

Vorlesung



Programmieren I und II

Unit 1

Einleitung und Grundbegriffe der Programmierung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

1

1

Disclaimer



Zur rechtlichen Lage an Hochschulen:

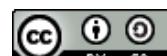
Dieses Handout und seine Inhalte sind durch den Autor selbst erstellt. Aus Gründen der Praktikabilität für Studierende lehnen sich die Inhalte stellenweise im Rahmen des Zitatrechts an Lehrwerken an.

Diese Lehrwerke sind explizit angegeben.

Abbildungen sind entweder selber erstellt, als Zitate kenntlich gemacht oder unterliegen einer Lizenz, die nicht die explizite Nennung vorsieht. Sollten Abbildungen in Einzelfällen aus Gründen der Praktikabilität nicht zweifelsfrei als Zitat kenntlich sein, so ergibt sich die Herkunft immer aus ihrem Kontext: „Zum Nachlesen ...“.

Creative Commons:

Und damit andere mit diesen Inhalten vernünftig arbeiten können, wird dieses Handout unter einer Creative Commons Attribution-ShareAlike Lizenz (CC BY-SA 4.0) bereitgestellt.



<https://creativecommons.org/licenses/by-sa/4.0>

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

2

2



Prof. Dr. rer. nat. Nane Kratzke

*Praktische Informatik und
betriebliche Informationssysteme*

- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: nane.kratzke@th-luebeck.de



@NaneKratzke

Updates der Handouts auch über Twitter #prog_inf und
#prog_itd

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

3

3

Units

1. Semester

Unit 1
Einleitung und
Grundbegriffe

Unit 2
Grundelemente
imperativer Programme

Unit 3
Selbstdefinierbare
Datentypen und
Collections

Unit 4
Einfache I/O
Programmierung

Unit 5
Rekursive
Programmierung,
rekursive
Datenstrukturen,
Lambdas

Unit 6
Objektorientierte
Programmierung und
UML

Unit 7
Konzepte
objektorientierter
Programmiersprachen,
Klassen vs. Objekte,
Pakete und Exceptions

Unit 8
Testen (objektorientierter)
Programme

Unit 9
Generische Datentypen

Unit 10
Objektorientierter Entwurf
und objektorientierte
Designprinzipien

Unit 11
Graphical User Interfaces

Unit 12
Multithread
Programmierung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

4

4

Abgedeckte Ziele dieser UNIT



Kennen existenter Programmierparadigmen und Laufzeitmodelle

Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)

Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenzytypen (insbesondere Liste, Stack, Mapping)

Verstehen des Unterschieds zwischen Werte- und Referenzsemantik

Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen

Kennen des Algorithmusbegriffs, Implementieren einfacher Algorithmen

Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung

Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)

Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)

Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software

Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien

Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Multithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

5

5

In dieser Unit



Einleitung

- Was ist Programmieren?
- Programmierparadigmen
- Laufzeitmodelle von Programmiersprachen
- Grundlegende Begrifflichkeiten bei Programmiersprachen (am Bsp. von Java)

Etwas mehr Java Syntax

- Weitere Begrifflichkeiten bei Programmiersprachen
- Eingaben von der Konsole einlesen
- Ausgaben auf der Konsole ausgeben

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

6

6

Denken wie ein Informatiker



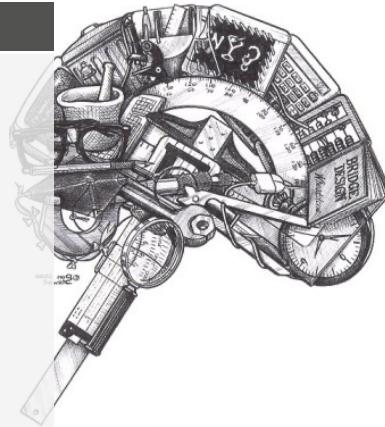
Computational Thinking

Decomposition: Große Probleme in kleinere Probleme zerlegen (ggf. mehrmals).

Pattern Recognition: Erkennen (und nutzen) von Mustern (in Daten).

Abstraction: Konkrete Probleme auf allgemeine Probleme zurückführen.

Algorithm Design: Schritt-für-Schritt-Anweisungen entwickeln.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

7

7

Kurze Anschauungsaufgabe



Nennen Sie mir die Anzahl an Sitzplätzen im gezeigten Hörsaal.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

8

8

Kurze Anschauungsaufgabe



Nennen Sie mir die Anzahl an Sitzplätzen im gezeigten Hörsaal.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

9

9

Wie haben Sie das eigentlich gemacht?

Pattern Recognition:

Sie haben vermutlich ein Muster erkannt: Das Muster wie Sitze in Form von Sitzreihen angeordnet sind (und wissen dies zu nutzen).

Abstraction:

Sie haben vermutlich die Aufgabe als Rechteckproblem (Länge mal Breite) abstrahiert. Der Hörsaal besteht für dieses Problem nur noch aus mehreren Sitzreihen gleicher Länge (die Studenten interessieren hier nicht, auch nicht die Fenster, Lampen, **oder wer gerade zu spät kommt**, etc.)

Decomposition:

Sie haben das Problem vermutlich in zwei Teilprobleme zerlegt. 1. Sitzreihenbestimmung und 2. die Bestimmung der Anzahl an Sitzreihen

Algorithm Design:

Sie haben vermutlich unbewusst eine Anweisungsfolge abgearbeitet.

1. Finde eine zum Abzählen geeignete Sitzreihe
2. Zähle die Anzahl an Sitzen in dieser Reihe
3. Bestimme wieviele Sitzreihen es gibt
4. Multipliziere beide Werte miteinander

10



11

Fokus dieser Vorlesung

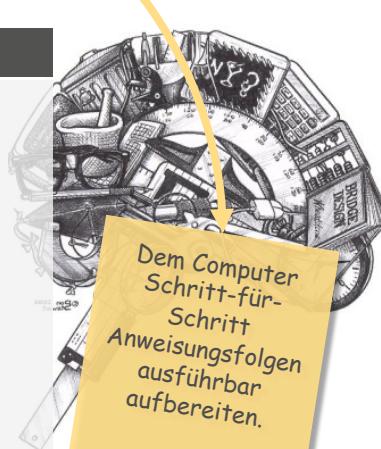
Computational Thinking

Decomposition: Große Probleme in kleinere Probleme zerlegen (ggf. mehrmals).

Pattern Recognition: Erkennen (und nutzen) von Mustern (in Daten).

Abstraction: Konkrete Probleme auf allgemeine Probleme zurückführen.

Algorithm Design: Schritt-für-Schritt-Anweisungen entwickeln.





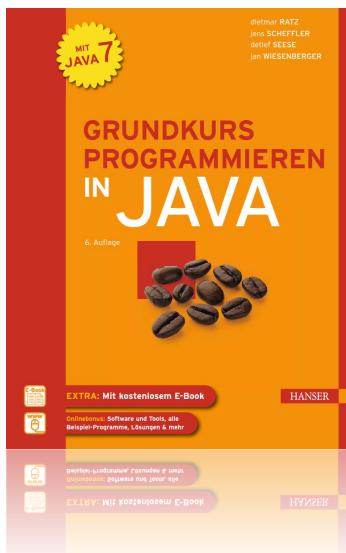
TECHNISCHE HOCHSCHULE LÜBECK

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

12

12

Zum Nachlesen ...



Kapitel 1

Einleitung

Kapitel 2

Grundbegriffe aus der Welt des Programmierens

Kapitel 3

Aller Anfang ist schwer

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

13

13

Grundbegriffe des Programmierens



- **Computer:** Programmierbares technisches Gerät zur Verarbeitung und Speicherung von Daten mittels Algorithmen
- **Algorithmus:** Berechnungsvorschrift zur automatischen Berechnung eines Problems (z.B. Sortieren von Zahlen)
- **Programm:** Formulierung eines Algorithmus in einer für einen Computer ausführbaren Form, d.h. in einer Programmiersprache (z.B. in JAVA)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

14

14

Programmiersprachen im Vergleich



Maschinensprache

```
01110110  
11100101
```

Assembler

```
011001 LD RG1 23  
010111 MOV RG7 RG2  
010010 ADD RG2 RG1  
010001 ADD 10 PRINT "HALLO"  
010011 LD 20 SET A = 7  
DIV 30 GOSUB public class HelloWorld {  
MOV 40 PRINT     public static void main(String[] args) {  
50 GOSUB         System.out.println("Hallo!");  
60 GOTO     }  
}
```

Frühe, problemorientierte Programmiersprache

Java

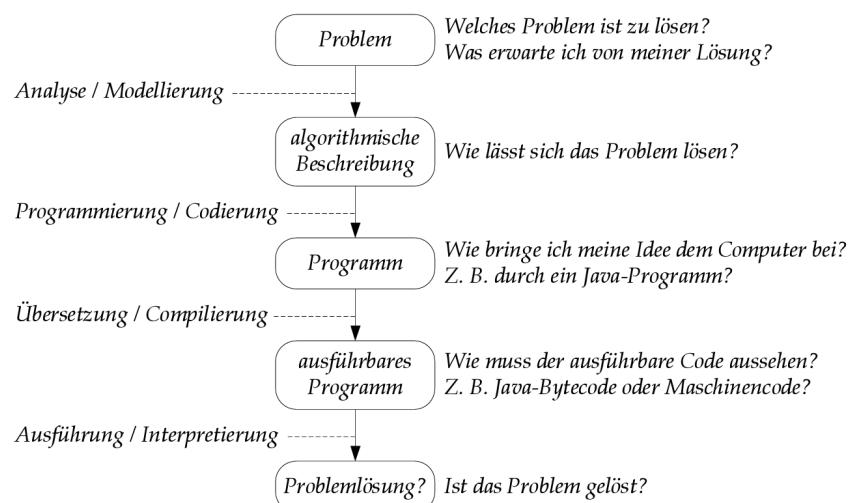
Quelle: Grundkurs Programmieren in Java

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

15

15

Was bedeutet Programmieren?

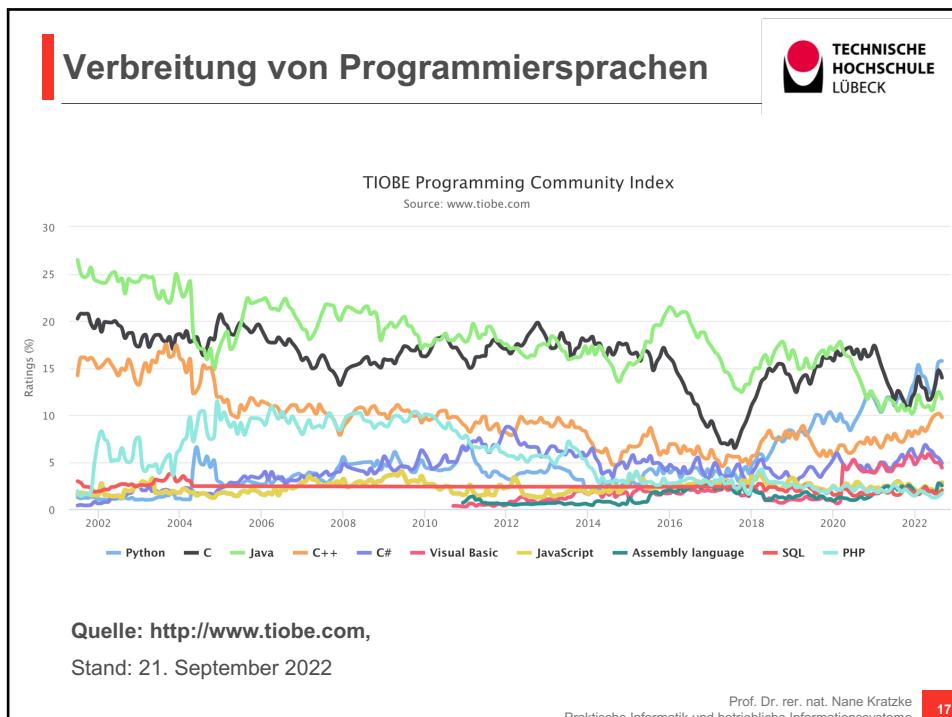


Quelle: Grundkurs Programmieren in Java

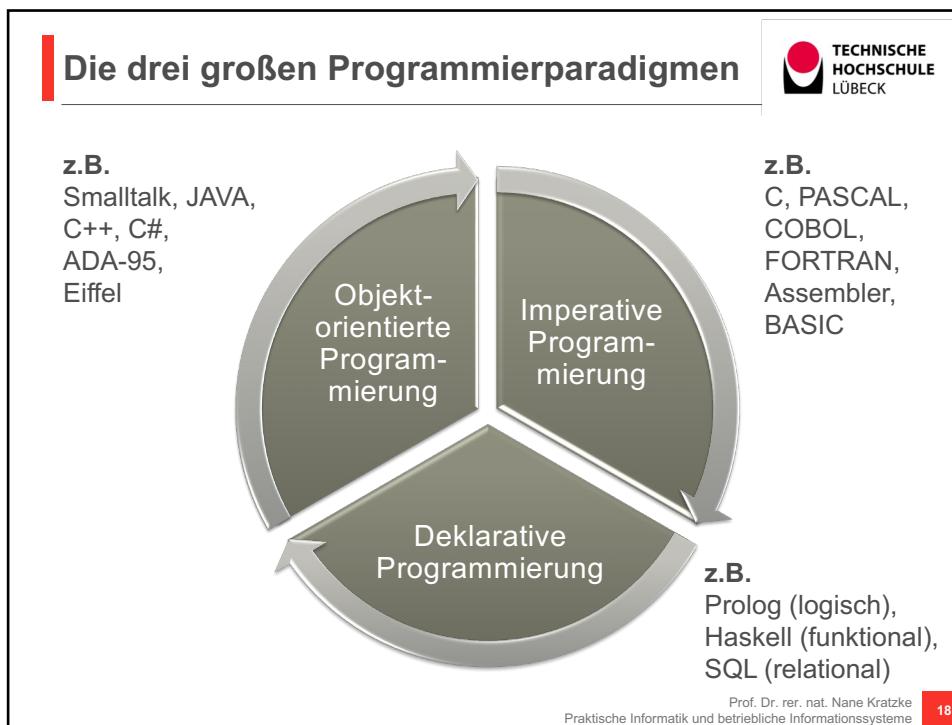
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

16

16



17



18

Imperative Programmierung



Bei allen imperativen Programmiersprachen versteht man ein Computerprogramm als

- lineare Folge von Befehlen, die der Rechner in einer definierten Reihenfolge abarbeitet.
- Daten werden häufig in Variablen gespeichert. Die Werte in Variablen können sich im Programmablauf durch Befehlsabarbeitung ändern.
- Daher kann man sie auch als zustandsorientierte Programmierung bezeichnen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

19

19

Deklarative Programmierung



In der deklarativen Programmierung wird formuliert, welches Ergebnis gewünscht ist.

- Bei deklarativen Paradigmen gibt es keine Nebeneffekte.
- Beweise (zum Beispiel Korrektheitsbeweis, Beweise über Programmeigenschaften) sind dank mathematischer Basis durchführbar.
- Aufgrund dessen jedoch teilweise geringe Akzeptanz (man spricht gern von sogenannten Akademikersprachen).

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

20

20

Objektorientierte Programmierung



Unter Objektorientierung versteht man eine Sichtweise auf komplexe Systeme, bei der ein System durch das Zusammenspiel kooperierender Objekte beschrieben wird.

- Ein Objekt hat
 - Attribute (Eigenschaften)
 - Methoden (Verhalten) und
 - kann Nachrichten empfangen und senden.
- Das Konzept der Objektorientierung wurde entwickelt, um die Komplexität von SW-Programme besser zu beherrschen.
- Das objektorientierte Programmierparadigma fasst Daten und zugehörige Programmteile zu einer Einheit zusammenzufassen, um Konzepte der realen Welt besser nachbilden zu können.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

21

21

Die drei gängigen Laufzeitmodelle von Programmiersprachen



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

22

22

Compiler



Ein Compiler ist ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm – genannt **Quellprogramm** – in ein semantisch äquivalentes Programm einer Zielsprache (**Zielprogramm**) umwandelt.

Üblicherweise handelt es sich dabei um die Übersetzung eines Quelltextes in direkt auf einem Rechner ausführbares Programm in Maschinensprache.

Das Resultat ist also ein nur auf einer spezifischen Rechnerarchitektur lauffähiges Programm. Vorteile liegen vor allem in der **Ausführungsgeschwindigkeit** der Programme.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

23

23

Interpreter



Interpreter **lesen** und **analysieren** den Quellcode eines Programmes und **führen** dann die entsprechenden **Aktionen aus**.

Dies ist im Vergleich zu Compilersprachen, bei denen das Programm vor seiner Ausführung in Maschinencode übersetzt wird, der dann vom Prozessor direkt ausgeführt wird, sehr **zeitaufwändig**.

Der Vorteil liegt darin, dass interpretierte Programmiersprachen auf jeder Rechnerarchitektur lauffähig sind, sofern es Interpreter für die Rechnerarchitektur gibt (**Portabilität**).

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

24

24

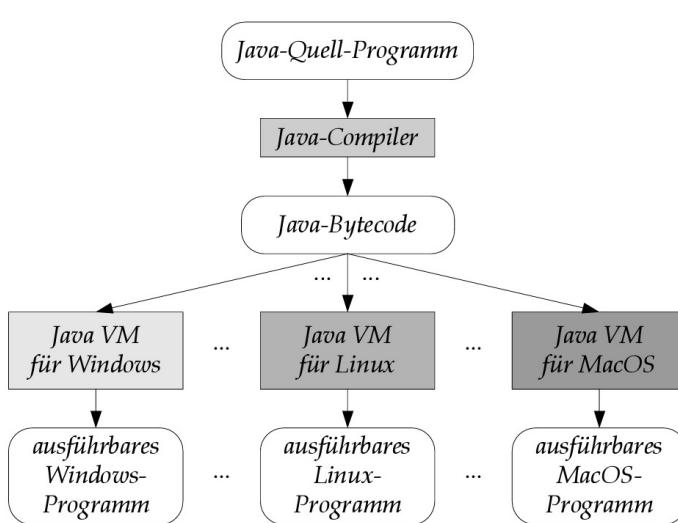
Byte Code

Bytecode ist eine Sammlung von Befehlen für eine **virtuelle Maschine**.

Bei Kompilierung eines Quelltextes mancher Programmiersprachen – wie beispielsweise **Java** – wird nicht direkt Maschinencode, sondern ein **Zwischencode**, der Bytecode, erstellt.

Dieser Code ist in der Regel unabhängig von realer Hardware und im Vergleich zum Quelltext oft relativ kompakt. Dieser Ansatz verbindet Vorteile von Compilern (**Geschwindigkeit**) und Interpretern (**Portabilität**) in einem **Mittelweg**.

JAVAs Virtuelle Maschine (JVM)



Einordnung der Sprache JAVA



Laufzeitmodell

Programmierparadigma	Interpreter	Compiler	Byte-Code
Imperativ	<i>z.B. BASIC</i>	<i>z.B. C</i>	<i>z.B. Python</i>
Deklarativ	<i>z.B. Prolog</i>		<i>z.B. Python (funktionale Anteile)</i>
Objekt-orientiert		<i>z.B. C++</i>	JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

27

27

Der JAVA Compile-Build-Run Zyklus



Java

- Der Compiler (**javac**) erzeugt **.class** Dateien, die in einer **Java Virtual Machine** (JVM, **java**) ausgeführt werden.
- Es gibt keinen Link-Lauf. Die **.class** Files werden **zur Laufzeit gebunden**.
- Die **.class** Dateien können auf unterschiedlichen Plattformen mit unterschiedlichen Compilern erzeugt werden.
- Die **.class** Dateien lassen sich in allen JVM ausführen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

28

28

Das allererste Programm

Aller Anfang ist schwer neu



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

Semikolon am Ende einer Zeile kennzeichnet eine **Anweisung**. Die JVM soll das was vor dem Semikolon steht ausführen. Mehrere Anweisungen werden sequentiell abgearbeitet.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

29

29

Das allererste Programm

Aller Anfang ist schwer neu



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

Ausdruck bezeichnet einen Term (Werte die über **Operatoren** verknüpft werden). Ein Ausdruck kann komplex sein und Variablen sowie Methodenaufrufe beinhalten.

Ein Ausdruck wird immer zu einem **Wert** durch die JVM ausgewertet. Hier: „3 + 4“ wird zu dem Wert sieben ausgewertet.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

30

30

Das allererste Programm

Aller Anfang ist **schwer** neu



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

Zuweisung werden durch ein `=` notiert. Eine Zuweisung soll den Wert eines Ausdrucks einer Variablen zuweisen.

`=` hat in JAVA also nicht die Bedeutung der mathematischen Gleichheit. `=` prüft nicht ob zwei Werte gleich sind. Soll die mathematische Gleichheit verglichen werden, muss der Gleichheitsoperator genutzt werden.

`i = 3` bedeutet also: Weise den Wert 3 der Variablen `i` zu.

`i == 3` bedeutet also: Prüfe ob die Variable `i` den Wert 3 hat

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

31

31

Das allererste Programm

Aller Anfang ist **schwer** neu



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

Variablen-deklarationen dienen als spezielle Form der Anweisung dazu einen Bereich im Hauptspeicher anzulegen und mittels eines Bezeichners zu benennen.

In JAVA (statisch typisierte Programmiersprache) muss hierzu für jede Variable ein **Datentyp** festgelegt werden. Datentyp `int` steht dabei für Integer (d.h. ganzzahlige positive und negative Werte). Der Variablen `i` können also bspw. die Werte -1, 2, 1000 und -7451 zugewiesen werden, aber nicht 0.451 (kein ganzzahliger Wert).

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

32

32

Das allererste Programm

Aller Anfang ist schwer neu



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

Bildschirmausgaben erfolgen in JAVA mittels einer Methode (Unterprogramm oder auch Routine genannt). Methoden kapseln Funktionalitäten, die man wieder und wieder benötigt. Auch ein Methodenaufruf ist eine Anweisung.

Die **println** Methode gibt einen Wert als Zeichenkette auf der Konsole aus und lässt die nächste Ausgabe in der folgenden Zeile beginnen. Sollen zwei Zeichenketten ausgegeben werden, ohne dass diese durch einen Zeilenumbruch voneinander getrennt werden, kann man die **print** Methode nutzen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

33

33

Das allererste Programm

Aller Anfang ist schwer neu



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

Blöcken dienen in Programmiersprachen der Strukturierung von Quelltexten. Sie beginnen in JAVA mit einer geschweiften Klammer **{** und enden mit einer geschweiften Klammer **}**.

Blöcke können ineinander **geschachtelt** sein, wie wir im vorliegenden Beispiel sehen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

34

34

Tabs vs Spaces



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

35

35

Das allererste Programm Aller Anfang ist schwer neu



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

Klassenblöcke. Klassen sind in JAVA eine der wichtigsten Struktureinheiten. Jedes Programm in JAVA besteht mindestens aus einer Klasse.

Vor der öffnenden Klammer steht der Name der Klasse eingeleitet mit dem Schlüsselwort **public class**. Innerhalb des Klassenblocks werden die Bestandteile der Klasse notiert (insb. Datenfelder und Methoden wie wir noch sehen werden). Eine Klasse muss gem. Konvention immer in einer Datei gespeichert werden, die denselben Namen trägt wie die Klasse (in unserem Fall als `Berechnung.java`).

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

36

36

Das allererste Programm

Aller Anfang ist schwer neu



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

Hauptmethode. Innerhalb von Klassen gibt es untergeordnete Struktureinheiten – sogenannte Methoden.

Jede Klasse, die ein ausführbares Programm (also nicht einfach nur Hilfsfunktionen ausführen soll) muss in JAVA eine sogenannte **main** Methode besitzen. Die **main** Methode muss immer so gestaltet sein, wie oben angegeben (im weiteren Verlauf der Vorlesung werden Sie die Bedeutung dieser „kryptischen“ Zeichenfolge verstehen lernen). Innerhalb der **main** Methode (also im Block der Methode) können Sie Ihrer Kreativität freien Lauf lassen, so lange Sie der Syntax von JAVA folgen und berechenbare Funktionalitäten implementieren.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

37

37

Grundstruktur eines JAVA Programms



```
// Klassen- bzw. Programmbeginn  
public class HalloWelt {  
    // Beginn des Hauptprogramms  
    public static void main(String[] args) {  
        // HIER STEHT EINMAL DAS PROGRAMM...  
    } // Ende des Hauptprogramms  
} // Ende des Programms
```

(1) Jedes JAVA Programm besteht aus mindestens einer Klasse und einer **main** Methode.

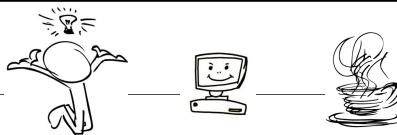
(2) Jedes JAVA Programm beginnt seine sequentielle Abarbeitung in der ersten Zeile der **main** Methode.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

38

38

Mini-Übung:



```
1 public class Uebung {  
2     public static void main(String[] args) {  
3         System.out.println("Guten Tag!");  
4         System.out.println("Mein Name ist Puter, Komm-Puter.");  
5     }  
6 }
```

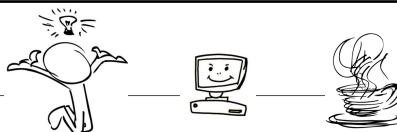
- (1) Was passiert, wenn Sie in Zeile 3 das Semikolon entfernen?
- (2) Warum passiert es?
- (3) Was passiert, wenn Sie statt einem zwei Semikolons einfügen?
- (4) Warum passiert es?

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

39

39

Mini-Übung:



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

Schreiben Sie oben stehendes Programm so um, dass

- (1) Die Variable *i* initial den Wert 32 erhält,
- (2) Der Wert von *i* durch eine Anweisung halbiert wird.
- (3) Diese Anweisung soll insgesamt dreimal ausgeführt werden.
- (4) Nach jeder Halbierung von *i*, soll der Wert von *i* in folgender Form auf der Konsole ausgegeben werden.

Der Wert von *i* beträgt 8.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

40

40

Zusammenfassung



- Was bedeutet Programmieren?
 - Welche Programmierparadigmen gibt es?
 - Welchem Programmierparadigma folgt JAVA?
 - Welche Laufzeitmodelle gibt es?
 - Welchem Laufzeitmodell folgt JAVA?
-
- Grundlegende Programmelemente
 - Anweisung
 - Ausdruck
 - Zuweisung
 - Methode
 - Block
 - Grundstruktur eines JAVA Programms



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

41

41

In dieser Unit



Einleitung

- Was ist Programmieren?
- Programmierparadigmen
- Laufzeitmodelle von Programmiersprachen
- Grundlegende Begrifflichkeiten bei Programmiersprachen (am Bsp. von Java)

Etwas mehr Java Syntax

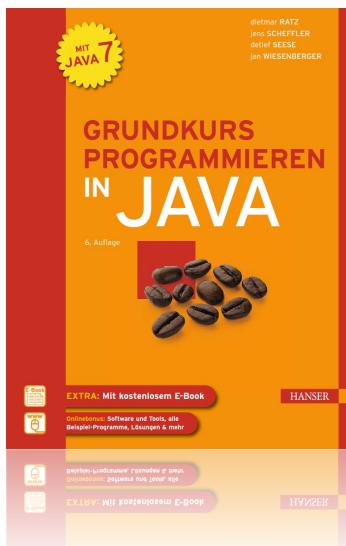
- Weitere Begrifflichkeiten bei Programmiersprachen
- Eingaben von der Konsole einlesen
- Ausgaben auf der Konsole ausgeben

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

42

42

Zum Nachlesen ...



Kapitel 4

Grundlagen der Programmierung in JAVA

Abschnitt 4.1

Grundelemente eines JAVA Programms

Abschnitt 4.2

Erste Schritte in JAVA

Abschnitt 19.3.5.2

Konsoleneingabe über ein Scanner-Objekt

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

43

43

Worum geht es jetzt?



Grundelemente

- Kommentare
- Bezeichner
- Literale
- Reservierte Wörter, Schlüsselwörter
- Trennzeichen
- Operatorsymbole
- `import`
Anweisung

Erstes Programmieren

- Ausgaben auf die Konsole
- Eingaben von der Konsole

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

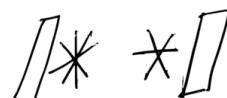
44

44

Kommentare



- Kommentare dienen dazu, die Funktionsweise oder Struktur eines Quelltextes zu beschreiben
- **Kommentare werden vom Compiler ignoriert**
- Kommentare dienen ausschließlich dem Verständnis
- In JAVA sind
 - **einzeilige** und
 - **mehrzeilige** Kommentare
 - sowie **JavaDoc Kommentare** bekannt
- JavaDoc ist ein Dokumentationsgenerator, der aus Quelltextkommentaren eine HTML Programmdokumentation erzeugt.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

45

45

Kommentarbeispiele



Einzeiliger Kommentar

```
a = b + c; // hier beginnt ein Kommentar
```

Mehrere einzeilige Kommentare

```
// Zeile 1  
// Zeile 2  
// ...  
// Zeile n
```

Ein mehrzeiliger Kommentar

```
/* Kommentar...  
Kommentar...  
immer noch Kommentar...  
letzte Kommentarzeile...  
*/
```

Ein JavaDoc Kommentar (speziell formatierter mehrzeiliger Kommentar)

```
/**  
 * Dieses Programm berechnet die Lottozahlen von naechster  
 * Woche. Dabei erreicht es im Schnitt eine Genauigkeit  
 * von 99,5%.  
 *  
 * @author Hans Mustermann  
 * @version 1.0  
 */
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

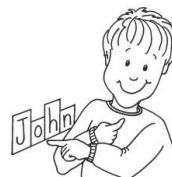
46

46

Bezeichner und Namen



- In Programmen müssen diverse Elemente benannt werden, damit diese ansprechbar sind.
- Hierzu sehen alle Programmiersprachen Bezeichnungsregeln vor. JAVA kennt die folgenden:
 - Ein Name kann aus Buchstaben a, b, c, ..., x, y, z, A, B, C, ..., X, Y, Z (keine sonstigen Sonderzeichen)
 - dem Unterstrich _
 - dem Dollarzeichen \$
 - und den Ziffern 0, 1, 2, ... 9 zusammengesetzt werden.
- Ein Bezeichner darf nicht mit einer Ziffer beginnen.
- Ein Bezeichner darf nicht identisch mit einem Schlüsselwort sein.



“There are only two hard things in computer science.
Cache invalidation and naming things.”

PHIL KARLTON

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

47

47

Beispiele für gültige und ungültige Bezeichner in JAVA



Gültige Bezeichner

- Hallo_Welt
- _H_A_L_L_O_
- hallo123
- hallo_123

Ungültige Bezeichner

- 101Dalmatiner
- Das_war's
- Hallo Welt
- class

Hinweis: JAVA unterscheidet Groß- und Kleinschreibung bei Bezeichnern!!!

Eine durch `hallo_123` bezeichnete Variable ist also nicht identisch mit einer durch `Hallo_123` bezeichneten Variablen. Für den Compiler sind dies absolut unterschiedliche Dinge!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

48

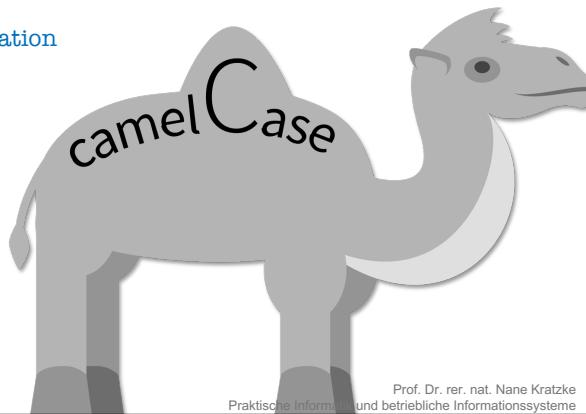
48

(lower/Upper) camelCaseNotation für Bezeichner



Wir nutzen in diesem Modul den Java „Default-Style“:

- `lowerCamelCaseNotation`
für Variablen und Methodenparameter
- `UpperCamelCaseNotation`
für Klassen



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

49

49

Literale



Ein **Literal** beschreibt in einer Programmiersprache einen konstanten Wert, der sich innerhalb eines Programms nicht ändern kann. Literale werden genutzt, um Werte in Quelltexten auszudrücken.

In JAVA treten folgende Arten von Literalen auf:

- **Ganze Zahlen:** z.B. 23 oder -166
- **Gleitkommazahlen:** z.B. 3.14
- **Wahrheitswerte:** `true` oder `false`
- **Einzelzeichen:** z.B. 'a'
- **Zeichenketten:** "Hello World"
- **Null-Literal für Referenzen:** `null`



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

50

50

Reservierte Wörter, Schlüsselwörter



In JAVA haben einige Wörter (z.B. die Literalkonstanten `true` und `false`) eine spezifische Bedeutung. Die folgenden so genannten Wortsymbole haben in JAVA eine besondere Bedeutung und dürfen daher nicht als Bezeichner genutzt werden. Die meisten von diesen **Schlüsselwörtern** werden Sie im weiteren Verlauf der Vorlesung noch kennen lernen.

<code>abstract</code>	<code>assert</code>	<code>boolean</code>	<code>break</code>	<code>byte</code>
<code>case</code>	<code>catch</code>	<code>char</code>	<code>class</code>	<code>const</code>
<code>continue</code>	<code>default</code>	<code>do</code>	<code>double</code>	<code>else</code>
<code>enum</code>	<code>extends</code>	<code>final</code>	<code>finally</code>	<code>float</code>
<code>for</code>	<code>goto</code>	<code>if</code>	<code>implements</code>	<code>import</code>
<code>instanceof</code>	<code>int</code>	<code>interface</code>	<code>long</code>	<code>native</code>
<code>new</code>	<code>package</code>	<code>private</code>	<code>protected</code>	<code>public</code>
<code>return</code>	<code>short</code>	<code>static</code>	<code>strictfp</code>	<code>super</code>
<code>switch</code>	<code>synchronized</code>	<code>this</code>	<code>throw</code>	<code>throws</code>
<code>transient</code>	<code>try</code>	<code>void</code>	<code>volatile</code>	<code>while</code>

Quelle: Grundkurs Programmieren in Java

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

51

51

Trennzeichen



Ein Compiler muss in der Lage sein, einzelne Bezeichner, Schlüsselwörter und Literale von einander zu trennen. Dies wird in JAVA durch die folgenden Trennzeichen ermöglicht.

- Leerzeichen
- Zeilenendezeichen (ENTER)
- Tabulatorzeichen (TAB)
- Kommentare
- Operatoren (wie z.B. `+`, `*`, `-`, `/`)
- Interpunktionszeichen `.`, `,`, `;`, `{`, `}`, `[]`

^
::
Ö

:-)

Unmittelbar aufeinanderfolgende Bezeichner, Schlüsselwörter oder Literale müssen durch eines der obigen Symbole voneinander getrennt werden, um sie als eigenständiges Element zu erkennen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

52

52

Operatorsymbole



Operatoren sind spezielle Symbole, die dazu dienen, jeweils bis zu drei unterschiedliche Werte (Operanden) zu einem neuen Wert zu verknüpfen. Nahezu alle Programmiersprachen (so auch JAVA) unterscheiden die folgenden Arten von Operatoren

$$\begin{aligned} r &< s + t \\ s &< r + t \\ t &< r + s \end{aligned}$$

- **Einwertige Operatoren** (monadische Operatoren) mit nur einem Operanden, z.B. die Inkrement und Dekrement-Operatoren ++ und --
- **Zweiwertige Operatoren** (dyadische Operatoren) mit zwei Operanden, z.B. die bekannten Operatoren +, -, * und /
- **Dreiwertige Operatoren** (triadische Operatoren) mit drei Operanden. JAVA kennt hier nur den ?: Operator (bedingte Auswertung).

Alle diese Operatoren werden Sie in UNIT 2 noch im Detail kennenlernen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

53

53

import Anweisung



Viele Dinge, die in JAVA benötigt werden, sind nicht Bestandteil des Sprachkerns, sondern müssen bei Bedarf dazugeladen werden. Man macht dies, um Programme möglichst klein zu halten.

Nachzuladende Funktionen müssen dem Compiler bekannt gemacht werden, indem sie **importiert** werden. Hierzu wird eine sogenannte **import** Anweisung verwendet.

In UNIT 3 und 4 werden Sie beispielsweise eine Reihe von Datenstrukturen kennenlernen (Streams, Listen, Stacks und Maps), die erst importiert werden müssen, bevor sie für die Programmierung genutzt werden können.

Dies erfolgt mit einem Aufruf der folgenden Art:

```
import java.util.List;  
import java.util.Stack;  
import java.util.Map;
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

54

54

Inhalte dieser UNIT



Grundelemente

- Kommentare
- Bezeichner
- Literale
- Reservierte Wörter, Schlüsselwörter
- Trennzeichen
- Operatorsymbole
- `import`
Anweisung

Erstes Programmieren

- Ausgaben auf die Konsole
- Eingaben von der Konsole

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

55

55

Ausgaben auf der Konsole



Sie wissen bereits, dass man Ausgaben auf der Konsole mittels einer sogenannten `println` Methode (ein Unterprogramm) in folgendem Stil vornehmen kann:

```
System.out.println("Hello World");
```

Sie können jedoch auch zusammengesetzte Werte ausgeben:

```
System.out.println("Hello" + " " + "World");
```

Oder auch komplexere Zeichenketten erzeugen und dabei Berechnungsergebnisse ausgeben lassen:

```
int i = 4;  
int j = 7;  
System.out.println("Die Addition von " + i + " und " + j +  
" ergibt " + (i + j) + ".");
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

56

56

Formatieren von Ausgaben I



Häufig müssen Ausgaben für eine Ausgabe auf der Konsole vorbereitet (formatiert) werden. Dafür kann man die `String.format()` Methode nutzen.

```
String s = String.format("Hallo %s!", "Walter");  
System.out.println(s);
```

```
Haloo Walter!
```

Formatstring

Platzhalter

Wert

In eine Zeichenkette werden an Platzhalterpositionen Werte eingesetzt.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

57

57

Formatieren von Ausgaben II



Abkürzung: Die `printf()` Methode gibt direkt auf der Konsole aus. Die Formatstrings sind genauso wie bei `String.format()`.

```
System.out.printf("Hallo %s!", "Walter");  
Haloo Walter!
```

Wir können auch mehrere Werte unterschiedlichen (oder gleichen) Typs in einen Formatstring einsetzen.

```
String s = String.format("Die %s ist %d.", "Antwort", 42);  
System.out.println(s);  
Die Antwort ist 42.
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

58

58

Formatieren von Ausgaben III

Links- und rechtsbündige Ausgabe von Strings



Wir können links- und rechtsbündige Ausgaben und der Längen formatieren.

```
String s = String.format("Hallo %6s!", "Walter");  
System.out.println(s);
```

Hallo Walter!

Rechtsbündige Ausgabe mit 6 Zeichen.

```
String s = String.format("Hallo %6s!", "Egon");  
System.out.println(s);
```

Hallo __Egon!

Linksbündige Ausgabe mit 6 Zeichen.

```
String s = String.format("Hallo %-6s!", "Egon");  
System.out.println(s);
```

Hallo Egon__!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

59

59

Formatieren von Ausgaben IV

mit ganzen Zahlen als Werten



Rechtsbündige Ausgabe mit führenden Leerzeichen

```
s = String.format("Aller guten Dinge sind %3d!", 3);  
System.out.println(s);
```

Aller guten Dinge sind __3!

Rechtsbündige Ausgabe mit führenden Nullen (Zeropadding)

```
s = String.format("Aller guten Dinge sind %03d!", 3);  
System.out.println(s);
```

Aller guten Dinge sind 003!

Linksbündige Ausgabe mit drei Stellen

```
s = String.format("Aller guten Dinge sind %-3d!", 3);  
System.out.println(s);
```

Aller guten Dinge sind 3__!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

60

60

Formatieren von Ausgaben V mit Fließkommazahlen als Werten



Ausgabe mit zwei Nachkommastellen

```
s = String.format("Pi ist %.2f", Math.PI);
System.out.println(s);
Pi ist 3.14
```

Ausgabe mit zwei Vorkomma- und fünf Nachkommastellen

```
s = String.format("Pi ist %8.5f", 3);
System.out.println(s);
Pi ist 3.14159
```

Ausgabe mit zwei Vorkomma- (Zeropadding) und fünf Nachkommastellen

```
s = String.format("Pi ist %07.4f", 3);
System.out.println(s);
Pi ist 03.1416
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

61

61

Format Cheat-Sheet



[flags]	Bedeutung
-	Linksausrichtung
+	Ausgabe wird immer ein Vorzeichen beinhalten
0	Zeropadding (führende Nullen)
(Negative Zahlen werden in () geschrieben

Minimale Breite der Ausgabe in Zeichen

%[flags][width].precision]conversion

conversion	Bedeutung
d	Dezimalausgabe eines ganzzahligen Werts
f	Dezimalausgabe eines Fließkommawertes
s	Zeichenkettenausgabe
x	Hexadezimalausgabe eines ganzzahligen Werts
tT	24 Stunden Angabe eines Date-Obj. (hour:minute:seconds)
tD	Jahresangabe eines Date-Obj. (year-month-day)

Anzahl an Nachkommastellen bei Fließkommawerten

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

62

62

Eingaben von der Konsole (I)



Mittels `println` können Sie in JAVA Ausgaben auf der Konsole veranlassen. Aber wie können Sie Daten von einem Benutzer einlesen? Sinnvoll wäre es, wenn Java eine `readln` als Pendant zur `println` Methode hätte. Prinzipiell hat Java dies, jedoch etwas „versteckt“. Sie müssen sich eine solche Lesefunktion nämlich erst aus mehreren Einzeltreilen zusammenbauen, die sich Ihnen alle erst im weiteren Verlauf der Vorlesung vollständig erschließen werden.

```
import java.util.Scanner;  
...  
System.out.print("Ihr Name: ");  
Scanner in = new Scanner(System.in); // Erz. eines „Leseobjekts“  
String eingabe = in.nextLine(); // Einlesen von Konsole  
System.out.println("Hello " + eingabe);
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

63

63

Eingaben von der Konsole (II)



Mittels `nextLine` können Sie Zeichenketten einlesen.

```
String eingabe = in.nextLine();
```

Mittels `nextInt` können Sie ganzzahlige Zahlen einlesen.

```
int ganzzahl = in.nextInt();
```

Mittels `nextFloat` können Sie Fließkommazahlen einlesen.

```
float kommazahl = in.nextFloat();
```

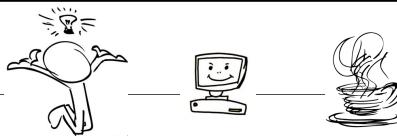
Programmiersprachen verarbeiten unterschiedliche Datentypen, wie Sie noch in UNIT 2 sehen werden. Dies müssen Sie in Java (da statisch typisierte Programmiersprache) bei der Auswahl der entsprechenden Lesemethoden berücksichtigen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

64

64

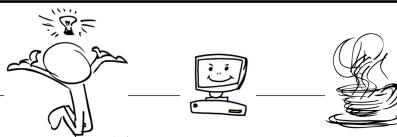
Mini-Übung:



```
1 public class Uebung {  
2     public static void main(String[] args) {  
3         System.out.println("Guten Tag!");  
4         System.out.println("Mein Name ist Puter, Komm-Puter.");  
5     }  
6 }
```

- (1) Markieren Sie alle Schlüsselwörter.
- (2) Markieren Sie alle Bezeichner.
- (3) Markieren Sie alle Literale.

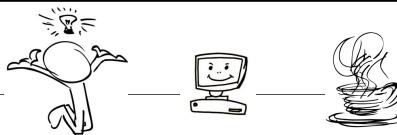
Mini-Übung:



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

- (1) Markieren Sie alle Bezeichner innerhalb des main Blocks.
- (2) Markieren Sie alle Literale.
- (3) Markieren Sie alle Operatoren.
- (4) Markieren Sie alle Ausdrücke.

Mini-Übung:



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

Schreiben Sie oben stehendes Programm so um, dass

- (1) Die Variable i durch den Nutzer eingegeben werden kann.
- (2) Die Variable j durch den Nutzer eingegeben werden kann.
- (3) Die Eingaben addiert werden und das Ergebnis in folgender Form ausgegeben wird:

Die Addition von 5 und 7 ergibt 12.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

67

67

Zusammenfassung



- **Grundlegende Programmelemente**

- Kommentare
- Bezeichner
- Trennzeichen,
- Schlüsselwörter
- Operatoren
- import Anweisung



- **Grundlegende Programmierung**

- Ausgaben auf der Konsole
- Eingaben von der Konsole



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

68

68

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad



Kapitel 2

- W2.1: Mein erstes Programm
- W2.2: Weihnachtsbaum

Nur eingeschränkt zu empfehlen

- *W2.3 bis W2.9*
- *Processing Framework erforderlich.*
- *Framework wird nicht in diesem Modul behandelt*

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

69