Software-Entwicklung 1

V04: Von Maschinen zu Objekten

Prof. Maalej & Team - @maalejw

Status der 3. Übungswoche

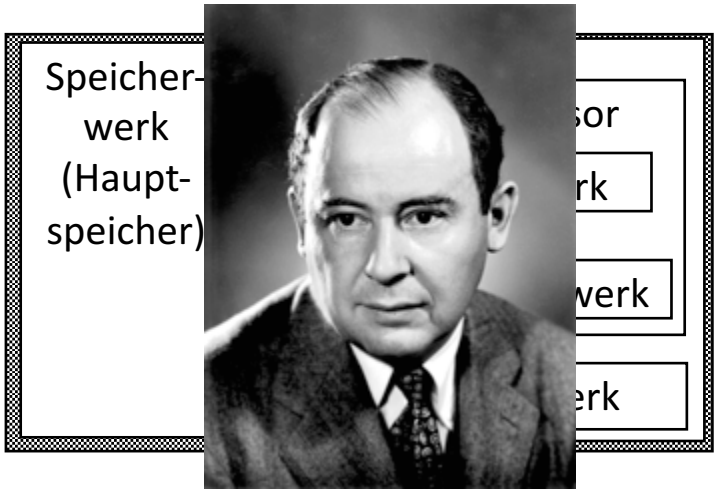
Zeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
Vormittag	Gruppe 1 **Erfüllt: 91%**	Gruppe 3 **Erfüllt: 81%**	Gruppe 5 **Erfüllt: 89%**	Gruppe 6 **Erfüllt: 88%**	Gruppe 8 **Erfüllt: 86%**
Nachmittag	Gruppe 2 **Erfüllt: 92%**	Gruppe 4 **Erfüllt: 87%**	Vorlesung	Gruppe 7 **Erfüllt: 93%**	

Überblick

1. **Übersetzung in Maschinensprachen**
2. Prozeduren und Methoden
3. Variablen und Zuweisungen

von Neumann-Rechner: Konzept (fast) aller Computer

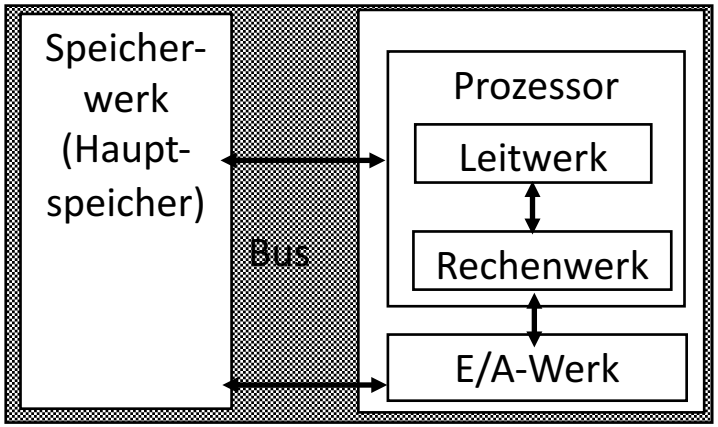
- Der Rechner besteht aus **4 Werken**.
- Die Rechnerstruktur ist **unabhängig** vom bearbeiteten Problem.
- Programme und Daten stehen **im selben Speicher**.



- Der Hauptspeicher ist in Zellen **gleicher Größe** unterteilt, die durchgehend **adressierbar** sind.
- Das Programm besteht aus Folgen von **Befehlen**, die generell **nacheinander** ausgeführt werden.
- Von der sequenziellen Abfolge kann durch **Sprungbefehle** abgewichen werden.
- Die Maschine benutzt **Binärcodes** für die Darstellung von Programm und Daten

Imperative Programme auf von Neumann-Maschinen

- Die **elementaren Operationen** eines von Neumann-Rechners:
  - Die CPU führt **Maschinenbefehle** aus
  - Sog. **Bus Befehle und Daten** werden vom Speicher in die CPU übertragen und die **Ergebnisse** zurück übertragen



- **Imperative Programmiersprachen** abstrahieren von diesen elementaren Operationen:
  - **Anweisungen** (engl.: statements) fassen Folgen von Maschinenbefehlen zusammen
  - **Variablen** (engl.: variables) abstrahieren vom physischen Speicherplatz

Ablaufsteuerung im Vergleich

von Neumann-Maschine:

- Aufeinanderfolgende Befehle stehen hintereinander im Speicher, werden vom Steuerwerk nach einander in den zentralen Prozessor geholt und dort geeignet decodiert und verarbeitet.
- Durch Sprungbefehle kann von der Reihenfolge der gespeicherten Befehle abgewichen werden.

Programmiersprache (Kontrollstrukturen):

- Innerhalb einer Methode:
  - Sequenz
  - Fallunterscheidung
  - Wiederholung
- Zusätzlich sind Methodenaufrufe spezielle Anweisungen, die ebenfalls in den sequenziellen Ablauf eingreifen.

Von der Klassendefinition zur Ausführung

- Eine Klassendefinition ist die **textuelle Beschreibung** einer Klasse.
- Sie liegt in einer Textdatei vor und kann mit **Editoren** bearbeitet werden.

- Wenn wir eine Klasse benutzen wollen (Exemplare erzeugen und diese benutzen), muss zuerst die menschenlesbare Klassendefinition in eine Form überführt wird, die ein Computer ausführen kann.
  - Diesen Vorgang nennen wir **Übersetzen** bzw. **Compilieren**.

- Nur durch eine **korrekt übersetzte** Klassendefinition entsteht eine Klasse, die bei der Ausführung eines Java-Programms zur Erzeugung von Exemplaren benutzt werden kann.

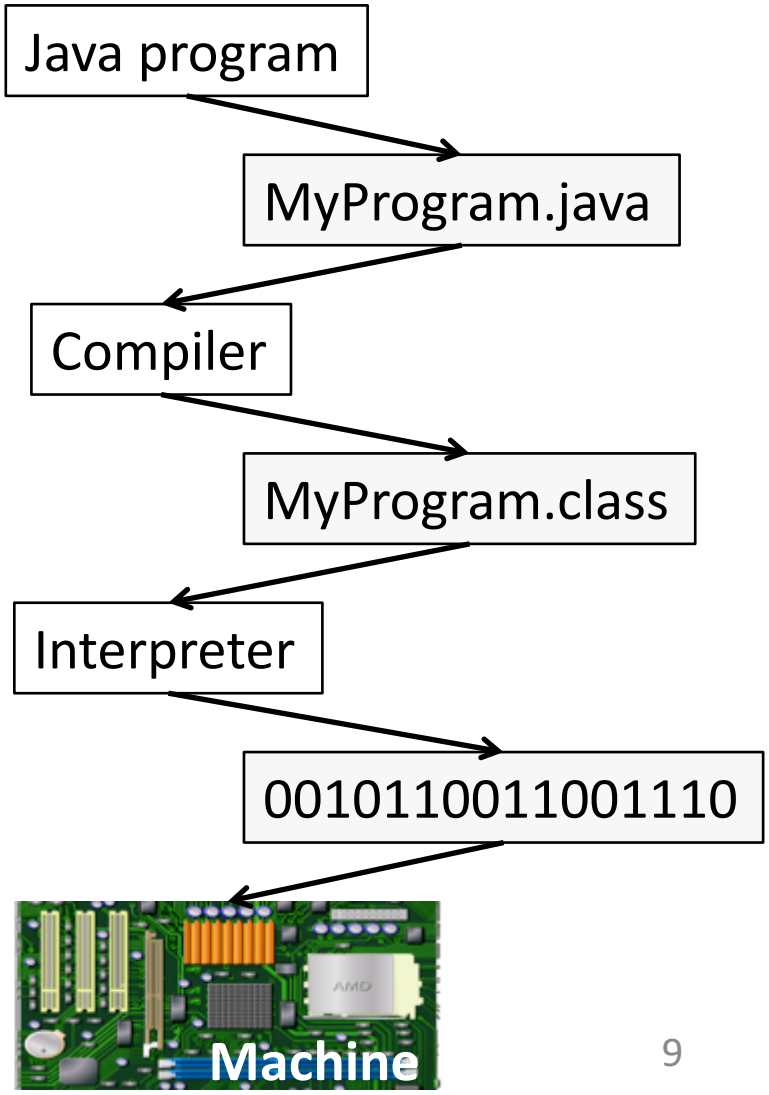
Verarbeitung von Programmen im Rechner

- **Compiler-Sprachen** (Bsp.: C++, Modula-2)
  - Alle Anweisungen eines Programms werden einmalig in eine Maschinensprache übersetzt und in dieser Sprache ausgeführt.
- **Interpretersprachen** (Bsp.: Lisp, Skriptsprachen wie PHP, Perl etc.)
  - Eine einzelne Anweisung eines Programms wird von einem anderen Programm (**Interpreter**) immer erst dann interpretiert (übersetzt), wenn sie ausgeführt werden soll.
- **Hybride Sprachen** (Bsp.: Java, C#)
  - Programme werden in eine Zwischensprache übersetzt, die sich gut für die Interpretation eignet.

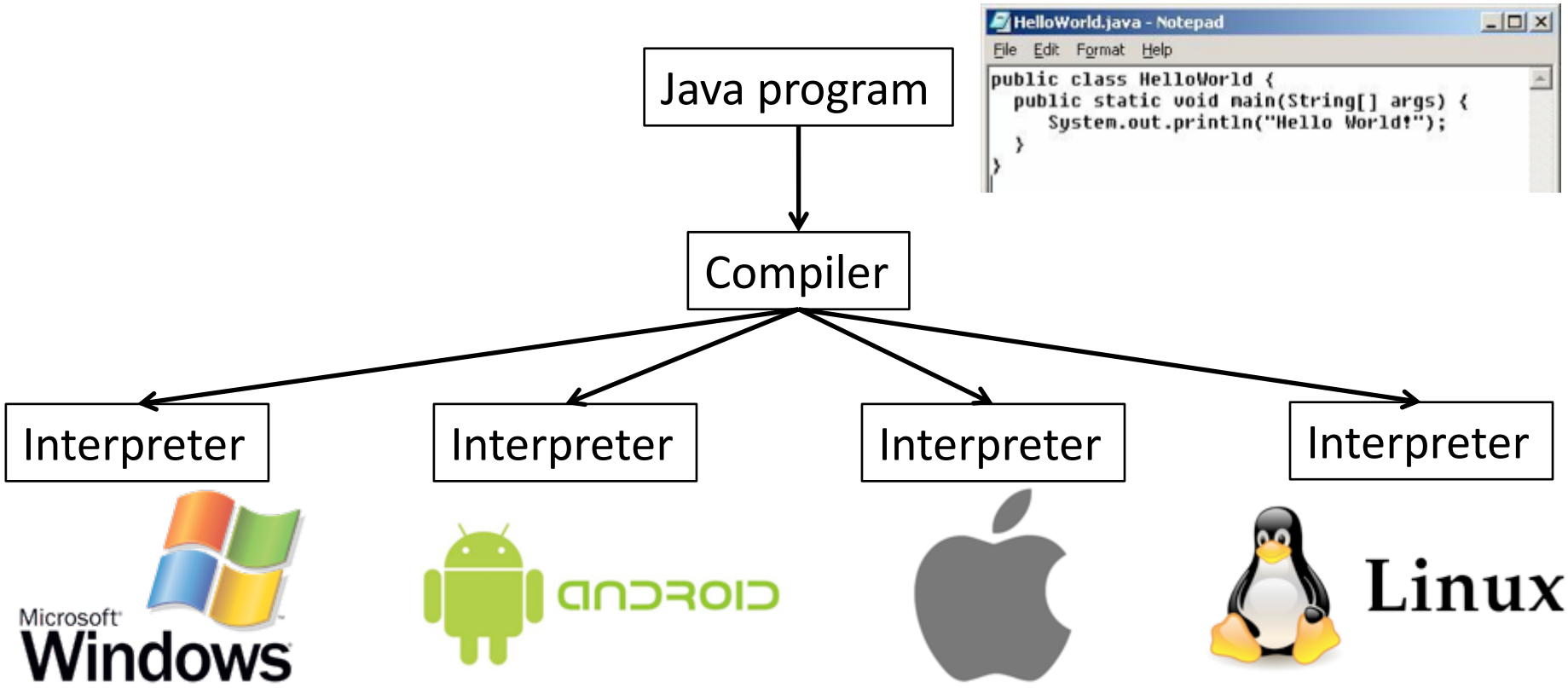
Hybride Verarbeitung in Java

Die hybride Verarbeitung bei Java ist eine Kombination:

- Der **Quelltext** wird von einem Compiler in **Zwischencode** transformiert, der **Java Bytecode** genannt wird.
- Dieser Zwischencode wird dem Interpreter übergeben, der sog. **Java Virtual Machine**.



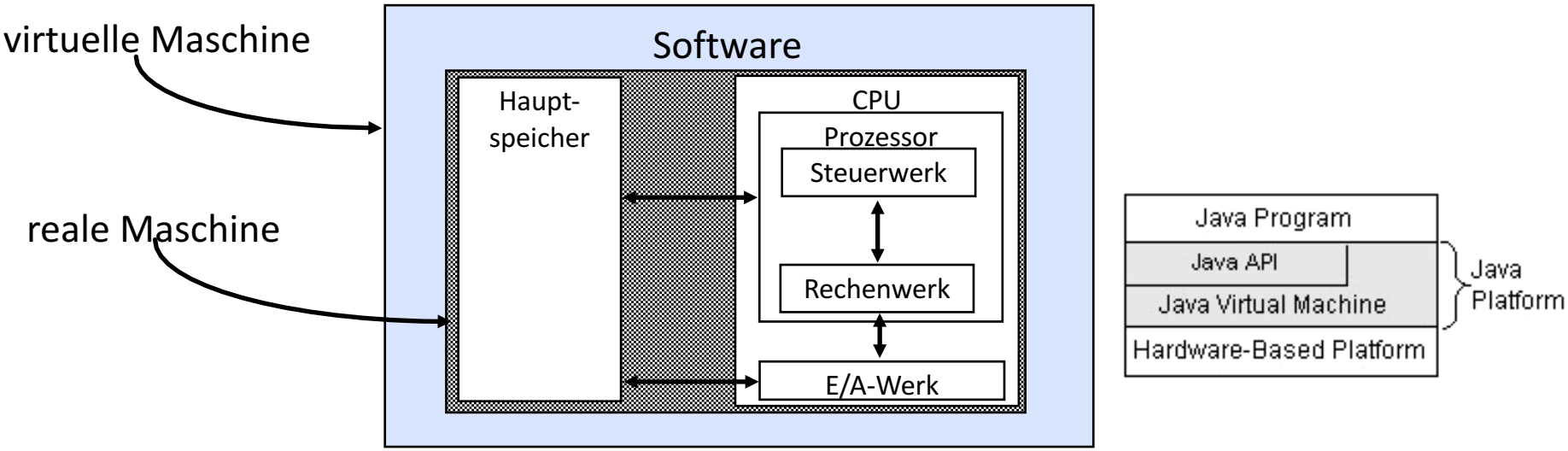
Java Virtual Machine



... You can think of Java bytecodes as the machine code instructions for the Java Virtual Machine (Java VM).

... Java bytecodes help make "write once, run anywhere" possible.

Virtuelle Maschinen: alles Software



- Ein **laufendes Programm** lässt sich verstehen als eine Folge von Befehlen für eine **Maschine**, die diese Befehle schrittweise ausführen kann.
- Diese Maschine muss nicht mechanisch oder elektronisch konstruiert werden.
- Es genügt, ihren Satz an **Maschinenbefehlen**, ihre **Speicherbereiche** und ihre **Kontrollstrukturen** festzulegen.
- Eine **virtuelle Maschine** ist eine durch Software definierte Maschine, die selbst auf einem Computer implementiert ist.

Imperative Programmierung

- Imperative Programmierung baut auf dem Konzept des v. Neumann-Rechners auf.
- Programme sind **Folgen von Anweisungen**.
- Ausführungsreihenfolge der Anweisungen ist durch die **textuelle Reihenfolge** oder durch **Sprunganweisungen** festgelegt.
- Höhere **Programmkonstrukte** fassen Anweisungsfolgen zusammen und bestimmen die Ausführungsreihenfolge.
- Benannte Variablen können Werte annehmen, die sich durch Anweisungen ändern lassen.

Beispiel: ein imperativer Algorithmus

Beispiel:

(1) Vergleiche zwei natürliche Zahlen *a* und *b*.

(2) Wenn *a* größer als *b* ist, setze das Ergebnis *max* gleich dem Wert von *a*.

(3) Sonst setze das Ergebnis *max* gleich dem Wert von *b*.

Auswertung des Beispiels:

- Algorithmus besteht aus einer Folge von Aktionen (hier vergleiche, setze)
- Jede Aktion bezieht sich auf *Variablen* (hier a,b,max), die durch die Aktion gegebenenfalls verändert werden.
- Jeder Variablen ist ein Typ zugeordnet (hier natürliche Zahlen).

Überblick

1. Übersetzung in Maschinensprachen
2. **Prozeduren und Methoden**
3. Variablen und Zuweisungen

Grundidee einer Methode / Prozedur

Eine Anweisungsfolge:

```
{
    int max;
    if (a > b)
    {
       max = a;
    }
    else
    {
       max = b;
    }
}
```

… erhält einen Namen und Parameter:

```
int maximum(int a, int b)
    {
       int max;
         if (a > b)
     {
        max = a;
     }
         else
     {
        max = b;
     }
         return max;
    }
```

… und kann **aufgerufen** werden:

```
…
int ergebnis = maximum(6, 9);
…
```

Hintergrund: der Prozedurbegriff

> •Pro|ze|dur [lat.-nlat.] *die; -, –en:* Verfahren, [schwierige, unangenehme] Behandlungsweise.

- Methoden in der OO Programmierung sind spezielle Ausprägungen des klassischen imperativen Prozedurbegriffs
- In der imperativen Programmierung sind Prozeduren das mächtigste Abstraktionsmittel
- Viele Prinzipien von Prozeduren gelten für die Methoden objektorientierter Sprachen wie Java

In imperativen Sprachen sind Prozeduren „frei“, d.h. sie sind nicht als Methoden einer Klasse und ihren Objekten zugeordnet.

Methoden/Prozeduren als Grundeinheiten eines Programms

- Methodenaufrufe (in OO Sprachen) oder Prozeduraufrufe (in imperativen Sprachen) bestimmen die Aktivitäten eines Programms
- Das gleiche Konzept:
  - Beide realisieren einen **Algorithmus** mit den Mitteln einer Programmiersprache
  - Sie sind eine **benannte Folge von Anweisungen**
  - Den **Namen** ist **„stellvertretend“ für diese Anweisungsfolge** anzusehen
  - Einer Methode/Prozedur können beim Aufruf unterschiedliche Informationen mitgegeben werden:
    - **Parameterübergabe**

Parametrisierung

- Damit die Anweisungsfolgen von Prozeduren nicht nur für einen bestimmten Fall zutreffen, wird ein zweiter Abstraktionsmechanismus eingesetzt: **Datenaustausch durch Parameter**.
- In maschinennahen Sprachen werden als Parameter Speicheradressen von Speicherzellen übergeben, in denen die Eingabe- oder Ausgabedaten stehen.
- Höhere Programmiersprachen verwenden das Konzept **getypter formaler Parameter**.

Formen der Parameterübergabe

- Unterscheidung der Art wie **Informationsaustausch** zwischen der aufrufenden Programmstelle und der Prozedur geregelt wird.
- Zwei Mechanismen zur Übergabe von Parametern:
  - Über **Eingabe-** oder **Wert-Parameter** (engl.: call by value):
    - Der Parameter dient zur Übergabe von Informationen an die Prozedur.
    - Der Aufrufer gibt die **aktuellen Parameter** in Form von **Ausdrücken** an.
    - Die Werte der Ausdrücke stehen der gerufenen Prozedur unter dem formalen Parameternamen zur Verfügung.
  - Über **Ausgabe-** oder **Variablen-Parameter** (engl.: call by reference):
    - Der Parameter kann entweder zusätzlich oder ausschließlich ein **Ergebnis** an den Aufrufer **liefern**.
    - An der Aufrufstelle muss eine **Variable** stehen, die das Ergebnis aufnehmen kann.

Formale und aktuelle Eingabe-Parameter

Definierende Stelle:

Formale Parameter

```
int maximum(int a, int b)
    {
      int max;
      if (a > b)
      {
          max = a;
      }
      else
      {
          max = b;
      }
      return max;
    }
```

- An der definierenden Stelle hat eine Methode/Prozedur sog. **formale Parameter** zur Datenübergabe.
- Im Beispiel a und b; beide sind vom Typ int.

Aufrufende Stellen:

Aktuelle Parameter

```
...
int ergebnis = maximum(6, 9);
int ergebnis2 = maximum(ergebnis, 2*x);
```

- An der aufrufenden Stelle erhält eine Methode/Prozedur **aktuelle Parameter**.
- Bei **Eingabe-Parametern** sind dies **Ausdrücke**, hier jeweils vom **passenden Typ** int.
- Die Werte der Ausdrücke werden den formalen Parametern **zugewiesen**.

Regeln bei der Parameterübergabe

Beim Prozeduraufruf werden die Werte der aktuellen Parameter an die formalen übergeben. Zur Übersetzungszeit wird überprüft:

- Der **Name** im Aufruf definiert die zu rufende Prozedur.
- Die **Anzahl** der aktuellen Parameter muss gleich der Anzahl der formalen sein.
- Die Bindung der jeweiligen Parameter wird entsprechend ihrer **Position** im Aufruf und in der Prozedurdeklaration vorgenommen.
- Die aktuellen Parameter müssen **typkompatibel** zu den formalen Parametern sein (d.h. zunächst typgleich).

Diese Regeln werden üblicherweise von einem Compiler überprüft!

Ergebnisprozedur

- Prozeduren, die die programmiersprachliche Form einer Funktion haben, nennen wir **Ergebnisprozeduren**.
  - Sie liefern ein Ergebnis, das an der aufrufenden Stelle direkt in einem Ausdruck verwendet werden kann.
  - In Java sind Methoden mit einem Ergebnistyp ungleich `void` für uns (vorläufig) die einzige Möglichkeit, Informationen von der gerufenen Prozedur an den Aufrufer zurück zu liefern.

Formales und aktuelles Ergebnis in Java

Definierende Stelle:

Formales Ergebnis

```
int maximum(int a, int b)
{
   int max;
   if (a > b)
   {
       max = a;
   }
   else
     ...
   return max;
 }
```

- An der definierenden Stelle kann eine Methode ein **formales Ergebnis** definieren.
- Steht dort hingegen **void**, ist die Methode keine Ergebnis-prozedur.
- Eine Ergebnisprozedur **muss** mit **return** ein Ergebnis liefern.

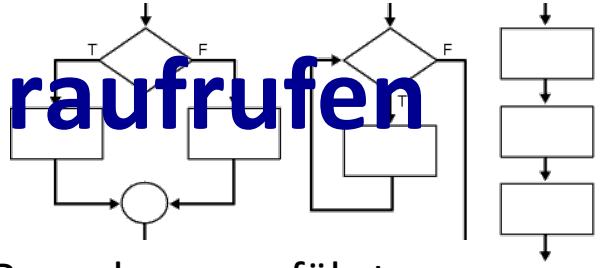
Aufrufenden Stellen:

Aktuelle Ergebnisse

```
...
int ergebnis = maximum(6, 9);
int ergebnis2 = maximum(ergebnis, 2*x);
```

- An der aufrufenden Stelle kann der Name der Ergebnisprozedur **stellvertretend** für das Ergebnis des Aufrufs angesehen werden.

Kontrollfluss bei Prozeduraufrufen



- Der **Prozeduraufruf** ist die explizite Anweisung, dass eine Prozedur ausgeführt werden soll.
- Eine Prozedur ist **aktiv**, nachdem sie gerufen wurde und in der Abarbeitung ihrer Anweisungen noch kein vordefiniertes Ende erreicht hat.
- Für den Prozeduraufruf in **sequenziellen imperativen** Sprachen ist charakteristisch:
  - Beim Aufruf wechselt die **Kontrolle** (d.h. die Abarbeitung von Anweisungen) vom Rufer zur Prozedur.
  - Dabei werden die (Werte der) **aktuellen Parameter** an die **formalen gebunden** (ihnen zugewiesen).
  - Prozeduren können **geschachtelt** aufgerufen werden. Dabei wird der Rufer unterbrochen, so dass die Kontrolle **immer nur bei einer Prozedur** ist; es entsteht eine **Aufrufkette**.
  - Nach der **Abarbeitung** der Prozedur kehrt die Kontrolle zum Rufer zurück; die Abarbeitung wird mit der Anweisung nach dem Aufruf fortgesetzt.

Kontrollfluss und Prozedur-Mechanismus

```
if (werteGegeben)
{
   volumen1 = berechne(n,r);
}
else
{
   volumen1 = 0;
}
gesamt = volumen1 + volumen2;
…
```

Prozedur-/Methodenaufruf

Prozedurdefinition

```
private int berechne(int x, int y)
{
   ...
   return result;
}
```

Kontrollfluss

Gemeinsamkeiten von Prozeduren und Methoden

- Beim Aufruf einer Methode wechselt der Kontrollfluss in die gerufene Methode, um nach dem Aufruf hinter die Aufrufstelle zurückzukehren.

- Beim Aufruf besteht die Möglichkeit, Parameter zu übergeben.

Unterschiede von Prozeduren und Methoden

- Weitere Eigenschaften von Methoden, die sie von simplen Prozeduren unterscheiden:
  - Eine Methode gehört immer **zu einem Objekt** (das beim Aufruf einer Methode angegeben werden muss);
  - Eine Methode kann immer auf **die Felder** des zugehörigen Objektes zugreifen.
  - Methoden haben eine **Sichtbarkeit** (in Java: `private` oder `public`).

Zwischenergebnis Prozedur/Methode

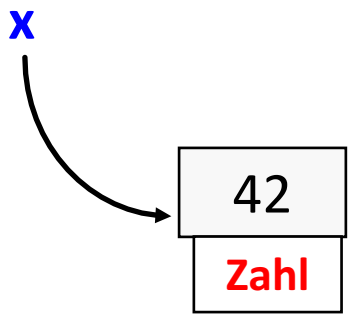
- Aufrufe von Methoden entsprechen imperativen Prozeduraufrufen.
- Prozeduren können **parametrisiert** werden; der Aufrufer übergibt aktuelle Parameter, die gerufene Methode bekommt diese als formale Parameter.
- Java: Die Werte der aktuellen Parameter werden beim Aufruf kopiert; die gerufene Methode arbeitet nur auf Kopien, die den formalen Parametern zugewiesen wurden.
- **Ergebnisprozeduren** liefern ein Ergebnis, das an der Aufrufstelle direkt in einem Ausdruck verwendet werden kann.
- Der Kontrollfluss wechselt von der aufrufenden Prozedur zur aufgerufenen Prozedur; nach dem Ende der Ausführung kehrt die Kontrolle zum Aufrufer zurück.
- Methoden sind ein „reicheres“ Konzept als Prozeduren: eine Methode ist immer einem Objekt zugeordnet und hat Zugriff auf die Felder dieses Objekts.

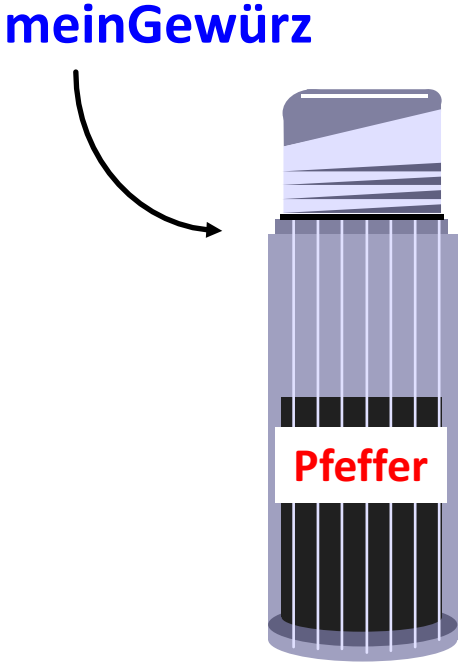
Überblick

1. Übersetzung in Maschinensprachen
2. Prozeduren und Methoden
3. **Variablen und Zuweisungen**

Variable

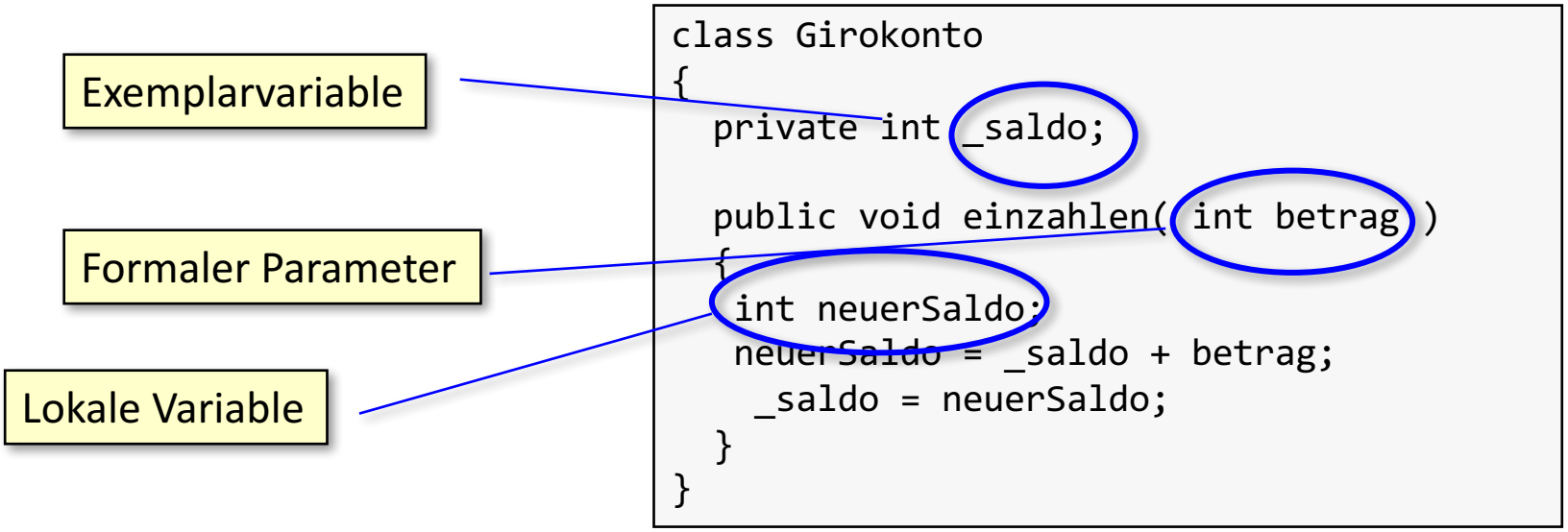
- Eine Variable hat
  - einen **Namen** (häufig auch: **Bezeichner**), über den sie angesprochen werden kann,
  - einen **Typ**
  - eine **Belegung** bzw. einen **Wert** während der Ausführung eines Programms





Drei Arten von Variablen

- **Exemplarvariablen** (Felder), die den Zustand von Objekten halten.
- **Formale Parameter**, mit denen Methoden parametrisiert werden können.
- **Lokale Variablen**, die als Hilfsvariablen in den Rümpfen von Methoden vorkommen können.



Deklaration der Variablenarten

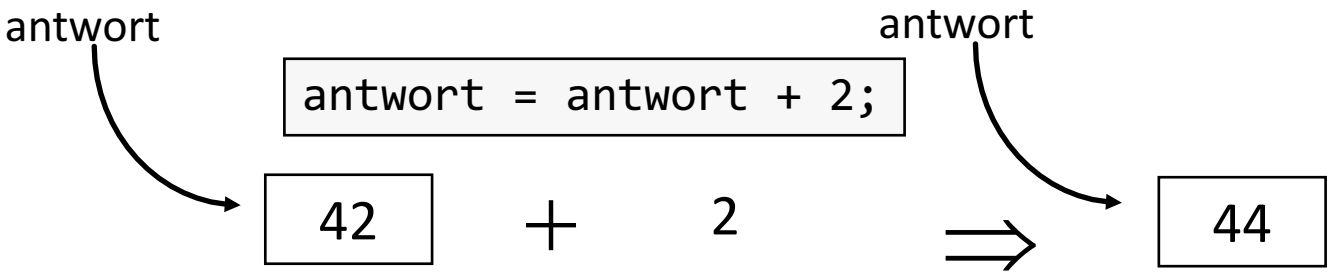
- **Vor der Verwendung** einer Variablen in muss sie **deklariert** werden. Dies geschieht durch Angabe des **Typs** und die Vergabe eines **Bezeichners**:
  - **Exemplarvariablen** werden in einer Klassendefinition für alle Exemplare der Klasse deklariert.
  - **Formale Parameter** werden jeweils in den **Köpfen** von Methoden deklariert.
  - **Lokale Variablen** werden in den **Rümpfen** von Methoden deklariert.

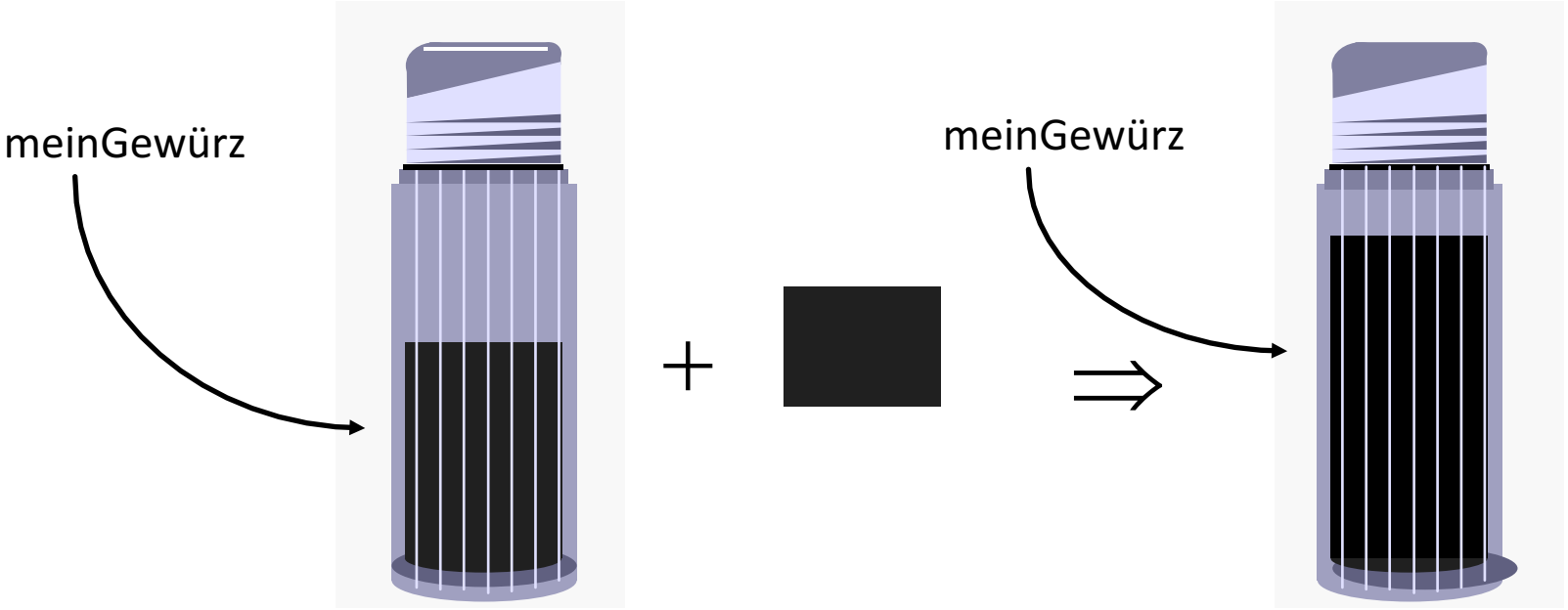
Anfangsbelegung der Variablenarten

- Die **Anfangsbelegung** einer Variablen kann bereits durch ihre Deklaration festgelegt sein:
  - **Exemplarvariablen** werden automatisch mit dem Standardwert (engl.: default value) des jeweiligen Typs belegt.
  - **Formale Parameter** werden mit den Werten der aktuellen Parameter eines Aufrufs belegt.
  - **Lokale Variablen** müssen explizit initialisiert oder erst zugewiesen werden, bevor ihre Belegung ausgelesen werden darf.

Veränderung eines Variablenwerts

- Wir können die **Belegung** einer Variablen **verändern**.
- Diese Aktion, bei der eine Variable unter Beibehaltung ihres Namens und Typs eine (neue) Belegung erhält, heißt **Zuweisung**.





Zuweisung

- Bei der **Zuweisung** wird ein **Ausdruck** ausgewertet und sein **Ergebnis** einer **Variablen** zugewiesen.
- Syntax-Schema der Zuweisung:

  `<Linke-Seite> <Zuweisungsoperator> <Rechte-Seite>`

  Rechte-Seite:
  - imperativ: meist arithmetische und boolesche Ausdrücke, Vergleiche und Zeichen oder Zeichenketten

  Zuweisungsoperator:
  - in Java (wie in C/C++) : `'='`
  - auch üblich (Pascal etc.): `':='`

  Linke-Seite: Bezeichner einer Variablen
- **Typkompatibilität**: Der Typ der linken Seite muss zum Typ des Zuweisungsausdrucks passen (zunächst, Typen müssen gleich sein).

Variablen in Zuweisungen

- Bei der **Zuweisung** sprechen wir oft von der **rechten** und der **linken Seite** einer Zuweisung *(*engl.: right-hand side - **RHS**, left-hand side - **LHS***)* oder von dem **L-Wert** und **R-Wert** *(*engl.: lvalue, rvalue*)*.
- **Bedeutung**:
  - L-Wert:
    Ist ein Bezeichner einer **Variablen**, der ein Speicherplatz zugeordnet ist. Dort wird der neu berechnete Wert gespeichert.
  - R-Wert:
    Ist ein **Ausdruck**, der einen Wert liefert. Ein R-Wert kann nur rechts vom Zuweisungsoperator stehen.
  - Im folgenden Beispiel haben die beiden Auftreten des Bezeichners **a** unterschiedliche Bedeutung:

    ```
    a = a + (3*i);
    ```

    Auf der linken Seite ist das **a** das Ziel, in dem etwas gespeichert werden soll; auf der rechten Seite ist es die Quelle eines Wertes, der mit anderen Werten in eine Berechnung einfließt.

Merke: Die Zuweisung ist komplizierter, als man auf den ersten Blick vermutet.

Zuweisung in Java

```
antwort = 40
antwort += 2
korrekt = (antwort == 42)
```

Operator	Funktion
==	Gleichheit
!=	Ungleichheit
=	Zuweisung

Der Gleichheitstest wird häufig mit der Zuweisung verwechselt:

```
saldo =  0  // Zuweisung
saldo == 0  // Gleichheit
saldo != 0  // Ungleichheit
```

Zusammenfassung

1. Java- Programme werden in Maschinensprache (von Neumann Architektur) kompiliert, interpretiert und ausgeführt.

2. Anweisungen und Prozeduren fassen Anweisungsfolgen zusammenfassen und bestimmen die Ausführungsreihenfolge.

3. Methoden sind das OO Pendant von Prozeduren. Beide können parametrisiert werden und können was zurückliefern.

4. **Variablen** müssen deklariert werden und können dynamisch ihre Belegung durch Zuweisungen ändern.