1. Grundbegriffe und Beispiele

Literatur: Reese Kapitel 1, 2, 3, 4.

Ein einfaches Programm in Java:

```
1. public class Berechnung
2. {
3.   public static void main(String[] args)
4.   {
5.     int i;
6.     i = 3 + 4;
7.     System.out.println(i);
8.   }
9. }
```

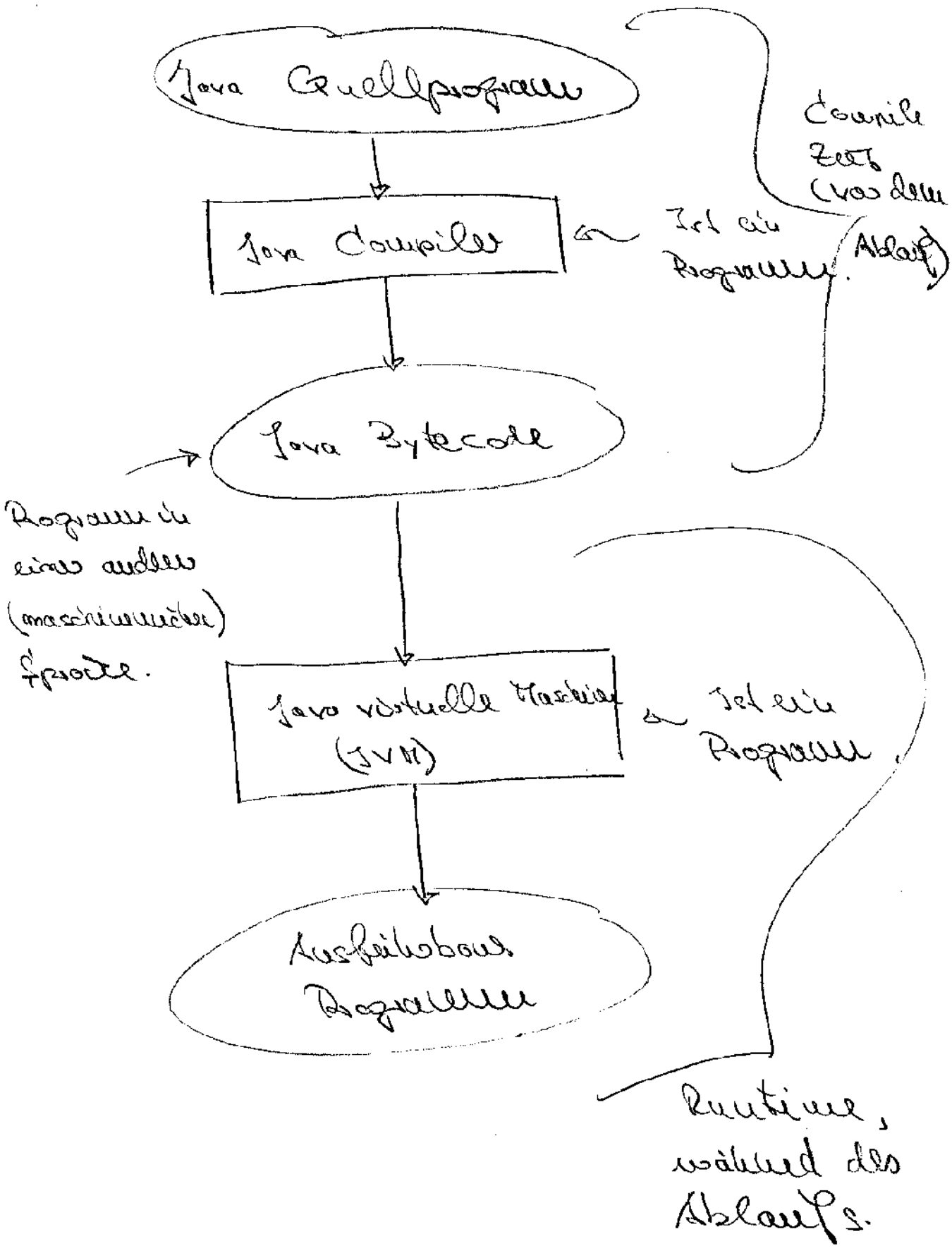
Berechnung.java

auf Programme/Woche1

Quelltext, Quellprogramm

= das ganze Programm.

Was geschieht dann?



Java Compiler = Java Übersetzer

Java virtuelle Maschine? Muß sie übersetzen? In gewisser Weise ja, arbeitet aber direkt ab. Ein

Interpreter.

Compiler übersetzt, Interpreter übersetzt und führt aus.

Was machen wir jetzt damit? Wie in der Schule? Dazu einmal das folgende Bild:

Diese Seite mußte aus rechtlichen Gründen entfernt werden!

Programm kommt auf die Festplatte.
Wie ist die Festplatte organisiert?
Sie enthält Dateien, die in
Verzeichnissen (directories) strukturiert
sind.

Die Dateien auf der Festplatte sind damit strukturiert auch vorzustellen:

Wo sind die .Dateien .login, .cshrc ?

Was ist das drinnen.

Wie jetzt das Programm

Eingeben,
Übersetzen,
Aufrufen.

Wir haben das Java Development Kit (JDK) oder Java Software Development Kit (JDK) installiert.

Programm in Datei

Berechnung.java

dann das Kommando

javac Berechnung.java

Das ist der Dateiname.

javac = Kommando für den Compiler,

also aufrufen des Namen eines Programms.

10 wichtig:

public class Berechnung

$\Leftrightarrow$

und Dateiname Berechnung.java.

Nach fehlerfreiem Lauf der Datei

Berechnung.class

Zusatz an Dateiname.

Diese enthält den sogenannten Bytecode.

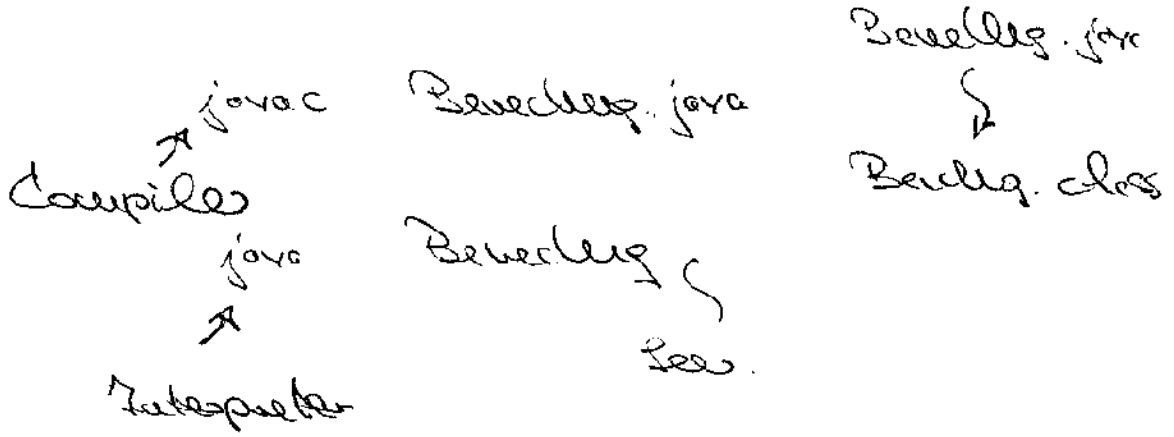
Aufruf des Interpreters mit Bytecode als Eingabe

java Berechnung

java = Name eines Programms, des Interpreters.

Die 7 sollte weiter.

Noch einmal:



Wo geschieht prinzipiell bei der Ausführung eines Programms?

Diese ist in der Zentraleinheit.

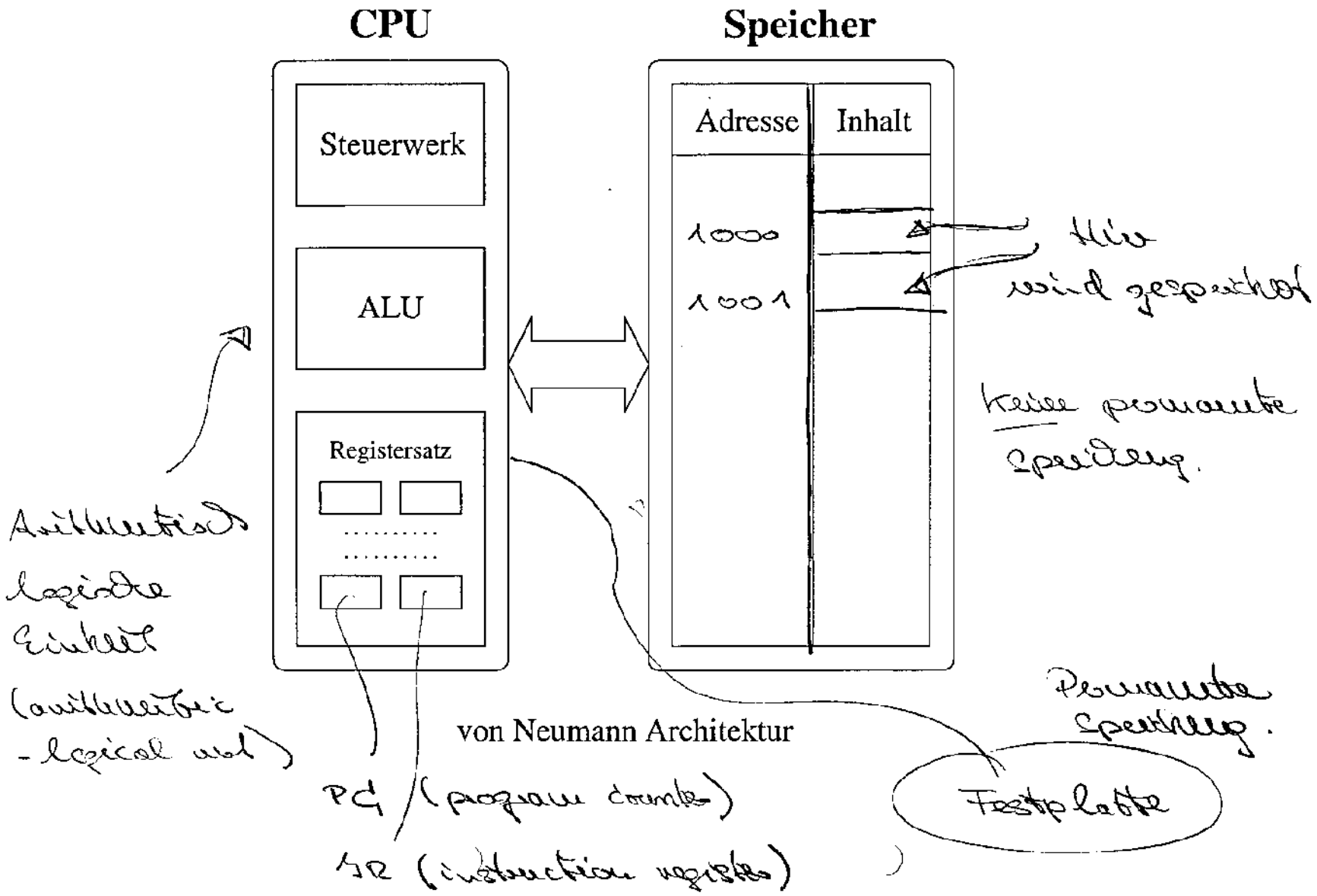
Zentraleinheit = Prozessor + Arbeitsspeicher.

CPU (central processing unit) und Hauptspeicher

Arbeitsspeicher temporär im Unterschied zu Festplatte.

Rückgesamt. Entwicklerheft.

Prozessor



Typische Befehle, die hier ausführbar sind:

Load i // Inhalt von Speicherplatz i in Register 0.

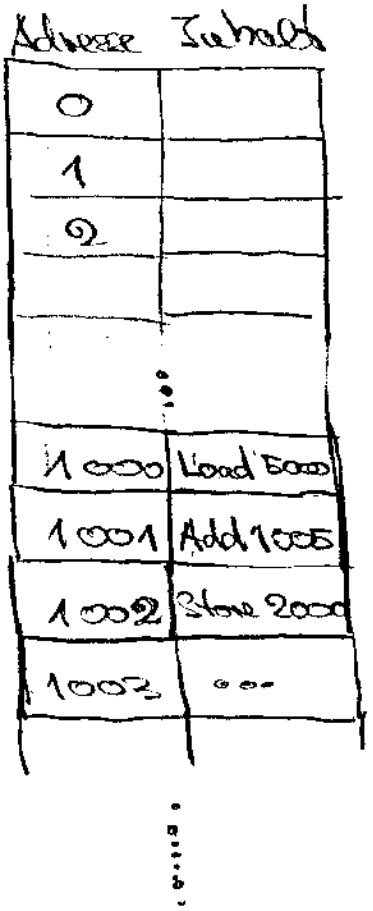
Store i // Reg 0 in Speicherplatz i

Add i // Addiert Reg 0 zu Speicherplatz i, Ergebnis in Register 0.

Programm dazu

```
Load   5000    // Addiert Speicherplatz
Add    1005    // 5000 zu 1005 und
Store  2000    // speichert in 2000.
```

Programme, die ausgeführt werden stehen auch im Hauptspeicher:



In dem Prozessor ein spezielles Register, das Programmierbare PC (programm counter), das die Adresse des aktuellen Befehls enthält.

Der Prozessor führt dann sogenannte Instruktionszyklen aus:

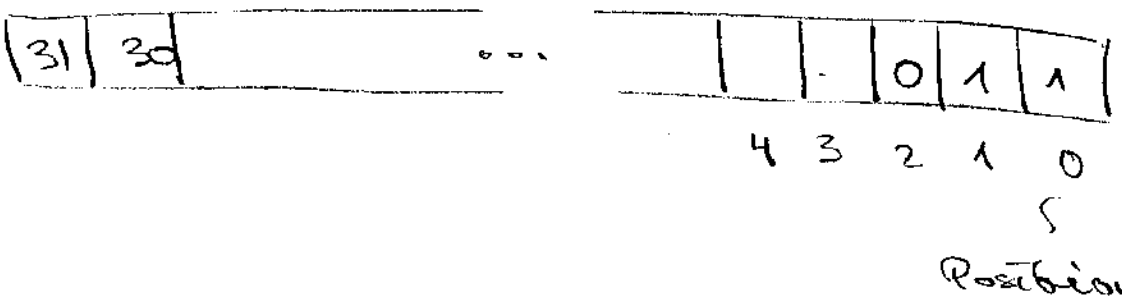
1. Fetch (Holt Befehl auf Adresse PC in IR)
2. Decode (Dekodiert die Operation)
3. Fetch Operands (Holt die Operanden aus den Speicher (= die Register))
4. Execute (Die ALU führt die Operation aus)
5. Next Instruction (PC bekommt Adresse des nächsten Befehls. In der Regel Erhöhung um 1)

Danach geht es wieder bei 1. los.

Der Arbeitsspeicher ist der RAM (random access memory) organisiert. Das heißt wahlfreier Zugriff. Das heißt: Speicherplatz zu jeder beliebigen Adresse kann gleich schnell erreicht (d.h. geladen werden).

Dagegen: Sequentieller Zugriff auf Bänder. Von der Platte werden größere Einheiten (Blocks) in den Hauptspeicher geladen. Ein Block schneller als außerhalb.

Arbeitsspeicher aus Speicherworten, die nummeriert sind. (alles eine Adresse haben). Ein Speicherwort so



Speicherwort aus 32 Bits von 0, ..., 31.

Ein Bit kann 0 oder 1 sein

(Byte = 8 Bits (hier von 0 ... 7))

Hier ist die Wortlänge 32.

Neuerdings auch Wortlänge 64.

1.16

Was passiert jetzt bei der Ausführung von unserem Programm Berechnung?

Zeile 1., ..., 5. : Vorbereitungen.

Variable i wird an einen Speicherplatz gebunden. Tabelle, die sagt i ist Speicherplatz 1004, zum Beispiel.

Variablen stehen für Speicherplätze.

Zeile 6.

Erst 3+4*7 ausrechnen, dann in Speicherplatz zu i speichern.

Platz zu i enthält dann 7 über 0,1 dargestellt.

Zeile 7.

Programmstücke zu Ausgabe.

Es werden dann eine folgende Maschinenbefehle ausgeführt:

Load x3 (Die 3 selbst (vorg. x3) in Register 0.)

Add x4 (Die 4 selbst zu 3 addieren: Ergebnis in Reg. 0.)

Store i (Reg. 0 in Speicherplatz von i. (MB evtl. modifiziert werden))

⋮

So weit zur Ausführung des Programms. Als nächstes zum Programm selbst!

Syntax = Struktur des Programms als Text.

Semantik = Das, was das Programm bewirkt.

Ein Programm besteht aus

Anweisungen.

Am Ende einer Anweisung
grundsätzlich ein ; (Semikolon).

Beispiel: int i;

Deklarationsanweisung, Variablendeklaration.

Wirkung (Semantik): Speicherplatz
wird gebunden. Inhalt eine
ganze Zahl, integer, int.

Also Zahl aus

$\mathbb{Z} = \{ \ldots -3, -2, -1, 0, 1, 2, 3, \ldots \}$.

int beschreibt den Datentyp
integer.

i ist ein benutzerdefinierter Bezeichner.

Zeile 6. i = 3+4

Eine Zuweisung. 3+4 ist ein arithmetischer Ausdruck.

Zeile 7: System.out.println(i);

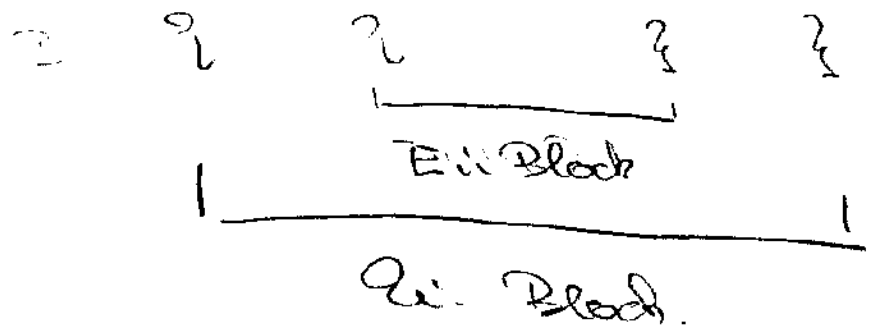
System.out.println ist eine Methode (fertiges Programmstück oder Unterprogramm) für die Ausgabe. Anweisung ist ein Methodenaufruf. i ist das Argument, der aktuelle Parameter. i meint den Inhalt des Speicherplatzes von i (als Text) meint "i" als Buchstaben.

Das noch bleibt: Der Rahmen

```
public class Bezeichnung
{
    public static void main(String[] args)
    {
    }
}
```

Klasse — Hauptmethode

Quelltext aus Blöcken, die zwischen den { } Klammern steht. Hier ein verschachtelter Block:



1. Block: Die Klasse

Java objektorientiert, arbeitet mit Klassen. Eine Klasse hat folgenden Inhalt

```
public class Berechnung
{
    Hier steht der Rest des
    Quelltexts der Klasse
    Berechnung!
}
```

Name der Klasse: Berechnung.

Die Datei mit dem Quelltext muß Berechnung.java heißen.

Von den Namen der Klasse müssen immer die Worte public und class stehen.

Klassennamen immer
mit Großbuchstaben beginnend.

2. Block: Die Hauptmethode

Klassen beinhalten Methoden.

Klassen, die selbst ablauffähig
sind, besitzen die Methode main.
Die zugehörige Programmzeile ist
dabei

public static void main(String[] args).

Diese ist ebenfalls so. String
ist mit "S" geschrieben.

Schlüsselwörter sind vordefinierte
festgelegte Wörter, die man
somit nicht verwenden sollte.

1.23.

public, class,

public, static, void

int

Benutzerdefinierte Bezeichner sind dagegen

Zeichen, main

String, args, [],

System.out.println

Bezeichner für eine Speicherplatz

Zeichen (Literalkonstanten) sind: 3, 4.

Empfehlung: 1 Anweisung pro Zeile.

Einige Modifikationen des Programms

```
7    System.out.println("Das Ergebnis ist:");
8    System.out.println(i);
```

"abc" steht für abc als Text.
System.out.println bedeutet:
Aktuellen Parameter ausgeben und
in neue Zeile gehen (line feed)
Manchmal ist auch interessant

```
7    System.out.print("Das Ergebnis ist:");
8    System.out.println(i);
```

Sollte ergeben:

Das Ergebnis ist: 7

Interessant ist auch:

7. System.out.print("Der Wert von i ist");
8. System.out.println(i);

Aus

7. System.out.print("i=");
8. System.out.print(i);

ist die Operation +
zum Aneinanderhängen von Texten
und die Bedeutung (Wirkung, Semantik)

"abc" + "chp" = "abcchp"

"abc", "chp" sind Konstanten (Literal-
konstanten) für Text (Zeichenketten)
"abc" + "chp" ist wieder Ausdruck.

Damit können wir auch schreiben

7 + System.out.println("Das Ergebnis ist:" + i)

Hier ist + die Operation, die Texte zusammenhängt, da Inhalt von i wird also als Text gesehen. Semantisch wird zunächst die Welt von

"Das Ergebnis ist:" + i ← i ist ein Ausdruck.

ausgewertet. Dieses ist der Text

"Das Ergebnis ist: 7"

Dann wird mit diesem aktuellen Parameter die Methode System.out.println ausgeführt.

Die Methoden System.out.print, System.out.println haben immer nur einen aktuellen Parameter (Argument). Dieser wird als Text verarbeitet. Also passiert bei

```
System.out.println(3+i);
```

Hier steht + für die Addition.

Noch einige Variationen des Programms

```
5    i = 100 - 10;

5    i = 10 * 10;
```

Noch ein Programm zur Übung:

Ein Programm, das einfach einen Text ausgibt

```
1  public class Uebung
1.a {                                   Uebung.java
2.    public static void main(String[], args)
2.a   {
3.      System.out.println("Algorithmen
                                  und Programmierung");
4.      System.out.println("in Chemnitz");
5     }
6   }
```

Semikolon aufpassen, ein weiteres Semikolon. Was geschieht dann?

Ausgeben können wir. Wie geben wir ein? Damit das geht, brauchen wir einen ganzen weiteren Satz von Klassen mit Methoden.

```
import Prog1Tools.IOTools;
public class EingBsp
{
  public static void main(String[] args)
  {
    int i,j;
    j = IOTools.readInteger();
    i = IOTools.readInteger();
    System.out.println("i= " + i);
    System.out.println("j= " + j)
  }
}
```

EingBsp.java

Eingabe mit „prompt":

```
System.out.print("j = ");
j = IOTools.readInteger;
```

Alternativ

```
j = IOTools.readInteger("j = ")
```

Das grundlegendste Prinzip der Programmierung ist das der Wiederholung, der wiederholten Ausführung von Anweisungen. Wiederholungen treten in verschiedener Form auf. Hier behandeln wir nur die while - Schleife.

```
1.  public class Wiederum
2.  {
3       public static void main(String[] args)
4       {
5       int i;
6       i = 10;                          <- Hier ist Stolperstelle!
7       while (i > 0)
8           {
9           i = i - 1;
10          System.out.println(i);
```

11 }
12 }
13 }

Das ist eine while-Schleife.
Diese besteht syntaktisch
aus einem Block, dem
Schleifenrumpf und dem Schleifenkopf,
der <u>while</u> $(i > 0)$. $i > 0$ ist
ein Boolscher Ausdruck, und
wahr oder falsch.

Semantik: Ausführung des
Rumpfes solange bis die
Bedingung falsch ist. Endet nicht!
Reihung: 1. Bedingung testen. 2. Rumpf
(falls Bedingung wahr) 3. testen. 4. Rumpf,
... Bis Bedingung falsch.

```
:
if (i > 0)
   {
     i: = i - 1 ;
     System. out. println(i);
   };
while (i>0)
   {
     i = i-1
     System. out. println(i)
   }
   :
```

Dasselbe mit 10-mal

```
i:=i-1;
System. out. println(i);
i:=i-1;
System. out. println(i);
   :
```

Mit Eingabe

```
import Prog1Tools.IOTools;
public class WiedomvsFini
{
   public static void main(String[] args)
   {
      int i;
      i = IOTools.readInteger("i=");
      while (i>0)
      {
         i=i-1;
         System.out.println(i);
      }
   }
}
```

Ausdel Schleifenausführung endlich, also von Eingabe abhängig!

(1.35)

Ein weiteres Beispiel: Berechnen von

$$1 + 2 + 3 + \ldots + k \,,$$

wobei k eingegeben.

```
import Prog1Tools.IOTools;

public class Summe
{
  public static void main(String[] args)
  {
    int i, j;
    j = 0;      // Initialisierung von j
    i = IOTools.readInteger("i=");
    while (i > 0)
    {
      j = i + j;
      i--;      // Für i = i - 1
      System.out.println("i= " + i + " j= " + j);
    }
  }
}
```

Berechnung des Programms bei $i=0$:

i	j
0	0

$i > 0$ ist falsch.
und Schluß.

$i = 1$

i	j
1	0

$i > 0$ wahr

$j = j + i$

1 1

$i --$

0 1

Ausgabe von j (=1)
$i > 0$ ist falsch. Schluß.

$i = 2$

⋮

Ausgabe von $j = 3$
$i > 0$ falsch. Ende.

Programm transformiert Variablenbelegungen.

Alle möglichen Belegungen; der:

Y ... Zustandsraum.

Hier: Menge der Paare (i, j) wobei

i, j ganze Zahlen. Deklaration bestimmt

den Zustandsraum.

Was berechnet das Programm?

Am Ende ist bei Eingabe von $k \geq 0$

$$j = 0 + 1 + 2 + 3 + \dots + k.$$

Wie sieht man das genauer? Dazu

sei für $l = 1, 2, 3, \dots$

i_l = Wert von i nach l-tem
Durchlauf der Schleife

j_l = Wert von j nach l-tem ...

Außerdem i_0, j_0 Werte vor dem
ersten Durchlauf.

Jetzt sei k die eingelesene Zahl. Dann ist $i_0 = 0$, $i_0 = k$.

Deshalb nach dem ersten Durchlauf

$$i_0 = k, \quad i_0 = k-1 .$$

Also auch vor dem zweiten. Dann nach dem zweiten:

$$i_2 = k+(k-1), \quad i_2 = \underbrace{k-2}_{\text{Alles } i} \quad \text{da } -1 \; -1$$

Nach dem dritten

$$i_3 = k+(k-1)+(k-2), \quad i_3 = k-3$$

$$\vdots$$

Nach dem k-ten $\quad k-k$

$$i_{k+1} = \underbrace{k+(k-1)+(k-2)+\ldots+3+2+1}_{k}, \quad i_k = 0 .$$

Wir haben insgesamt genau k Schleifen-durchläufe wegen der $i--$ und der Bedingung $i > 0$.

Für den Sonderfall $k = 0$ gilt alles ebenso. Kein Durchlauf

Betrachten wir folgende Aussage für l mit $1 \leq l \leq k$: Nach dem l-ten Durchlauf der Schleife ist

$$j = \underbrace{k + (k-1) + \dots + (k-l+1)}_{l \text{ Summanden}}, \quad i = k - l$$

j: Wert von j | i: Wert von i

Ist diese Aussage wahr und zwar für alle l?

Nach dem ersten Durchlauf, also $l=1$, ist $i \leq k$, also $j = k$ und $i = k-1$.

Nach dem zweiten auch.

(1.40)

gilt sie für $l \leq k$, dann auch für $l+1 \leq k$. Denn ist vor dem $(l+1)^{\text{ten}}$ Lauf

$$j_l = k + k-1 + \dots + k-l+1 \,, \quad i_l = k-l$$

dann, wie man sieht, danach

$$j_{l+1} = \underbrace{k + k-1 + \dots + k-l+1}_{\text{altes } j} + \underbrace{k-l}_{\text{altes } i}$$

$$i_{l+1} = k-l-1 = k-(l+1).$$

Also gilt die Aussage nach dem $l+1$-ten Durchlauf (und damit vor dem $l+2$-ten).

Zusammenfassend: Aussage gilt
nach dem ersten Durchlauf.

Haben gezeigt: Gilt sie nach
dem ℓ-ten, $1 \leq \ell < k$, dann
nach dem $(\ell+1)$-ten.

Also: Gilt nach dem ersten,
nach dem zweiten (obiges
Schluß mit $\ell=1$), nach dem
dritten ($\ell=2$), ..., nach dem
k-ten.

Haben k Durchläufe.

Also Programm ist korrekt.

Um Schleifen machen, weist man durch einige Beweise, Induktions beweise, noch eine Aufgabe noch eine Aussage über die # Durchläufe.

Für jeden l mit $1 \leq l \leq k$, ist die Aussage:

"Nach dem l-ten Durchlauf

ist

$$i = k + (k-1) + \dots + (k-l+1), \; j = k - l$$

eine Schleifeninvariante!"

Zu jeder Schleife eine Invariante. (Schwierigkeiten

der Programme kommen erst

durch Schleifen.)

Noch einmal analog

```
j = 1;
while (i > 0)
{
    j := j * i;
    i -- ;
    System.out.println("i=" + i + "j=" + j);
}
```

Bei Eingabe $k \geq 0$ k Durchläufe.

Für $1 \leq l \leq k$ gilt: Nach l-tem

Lauf

$$j_l = k \cdot (k-1) \cdots (k-l+1) = (k)_l \,,\; i_l = k-l.$$

(1.44)

Am Ende

$j_k = k(k-1)(k-2)\cdot\ldots\cdot 3\cdot 2\cdot 1 = k)_{k} = k!$

und $i_k = 0$. Für $k \geq 0$

sagt man $k!$ ist k Fakultät.

Beachte $0! = 1$.

wird die Schleife nicht

durchlaufen, also $k = 0$, dann

auch $j = k!$ und $i = 0$.

Ein Beispiel geschachtelter Schleifen

```
import Prog1Tools.IOTools

public class GeschSchl

{
  public static void main(String[] args)
  {
    int i, l, k;
    i = IOTools.readInteger("i=");
    while (i > 0)        // Äußere Schleife
    {
      k = 1;
      l = i;
      while (l > 0)      // Innere Schleife
      {
        k = l * k;
        l = l - 1;
      }                  // Haben i! in k.
      System.out.println(i + "! ist " + k);
      i--
    }
  }
}
```

Innere Schleife: auch vorher $k = 1$; $l = i$.

Innere i Durchläufe.

Nach h-tem Durchlauf $1 \leq h \leq 0$

$k_h = i(i-1) \quad (i-h+1)$, $l_h = i - k$

Schleifeninvariante

Nach i Läufen $k_i = i!$, $l = 0$.
gilt auch für $i = 0$.

Äußere Schleife: i innere i Durchläufe: für $i, i-1, \ldots, 1$.

Schleifeninvariante

Nach j-tem Lauf ist ausgehend bei Eingabe s in das i

$s!$ "ist" $s!$ // Nach dem vorherigen Durchlauf
$(s-1)!$ ist $(s-1)!$
$\vdots$
$(s-j+1)!$ ist $(s-j+1)!$

Das geht dann bis i mit 1
Vorbei ist.

Also bei Eingabe von i = 0
wird nicht 0! ausgegeben.
Änderung durch den Test
i >= 0 in der äußeren Schleife.