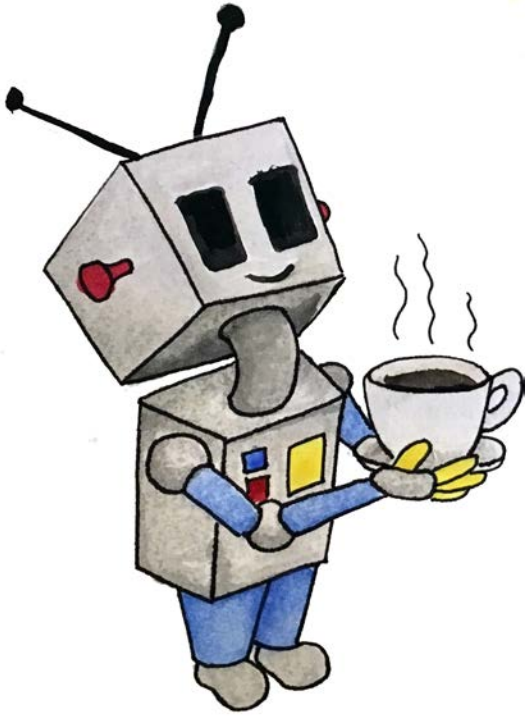
Konventionen in Java

openHPI-Java-Team

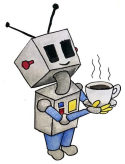
Hasso-Plattner-Institut

Motivation Konventionen

- Vereinbarungen zur
  - Strukturierung des Codes
  - Benennung von Klassen und Variablen
  - Erklärung von Codestellen mit Kommentaren
- Ziel: Navigation im Code erleichtern
- Empfehlungen, keine Pflicht



Einfachere Lesbarkeit

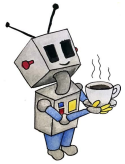


```java
public class Robot {
    public static void main(String[] args) {
        System.out.println("Hallo!");
    }
}
```

```java
public class Robot{public static void
main(String[]args){System.out.println("Hallo!");}}
```

- Code-Formatierung hilft:
  - beim schnellen Erfassen des Codes
  - beim Vermeiden von typischen Fehlern

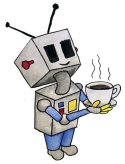
Aufbau einer Java-Datei



- Nur eine Klasse pro Datei
- Dateiname = Klassenname

- Reihenfolge der Elemente:
  1. Konstanten
  2. Attribute
  3. Konstruktoren
  4. Methoden
  5. Getter / Setter für Attribute

Aufbau einer Java-Datei



- Reihenfolge der Attribute nach Sichtbarkeiten:
  1. `public`
  2. `protected`
  3. (default)
  4. `private`

- Separate Zeilen für:
  - Jede Anweisung mit Semikolon ;
  - Neue Code-Blöcke { }
  - Bedingungen in Schleifen und Verzweigungen
- Jeweils eine Leerzeile zwischen den verschiedenen Methoden

Parrot.java

HPI Hasso Plattner Institut

```
1  public class Parrot {
2      private String name;
3
4      Parrot(String name) {
5          this.name = name;
6      }
7
8      public String greet(Parrot other) {
9          return "Hallo " + other.name;
10     }
11
12     public String getName() {
13         return name;
14     }
15 }
```

Leerzeilen, Leerzeichen und Einrückung

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

Leerzeilen

- Nach den Attributen
- Nicht zwischen verschiedenen Attributen

- Zwischen verschiedenen Methoden
- Innerhalb von Methoden zur logischen Gruppierung von Code

Leerzeilen, Leerzeichen und Einrückung

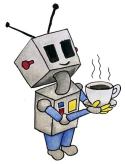
Hasso Plattner Institut

```
1  public class Parrot {
2      String name;
3
4      Parrot(String name) {
5          this.name = name;
6          if (name.equals("Paco")) {
7              System.out.println("Willkommen zurück!");
8          }
9      }
10 }
```

Leerzeichen

- Vor { und um Operatoren = und Vergleiche ==, <, <=, ...
- Vor (und hinter) bei Kontrollstrukturen
- Nicht bei Methoden, also nicht: System. out .println ("Paco");

Leerzeilen, Leerzeichen und Einrückung

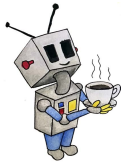


```
1 public void countdown() {
2     for (int i = 10; i > 0; i--) {
3         if (i == 5) {
4             System.out.print("Halbzeit! - ");
5         }
6         System.out.println(i);
7     }
8     [...]
9 }
```

Einrückung

- Geschweifte Klammern auch für einzeilige Code-Blöcke nutzen, z.B. `if`
- Code-Blocke erhöhen jeweils die Einrückung
- Standardmäßig vier Leerzeichen verwenden

Leerzeilen, Leerzeichen und Einrückung



```
1 int calculateRemainingRuntime(Battery battery,
2                                Task task, int speed) {
3     if (battery == null || task == null || speed == 0
4             || this.canPerfom(task) == false) {
5         return 0;
6     }
7     [...]
8 }
```

Einrückung

- Einzelne Zeilen nicht zu lang, im Zweifel manuell umbrechen:
  - Auflistung auf selber Höhe oder mit acht Leerzeichen fortsetzen
  - Operatoren in die neue Zeile
  - Sonderfall Umbruch in `if`-Bedingung: Immer mit acht Leerzeichen

Benennung

```java
public class Parrot {
    private static final int MAX_NAME_LENGTH = 20; //chars
    private static final int MAX_WEIGHT_IN_KILOGRAM = 5;
    private String name;

    [...]
}
```

- Benennung von Variablen und Attributen in CamelCase
- Konstanten:
  - mit `final` kennzeichnen
  - Namen GROSS_SCHREIBEN
  - Direkt zu Beginn einer Klasse, vor Attributen
  - Rechenweg aufschreiben

Benennung

Hasso Plattner Institut

```
1 public class Parrot {
2     private static final int MAX_NAME_LENGTH = 20; //chars
3     private static final int MAX_WEIGHT_IN_KILOGRAM = 5;
4     private String name;
5
6     [...]
7 }
```

- Benennung von Variablen, Attributen und Konstanten:
  - Kurz und aussagekräftig!
  - Ohne Angabe des Typs (also nicht `nameString`)
  - Sollten nicht mit $ oder _ beginnen
  - Ein-Buchstaben-Variablen (`i`, `j`, k, …) nur temporär, z.B. in Schleifen
  - Eigene Zeile pro Variable;
- Möglichst direkt initialisieren

Benennung

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

- Benennung von Klassen:
  - Sollten Nomen sein, Akronyme vermeiden
  - Erster Buchstabe groß und CamelCase
- Benennung von Methoden:
  - Sollten auf Verben basieren
  - Klein beginnend und mit CamelCase

Kommentare für Dokumentationen

```
1  public class Parrot {
2      String name;
3
4      /**
5       *  Returns a greeting for another Parrot
6       *  @param  other  the other Parrot to greet
7       *  @return        the greeting
8       */
9      public String greet(Parrot other) {
10         return "Hallo " + other.name;
11     }
12 }
```

JavaDoc

- Mehrzeilige Kommentare, die mit `/**` beginnen
- Standardisiertes Format mit `@param`, `@return`
- wird zur Generierung von Klassen-Dokumentationen genutzt