University of Basel

Best Practices

Rahel Arnold <rahel.arnold@unibas.ch>
Cristina Illi <cristina.illi@unibas.ch>
Nikodem Kernbach <nikodem.kernbach@unibas.ch>
Simon Peterhans <simon.peterhans@unibas.ch>
Jan Schönholz <jan.schoenholz@unibas.ch>

Departement Mathematik & Informatik, Universität Basel

Bugs

Ein Programmfehler [...], häufig auch Bug [...] genannt, bezeichnet im Allgemeinen ein Fehlverhalten von Computerprogrammen.

Quelle: Wikipedia

- Softwarefehler kosteten Mittelstands- und Grossunternehmen 2006 circa 84,4 Mrd. **allein in Deutschland**![1]
- Bugs kommen in **jedem** Softwareprojekt vor!
- Aufgeschoben $\neq$ Aufgehoben...

[1] `https://de.wikipedia.org/wiki/Programmfehler`

Bugs vermeiden

- **D**on't **R**epeat **Y**ourself
- **Y**ou **A**int **G**onna **N**eed **I**t
- Häufige Code Reviews (IntelliJ *inspect code* Funktion)
- Automatisierte Tests (Unit Tests)
- So oft wie möglich "von Hand" testen (GUI, ...)
- **Coding Conventions**

Coding Conventions

Ziel: Einheitlicher Programmierstil für verbesserte Lesbarkeit mittels

- Naming Conventions
- Package-Struktur (Conventions für Java beachten!)
- Einrückungsstil
- Klammerpositionen
- ...

Nutzen von Coding Conventions

Verbesserte Lesbarkeit bringt einige Vorteile mit sich:

- Bessere Code-Übersicht
- Einfachere Fehlersuche
- Einfacheres Refactoring
- Einfacheres Arbeiten im Team

Imports

- Keine manuellen Wildcard Imports
  - Beispiel: java.awt.* und java.util.* haben beide eine Klasse **List**!
  - Moderne IDEs (IntelliJ, Eclipse) verstecken Imports ohnehin
- Keine ungebrauchten Imports (automatisches Cleanup unterstützt in modernen IDEs)

Klassen-Deklaration

Eine top-level Klasse pro Datei!

Beispiel Card.java:
Gut:

```
public class Card {
  ...
}
```

Schlecht:

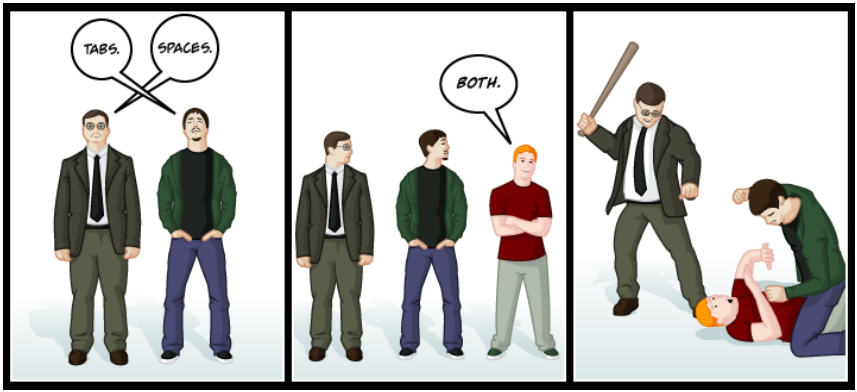
```
public class Card {
  ...
}

class CardHelper {
  ...
}
```

Anordnung von Methoden & Variablen

- Typisch: Zuerst alle Variablen, dann alle Methoden
- Kann bei guten Gründen auch geändert werden
- Wichtig: Logischer Aufbau
  - Neue Variablen sollten bspw. nicht einfach am Ende angehängt werden, so dass eine chronologische Entstehung sichtbar ist...
  - Zu viele Methoden für eine Ordnung $\rightarrow$ Die Klasse macht wahrscheinlich zu viel!

Tabs vs Spaces



Choose wisely.

Wo Klammern?

Google Style Guide / Allgemeine Empfehlung:

```
public void method() {
  if (condition()) {
    something();
  }
}
```

Importierbar via IntelliJ!

Namen

- Konsistenz (bspw. camelCase für Variablen)
- Sinnvolle, aussagekräftige Namen
- Guter Code dokumentiert sich (fast) selbst
- Schlecht: if(state == 5) ...
- Gut: if(player.ready()) ...

Exceptions nie ignorieren!

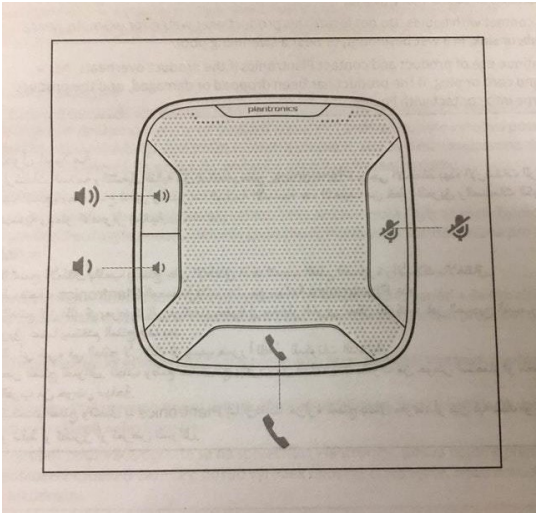
- **Immer** etwas in den catch-Block schreiben!
  - Fehlerbehebung
  - Fehlereskalation (neue Exception werfen)
  - Debug Output ausgeben
  - Kommentar, weshalb das Problem nicht behandelt werden muss
- Im Minimum:

```
// TODO(Florian): Figure out how to handle error.
```

Javadoc I

- So früh wie möglich
- Kurz und aussagekräftig
- Nur wenn nötig
  Schlecht: `@param playerItem the player's item`

Javadoc II (the useless version)



Tools

- IntelliJ Code Style (bspw. Google Java Style Guide[2][3])
- EditorConfig[4]
- Und viele mehr...

[2] https://github.com/google/styleguide
[3] https://google.github.io/styleguide/javaguide.html
[4] https://editorconfig.org/

Housekeeping

- Die Wohnung wird wöchentlich geputzt!
- *Reformat Code*
- Dokumentation verbessern oder löschen
- Variablennamen anpassen
- Tests schreiben
- Den Code schöner zurücklassen als man ihn aufgefunden hat!

Weiteres Lesematerial

- Clean Code von Robert Martin
- Design Patterns: Elements of Reusable Object-Oriented Software
- (`https://github.com/kelseyhightower/nocode`)

Fragen?