

# Praktikum „SEP: Java-Programmierung“

## WS 2018/19

### Programmierstil

Thomas Lemberger und Martin Spieß  
Basierend auf Folien von Matthias Dangl und Karlheinz Friedberger

# Camel Code

```

    $_='ev
    al("seek\040D
ATA,0,
    0;");foreach(1..3)
    {<DATA>;}my
    @camel1hump;my$camel;
my$camel ;while(
    <DATA>){$_=sprintf("%-6
9s",$_);my@dromedary
    =<DATA>)}{@camel1hum
    p=split(/);if(defined($
ry1){my$camel1hump=0
    p=split(/);while(@dromeda
    );my$CAMEL=3;if(defined($=shif
    t(@dromedary1
    ))&&/\S/){$camel1hump+=1<<$CAMEL;}
    $CAMEL--;if(d
    efined($=shift(@dromedary1))&&/\S/){
    $camel1hump+=1
    <<$CAMEL;}$CAMEL--;if(defined($=shift(
    @camel1hump))&&/\S/){$camel1hump+=1<<$CAMEL;}$CAMEL--;if(
    defined($=shift(@camel1hump))&&/\S/){$camel1hump+=1<<$CAME
    L;};$camel=(split(/,"\\040..m`{/J\047\134\L^7FX"}))[$camel1h
    ump];$camel=""\n";}@camel1hump=split(/n/, $camel);foreach(@
    camel1hump){chomp;$CAMEL=$_;y/LJF7\173\175\047/\061\062\063\
    064\065\066\067\070;/y/12345678/JL7F\175\173\047`/;$_=reverse;
    print"$_\040$CAMEL\n";}foreach(@camel1hump){chomp;$CAMEL=$_;y
    /LJF7\173\175\047/\12345678;/y/12345678/JL7F\175\173\0
    47`/;
    $_=reverse;print"$040$CAMEL\n";};s/s\*/g;eval; eval
    ("seek\040DATA,0,0;");undef$;$_=<DATA>;s/s\*/g;(
    );s
    ;^.*_;;map{eval"print"$_\n";}/.}/g; __DATA__ \124
    \1
    50\145\040\165\163\145\040\157\1
    46\040\1
    41\0
    40\143\141
    \155\145\1
    54\040\1
    51\155\
    141
    \147\145\0
    40\151\156
    \040\141
    \163\16
    3\
    157\143\
    151\141\16
    4\151\1
    57\156
    \040\167
    \151\164\1
    50\040\
    120\1
    45\162\
    154\040\15
    1\163\
    040\14
    1\040\1
    64\162\1
    41\144
    \145\
    155\14
    1\162\
    153\04
    0\157
    \146\
    040\11
    7\047\
    122\1
    45\15
    1\154\1
    54\171
    \040
    \046\
    012\101\16
    3\16
    3\15
    7\143\15
    1\14
    1\16
    4\145\163
    \054
    \040
    \111\156\14
    3\056
    \040\
    125\163\145\14
    4\040\
    167\1
    51\164\1
    50\0
    40\160\
    145\162
    \155\151
    \163\163
    \151\1
    57\156\056
```

Source:

[https://www.perlmonks.org/?node\\_id=45213](https://www.perlmonks.org/?node_id=45213)

- ▶ Programme sollen nicht nur funktionieren sondern auch lesbar sein
  - ▶ Können andere den Code verstehen? Stichwort: Teamarbeit!
  - ▶ Kann man selbst den Code noch verstehen in zwei Stunden/Wochen/Monaten/Jahren?
  - ▶ Quelltext ist die einzig zuverlässige Dokumentation
  - ▶ Quelltext wird (häufig) mit ausgeliefert
  - ▶ JAVA Coding Conventions (JCC) → internationaler Standard
  - ▶ Revisionsverwaltung (z.B. SVN, GIT)

- ▶ Sprechende Bezeichner
- ▶ Qualität der Dokumentation: Genügend, sinnvoll und korrekt
- ▶ Formatierung/Layout unterstützt Programmstruktur
- ▶ Sinnvolle und feingranulare Aufteilung, keine Monstermethoden oder Monsterklassen
- ▶ Geheimnisprinzip (Information Hiding)
- ▶ Erweiterbarkeit
- ▶ Das Eine-Zuständigkeit-Prinzip (Single Responsibility): Jede Klasse hat **eine** Aufgabe
- ▶ Konsistenz

- ▶ Sprechende Bezeichner: Nicht `int a, b, c`; sondern `int length, width, height`;
- ▶ Einzelne Buchstaben nur in trivialen Fällen, z.B. Laufindex `i` oder `double mean(int x, int y)`
- ▶ Keine Typinformationen im Namen wie z.B. `int lengthInt`; oder `Node rootNode`; → überflüssig, da über die IDE jederzeit einsehbar!
- ▶ Groß- und Kleinschreibung in Java:
  - `StringBuilder` Klasse, Substantiv, jeder Wortteil beginnt mit Großbuchstaben
  - `childCount` Variable, Substantiv, erster Wortteil klein
  - `getPoints` Methode, Verb, erster Wortteil klein
  - `NUMBER_OF_CHARS` Konstante, nur Großbuchstaben und Unterstriche

- ▶ Ziel: Erklären, **was** mit **wem** gemacht wird (und ggf. **warum**), wenn es **Mehrwert** bringt und **nicht offensichtlich** ist
- ▶ Sinnlos: `frobulateWidgets(); // frobulates the widgets`
- ▶ Möglichkeiten für Dokumentation:
  - ▶ JAVADOC
  - ▶ Kommentare
  - ▶ Code
- ▶ Dokumentation und Kommentare **aktuell** halten (schwierig!)
- ▶ **Lügende Dokumentation ist noch schlimmer als gar keine!**

# Dokumentation: JavaDoc

- ▶ Für Methoden und Klassen
- ▶ **Was** passiert
- ▶ **Nicht wie** ist es umgesetzt
- ▶ Mindestens alles, was `public` und `protected`
- ▶ **Immer**, wenn Methodenname nicht ausreicht, um Verhalten zu verstehen

Good:

```
/**
 * Check if enough tokens are
 * provided. If not, a
 * corresponding error message
 * is printed.
 *
 * @param ...
 */
boolean hasCorrectLength(
    String [] tokens,
    int expectedLength) {
    ...
}
```

Bad:

```
/**
 * Check if enough tokens are
 * provided by calling
 * tokens.len and checking that
 * it is the same value as
 * expectedLength.
 * If not, we use
 * System.out.println and
 * both values to print a
 * meaningful error message.
 * ...
 */
```

- ▶ Kommentare für zusätzliche Informationen innerhalb von Methoden
- ▶ **Vor** dem Kommentierten
- ▶ **Kurze** Kommentare am Ende der selben Zeile
- ▶ **Wichtig: Wieso** etwas (so) gemacht wird
- ▶ „Was mit wem“ gemacht wird sollte schon in Javadoc stehen!

```
try {  
    // points is a user-input and hasn't been checked  
    // yet, so it may not be a valid String and  
    // a NumberFormatException can be thrown here  
    return Integer.parseInt(points);  
} catch (NumberFormatException e) {  
    showError("Could not parse number_" + points + ".");  
}
```



# Layout

- ▶ Nicht mehr als eine Anweisung pro Zeile
- ▶ Zeilen nicht zu lang werden lassen (z.B. maximal 80 oder 100 Zeichen)
- ▶ Leerzeilen
- ▶ Einrückung (Leerzeichen, keine Tabs)
- ▶ Konsistente Positionen von Klammern
- ▶ Boolesche Variablen wenn möglich positiv formulieren

```
public static void main(String[] args) throws IOException {
    BufferedReader in
        = new BufferedReader(new InputStreamReader(System.in));
    boolean run = true;

    while (run) {
        ...
    }
    ...
}
```

- ▶ Mehrwertige Boolesche Ausdrücke immer eindeutig Klammern
- ▶ Booleans nie gegen Konstanten `true` oder `false` vergleichen
- ▶ Wo und wie werden überlange Zeilen umgebrochen:
  - ▶ Hinter Kommata und öffnenden Klammern
  - ▶ Vor Operatoren
  - ▶ Besser „außen“ als „innen“ bei geschachtelten Ausdrücken
  - ▶ Neue Zeile nach umgebendem Ausdruck ausrichten
  - ▶ Konkatenation (+) bei überlangen Zeichenketten (z.B. bei `private static final String HELP_TEXT = "...)`
- ▶ Methodenlänge (Richtwert): Nicht mehr als ca. 20 Zeilen (oft weniger)
- ▶ Klassenlänge: Schwieriger festzulegen, aber nicht mehr als 1000 Zeilen

# Geheimnisprinzip (Information Hiding) und Zugriffsmodifikatoren

- ▶ Jedes Attribut, jede Methode, jeder Konstruktor nur so lokal wie möglich zugreifbar
- ▶ Nur was geheim ist, kann beliebig verändert werden
- ▶ Nur was geheim ist, ist kontrollierbar
- ▶ Jeder (potentielle) Zugriff potenziert die Komplexität
- ▶ Lesezugriff auf Attribute von außen nur über Getter und nur wenn nötig: `getXXX`, `isXXX`, `hasXXX`
- ▶ Schreibzugriff auf Attribute von außen nur über Setter und nur wenn unbedingt nötig: `setXXX`
- ▶ Variablen, die nach der Initialisierung nie mehr verändert werden immer `final`
- ▶ Statische Attribute sollten immer `final` sein
- ▶ Idealerweise sind Objekte unveränderlich (immutable), z.B. `String` (in Aufgabe 1 aber nur schwer umsetzbar)

- ▶ Grundprinzip: Antizipation des Wandels
- ▶ **Naheliegende** Veränderungen müssen leicht umsetzbar sein
- ▶ Umsetzung durch
  - ▶ Geheimnisprinzip (siehe vorherige Folie)
  - ▶ Vermeidung von (insbesondere globalem oder veränderbarem) Zustand wo immer möglich (siehe vorherige Folie)
  - ▶ Verwendung gängiger Programmiermuster
  - ▶ Vermeidung von Redundanz: DRY (don't repeat yourself)
  - ▶ Sprechende Konstanten statt „Magic Numbers“
- ▶ Nicht sinnvoll: Programmieren für in der Zukunft **vielleicht** notwendige Features (YAGNI: you ain't gonna need it, KISS: Keep it simple, stupid)

- ▶ Jon Bentley: „Bumper-Sticker Computer Science“ (kurze Tipps zu Programmierung & Co.):  
[http://www.bowdoin.edu/~ltoma/teaching/cs340/spring05/coursestuff/Bentley\\_BumperSticker.pdf](http://www.bowdoin.edu/~ltoma/teaching/cs340/spring05/coursestuff/Bentley_BumperSticker.pdf)
- ▶ Google Java Code Style:  
<http://google.github.io/styleguide/javaguide.html>
- ▶ Joshua Bloch: Effective Java (3rd Edition!)