Kapitel 1

Einführung und Grundbegriffe

Ziele

- Begriffsbildungen: Informatik, Algorithmus, Programm, Compiler, …
- Einordnung von Java
- Ein einfaches Java-Programm erstellen, übersetzen und ausführen
- Java-Programme dokumentieren

Informatik

Informatik
ist ein Kunstwort,
das in den 60ziger Jahren
in Frankreich kreiert wurde,

entstanden aus
Information + Mathematik

englisch: Computer Science
neuerdings auch: Informatics

bedeutet
Wissenschaft der
maschinengestützten
Informationsverarbeitung

Teilgebiete der Informatik

Praktische Informatik

- Programmierung und Software-Entwicklung
- Datenbanksysteme
- Betriebssysteme, Middleware für verteilte Systeme

Theoretische Informatik

- Formale Sprachen
- Syntax und Semantik von Programmiersprachen
- Algorithmen und Komplexität

Technische Informatik

- Rechenanlagen und Rechnernetze

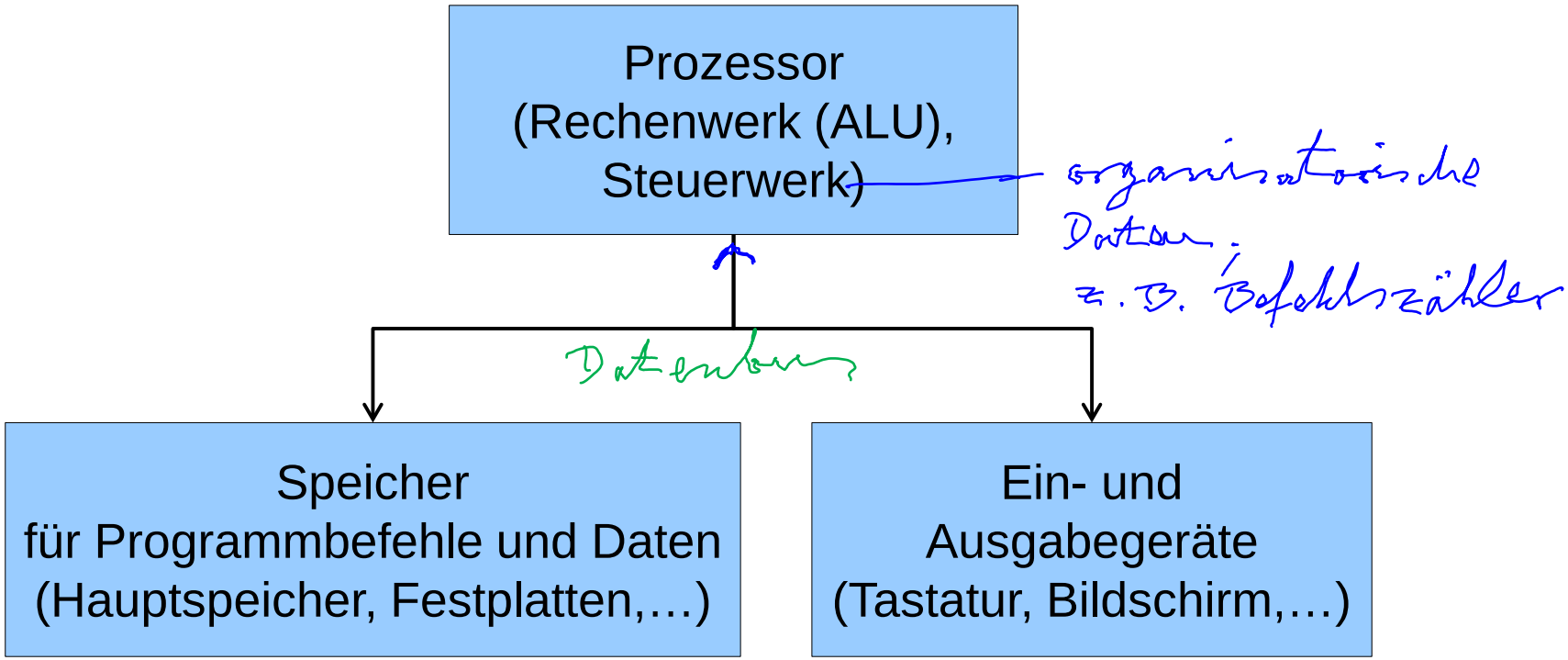
Informatik und Gesellschaft

- Auswirkungen der Informatik auf die Gesellschaft

  (Datensicherheit, Allgegenwärtigkeit von Rechnern, soziale Netzwerke, …)

Aufbau eines Computers (von Neumann Modell)

Die meisten heute benutzten Computer entsprechen der Von-Neumann-Architektur. 1945

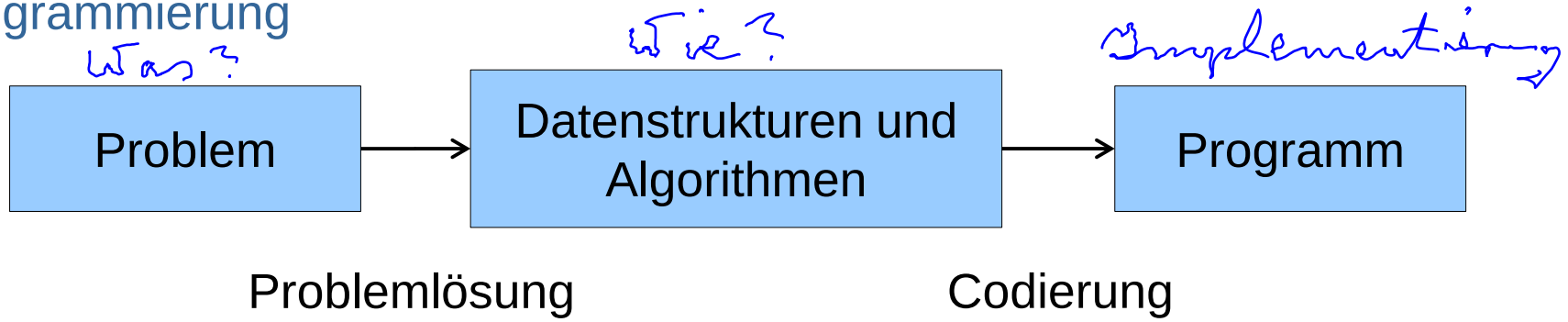


Programmierung und Software-Entwicklung

Programm

Beschreibung von Datenstrukturen und Algorithmen in einer „dem Computer verständlichen“ Sprache (Programmiersprache)

Programmierung



Software-Entwicklung

Systematische Konstruktion von Programmen und komplexen Softwaresystemen (→ Systemarchitektur)

großes Sw-System ≥ 50.000 loc (lines of code)
sehr " " " ≥ 1 Mio "
z.B. Windows 7: ≈ 40 Mio loc

Zentraler Begriff: Algorithmus

Algorithmus (nach Al-Khwarizmi, um 800)

- Allgemeines Verfahren zur Lösung eines Problems, das durch eine eindeutige Vorschrift so genau festgelegt ist, dass man es anwenden kann, ohne es verstanden zu haben.
- Eigenschaften:
  - Jeder Schritt ist eindeutig festgelegt und **berechenbar**.
  - Das Verfahren liefert nach endlich vielen Schritten eine Lösung (es terminiert).
- Beispiele:
  - Modellbau: Montageanleitung
  - Küche: Kochrezept
  - Informatik: Such- und Sortieralgorithmen

Al'Khwarizmi
790-840
Author von
Hisab **al-jabr**
w'al-muqabala

Berechenbarkeit

- Es gibt verschiedene äquivalente **Definitionen der Berechenbarkeit**, z.B.:
  - Ein Problem ist berechenbar, wenn es durch eine Turing-Maschine berechnet werden kann (Alan Turing: 1912 – 1954).
  - Ein Problem ist berechenbar, wenn es durch eine partiell-rekursive Funktion berechnet werden kann.
  - Ein Problem ist berechenbar, wenn es durch ein While-Programm berechnet werden kann.

- **Beachte:** Nicht alle Probleme sind berechenbar!
  Beispiel „Halteproblem“:
  Bestimme ob ein x-beliebiges Programm für eine x-beliebige Eingabe terminiert.

- **Gödelscher Unvollständigkeitssatz** (1931):
  In der Arithmetik gibt es Aussagen, die weder formal bewiesen noch widerlegt werden können.

Beispiel: Sortieren einer Liste

Gegeben: Eine Liste von Elementen, die geordnet werden können.

Gesucht:
Liste mit denselben Elementen in aufsteigender Reihenfolge angeordnet

Algorithmus: z.B. **Sortieren durch Vertauschen („Bubble Sort“)**:

Falls die Liste leer ist: fertig.

Ansonsten:

Sei „outer“ ein Zeiger auf das letzte Element der Liste.

Solange „outer“ nicht auf das erste Element zeigt:

1. Sei „inner“ ein Zeiger auf das erste Element der Liste.
   Solange „inner < outer“:
   1.1 Vertausche Elemente an den Positionen „inner“ und „inner+1“,
   wenn sie in falscher Reihenfolge stehen.
   1.2 Rücke mit „inner“ eine Position vorwärts.
2. Rücke mit „outer“ eine Position rückwärts.

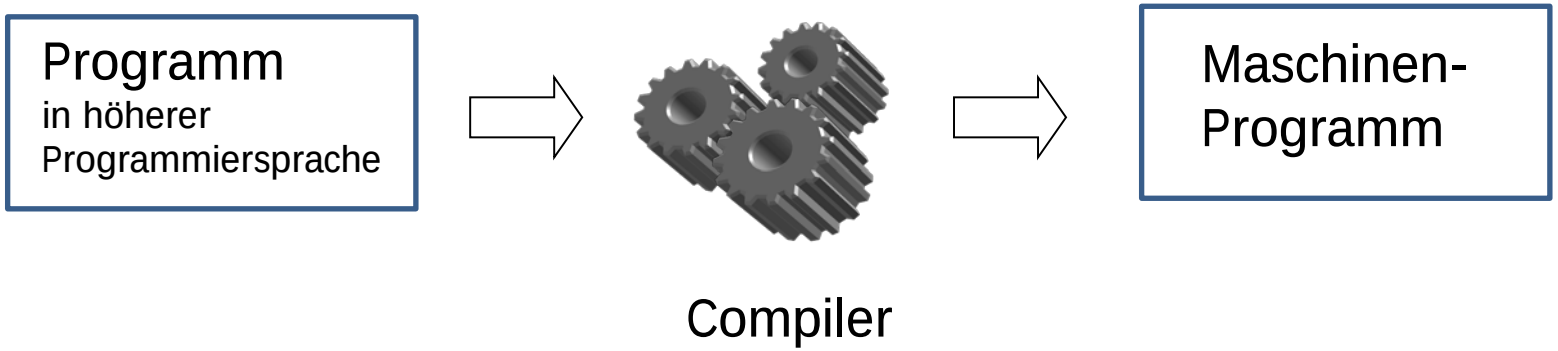
Beispiel: Sortiere die Liste 5, 33, 12, 13, 8, 1

Bubble Sort in Java

```
static void bubbleSort(double[] a){
   for (int outer = a.length - 1; outer > 0; outer--){
      for (int inner = 0; inner < outer; inner++){
         if (a[inner] > a[inner + 1]){
            // tausche a[inner] und a[inner + 1]
            int temp = a[inner];
            a[inner] = a[inner + 1];
            a[inner + 1] = temp;
         }
      }
   }
}
```

Höhere Programmiersprachen

- Formale Sprachen, in denen Algorithmen und Datenstrukturen möglichst verständlich beschrieben werden können.
- Beispiel: Bubble Sort in Java
- Programme in höheren Sprachen werden automatisch, durch *Compiler (Übersetzer)* genannte Programme, in Maschinencode übersetzt.



Arten höherer Progammiersprachen

Imperative Sprachen

typisch: Variable, Anweisungen (Befehle), Schleifen (Iteration)

Beispiele: Fortran, Algol, Pascal, Modula, C, …

Funktionale Sprachen

typisch: Ausdrücke, Funktionsauswertung, rekursive Funktionen

Beispiele: Lisp, SML, Haskell, …

Objektorientierte Sprachen

typisch: wie imperative Sprachen + Klassen, Objekte, Vererbung

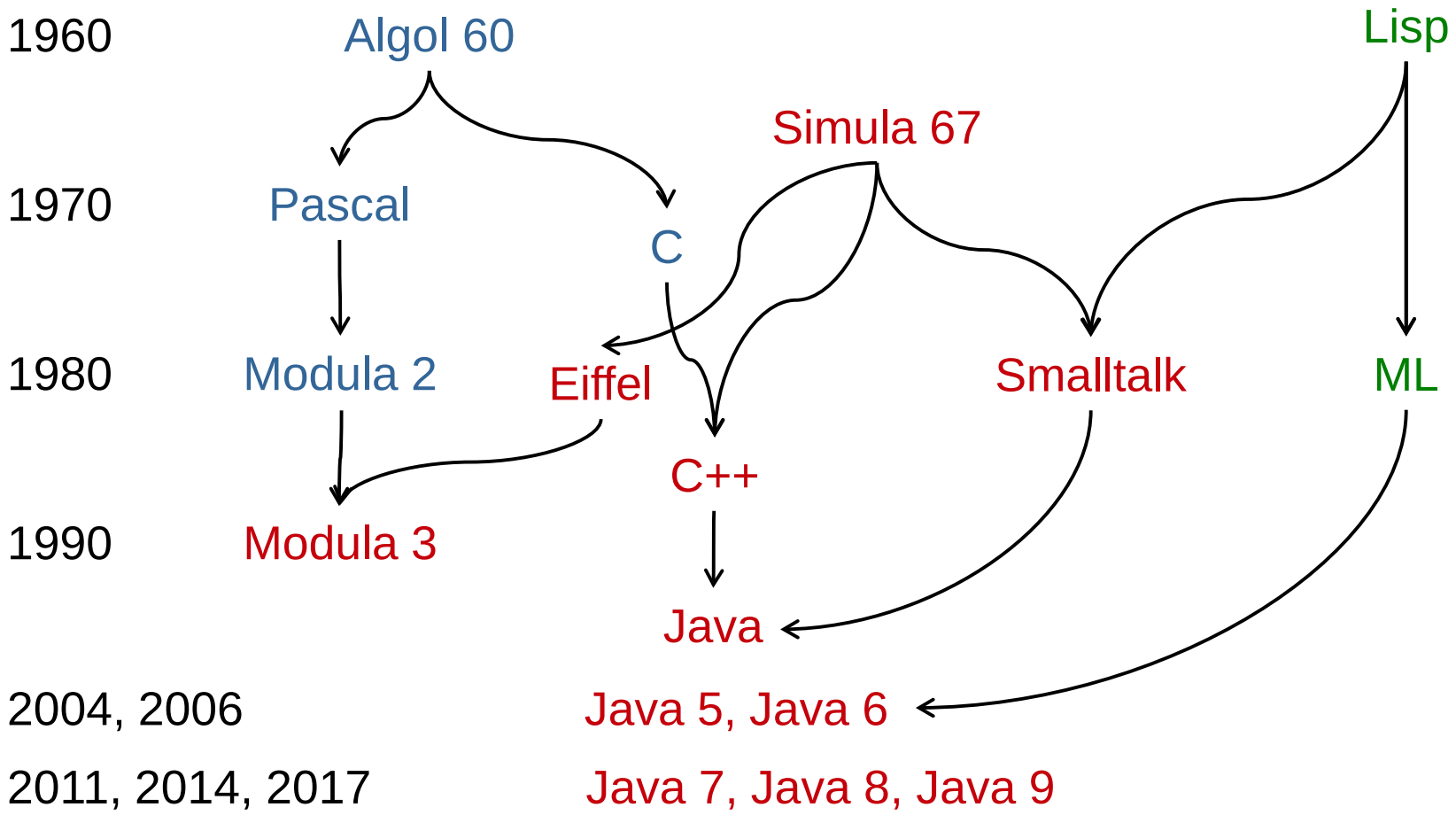
Beispiele: Simula, Smalltalk, C++, C#, Java, …

Logische Sprachen

typisch: Fakten, Regeln, logische Deduktion von neuen Fakten

Beispiele: Prolog, Mercury, …

Entwicklungsgeschichte von Java



Charakteristika von Java

- Imperativ
- Objektorientiert: Klassenkonzept, strenge Typisierung
- Unabhängig von Plattform:
  Durch Übersetzung in Bytecode,
  der von einer Virtuellen Maschine (VM) interpretiert wird.
- Unterstützt parallele Ausführungen (Nebenläufigkeit)
- Besitzt eine reichhaltige Klassenbibliothek (API, "Application Programming Interface") zur Wiederverwendung von Programmen

Grober Aufbau eines Java Programms

- Java Programme bestehen aus einer Menge von **Klassen**.
- Eine **Klasse** besteht aus
  - **Attributen:**

    Beschreiben charakteristische Merkmale der Objekte dieser Klasse
  - **Methoden:**

    Beschreiben Operationen in Form von Algorithmen

Syntaktische Struktur eines (sehr) einfachen Java-Programms

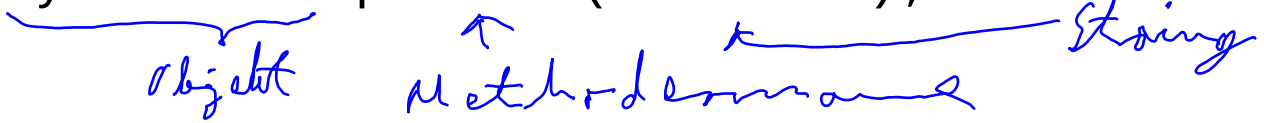
```
public class <Klassenname> {

    public static void main(String[] args) {
        <Anweisungen>
    }
}
```

Folgendes Beispiel-Programm gibt den Text „Hallo!“ aus:

```
public class Hallo {

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



Methodenaufruf

- **Methodenaufruf allgemein:**

```
object.methodName(parameters);
//bei statischen Methoden:
class.methodName(parameters);
```

- **Beispiel:**

```
System.out.println("Hallo!");
```

Leerzeichen und Formatierung

Anstelle von

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

hätten wir auch schreiben können

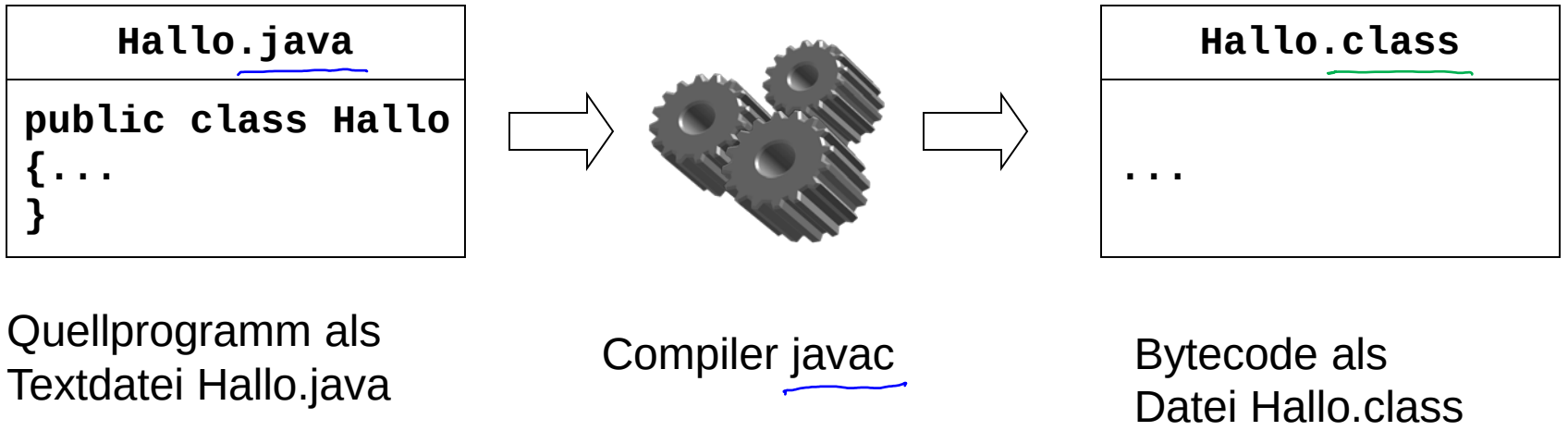
```
public class
    Hallo { public static void
        main(String[] args
  ) {System.out.println("Hallo!"); }         }
```

Meist folgt man bestimmten Konventionen für die Formatierung.

Übersetzung von Java-Programmen

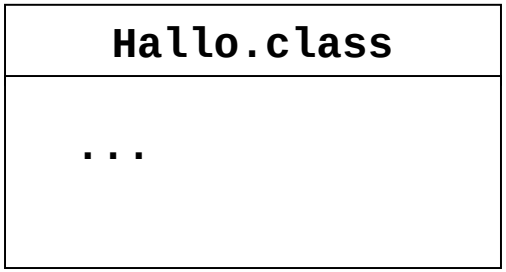
Übersetzung in Bytecode (für die Java Virtual Machine (JVM))

- Aus einer Textdatei mit Endung „.java“ erzeugt der Compiler **javac** eine Datei mit gleichem Namen, aber Endung „.class“
- Diese enthält den Bytecode für die JVM.

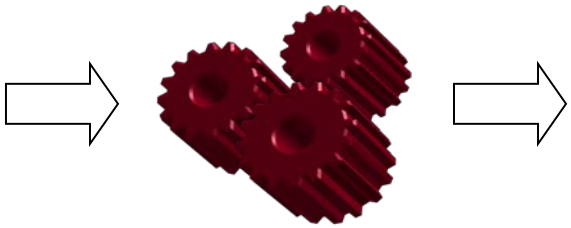


Ausführung von Java-Programmen

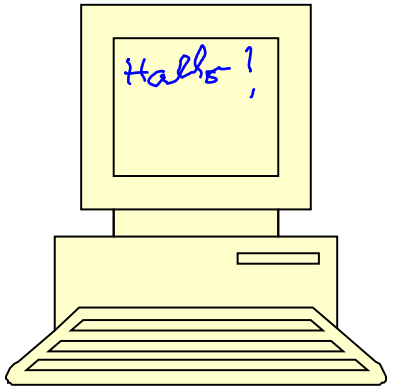
Die Datei mit dem Bytecode wird der JVM übergeben und von der JVM ausgeführt.



Bytecode als
Datei Hallo.class



Java Virtual Machine
java



Ergebnisausgabe (von
Hallo auf Bildschirm)

Übersetzung und Ausführung von Hallo.java

Übersetzung von Hallo.java:

```
C:> javac Hallo.java
```

Unter Windows (falls Systemvariable zum Auffinden von javac nicht gesetzt sind):

```
C:> "C:\Program Files (x86)\Java\jdk1.7.0\bin\javac" Hallo.java
```

Ausführung von Hallo.class:

```
C:> java Hallo
```

Gibt auf Bildschirm aus:

```
Hallo!
```

Korrektheit von Programmen

Syntaktische Korrektheit:
Ein Programm ist syntaktisch korrekt, wenn es gemäß den (syntaktischen) Regeln der Programmiersprache geschrieben ist.

Semantische Korrektheit:
Ein Programm ist semantisch korrekt, wenn bei der Ausführung des Programms die gewünschte Wirkung erzielt wird.

Beachte:
Syntaktisch korrekte Programme sind häufig semantisch nicht korrekt.

Bemerkung: Man spricht von

- normaler Software bei bis zu 25 Fehlern pro 1000 LOC (lines of code), also 2,5% Defektniveau,
- guter Software bei bis zu 2 Fehlern pro 1000 LOC (lines of code), also 0,2% Defektniveau.

Kommentare in Programmen

„The view that documentation is something that is added to a program after it has been commissioned seems to be wrong in principle, and counterproductive in practice. Instead, documentation must be regarded as an integral part of the process of design and coding.“

C. A. R. Hoare: Hints on Programming Language Design (1973)

Kommentare in Java-Programmen

- Durch

```
// bla bla
```

  wird eine Zeile oder ein Rest einer Zeile zum Kommentar.

- Mehrere Zeilen können folgendermaßen auskommentiert werden:

```
/* bla
bla
bla */
```

Die Klasse Hallo dokumentiert

```
/*
Diese Klasse dient zum Anzeigen des
Strings "Hallo!" auf dem Bildschirm
*/
public class Hallo {
    /*
    * Die Methode main gibt aus "Hallo!"
    */
      public static void main(String[] args) {
         System.out.println("Hallo!");
      }
}
```