Best Practices

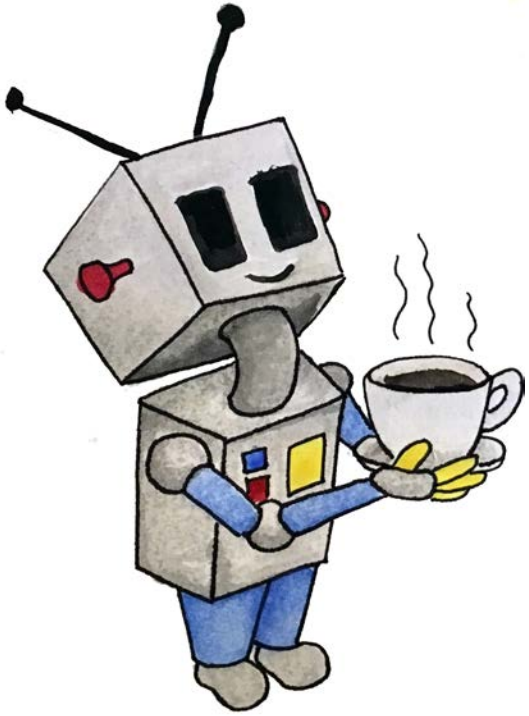
openHPI-Java-Team

Hasso-Plattner-Institut

Aufteilung von Klassen und Methoden

- Klassen und Methoden sollen jeweils eine abgeschlossene Einheit bilden

- Kontra-Indikatoren:
  - Lange, uneindeutige Bezeichner
  - Lange Methoden und Klassen
  - Seiteneffekte beim Aufruf



Kohäsion
Negativbeispiel

```
1 public class Parrot {
2       String name;
3       Parrot recentlyGreeted;
4
5       public String greet(Parrot other) {
6           recentlyGreeted = other;
7           return "Hallo " + other.name;
8       }
9 }
```

Kohäsion

- Maß für die Abgeschlossenheit einer Aufgabe
  - schwache / starke Kohäsion
- Kann nur schwer maschinell bewertet werden
- Seiteneffekte führen zu schlechterer Kohäsion

Kohäsion
Positivbeispiel

```
1  public class Parrot {
2      String name;
3      Parrot recentlyGreeted;
4
5      public String greet(Parrot other) {
6          return "Hallo " + other.name;
7      }
8
9      public void setRecentlyGreeted(Parrot other) {
10         recentlyGreeted = other;
11     }
12 }
```

Kohäsion

- Besser:
  - Aufteilung in einzelne Methoden
  - Klare Zuständigkeit für Methoden

Kohäsion
Positivbeispiel

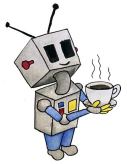
Hasso Plattner Institut

```
1  public class Parrot {
2      String name;
3
4      public String greet(Parrot other) {
5          if (other.isPaco()) {
6              return "Willkommen zurück, Paco!";
7          } else {
8              return "Hallo " + other.name;
9          }
10     }
11
12     public boolean isPaco() {
13         return name.equals("Paco");
14     }
15 }
```

Starke Kohäsion

- Führt zu kürzeren, spezifischeren Methoden

Kopplung
Negativbeispiel

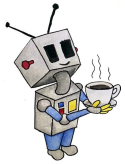


```
1 public class Robot {
2     public void identifyPaco(Parrot parrot) {
3         parrot.calculateWeight();
4         boolean isPaco = parrot.weight == 1.337;
5         if (isPaco) {
6             System.out.println("Paco gefunden!");
7         }
8     }
9 }
```

Kopplung

- Maß für den Abhängigkeit von Klassen untereinander
  - lose / enge Kopplung
- Direkten Zugriff auf Attribute vermeiden
- Genau Kenntnisse über die innere Funktionsweise der Klasse sollten nicht notwendig sein

Kopplung
Positivbeispiel



Hasso Plattner Institut

```java
public class Robot {
    private static final double PACOS_WEIGHT_IN_KILOGRAM = 1.337;

    public void identifyPaco(Parrot parrot) {
        if (parrot.getWeight() == PACOS_WEIGHT_IN_KILOGRAM) {
            System.out.println("Paco gefunden!");
        }
    }
}
```

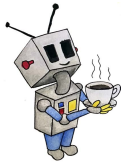
Kopplung

- Besser:
  - Kein Wissen über Berechnung / Aktualität des Gewichts
  - Kein direkter Attribut-Zugriff

Aufteilung von Klassen und Methoden

- Qualitätsmerkmale:
  - Kohäsion für die Abgeschlossenheit einer Aufgabe
  - Kopplung für die Abhängigkeit von Klassen untereinander
- Ziel: Starke Kohäsion, lose Kopplung
- Klassen und Methoden sollen jeweils eine abgeschlossene Einheit bilden

Diskussion: Return Early



```
1 [...]
2     public String greet(Parrot other) {
3         if (other.isPaco()) {
4             return "Willkommen zurück, Paco!";
5         } else {
6             return "Hallo " + other.name;
7         }
8     }
9 [...]
```

Return Early

- Mehrere `return` Statements pro Methode
- \+ Gedankliche Abarbeitung einiger (Sonder-)Fälle
- \+ Vermindert viele Verzweigungen (und dadurch Einrückungen)
- − Code möglicherweise schlechter nachvollziehbar durch mehrere `return`

Diskussion: this

```
1 public class Parrot {
2     String name;
3
4     public String getName() {
5         return this.name;
6     }
7 }
```

this

- Für Attribute meist nicht notwendig
- Konsequente Verwendung:
  - + führt zu eindeutigem Attribut-Zugriff
  - + vermindert Fehler beim Hinzufügen neuer Argumente
  - − redundant, mehr Code zum Erfassen