

ex|Xcellent  
solutions



B.Schmid & O.Pehnke| eXXcellent solutions gmbh

# **Vital und fit bis ins hohe Alter: Refactoring im Projekt**

# Wie soll es sein – Entwurfsprinzipien der OO

---

## » Open-Closed Principle

Software-Einheiten sollten offen für Erweiterungen, aber geschlossen gegenüber Modifikationen sein

## » Dependency Inversion Principle (DIP)

Baue auf Abstraktionen, nicht konkreten Implementierungen

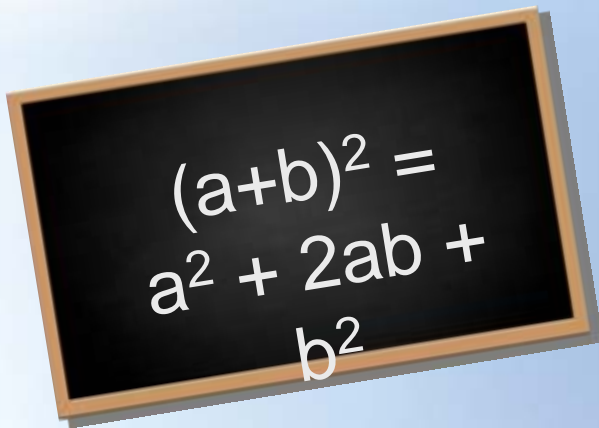
## » Law of Demeter (LoD)

„Sprich nur zu deinen nächsten Freunden“

# Was ist Refactoring?

*“A **change** to the system that leaves its behavior unchanged, but **enhances** some nonfunctional quality – simplicity, flexibility, understandability, performance”*

Kent Beck, „Extreme Programming Explained“ S.179


$$(a+b)^2 = a^2 + 2ab + b^2$$

# Refactoring ist ...

eine Transformation in isolierten Einzelschritten welche

» das **äußere Verhalten** der Software **beibehält**

» die **innere Struktur** der Software **verbessert**

Grundlagen: W. F. Opdyke., W.Cunningham, K.Beck (XP), M.Fowler

Refactoring ist aber nicht:

» Bugfixing

» Änderung der Funktionalität

→ „*The Two Hats*“ (Beck)

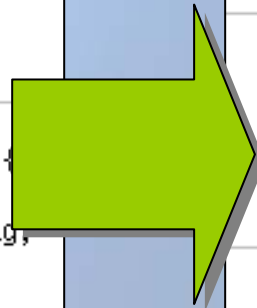


# Ein einfaches Motivations-Beispiel

F206: „Encapsulate Field“ (Fowler)

- » Symptom: Eine Klasse hat öffentliche Attribute
- » Änderung: private + Getter/Setter einführen

```
public class Konto {  
    public int saldo;  
    public Person inhaber;  
  
    public void ueberweise(int betrag,  
                           Konto ziel) {  
        this.saldo = saldo - betrag;  
        ziel.saldo = ziel.saldo + betrag;  
    }  
}
```



```
public class Konto {  
    private int saldo;  
    private Person inhaber;  
  
    public void ueberweise(int betrag,  
                           Konto ziel) {  
        this.setSaldo(saldo - betrag);  
        ziel.setSaldo(ziel.saldo + betrag);  
    }  
  
    public int getSaldo() {  
        return saldo;  
    }  
  
    public void setSaldo(final int saldo) {  
        this.saldo = saldo;  
    }  
}
```

# Ein einfaches Motivations-Beispiel

Welche Verbesserungen bringt dieses Refactoring?

Kapselung ist ein Grundpfeiler der OO:

- » **begrenzt Kontrolle** auf Objekt  
(→ Zuständigkeit)
- » erlaubt **isolierte Änderung**  
von Internas  
(→ Modularisierung)
- » **Klar definierte Schnittstelle**  
(→ bessere Wartbarkeit)
- » **Raum für zusätzliche Logik**  
(→ Erweiterbarkeit)



# Charakteristik von Refactoring

Refactoring bedeutet den Code



- » in **einfachen** und **kleinen Schritten** stufenweise **fehlerarm** zu ändern
- » ihn dabei **besser verstehen** zu lernen (Bugs!)
- » seine nicht-funktionale Aspekte zu **verbessern**
- » und damit seine **Wartbarkeit** zu begünstigen

# Welche Refactorings gibt es?

## Kategorien & Refactorings nach Fowler

» Composing Methods	( 9)
» Moving Features Between Objects	( 8)
» Organizing Data	(16)
» Simplifying Conditional Expressions	( 8)
» Making Method Calls Simpler	(15)
» Dealing with Generalization	(12)
» Big Refactorings	( 4)
	<hr/>
	72

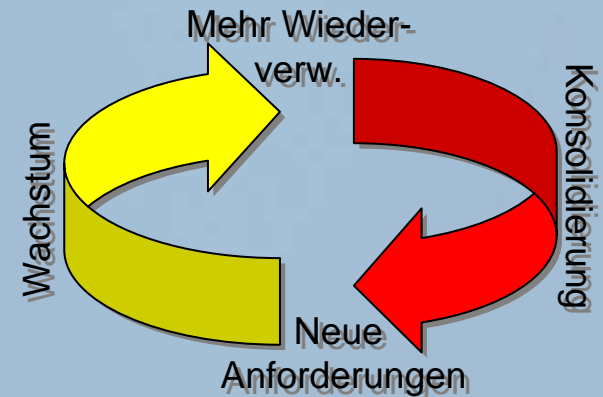
# Warum Refactoring?

## Die Theorie:



## Die Praxis:

- » Iterativ **wachsendes Verständnis** der Problem-Domäne
- » **Gutes Design** ist
  - im ersten Wurf extrem schwierig
  - erst durch die Implementierung verifiziert
  - Langfristige Entwicklung/Anforderungen oft unklar
- » Natürliche **Alterungsprozesse**



# Warum Refactoring?

---

*“**Grow**, don’t build software”*

Fred Brooks, “No Silver Bullet”

*„Using **patterns to improve** an existing design is better than using patterns early in a new design“*

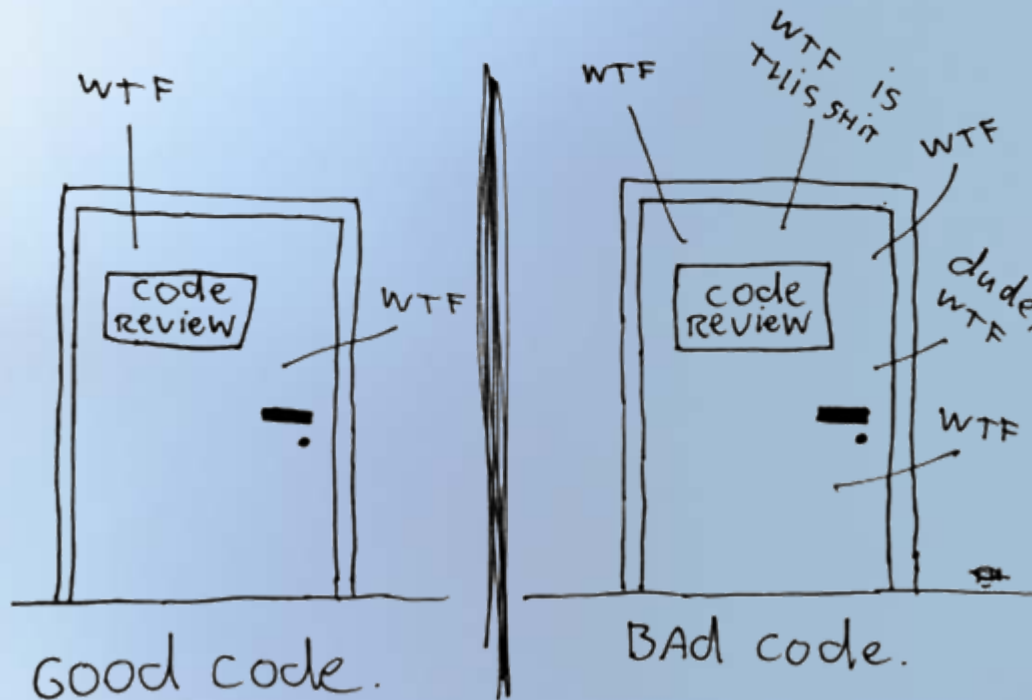
Joshua Kerievsky, “Refactoring to Patterns”

“Refactoring may be the single most important technical factor in achieving **agility**”

*Jim Highsmith, „Agile Software Development“*

# Welche Code refactoren?

The ONLY VALID MEASUREMENT  
OF code QUALITY: WTFs/minute



© osnews.com

# Welche Code refactoren?



„If it stinks, change it!“

Grandma Beck

**Wunsch:** Formale Indikatoren

**Praxis:** Erfahrung und Intuition erforderlich

Wegweiser:

- » Beck/Fowler: 22 „**Code Smells**“
- » Hotspots von Problemen & Bugs
- » Hohe Änderungsrate
- » Unklare Codeownership („Volle-Windel Effekt“)
- » Metriken & Bug Pattern Detectors

# Wann refactoren?

- » Beim Hinzufügen von Funktionalität, wenn...
  - diese nur schwer integrierbar ist
  - Code **wieder verwendet** werden kann
  - Zukünftige Änderungen leichter werden
- » Beim Bug-Fixing wenn ...
  - Code **unklar** oder **schwer nachvollziehbar** ist
- » Im Rahmen von Code-Reviews
  - Als Ergebnis der **Design-Verifizierung**
  - Als Plattform zum **Know-How Austausch**



# Nun in die Praxis ...

---



Eclipse 3.3.2 (Europa)



IntelliJ 7.0.3



# Vergleich: Automatische Refactorings der IDEs

Refactoring-Kategorie	Fowler	Eclipse	IntelliJ
Composing Methods	9	5	5
Moving Features Between Objects	8	2	3
Organizing Data	16	3	3
Simplifying Conditional Expressions	8	1	1
Making Method Calls Simpler	15	5	5
Dealing with Generalization	12	7	7
Big Refactorings	4	-	-
Andere Refactorings		6	11

- » Viele, nicht direkt-unterstützte Refactorings lassen sich über **Zwischenschritte** ebenfalls automatisiert erreichen!
- » Eclipse & IntelliJ zahlenmäßig gleichauf; der Teufel steckt jedoch im Detail!

# Vorbereitung

---

## Vor dem Refactoring:

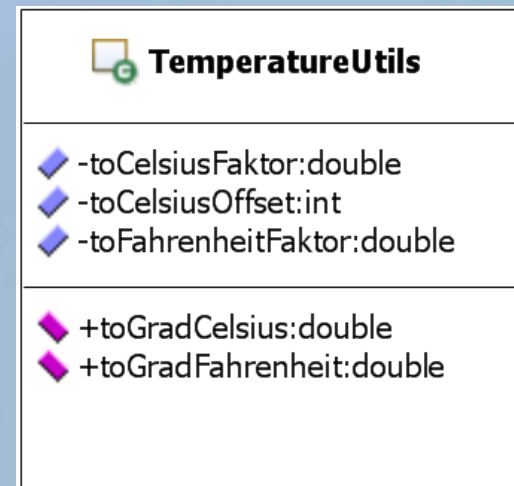
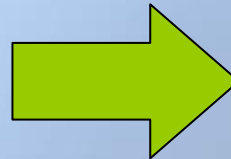
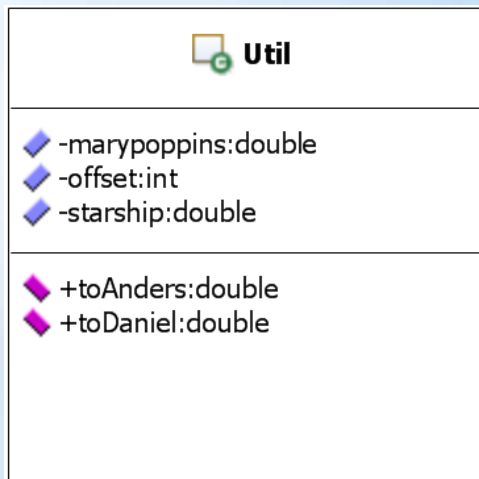
- » Läuft der Code?
- » Welche Regression-Test vorhanden?
- » Was ist das Problem?

## Und dann:

- » **Kleine Schritte**, eins nach dem anderen
- » Nach jedem Schritt **testen**, um sicher zu stellen keinen Fehler (Defekt) eingebaut zu haben
- » Falls ein Fehler auftritt, gibt es nur einen kleinen Bereich in dem er liegen kann

# Refactorings – Rename Method/Field/Class/...

👁️ Code Smells: *Block Comments, Speculative Generality, Alternative Classes with Different Interfaces*



Rename



# Live Beispiel #1

---

## » Refactoring in Eclipse: Marry Poppins

- Rename class
- Rename parameter
- Rename field
- including JavaDoc und Code links


## » Inklusive Unit-Tests!

-> Validierung vorher/nachher

# Code Smells und ihre Refactorings (1)

## *Long Method*

 **Unpassende Dekomposition des Problems**

 F110: Extract Method

F120: Replace Temps with Query

Live & in Farbe

## *Block Comments*

 **Erklärungsbedürftiger, vermutlich schlechter Code**

 F110: Extract Method

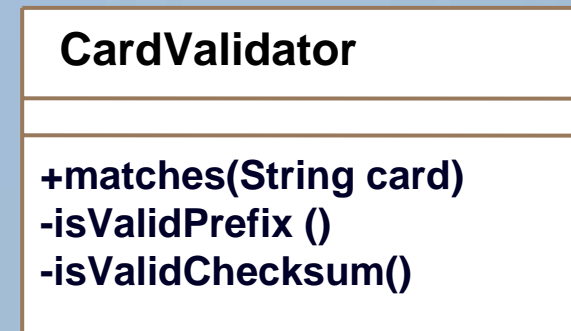
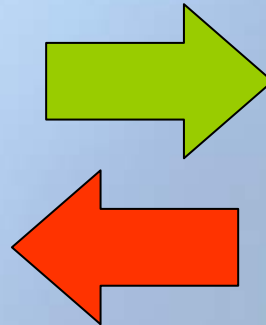
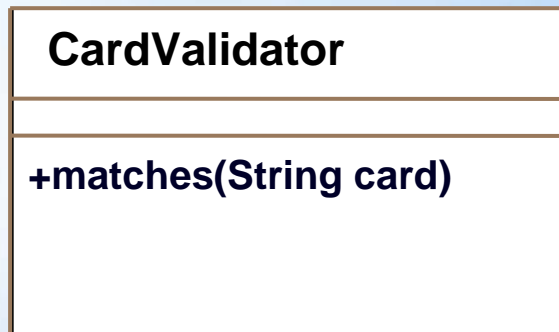
F273: Rename Method

F123: Introduce Explaining Variable

Live & in Farbe

# Refactorings - Extract & Inline Method

👁️ Code Smells: *Comments, Duplicated Code, Long Method, ...*  
*Middle Man*



Extract Method

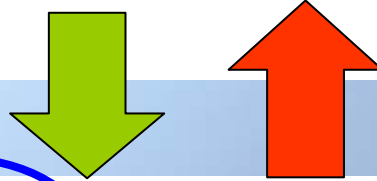
Inline



# Refactorings - Introduce & Inline Variable

👁️👁️ Code Smells: (n/a) [Codeduplikate, komplex. Logik]

```
if ((platform.toUpperCase().indexOf("MAC") > -1) &&  
    (browser.toUpperCase().indexOf("IE") > -1) &&  
    wasInitialized() && resize > 0) {  
    // do something  
}
```



```
final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;  
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;  
final boolean wasResized = resize > 0;  
  
if (isMacOs && isIEBrowser && wasInitialized() && wasResized) {  
    // do something  
}
```

Extract Variable

Inline



# Live Beispiel #2


---

## » Refactoring in Eclipse: StoreManager

1. „Extract Method“: Methode checkdigit
2. „Introduce Explaining Temp“: Variable prefix
3. „Decompose Conditional“: Methode checkprefix
4. Coment → „Extract Method“: checkCardNumber
5. „Inline Temp“: checkDigit  
*String checkDigit = Integer.toString(sum % 10);  
return checkDigit;*
6. Manuelle Aufarbeitung

# Code Smells und ihre Refactorings (2)

## *Duplicated Code*

 **Fehlende** Abbildung von Gemeinsamkeiten durch **Vererbung** oder **fehlende Delegation** an zuständiges Methode/Objekt



F110 Extract Method

F322 Pull up Method

F336 Extract Superclass

F149 Extract class

## *Long Parameter List*

 **Parameterobjekt fehlend** bzw. zu früh aufgelöst



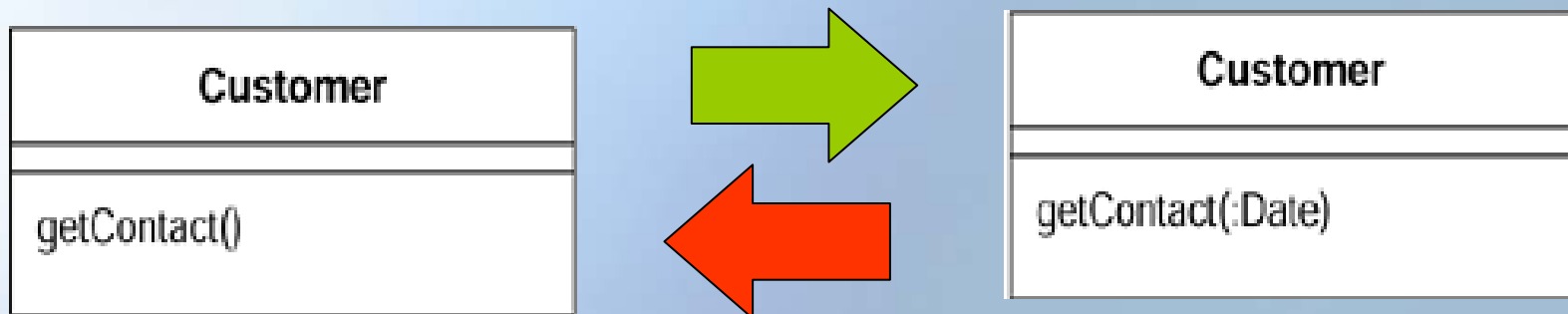
F288 Preserve Whole Object

F295 Introduce Parameter Object

← Live & in Farbe

# Refactoring – Add & Remove Parameter

👁👁 Code Smells: *Data Clumps, Long Method, Long Parameter List, Primitive Obsession, ... Speculative Generality*



Change Signature

Safe delete



# Refactorings - Preserve Whole Object

👁️👁️ Code Smells: *Long Method, Long Parameter List, Data Clumps*

» Kein direktes, automatisches Refactoring - aber über Zwischenschritte möglich!

```
◆ +checkBirthdays(personen: List<Person> ):void  
◆ -checkBirthdayAndSendGreetings(vorname: String, nachname: String, email: String, geburtstag: Date):void
```

» 1. Übergeordnetes Objekt als neuer Parameter

```
◆ +checkBirthdays(personen: List<Person> ):void  
◆ -checkBirthdayAndSendGreetings(vorname: String, nachname: String, email: String, geburtstag: Date, person: Person):void
```

» 2. Referenzen austauschen (name -> person.getName())

» 3. Unnötige Parameter nun entfernen

```
◆ +checkBirthdays(personen: List<Person> ):void  
◆ -checkBirthdayAndSendGreetings(person: Person):void
```

# Live Beispiel #3

---

- » Refactoring in IntelliJ: Birthday-Greeter
  1. „Change Method signature“ (neu: Person p)
  2. Ändere Methodenimplementierung auf p
  3. „Change Method signature“ / „Safe delete“
    - Entferne überflüssige Parameter

# Code Smells und ihre Refactorings (3)

## *Large Class / God Class*


 Klasse vereint zu viele Verantwortlichkeiten

 F330 Extract Subclass F341 Extract Interface

F149 Extract Class

 Live & in Farbe  
Live & in Farbe

## *Data Clumps*

 Zusammengehörende Daten sind nicht in einer gemeinsamen Klasse zusammengefasst

F288 preserve whole object

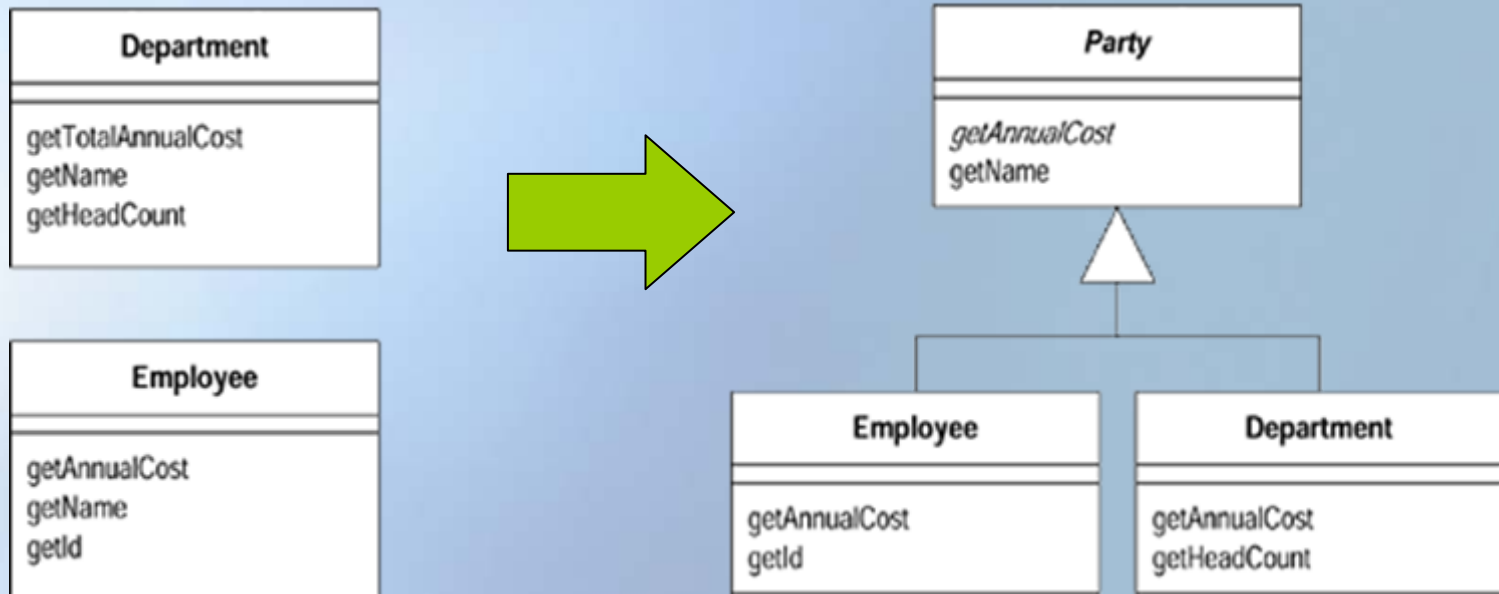
F149 Extract Class

F295 introduce parameter object



# Refactorings – Extract Superclass

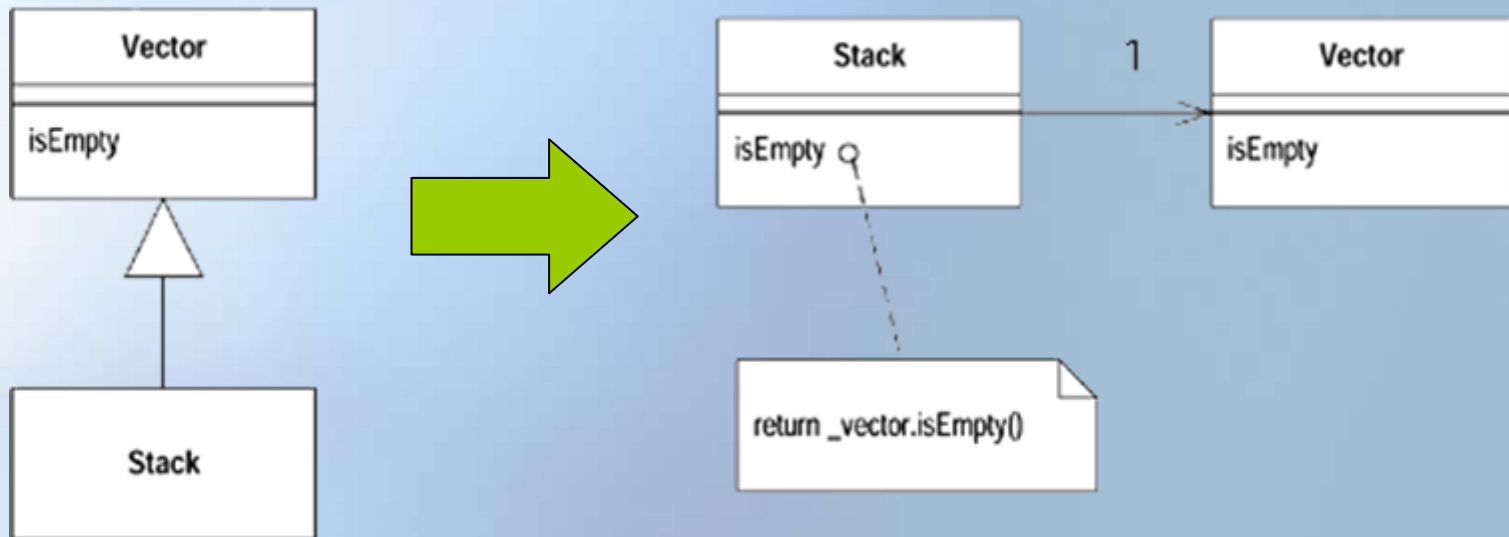
## 👁️ Code Smells: Large Class



» Motivation: Abbilden von Gemeinsamkeiten (gleichem Code) in gemeinsamer Überklasse

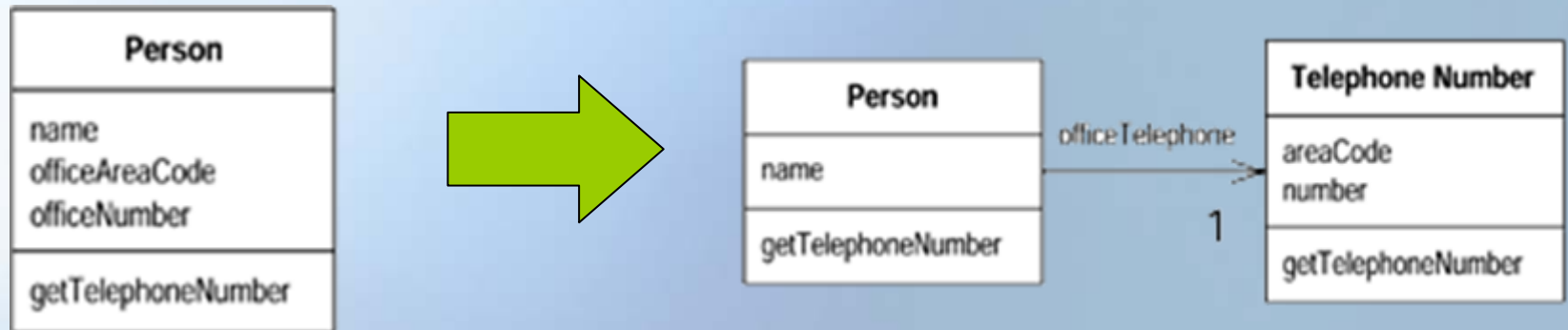
# Refactorings – Replace Inheritance with Delegation

👁👁 Code Smells: *Middle Man, Inappropriate Intimacy, ...*



# Extract Class

👁️👁️ Code Smells: *Duplicated Code, Large Class, Primitive Obsession, Divergent Change, ...*



# Live Beispiel #4

---

» Refactoring in IntelliJ: Person

» „Extract Class“ via

- „Extract Superclass“
- „Replace Inheritance via Delegation“
- „Inline method“

# Weitere Code Smells

## » Shotgun Surgery

Eine Erweiterung erfordert Änderungen an vielen, verschiedenen Klassen

## » Divergent Change

Eine Klasse muss für aus unterschiedlichen Gründe immer wieder modifiziert werden

## » Primitive Obsession

Primitive Datentypen erfüllen Funktionen fachlicher Klassen

## » Type Tests / Switch Statements

Varianten werden nicht durch Polymorphie abgebildet

## » Feature Envy

Eine Methode ist mehr an Elementen einer fremden Klasse, als an den eigenen interessiert

... und weitere!



# Die wertvollen Refactorings im Alltag



oder: Was Großmutter noch wusste!

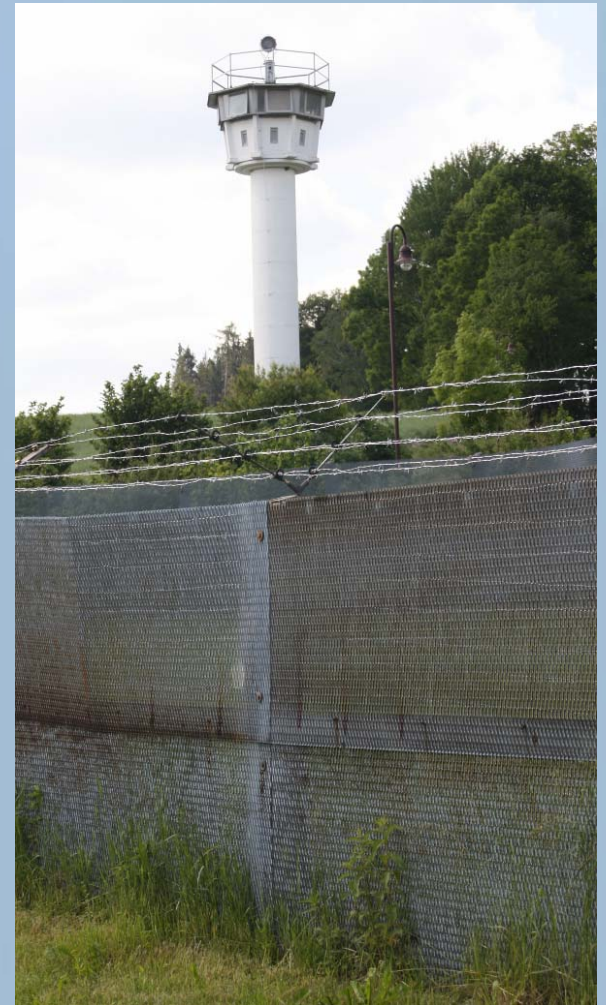
- » Renaming, Renaming, Renaming
- » Inline & Extract
- » Sichtbarkeit von Methoden & Attributen **reduzieren**  
(z.B. via ‚Search for Usage‘)
- » „final“ fields  
(Immutable Objects)

# Grenzen von Refactoring

- » Veröffentlichte Schnittstellen  
(→ API minimieren)
- » Datenbanken  
(→ Tool: Liquibase)
- » FUBAR Code  
(→ Neu gestalten)

## Grenzen autom. Refactorings:

- » MDSD / UML
- » XML (Struts, Spring, etc.)



# Fazit

---

## » Refactoring = Way to go!

- Heutzutage: **Hervorragende Werkzeugunterstützung**  
also: **Nur Mut!**
- Wichtig: Unit Tests für autom. Regressionstests
- Strenge **Typisierung** (OOD, Generics) hilft  
Defekte erkennen & vorbeugen

*„Nichts ist beständiger als der Wandel“*

*Heinrich Heine*

# Vielen Dank für Ihre Aufmerksamkeit!

## Besuchen Sie uns!

ex|Xcellent  
solutions

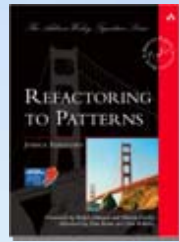
Direkt am Eingang der Ausstellung

<http://www.excellent.de>

B.Schmid@excellent.de



# Referenzen



## » „Refactoring object-oriented frameworks“,

W. F. Opdyke, 1992

<http://www.laputan.org/pub/papers/opdyke-thesis.pdf>

## » „Refactoring: Improving the Design of Existing Code“,

M. Fowler et al., 1999

## » “Refactoring to Patterns“,

J. Kerievsky, 2004

## » „Der 10-Punkte-Plan für den sicheren Weg zum nicht-wartbaren Code“,

B.Schmid, O.Pehnke, 2007

[http://www.excellent.de/download/WJAX07-Der\\_10\\_Punkte\\_%20Plan.pdf](http://www.excellent.de/download/WJAX07-Der_10_Punkte_%20Plan.pdf)



## » “Refactoring“,

<http://sourcemaking.com/refactoring>

Anonymous