

# Softwareentwicklung I (IB)

## — Einführung —

Prof. Dr. Oliver Braun

Letzte Änderung: 13.07.2017 19:55

- ▶ **Programm** = Anweisungen für einen Computer
- ▶ **Software** = alle Arten von Programmen und Programmsammlungen
- ▶ In der Regel fertig geliefert, heruntergeladen, bereit zur Ausführung
- ▶ Programme haben **Namen** („Programmnamen“)
- ▶ **Programmstart** („Aufruf“) durch Eingabe des Namens, Klick auf ein Icon, automatisch ohne Zutun ...

- ▶ **Softwareentwicklung** = Erstellen neuer Programme
- ▶ Einmal erstellen, oft verwenden  $\Rightarrow$  Aufwand lohnt sich
- ▶ Reines „Schreiben“ des Programmcodes nur Teil der tatsächlichen Arbeit
- ▶ Softwareentwicklung umfasst ...
  - ▶ Klären, welches Problem das neue Programm überhaupt lösen soll („Anforderungsdefinition“)
  - ▶ Aufbau des neuen Programms überlegen („Entwurf“)
  - ▶ Programm schreiben („Programmierung“)
  - ▶ Sicherstellen, dass das Programm zuverlässig funktioniert („Test“)

# Programmiersprachen

- ▶ Computer verstehen keine natürlichen Sprachen, wie Deutsch oder Englisch
- ▶ **Programmiersprache** = Sprache zur Kommunikation mit dem Computer
- ▶ Von Menschen geschrieben, von Computern gelesen
- ▶ **Formale Sprachen** = Bedeutung jedes Satzes eindeutig festgelegt, kein Interpretationsspielraum
- ▶ Im Vergleich zu natürlichen Sprachen kleines Vokabular, aber komplexe Grammatik

# Generationen von Programmiersprachen

- ▶ Erste Programmiersprachen etwa 1950, seither verschiedene „Generationen“
  - ▶ Maschinensprachen, Assembler
  - ▶ Prozedurale Sprachen (Algol, Fortran, Pascal, C)
  - ▶ Objektorientierte Sprachen (C++, Java, C#)
  - ▶ Funktionale Sprachen (Haskell, Lisp/Scheme, Clojure) <sup>1</sup>
  - ▶ viele Sprachen sind sog. *Hybrid*-Sprachen

---

<sup>1</sup>Funktionale Sprachen passen eigentlich nicht in diese Generationenfolge. Sie sind einerseits älter als viele prozeduralen Sprachen, werden andererseits aber auch als potentielle Nachfolger objektorientierter Sprachen behandelt.

- ▶ **Java** seit Jahren populär
  - ▶ vergleichsweise einfach
  - ▶ ausdrucksstark, geeignet zur Konstruktion großer Programme
  - ▶ viele frei verfügbare Hilfsmittel
  - ▶ läuft auf nahezu allen Computern

## Erstes Beispiel

- ▶ Programm Hello gibt den Text Hello, World! auf dem Bildschirm aus:

```
class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

- ▶ **Quelltext** (source code) in einer Textdatei namens Hello.java aufschreiben und abspeichern <sup>2</sup>

---

<sup>2</sup>Ein Textverarbeitungsprogramm, wie Microsoft Word oder OpenOffice, eignet sich nicht.



- ▶ Computer könnte Java-Quelltext nur langsam direkt ausführen
- ▶ Quelltext wird zuerst in **Bytecode** transformiert („übersetzt“)
- ▶ Bytecode kann schnell ausgeführt werden („effizient“), ist aber für Menschen kaum lesbar
- ▶ Transformation Quelltext → Bytecode automatisch mit Hilfe eines **Compilers** (Übersetzers)

```
$ javac Hello.java
```

- ▶ Bytecode abgespeichert in einer neuen Datei `Hello.class`

- ▶ Programmstart führt Bytecode aus (.class wird automatisch angefügt):

```
$ java Hello
```

- ▶ Der Bytecode wird ausgeführt, auf dem Bildschirm erscheint die Ausgabe Hello, World!
- ▶ Bytecode kann beliebig oft ausgeführt werden
- ▶ Quelltext ist zur Ausführung nicht mehr nötig

- ▶ Schritte zum Entwickeln eines neuen Programms:
  1. Quelltext **erstellen** oder früher erstellen Quelltext **ändern**
  2. Quelltext mit Compiler in Bytecode **übersetzen**
  3. Bytecode **ausführen**
- ▶ Am Beispiel
  1. Beliebigen Text-Editor starten, Quelltext eintippen und abspeichern
  2. Compiler auf der Kommandozeile starten (liest `Hello.java`, erzeugt daraus `Hello.class`): `javac Hello.java`
  3. Programm starten (liest `Hello.class`, führt es aus):  
`java Hello`

# Java-Software

# Versionen

- ▶ Java veröffentlicht 1996
- ▶ Ursprünglich entwickelt von Sun Microsystems
- ▶ Verschiedene Anbieter, beispielsweise Oracle, IBM
- ▶ Mehrere Versionen, immer aufwärtskompatibel<sup>3</sup>
- ▶ Für diese Vorlesung nötig: **Java 8**, ältere Versionen reichen nicht aus
- ▶ Konkrete Installation überprüfen mit:

```
$ java -version
```

```
java version "1.8.0_131"
```

```
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

Entscheidend ist 1.8

<sup>3</sup>„Aufwärtskompatibel“ bedeutet, dass Programme älterer Version ohne Änderung mit Programmen neuerer Versionen zusammenpassen.

- ▶ Java läuft auf ganz unterschiedlichen Systemen
- ▶ Wird in verschiedenen „Packungsgrößen“ angeboten
  - ▶ **EE** (enterprise edition): große Unternehmensserver
  - ▶ **SE** (standard edition): Desktop-Systeme
  - ▶ **ME** (micro edition): Handys, PDAs, Embedded Systems
- ▶ Editionen unterscheiden sich in den mitgelieferten Zugaben, nicht in der Programmiersprache
- ▶ Für diese Vorlesung: Java **SE 8**

- ▶ Zum Schreiben neuer Programm ist der Compiler javac nötig, zum Ausführen fertiger Programm nicht
- ▶ Java für verschiedene Einsatzzwecke:
  - ▶ **JRE** (java runtime environment) zum Ausführen fertiger Programme
  - ▶ **JDK** (java development kit) = JRE + Compiler und weitere Werkzeuge zum Entwickeln neuer Programme
- ▶ Für diese Vorlesung: **JDK SE 8**
- ▶ überprüfen mit

```
$ javac -version
javac 1.8.0_131
```

# Java-Quelltext



```
1 class Hello {  
2     public static void main(String[] args) {  
3         System.out.println("Hello, World!");  
4     }  
5 }
```

- ▶ class kündigt ein neues Programm an, dessen Name, Hello, direkt danach folgt.
- ▶ Geschweifte Klammern grenzen zusammen gehörende Quelltext-Abschnitte ab.
  - ▶ Das Klammernpaar in Zeile 1 und 5 umschließt das gesamte Programm
  - ▶ Das Klammernpaar in Zeile 2 und 4 die Hauptfunktion

```
1 class Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

- ▶ Zeile 2 mit dem Wort `main` markiert den Start der eigentlichen Anweisungen. Sie wird auch als „Einsprungpunkt“ bezeichnet, weil hier die Ausführung des Programms beginnt
- ▶ Zeile 3 nennt die einzige Anweisung, die dieses Programm ausführt. `println` fordert zur Ausgabe auf dem Bildschirm auf. Der Text, der auf dem Bildschirm erscheinen soll, steht zwischen den Gänsefüßchen.

# Layout

- ▶ Layout = optisches Arrangement des Quelltextes
- ▶ Umfasst Leerzeilen, Zeilenumbruch, Einrückung, Zwischenraum, Kommentare
- ▶ Compiler ignoriert Layout weitgehend
- ▶ Aber: Zwischen aufeinanderfolgenden Wörtern muss Zwischenraum stehen. Falsch wäre zum Beispiel:

```
classHello {
```

# Schreiben Sie lesbaren Code

- ▶ Sinnvoll mindestens:
  - ▶ Höchstens eine Anweisung pro Zeile
  - ▶ Text zwischen geschweiften Klammern einrücken
- ▶ Empfehlenswert: Java Code Conventions einhalten
- ▶ Der Compiler akzeptiert zwar auch:

```
class
Hello{public static void
main(String[
]args){System
.out.
println("Hello, World!"
);}}
```

Diese Fassung ist aber schwer lesbar und schwer zu verstehen.

# Kommentare

- ▶ Erläuterung zum Programm als Freitext
- ▶ Compiler behandelt Kommentare als Zwischenraum und ignoriert den Inhalt ansonsten
- ▶ Kommentare dienen einem menschlichen Leser zur Orientierung und zum Verständnis
- ▶ Zwei Formen:

- ▶ **Zeilenkommentar**

```
// Text bis zum Ende dieser Zeile
```

- ▶ **Blockkommentar**

```
/* beliebig viele Textzeilen,  
 * alles Kommentar....  
 * blah, fasel, sülz ...  
 */
```

# Was kommentieren?

- ▶ Kommentare sollen Sinn, Zweck, Wirkung von Programmfragmenten erklären, aber **nicht** ...
  - ▶ Offensichtliches wiederholen

```
// hier wird "blah" ausgegeben  
System.out.println("blah");
```

- ▶ Schlechten Code rechtfertigen

```
/*  
 * Das hab ich zwar gestern geschrieben,  
 * aber heute verstehe ich nicht mehr,  
 * was die folgende Anweisung soll.  
 * Wenn man sie löscht, funktioniert nichts mehr.  
 * Also einfach stehen lassen,  
 * wird schon irgendwie gut gehen.  
 */  
for(int i=j++;j<=i+i--;--j,i-=i++);
```

# Was denn dann kommentieren?

- ▶ Beispiel für Kommentare:

```
/*
 * Vorlesung Softwareentwicklung I (IB) WS2017/18
 * Autor: Oliver Braun
 * Entwickelt mit: Oracle JDK SE 8, Darwin, Atom
 * Zweck: Erstes Beispielprogramm
 */
class Hello {
    /*
     * Das Programm gibt immer den selben Text aus,
     * der nicht beeinflusst werden kann.
     */
    public static void main(String[] args) {
        // Die einzige Anweisung im Programm
        System.out.println("Hello, World!");
    }
}
```

- ▶ Generell großzügig verwenden:
  - ▶ zu viel Kommentar schadet kaum,
  - ▶ zu wenig Kommentar kann funktionierenden Code wertlos machen



# Namen

- ▶ An vielen Stellen: frei wählbare **Namen** = „Bezeichner“, „Identifizier“
- ▶ Im ersten Beispielprogramm

```
class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

können die folgenden Namen geändert werden:

- ▶ `Hello`: Name des ganzen Programms, sollte mit dem Dateinamen übereinstimmen.
- ▶ `main`: Name, der von `java` gesucht wird, um das Programm zu starten.
- ▶ `args`: Name der Kommandozeilenargumente, wird hier nicht weiter verwendet.

# Syntax und Konventionen

- ▶ Namen müssen der **Syntax** genügen:
  - ▶ Nur große und kleine Buchstaben, Ziffern, Underscore (`_`)<sup>4</sup>
  - ▶ Als erstes Zeichen keine Ziffer
  - ▶ Keines der etwa fünfzig reservierten Wörter, wie zum Beispiel `class`, `int`, `public`
- ▶ Namen sollten außerdem **Konventionen** (= allgemeine Übereinkunft) einhalten, nämlich
  - ▶ allgemein gültige und
  - ▶ für Namen in bestimmten Rollen.
- ▶ **Konvention** = allgemeine Übereinkunft, Einhaltung freiwillig
- ▶ Compiler ignoriert Konventionen
- ▶ Siehe wieder Java Code Conventions

---

<sup>4</sup>Formal ist auch das `$`-Zeichen in Bezeichnern erlaubt. Es wird allerdings für system-generierte Namen eingesetzt und sollte deshalb nicht vom Benutzer verwendet werden.

# Beispiele

---

|                  |                                                         |
|------------------|---------------------------------------------------------|
| counter          | OK                                                      |
| colorDepth       | OK                                                      |
| iso9660          | OK                                                      |
| XMLProcessor     | OK                                                      |
| MAX_VALUE        | OK                                                      |
| 1stTry           | <b>Nicht OK</b> erster Buchstabe darf keine Ziffer sein |
| Herz Dame        | <b>Nicht OK</b> Leerzeichen im Namen nicht erlaubt      |
| const            | <b>Nicht OK</b> reserviertes Wort                       |
| muenchen-erdning | <b>Nicht OK</b> Bindestrich im Namen nicht erlaubt      |

---

# Allgemeine Benennungsregeln

|                                                             | <i>OK</i>                               | <b>Nicht OK</b>            |
|-------------------------------------------------------------|-----------------------------------------|----------------------------|
| Neue Wortteile mit großen Buchstaben (CamelCode) beginnen   | sortByType                              | sortbytype<br>sort_by_type |
| Ganze Wörter statt Abkürzungen                              | counter                                 | c cntr                     |
| Aussagekräftige Namen statt nichtssagender Kürzel           | counter                                 | 0o00o0 n<br>x              |
| Mehrdeutige Abkürzungen ausschreiben                        | binaryUpload<br>bottomUp<br>bulletproof | dup                        |
| Gebräuchliche Akronyme bis 3 Buchstaben in Großbuchstaben   | XML                                     | xml                        |
| Gebräuchliche Akronyme ab 4 Buchstaben in CamelCode         | Html                                    | HTML                       |
| Englische Begriffe statt Landessprache (auch nicht Deutsch) | counter<br>quotes                       | dirisha<br>Gänsefüßchen    |

# Konventionen für Java-Identifizier

- ▶ **Variablen, Methoden, primitive Typen:** Camelcode, erster Buchstabe klein:  
counter  
find1stToken  
bottomUp
- ▶ **Referenztypen:** Camelcode, erster Buchstabe groß:  
Hello  
String  
ServerSocket
- ▶ **Typvariablen (Generics):** einzelne große Buchstaben:  
T  
U
- ▶ **statische, öffentliche Konstanten:** alle Buchstaben groß, Wortteile getrennt mit Underscore:  
MAX\_VALUE  
PI  
RGB24

# Regel-Ebenen

- ▶ Alle Sprachen, einschließlich Java, unterliegen Regeln auf verschiedenen Ebenen:
  - ▶ **Syntax** (Rechtschreibung): Verteilung von Semicolons, Klammern, Schreibweise von Namen
  - ▶ **Semantik** (Bedeutung): Zulässige Kombination von Sprachelementen
  - ▶ **Pragmatik** (Gebrauch): Bewährte und sinnvolle Konstruktionen
- ▶ Verstöße gegen die Regeln äußern sich auf unterschiedliche Art:
  - ▶ Syntaxfehler: Compiler meldet den Fehler
  - ▶ Semantische Fehler: Compiler meldet den Fehler oder Programm startet, stürzt aber irgendwann ab
  - ▶ Fehler der Pragmatik: Programm ist unleserlich, umständlich, unverständlich

# Beispiele in natürlicher Sprache

- ▶ Natürliche Sprachen haben auch Syntax, Semantik, Pragmatik
- ▶ Beispiele für Verstöße:
  - ▶ Singen tröt Öltanker die blau.
    - ▶ Syntax falsch
  - ▶ Der Öltanker singt blau.
    - ▶ Syntax ok
    - ▶ Semantik falsch
  - ▶ Das Zeitmessgerät läuft versetzt gegenüber der Standardzeit.
    - ▶ Syntax ok
    - ▶ Semantik ok
    - ▶ Pragmatik falsch
  - ▶ Die Uhr geht nach.
    - ▶ Syntax ok
    - ▶ Semantik ok
    - ▶ Pragmatik ok

# Beispiele für Verstöße in Java — Syntax

- ▶ Syntax falsch

```
class Hello {  
    public static void main(String[] args);  
    {  
        System.out.println("Hello, World!");  
    }  
}
```



# Beispiele für Verstöße in Java — Semantik

- ▶ Semantik falsch
  - ▶ Compiler entdeckt den Fehler:

```
class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello", "World!");  
    }  
}
```

- ▶ Compiler bemerkt nichts:

```
class Hello {  
    public static void main(String args) {  
        System.out.println("Hello, World!");  
    }  
}
```

## Beispiele für Verstöße in Java — Pragmatik

- ▶ Pragmatik falsch

```
class Hello {  
    public static void main(String[] args) {  
        System.out.print("H");  
        System.out.print("e");  
        System.out.print("l");  
        System.out.print("l");  
        System.out.print("o");  
        System.out.print(",");  
        System.out.print(" ");  
        System.out.print("W");  
        System.out.print("o");  
        System.out.print("r");  
        System.out.print("l");  
        System.out.print("d");  
        System.out.println("!");  
    }  
}
```

# Werkzeuge

- ▶ Arbeit auf der Kommandozeile umständlich bei größeren Programmen
- ▶ **IDEs** (integrated development environments) kapseln Editor, Compiler und Start hinter Buttons
- ▶ Frei verfügbar und abgestimmt auf Java: **Eclipse, Netbeans**
- ▶ Für Lehre und nichtkommerzielle Anwendung frei: **IntelliJ IDEA**
- ▶ Aber:
  - ▶ IDEs sind selbst komplexe Programme
  - ▶ Umgang muss erlernt werden
  - ▶ IDEs kapseln viele Routineschritte, die Sie hier aber lernen müssen
- ▶ Nicht Gegenstand der Vorlesung
- ▶ Empfehlung für das Praktikum
  - ▶ Atom (Windows, macOS, Linux)