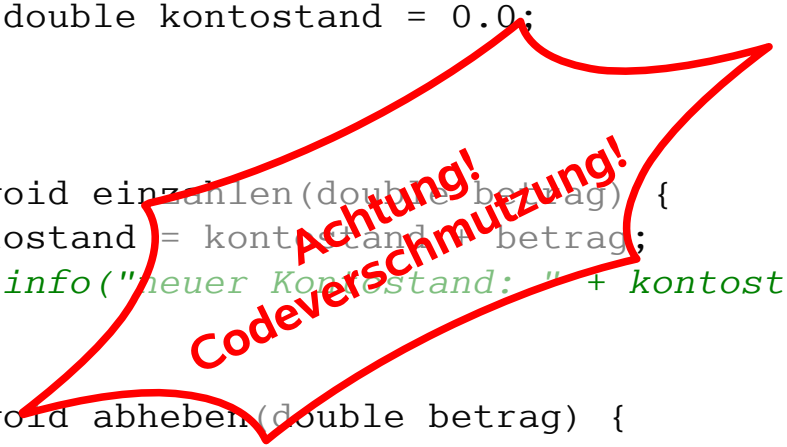
Ein Fallbeispiel...


```
public class Konto {

    private double kontostand = 0.0;

    public double abfragen() {
        return kontostand;
    }

    public void einzahlen(double betrag) {
        kontostand = kontostand + betrag;
    }

    public void abheben(double betrag) {
        kontostand = kontostand - betrag;
    }

    public void ueberweisen(double betrag, Konto anderesKonto) {
        this.abheben(betrag);
        anderesKonto.einzahlen(betrag);
    }

}
```



...mit Logging

```
public class Konto {

    private static Logger log = Logger.getLogger(Konto.class)
    private double kontostand = 0.0;

    ...

    public void einzahlen(double betrag) {
        kontostand = kontostand + betrag;
        log.info("neuer Kontostand: " + kontostand);
    }

    public void abheben(double betrag) {
        kontostand = kontostand - betrag;
        log.info("neuer Kontostand: " + kontostand);
    }

    ...

}
```



neue Anforderung:

alle Konto-Bewegungen müssen protokolliert werden!

...mit AspectJ

```
public aspect LogAspect {

    private static Logger log = Logger.getLogger(LogAspect.class);

    after(double neu) : set(double Konto.kontostand) && args(neu) {
        log.info("neuer Kontostand: " + neu);
    }

}
```

Do you speak AspectJ?

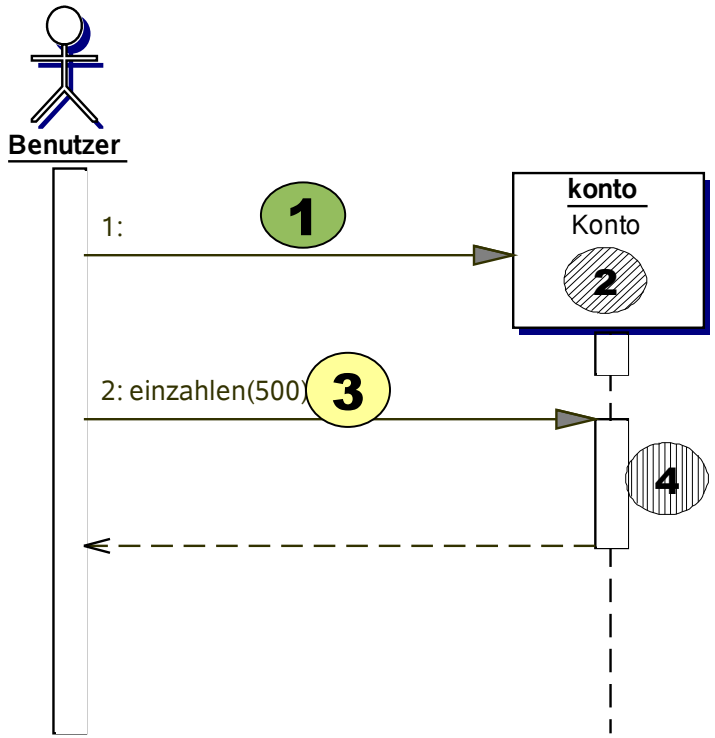
agentes

- **Joinpoint**
  - Eingriffspunkte im Programm, an denen Code erweitert oder modifiziert werden soll
- **Pointcut**
  - eine Auswahl von Joinpoints
- **Advice**
  - der Code für den Pointcut
- **Introduction**
  - Erweiterung anderer Klassen, Interfaces, Aspekte um zusätzliche Funktionalität
- **Aspect**
  - Konstrukt, in dem das obige abgelegt wird

Joinpoints können sein

- Aufrufen einer Methode
- Ausführen einer Methode
- Zugriff auf eine Variable
- Behandeln einer Exception
- Initialisierung einer Klasse
- Initialisierung eines Objekts

```
Konto konto = new Konto();
konto.einzahlen(500.0);
```



(1) Konstruktor aufrufen

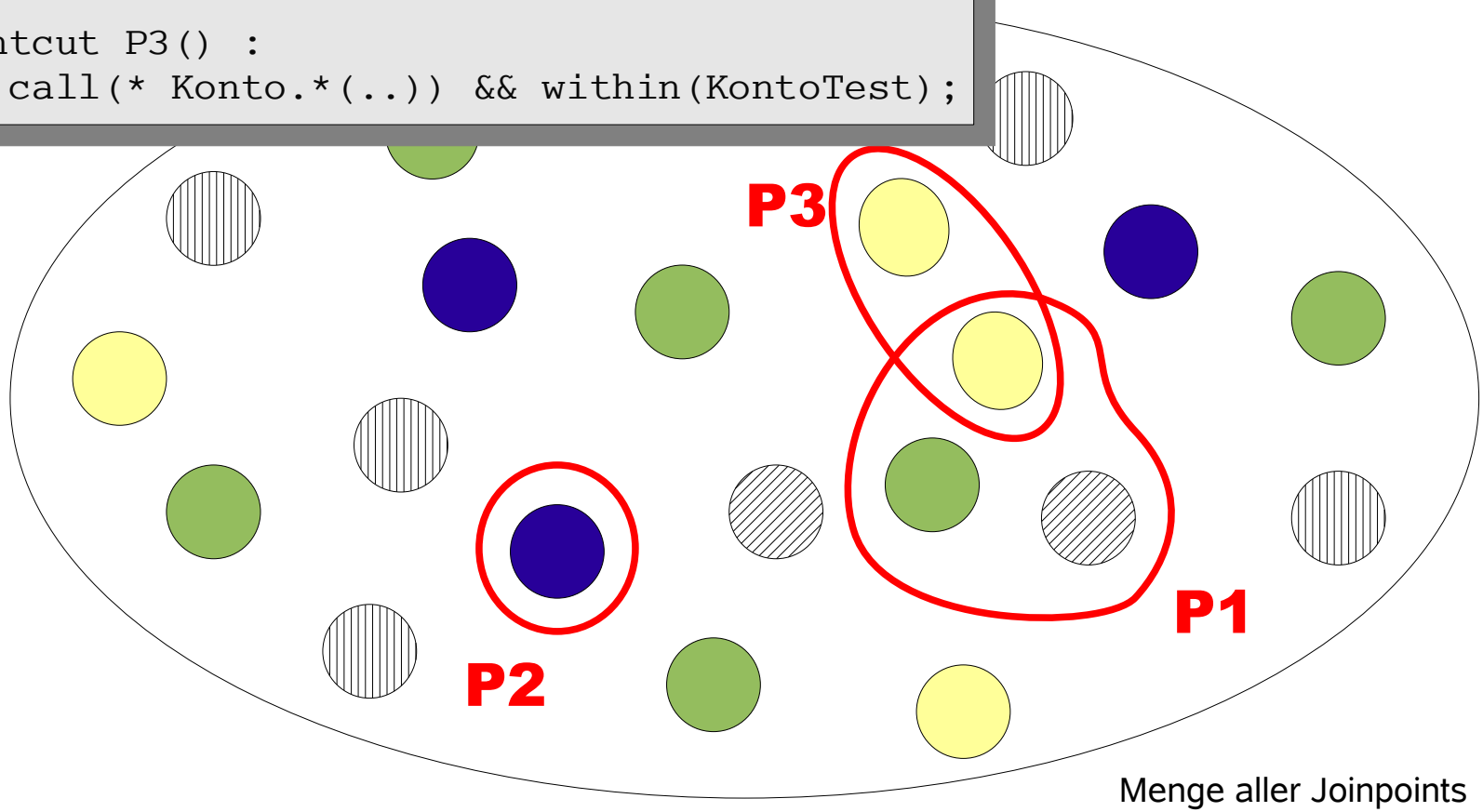
(2) Objekt initialisieren

(3) Methode aufrufen

(4) Methode ausführen

Pointcuts

```
pointcut P2() : set (double Konto.kontostand)

pointcut P3() :
    call(* Konto.*(..)) && within(KontoTest);
```



Menge aller Joinpoints

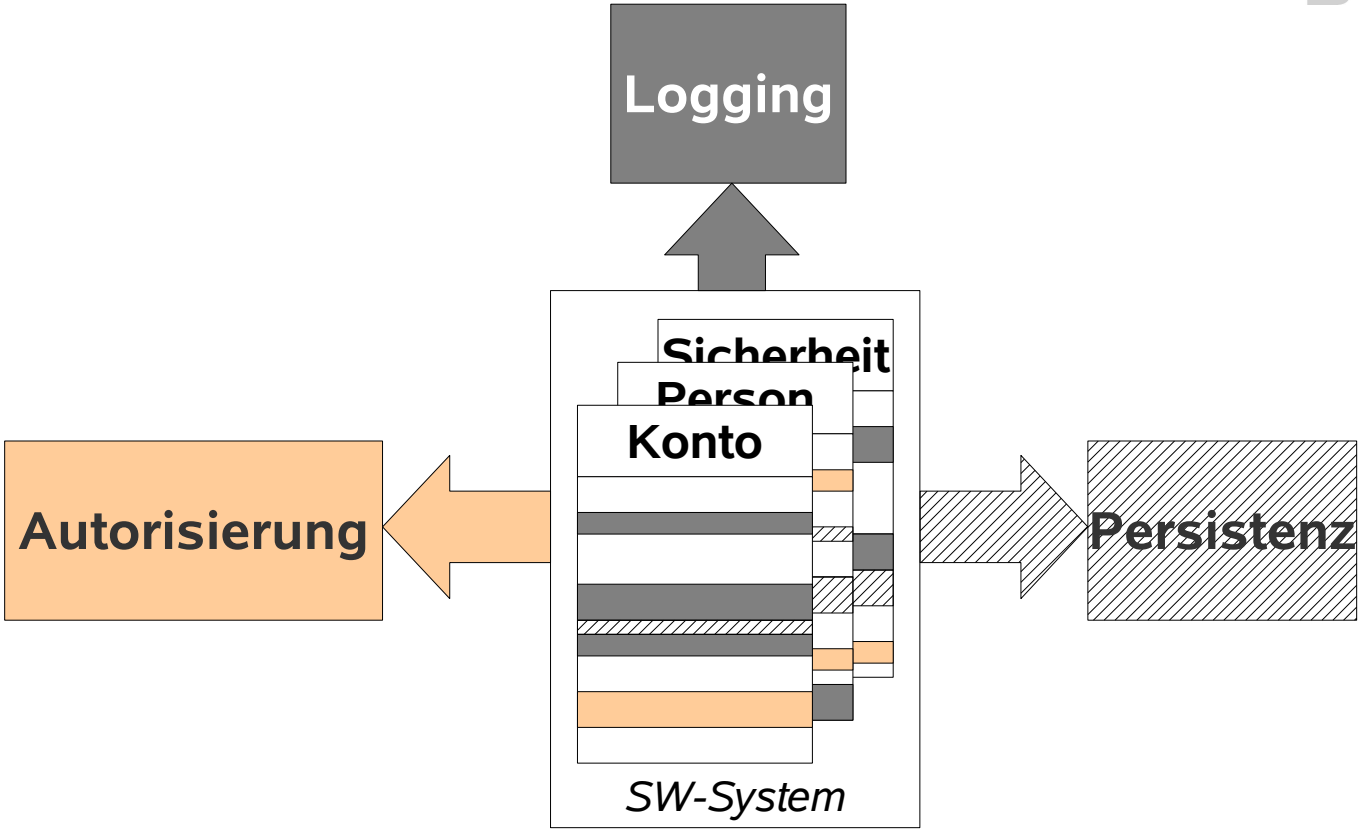
Advice

- **Code, der eingewebt wird**
  - before()
  - after()
  - around()
- **Ähnlichkeit mit Methoden**

```
after() : P3() {
    log.info("TEST: " + thisJoinPoint);
}
```

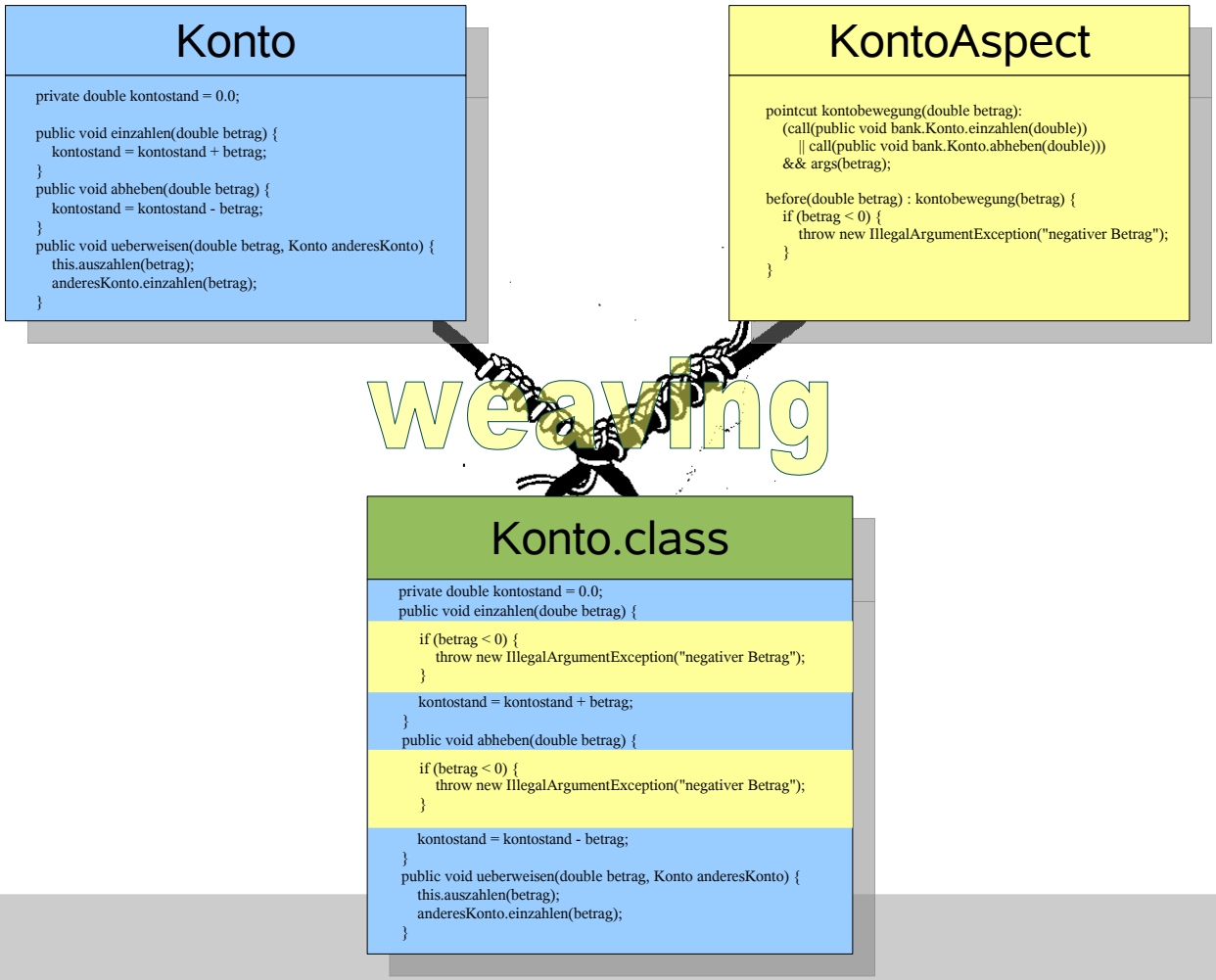
```
TEST: call(void bank.Konto.einzahlen(double))
TEST: call(double bank.Konto.abfragen())
...
```

Do you think in AOP?



Das System als Menge von “Concerns”

Der Webe-Vorgang (Weaving)



Die Realisierung

Lösung mittels AOP

- **Joinpoints definieren:**
  - alle Methoden u. Konstruktoren mit mind. 1 Parameter

```
pointcut methodsWithNoNullArgs() :
    execution(public * *..*(*, ..))
    || execution(public *..new(*, ..));
```

- **Code einfügen (über Advice):**

```
before() : methodsWithNoNullArgs() {
    Object[] args = thisJoinPoint.getArgs();
    for(int i = 0; i < args.length; i++) {
        if (args[i] == null) {
            throw new AssertionError("arg" + i + " is null");
        }
    }
}
```

erstes Ergebnis

- **Erschrecken**
  - verdammt viele unsaubere Code-Stellen
  - zu oft zu sorglos

- **Verzweiflung**
  - nicht alle gefundenen „Unsauberkeiten“ ließen sich einfach beheben
  - Ausnahmeregelung gefordert!

Lösung II

- **Ausnahmen definieren**

```
pointcut nullArgsAllowed() :
    execution(public int
        ebanking.jaas.XmlLogin.loginWithVrNetKey(String,
        String, String))
    || execution(public void ebanking.design.Setting.set*(*))
    ;
```

- **Advice anpassen**

```
before() : methodsWithNoNullArgs()
        && !nullArgsAllowed() {
    ...
}
```

Rückgabe überwachen

- **Joinpoint mit Ausnahme**

```
pointcut nonVoidMethods() :
    execution(public Object+ *..*(..));
```

- **Ausnahmen**

```
pointcut mayReturnNull() :
execution(* ebanking.PersonOverview.getPerson(String))
|| execution(* *..*.find*())
;
```

- **Advice**

```
after() returning(Object returned) :
        nonVoidMethods() && !mayReturnNull() {
    if (returned == null) {
        throw new AssertionError(thisJoinPoint.getSignature()
                + " returns (null)!");
    }
}
```

Ergebnis

- **Aufdeckung vieler unsauberer Stellen**
  - setName(null) -> resetName()
- **Dokumentation der Ausnahmen**
  - Sorglosigkeit ersetzt durch Zwang zum Überlegen (dokumentiere ich oder mache ich es richtig?)
- **weniger NullPointerExceptions**
- **aber:**
  - *Tests nach wie vor notwendig!!!*

agentes

PatternTesting

Was ist PatternTesting?

agentes

PatternTesting is a testing framework that allows to automatically verify that Architecture/Design/Best practices recommendations are implemented correctly in the code. It uses AOP and AspectJ to perform this feat.

(aus: http://patterntesting.sf.net)

die Idee dahinter

- **Auspüren von „BugPattern“**
  - Übergabe von *null*-Argumenten
  - *null* als Rückgabewert
  - „bad smells“ (z.B. *e.printStackTrace()* statt Logging)
- **Einhalten von Programmierrichtlinien**
  - kein System.out.println()
  - Lasagne-Architektur:
    - obere Schicht darf nur darunterliegende Schicht aufrufen
    - nicht umgekehrt

Beispiel

agentes

- **declare error / declare warning**

```
declare warning:
    call(public * java.sql.*(..))
        "please use persistence framework...";
```

- **Exception Handling**

```
pointcut handleException(Throwable ex):
    handler(java.lang.Throwable+) && args(ex);

/** protokolliere alle abgefangen Exceptions */
before(Throwable ex) : handleException(ex) {
    System.err.println("*** " + ex);
}
```

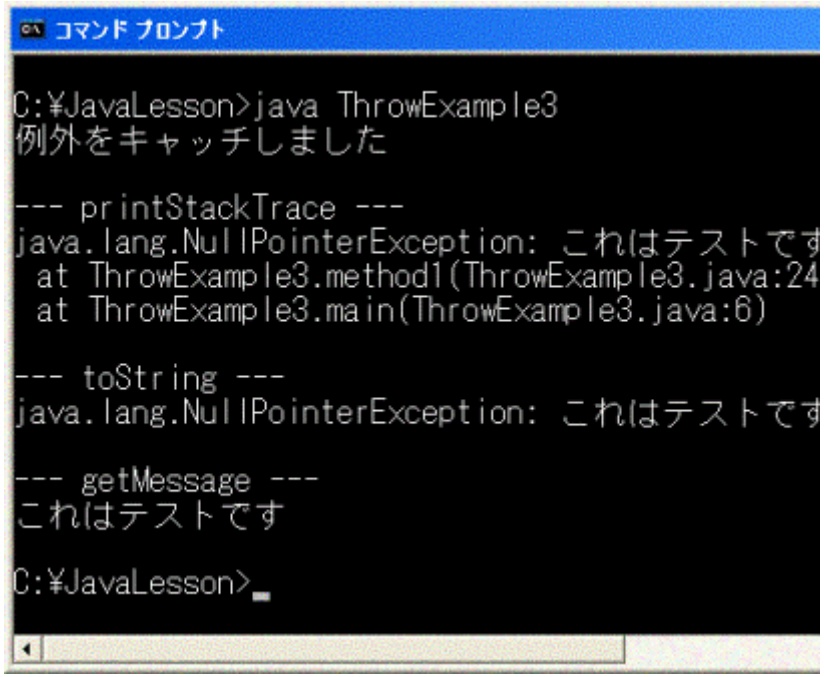
aktueller Status

- **etwas verwahrlost**
- **seit 2 Jahren Stillstand**
  - letzte Version: PatternTesting-0.3 vom Sept. 2004
- **aber: gute Ideen**
  - z.B. Einbindung als Maven-Plugin

PatternTesting-0.5-SNAPSHOT

- **Anfrage / Kontaktaufnahme im April**
  - mit Vincent Massol u. Matt Smith
  - urspr. Grund: AOP-Day '07
  - zurzeit mit anderen Projekten beschäftigt (Ruleby, Xwiki, Cargo/Maven)
- **derzeitige Planung**
  - ✔ Umstellung Maven 1 -> Maven 2
  - ✔ Test-Fälle zum Laufen bringen
  - autom. Build (Continuum)
    - ✗ Build-Server gesucht
    - ✗ noch kein autom. Deployment
- **künftige Planung (0.6)**
  - Einsatz von Annotations (-> Java5, AspectJ5)
  - weitere Aspekte aus bestehenden Projekten einpflegen

Abspann



Zusammenfassung

agentes

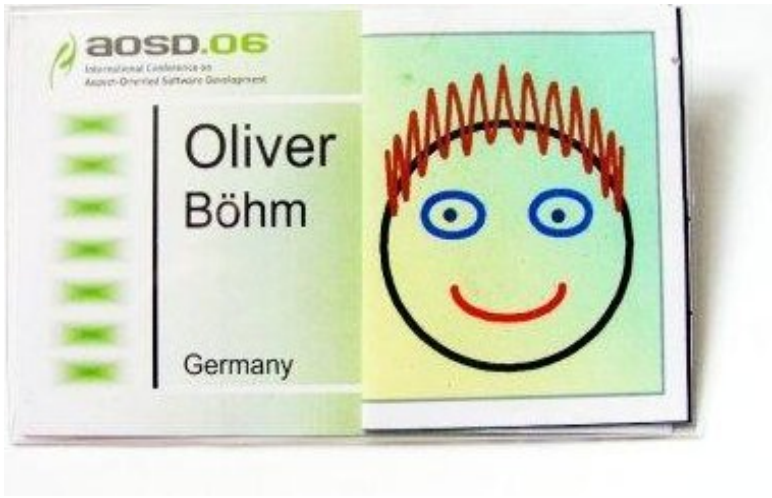
- **NullPointerExceptions sind Programmierfehler!**
- **AOP kann helfen**
  - Krankheitssymptome für NullPointerExceptions aufdecken
  - Architektur-Verletzungen erkennen
  - BugPatterns aufspüren
- **Was kann AOP noch?**
  - Testen vereinfachen / Mock-Ersatz
  - Code vereinfachen
- **Was kann AOP nicht?**
  - Testen ersetzen

Zu Risiken und Nebenwirkungen fragen Sie Ihren Arzt oder AOP'ler.

neugierig auf AOP?

- Oliver Böhm
  Aspektorientierte Programmierung mit AspectJ 5
  http://www.dpunkt.de/buch/3-89864-330-1.html
- die Seite zum Buch
  http://www.aosd.de
- AOP-Ecke
  http://www.agentes.de/aop/
- Ramnivas Laddad
  AspectJ in Action
  Manning Publications Co., 2003, ISBN 1-930110-93-6
- PatternTesting
  http://patterntesting.sf.net

Vielen Dank



agentes AG

Oliver Böhm

oliver.boehm@agentes.de

Telefon 0711/2012-2734
Telefax 0711/2012-6734

Räpplenstaße 17
70191 Stuttgart