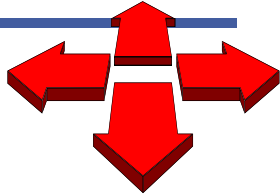


Imperative Grundkonzepte

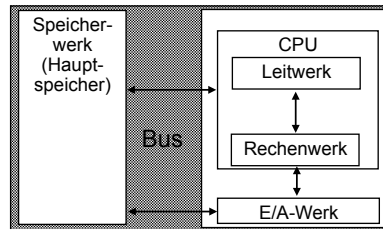
- **Grundlage: der von Neumann-Rechner**
- **Der Prozedurbegriff**
 - Parameter
 - Ergebnisse
 - Kontrollfluss
- **Übersetzung**
 - Hybride Sprachen
 - Virtuelle Maschinen



SE1 – Level 1 2

Konzept (fast) aller Computer: der von Neumann-Rechner

- 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.

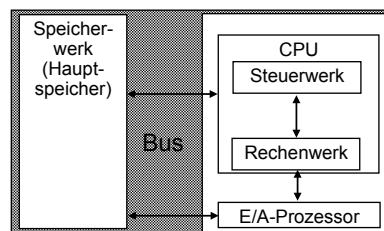


SE1 - Level 1

3

Imperative Programme auf von Neumann-Maschinen

- Die **elementaren Operationen** eines **von Neumann-Rechners**:
 - Die CPU führt **Maschinenbefehle** aus.
 - Dabei werden über den sog. **Bus Befehle und Daten** 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.



SE1 - Level 1

4

Ablaufsteuerung im Vergleich



- Ablaufsteuerung in der **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.
- Ablaufsteuerung auf **Ebene der Programmiersprache (Kontrollstrukturen)**:
 - Innerhalb einer Methode:
 - **Sequenz**
 - **Fallunterscheidung**
 - **Wiederholung**
 - Zusätzlich sind **Methodenaufrufe** spezielle Anweisungen, die ebenfalls in den sequenziellen Ablauf eingreifen.

Methoden/Prozeduren als Grundeinheiten eines imperativen Programms

- Methodenaufrufe (in objektorientierten Sprachen) oder Prozeduraufrufe (in klassischen imperativen Sprachen) bestimmen die Aktivitäten eines Programms.
- Hinter Methoden und Prozeduren steht das gleiche Konzept:
 - **Fachlich** realisieren sie einen **Algorithmus** mit den Mitteln einer Programmiersprache.
 - **Softwaretechnisch** sind sie eine **benannte Folge von Anweisungen**.
 - Die Grundidee ist, den **Namen** der Methode oder Prozedur „stellvertretend“ für diese **Anweisungsfolge** anzusehen.
 - Einer Methode/Prozedur können beim Aufruf unterschiedliche Informationen mitgegeben werden. Dies geschieht durch Konzepte der **Parameterübergabe**.



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

Hintergrund: zum Begriff „Algorithmus“

(Quelle: http://www.info-wsf.de/index.php/Definition_Algorithmus)

Ein **Algorithmus** ist eine eindeutige, ausführbare Folge von Anweisungen endlicher Länge zur Lösung eines Problems.

Eigenschaften:

- **Allgemeinheit**
Ein Algorithmus ist allgemeingültig, d. h. er löst eine Vielzahl von Problemen (der gleichen Art). Die Auswahl eines einzelnen konkreten Problems erfolgt über Eingabedaten oder Parameter.
- **Eindeutigkeit**
An jeder Stelle des Algorithmus muss eindeutig festgelegt sein, was zu tun ist und welcher Schritt der nächste ist. Dafür muss jede Anweisung unmissverständlich formuliert sein.
- **Ausführbarkeit**
Jede einzelne Anweisung eines Algorithmus muss vom Computer (vom Mensch) ausführbar sein.
- **Endlichkeit (Finitheit)**
Die Beschreibung eines Algorithmus besitzt eine endliche Länge, d. h. er besteht aus einer begrenzten Anzahl von Anweisungen mit begrenzter Länge. Zudem darf ein Algorithmus zu jedem Zeitpunkt für seine Daten nur endlich viel Platz belegen.

Für die Praxis werden Algorithmen häufig auf die folgenden Eigenschaften eingeschränkt:

- **Determiniertheit**
Wird ein Algorithmus mit den gleichen Eingabewerten und Startbedingungen wiederholt, so liefert er stets das gleiche Ergebnis.
- **Terminierung**
Der Algorithmus ist nach endlich vielen Schritten beendet, d. h. er liefert ein Ergebnis oder hält an.

Algorithmus? al-Chwarizmi!

(Quelle: Wikipedia)

Abu Dscha'far Muhammad ibn Musa al-Chwarizmi, Var. Chwarazmi; (arabisch **أبو جعفر محمد بن موسى الخوارزمي**; DMG Abū Ga'far Muḥammad b. Mūsā al-Ḥwārizmī, Var. al-Ḥwārazmī; * um 780; † zwischen 835 (?) und 850) war ein choresmischer Universalgelehrter, Mathematiker, Astronom und Geograph während der abbasidischen Blütezeit, der zwar aus dem **iranischen Choresmien** stammte, jedoch den größten Teil seines Lebens in **Bagdad** verbrachte und dort im „Haus der Weisheit“ tätig war.

Von seinem Namen leitet sich der Begriff Algorithmus ab.

Ursprünglich Bedeutung von „Algorithmus“: genau definiertes Rechenverfahren



al-Chwarizmi auf einer sowjetischen Briefmarke anlässlich seines 1200. Geburtstags.

Hintergrund: der Prozedurbegriff

- Die **Methoden** in der objektorientierten Programmierung sind spezielle Ausprägungen des klassischen imperativen **Prozedurbegriffs**.
- In der imperativen Programmierung sind Prozeduren das **mächtigste Abstraktionsmittel**.
- Viele Prinzipien von Prozeduren gelten analog für die Methoden objektorientierter Sprachen wie Java.
- Im Folgenden wird der Prozedurbegriff ausführlicher erläutert.



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

Die Grundidee einer Methode / Prozedur

Eine **Anweisungsfolge**:

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

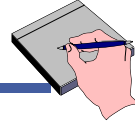
... erhält einen **Namen** und **Parameter**:

```
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);
...
```

Parametrisierung

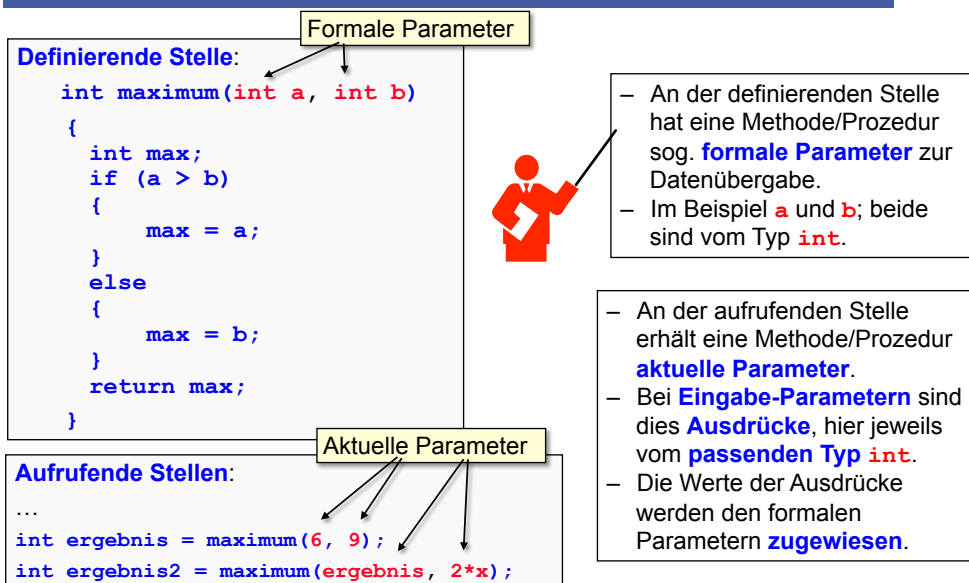


- Damit die Anweisungsfolgen von Prozeduren nicht nur für einen bestimmten Fall zutreffen, wird ein zweiter Abstraktionsmechanismus eingesetzt - **Datenaustausch durch Parameter**.
- In maschinennahen Sprachen werden als Parameter Speicheradressen von Speicherzellen übergeben, in denen die Eingabe- oder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

SE1 - Level 1

11

Formale und aktuelle Eingabe-Parameter



SE1 - Level 1

12

Regeln bei der Parameterübergabe



- Beim Prozeduraufruf werden die aktuellen Parameter an die formalen gebunden (zugewiesen). 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!

SE1 – Level 1

13

Formales und aktuelles Ergebnis in Java

Definierende Stelle:

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

Aktuelles Ergebnis

Formale Ergebnisse

Aufrufenden Stellen:

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

- Eine Methode liefert mit der `return`-Anweisung ein **aktuelles Ergebnis** an den Aufrufer zurück.
- Der Compiler stellt sicher, dass in allen Pfaden durch den Code eine `return`-Anweisung steht.

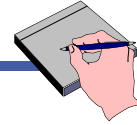
- An der aufrufenden Stelle kann der Aufruf der Methode **stellvertretend** für das Ergebnis des Aufrufs angesehen werden.



SE1 – Level 1

14

Kontrollfluss bei Prozeduraufrufen

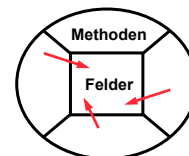


- Der **Prozeduraufruf** ist die explizite Anweisung, dass eine Prozedur ausgeführt werden soll.
- Eine Prozedur ist **aktiv**, nachdem sie gerufen wurde und in der Abarbeitung ihrer Anweisungen noch kein vordefiniertes Ende erreicht hat.
- Für den Prozeduraufruf in **sequenziellen imperativen** Sprachen 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.

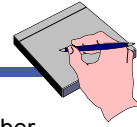
Unterschiede zwischen Prozeduren und Methoden



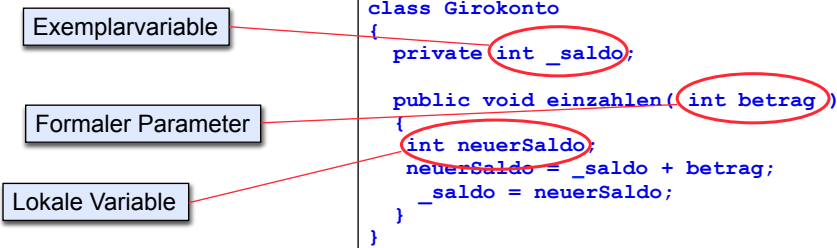
- Die **Methoden** der objektorientierten Programmierung haben die gleichen Eigenschaften wie die **Prozeduren** der imperativen Programmierung:
 - Beim Aufruf einer Methode **wechselt der Kontrollfluss** in die gerufene Methode, um nach dem Aufruf hinter die Aufrufstelle zurückzukehren.
 - Beim Aufruf besteht die Möglichkeit, **Parameter** zu übergeben.
- Darüber hinaus haben Methoden jedoch **weitere Eigenschaften**, 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**).



Drei Arten von Variablen



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



SE1 – Level 1

17

Zwischenergebnis Prozedur/Methode



- Aufrufe von Methoden entsprechen weitgehend **imperativen Prozeduraufrufen**.
- Prozeduren können **parametrisiert** werden; der Aufrufer übergibt **aktuelle Parameter**, die gerufene Methode bekommt diese als **formale Parameter**.
- **Java kennt nur Wert-Parameter**: Die Werte der aktuellen Parameter werden beim Aufruf kopiert; **die gerufene Methode arbeitet nur auf Kopien**, die den formalen Parametern zugewiesen wurden.
- 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.

SE1 – Level 1

18

Verständnisfragen (1)

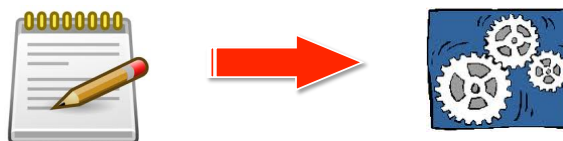
- Gegeben sei folgende Klassendefinition:

```
class A
{
    public int m(int a, int b, int c)
    {
        System.out.print("Hallo Welt! ");
        System.out.println(a + b + c);
        return 0;
    }
}
```

- Frage: Wie viele aktuelle Parameter enthält diese Klassendefinition?
 - 1
 - 2
 - 3
 - 4
 - 5
 - 6
 - 7
 - 8

Von der Klassendefinition zur Ausführung

- Eine **Klassendefinition** ist lediglich die **textuelle Beschreibung** einer Klasse. Sie liegt in einer **Textdatei** vor und kann mit einem **Editor** bearbeitet werden.
- Wenn wir eine Klasse benutzen wollen (indem wir Exemplare von ihr erzeugen und diese Exemplare benutzen), müssen wir zuerst dafür sorgen, dass 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



Die Programme höherer Programmiersprachen werden nach drei Ansätzen verarbeitet:

- **Compiler-Sprachen** (Bsp.: C++, Modula-2)
 - Alle Anweisungen eines Programms werden einmalig in eine Maschinensprache übersetzt und dann direkt in dieser Sprache ausgeführt.
- **Interpretersprachen** (Bsp.: Lisp, Skriptsprachen wie PHP, Perl etc.)
 - Eine einzelne Anweisung eines Programms wird von einem anderen Programm (dem **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 bzw. JIT-Compilierung 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**.

... each Java program is both compiled and interpreted. With a **compiler**, you translate a Java program into an intermediate language called **Java bytecode** -- the platform-independent code interpreted by the Java interpreter. With an **interpreter**, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

