

PROGRAMMIERGRUNDLAGEN

Kurzbeschreibung

Diese Kurseinheit liefert die Grundlagen, einfache Programme in der Sprache Java zu entwickeln und auf einem Rechner ausführen zu lassen.

Schlüsselwörter

Programmerstellung, Grundsymbol, Variable, Anweisung, bedingte Anweisung, Schleife, Programmstruktur, Ein-/Ausgabe, Methode, Datentyp, Zahlen, Zeichen

Lernziele

1. Das grundsätzliche Vorgehen, das der Programmierung und der Ausführung eines Programms auf dem Rechner zugrunde liegt, wird verstanden.
2. Elementare Sprachelemente, wie Variablen, Zuweisungen, Anweisungen und Methoden sind bekannt und können zur Erstellung von Programmen genutzt werden.
3. Ein vollständiges und lauffähiges Java-Programm kann geschrieben und auf dem Rechner ausgeführt werden.

Hauptquellen

- Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Inhaltsverzeichnis

1	PROGRAMM UND PROGRAMMIEREN.....	3
1.1	Vorgehen bei der Programmerstellung und -ausführung.....	4
1.2	Programmsyntax.....	6
2	GRUNDLEGENDE SPRACHELEMENTE	8
2.1	Grundsymbole.....	9
2.2	Variablen und Zuweisung.....	12
2.3	Ausdrücke.....	14
2.3.1	Arithmetische Ausdrücke	15
2.3.2	Boolesche Ausdrücke	16
2.4	Anweisungen zur Ablaufsteuerung.....	17
2.4.1	Bedingte Anweisung	17
2.4.2	Schleife.....	19
3	PROGRAMMSTRUKTUR.....	21
3.1	Grundstruktur.....	21
3.2	Ein-/Ausgabe und Programmausführung.....	22
3.2.1	Klassen In und Out	23
3.2.2	Programmübersetzung und -ausführung.....	25
4	METHODEN	26
4.1	Deklaration und Aufrufketten.....	27
4.2	Parameter	28
4.3	Funktionen	30
4.4	Lokale und globale Namen.....	31
5	DATENTYPEN.....	34
5.1	Gleitkommazahlen.....	35

5.2 Zeichen	37
VERZEICHNISSE	42
Abkürzungen und Glossar	42
Index	43
Informationen und Interaktionen	43
Literatur	45

- PROGRAMMERSTELLUNG UND PROGRAMMABLAUF
 - Vorgehen bei der Programmierung, Programmsyntax
- GRUNDLEGENDE BESTANDTEILE EINES PROGRAMMS
 - Variablen, Verzweigungen, Schleifen
- PROGRAMMSTRUKTUR
 - Grundstruktur, Ein-/Ausgabe, Methoden
- DATENTYPEN
 - Gleitkommazahlen, Zeichen

Information 1: PROGRAMMIERGRUNDLAGEN

1 PROGRAMM UND PROGRAMMIEREN

Programmieren bedeutet, ein Problem so zu beschreiben, dass es mittels eines Rechensystems gelöst werden kann.

Der Beschreibung liegt ein Algorithmus (siehe Kurseinheit GRUNDBEGRIFFE DER INFORMATIK, [C&M-GI]) zugrunde, der in einer Sprache verfasst ist, die die Eigenschaft hat, auf dem Rechner ausgeführt werden zu können. Eine solche algorithmische Sprache heißt Programmiersprache. In der vorliegenden Kurseinheit werden die Programmiergrundlagen, die zur Erstellung einfacher Programme benötigt werden, am Beispiel der Sprache Java vermittelt [Mö03].



- Programmieren
 - ist eine kreative, anspruchsvolle und interessante Tätigkeit
 - sollte jeder Informatiker beherrschen, da hierdurch erst die Arbeitsweise eines Informatiksystems verstanden wird
- Programm
 - ist ein in einer Programmiersprache verfasster Algorithmus
 - ist auf einem Rechner ablauffähig
 - besteht aus Daten und Befehlen
- Sprachen

	Beispiel
• Höhere Programmiersprachen	_____
• Maschinennahe Sprachen	_____
• Modellierungssprachen	_____
• Datentransformationssprachen	_____

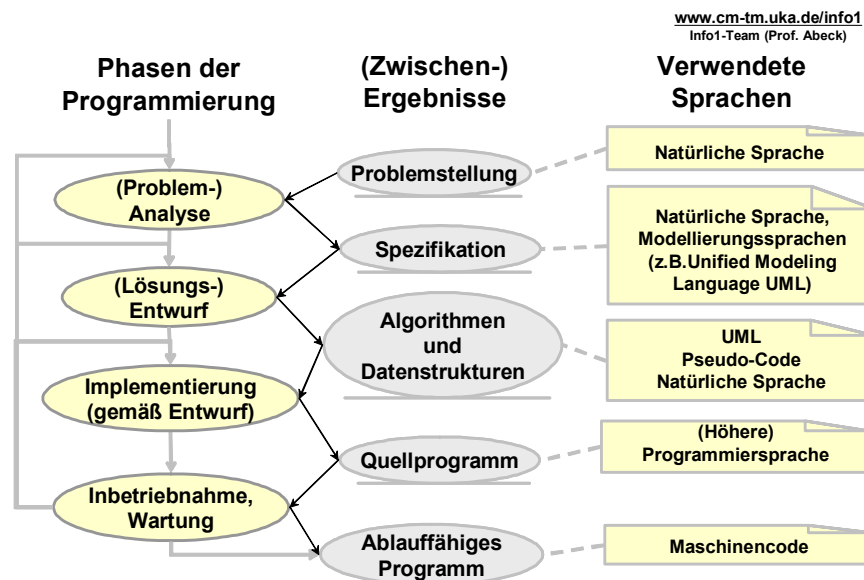
Interaktion 1: PROGRAMMERSTELLUNG UND PROGRAMM - Einführung

Das Programmieren umfasst sehr viel mehr als das reine Codieren einer Lösungsvorschrift in der vorgegebenen Programmiersprache. Das Finden und Analysieren einer Lösungsvorschrift sowie das Umsetzen in ein Programm sind kreative und anspruchsvolle Tätigkeiten (siehe Interaktion 1).

Zum Programmieren ist eine Sprache notwendig, in der die Programme so erstellt werden können, dass diese von einem Rechner "verstanden" und bearbeitet werden können. Mit Programmiersprachen sind meist die so genannten höheren Programmiersprachen gemeint, wozu u.a. Java oder C gehören. Daneben existieren aber noch einige weitere Typen von Sprachen, die ebenfalls die Eigenschaft erfüllen, auf einem Rechner (zumindest teilweise) ausgeführt werden zu können. In Interaktion 1 ist zu jeder dieser Sprachtypen ein Beispiel zu nennen.

1.1 Vorgehen bei der Programmerstellung und -ausführung

Bei der Erstellung eines Programms kommen in der Praxis mehrere Sprachen zum Einsatz, wie in Information 2 [C&M-SMU] aufgezeigt wird.



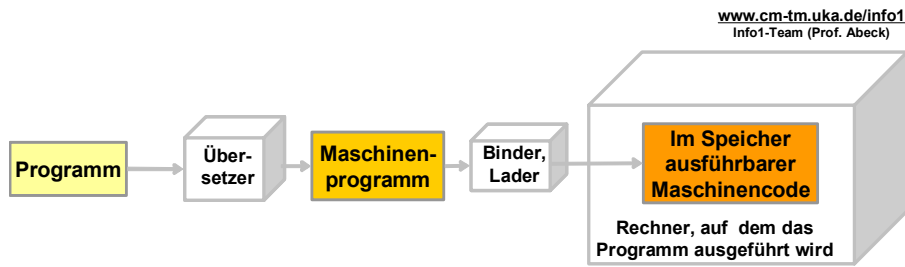
- Das Programmieren erfolgt iterativ, d.h. man springt häufig in eine der vorhergehenden Phasen zurück

Information 2: Dem Programmieren zugrunde liegendes Vorgehen

Die Abbildung verdeutlicht, welche Phasen bei der Programmierung zu durchlaufen sind. Bei der Erstellung von größeren Programmen ("Programmieren im Großen") wird häufig der Begriff der Software-Entwicklung bzw. *Software Engineering* verwendet.

Durch den Ablauf wird aufgezeigt, dass vor der eigentlichen Implementierung – also der Beschreibung der Problemlösung in einer Programmiersprache – zunächst eine Analyse- und eine Entwurfsphase steht. Ist das Problem "klein", fallen diese Phasen entsprechend kurz aus. Da reale Informatikprobleme aber üblicherweise groß sind, kommt den beiden ersten Phasen in der Praxis eine erhebliche Bedeutung zu.

Neben den Phasen, die in Iterationen durchlaufen werden, sind in Information 2 auch die (Zwischen-) Resultate und die Sprachen aufgeführt, in denen diese Resultate vorliegen können. Neben der Programmiersprache Java wird heute intensiv die *Unified Modeling Language* (UML, [Oe01]) zur Analyse und zum Entwurf genutzt.



- Übersetzer, Binder und Lader sind auf einem Rechner ablauffähige Programme
- der Übersetzer transformiert das Programm in ein bedeutungstreues Maschinenprogramm
- der Binder wird benötigt, wenn das Programm mit anderen Programmteilen gebunden werden soll
- der Lader lädt das Maschinenprogramm in den Speicher des (von-Neumann-) Rechners, auf dem das Programm ausgeführt wird

Information 3: Vom Quell-Programm zum ablauffähigen Programm

Information 3 zeigt, welche Schritte erforderlich sind, damit ein Programm auf einem in der Kurseinheit INFORMATIK I IM ÜBERBLICK [C&M-IÜ] behandelten (von-Neumann-) Rechner ablauffähig ist. Die hierzu benötigten Werkzeuge wie Übersetzer, Binder und Lader sind ebenfalls Programme.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

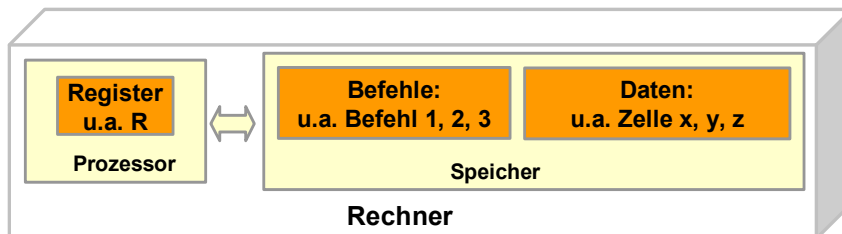
- Folgender Programmbefehl soll ausgeführt werden:

```
z = x + y;
```

- Übersetzung in Maschinensprache (Pseudo-Code)
- Befehl 1: Lade Zelle x in ein Register R
 Befehl 2: Addiere Zelle y zu R
 Befehl 3: Speichere R in Zelle z

Beispiel

R	x	y	z
_	9	7	_
_	_	_	_
_	_	_	_



Interaktion 2: Ausführung eines Programmbefehls

In Interaktion 2 wird anhand eines einfachen Programmbefehls skizziert, wie die Umsetzung in ein Maschinenprogramm und die Ausführung auf dem Rechner erfolgt. An dem Beispiel kann man sich den in [C&M-GI] beschriebenen von-Neumannschen Flaschenhals klar machen.

Befehle und Daten sind jeweils binär codiert im Speicher abgelegt. Die Bits werden zu Bytes (8 Bit) gruppiert, Bytes wiederum zu Worten (2 oder 4 Byte) bzw. zu Doppelworten (2 Worte).

1.2 Programmsyntax

Der oben verwendete Programmbefehl

$$z = x + y;$$

gehört zum Sprachschatz der Programmiersprache Java. Die Festlegung des Sprachschatzes und damit des Aussehens – der Syntax – eines (Java-) Programms erfolgt durch eine so genannte Grammatik.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Syntax einer Sprache legt mittels Regeln fest, welche Sätze zu der Sprache gehören
- Eine Grammatik ist eine Menge von Syntaxregeln, durch die ein Sprachschatz beschrieben wird
 - Erweiterte Backus-Naur-Form (EBNF) ist eine weit verbreitete Schreibweise für Grammatiken
- Beispiel: EBNF zur Beschreibung einer Dezimalziffer

```
Ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
Zahl = Ziffer {Ziffer}.
```

- Unterscheidung von so genannten Terminal-Symbolen und Nichtterminal-Symbolen
 - "0" ... "9" Terminal-Symbole
 - Zahl, Ziffer Nichtterminal-Symbole

Information 4: Syntaxfestlegung mittels einer Grammatik

Zur Beschreibung einer Grammatik hat sich die nach den Informatikern John Backus und Peter Naur benannte Erweiterte Backus-Naur-Form, kurz EBNF, in der Praxis durchgesetzt. Wie Information 4 zeigt, lässt sich durch die EBNF z.B. die Syntax einer vorzeichenlosen Dezimalzahl kompakt beschreiben.

Die Backus-Naur-Form (BNF) besteht aus so genannten Terminal-, Nichtterminalsymbolen und den zwei Metazeichen =, . und |. Die Erweiterung der BNF zur EBNF betrifft gewisse Metazeichen. Alle Metazeichen der EBNF werden im Folgenden beschrieben.

- Metazeichen dienen zur Beschreibung der Grammatikregeln, durch die die zu einer Sprache gehörenden Sätze festgelegt werden

= trennt linke und rechte Regelseite
 . schließt Regel ab

| trennt Alternativen
 • Beispiel: $x | y$ beschreibt: x, y

() klammert Alternativen
 • Beispiel: $(x | y) z$ beschreibt: xz, yz

[] wahlweises Vorkommen
 • Beispiel: $[x] y$ beschreibt: xy, y

{ } 0-maliges bis n-maliges Vorkommen
 • Beispiel: $\{x\} y$ beschreibt: $y, xy, xxy, xxxxy, \dots$

Information 5: Metazeichen der Erweiterten Backus-Naur-Form (EBNF)

Die Metazeichen (siehe Information 5) geben die Syntax der EBNF-Regeln vor. Die Erklärungen und die zu einigen Zeichen gegebenen Beispiele beschreiben, was die Metazeichen bewirken (Semantik der Metazeichen).

- Ein Syntaxdiagramm ist eine graphische Darstellungsform einer Grammatik

- Jedes EBNF-Metazeichen wird hierzu in Form einer graphischen Darstellung umgesetzt



beschreibt: _____

beschreibt: _____



beschreibt: _____

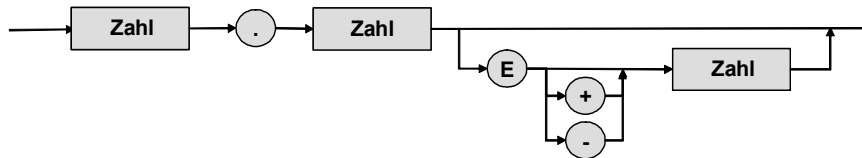
beschreibt: _____

Interaktion 3: Syntaxdiagramm

Neben der textuellen Beschreibung besteht mit den Syntaxdiagrammen die Möglichkeit, EBNF-Regeln graphisch darzustellen. Dabei kann jedes Metazeichen durch eine graphische Darstellung umgesetzt und in einem Gesamtdiagramm zusammengestellt werden.

Die Umsetzung jedes Metazeichens in seine graphische Darstellung zeigt Interaktion 3.

- Unterscheidung der beiden Symbolarten in Syntaxdiagrammen
 - Terminale in Kreis-Symbolen
 - Nichtterminale in Rechteck-Symbolen



- Die untere EBNF-Grammatik ist so zu ergänzen, dass sie die durch obiges Syntaxdiagramm dargestellte Sprache beschreibt

Ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

Zahl = Ziffer {Ziffer}.

Gleitkommazahl = _____

Interaktion 4: Syntaxdiagramm und Grammatik zu Gleitkommazahlen

Am Beispiel der Gleitkommazahlen wird in Interaktion 4 ersichtlich, wie sich die graphischen Elemente zu einem Syntaxdiagramm zusammenstellen lassen. In diesem Beispiel sind '.', 'E', '+' und '-' weitere Terminalsymbole, die zu den in Information 4 aufgeführten Symbolen hinzukommen. In der anzugebenden Grammatik ist neben Ziffer und Zahl ein drittes Nichtterminal-Symbol – hier Gleitkommazahl genannt – einzuführen.

In den folgenden Kapiteln wird die EBNF zur Beschreibung eines Ausschnitts der Sprache Java genutzt.

2 GRUNDLEGENDE SPRACHELEMENTE

Bereits das Schreiben einfachster Programme setzt voraus, dass gewisse Sprachelemente der verwendeten Programmiersprache bekannt sind. In Information 6 sind solche grundlegenden Sprachelemente am Beispiel von Java aufgelistet [Mö03].

- Grundsymbole
 - Schlüsselwörter, Zahlen, Zeichen
- Variablen und Zuweisung
 - Deklaration, Standardtypen
 - Initialisierung
- Ausdrücke
 - Arithmetische Ausdrücke
 - Boolesche Ausdrücke
- Anweisungen zur Ablaufsteuerung
 - Bedingte Anweisung
 - Schleife

Information 6: GRUNDLEGENDE SPRACHELEMENTE – Überblick

Die Grundsymbole liefern die Grundlage der Programmiersprache. Mittels dieser Symbole lassen sich die Datenelementen (z.B. Konstanten, Variablen) und Befehle (z.B. Zuweisungen, Schleifen) bilden.

2.1 Grundsymbole

Ein Programm ist eine Folge von Symbolen, die nach bestimmten Regeln aufgebaut sind.

- Namen bezeichnen Dinge (Variablen, Methoden, ...) in einem Programm
- Syntaktische Festlegungen
 1. erlaubte Zeichen: Buchstaben, Ziffern, "_" oder "\$"
 2. erstes Zeichen entweder ein Buchstabe oder "_" oder "\$"
 3. Groß-/Kleinschreibung ist signifikant
 4. beliebige endliche Länge
- Beispiele für syntaktisch korrekt gebildete Namen
 - x, _x, X1, first, FIRST, firstCourse
- Zu beachten: Nicht alle syntaktisch korrekten Namen machen Sinn
 - eine sinnvolle Namensgebung zählt zu den Eigenschaften, die ein gutes Programm ausmachen

Information 7: Namen

Ein zentrales Grundsymbol in Programmen sind die Namen (siehe Information 7).

Durch Namen legt der Programmierer fest, wie die im Programm eingeführten "Dinge" – z.B. Konstanten, Variablen, Klassen, Methoden – genannt werden sollen.



- Eine durchdachte und konsistente Namensgebung wird nur durch die konsequente Einhaltung von geeigneten Richtlinien erreicht
- Folgende Richtlinien gelten für alle INFORMATIK-I-Java-Programme (bitte bei den im Rahmen der Übungen zu erstellenden Programmen beachten):
 - Namenslänge
 - kurz im Falle von lokale, temporär verwendeten Namen (z.B. Laufvariablen) i, j
 - eher etwas länger bei wichtigen Semantik-tragenden Namen course, catalog
 - Worttrennung
 - Großbuchstaben zur Trennung von Wörtern in einem Namen _____
 - Sprache
 - englisch sowohl für Namen als auch für Kommentare // ordered by
// course number

Interaktion 5: Richtlinien

Für alle im Rahmen dieser Veranstaltung entwickelten Java-Programme gelten die im Folgenden ausgeführten Richtlinien.

Namenslänge

Mit der Namenslänge ist ganz wesentlich die Verständlichkeit eines Namens verbunden. Grundsätzlich gilt, dass ein "Ding" durch einen längeren Namen verständlicher ausgedrückt werden kann. Zu lange Namen führen aber zu langen und schlecht lesbaren Programmen, weshalb kurze Namen zu bevorzugen sind. Grundsätzlich ist zu unterscheiden, um welche Art von Namen es sich handelt:

- Namen, die in einem kleineren, abgegrenzten Bereich des Programms genutzt werden, sollten kurz sein.
- Namen, die ein bedeutungstragendes Element der Problemlösung darstellen, sollten sprechend, aber nicht zu lang sein.

Worttrennung

Bedeutungstragende Namen setzen sich häufig aus mehr als einem Wort zusammen. Zur Trennung der Wörter bieten sich zwei Varianten an:

1. Trennung durch Großbuchstaben
2. Trennung durch Unterstreichungszeichen

Für die im Rahmen dieser Veranstaltung zu erstellenden Programme kommt ausschließlich die erste Variante zur Anwendung, d.h. Trennung durch Groß-/Kleinschreibung (siehe Interaktion 5).

Sprache

Aus folgenden Gründen sind englische Namen zu wählen:

- Englische Namen passen besser zu den ebenfalls englischen Schlüsselwörtern.
- Der mögliche Verbreitungsgrad der Programme ist größer.

Es gehört zum guten Programmierstil, zu allen nicht unmittelbar verständlichen Variablen durch einen Kommentar zusätzliche Erläuterungen in Form eines Kommentars anzubringen. Die

Syntax von Kommentaren wird später anhand konkreter Beispiele eingeführt. Kommentare werden in allen Programmen, die im Rahmen der Vorlesung und Übungen in INFORMATIK I erstellt werden, aus den gleichen Gründen, die bereits bei den Namen genannt wurden, in englischer Sprache formuliert.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Schlüsselwörter
 - bezeichnen die Sprachelemente und dürfen daher nicht als Namen verwendet werden
 - konsequente Kleinschreibung
 - Beispiele: if, for, boolean, int, static
- Zahlen
 - ganze Zahlen: dezimal, z.B. 10, hexadezimal, z.B. 0xC3A
 - Gleitkommazahlen: z.B. 4.7E11
- Zeichen
 - Buchstaben, Ziffern und Sonderzeichen, die in Unicode codiert sind
 - in einfache Hochkommata gesetzt
- Zeichenketten
 - in Hochkommata gesetzte Zeichen, z.B. "Informatik I"
 - " wird als \" geschrieben, z.B. "\"Informatik I\""

Information 8: Erste Sprachelemente

Durch die in Information 8 beschriebenen Schlüsselwörter einer Sprache wird die grundlegende Syntax einer Sprache angegeben. Die Sprache Java besteht aus den folgenden 48 Schlüsselwörtern:

1. abstract	11. boolean	21. break	31. byte	40. case
2. catch	12. char	22. class	32. const	41. continue
3. default	13. do	23. double	33. else	42. extends
4. final	14. finally	24. float	34. for	43. goto
5. if	15. implements	25. import	35. instanceof	44. int
6. interface	16. long	26. native	36. new	45. null
7. package	17. private	27. protected	37. public	46. return
8. short	18. static	28. super	38. switch	47. synchronized
9. this	19. throw	29. throws	39. transient	48. try
10. void	20. volatile	30. while		

Es fällt unmittelbar auf, dass alle Schlüsselwörter konsequent klein geschrieben sind. Da Java zwischen Klein- und Großschreibung unterscheidet, werden z.B. Int oder instanceof vom Übersetzer nicht als Schlüsselwörter akzeptiert und führen zu einem Syntaxfehler.

Eine weitere Art von Grundsymbolen sind die Zahlen, wobei in Java ganze Zahlen und Gleitkommazahlen unterschieden werden. Ganze Zahlen können als (gewöhnliche) Dezimalzahlen oder als (der Bitdarstellung nahe liegenden) Hexadezimalzahlen geschrieben werden.

Zeichen und Zeichenketten sind in unterschiedliche Arten von Hochkommata (einfach bzw. doppelt) gesetzt.

Zeichen sind intern durch Zahlen codiert, wobei der den ASCII-Code (*American Standard Code of Information Interchange*) [C&M-GI] erweiternde Unicode verwendet wird.

In Zeichenketten lassen sich gewisse Zeichen als Zeichenkombinationen (so genannte *Escape-Sequenz*) bestehend aus einem Schrägstrich (*Backslash* \, so genanntes *Escape-Zeichen*) und einem sich anschließenden Zeichen angeben.

2.2 Variablen und Zuweisung

Eine Variable stellt in Java ein grundlegendes Datenelement dar, das vor dessen Nutzung zunächst zu deklarieren, d.h. bekannt zu machen ist.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Variablen
 - sind Behälter (Container) von Werten
 - haben einen Namen
 - haben einen bestimmten Typ bzw. Datentyp, der die mögliche Wertemenge festlegt
 - müssen vor ihrer Verwendung deklariert (bekannt gemacht) werden
- Beispiel:

```
int max = 1000; // maximal number of something
```

 - Deklaration einer Variablen mit Namen max
 - max hat den Typ int (integer, d.h. ganze Zahl)
 - Compiler reserviert einen entsprechend großen Speicherplatz
 - max kann bei der Deklaration ein Wert (im Beispiel 1000) zugewiesen werden, der im Speicherplatz abgelegt wird
 - Kommentar (//) gibt zusätzliche Erläuterungen zur Variablen

Information 9: Variablen und deren Deklaration

Information 9 zeigt ein Beispiel eines Java-Programmausschnitts, durch den eine int-Variable mit dem Namen max deklariert wird.

Neben der Variablendeklaration bietet Java die Möglichkeit, Konstanten zu definieren. Die Konstanten kann man sich wie Variablen vorstellen, die bei ihrer Deklaration initialisiert werden und anschließend ihren Wert nicht mehr ändern.

- Es bestehen folgende Standardtypen für ganze Zahlen in Java:
 - byte 8-Bit-Zahl $-2^7 \dots 2^7-1$ (-128 ... 127)
 - short 16-Bit-Zahl $-2^{15} \dots 2^{15}-1$ (-32768 ... 32767)
 - int 32-Bit-Zahl $-2^{31} \dots 2^{31}-1$ (-2.147.483.648 ... 2.147.483.647)
 - long 64-Bit-Zahl $-2^{63} \dots 2^{63}-1$
- Die Längen der einzelnen Typen gelten plattformunabhängig
- Ganze Zahlen sind in Java grundsätzlich vorzeichenbehaftet
- Der Datentyp bestimmt
 - die Menge von Werten, die zu diesem Typ gehören
 - die Menge von Operationen, die mit den Werten des Typs ausgeführt werden dürfen

Information 10: Standardtypen für ganze Zahlen und Datentyp

Wie die in Information 10 enthaltene Auflistung der Standardtypen von den in Java angebotenen ganzen Zahlen aufzeigt, reserviert der Compiler für eine int-Zahl grundsätzlich – d.h. auf jedem Rechner mit beliebigem Betriebssystem – 32 Bit Speicher. Hierdurch wird bestimmt, welche Zahlenwerte in der int-Variablen aufgenommen werden können. Je nach benötigter Wertemenge können statt int die Standardtypen byte oder short (bei kleineren Wertebereichen) und long (im Falle eines größeren Wertebereichs) als Typ vereinbart werden.

Die zu den ganzen Zahlen kennen gelernten Typen sind Beispiele für Datentypen. Datentypen gehören zu den fundamentalen Konzepten einer modernen Programmiersprache. Dieses Konzept ermöglicht dem Compiler eine statische Typprüfung, in der festgestellt wird, ob eine Variable einen zulässigen Wert enthält und ob die auf einer Variablen ausgeführte Operation aufgrund des deklarierten Datentyps erlaubt ist. Zahlreiche Programmierfehler lassen sich auf diese Weise aufdecken.

- Eine Zuweisung ist eine Anweisung, durch die einer (deklarierten) Variablen der Wert eines Ausdrucks zugewiesen wird

- Beispiel:

<code>int x = max - 1;</code>

<code>int x;</code>
<code>x = max - 1;</code>

 - int x Deklaration der Variablen x
 - '=' Zuweisungszeichen
 - max - 1 Wert, der x zugewiesen wird

- Die in den Kommentaren beschriebenen Zuweisungen sind zu ergänzen

```

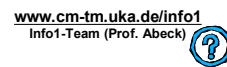
_____ // assign value of x to long variable l
_____ // assign 250 to short variable s
_____ // assign value of s minus value of x to l
  
```

Interaktion 6: Zuweisung

Ein grundlegender Befehl im Zusammenhang mit Variablen ist die Zuweisung (*Assignment*). Anstelle von Befehl wird auch der Begriff der Anweisung (*Statement*) verwendet. Wie die beiden Programmausschnitte in Interaktion 6 zeigen, kann einer Variablen bei ihrer Deklaration direkt ein Wert zugewiesen werden. Alternativ können die Deklaration und die Zuweisung in zwei getrennten Befehlen erfolgen.

Als Zuweisungsoperator wird in Java das Gleichheitszeichen '=' verwendet. Hierbei ist zu beachten, dass die Zuweisungsbeziehung keine Gleichheitsbeziehung darstellt.

Am Beispiel der in den Kommentaren in Interaktion 6 angegebenen informellen Zuweisungsbeschreibungen soll diese Anweisung praktisch geübt werden.



- Die Forderung der Zuweisungskompatibilität ist erfüllt, wenn
 - linke und rechte Seite denselben Typ haben oder
 - der Typ der linken Seite schließt den Typ der rechten Seite ein
 - es gilt folgende Typhierarchie:

long \supset int \supset short \supset byte

	richtig	falsch
int i, j;		
short s;		
byte b;		
i = j;	<input type="checkbox"/>	<input type="checkbox"/>
i = s;	<input type="checkbox"/>	<input type="checkbox"/>
s = i;	<input type="checkbox"/>	<input type="checkbox"/>
s = b;	<input type="checkbox"/>	<input type="checkbox"/>
i = 300;	<input type="checkbox"/>	<input type="checkbox"/>
s = 300;	<input type="checkbox"/>	<input type="checkbox"/>
b = 300;	<input type="checkbox"/>	<input type="checkbox"/>

Die Einhaltung der Zuweisungskompatibilität ist für die folgenden Beispiele zu überprüfen

Interaktion 7: Zuweisungskompatibilität von Variablen ganzzahligen Typs

Nicht jeder Variablen kann ein beliebiger Wert zugewiesen werden, weshalb eine so genannte Zuweisungskompatibilität erforderlich ist. Die Kompatibilität ist dann gegeben, wenn der Typ der rechten Seite den Typ der linken Seite einschließt, d.h. die Wertemenge, die der Typ der rechten Seite beschreibt, umfasst die Wertemenge, die durch den Typ der linken Seite beschrieben ist.

Aufgrund der durch die verschiedenen ganzzahligen Typen beschriebenen Wertemengen (siehe Information 10 weiter oben) ergibt sich die in Interaktion 7 angegebene Typhierarchie. Anhand der Beispiele ist zu überprüfen, ob die Zuweisungskompatibilität eingehalten oder verletzt wird.

Zahlkonstanten werden, wie im obigen Beispiel die Zahl 300, in Java (unabhängig von deren Wert) immer als int angenommen. Allerdings akzeptiert der Compiler eine Zuweisung einer Zahl zu einer short- bzw. byte-Variablen, falls der Wert von dieser Variablen aufgenommen werden kann.

2.3 Ausdrücke

Mit Variablen und Konstanten lässt sich mittels der Operatoren, die zu dem jeweiligen Typ definiert sind, rechnen. Im Folgenden werden für ganzzahlige und boolesche Typen die in Ausdrücken formulierbaren Berechnungsformeln vorgestellt.

2.3.1 Arithmetische Ausdrücke

Mit dem Beispiel $y + 1$ wurde bereits ein erstes einfaches Beispiel eines arithmetischen Ausdrucks eingeführt. Information 11 zeigt ein Beispiel eines weiteren, komplexeren arithmetischen Ausdrucks.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Berechnung eines numerischen Wertes aus Variablen und Konstanten mittels arithmetischer Operatoren
 - Beispiel: $-x / (y - 1)$
- Syntax in Form von EBNF-Regeln

Expression = Operand {BinaryOperator Operand}.
Operand = [UnaryOperator] (Variable | Number | "(" Expression ")").

- Standard-Operatoren
 - binär: + - * / %
 - unär: + -
- Spezielle Operatoren in Java
 - Shift: $x \ll y$ $x \gg y$
 - Inkrement: $x++$ $++x$
 - Dekrement: $x--$ $--x$
 - Zuweisungsoperator: $x += y$ $x -= y$ $x *= y$ $x /= y$ $x \% = y$

Information 11: Arithmetischer Ausdruck

Die Syntax eines arithmetischen Ausdrucks lässt sich durch die zwei angegebenen EBNF-Regeln einfach festlegen.

Neben den bekannten binären (2-stelligen) und unären (1-stelligen) Standardoperatoren, für die die bekannten Vorrangregeln gelten, bietet Java eine Reihe von Spezialoperatoren an. Durch die *Shift*-Operationen lassen sich im Falle von zeitkritischen Anwendungen schnelle Multiplikationen und Divisionen mit Zweierpotenzen durchführen.

Inkrement ($x++$ entspricht $x = x+1$) bzw. Dekrement ($x--$ entspricht $x = x-1$) und der Zuweisungsoperator ($x += y$ entspricht $x = x + y$) sind Kurzschreibweisen, die von Java angeboten werden. Inkrement bzw. Dekrement sollten aufgrund der komplizierten Semantik, die bei deren Verwendung in einem arithmetischen Ausdruck entstehen, möglichst nur als eigenständige Anweisung auftreten.

```
// Example: result type and type cast
short s;
int i;
long x;
... x + 1 ...           // long
... s + 1 ...           // int (1 is of type int)
s = (short)(s + 1)      // type cast required
```

- Zu beachten: Zahlenkonstanten (im Beispiel: 1) sind vom Typ int
 - Operation mit short-Variablen liefert eine int-Variable als Ergebnis
- Durch eine Typkonversion wird der Typ eines Ausdrucks in den gewünschten Typ umgewandelt
 - im Beispiel: es werden die ersten beiden Bytes des int-Werts abgeschnitten

Information 12: Ergebnistyp eines arithmetischen Ausdrucks und Typkonversion

Im Zusammenhang mit Ausdrücken stellt sich die Frage, von welchem Typ das Ergebnis eines ganzzahligen Ausdrucks ist. Hierzu besteht die folgende Festlegung: Wenn mindestens ein Operand vom Typ long ist, so ist der Ergebnistyp long, sonst ist der Ergebnistyp int.

Das bedeutet, dass eine Zuweisung des Ergebnisses eines arithmetischen Ausdrucks an eine Variable mit dem Typ short oder byte immer eine so genannte, im Programm in Information 12 enthaltene Typkonversion erforderlich macht.

2.3.2 Boolesche Ausdrücke

Der Datentyp boolean ist ein weiterer elementarer Datentyp, der insbesondere für die im folgenden Abschnitt behandelte Ablaufsteuerung in einem Programm benötigt wird.



- Typ boolean ist neben den bereits kennen gelernten ganzzahligen Typen (long, int, short, byte) ein weiterer wichtiger elementarer Datentyp
 - kann die Werte true oder false annehmen
 - Operatoren: && || !
- Ein boolescher Wert kann aus einem Vergleich von Zahlenwerten resultieren
 - Operatoren: == != > < >= <=

```
// Example: data type boolean
int x = 1;
boolean p, q;
p = false;
q = 0 < x;           // true or false? _____
p = (p || q) && x < 10; // true or false? _____
```

Interaktion 8: Datentyp boolean

Der Einsatz der booleschen Operatoren (und &&, oder ||, nicht !) sowie der Vergleichsoperatoren, die einen booleschen Wert als Ergebnis liefern, lässt sich anhand des Programmausschnitts in Interaktion 8 nachvollziehen.

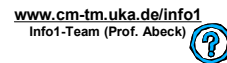
Die den Ausdrücken zugrunde liegenden algebraischen Grundlagen werden in einer späteren Kurseinheit behandelt.

2.4 Anweisungen zur Ablaufsteuerung

Die Steuerung des Kontrollflusses gehört zu den wesentlichen Bestandteilen eines Algorithmus. Mit der bedingten Anweisung und der Schleife werden zwei grundlegende Anweisungen eingeführt und am Beispiel von Java anhand der if-Anweisung und der while-Anweisung präzisiert.

2.4.1 Bedingte Anweisung

In Abhängigkeit einer als boolescher Ausdruck formulierten Bedingung wird eine von zwei alternativen Anweisungen ausgeführt.



- Ziel: Eine Anweisung soll nur unter bestimmten Bedingungen ausgeführt werden

- Syntax (Erweiterte Backus-Naur-Form EBNF)

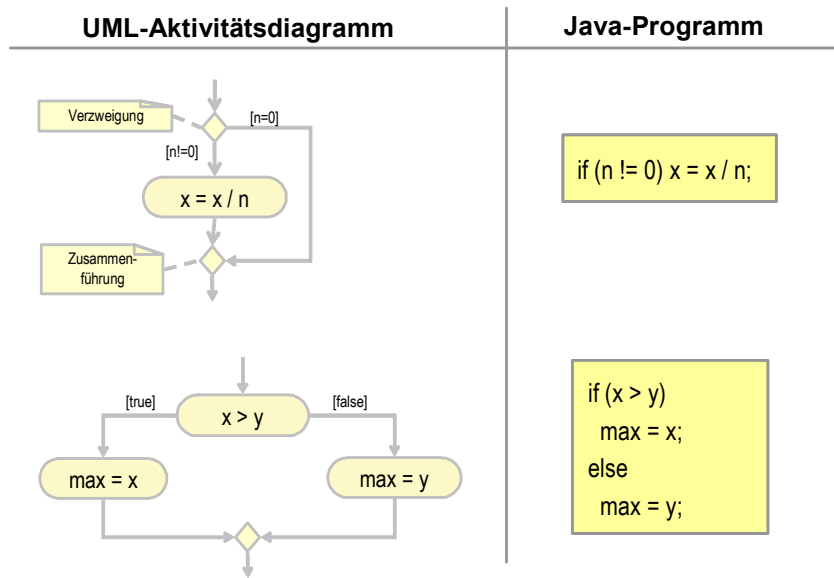
```
ifStatement = "if" "(" Expr ")" Statement ["else" Statement].
```

- Nichtterminale: _____
- Terminale: _____
- Wie heißt die dritte Art von EBNF-Zeichen? _____
- Das Nichtterminal Expr ist ein Ausdruck (Expression), dessen Auswertungsergebnis als boolescher Wert wahr (true) oder falsch (false) interpretiert wird

Interaktion 9: Bedingte Anweisung

In Interaktion 9 ist die Syntax einer bedingten Anweisung in einer EBNF-Regel angegeben.

Bedingte Anweisungen werden (wie bedingte Ausdrücke) in ihrer Bedeutung auf die Fallunterscheidung zurückgeführt. Hat der nach der if-Anweisung folgende Ausdruck (expr) den Wert true, so wird die Anweisung im then-Zweig (der Anweisungsblock direkt nach der Bedingung) ausgeführt, andernfalls der else-Zweig.



Information 13: Beispiele

In der *Unified Modeling Language* (UML) lassen sich Bedingungen in einem Aktivitätsdiagramm darstellen.

Wie die beiden Beispiele in Information 13 zeigen, stehen zwei Möglichkeiten zur Verfügung: Im ersten Fall wird sowohl die Zusammenführung als auch die Verzweigung des Kontrollflusses explizit durch eine Raute (*Diamond*) angegeben, im zweiten Fall werden die Bedingungen direkt an die ausgehenden Transitionen der Aktivität geschrieben, die den booleschen Ausdruck berechnet. Die Beispiele verdeutlichen, dass die zweite Variante besser lesbar ist und eine einfachere Umsetzung in ein Java-Programm ermöglicht.

- Die Syntax von Java erlaubt nur eine Anweisung im then-Zweig (true-Zweig) und im else-Fall (false-Zweig)
 - gibt die Syntax der Programmiersprache vor
 - stellt aber keine grundsätzliche Einschränkung dar, weil
 - Anweisungsfolgen lassen sich durch Setzen in geschweifte Klammern {...} zu einem Block zusammenfassen
 - ein Block wird als eine Anweisung betrachtet

```
if (x < 0) {
    negNumbers++;
    Out.print(-x);
} else {
    posNumbers++;
    Out.print(x);
}
```

Was leistet der Programmausschnitt?

Interaktion 10: Anweisungsblöcke

Sollen im then- oder else-Zweig mehrere Anweisungen ausgeführt werden, so sind diese aufgrund der Syntax-Vorschrift zu jeweils einem Anweisungsblock zusammen zu führen.

Interaktion 10 zeigt ein Beispiel eines Programmausschnitts, dessen Ergebnis informell zu beschreiben ist.

An den Beispielen wird deutlich, dass die Lesbarkeit eines Java-Programms durch entsprechende Einrückungen erhöht wird. Als Einrückung wurden zwei Leerzeichen vorgesehen. Wie das erste Beispiel aus Information 13 zeigte, werden kurze if-Anweisungen auch in einer Zeile geschrieben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

<pre>// Example 1 if (a > b) if (a > 0) max = a; else max = b;</pre>	<pre>// Example 2 if (a > b) { if (a > 0) max = a; else max = b; }</pre>	<pre>// Example 3 if (a > b) { if (a > 0) max = a; } else max = b;</pre>
--	--	--

- Problem: Zuordnung des else-Zweigs bei mehreren vorausgehenden verschachtelten if-Anweisungen
- Lösung: Der else-Zweig gehört immer zur unmittelbar vorausgehenden if-Anweisung
 - Klammern in Beispiel 2 damit überflüssig
- Zuordnung des else-Zweigs zu einer anderen if-Anweisung erfolgt über Zusammenführung von Anweisungsfolgen zu einem Block
 - siehe Beispiel 3

Information 14: Hängender else-Zweig

Mehrere if-Anweisungen können verschachtelt werden, d.h. es kann in einem then-Zweig oder else-Zweig eine weitere if-Anweisung erfolgen. Eine if-Anweisung im then-Zweig kann zu einem so genannten *dangling else* – also einem "in der Luft" hängenden else-Zweig führen. Die Problematik und deren Lösung in Java verdeutlicht Information 14.

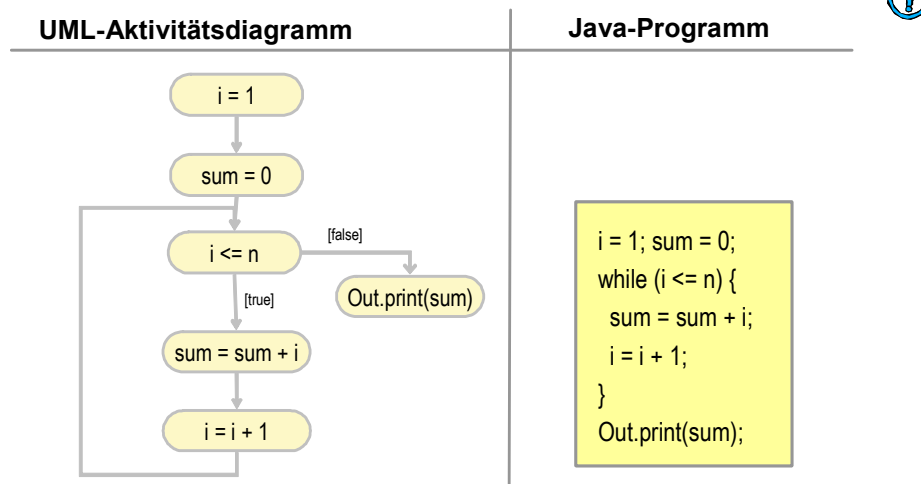
2.4.2 Schleife

Die Schleife ist ein Sprachelement, durch das eine mehrmalige Ausführung von Programmteilen ermöglicht wird.

- Ziel: Gewisse Berechnungen sollen wiederholt ausgeführt werden, bis eine bestimmte Bedingung eintritt
- Dieses Ziel wird u.a. durch die while-Schleife (Abweisschleife) erreicht
- Syntax der while-Schleife
 - while (<Schleifenbedingung>) <Schleifenrumpf>
 - als EBNF-Regel
WhileStatement = "while" "(" Expr ")" Statement.
 - durch die Bildung eines Blocks können beliebig viele Anweisungen den Schleifenrumpf bilden

Information 15: Schleifen

Es lassen sich im Wesentlichen drei Arten von Schleifen unterscheiden: Abweisschleife, Durchlaufschleife, Zählschleife. In dieser Kurseinheit soll zunächst die Abweisschleife behandelt werden (siehe Information 15), die sich durch die while-Anweisung ausdrücken lässt.



- Was wird durch den Programmausschnitt berechnet?

Interaktion 11: Beispiel zur while-Anweisung

Ein Beispiel eines Programmausschnitts mit einer while-Schleife zeigt Interaktion 11.

Beim Programmieren einer Schleife ist sicher zu stellen, dass die Schleifenbedingung auch tatsächlich nach endlich vielen Durchläufen durch den Schleifenrumpf nicht mehr erfüllt wird und die Schleife verlassen wird. Andernfalls würde der Rechner endlos die Anweisungen in der Schleife ausführen und das Programm würde nicht terminieren.

Auf das Problem der Schleifenterminierung sowie die oben erwähnten weiteren Schleifenarten wird in der Kurseinheit IMPERATIVE PROGRAMMIERUNG [C&M-IP] detailliert eingegangen.

3 PROGRAMMSTRUKTUR

Die bislang behandelten Java-Beispiele stellen nur jeweils Ausschnitte aus einem Java-Programm dar. In diesem Abschnitt sollen nun alle noch fehlenden Aspekte zusammengetragen werden, damit vollständige, auf dem Rechner ausführbare Java-Programme erstellt werden können [Mö03].

3.1 Grundstruktur

Jedes Java-Programm folgt einer ganz bestimmten Grundstruktur, die vom Compiler verlangt wird, um ein Programm übersetzen zu können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class ProgramName {

    public static void main (String[] arg) {
        ... Declarations ...
        ... Statements ...
    } // main

} // class
```

- Die Grundstruktur ist vorläufig als ein einfacher syntaktischer Rahmen zu verstehen
 - kursiv geschriebene Teile sind vom Programmierer zu ersetzen
 - zunächst gibt es nur eine Klasse (class *ProgramName*), die das gesamte Programm beinhaltet
 - main() ist eine Methode der Klasse *ProgramName*
- Namens-Schreibweise
 - Klassennamen beginnen mit einem Großbuchstaben
 - Methodennamen beginnen mit einem Kleinbuchstaben

Information 16: PROGRAMMSTRUKTUR – Grundstruktur eines Java-Programms

Diese Grundstruktur zeigt Information 16. Das gesamte Programm besteht aus einer Klasse `class`, die den Programmnamen trägt. Das Klassen-Konzept von Java wird erst in einer späteren Kurseinheit eingeführt.

Zum Klassenkonzept muss zunächst nur bekannt sein, dass in Klassen so genannte Methoden auftreten. Methoden sind Rechenvorschriften, denen Parameter übergeben werden können und die einen Rückgabewert abliefern können. Die Methode `main()` ist eine in Java ausgezeichnete Methode, da diese den Einstiegspunkt in das Programm markiert.

```

class FirstProgram {
    public static void main (String[] arg) {
        int v1, v2, max;           // declare variables
        v1 = 17; v2= -9;          // assign variables
        max = v1;                 // determine maximum
        if (max < v2) max = v2;
    }
}
  
```

- Zu klären
 - Ausgabe des Ergebnisses bzw. Eingabe von Werten
 - Ausführung des Programms auf dem Rechner

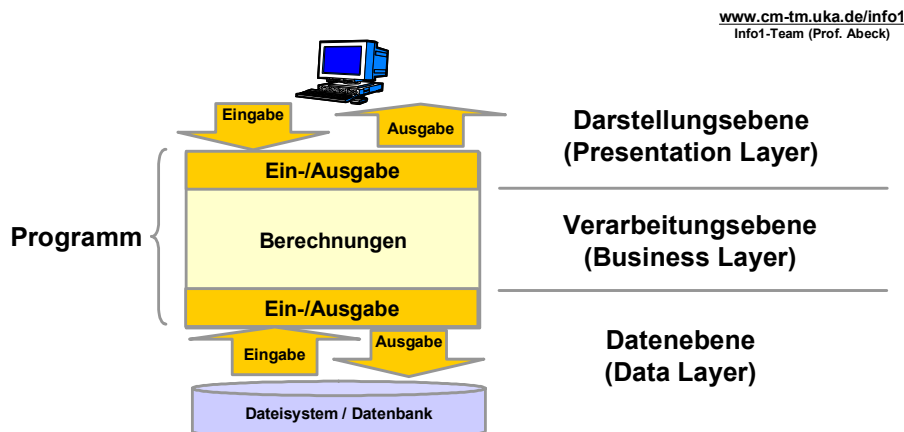
Information 17: Erstes vollständiges Java-Programm

Information 17 führt ein erstes vollständiges Java-Programm ein, durch das das Maximum max von zwei gegebenen Variablenwerten v1 und v2 ermittelt wird.

Die in Information 17 genannten Punkte, die zwingend zu klären sind, betreffen die nachfolgend behandelte Ein-/Ausgabe..

3.2 Ein-/Ausgabe und Programmausführung

Ein wichtiger technischer Aspekt, der bislang noch nicht im Detail betrachtet wurde, betrifft die Frage, wie einem Programm Daten zugeführt (Eingabe) bzw. aus dem Programm Daten bereitgestellt (Ausgabe) werden können.



- Ein Programm erhält die zu verarbeitenden Daten von seiner Umgebung und gibt Daten an die Umgebung ab
 - die Umgebung besteht aus einer persistenten Datenhaltung und der Schnittstelle zum Benutzer
 - das Ergebnis ist eine Drei-Schichten-Architektur
 - Daten – Verarbeitung – Darstellung

Information 18: Ein-/Ausgabe

Wie die Abbildung in Information 18 zeigt, besteht die Umgebung eines Programms, mit der Daten ausgetauscht werden, zum einen aus der Benutzerschnittstelle mit der Tastatur als Eingabegerät und dem Bildschirm als Ausgabegerät. Zum anderen kann das Programm mit dem

Dateisystem (oder auch mit einem Datenbanksystem) Daten austauschen, indem es auf die Datei (bzw. Datenbank) lesend oder schreibend zugreift.

Mit den beiden Ein-/Ausgabeformen sind zwei Ebenen, die Daten- und die Darstellungsebene, verbunden. Die durch das Programm realisierte Funktionalität stellt die eigentliche Verarbeitungsebene dar. Insgesamt liefern die drei Ebenen eine Architektur, die auf beliebige Software-Systeme angewendet werden kann.

3.2.1 Klassen In und Out

Nachfolgend wird zunächst aufgezeigt, wie Ausgaben zur Benutzerschnittstelle (Darstellungsebene) realisiert werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
if (x < 0) {
    negNumbers++;
    Out.print(-x)
} else {
    posNumbers++;
    Out.print(x);
}
```

- Out.print() ist ein Befehl zur Ausgabe von Daten (hier -x bzw. x)
 - Out Klassenname
 - print() Methodenname
 - -x bzw. x Parameter

Information 19: Methode Out.print()

In einem früheren Beispiel kam bereits ein Java-Befehl
Out.print(x)

zum Einsatz. Dieser Befehl bewirkt, dass der Wert der Variablen x ausgegeben wird. Ein Blick auf die Liste der Schlüsselwörter der Sprache Java macht deutlich, dass weder Out noch print noch die Kombination auftreten.

Wie Information 19 beschreibt, handelt es sich hierbei um eine Java-Klasse Out und die zu dieser Klasse gehörende Methode print() mit dem Aufrufparameter -x bzw. x. Die Klasse stammt aus [Mö03] und wird in den folgenden Programmen zur Ausgabe von Daten genutzt.



```

class FirstOutProgram {
    public static void main (String[] arg) {
        int v1, v2, max;           // declare variables
        v1 = 17; v2 = -9;         // assign variables
        max = v1;                 // determine maximum
        if (max < v2) max = v2;
        Out.print("The maximum of v1 = " + v1 + " and v2 = " + v2 + " is " + max + ".\n");
    }
}

```

- Der an die Methode print() übergebene Parameter setzt sich aus mehreren mittels des Additionssymbols verknüpften Zeichenketten zusammen
 - Zeichenketten (z.B. "The maximum of v1 = ")
 - Integer-Werte (v1, v2, max)
- [Ausgabe des Programms](#)

Interaktion 12: Ergänzung des Beispiel-Programms um die Ergebnis-Ausgabe

Interaktion 12 zeigt, wie das Beispiel-Programm unter Nutzung der Klasse Out zu ergänzen ist, damit das Ergebnis der Berechnung – das Maximum der beiden Variablenwerte – auf dem Bildschirm ausgegeben wird.

```

class FirstInOutProgram {
    public static void main (String[] arg) {
        int v1, v2, max;           // declare variables

        Out.print("\n enter v1: "); v1 = In.readInt();    // read v1 from console
        Out.print("\n enter v2: "); v2 = In.readInt();    // read v2 from console

        max = v1;                 // determine maximum
        if (max < v2) max = v2;
        Out.println("The maximum of v1 = " + v1 + " and v2 = " + v2 + " is " + max);
    }
}

```

- Neben der Klasse Out zur Ausgabe wird eine Klasse In zur Eingabe über Tastatur oder Datei bereitgestellt
- Methode readInt() liest ganze Zahlen über die Tastatur ein

Information 20: Ergänzung des Beispielprogramms um die Eingabe von Werten

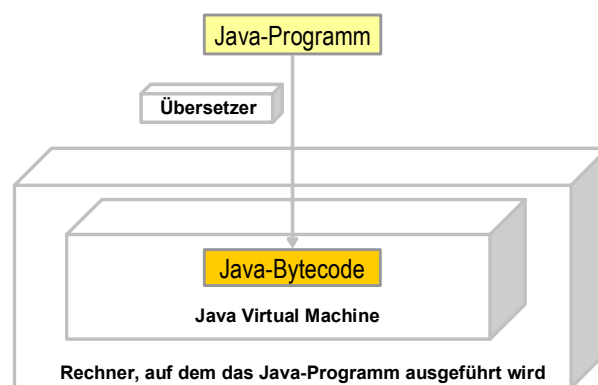
In Information 20 wird das Programm FirstOutProgram zu einem Programm FirstInOutProgram weiterentwickeln, indem die beiden Zahlenwerte v1 und v2 nicht durch eine Zuweisung im Programm sondern durch eine Eingabe über Tastatur bestimmt werden. Hierzu wird die Methode readInt() genutzt, die Bestandteil der Klasse In [Mö03] ist.

3.2.2 Programmübersetzung und -ausführung

Das zuletzt entwickelte Programm (FirstInOutProgram) ist unter der Voraussetzung, dass die Ein- und Ausgabeklassen zur Verfügung stehen, vollständig und soll im Folgenden auf dem Rechner ausgeführt werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Um ein in Java geschriebenes Programm auf einem Rechner ausführen zu können, wird ein spezielles Programm benötigt
 - Übersetzerprogramm, das Java in Java-Bytecode übersetzt
 - der Bytecode wird von einer auf dem Rechner laufenden Java Virtual Machine (JVM) ausgeführt



Information 21: Übersetzung und Ausführung des Java-Programms

Wie die Abbildung in Information 21 zeigt, erzeugt der Übersetzer, der selbst ein umfangreiches ablauffähiges Programm darstellt, aus dem Java-Programm einen so genannten Java-Bytecode. Der Java-Bytecode ist eine maschinennahe Sprache, die von der *Java Virtual Machine* (JVM) ausgeführt wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- (0.1) Installieren des Java Development Kit (JDK) mit dem Java-Übersetzer und der Java Virtual Machine
- (0.2) Installieren der Klassen In und Out
- (1) Erstellung einer Datei
FirstInOutProgram.java
mit dem Java-Quelltext
- (2) Übersetzen des Programms mittels
javac FirstInOutProgram.java
- (3) Ausführen des Programms mittels
java FirstInOutProgram

Information 22: Konkrete Arbeitsschritte am Beispiel von FirstInOutProgram

Der Java-Übersetzer und die *Java Virtual Machine* sind zunächst auf dem Rechner, auf dem die Java-Programme ausgeführt werden sollen, zu installieren. Diese beiden Programme sind Bestandteil des kostenfrei nutzbaren *Java Development Kit* (JDK). Das JDK sowie die beiden weiter oben eingeführten Klassen `In` und `Out` sind auf dem Rechner zu installieren, wobei die jeweiligen Installationshinweise zu beachten sind.

Der erste Schritt der eigentlichen Programmierung besteht dann in der Erstellung des Java-Programms. Hierzu ist auf dem Rechner ein Editor-Programm zu nutzen, mit dem ASCII-Textdateien bearbeitet werden können. Der Name dieser Datei muss mit dem Namen des Programmnamens übereinstimmen, die Endung (*Extension*) muss `.java` lauten.

Die Übersetzung des in Quelltext vorliegenden Java-Programms in den entsprechenden Bytecode erfolgt durch Aufruf des Befehls `javac` und der Angabe des Dateinamens und der Endung `.java`.

Falls der Übersetzungsvorgang erfolgreich abgeschlossen werden konnte, liegt der Bytecode in einer Datei mit dem gleichen Namen wie die Textdatei sowie der Endung `.class` vor.

Die Ausführung dieses Bytecodes durch die *Java Virtual Machine* erfolgt dann durch den Befehl `java` und der Angabe des Datei- bzw. Programmnamens ohne Endung.

4 METHODEN

Mit den Methoden wird in diesem Kapitel ein wichtiges Programmierkonzept zur Aufteilung von Anweisungsfolgen eines Programms behandelt [Mö03].

Der Begriff der Methode resultiert aus dem übergeordneten Klassenkonzept, das in der Kurseinheit OBJEKTORIENTIERTE PROGRAMMIERUNG [C&M-OP] behandelt wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Methoden sind benannte Anweisungsfolgen
 - dienen zur Zerlegung und Modularisierung eines Programms
 - können parametrisiert oder auch parameterlos sein
 - können einen oder keinen Wert zurückliefern
 - werden entsprechend als Funktion oder Prozedur bezeichnet
- Es wurden bereits Methoden verwendet
 - Deklaration der `main`-Methode: `public static void main(String[] arg)`
 - Schlüsselwörter "public" und "static" werden später im Zusammenhang mit dem Klassenbegriff geklärt
 - Schlüsselwort "void" besagt, dass die Methode keinen Wert zurückliefert
 - `String[] arg` ist ein Parameter mit dem Namen `arg` vom Typ `String[]`
 - Aufruf der von den Klassen `In` und `Out` angebotenen Methoden, z.B.
 - `int i = In.readInt();`
 - `Out.println("Text");`

Information 23: METHODEN - Einführung

Methoden traten bereits an verschiedenen Stellen auf (siehe Information 23). So kann kein Java-Programm auf eine Deklaration der `main()`-Methode verzichten. Es handelt sich hierbei um eine Methode mit einem Parameter und ohne Rückgabewert.

Die angegebenen Methoden `readInt()` und `println()` der Klassen `In` und `Out` sind zwei weitere Beispiele für unterschiedliche Formen von Methoden.

4.1 Deklaration und Aufrufketten

Information 24 zeigt ein Java-Programm mit einer neben der obligatorischen `main()`-Methodendeklaration weiteren Methodendeklaration `printHeader()`.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

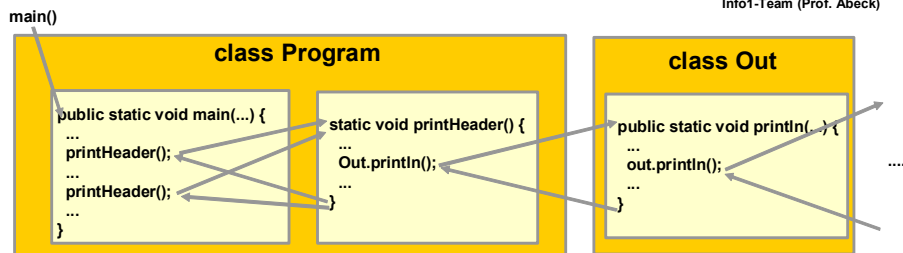
```
class Program {
    static void printHeader() {           // method declaration: printHeader()
        Out.println("Teilnehmerliste"); // method call: println()
        Out.println("-----");        // method call: println()
    }
    public static void main (String[] arg) { // method declaration: main()
        printHeader();                   // method call: printHeader()
        // names of participants
        printHeader();                   // method call: printHeader()
        // ....
    }
}
```

- Die Methode `printHeader()`
 - ist wie `main()` eine Methode zur Klasse `Program`
 - wird in `main()` aufgerufen
 - ruft die Methode `println()` der Klasse `Out` auf

Information 24: Methodendeklaration und Methodenaufruf

Die Methode `printHeader()` gehört zur Klasse `Program` und befindet sich auf der gleichen Ebene wie die `main()`-Methode.

`printHeader()` wird von `main()` zweimal aufgerufen und ruft seinerseits eine andere Methode – `println()` aus der Klasse `Out` – ebenfalls zweimal auf.



- Namenskonventionen zu Methodennamen
 - beginnen mit einem Kleinbuchstaben
 - beginnen mit einem Verb
 - falls aus mehreren Wörtern gebildet, beginnen die Folgewörter mit einem Großbuchstaben
- Namenskonventionen zu Klassennamen
 - beginnen mit einem Großbuchstaben
 - falls aus mehreren Wörtern gebildet, beginnen die Folgewörter mit einem Großbuchstaben

Information 25: Methodenaufkette und Namenskonventionen

Es entsteht eine Methodenaufkette, die graphisch in der Abbildung in Information 25 verdeutlicht wird. Die Kette beginnt mit dem Aufruf der Methode `main()`. Dieser Aufruf erfolgt automatisch beim Starten des Programms. Wie in der Abbildung angedeutet wird, ist die Kette nicht vollständig, da in der Methode `println()` der Klasse `Out` ein weiterer Methodenaufruf – eine gleichnamige Methode `println()`, die aber zu einer Klasse `out` mit klein geschriebenem `o` gehört – erfolgt.

Die in Information 25 zusammengefassten Namenskonventionen sind aus Gründen der besseren Lesbarkeit eines Programms zwingend einzuhalten. Die Großschreibweise der Klassen wird u.a. auch dadurch motiviert, dass die Sprache Java mit eigenen Klassenbibliotheken arbeitet, für die durchgängig die Kleinschreibweise gewählt wurde. Hierdurch ist somit eine zufällig gleiche Benennung von Klassen ausgeschlossen.

4.2 Parameter

Die Übergabe von Parametern eröffnet erst die zahlreichen Möglichkeiten, die Methoden zur Strukturierung von Programmen bieten.

- Parameter sind (Eingabe-) Werte, die eine aufrufende Methode der aufgerufenen Methode übergibt
 - beeinflussen die Ausführung bzw. das Ergebnis der aufgerufenen Methode

```

class PrintMaxProgram {

    static void printMax(int x, int y) {      // x, y: formal method parameters
        if (x > y) Out.println(x); else Out.println(y);
    }

    public static void main (String[] arg) {
        printMax(3, 7);                      // 3, 7: parameter values of method call
        int x = readInt(); int y = readInt();
        printMax(x, y);                      // method call with variables
        printMax(1000 * y, x * y);          // method call with expressions
    }
  
```

Information 26: Methoden mit Parametern

Wie das Beispiel der in Information 26 programmierten Methode `printMax()` verdeutlicht, werden die Parameterdeklarationen (hier: `int x, int y`) geklammert nach dem Methodennamen (hier: `printMax()`) geschrieben.

Der bei der Methodendeklaration angegebene Methodenkopf bestehend aus Name und Parameterliste bildet die Schnittstelle der Methode, die von einer aufrufenden Methode eingehalten werden muss.

Beim Aufruf einer Methode ist eine Parameterliste anzugeben, der den Parametern, die durch die Schnittstelle vorgegebenen sind, gerecht wird. Die Schnittstellen-Parameter einer Methode werden auch als formale Parameter bezeichnet, während die Aufrufparameter aktuelle Parameter genannt werden.

Wie das Programm zeigt, können die beim Aufruf der Methode `printMax()` übergebenen aktuellen Parameter einfache ganzzahlige Werte, Variablen oder Ausdrücke sein.

- Übergabe der aktuellen Parameter an die formalen Parameter
 - ggf. Berechnung der Ausdrücke der aktuellen Parameter
 - es handelt sich um Zuweisungen, weshalb Typkompatibilität gefordert ist
- Die Art der Übergabe wird als Call-by-Value bezeichnet
 - formaler Parameter kopiert den Wert des aktuellen Parameter in eine getrennte Speicherzelle
 - Änderungen, die ggf. am formalen Parameter im Methodenrumpf vorgenommen werden, beeinflussen den Wert des aktuellen Parameters nicht


Information 27: Parameterübergabe

Im Zusammenhang mit der Parameterübergabe sind eventuelle Ausdrücke, die als aktuellen Parameter übergeben werden, zunächst zu berechnen. Wichtig ist dabei die Einhaltung der Typkompatibilität (siehe Information 27).

Die Parameterübergabe erfolgt nach dem *Call-by-Value*-Prinzip. Alternative Übergabekonzepte sind das *Call-by-Name*-Prinzip bzw. das *Call-by-Reference*-Prinzip, die eine Änderung des aktuellen Parameterwertes gemäß den Änderungen des formalen Parameters bewirken. Von Java wird ausschließlich das *Call-by-Value*-Prinzip unterstützt.

4.3 Funktionen

Bislang wurden Methoden ohne Rückgabewerte behandelt. Solche Methoden werden auch als Prozeduren bezeichnet. Liefert eine Methode einen Rückgabewert an die aufrufende Methode, handelt es sich um eine Funktion.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck) 

```
class FctMaxProgram {
    static int fctMax(int x, int y) {           // function which provides int result
        if (x > y) return x; else return y;    // return statement
    }

    public static void main (String[] arg) {
        int x = 2;
        Out.println(fctMax(x, 5 * x) * 100);  // function call in parameter expression
                                                // prints: _____
    }
}
```

- Rückgabe des Funktionsergebnisses mittels return-Anweisung
- Funktionen werden üblicherweise als Operanden in einem Ausdruck verwendet
- Es kann immer nur ein Ergebniswert zurückgeliefert werden
 - mehrere Ergebniswerte sind in Form eines Objekts zusammen zu fassen

Interaktion 13: Funktionen

&1

Methoden, die Prozeduren realisieren, sind syntaktisch beschrieben durch einen "leeren" Rückgabewert

-> void (wörtlich übersetzt leer)

Es bietet sich an, die Prozedur printMax() in eine entsprechende Funktion fctMax() zu überführen, die das Maximum nicht ausdrückt sondern als Ergebniswert an die aufrufende Methode übergibt. Hierdurch ergibt sich unmittelbar der Vorteil, dass die Funktion und das von ihr gelieferte Ergebnis als Operand in einem komplexeren Ausdruck genutzt werden kann, wie das Programmbeispiel in Interaktion 13 zeigt.

Zur Rückgabe des Funktionsergebnisses dient die return-Anweisung. Eine Funktion muss durch eine solche return-Anweisung zwingend abgeschlossen werden, was auch durch den Compiler überprüft wird.

Mittels einer `return`-Anweisung kann immer nur ein Ergebniswert zurückgegeben werden. Besteht das Ergebnis aus mehreren Einzelergebnissen, sind (zusammengesetzte) Objekte einzuführen, wie in einer späteren Kurseinheit verdeutlicht wird.

Es sei angemerkt, dass die `return`-Anweisung auch bei Prozeduren eingesetzt werden kann. Hier wirkt diese ohne Parameter ausgeführte Anweisung wie die `break`-Anweisung, d.h. die Prozedur wird abgeschlossen und die Kontrolle geht an die aufrufende Methode über.

4.4 Lokale und globale Namen

Die bislang behandelten Methoden haben ausschließlich Anweisungen enthalten. Daneben können auch Deklarationen von Variablen und Konstanten auftreten.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In einer Methode können Variablen und Konstanten (aber keine weiteren Methoden) deklariert sein
 - sind wie die formalen Parameter lokal zu dieser Methode deklariert, d.h. nur in dieser Methode gültig

```
static void m (int x) {
    final boolean debug = true;
    int y;
    float z;
    ...
    if (debug) Out.println(...);
    ...
}
```

- Methode `m()` deklariert drei lokale Variablen `x`, `y`, `z` und eine lokale Konstante `debug`
 - dürfen nur in `m()` verwendet werden
 - sind außerhalb `m()` nicht sichtbar
 - "leben" nur während der Ausführung der Methode `m()`

Information 28: Lokale Variablen und Konstanten

Die in einer Methode deklarierten Variablen und Konstanten sind wie die formalen Parameter lokale Namen der Methode.

Bezogen auf das in Information 28 angegebene Beispiel bedeutet diese Festlegung, dass die Variablen `x`, `y`, `z` und die Konstante `debug` nur lokal innerhalb der Methode `m()` verwendet werden dürfen. Es handelt sich um so genannte lokale Variablen und Konstanten.

Bei Aufruf der Methode `m()` wird für die lokalen Größen `x`, `y`, `z` und `debug` der benötigte Speicherplatz bereitgestellt und am Ende der Methodenausführung wird dieser Speicherplatz wieder frei gegeben.



```

class Program {
    static int a; static float b;    // declarations of global variables
    static final c;                 // declaration of global constant

    static void m (int x) {
        int y; float z;             // x, y, z: local
        ... // which variables/constants can be used here? _____
    }

    public static void main (String[] arg) {
        ... // which variables/constants can be used here? _____
    }
}

```

- Globale Variablen und Konstanten sind außerhalb einer Methode in der Klasse deklariert
 - sind mit static deklariert
 - können in allen Methoden der Klasse benutzt werden

Interaktion 14: Globale Variablen und Konstanten

Jede Methode ist in einer Klasse deklariert. Auf Klassenebene können neben Methoden auch Variablen und Konstanten deklariert werden, die dann in allen Methoden dieser Klasse benutzt werden können. Solche außerhalb der Methoden deklarierten Variablen und Konstanten heißen globale Variablen.

Am Beispiel des Programms in Interaktion 15 ist zu klären, welche lokalen bzw. globalen Variablen an den beiden gekennzeichneten Stellen im Programm genutzt werden können.



```

class Program1 {

    static void add (int x) {
        int sum = 0;
        sum = sum + x;
    }

    public static void main (String[] arg) {
        add(3); add(17); add(5);
        Out.println(sum)
    }
}

```

```

class Program2 {

    static int sum = 0;
    static void add (int x) {
        sum = sum + x;
    }

    public static void main (String[] arg) {
        add(3); add(17); add(5);
        Out.println(sum)
    }
}

```

- Welche Ausgabe liefern die Programme?
 Program1 _____ Program2 _____

Interaktion 15: Lokale und globale Variable sum

Das in Interaktion 15 behandelte Beispiel hat offensichtlich zum Ziel, die Summe einer Zahlenfolge zu bilden. Während Program1 die Aufgabe durch die Verwendung einer lokalen Variablen `sum` zu lösen versucht, verwendet Program2 eine globale Variable `sum`, wodurch die Aufgabenstellung gelöst wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

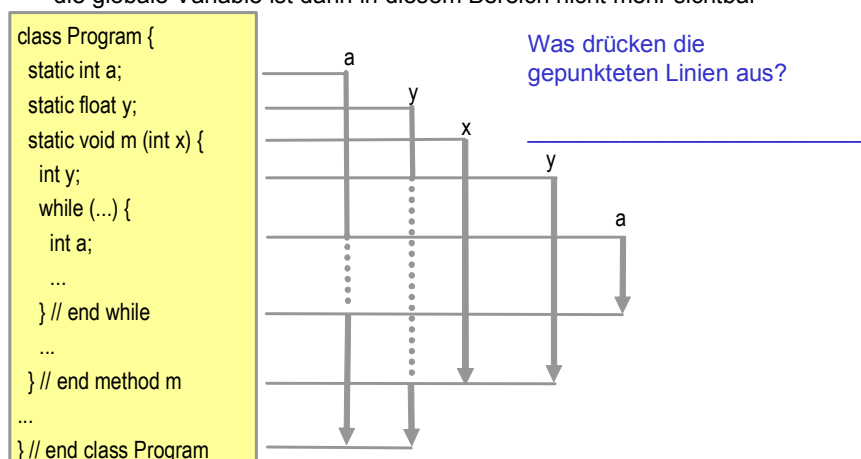
- Grundsätzlich gilt: lokal vor global
- Wenn möglich, sind Variablen lokal und nicht global zu deklarieren
 - globale Variablen nur in begründeten Fällen verwenden
- Vorteile lokaler Variablen gegenüber globalen Variablen
 - eine Methode lässt sich verstehen, ohne das Umfeld kennen zu müssen
 - eine Methode manipuliert bei schreibendem Zugriff auf globale Variablen in schwer nachvollziehbarer Form das Umfeld (Seiteneffekte)
 - Problem der Namenskonflikte zwischen lokalen und globalen Variablen besteht nicht

Information 29: Verwendung lokaler oder globaler Variablen

Globale Variablen existieren im Gegensatz zu lokalen Variablen über Methodengrenzen hinweg. Daher sind Änderungen von globalen Variablen, die in einer Methode erfolgen, auch außerhalb der Methode wirksam. Eine Methode kann somit Ergebnisse über globale Variablen als so genannte "Seiteneffekt" nach außen weitergeben. Diese Form der "impliziten" Ergebnisweitergabe von Methoden sollte nur in ganz bestimmten Fällen genutzt werden, da hierdurch die Wirkungsweise einer Methode undurchsichtig und die Methoden-Schnittstelle "aufgeweicht" wird. Information 29 fasst die Vorteile von lokalen gegenüber den globalen Variablen zusammen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Gültigkeitsbereich einer Variablen ist der Programmbereich, in dem auf diese Variable zugegriffen werden kann
- Der Gültigkeitsbereich einer globalen Variablen wird durch eine gleichnamige lokale Variable unterbrochen
 - die globale Variable ist dann in diesem Bereich nicht mehr sichtbar



Interaktion 16: Gültigkeitsbereich (Sichtbarkeitsbereich) einer Variablen

Wie der Programmausschnitt in Interaktion 16 verdeutlicht, können lokale und globale Variablen (zufällig) den gleichen Namen haben. Es stellt sich die Frage, auf welche Variable im Falle der Existenz einer gleichnamigen globalen und lokalen Variablen zugegriffen wird. Die Antwort führt über den Gültigkeitsbereich, der auch als Sichtbarkeitsbereich bezeichnet wird. Wie die gepunktete Linie andeutet, werden globale Variablen durch lokale Variablen verschattet, d.h. die lokale Variable ist gültig und wird zugegriffen. Konkret gilt also innerhalb der Methode `m()` die deklarierte lokale Variable `int y` und nicht die in der Klasse `Program` deklarierte globale Variable `static float y`. Wie die durchgezogene Pfeillinie andeutet, wird der Gültigkeitsbereich der globalen Variablen `float y` nach Verlassen der Methode `m()` fortgesetzt, weil hiermit der Gültigkeitsbereich der lokalen Variablen `int y` endet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Das Zeitintervall, in dem eine Variable einen Speicherplatz beansprucht, wird als Lebensdauer der Variablen bezeichnet
- Lokale und globale Variablen werden in verschiedenen Speicherbereichen angelegt
 - lokale Variablen im (Laufzeit-) Keller (*stack*)
 - globale Variablen auf der Halde (*heap*)
- Der Keller wächst bzw. schrumpft mit jedem Methodenaufruf bzw. mit jedem Verlassen einer Methode
 - im Keller sind zu jedem Zeitpunkt die lokalen Variablen und Konstanten der aktuellen Methodenaufkette gespeichert

Information 30: Lebensdauer und Speicherorganisation

Während der Gültigkeitsbereich festlegt, wann eine Variable benutzt werden kann, macht die Lebensdauer eine Aussage darüber, wie lange die Variable einen Platz im Speicher belegt. Offensichtlich gibt die globale Variable im obigen Beispiel während der Verschattung durch die lokale Variable ihren Speicherplatz nicht frei; man kann in diesem Programmbereich nur nicht auf die Speicherzellen zugreifen.

Wie Information 30 beschreibt, werden globale und lokale Variablen in unterschiedlichen Speicherbereichen gehalten. Ein Charakteristikum des als Keller (*Stack*) bezeichneten Speicherbereichs, in dem die lokalen Variablen gehalten werden, ist dessen Anwachsen und Schrumpfen. Der Keller ist eine wichtige Datenstruktur in der Informatik.

5 DATENTYPEN

Die einzigen Datentypen, die in den bisher behandelten Programmen bzw. Programmausschnitten vorkommen, sind boolesche Werte und ganze Zahlen. Zur Darstellung ganzen Zahlen stehen – je nach dem benötigten Wertebereich – die in Information 31 angegebenen Datentypen in Java zur Verfügung [Mö03].

- Bislang wurden neben dem Datentyp boolean ausschließlich Datentypen zur Darstellung ganzer Zahlen benutzt
 - Datentypen byte, short, int, long
- Neben den ganzen Zahlen unterstützt Java auch das Rechnen mit Gleitkommazahlen
 - Datentypen float, double
- Eine weitere elementare Form von Daten sind die Zeichen
 - Datentyp char

Information 31: WEITERE DATENTYPEN - Überblick

In diesem Kapitel werden mit den Gleitkommazahlen und den Zeichen zwei weitere elementare Datentypen eingeführt [Mö03].

5.1 Gleitkommazahlen

Gleitkommazahlen werden in Form der in Information 32 dargestellte Exponentendarstellung geschrieben.

- Gleitkommaformat
 - Syntax: Kommazahl 'E' GanzeZahl
 - Wert: Kommazahl * $10^{\text{GanzeZahl}}$
 - Beispiele
 - 0.0314E2 Wert: $0.0314 * 10^2 = 3.14$
 - 999E-3 Wert: $999 * 10^{-3} = 0.999$
- Datentypen in Java zur Darstellung von Gleitkommazahlen
 - float 32 Bits $\approx 1.4\text{E-}45$... $\approx 3.4\text{E}38$
 - double 64 Bits $\approx 4.9\text{E-}324$... $\approx 1.8\text{E}308$

```
float x,y; // declaration of two float variables x and y
double z; // declaration of one double variable z
```

Information 32: Gleitkommazahlen

Die ganze Zahl nach dem Buchstaben 'E' (auch die Kleinschreibweise 'e' ist zulässig) gibt die Zehnerpotenz an, mit der die Kommazahl multipliziert wird.

Es werden die zwei Datentypen float und double zum Arbeiten mit Gleitkommazahlen angeboten. Der Datentyp double ist dann zu nutzen, wenn der durch den Datentyp float angebotene Wertebereich nicht ausreicht.



- Harmonische Reihe bis zum n. Glied:
 - $1/1 + 1/2 + 1/3 + \dots + 1/n$
- Berechnung in einem Java-Programm mittels
 1. Einlesen der ganzen Zahl n
 2. Deklaration einer float-Variablen sum, die zur Aufnahme des Ergebnisses dient
 3. Berechnung der harmonischen Reihe in einer while-Schleife
 - hier absteigend, d.h. $1/n + 1/(n-1) + 1/(n-2) + \dots + 1/2 + 1$
 4. Ausgabe des Ergebnisses

```

int n = _____;           // 1
_____ ;                     // 2
int i = n;                    // 3
while (i > 0) {               // 3
    sum = sum + (float) _____; // type casting
    i = i - 1;                // 3
}                               // 3
_____ ;                     // 4

```

Interaktion 17: Berechnung der Harmonischen Reihe

In Interaktion 17 wird anhand einer einfachen Programmieraufgabe, der Berechnung der Harmonischen Reihe, das Arbeiten mit Gleitkommazahlen aufgezeigt. Das Programm ist mit Hilfe der angegebenen Beschreibung entsprechend zu vervollständigen.

Auf das Rechnen mit Gleitkommazahlen und Typkonversionen wird in der folgenden Interaktion 18 näher eingegangen.



- Übliche Rechen- und Vergleichsoperationen
- Kompatibilitätsbeziehungen
double \supset float \supset long \supset int \supset short \supset byte
- Eine Gleitkommakonstante ist in Java immer double
- Konversionsregel:
Ein "kleinerer" Operandentyp wird vor Ausführung einer Operation in den "größeren" konvertiert

```

double d; float f; int i;
f = i; // ok: float contains int
i = f; // wrong: float not in int
i = (int) f; // ok: conversion to int

```

```

double d; float f;
d = 3.14; // ok: variable and
           // constant are double
f = 3.14; // wrong: double not in float
f = 3.14f; // f attached is called float suffix

```

```

double d; float f; int i;
// what is the type of the result?
... f + i ... // _____
... d * (f + i) ... // _____
... f / 3 ... // _____
... (float) i / 3 // _____

```

Interaktion 18: Eigenschaften der Datentypen double und float

Mit Gleitpunktkommazahlen kann ähnlich wie mit den ganzen Zahlen gerechnet werden. An Rechenoperationen stehen die Addition (+), die Subtraktion (-), die Multiplikation (*), die Division (/) und die Modulo-Operation (%) zur Verfügung. Dabei ist die Modulo-Operation so definiert, dass für zwei Gleitkommazahlen x und y die Operation $x \% y$ eine Gleitkommazahl r ergibt, die den Rest der Division von x und y darstellt (also $r = x - q * y$, wobei q der ganzzahlige Teil von x / y ist).

Interaktion 18 zeigt die Enthaltenseins-Beziehung der beiden Datentypen `double` und `float` sowie der bereits eingeführten Datentypen zu den ganzen Zahlen auf. Hieraus ergeben sich die bei einer Zuweisung zu beachtenden Kompatibilitätsbeziehungen. Eine Zuweisung eines umfassenden Typs ist nur nach vorheriger Typ-Konversion möglich, wobei dadurch ein Informationsverlust entstehen kann.

Während Java ganzzahligen Konstanten bekanntlich den Datentyp `int` zuordnet, sind Gleitkomma-Konstanten immer automatisch vom Datentyp `double`. Daher muss bei einer Zuweisung einer Gleitkomma-Konstanten zu einer `float`-Variablen an die Konstante ein `f` angehängt werden, damit dieses nicht vom Typ `double` sondern vom Typ `float` ist.

Aufgrund des erheblich höheren Rechenaufwands, der durch die Gleitkommaoperationen entsteht, sollten die Datentypen `double` und `float` auch wirklich nur dann genutzt werden, wenn die Nachkommastellen erforderlich sind.

5.2 Zeichen

Neben den Zahlen gehören die Zeichen zu den wichtigsten Daten, die in Programmen verarbeitet werden müssen. In Java steht für den Umgang mit Zeichen der Datentyp `char` zur Verfügung. Wie bei Zahlen können Zeichen als Konstanten und Variablen auftreten. Zunächst werden die Zeichen-Konstanten näher betrachtet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Zeichen sind wie Zahlen Daten von einem besonderen Typ
 - der Typ trägt in Java die Bezeichnung `char` (steht für character)
 - wie bei Zahlen lassen sich Zeichen-Konstanten und Zeichen-Variablen unterscheiden
- Zeichen-Konstanten
 - werden in Hochkommata (') gestellt
 - Beispiele: `'a'` `'b'` `'A'` `'?'`
 - werden durch einen Zeichencode in Zahlen umgewandelt
 - bekannte Zeichencodes sind ASCII (American Standard Code of Information Interchange) und Unicode

Information 33: Zeichen

Information 33 zeigt Beispiele von Konstanten, die sich durch einem zugrunde liegenden Zeichencode in Zahlenwerte umwandeln lassen.

- ASCII definiert 128 Zeichen
- Ein ASCII-Zeichen wird durch 1 Byte dargestellt
- Zeichenbereich 0x00 bis 0x1f und Zeichen 0x7f sind nicht sichtbare Zeichen, wie z.B.
 - 0x0a neue Zeile linefeed (LF)
 - 0x0d Zeilenende carriage return (CR)
 - 0x09 Tabulatorsprung horizontal tab (HT)
 - 0x7f Löschzeichen delete (DEL)
- Zeichenbereich 0x20 bis 0x7E enthält druckbare Zeichen, wie z.B.
 - 0x20 Leerzeichen
 - 0x30 - 0x39 Zahlbereich 0 bis 9
 - 0x41 - 0x5a Großbuchstaben 'A' bis 'Z'
 - 0x61 - 0x7a Kleinbuchstaben 'a' bis 'z'

Information 34: Zeichencode ASCII

Der in Information 34 näher beschriebene Zeichencode ASCII – *American Standard Code of Information Interchange* – ist ein 1-Byte-Code, durch den insgesamt 128 Zeichen definiert sind.

Die Organisation des Zeichenbereichs in nicht sichtbare und druckbare Zeichen ist in Information 34 ausgeführt.

- Im ASCII-Zeichensatz fehlen verschiedene Zeichen, die in bestimmten Problembereichen zwingend erforderlich sind
 - z.B. mathematische Zeichen, griechische Symbole, Umlaute, arabische, chinesische, ... Zeichen
- In Java wird daher der den ASCII-Zeichensatz erweiternde Zeichensatz Unicode verwendet
 - 2-Byte-Code (d.h. $2^{16} = 65\,536$ mögliche Zeichen)
 - wird als 4-stellige Hexadezimalzahl \unnnn beschrieben
 - z.B. \u000a (LF), \u0009 (HT), \u00e4 (ä), \u03b1 (α)
- Für häufig vorkommende Steuerzeichen können speziell von Java vorgesehene Escape-Sequenzen benutzt werden
 - z.B. '\n' (\u000a), '\\' (backslash \u005c), '\"' (single quote \u0027)

Information 35: Zeichencode Unicode

Die 128 durch den ASCII-Zeichencode festgelegten Zeichen reichen in Bereichen, in denen z.B. mathematische Zeichen oder Zeichen gewisser Sprachen (z.B. griechisch, kyrillisch, arabisch oder Umlaute der deutschen Sprache) benötigt werden, nicht aus. Daher wird in Java der so genannte Unicode-Zeichensatz verwendet, der die ASCII-Zeichen (\u0000 bis \u007f, zur Schreibweise siehe Information 35) übernimmt und diese ergänzt:

Durch die Bereitstellung von *Escape*-Sequenzen – also Zeichenreihen, in denen der als *Escape*-Zeichen genutzte Backslash '\ ' verwendet wird – lassen sich die häufig verwendeten Steuerzeichen in einer übersichtlicheren Form innerhalb von Programmen nutzen.

Variablen, die anstelle von Zahlen die oben beschriebenen Unicode-Zeichen als Werte tragen sollen, werden in Java mit dem Datentyp `char` deklariert.

- Zeichenvariablen werden mit dem Typ `char` deklariert
- Zeichen werden intern durch den Unicode in Zahlenwerte codiert
 - mit `char`-Werten lässt sich "rechnen"
- Typ `char` wird in der Hierarchie der Standardtypen auf der gleichen Stufe wie `short` eingeordnet

double \supset float \supset long \supset int \supset short \supset byte
 \supset char

```
char ch1, ch2 = 'c';
ch1 = 'a';

...

int i = ch1 - ch2;    // i == ____
...
ch1 = ch2;          // ok? ____
i = ch1;            // ok? ____
                    // i == ____
ch2 = (char) (i + 1); // ok? ____
                    // ch2 == ____
```

Interaktion 19: Zeichenvariablen

Einer Zeichenvariablen kann bereits bei der Deklaration oder in einer separaten Zuweisung ein Zeichenwert zugewiesen werden. Intern wird bei der Deklaration einer Zeichenvariablen ein 2-Byte großer Speicherplatz vorgesehen, in dem bei Zuweisung eines Zeichenwertes zu dieser Variablen der entsprechende Unicode-Zahlenwert gespeichert werden.

Aufgrund dieser internen Darstellung ist einfach nachvollziehbar, dass man mit Zeichenvariablen so wie mit Zahlvariablen "rechnen" kann.

In der in Interaktion 19 gezeigten Standardtypen-Hierarchie ist der Datentyp `char` wie `short` eingeordnet, weshalb z.B. die Zuweisung des Wertes einer Zeichenvariablen zu einer `int`-Variablen ohne Typ-Konversion zulässig ist. Im umgekehrten Fall – also im Falle einer Zuweisung des Wertes einer `int`-Variablen zu einer Zeichenvariablen – ist eine Typ-Konversion erforderlich.



- Zeichen können über die Methoden von In und Out gelesen und geschrieben werden
- Das Einlesen einer Zahl erfolgt durch Einlesen von Ziffern (Zeichen) und entsprechender Umwandlung
- Beispiel: "123"
 - wert("123")
$$= \text{wert}('1') * 10^2 + \text{wert}('2') * 10^1 + \text{wert}('3') * 10^0$$

$$= (\text{wert}('1') * 10 + \text{wert}('2')) * 10 + \text{wert}('3')$$

```
char ch = In.read();
if ('0' <= ch && ch <= '9')
    Out.println(ch + "is a digit")
```

```
int val = 0;
// only digit chars should be allowed
char ch = In.read();
// compute value
while _____ {
    val = _____;
    ch = In.read(); // read next char
}
Out.println("val is " + val);
```

Interaktion 20: Zeichen und Ein-/Ausgabe

Zeichen müssen sich auch in ein Programm einlesen und von einem Programm ausgehen lassen. Hierzu stellen die weiter oben eingeführten Klassen In und Out entsprechende Methoden zur Verfügung.

Wie der zweite Programmausschnitt zeigt, baut die Eingabe einer Zahl auf der Zeicheneingabe auf: Die eingegebene Zahl ist für das Programm zunächst eine Folge von Zeichen, die in der Menge {0, ..., 9} enthalten sind. Die eigentliche Zahl wird durch das in Interaktion 20 angegebene Vorgehen, das in eine entsprechende Schleife umzusetzen ist, ermittelt. Die Umformung der Berechnung in die geklammerte Form wird als Horner-Schema bezeichnet.

- Die Java-Bibliothek bietet zahlreiche Operationen im Zusammenhang mit Zeichen an
 - Verwendung von Bibliotheksfunktionen (Standardfunktionen) ist Teil eines guten Programmierstils
 - erhöht die Stabilität und vermindert die Komplexität von Programmen
- Beispiele
 - Character.isLetter(ch)
 - liefert true, falls ch ein Buchstabe ist
 - Character.isDigit(ch)
 - liefert true, falls ch eine Ziffer ist
 - Character.toUpperCase(ch)
 - liefert den entsprechenden Großbuchstaben, falls ch ein Kleinbuchstabe ist

Information 36: Bibliotheksfunktionen

Da verschiedene Zeichen-Operationen – wie z.B. die im obigen Programm verwendete Funktion zur Überprüfung, ob ein Zeichen eine Ziffer ist – immer wieder auftreten, stellt Java eine umfangreiche Bibliothek dieser Operationen als so genannte Standardfunktionen bereit.

In dieser Veranstaltung werden solche nahe liegenden Operationen als Beispiele verwendet, um die Programmierung im Kleinen zu üben. Daher werden diese Standardfunktionen – wie im obigen Beispiel die Standardfunktion `isDigit(ch)` – ausnahmsweise nicht verwendet.

Hiermit sind die Grundlagen der Programmierung, die das Erstellen erster vollständiger Java-Programme ermöglichen, abgeschlossen. Die vorgestellten Programmier-elemente werden durch die Kurseinheiten zur imperativen und objektorientierten Programmierung [C&M-IP, C&M-OP] sukzessive ergänzt.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
ASCII	<i>American Standard Code of Information Interchange</i> Weit verbreiteter 1-Byte-Code, durch den Zeichen in 7 Bit langen Binärzahlen codiert werden (achtes Bit ist ungenutzt). Ein den ASCII-Code erweiternder Code ist der so genannte Unicode.
BNF	Backus-Naur-Form Schreibweise für Regeln einer Grammatik.
<i>Call-by-Value</i> -Prinzip	Prinzip im Zusammenhang mit der Parameterübergabe, das besagt, dass die Werte der Aufrufparameter (aktuelle Parameter) an die entsprechenden formalen Parameter übergeben werden. Insbesondere werden die Werte der aktuellen Parameter durch den Aufruf nicht verändert.
EBNF	Erweiterte BNF Erweiterung der BNF um Metaregeln zur ausdrucksstärkeren Formulierung von Regeln einer Grammatik.
<i>Escape</i> -Zeichen	Ein ausgezeichnetes Zeichen, das zur Darstellung spezieller Sequenzen (<i>Escape</i> -Sequenzen) dient. Beispiel: In Java dient das Zeichen '\ (Backslash) als <i>Escape</i> -Zeichen, um beispielsweise mittels '\n' einen Zeilenumbruch darzustellen.
Funktion	Im Kontext der Java-Programmierung ist eine Funktion eine Methode, die einen Rückgabewert an die aufrufende Methode liefert.
Grammatik	Regelsystem, durch das die Syntax einer (formalen) Sprache, z.B. einer Programmiersprache, festgelegt wird.
JDK	<i>Java Development Kit</i> Softwarepaket, das einen Java-Übersetzer und die JVM beinhaltet.
JVM	<i>Java Virtual Machine</i> Software, die den Java-Bytecode ausführt. Eine virtuelle Maschine bezeichnet einen in Software realisierten Prozessor.
Methode	Bezeichnung der in Klassen auftretenden Rechenvorschriften, denen Parameter übergeben werden können und die einen Rückgabewert abliefern können. Namenskonvention: Methodennamen beginnen mit einem kleingeschriebenen Buchstaben. Beispiel: Methode <code>main()</code> ist eine in Java ausgezeichnete Methode, da diese den Einstiegspunkt in das Programm markiert.

Programmieren	Beschreibung eines Problem in einer Form, dass es mittels eines Rechensystems gelöst werden kann.
Schleife	Sprachelement einer Programmiersprache, durch das eine mehrmalige Ausführung von Programmteilen ermöglicht wird.
Schlüsselwörter	Wörter einer Sprache, die deren grundlegende Syntax angibt. Die Sprache Java besteht beispielsweise aus 48 Schlüsselwörtern. Beispiele: if, for, class, void
Software-Entwicklung	Prozess der Erstellung eines Programms. Der Prozess zerfällt in mehrere iterativ durchlaufene Phasen. Verwandte Begriffe: <i>Software Engineering</i> , Programmieren (im Großen)
UML	<i>Unified Modeling Language</i> Sprache, die aus graphischen Elemente zur semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) besteht.
von-Neumannscher Flaschenhals	Bezeichnung eines Engpasses, der in der von-Neumann-Architektur aus der im Vergleich zur Prozessorbearbeitungszeit relativ langen Speicherzugriffszeiten entsteht.
Zuweisung	Wichtige Form einer Anweisung, durch die einer Variablen (linke Seite) ein Wert (rechte Seite) zugewiesen werden kann.
Zuweisungskompatibilität	Bei einer Zuweisung muss der Typ der rechten Seite den Typ der linken Seite einschließen.

Index

1-Byte-Code 38	Methoden 21, 26
American Standard Code of Information Interchange 11	Programmieren 3
Call-by-Value-Prinzip 30	Schleife 19
Erweiterte Backus-Naur-Form 6	Schlüsselwörter 11
Escape-Zeichen 39	Software-Entwicklung 4
Funktion 30	Unified Modeling Language 4
Grammatik 6	von-Neumannscher Flaschenhals 5
Java Development Kit 26	Zuweisung 14
Java Virtual Machine 25	Zuweisungskompatibilität 14

Informationen und Interaktionen

Information 1: PROGRAMMIERGRUNDLAGEN	3
Information 2: Dem Programmieren zugrunde liegendes Vorgehen	4
Information 3: Vom Quell-Programm zum ablauffähigen Programm	5
Information 4: Syntaxfestlegung mittels einer Grammatik	6
Information 5: Metazeichen der Erweiterten Backus-Naur-Form (EBNF)	7
Information 6: GRUNDLEGENDE SPRACHELEMENTE – Überblick	9

Information 7: Namen	9
Information 8: Erste Sprachelemente	11
Information 9: Variablen und deren Deklaration	12
Information 10: Standardtypen für ganze Zahlen und Datentyp	13
Information 11: Arithmetischer Ausdruck	15
Information 12: Ergebnistyp eines arithmetischen Ausdrucks und Typkonversion	16
Information 13: Beispiele	18
Information 14: Hängender else-Zweig	19
Information 15: Schleifen	20
Information 16: PROGRAMMSTRUKTUR – Grundstruktur eines Java-Programms	21
Information 17: Erstes vollständiges Java-Programm	22
Information 18: Ein-/Ausgabe	22
Information 19: Methode Out.print()	23
Information 20: Ergänzung des Beispielprogramms um die Eingabe von Werten	24
Information 21: Übersetzung und Ausführung des Java-Programms	25
Information 22: Konkrete Arbeitsschritte am Beispiel von FirstInOutProgram	25
Information 23: METHODEN - Einführung	26
Information 24: Methodendeklaration und Methodenaufruf	27
Information 25: Methodenaufrufkette und Namenskonventionen	28
Information 26: Methoden mit Parametern	29
Information 27: Parameterübergabe	29
Information 28: Lokale Variablen und Konstanten	31
Information 29: Verwendung lokaler oder globaler Variablen	33
Information 30: Lebensdauer und Speicherorganisation	34
Information 31: WEITERE DATENTYPEN - Überblick	35
Information 32: Gleitkommazahlen	35
Information 33: Zeichen	37
Information 34: Zeichencode ASCII	38
Information 35: Zeichencode Unicode	38
Information 36: Bibliotheksfunktionen	40
Interaktion 1: PROGRAMMERSTELLUNG UND PROGRAMM - Einführung	3
Interaktion 2: Ausführung eines Programmbefehls	5
Interaktion 3: Syntaxdiagramm	7
Interaktion 4: Syntaxdiagramm und Grammatik zu Gleitkommazahlen	8
Interaktion 5: Richtlinien	10
Interaktion 6: Zuweisung	14
Interaktion 7: Zuweisungskompatibilität von Variablen ganzzahligen Typs	14
Interaktion 8: Datentyp boolean	16
Interaktion 9: Bedingte Anweisung	17
Interaktion 10: Anweisungsblöcke	18
Interaktion 11: Beispiel zur while-Anweisung	20
Interaktion 12: Ergänzung des Beispiel-Programms um die Ergebnis-Ausgabe	24
Interaktion 13: Funktionen	30
Interaktion 14: Globale Variablen und Konstanten	32
Interaktion 15: Lokale und globale Variable sum	32
Interaktion 16: Gültigkeitsbereich (Sichtbarkeitsbereich) einer Variablen	33
Interaktion 17: Berechnung der Harmonischen Reihe	36
Interaktion 18: Eigenschaften der Datentypen double und float	36
Interaktion 19: Zeichenvariablen	39
Interaktion 20: Zeichen und Ein-/Ausgabe	40

Literatur

- [C&M-GI] Cooperation&Management: GRUNDBEGRIFFE DER INFORMATIK, Kursdokument zur Vorlesung "INFORMATIK I", <http://www.cm-tm.uka.de/info1>, Universität Karlsruhe (TH), C&M (Prof. Abeck).
- [C&M-IP] Cooperation&Management: IMPERATIVE PROGRAMMIERUNG, Kursdokument zur Vorlesung "INFORMATIK I", <http://www.cm-tm.uka.de/info1>, Universität Karlsruhe (TH), C&M (Prof. Abeck).
- [C&M-IÜ] Cooperation&Management: INFORMATIK I IM ÜBERBLICK, Kursdokument zur Vorlesung "INFORMATIK I", <http://www.cm-tm.uka.de/info1>, Universität Karlsruhe (TH), C&M (Prof. Abeck).
- [C&M-OP] Cooperation&Management: OBJEKTORIENTIERTE PROGRAMMIERUNG, Kursdokument zur Vorlesung "INFORMATIK I", <http://www.cm-tm.uka.de/info1>, Universität Karlsruhe (TH), C&M (Prof. Abeck).
- [C&M-SMU] Cooperation&Management, SYSTEMMODELLIERUNG MIT DER UML, Kursdokument zur Vorlesung "INTERNET-SYSTEME UND WEB-APPLIKATIONEN", <http://www.cm-tm.uka.de/iswa>, Universität Karlsruhe (TH), C&M (Prof. Abeck).
- [Me90] Bertrand Meyer: Objektorientierte Softwareentwicklung, Carl Hanser Verlag, 1990.
- [Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.
- [Oe01] Bernd Oestereich: Objektorientierte Software-Entwicklung – Analyse und Design mit der Unified Modeling Language, Oldenbourg Verlag, 2001.