Deep Dive Java: Woche 4

openHPI-Java-Team

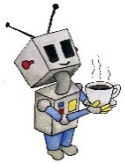
Hasso-Plattner-Institut

Inhalte Deep Dive Woche 4

HPI Hasso Plattner Institut

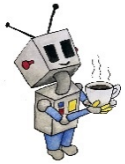
- Weitere Modifikatoren: `static` und `final`
- Konstantenlisten
- Wer oder was bin ich? - `instanceof` und `getClass()`
- `equals(…)` - reloaded
- Short Circuit Evaluation

Statische Methoden und Attribute (1/2)



- Schlüsselwort: `static`, anwendbar auf Attribute und Methoden
- Attribut bzw. Methode ist für die Klasse definiert, nicht für das Objekt
- Für statische Methoden gilt:
  - Es muss kein Objekt von der Klasse instanziiert werden
  - Der Aufruf erfolgt direkt auf der Klasse
  - Beispiele:
    - Math-Klasse: Math.min(…), Math.pow(…), …
    - Wrapper-Klassen: Double.parseDouble(…), …
  - Attribute des Objekts, können nicht gelesen oder geschrieben werden
  - `this.` kann in einem statischen Kontext **nicht** benutzt werden

Statische Methoden und Attribute (2/2)

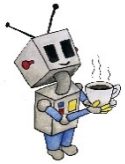


- Für statische Attribute gilt:
  - Alle Instanzen der Klasse (Objekte) greifen auf denselben Wert zu
  - Nicht-statische Methoden können statische Attribute lesen und schreiben
  - Beispiel:

```
1 public class Robot {
2    public static int roboCount;
3    public Robot() {
4       roboCount++;
5    }
6    public int numberOfRobots() {
7       return roboCount;
8    }
9 }
```

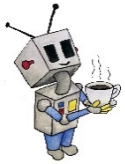
- Jeder neue Roboter erhöht den Zähler um eins (für alle Roboter)

Finale Klassen, Methoden und Konstanten



- Schlüsselwort `final`, anwendbar für Klassen, Attribute und Methoden
- Finale **Methoden** können nicht überschrieben werden
- Finale **Klassen** können nicht erweitert werden
- Finale **Attribute** können nach der Initialisierung nicht mehr neu beschrieben werden ➔ **Konstanten**
- Beispiel:
  - Integer.*MAX_VALUE*
  - `public final String PRODUCER = "Daniels Roboterfabrik";`
- Konventionen:
  - Konstanten schreibt man in Großbuchstaben
  - Underscore ersetzt CamelCase

Interfaces und Konstanten

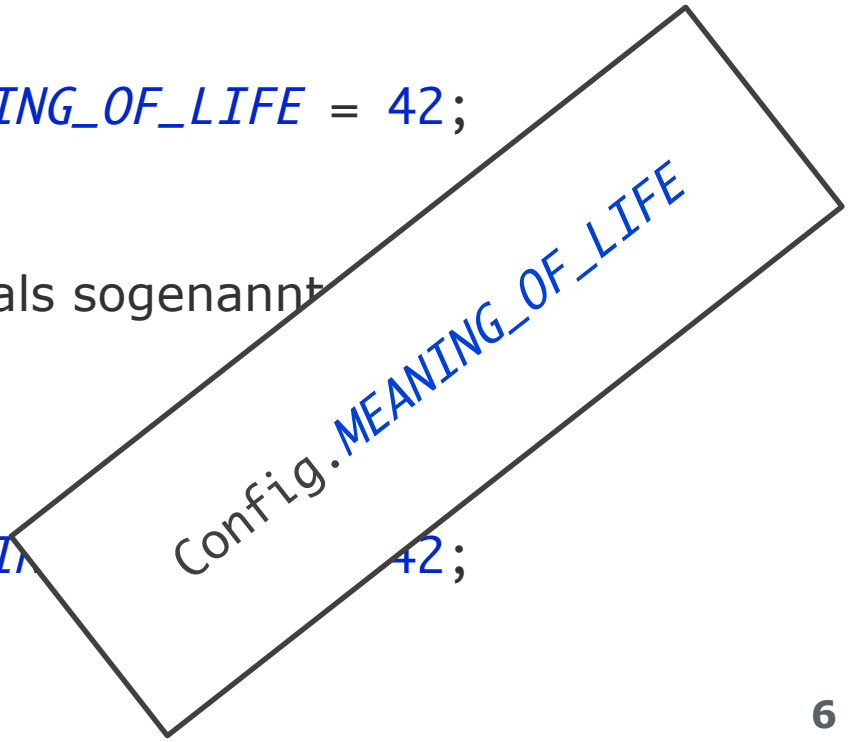


- Mögliche Elemente in Interfaces
  - Methoden: `public abstract`
  - Attribute: `public static final`
- Interfaces werden daher manchmal benutzt, um programmweite Konstanten zu definieren

```
1 public interface Config {
2    public static final int MEANING_OF_LIFE = 42;
3 }
```

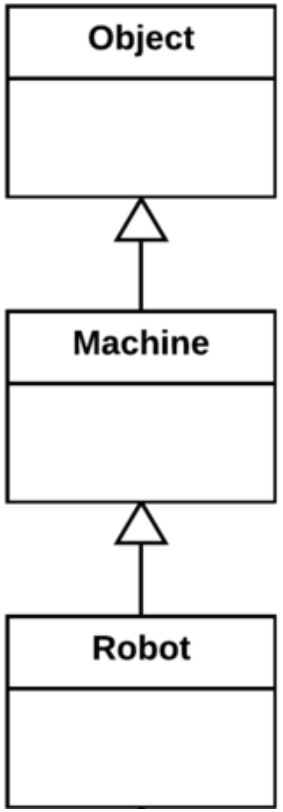
- Dies gilt aus verschiedenen Gründen als sogenannt
  (schlechter Stil). Bessere Lösung:

```
1 public final class Config {
2    private Config() {}
3    public static final int MEANI
4 }
```



Welchen Typ hat mein Objekt?
Aus welcher Klasse wurde mein Objekt instanziiert

- Manchmal will ich wissen welchen Typ mein Objekt hat bzw. aus welcher Klasse es instanziiert wurde
- Java bietet hierfür 2 Optionen
  - `instanceof` Operator
  - `getClass()` Methode



- Unterschiedliches Verhalten

```
1 Machine robot = new Robot("Ronja");
2 robot.getClass() ➔ Robot
3 robot instanceof Robot ➔ true
4 robot instanceof Machine ➔ true
5 robot instanceof Object ➔ true
6
7 Machine machine = new Machine("Ronja");
8 machine instanceof Robot ➔ false
```

Welchen Typ hat mein Objekt?
Aus welcher Klasse wurde mein Objekt instanziiert?

Hasso Plattner Institut

- Problem: `equals(...)` Methode

- Zur Erinnerung:

```
1 @Override
2 public boolean equals(Object o) {
3     // [...] sicherstellen dass o vom richtigen Typ ist
4
5     return (o != null) &&
6            this.name.equals(((Parrot) o).name) &&
7            this.age == ((Parrot) o).age;
  }
```

Welchen Typ hat mein Objekt?
Aus welcher Klasse wurde mein Objekt instanziiert?

HPI Hasso Plattner Institut

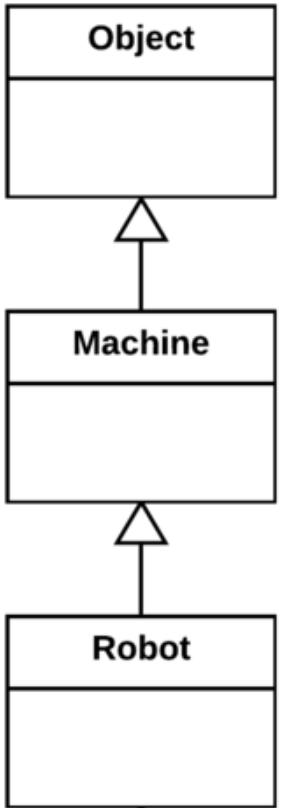
- **Jetzt richtig:**

```
1  @Override
2  public boolean equals(Object obj) {
3      if (this == obj) { return true; }
4      if (obj == null) { return false; }
5
6      // sicherstellen, dass obj vom richtigen Typ ist
7      // entweder
8      if (!(obj instanceof Robot)) { return false; }
9      // oder
10     if (getClass() != obj.getClass()){ return false; }
11
12     Robot other = (Robot) obj;
13     if (name == null) {
14             if (other.name != null) { return false; }
15     } else if (!name.equals(other.name)) { return false; }
16     return true;
   }
```

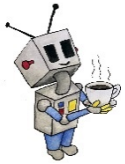
Welchen Typ hat mein Objekt?
Aus welcher Klasse wurde mein Objekt instanziiert?

- Problem: Beide Lösungen sind nicht optimal

```
Machine machine = new Machine("Ronja");
Machine robot = new Robot("Ronja");

if (!(obj instanceof Robot)) { return false; }

machine.equals(robot) ➔ true
robot.equals(machine) ➔ false


if (getClass() != obj.getClass()) { return false; }

machine.equals(robot) ➔ false
robot.equals(machine) ➔ false
```



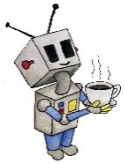
Short Circuit Evaluation



- Wiederholung Boolsche Ausdrücke (Wahr oder Falsch):
  - UND (&&) gesamter Ausdruck ist falsch, wenn ein Teil falsch ist
  - ODER (||) gesamter Ausdruck ist wahr, wenn ein Teil wahr ist
  - Java nutzt das aus, um Bedingungen abzubrechen, die nicht mehr **wahr**, bzw. **falsch** werden können
- Short Circuit Evaluation (deutsch: Kurzschlussauswertung)
  - &&-verknüpfte Ausdrücke werden bei erstem falschen Teilausdruck abgebrochen
  - ||-verknüpfte Ausdrücke werden bei erstem richtigen Teilausdruck abgebrochen
  - Geschickte Anordnung der Teilausdrücke kann die Abfragen vereinfachen

```
return (o != null) &&
        this.name.equals(((Parrot) o).name);
```

Programming Toolbox



Hasso Plattner Institut