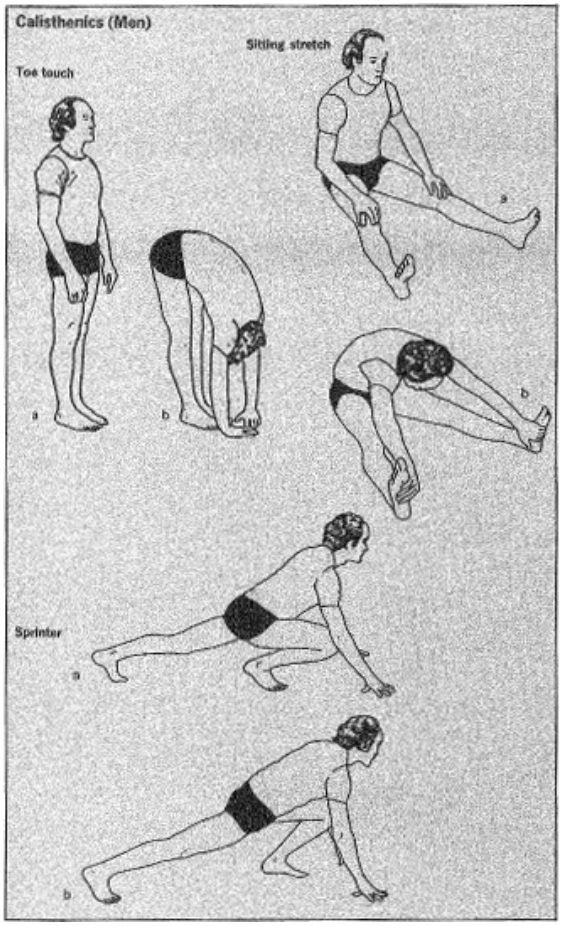
Object Calisthenics

Calisthenics

- einfache Gymnastikübungen ohne Hilfsmittel
- Training rein durch Körpergewicht
- Wird häufig als Grundlage für Gruppentraining verwendet
  - wurde in der Antike schon mit Tanzeinlagen des gegnerischen Heeres verwechselt



Objektorientierte Freiübungen (OC)

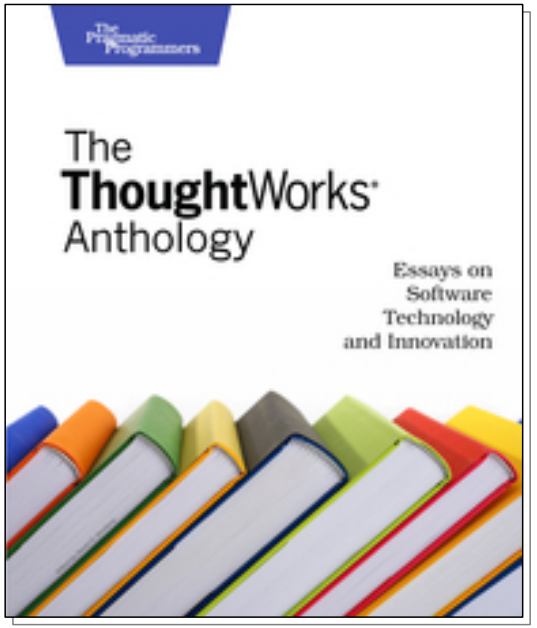
- Kleines Projekt (1k Zeilen)
- Härteste Regelbefolgung
- Zweifel hintenanstellen

```
2  public class PassByReference{
3      public static class Holder{
4          public String value;
5          public Holder( String value ){
6              this.value = value;
7          }
8      }
9
10     public static void passByReference( Holder holder ){
11         holder = new Holder( "test" );
12     }
13
14     public static void main( String[] args ){
15         Holder holder = new Holder( "unchanged" );
16         passByReference( holder );
17         System.out.println( holder.value );
18     }
19 }
```

- Danach: Als Richtlinien ansehen
- Mit erweiterten Möglichkeiten besseren Code schreiben

Herkunft

- Sammlung „Thoughtworks Anthology 2008“
  - Aufsatz „Object Calisthenics“ von Jeff Bay



OC – Die neun Regeln

1. Pro Methode nur eine Einrücktiefe
2. Kein else
3. Kapsele alle primitiven Datentypen und Strings
4. Nur ein Punkt (Methodenaufruf) pro Zeile
5. Keine Abkürzungen und trotzdem kurze Namen
6. Kleine Entitäten (50/10-Regel)
7. Maximal zwei Instanzvariablen pro Klasse
8. Collections nur alleinstehend verwenden
9. Keine Getter, Setter und verwandte Konstrukte (z.B. public Fields)

Pro Methode nur eine Einrücktiefe

- Verschachtelte Kontrollstrukturen arbeiten auf mehreren Abstraktionsstufen
- Reduzierung der Cyclomatic Complexity auf Minimum
- Methoden haben nur noch eine einzige Aufgabe
  - „do one thing“

Kein else

- Zahlreiche Alternativen
  - z.B. Frühe Methodenrückkehr
- Weniger Abhängigkeit von expliziten Kontrollstrukturen
  - Polymorphie kann als implizite Kontrollstruktur verwendet werden
- Zwischenschritt zum eigentlichen Ziel
  - Praktisch keine expliziten Kontrollstrukturen mehr
  - Siehe auch:

Kapsele alle primitiven Datentypen und Strings

- Primitive Datentypen arbeiten nur über den Namen
  - Und immer im vollen Wertebereich

```
int time = 60;
int timeInSeconds = 60;
int pleaseNoNegativeValues = -1;
```

- Durch eigene Datentypen kann der Compiler zur Mitarbeit bewegt werden

```
public void setDate(int year, int month, int day);
setDate(31, 3, 2014); // compiles but will fail at runtime

public void setDate(Year year, Month month, Day day);
setDate(new Day(31), new Month(3), new Year(2014)); // won't even compile
```

- Die neuen Datentypen können zugehöriges Verhalten sammeln

Nur ein Punkt pro Zeile

- Verhindert „Train Wreck Lines“

```
getPrinter().getTray(0).getProperties().forPage().getOrientation().width();
```

- Begrenzt Komplexität von Methoden implizit
- Zwischenschritt zum „Law of Demeter“
  - „Don't talk to strangers“

```
getPrinter().getPageWidthOfTray(0);
```

Verwende keine Abkürzungen

- Abkürzungen sind tödlich
- Klassen- und Methodennamen mit einem oder zwei Wörtern
- Statt order.shipOrder() reicht order.ship()

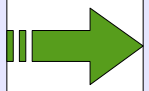
Kleine Entitäten (50/10-Regel)

- Keine Klasse über 50 Zeilen
  - Für Einsteiger: ohne Whitespace und Imports (Nettozeilen)
  - Für Profis: Bruttozeilen, also alles inklusive
- Kein Package mit mehr als 10 Klassen

Maximal zwei Instanzvariablen pro Klasse

- Aufspalten des Objektmodells
- Erfordert klares Verständnis der Problemdomäne
- Beispiel:

```
class Birthday {
  private Day day;
  private Month month;
  private Year year;
}
```



```
class Birthdate {
  private Birthday birthday;
  private Year year;
}
```



```
class Birthday {
  private Day day;
  private Month month;
}
```

Collections nur alleinstehend verwenden

- Collections sind Datentypen, die Gruppen von Elementen enthalten

```
List, ArrayList, LinkedList, DoubleLinkedList, Vector, etc.
Map, Dictionary, HashMap, AssociativeArray, etc.
Set, HashSet, TreeSet, LinkedHashSet, etc.
Arrays
```

- Operationen auf der Collection sind die einzigen Methoden des neuen Typs
- Collections enthalten zu wenig Semantik für Folgeleser

Keine Getter, Setter und verwandte Konstrukte (Properties, public Fields)

- Grundregel: Daten, die in eine Instanz geschrieben werden, werden aus dieser nicht mehr gelesen
- Führt direkt auf das Konzept „Tell, don't ask“
  - Fundamentale Umkehr des Datenflusses: Geben (lassen) statt Nehmen

```
final List<Student> students = loadStudents();
final Printer printer = acquirePrinter();
foreach (final Student each: students) {
  printer.print(each.registrationNumber() + ':' + each.name());
}
```

```
final List<Student> students = loadStudents();
final Printer printer = acquirePrinter();
foreach (final Student each: students) {
  each.printOn(printer);
}
```