

Validating the Object Calisthenics

Evaluation and Prototypical Implementation
of Tool Support

Agenda

- Wer bin ich?
- Die Object Calisthenics
- Vorstellung der neun Regeln
- Validierung der neun Regeln

Wer bin ich?

- Fabian Schwarz-Fritz
21 Jahre
- 6. Semester
Duales Studium bei SAP
- Studienarbeit an der
Dualen Hochschule



Karlsruhe

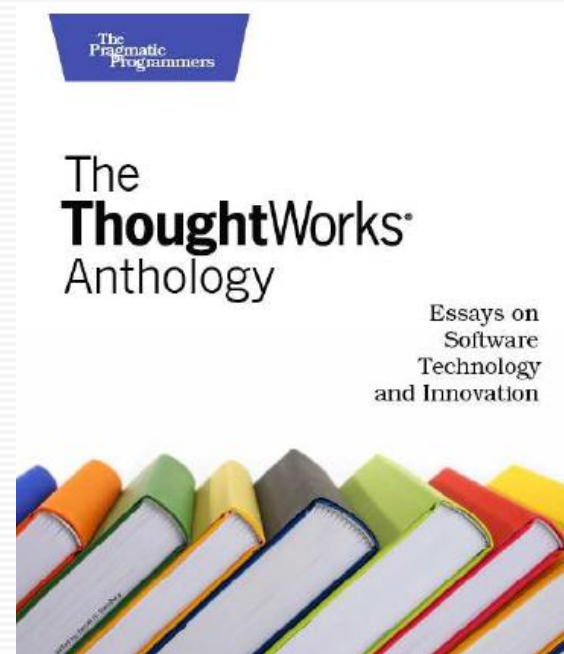
Object *Calisthenics*

- Calisthenics
(von griechisch kalos = „schön, gut“ und sthenos = „Kraft“)
- Gymnastikübungen
 - Extrem anstrengend
 - Einfaches Setup
- Liegestützen, Klimmzüge, Kniebeugen...



Object Calisthenics

- Objektorientierte Gymnastikübungen
- Im Training
 - 9 extreme Regeln
 - Kleines Projekt
 - Härteste Regelbefolgung ohne Zweifel
- Im Wettkampf
 - Erfahrung durch Training



ThoughtWorks Anthology 2008,
Essay „Object Calisthenics“
von Jeff Bay

Validating the Object Calisthenics

- Meine Studienarbeit:
Tool Support für die Regeln
untersuchen
- Hinweis auf Regelverstoß
- Erleichterung des Trainings



Verstoß:
Regel 1, Zeile 24,
Entwicklertag.java

Vorstellung der neun Regeln

1. Nur eine Einrücktiefe pro Methode!
2. Kein else Schlüsselwort!
3. Kapsle alle primitiven Datentypen und Strings!
4. Nur ein Punkt pro Zeile!
5. Keine Abkürzungen!
6. Kleine Entitäten!
7. Keine Klasse mit mehr als zwei Instanzvariablen!
8. Collections nur alleinstehend verwenden!
9. Keine Getter/Setter/Properties!

1. Nur eine Einrücktiefe pro Methode!

- Weniger Komplexität
- Gleiches Abstraktionslevel pro Methode
- „Do one thing“
- Wiederverwendbarkeit

```
public String board() {
    StringBuffer buf = new StringBuffer();
    collectRows(buf);
    return buf.toString();
}
void collectRows(StringBuffer buf) {
    for (int i = 0; i < 10; i++) {
        collectRow(buf, i);
    }
}
void collectRow(StringBuffer buf, int i) {
    for (int j = 0; j < 10; j++) {
        buf.append(data[i][j]);
    }
    buf.append("\n");
}
```


2. Kein else Schlüsselwort!

```
if(bedingung) {  
    // do something  
} else {  
    // do something else  
    if(anothercondition) {  
        //...  
    } else {  
        //..  
    }  
}
```

- Alternativen:
 - Early return
 - Polymorphismus



3. Kapsle alle primitiven Datentypen und Strings!

```
int ageFabian = 21;  
int dayInYear = 3600;  
int positiveValue = -100;
```



Problem:
Semantik nur über Namen

Compiler ausnutzen!

```
public void setDate(int year, int month, int day)  
public void setDate(Year year, Month month, Day day)  
setDate(15, 10, 1992);  
setDate(new Year(1992), new Month(10), new Day(15));
```

Programmlogik in neuen Datentypen

4. Nur ein Punkt pro Zeile!

```
String name = school.getClasses().get(classId).getStudents()  
                .get(studentId).getName();
```



- „Train Wreck Lines“:



```
String name = school.studentName(classId, studentId);
```



- Richtige Kapselung
- Erfüllt „Law of Demeter“
- Weniger komplex

5. Keine Abkürzungen!



Abkürzungen schwer
verständlich

- Klasse: `BpmPOrder`
- Klasse: `BusinessProcessManagerPurchaseOrder`
- Methode: `order.retrieveOrder()`

- Zwei Wörter für Klassen- und Methodennamen

- Klassen:

~~`BusinessProcessManagerPurchaseOrder`~~

- Methoden:

~~`order.retrieveOrder()`~~



Kontext verwendet

6. Kleine Entitäten!

- Keine Klasse mit mehr als 50 Zeilen
- Kein Paket mit mehr als 10 Klassen
- Kleine Einheiten
 - => besser wart- und testbar
 - => Wiederverwendung

7. Keine Klasse mit mehr als zwei Instanzvariablen

- Problem: Niedrige Kohäsion



- Feingranulares Objektmodell



```
class Name {  
    String firstname;  
    String middlename;  
    String lastname;  
}
```




```
class Name {  
    Surname family;  
    GivenNames given;  
}  
  
class Surname {  
    String family;  
}  
  
class GivenNames {  
    List<String> given;  
}
```

8. Collections nur alleinstehend verwenden

- `ArrayList customers = new ArrayList();` 
- Eine Instanzvariable: die Collection
- Operationen auf neuem Typ: `customers.sendOffer()` 
- => Neuer Typ hat Semantik

9. Keine Getter/Setter/Properties

- Datenkapselung 
- Schreibt „Tell, don't ask“ vor
- Änderung des Kontrollflusses

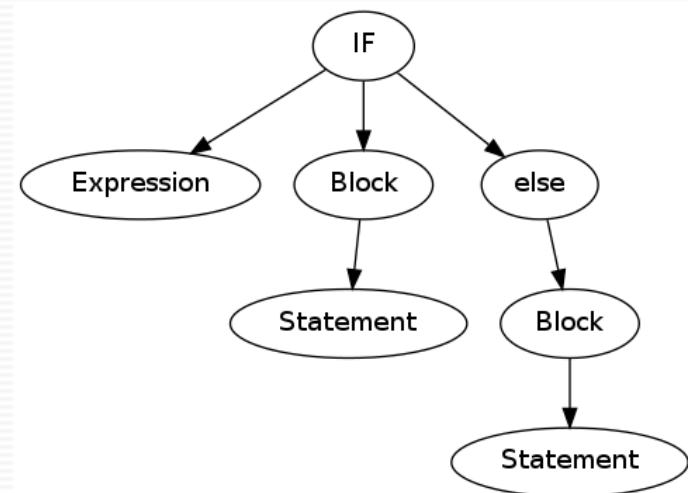
```
Speaker speaker = new Speaker();  
speaker.setVolume(100);  
display.show(speaker.getVolume());
```



```
Speaker speaker = new Speaker();  
speaker.turnup();  
speaker.adjustVolume(100);  
speaker.printOn(display);
```


Validierung der Regeln

- Validierung über Java Development Tools
- Analyse des Abstrakten Syntaxbaums (AST)
- AST stellt Syntax und Semantik als Baum zur Verfügung
- Visitor Pattern



Darstellung im Prototyp

```

5         if (true) {
6             StringBuilder theArgumentsBuilder = new StringBuilder();
7             for (String string : args) {
8                 theArgumentsBuilder.append(string);
9                 theArgumentsBuilder.append("/");
10            }
11        }




```

Information im Editor

Name	Message	Degree	Positi...	Resource
Rule 3 or 8	Wrap all th...	MIDDLE	8	HtmlElement.java
Rule 4	Using mor...	MIDDLE	14	HtmlHello.java
Rule 4	Using mor...	MIDDLE	25	HtmlHello.java
Rule 5	No Abbrev...	MIDDLE	5	IndentationCorrect.java
Rule 5	No Abbrev...	MIDDLE	3	IndentationCorrect.java
Rule 5	No Abbrev...	MIDDLE	6	IndentationWrong.java
Rule 5	No Abbrev...	MIDDLE	3	IndentationWrong.java
Rule 1	The given i...	MIDDLE	7	IndentationWrong.java
Rule 5	No Abbrev...	MIDDLE	3	InstanceVariableCountCorrect.java
Rule 7	More than ...	MIDDLE	3	InstanceVariableCountWrong.java
Rule 5	No Abbrev...	MIDDLE	7	InstanceVariableCountWrong.java
Rule 5	No Abbrev...	MIDDLE	3	InstanceVariableCountWrong.java

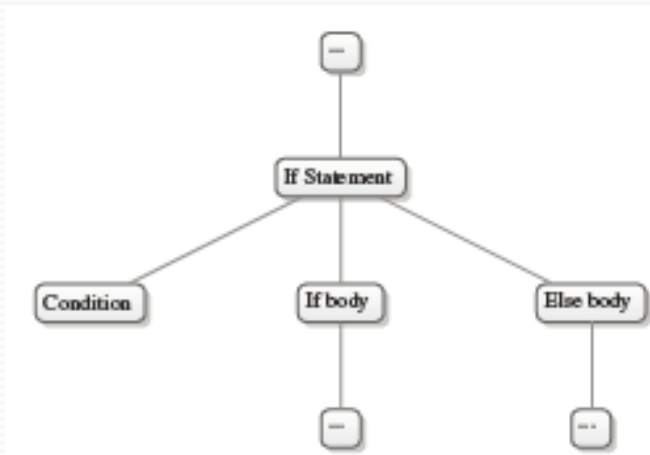
Information in Tabelle

1. Nur eine Einrücktiefe pro Methode!

Signatur  `public static void main(String[] args) {`
einrückendes Statement  `if (true) {`
einrückendes Statement  `for (String string : args) {`
`argumentsBuilder.append(string);`
`argumentsBuilder.append("/");`
`}`
`}`
`}`

- Schachtelung von Einrückenden Statements innerhalb einer Methode
- while, if, switch, for...

2. Kein else Schlüsselwort!



- Suche nach jedem `else` Schlüsselwort im Code

3. Kapselung aller primitiven Datentypen und Strings!

```
public class Position {  
    public int x;  
    public int y;  
  
    public Position(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public Position moveTo(Position position) {  
        int newX = x + position.x;  
        int newY = y + position.y;  
        return new Position(newX, newY);  
    }  
}
```

Kapselt int 

- **Wrapper Typen**

Enthalten nur den einen primitiven/String Typ

- **Nicht-Wrapper Typen**

Enthalten keinen primitiven/String Typ

4. Nur ein Punkt pro Zeile!

Statements „in einer Zeile“

```
String name = school.getClasses().get(classId).getStudents()  
    .get(studentId).getName();
```

Statements „in einer Zeile“

```
if(someobject.var.isEmpty()) {
```

*ExpressionStatement,
IfStatement,
AssertStatement...*

Ignorieren bestimmter Statements


```
this.someobject.method();  
someobject.method();
```

*ThisStatement,
QualifiedNames...*

5. Keine Abkürzungen!

- Regex
- Klassennamen
 - BoatRental
- Methodennamen
 - `rentSailboat()`;
- Keine Prüfung der Semantik
 - Variable:
`Day month = new Day()`
 - Klassename:
`AbcDef`

6. Kleine Entitäten!

- Formatierung nicht immer einheitlich 
- Zählen der Zeilen
- Zählen der Statements im Code
- Maximal 10 Klassen pro Paket

7. Keine Klasse mit mehr als zwei Instanzvariablen

- Zählen der Instanzvariablen einer Klasse

Dritte Instanzvariable



```
public class ManyInstancevariables {  
    private static Object staticObject;  
    private Object fieldOne;  
    private Object fieldTwo;  
    private Object fieldThree;  
}
```

8. Collections nur alleinstehend verwenden

- **Wrapper Typen**
Enthalten nur die Collection
- **Nicht-Wrapper Typen**
Enthalten keine Collection

Kapselt int

```
public class Customers {  
    private List<Customer> customers;  
  
    public void sendOffer(Offer offer) {  
        for (Customer customer : customers) {  
            customer.sendOffer();  
        }  
    }  
}
```

9. Keine Getter/Setter/Properties

```
public int getVolume() {  
    return volume;  
}  
  
public void setVolume(int volume) {  
    this.volume = volume;  
}
```

- Typische Struktur der Getter/Setter erkennen
- **Getter**: Rückgabe ist member
- **Setter**: Zuweisung Parameter zu member

Fabian Schwarz-Fritz
0049 172 7679803
fabian@schwarz-fritz.de
github/fabianschwarzfritz

Vielen Dank!

Validating the Object Calisthenics
Evaluation and Prototypical Implementation
of Tool Support



ENTWICKLERTAG

meet the **SPEAKER**
@speakerlounge



1. OG DIREKT ÜBER DEM EMPFANG