Interreg

EUROPÄISCHE UNION

Österreich-Tschechische Republik

Europäischer Fonds für regionale Entwicklung

INFORMATIK

Einführung in die Programmierung mit Java

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

VŠTE

EUROPÄISCHE UNION

Inhalt

1. ERSTELLEN VON VARIABLEN

Werte werden in Variablen abgespeichert. Eine Variable ist eine mit einer Bezeichnung versehene Stelle in einem Speicher. Im Rahmen der Variablenerstellung ist dies ihre Deklaration. Für jede Variable in der Programmiersprache JAVA muss zuerst festgelegt werden, welcher Datentyp darin enthalten sein soll (in diesem Fall, siehe Beispiel 1 – int, boolean, char). Ihre Bedeutung ist im nächsten Absatz erklärt.

Im Anschluss daran werden für die Variablen Bezeichnungen ohne diakritische Zeichen oder Leer-zeichen vergeben. Die Leerräume zwischen den Wörtern werden einfach ausgelassen, jedes andere Wort beginnt mit einem Großbuchstaben und der Begriff wird mit einem Semikolon abgeschlossen.

Beispiel 1

```
int a;
boolean IsTrue;
char CarConsumption;
```

In Beispiel 1 wurden Variablen erstellt, welchen jedoch noch kein Wert zugewiesen wurde. In Beispiel 2 hingegen wurden Variablen erstellt, welche bestimmten Werte zugewiesen wurden. Die erste Zuordnung von Werten wird Initialisierung genannt.

Beispiel 2

```
int a=2;
boolean IsTrue =true;
char CarConsumption ='A';
```

Möchte man mehrere Variablen gleichzeitig deklarieren, muss man diese durch ein Komma trennen:

```
int a, b;
```

Es wird zwischen einem Ausdruck und einem Befehl unterschieden. Ein Ausdruck hat immer einen Wert, welcher durch die Auswertung des Ausdrucks gewonnen wird. Somit sind z.B. 42 und x+1 Aus-drücke. Ein Befehl ist ein Code, der etwas macht, z.B. X=1; hierbei

handelt es sich um einen Befehl, welcher der Variable x den Wert 1 zuordnet. Wenn man einen Ausdruck hat, der etwas evaluiert, kann dieser in der Regel durch das Anhängen eines Semikolons befohlen werden. In diesem Fall sagt man, dass die Evaluation des Ausdrucks einen Nebeneffekt hat.

2. DATENTYPEN

2.1. Integer

Int: Man kann nur ganze Zahlen eingeben, welche in einem Spektrum zwischen -2,147,483,648 und 2,147,483,647 liegen, inklusive Null. Ein Integer kann z.B. der Anzahl von Computeroperationen pro Sekunde entsprechen.

Andere Integer-Datentypen sind in der Tabelle aufgelistet:

Art	**Bytes**	**Spektrum**	
byte	1	-128	127
short	2	-32 768	32 767
int	4	-2147 483 648	2147 483 647
long	8	-9,22 x 1018	9,22 x 1018

Liegt das Ergebnis einer Operation nicht innerhalb des erlaubten Intervalls eines bestimmten Datentyps, tritt ein Überlauf auf. In diesem Fall ist das Ergebnis das Gegenteil des Ergebnisses der mathematischen Operation. Es führt zwar zu keinem Fehler in JAVA, aber es sollte bedacht wer-den, z.B. wenn man einer Byte-Variable den Wert 127 zugewiesen hat und die Zahl 1 addiert, wird die Variable den Wert -128 erhalten.

2.2. Echte Zahlen

Um mit echten Zahlen zu arbeiten, verfügt Java über zwei primitive Datentypen: Float und Double. Beide verwenden dieselbe Darstellungsmethode: Die echte Zahl wird als ein Dreifachzeichen, Mantisse und Hochzahl gespeichert, weshalb Zahlen nur ungefähr abgespeichert werden können. Der Typ Float verwendet 32 Bit: ein Bit deckt ein Zeichen ab, acht Bit die Hochzahl und 23 Bit die Mantisse. Double verwendet 64 Bit: ein Bit für das Zeichen, elf für die Hochzahl und 52 für die Mantisse.

2.3. Umwandlung

Darunter versteht man die Umwandlung eines Werts in einen anderen Datentyp, zB die Umwandlung eines Double in einen Integer. Manche Umwandlungen werden automatisch vorgenommen, z.B. von einem Integer in ein Double, andere müssen erst beauftragt werden, z.B. jene vom Typ Long in einen Typ int. Nachfolgend wird gezeigt, wie man eine automatische Umwandlung festlegen kann:

```
int a = 100;
long a = b;  // Automatic conversion from int to long
```

Wenn eine Umwandlung benötigt, wird der Zieldatentyp in die Klammer vor dem umgewandelten Wert geschrieben. So sieht z.B. eine Umwandlung von Double in einen Integer wie folgt aus:

```
double d1 = 5.85;
int i1 = (int) d1;
```

Das Umwandlungsergebnis wird der Wert sein, welcher dem Eintrag der ursprünglichen Zahl vor dem Dezimalpunkt entspricht (bei nicht-negativen Werten ist dies der gesamte Teil der Zahl), sprich für Variable i2 wird der Wert 5 betragen.

```
double d2 = -4.99;
int i2 = (int) d2;
```

Für Variable i2 ist der Wert -4. Man muss sich bewusst sein, dass dies nicht der gesamte Teil der Zahl ist, da der gesamte Teil der Zahl -4,99 die Zahl -5 ist.

2.4. Logische

Boolean: Enthält nur die Ausdrücke “wahr“ oder „falsch“. Er wird auch logische Variable genannt. Vergleicht man zwei numerische Variablen in a<b, sollen sie a und b heißen, und wenn a tatsächlich kleiner als b ist, ist der Ausdruck a<b wahr und „wahr“ wird festgelegt und so wird auch a zur Variable c definiert. Befindet sich „wahr“ nicht in c, wird „falsch“ darin gespeichert.

```
int a = 5;
int b = 6;
boolean c = a<b;
System.out.println(c);
```

- In diesem Beispiel schreibt das System “wahr”;
- „Falsch“ wird intern als 0 dargestellt, der Wert „wahr“ als 1.

2.5. Zeichen

Char: Darin lässt sich ein Zeichen speichern. Vor und nach diesem Zeichen muss ein Apostroph (ein einfaches Anführungszeichen) stehen. Es lassen sich Zeichen aus der Unicode-Tabelle drucken. Der char-Wert kann als der Zeichenindex in der Unicode-Tabelle verstanden werden.

```
char c ='H';
System.out.println(c) ;
```

2.6. Zeichenketten

Man verwendet string um mit Zeichenketten zu arbeiten. Die Konstanten der Kette werden in Anführungszeichen geschrieben.

```
String s = " Hi how are you?";
```

Ketten können durch den Operator + miteinander verbunden werden.

```
String s1 = "jdk", s2 = "7.0";
String s3 = s1 + s2;  // Creates a string "jdk7.0"
```

Der Zeichenkette lässt sich noch ein weiterer Wert hinzufügen. In diesem Fall wird der Wert zuerst in eine Zeichenkette umgewandelt und anschließend werden die Zeichenketten zusammengeführt.

```
int x = 42;
String s = "answer is " + x;
```

Das obere Beispiel erzeugt diese Zeichenkette: "answer is 42"

2.7. Kommentare

Es gibt in JAVA drei Arten von Kommentaren

- einzeilige – beginnen mit // und gehen bis an das Ende der Zeile
- mehrzeilige – beginnen mit / * und werden mit den Zeichen * / beendet
- Dokumentation – beginnt mit den Zeichen / ** und endet mit den Zeichen * /

Dokumentationskommentare sind für die Verarbeitung durch Java.doc vorgesehen, welches Dokumentationen im HTML-Format erzeugt.

```
/**
   * Main program class.
   */

   public class Main {
         public static void main( String[] args ) {
               /* Prints the answer */
               System.out.println( 42 ); // answer is 42
         }
   }
```

Mithilfe von Kommentaren werden im Quelltext zusätzliche Informationen ergänzt. Der Transporter überspringt Kommentare, weshalb sie keine Auswirkungen auf die Programmausführung haben. Kommentiert werden sollten wichtige Variablen, ungewöhnliche Verfahren und nicht-traditionelle Algorithmen, sprich Seiten, deren Bedeutung sich den Lesern nicht auf den ersten Blick erschließt.

2.8. Bildschirmeingabe und - Ausgabe

In diesem Kapitel wird gezeigt, wie man etwas mit der Tastatur etwas auf den Bildschirm schreibt sowie dort liest. Die Bildschirmausgabe ist die sogenannte Output Current (System.out). Es werden hierfür drei Methoden verwendet: print (), println () und printf (). Der Abruf unten zeigt Hi Baby an.

```
   System.out.println( ” Hi Baby!” );
```

Die Methoden print () und println () drucken den Wert aus, welcher in den Klammern nach der Bezeichnung der Methode festgelegt wurde (dieser Wert wird Parameter genannt). Er weicht durch das Hinzufügen eines Übergangs zu einer neuen println ()-Zeile ab, weshalb der nächste Abruf von print () oder println () vom Anfang der Zeile an ausgeben wird. Bei der Ausgabe kann man Zeichenketten mithilfe des + Operators dazu bringen, sich aneinander anzuschließen.

```
int v = 200;
System.out.println( ”Car moved ” + v + ” km/h” );
```

Eine elegantere Ausgabe wird durch die Methode printf () ermöglicht (die sogenannte formatierte Ausgabe), welche der Sprache C ähnlich ist.

```
int m = 6;
System.out.printf( ” African elephant weighs %d tons”, m );
```

Die printf () Parameter sind eine Formatierungszeichenkette und eine Liste von Werten. Der Formatierungsstring kann die sogenannten Ausgabeumwandlungen enthalten, welche die Form fest-legen, in welcher der passende Wert ausgegeben wird. Jede Umwandlung beginnt mit einem %-Zeichen, so entspricht zB %D der Ausgabe des Dezimal-Integers. Wenn der Befehl ausgeführt wird, wird der passende Wert aus der Werteliste anstelle der Umwandlung eingesetzt. Fehlt der Wert oder passt er nicht mit der Ausgabeumwandlung zusammen, tritt ein Fehler auf.

Ein spezieller Fall ist eine %n-Umwandlung, die mit keinem Wert in der Werteliste übereinstimmt. Diese Umwandlung wird durch eine Verschiebung in eine neue Zeile wiedergegeben. In MS Windows wird das Zeichenpaar \ r (Zellenrückgabe) und \ n (Zeileneingabe) zur Verschiebung in eine neue Zeile verwendet. Unix-Systeme verwenden \n. Die %n-Umwandlung stellt sicher, dass der richtige Übergang zur neuen Zeile verwendet wird, sprich \ r \ n auf MS Windows \ n auf Unix.

Für echte Zahlen gibt es eine %f-Umwandlung. Hier kann man die Anzahl an Ziffern nach dem Dezimalpunkt festlegen (der Standardwert ist 6), sprich %. 2f gibt zwei Ziffern nach dem Dezimal-punkt aus.

```
System.out.printf( ” Euler’s constant is about %.2f%n”,
Math.E );
```

Siehe die unter Verwendung der %c-Umwandlung ausgegebenen Zeichen:

```
char c = '@';
int i = c;
System.out.printf( "Character '%c' Is in the Unicode table
in position %d%n", c, i );
```

Für Strings wird eine %s-Umwandlung verwendet:

```
String Java = "Java";
System.out.printf( " Our favorite programming language
is %s%n", Java );
```

Zur Ablesung von der Tastatur gibt es in JAVA einen sogenannten „Input Stream" (System.in). Dieser wird zumeist nicht direkt, aber durch die Scanner-Klasse verwendet. Diese Klasse bietet Methoden zum Ablesen einfacher Typen und Strings. Wenn die Scanner-Klasse zum Einsatz kommt, beginnt das Programm für gewöhnlich mit einem wichtigen Befehl, welcher dem Übersetzer sagt, wo er nach der Scanner-Klasse suchen soll. Vor dem ersten Ablesen wird mithilfe des neuen Schlüsselworts eine Instanz der Scanner-Klasse erstellt. Um einen Integer-Wert wiederzugeben, muss man die nextInt ()-Methode bei dieser Instanz abrufen.

```
import Java.util.Scanner;
   public class Read {
      public static void main( String[] args ) {
           Scanner sc = new Scanner( System.in );
           int x = sc.nextInt();
           System.out.printf( " Readed value is: %d%n", x );
      }
   }
```

Die Methode nextDouble () wird verwendet, um einen Double-Wert wiederzugeben. Gelesen wird der String durch next ().

```
Scanner sc = new Scanner( System.in );
double d = sc.nextDouble();
System.out.printf( ”Readed number is: %f%n”, d );
String s = sc.next();
System.out.printf( ”Readed string is: %s%n”, s );
```

3. OPERATOREN

Zahlen können gängige arithmetische Operationen in Java ausführen: Addition, Subtraktion, Multiplikation, Division und Modulo (der Rest nach einer Integer-Division). Operatoren werden verwendet, um Operationen zu schreiben. So wird z.B. die Addition mit dem Operator + und der Modulo mit dem Operator % geschrieben. Daher wird mit dem Eintrag x + 2 eine Additions-Operation geschrieben. Jeder Operator funktioniert mit einem oder mehreren Werten oder Variablen, welche Operanden genannt werden. Je nach Anzahl der Operanden unterscheidet man zwischen unären (ein Operand), binären (mit zwei Operanden) und ternären Operatoren (mit drei Operanden). Der föderative Operator ist zB ++ (Erhöhung) und ein binärer Operator ist z.B. = (Zuordnung). Der einzige ternäre Operator ist ? (Bedingungsoperator). Das Ergebnis der Operatorausführung ist ein bestimmter Wert (man sagt, dass der Operator den Wert zurückgibt), z.B. liefert ein binärer Operator + (Addition) die Summe seiner Operanden. Der Ergebnistyp hängt vom Operator und manchmal auch von den Operanden ab. Geben Operatoren einen Integer-Wert zurück, ist das Ergebnis entweder int oder long. Fügt man z.B. zwei int-Werte hinzu, wird das Ergebnis ein int sein, fügt man zwei long-Werte hinzu, wird das Ergebnis long sein und fügt man Zwei-Byte-Werte hinzu, wird das Ergebnis ein int sein. Ein Operator / (aufgeteilt) gibt Integer-Anteile für Integer-Operanden zurück, z.B. 15/4 entspricht 3 und -15/4 ist -3.

Wenn der Wert des zweiten Operanden 0 ist, tritt ein Fehler auf. Wenn zumindest ein Operand eine echte echte Zahl ist (z.B. Float oder Double), wird der zweite Operand in eine echte Zahl umgewandelt und der dividierte Operator wird das Verhältnis beider Operanden zurückgeben, z.B. 4,5/3 ist 1,5. Ist der zweite Operand 0, wird die echte Division zu einem dieser Werte führen: Plus unendlich, wenn der erste Operand positiv ist, minus unendlich, wenn der zweite Operand negativ ist oder NaN (Not a Number), wenn der erste Operant positiv unendlich, negativ unendlich, NaN oder 0 ist. Der Operator % (Modulo) gibt den Rest nach einer Integer-Division zurück, zB 17% 4 ist 1. Die-ser Operator gilt auch für negative Werte, unterscheidet sich jedoch von der Mathematik. Das Vor-zeichen des Ergebnisses ist immer dasselbe wie jenes des ersten Operanden: -17% 4 ist -1, 17% -4 ist 1 und -17% -4 ist 1.

Operatoren kann man auch erhalten, sprich mehrere Operatoren können geschrieben werden. So ist z.B. X +2 *y ein Ausdruck, welcher Additions- und Multikplikationsoperatoren enthält. Die Rang-ordnung von Operatoren ist die Grundlage für die Bewertung der Operatoren-Priorität (Präzedenz), z.B. im Ausdruck x + 2 * y werden zunächst die Multiplikation und dann die Addition ausgeführt, da der Multiplikationsoperator eine höhere Priorität als der Additionsoperator hat. Eine andere Rangordnung kann in Klammern festgelegt werden: (x + 2) * y. Wenn mehrere Operatoren mit derselben Priorität in einem Ausdruck verwendet werden, wird die Assoziativität des Operators durch die Evaluationsreihenfolge festgelegt, z.B. in x – y – z, wird zuerst x – y berechnet, ehe z vom Ergebnis abgezogen wird. Man sagt, dass der Subtraktionsoperator von links nach rechts geht. Aus diesem Grund hat der Ausdruck x – y – z denselben Wert wie der Ausdruck (x – y) – z.

Einige Operatoren arbeiten anders herum, sprich von rechts nach links, wobei der Zuschreibungs-operator eine Ausnahme bildet. Der zurückgegebene Wert des Operators ist der Wert des linken Operanden nach der Zuordnung. Im Ausdruck x = y = 1 ist die erste Zuordnung y = 1 und der zu-rückgegebene Operatorwert (in diesem Fall 1) wird x zugeordnet. Deshalb wird der Ausdruck x = y = 1 als x = (y = 1) gewertet.

3.1. Steigernde und verringernde Operatoren

Der Erhöhungsoperator (++) ist dafür verantwortlich, dass der Wert einer Variable um 1 erhöht wird. Er kann auf zwei Arten geschrieben werden: als Präfix oder als Suffix. In der Präfix-Notation steht der Operator vor dem Operanden, in der Suffix-Notation dahinter. In beiden Fällen wird die Variable erhöht, allerdings gibt es einen Unterschied betreffend den Wiedergabewert des Opera-tors. Der Präfix-Operator gibt den Wert der Variable nach der Vergrößerung zurück und der Suffix-Operator den Wert vor der Vergrößerung.

```
int x = 1;
int y = ++x;
```

Im Ausdruck y = ++ x wird der Erhöhungsoperator zuerst ausgeführt, da er eine höhere Priorität als der Zuordnungsoperator hat. Dadurch erhöht sich der Wert x um 1. Der zurückgegebene Wert des Operators ist x nach der Vergrößerung, sprich 2. Dieser wird als ein Zuordnungsoperator verwendet. Y wird daher 2 sein.

```
int x = 1;
int y = x++;
```

Im Ausdruck y = x++ wird der Erhöhungsoperator zuerst ausgeführt. Sein Rückgabewert ist der Wert der x-Variable vor der Vergrößerung, sprich 1. Der Wert y wird in der Variable y gespeichert. Der Verringerungsoperator (-) sorgt dafür, dass sich der Wert der Variable um 1 verringert. Er wird ähnlich wie der Erhöhungsoperator verwendet.

```
int x = 1;
System.out.println( --x );
```

3.2. Logische Operatoren

Logische Ausdrücke können mittels logischer Operatoren aneinandergereiht werden. Der logische Operator hat boolesche Operanden und gibt einen booleschen Wert zurück. Es gibt zwei logische Operatoren. Der Operator „und“ && gibt den Wert „wahr“ dann und nur dann zurück, wenn beide Operanden wahr sind.

```
if( x == 0 && y == 0 ) {
      system.out.println( ”x and y are equal to zero” );
  }
```

Der Operator „oder“ || gibt den Wert „wahr“ zurück, wenn zumindest einer der Operanden wahr ist.

```
if( x == 0 || y == 0 ) {
      System.out.println( ” At least one of the numbers x,
      y is equal 0” );
}
```

Beide Operatoren verwenden die sogenannte abgekürzte Evaluation, sprich der zweite Operand wird nur dann ausgewertet, wenn der Wert des gesamten Ausdrucks nach der Auswertung des ersten Operanden nicht bekannt ist. Hat der erste Operand einen falschen Wert im logischen Produkt, wird der zweite Operand gar nicht erst ausgewertet und der Wert des gesamten Ausdrucks ist falsch. Da diese Operatoren oft in Bedingungen verwendet werden, sind sie konditional.

Neben Bedingungsoperatoren hat Java auch Operatoren, die immer beide Operanden evaluieren. A & (logisches Produkt) wird „und“ | (logische Summe) geschrieben. Sie können an derselben Stelle wie Bedingungsoperatoren verwendet werden.

```
boolean b1 = x > 0 & y == 1;
boolean b2 = x <= 0 | y <= 0;
```

Kombiniert man „und" und „oder", ist es notwendig zu bedenken, dass „und" eine höhere Priorität als „oder" hat:

```
if( x == 0 || y > 0 && z > 0 ) {
      System.out.println( "x is zero or y and z are positive " );
  }
```

3.3. Zuschreibungsoperatoren

Einer der Zuschreibungsoperatoren, nämlich der Operator =, ist bereits bekannt. Die anderen Zuschreibungsoperatoren ermögliche es, besondere arithmetische Operationen mit Variablen auszuführen, zB der Operator + = fügt der Variable einen zweiten Operanden hinzu.

```
    x += 5;
```

Andere Zuschreibungsoperatoren sind - =, * =, / =, % =. Jeder davon ist eine Aufzeichnung einer damit übereinstimmenden Operation mit der jeweils links befindlichen Variablen. So führt z.B. der Operator % = eine Modulo-Operation aus:

```
    x %= 6;  // dasselbe wie x = x % 6
```

3.4. Priorität von Operatoren

Alle Operatoren geben einen Wert zurück, sprich ein Operationsergebnis. Sie alle haben dieselbe Priorität und gehen von rechts nach links. In der Aussage

```
    int y = x + = 1;
```

führt der Operator zunächst = = aus und erst dann =. Der + Operator = gibt den Wert x zurück, nachdem er um 1 erhöht wurde. Dieser Wert wird als Wert für den zweiten Operator = verwendet.

Die untersuchten Operatoren lassen sich nach Priorität (von der höchsten zur geringsten) sortieren:

- erhöhend (++), verringernd (-)
- verteilend
- Multiplikation (*), Division (/), Modulo (%)
- Addition (+), Subtraction (-)
- Zuschreibungsoperatoren (=, + =, - =, * =, / =,% =)

Die Priorität legt fest, wie stark Operatoren Operanden gewichten. So hat zB die Verteilung eine höhere Priorität als die Multiplikation, weshalb beim Ausdruck (int) d * 2 zuerst die Verteilung und dann erst die Multiplikation vorgenommen wird. Das Ergebnis ist 10.

```
double d = 5.8;
int i = (int) d * 2;  // same like ((int) d) * 2
System.out.println( i );
```

Zuschreibungsoperatoren gehen von rechts nach links und arithmetische Operatoren (Addition, Subtraktion, Multiplikation, Modulo) von links nach rechts.

Alle binären Operatoren werten die Operanden in derselben Reihenfolge aus: zuerst links und dann rechts. Diese Reihenfolge ist von Bedeutung, wenn die Operanden-Evaluation einen Neben-effekt hat.

```
int x = 0;
int y = x + x++;
```

In der zweiten Zeile wird der Operator + zuerst ausgewertet. Dessen linker Operand ist 0, weshalb der rechte Operand auch 0 ist. Darum gibt der Operator 0 zurück und ordnet den Wert y zu. Wird der rechte Operand ausgewertet, wird die Variable x 1 erhöht (die Evaluation hat einen Nebeneffekt). Verändert sich die Reihenfolge der Operanden, wird y der Wert 1 zugeordnet.

```
int x = 0;
int y = x++ + x;
```

3.5. Relationale Operatoren und Gleichheits-operatoren

Um Bedingungen zu formulieren, verwendet man sogenannte relationale und Gleichheitsoperatoren. Relationale Operatoren sind:

- kleiner als (<)
- größer als (>)
- kleiner gleich (<=)
- größer gleich (> =)

Gleichheitsoperatoren sind:

- ist gleich (==)
- ist nicht gleich (! =)

4. GRUNDLEGENDE PROGRAMMIERSTRUKTUREN

4.1. IF

Die Befehle “if” und “switch” werden verwendet, um das Programm zu verzweigen. “If”-Statements erlauben es, ein Programm durch eine Bedingung abzubrechen. Sie starten mit dem Schlüsselwort „if“, gefolgt von booleschen Operatoren in Klammern. Boolesch ist ein Ausdruck, dessen Wert wahr oder falsch ist. Es folgt der eigentliche Befehl. Wird dieser ausgeführt, wird der Ausdruck in den Klammern ausgewertet und ist dieser wahr (wird die Bedingung erfüllt), wird der Befehl ausgeführt. Im nachfolgenden Beispiel wird getestet, ob die Variable x null ist. Wenn diese gleich null ist, wird der x-Variable der Wert 2 zugeordnet.

```
if (x==0) x=2;
```

Man verwendet sowohl relationale als auch Gleichheitsoperatoren, um Bedingungen zu formulieren. Ein “If”-Statement kann auch den “else”-Zweig enthalten, welcher ausgeführt wird, wenn die Bedingung nicht erfüllt wird. Wird die Bedingung nicht erfüllt und fehlt ein alternativer Zweig, wird nichts geschehen (es wird nach dem „if“-Statement fortgesetzt).

```
if( x == 0 )
      System.out.println( ” Can not be divided by zero!” );
else
      z=5/x;
```

Möchte man mehrere Befehle schreiben, wird ein Block verwendet. Der Block beginnt mit einer öffnenden Klammer {und endet mit einer schließenden Klammer}. Ein Block ist ein Java-Befehl, weshalb man ihn in Operationen verwenden kann, welche einen Befehl enthalten:

```
if( x == y ) {
      x++;
      y--;
}
```

Ein Block begrenzt die Validität der Variablendeklaration. Jede Deklaration ist nur bis zum Ende des Blocks, in dem sie gelistet ist, gültig. Man spricht von einer Lokalität im Block.

```
if( x == y ) {
      int z = y;
   }  // Diese Klammer beendet die Deklaration der
      //Variable z
      //Hier kann man z nicht verwenden
```

Man kann boolesche Variablen in Klammern verwenden:

```
// In the month variable we have the serial number of the month
boolean isMay = (month == 5);
     if( isMay ) {
          System.out.println( ”time for love” );
     }
```

Um eine gegenteilige Bedingung zu formulieren, wird der logische Negierungsoperator (!) verwendet.

```
boolean isHere = true;
if( ! isHere ) {
     System.out.println( ”Is not here” );
}
```

Soll das Programm vergezweigt werden, z.B. durch den Wert von Integer-Variablen, lässt sich hierfür eine Verkettung von „if“-Statements verwenden.

```
if( x == 1 ) {
     System.out.println( ”one” );
} else if( x == 2 ) {
     System.out.println( ”two” );
} else if( x == 3 ) {
     System.out.println( ”three” );
}
```

4.2. Switch

Daraus ergibt sich, dass wenn „if“-Statements wie im vorherigen Beispiel aneinandergereiht wer-den, diese manchmal durch den Befehl „switch“ ersetzt werden können. Dessen Formulierung beginnt mit dem Schlüsselwort „switch“. Es folgt ein Integer- oder String-Datentyp (oder ein Daten-typ, der in einen Integer umgewandelt werden kann) in Klammern sowie ein Block mit einer beliebigen Anzahl an Fall-Kennsätzen. In jedem Fall gibt es eine Konstante (oder einen konstanten Aus-druck, einen Ausdruck, dessen Wert zur Übersetzung dient), ein Semikolon und eine Befehlssequenz.

```
switch( x ) {
      case 1:
            System.out.println( ”one” );
            break;
      case 2:
            System.out.println( ”two” );
            break;
      case 3:
            System.out.println( ”three” );
}
```

Bei der Ausführung wird der Ausdruck als das Schlüsselwort “switch” angesehen und dessen Wert wird mit jenen Werten verglichen, die in so einem Fall vorgegeben sind (in der Reihenfolge, in der sie aufgelistet sind). Einmal angefangen, starten die Befehle und werden Ausführung endet mit dem „break“-Befehl. Fehlt ein solcher „break“-Befehl, werden alle Befehle ausgeführt, bis das En-de des Switch-Befehls erreicht ist. Ein „switch“-Befehl kann einen Standardzweig enthalten, welcher ausgeführt wird, wenn kein Fall-Kennsatz passt.

5.ZYKLEN

Zyklen werden verwendet, um Befehle erneut auszuführen.

5.1. While

Der While-Zyklus beginnt mit dem Schlüsselwort “while”, auf welches die Bedingung und der Zyk-lus-Körper in Klammern folgen. Der Körper eines Zyklus ist entweder ein Befehl oder ein Block. Innerhalb eines „while“-Zyklus wird die Bedingung überprüft und wenn sie erfüllt wird, wird der Zyklus-Körper ausgeführt. Anschließend wird die Bedingung erneut überprüft und, wenn sie zu-trifft, wird der Zyklus-Körper erneut ausgeführt usw. Wird die Bedingung nicht erfüllt, wird mit den Befehlen anderer Zyklen fortgesetzt. Wird die Bedingung bereits zu Beginn nicht erfüllt, wird der Zyklus-Körper nicht ein einziges Mal ausgeführt. Der „while“-Zyklus ist also ein Zyklus mit einer Wiederholungsrate von 0 oder mehr.

Beispiel einer Syntax:

```
While(condition) {
// body
}
```

Ein konkretes Beispiel

```
int x = 5;
while( x > 0 ) {  // Do until x is greater than zero
      System.out.println( x );
      x --;
}
```

Dieses Beispiel schreibt Zahlen von 5 bis 1. Der Zyklus wird deshalb fünfmal wiederholt.

5.2. Do while

Der “do while”-Zyklus beginnt mit dem Schlüsselwort „do“, gefolgt von dem Zyklus-Körper. Im Anschluss daran wird das Schlüsselwort „while“ mit der dazugehörigen Bedingung geschrieben, wie das Syntax-Beispiel zeigt. Die Abfolge sieht wie folgt aus: Zunächst wird der Zyklus-Körper ausgeführt, die Bedingung wird überprüft. Wird sie erfüllt, wird der Zyklus-Körper erneut ausgeführt, gefolgt von der Überprüfung der Bedingung usw. Wird die Bedingung nicht erfüllt, wird nach dem Zyklus fortgesetzt. Der Zyklus-Körper ist immer zumindest einmal „do“.

Beispiel einer Syntax:

```
do  {
// body
} while (condition);
```

Ein konkretes Beispiel

```
do  {
System.out.println( x );
      x --;
} while (x > 0);
```

5.3. For

Der “for”-Zyklus hat diese Gestalt: for (Zyklus-Variable, Bedingung, Befehl) und wird wie folgt um-gesetzt: Zunächst wird der Kontrollvariablenzyklus initialisiert, anschließend die Bedingung über-prüft und wenn diese erfüllt wird, wird der Zyklus-Körper ausgeführt. Anschließend wird der Befehl ausgeführt, die Bedingung erneut überprüft und, sofern zutreffend, wird der Zyklus-Körper erneut ausgeführt usw. Die Initialisierung der Kontrollvariable des Zyklus erfolgt nur einmal ganz zu Beginn. Wird die Bedingung zu Beginn nicht erfüllt, wird der Zyklus-Körper kein einzige Mal ausgeführt.

Beispiel einer Syntax:

```
for (cycle variable; condition; command){
// body
}
```

Ein konkretes Beispiel:

```
int a;
for( a = 1; a < 10; a++ ) {
      System.out.println( a );
}
```

Die Zyklus-Variable kann innerhalb des Zyklus neu deklariert werden. Diese Deklaration ist nur in diesem bestimmten Zyklus gültig (sprich im Header und im Zyklus-Körper). Man spricht davon, dass die Variable in diesem Zyklus lokal ist.

```
for( int a = 1; a <= 10; a++ ) {
      System.out.println( a * a );
}
```

Es ist kein Problem, wenn jene Teile, welche die Zyklus-Variable, Bedingung oder den Befehl kontrollieren, fehlen. Fehlt jedoch die Bedingung, wird der Zyklus unendlich (da die Bedingung stets erfüllt wird). Wenn es keinen Kontroll-Zyklus oder keine Befehlsvariable gibt, wird kein Befehl aus-geführt.

5.4. Abbrechen und fortsetzen

Im Zyklus-Körper kann man einen “break”- und einen “continue”-Befehl verwenden. Der “break”-Befehl beendet sofort die Ausführung des Zyklus.

```
int s = 100;
while( s > 0 ) {
      int n = sc.nextInt();
      if( n == 0 ) {
            break;
      }
      s -= n;
      System.out.println( s );
}
// Here will continue after the break executed
```

Der “continue”-Befehl beendet die Ausführung des Zyklus-Körpers und springt zur Zyklus-Bedingung.

```
int s = 0;
do {
      int n = sc.nextInt();
      if( n == 0 ) {
            continue;
      }
      s += n;
      System.out.println( s );
      // here goes continue
} while( s < 100 );
```

6.STATISTISCHE METHODEN

Bis jetzt wurde das gesamte Programm in die Hauptmethode geschrieben. Wenn dieselbe Befehls-sequenz an mehreren Stellen ausgeführt werden soll, muss man diese Befehle wiederholen. Das kann vermieden werden. Methoden ermöglichen es, den Code in logische Einheiten zu unterteilen und diese wiederzuverwenden. In diesem Kapitel geht es darum, Methoden als statische Methode zu verstehen. Eine besondere statische Methode ist bereits bekannt: Es handelt sich um die zuvor genannte Hauptmethode. Neben dieser Hauptmethode lassen sich andere Methoden in Klassen deklarieren, wie z.B. die Methode zur Ausgabe von Programminformationen.

```
class MainClass {
      static void printInfo() {
            System.out.println( "Version: 1.0" );
            System.out.println( " Autor: 007" );
      }
      public static void main( String[] args ) {
            printInfo ();
      }
}
```

Die Deklaration einer statischen Methode beginnt mit dem statischen Schlüsselwort (siehe Beispiel oben), gefolgt vom Wiedergabetyp und der Bezeichnung der Methode. Der Wiedergabetyp kann ein beliebiger Java-Typ sein. Gibt die Methode keinen Wert zurück, wird der Wiedergabetyp als leer festgelegt. Die Bezeichnung der Methode beginnt für gewöhnlich mit einem Kleinbuchstaben. Besteht die Bezeichnung aus mehreren Wörtern, werden die einzelnen Wörter durch Großschreibung des jeweils ersten Wortbuchstaben kenntlich gemacht, z.B. SpoctiPolomerCurzniceOpsane. Es ist nicht üblich, Unterstrichen in Namen zu verwenden. Auf die Methodenbezeichnung folgt in Klammern eine Parameter-Liste. Die Parameter-Liste besteht aus Deklarationen von Variablen. Um die Deklarationen in der Parameter-Liste voneinander zu trennen, wird ein Komma eingesetzt.

6.1. Abrufmethoden

Der Methodenabruf wird an jene Stelle geschrieben, wo die Methode ausgeführt werden soll. Der Abruf einer Methode besteht aus der Methodenbezeichnung und der Parameter-Liste. Der Rückgabewert der Methode wird der Wert des Ausdrucks sein, welcher nach dem Wiedergabe-Statement festgelegt ist.

Die Reihenfolge, in welcher die Methoden deklariert sind, ist unerheblich. Man kann auch eine Methode abrufen, welche erst später deklariert ist.

```
class MainClass {
    public static void main( String[] args ) {
        Scanner sc = new Scanner( System.in );
        int x = sc.nextInt();
        long factorial = countFactorial( x );
        System.out.printf( "%d! = %d%n", x, faktorial );
    }
    static long countFactorial ( int n ) {
        long fact = 1;
        for( ; n > 1; n-- ) {
            fact *= n;
        }
        return fact;
    }
}
```

In der leeren Methode kann man das Wiedergabe-Statement ohne Parameter nutzen. Dieses kommt beim vorzeitigen Beenden der Methode zum Einsatz.

```
// print rectangle a x b from @
static void printRectangle ( int a, int b ) {
       // The minimum rectangle side size is 2
      if( a < 2 || b < 2 ) {
            return;
      }
      for( ; a > 0; a-- ) {
            for( int i = 0; i < b; i++ ) {
                  System.out.print( '@' );
            }
            System.out.pritln();
      }
}
```

Das Wiedergabe-Statement kann in der Methode mehrfach vorkommen. Es wird jedoch immer einmalig als letzter Befehl der Methode ausgeführt. Mit der Ausführung des Wiedergabe-Statements wird die Methode sofort beendet.

```
static boolean isPrimeNumber ( int n ) {
       if( n == 2 ) {  // 2 prime number
            return true;
       }
       if( n % 2 == 0 ) {
            // Even number is not a prime number (except 2)
            return false;
       }
       int sqrt = (int) Math.sqrt( n );
            for( int i = 3; i <= sqrt; i += 2 ) {
            if( n % i == 0 ) {
              // We found a divisor, so n is not a prime number
              return false;
            }
       }
       return true;
}
```

7.INSTANZVARIABLEN

Instanzvariablen werden auch als Instanzattribute oder kürzere Attribute (Instanzen von Instanzen oder Arrays) bezeichnet. Die statischen Methoden sind bekannt und wurden mithilfe des statischen Schlüsselworts bereits deklariert.

Instanziierte Methoden werden auf eine Weise deklariert, die statischen Methoden recht ähnlich ist. Anders als bei statischen Methoden enthält deren Header jedoch kein statisches Schlüsselwort. Darüber hinaus arbeiten instanziierte Methoden oft mit den Instanzattributen

8. ARRAYS

Ein Array ermöglicht es, mit mehreren Werten desselben Datentyps zu arbeiten. So kann man z.B., um 20 Integer-Werte zu speichern, entweder 20 Variablen eingeben oder 20 Elemente umfassen-de Arrays erstellen. In vielen Fällen gestaltet sich die Arbeit mit einem Array einfacher. Eingegeben wird der Array-Typ in Java mittels eckiger Klammern. Die Deklaration des Variablentyp-Arrays für Integer-Objekte sieht wie folgt aus:

```
int[] p;
```

Ein Array-Typ einer Variable ist die sogenannte Referenz. Sie wird einen Verweis auf das Array ent-halten, die Array-Deklaration selbst jedoch nicht. Ein Array lässt sich mithilfe des Schlüsselworts „new“ erstellen:

```
p = new int[6];
```

In oberen Fall wurde ein Array für sechs Integer-Elemente erstellt. Benötigt man eine Anzahl an Array-Elementen, verwendet man *ArrayBezeichnung.length*, sprich in diesem Fall

```
p.length
```

Der Variablen-Wert wird beim Erstellen eines Arrays festgelegt und kann nicht geändert werden (nur gelesen).

```
System.out.printf( ”array p has %d elemets%n”, p.length );
```

Zugriff auf die einzelnen Elemente des Arrays erhält man mithilfe von Indizes, welche in eckige Klammern geschrieben werden:

```
p[1] = 5;
```

Indizes beginnen immer bei null, die erste Zahl wird den Index 0, die zweite den Index 1, die dritte den Index 2 haben usw. Die Validität eines Index wird immer während der Laufzeit überprüft. Verwendet man einen ungültigen Index, führt dies zu einem Laufzeitfehler.

Einmal erstellt, werden Array-Elemente zu Werten initialisiert, deren interne Darstellung 0 entspricht. Bei numerischen Typen ist das die Zahl 0, bei booleschen Typen ist es „falsch“ und beim Zeichentyp char entspricht dies jenem Zeichen, welches in der Unicode-Tabelle an der Position 0 steht. Man verwendet die „for“-Schleife, um Werte im Feld auszulesen:

```
int[] p = new int[10];
        for( int i = 0; i < a.length; i++ ) {
             p[i] = i;
}
```

Mit der “for”-Schleife wird das meiste der Feldelement-Liste erneut ausgeführt:

```
for( int i = 0; i < a.length; i++ ) {
        System.out.println( a[i] );
}
```

Die Feldlänge darf nicht negative sein. Der Versuch, ein Feld mit negativer Länge zu erstellen, führt zu einem Fehler. Die Erstellung eines speziellen Felds kann mit einer Initialisierung kombiniert wer-den. In diesem Fall wird „new“ nicht verwendet.

```
int[]numbers = { 3, 5, 6, 7};
```

Die Feldgröße wird durch die Anzahl an Werten in den Verbindungsklammern vorgegeben. Das Array kann ein Parameter der Methode und auch ein Wiedergabetyp sein.

8.1. Multidimensionale Arrays

Bis jetzt wurde ein Index verwendet, um das Feldelement festzulegen, welches eindimensionales Feld genannt wird. Java erlaubt jedoch auch das Deklarieren und Erstellen multidimensionaler Arrays. So sieht z.B. ein zweidimensionales Feld aus Integer-Objekten wie folgt aus:

```
int[][] p;
```

Ein multidimensionales Feld in Java ist ein Feld aus Feldern. Die Variable p ist ein Verweis auf ein Array, dessen Elemente eindimensionale Integer-Arrays sind. Es ist möglich, mithilfe des Schlüssel-worts „new“ ein neues Feld zu erstellen.

```
p = new int[2][3];
```

Der Zugriff auf die Feldelemente erfolgt über Indizes:

```
p[0][1] = 1;
```

Die Anzahl von Array-Elementen wird durch die Variablenfeldlänge angegeben:

```
System.out.printf( ”pole p má d řádkůn”, p.length );
System.out.printf( ”první řádek má d sloupcůn”, p[0].length );
```

Man verwendet verschachtelte Zyklen, wenn man mit mehrdimensionalen Arrays arbeitet:

```
for( int i = 0; i < p.length; i++ ) {
        for( int j = 0; j < p[i].length; j++ ) {
             p[i][j] = 1;
        }
}
```

Multidimensionale Arrays können sequentiell erstellt werden. Die Sub-Felder können eine andere Anzahl an Elementen haben. Deshalb ist ein zweidimensionales Array nicht zwangsläufig rechteckig.

9. KLASSEN

Eine Klasse ist ein Format, welches einen eindeutig eingeschränkten Datensatz (oder Datensätze) und damit verbundene Operationen beschreibt. Man verwendet Klassen, um Instanzen zu erstellen, sprich einzelne Objekte, welche die Daten selbst enthalten (für welche die übereinstimmen-den Operationen abgerufen werden).

Als Beispiel sei an dieser Stelle eine Auto-Klasse angeführt. Diese Klasse beschreibt, dass das Auto eine Marke, einen Fahrzeugtyp, ein Alter und einen bestimmten Kilometerstand hat. Darüber hin-aus enthält sie die Informationsoperation, das object (), welches den Typ, das Alter und den nicht-Kennzeichen-Typ auflistet. Unter Verwendung dieser speziellen Klasse werden individuelle Instanzen erstellt. Im echten Leben würde man diese als spezielle Fahrzeuge beschreiben.

Jede neu erstellte Instanz (Auto) im Programm ordnet weitere Daten (Marke, Typ, Alter und Kilo-meterstand) zu. Wird die Operatorinformation später abgerufen (), wird dieses Objekt (Instanz) die mit dem Fahrzeug in Verbindung stehende Nachricht zurückgeben.

9.1. Klassendeklaration

Um Klassen zu deklarieren, wird das Schlüsselwort “class” verwendet, welchem ein Zugangsspezifikationssymbol vorangestellt ist und auf welches der Klassenname folgt. Der Klassen-Körper selbst ist in einen Block (Klammern) eingebettet. Besteht der Name aus mehreren Wörtern, wird der erste Buchstabe des jeweiligen Worts in Großbuchstaben geschrieben (z.B. ListingInfo). Unterstriche werden nicht verwendet. In der Klasse können spezifische Variablen und Methoden deklariert werden, welche entweder statisch oder instanziiert sein können.

```
class Cat {
        int weight;  // instance atribut
        int age;
        void showInfo() {  // Instance method
              System.out.println( info );
        }
}
```

Man kann von einer Klasse aus instanziieren. Die Klasse gibt vor, wie Objekte aussehen werden und welche Attribute und Methoden sie haben werden. In diesem Fall wird jede Instanz von „Cat“ zwei Integer-Variablen haben. Durch das Deklarieren der Variable Cat wird eine Variable eingeführt, in welcher sich ein Verweis auf eine Instanz der Klasse Cat speichern lässt.

```
Cat  v;
```

Da sich Variablen des Typs “class” auf Objekte beziehen, werden sie auch Referenzvariablen genannt. Jede Referenzvariable nimmt denselben Platz ein: 32 Bit im 32-Bit JVM und für gewöhnlich 64 Bit im 64-Bit JVM. Andererseits haben Objekte desselben Typs für gewöhnlich eine unterschied-liche Größe. Eine Objektgröße ergibt sich durch dessen Attribute. Objekte werden durch die Verwendung des Schlüsselworts „new“ erstellt:

```
V = new Cat();
```

Nachdem dieser Befehl ausgeführt wurde, wird die Variable v einen Verweis auf eine Instanz der Kubikklasse enthaltene. Auf Attribute und Methoden greift man mittels eines Punkts zu. Methoden werden abgerufen, indem man den Methodennamen und den Klammerinhalt verwendet:

```
v.info = 1;
v. showInfo ();
```

Man kann für eine Klasse eine beliebige Anzahl an Instanzen erstellen. Diese Instanzen sind unabhängig voneinander.

```
Cat v2 = new Cat();
v2. showInfo = 2;
v2. showInfo ();
```

Die instanziierten Methoden werden verwendet, um Operationen mit Objekten eines bestimmten Typs auszuführen. So lässt sich zB in der Klasse „My“ eine Methode deklarieren, welche den Wert des Attributs x um 1 erhöht:

```
class My {
  int x;
  void IncreaseA () {
        a++;
  }
}
```

Instanziierte Methoden (ebenso wie die Klassen-Methoden) können bestimmte Parameter und einen Rückgabewert haben. Sowohl die Parameter als auch der Rückgabewert werden in der Deklaration der Methode festgelegt.

```
class My {
        int x;
        // Adds dx to x and returns a new x value
        int moveX( int dx ) {
              x += dx;
              return x;
        }
}
```

10. ZUGANGSSPEZIFIKATIONSSYMBOL

Zugangsspezifikationssymbole werden verwendet, um die Zugangsrechte zu individuellen Klassen, deren Operationen und Variablen festzulegen. Sie sind vor allem deshalb wichtig, weil sie Anwendungsdetails verstecken, damit sie ein Benutzer nicht sehen oder möglicherweise verwenden kann.

Öffentlich	Von jeder Klasse aus.
Privat	Nur innerhalb der vorgegebenen Klasse, kein Zugang von außen.
Geschützt	Von jeder Klasse desselben Pakets aus, oder ausgehend von irgendeinem Abkömmling der Klasse.
keines (paketfreundlich)	Von jeder Klasse desselben Pakets aus.

Als Beispiel wird hier ein Mitarbeiter mit einem bestimmten Alter und regulären Einkommen angeführt. Auch die IntroduceYourself-Methode kommt hier zum Einsatz.

```
class Employee {
    public Employee (int age, int wage) {
          this.age = age;
          this.wage = wage;
   }
   private int age = 1;
   public int getAge () { return age; }
   public void setAge(int age) { this. age = age; }
   private int wage = 1;

   public int getWage() { return wage; }
   public void setWage(int wage) { this.wage = wage; }
   public void introduceYourself(){
          System.out.println("My age a wage are "  + age +
          "years"+ wage + "Euros");
   }
   public static void main(String[] args) {
          Employee employee = new employee (30,100);
   }
}
```

II. VERERBUNG

Mittels Vererbung kann man eine Klassen-Hierarchie erstellen, wo sich jeder Knoten als ein spezieller Fall beliebiger Ahnenknoten definieren lässt. In der realen Welt würde man sagen, dass ein Stuhl eine Art von Möbelstück ist und ein Flugzeug ein Transportmittel. Ein Stuhl lässt sich jedoch nicht von Tieren unterscheiden, da diese auch vier Beine haben! Um in Java einen Sub-Typ im Klassen-Header zu erstellen, und zwar direct nach dem Na-men, wer-den das Schlüsselwort “extends” und der erweiterte Klassenname verwendet. Diese Klasse wird alle nicht-privaten Methoden (inklusive der paketfreundlichen Metho-den, vorausgesetzt, die Er-weiterungsklasse befindet sich im selben Paket) und Ahnenva-riablen der Klassen übernehmen, welche erneut deklariert und übereinander gelegt wer-den können.

Im Gegensatz dazu übernimmt die Klasse weder die privaten noch die statischen Metho-den ihres Vorgängers, da sich diese nur auf eine bestimmte Klasse des Ahnen beziehen. Die finalen Methoden werden als Nachfahren bezeichnet und die Klasse übernimmt sie, kann sie jedoch nicht über-einander legen. Jedes Kindobjekt kann eine Typumwandlung oder ein Ahne sein.

II.I. Super

Werden Sub-Typ-Methoden abgerufen, stellt man oft fest, dass man nicht die gesamte Methode überlappen, sondern sie nur mit einer umfangreicheren Funktionalität ausstat-ten möchte. An diesem Punkt kann man eine *super.method ()* abrufen, womit man den Ahnen einer Funktionalität gleichsetzt. Man kann den Vorfahren auch einem Konstruktor gleichsetzen, indem man super () abruft. Dies muss allerdings der erste Abruf im Nach-fahren-Konstruktor sein.

Um ein Beispiel zu geben, wird die Klasse Director erstellt, welche sich von der Klasse Employee ableitet. Dies bedeutet jedoch nicht, dass diese Klasse automatisch Zugriff auf alle privaten Variablen und Methoden der ursprünglichen Klasse haben soll (siehe Zu-gangsspezifikationssymbole).

Man setzt die Elternklasse einem Konstruktor gleich, indem man das Schlüsselwort “su-per” verwendet. Darüber hinaus muss es der erste Befehl im Körper des Konstruktors sein. Ruft man den Konstruktor der Elternklasse nicht ab, wird dieser automatisch in das Programm eingereiht – in diesem Fall als sogenannter impliziter Konstruktor, sprich ein Konstruktor ohne Parameter.

```
class Director extends Employee {
    public Director(int age, int wage)
    {
          super(age,wage);
    }
    public static void main(String[] args) {
          Employee k = new Director(30, 50);
    }
}
```

12. POLYMORPHISMUS

Man kann jede Methode in einer eigenen Kind-Implementation überlagern. Ruft man diese Methode in einem bestimmten Objekt ab, wird der überlappende Code immer ausgeführt – unabhängig davon, ob man sich der Methode mittels einer Referenz zum Vorfahrobjekt oder zu einem Nachfahrobjekt (dessen Klasse diese Überlappung enthält) nähert.

Diese Eigenschaft wird durch späte Bindung erreicht, wenn der Objekttyp, für welchen die Methode abgerufen wird, während der Laufzeit festgelegt wird, nicht während der Erstellung.

Dies lohnt sich jedoch nur für das Überlappen von Methoden, jedoch nicht für deren Überlastung. Hat man zwei Methoden für ein bestimmtes Objekt, welche sich nur durch die Parameter unter-scheiden (eine für den Ahnen, die andere für den Nachfahren), wird die Methode abgerufen, welche dieselben Parameter wie der aktuelle Verweis auf das Objekt hat. Der Abruf überlastender Methoden wird zum Zeitpunkt der Übersetzung festgelegt, wenn noch immer nicht klar sein muss, ob die Referenz auf einen Ahnen oder ein Nachfahrobjekt zeigt.