Vorlesung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Programmieren I und II

Unit 1

Einleitung und Grundbegriffe der Programmierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

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

- **Raum: 17-0.10**
- **Tel.:** 0451 300 5549
- **Email: kratzke@fh-luebeck.de**

@NaneKratzke
Updates der Handouts auch über Twitter #prog_inf und #prog_itd

Units

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

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
2. Semester	Unit 9 Generische Datentypen	Unit 10 Objektorientierter Entwurf und objektorientierte Designprinzipien	Unit 11 Graphical User Interfaces	Unit 12 Multithread Programmierung

Abgedeckte Ziele dieser UNIT

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (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 Mulithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

In dieser Unit

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Einleitung

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

Etwas mehr Java Syntax

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

Zum Nachlesen ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Kapitel 1

Einleitung

Kapitel 2

Grundbegriffe aus der Welt des Programmierens

Kapitel 3

Aller Anfang ist schwer

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)

Programmiersprachen im Vergleich

Maschinensprache

```
01110110
11100101
011001
010111
010010
010001
010011
```

Assembler

```
LD   RG1 23
MOV  RG7 RG2
ADD  RG2 RG1
ADD
LD
DIV
MOV
```

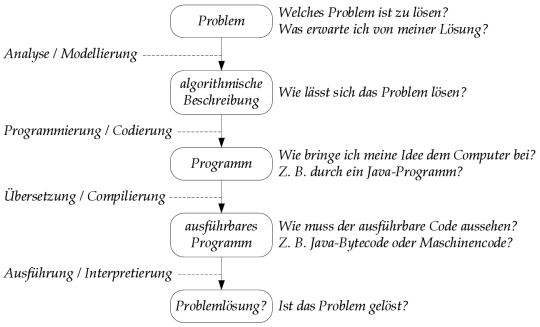
Frühe, problemorientierte Programmiersprache

```
10 PRINT "HALLO"
20 SET A = 7
30 GOSUB
40 PRINT
50 GOSUB
60 GOTO
```

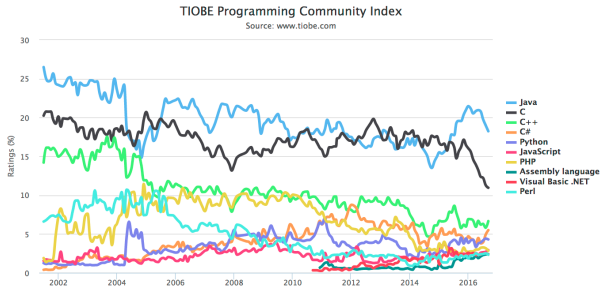
Java

```
public class HelloWorld {
  public static void main(String[] args){
    System.out.println("Hallo!");
  }
}
```


Was bedeutet Programmieren?



Verbreitung von Programmiersprachen

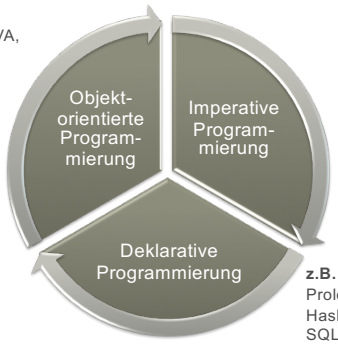


Quelle: http://www.tiobe.com,

Stand: 22. September 2016

Die drei großen Programmierparadigmen

z.B.
Smalltalk, JAVA, C++, C#, ADA-95, Eiffel



z.B.
C, PASCAL, COBOL, FORTRAN, Assembler

z.B.
Prolog (logisch), Haskell (funktional), SQL (relational)

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.

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).

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.

Die drei gängigen Laufzeitmodelle von Programmiersprachen

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.

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**).

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.

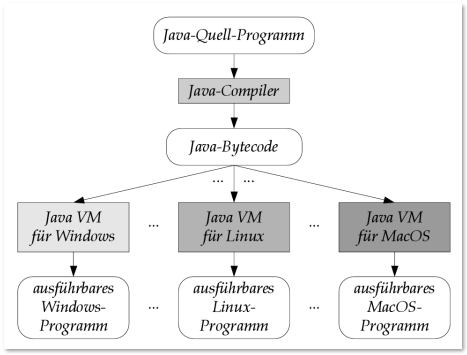
Dieser Code ist in der Regel unabhängig von realer Hardware und im Vergleich zum Quelltext oft relativ kompakt. Dieser Ansatz verbindet Vorteile von Compilern (**Geschwindigkeit**) und Interpretern (**Portabilität**) in einem **Mittelweg**.

Einordnung der Sprache JAVA

Laufzeitmodell

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

JAVAs Virtuelle Maschine (JVM)



Der JAVA Compile-Build-Run Zyklus

- 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.

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.

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.

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

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  }
```

Variablendeklarationen 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 zugeweisen werden, aber nicht 0.451 (kein ganzzahliger Wert).

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.

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.

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`).

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.

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.

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?

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.
```


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

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 Programmier-sprachen
- Eingaben von der Konsole einlesen
- Ausgaben auf der Konsole ausgeben

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

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

Kommentare

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

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 99,58.
 *
 * @author Hans Mustermann
 * @version 1.0
 */
```


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.

Beispiele für gültige und ungültige Bezeichner in JAVA

Gültige Bezeichner	Ungültige Bezeichner
• Hallo_Welt	• 101Dalmatiner
• _H_A_L_L_O_	• Das_war's
• hallo123	• Hallo Welt
• hallo_123	• 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!

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`

Reservierte Wörter, Schlüsselwörter

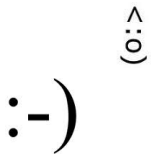
In JAVA haben einige Wörter (z.B. die Literalkonstanten `true` und `false`) eine spezifische Bedeutung. Die folgenden so genannten Wortsymbole haben in JAVA eine besondere Bedeutung und dürfen daher nicht als Bezeichner genutzt werden. Die meisten von diesen **Schlüsselwörtern** werden Sie im weiteren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

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

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 . , ; () {} []



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

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

- **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.

`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;
```


Inhalte dieser UNIT

Grundelemente

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

Erstes Programmieren

- Ausgaben auf die Konsole
- Eingaben von der Konsole

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) + ".");
```


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 Einzelteilen 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);
```

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.

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.
```


Zusammenfassung

- **Grundlegende Programmelemente**
  - Kommentare
  - Bezeichner
  - Trennzeichen,
  - Schlüsselwörter
  - Operatoren
  - `import` Anweisung

- **Grundlegende Programmierung**
  - Ausgaben auf der Konsole
  - Eingaben von der Konsole