JAVA I

Grundlagen der Programmierung

Teil 1 von 5:

Imperative Sprachelemente

Letzte Änderung: 13.03.2017

Ihr Dozent: Steffen Kemena

- Seit 2008 beruflich Softwareentwickler
    - **JAVA** (Web und Serverentwicklung, Spring Framework)
    - **PHP** (Symfony Framework, Zend Framwork)
    - Weitere Programmiersprachen: **JavaScript , C#, C/C++, Python**
    - Datenbanken (**MySql**)
    - Webentwicklung Allgemein (**HTML, CSS, Bootstrap**)
    - Template-Engines (**Thymeleaf**, **Twig**, Smarty)
    - **Unity3D** und Spieleentwicklung
- Seit 2016 Dozent
- Weitere Infos unter http://www.leycarno.com

Die Kursteilnehmer

- **Wer sind Sie? Wer bist du?**
- **Wissensstand: Java? Programmiersprachen? Komplett planlos?**
- **Welche Erwartungen gibt es an den Dozenten und diesen Kurs?**

Inhalte des Kurses - Kurzübersicht

Teil 1: **Imperative Sprachkonzepte**

Teil 2: **Objektorientierung Programmierung**

Teil 3: **Datensammlungen und Aufzählungen**

Teil 4: **Persistenz mit JSON und Files**

Teil 5: **Softwareentwicklung**

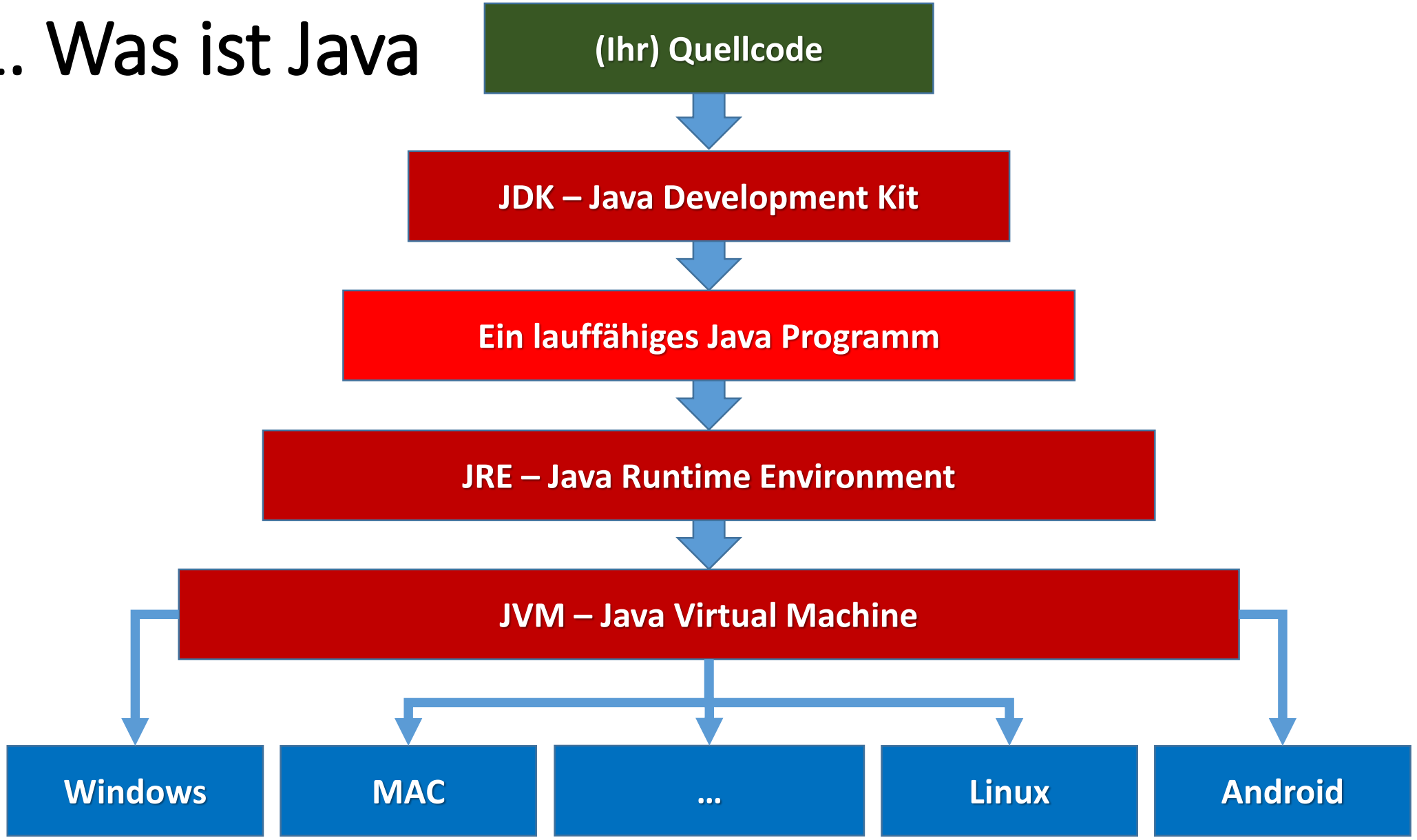
Teil 1 - Imperative Sprachkonzepte

1. **Was ist Java?**
2. **Hello World!**
3. **Bezeichner, Variablen und Datentypen**
4. **Ausdrücke und Operatoren**
5. **Kontrollstrukturen**

1. Was ist Java

- Eine komplett **Objektorientierte** Programmiersprache
- Aktuell in der **Version 1.8** = **Java 8**
- Erste Java Version **1995** von Sun Microsystems (Seit 2010 Oracle)
- „***Einfach***“ im Vergleich zu **C++** oder **C#** durch reduzierten Umfang
- **Typsicher** im Gegensatz zu z.B. *PHP* und gilt als **rubust** durch geringe Fehleranfälligkeit
- Plattformunabhängig durch die **Java Virtual Machine**
- Anmerkung: **JavaScript** hieß früher ***LiveScipt*** und hat NICHTS mit JAVA zu tun

1. Was ist Java



1. Was ist Java

JDK – Java Development Kit

- Wird für die Entwicklung von Java Programmen benötigt
- Benötigt zur Ausführung der Programme eine passende JRE

JRE – Java Runtime Environment

- Wird zur Ausführung von Java-Programmen benötigt.
- Sollte auf dem neuesten Stand gehalten werden

JVM – Java Virtual Maschine

- Vermittelt zwischen Java Programmen und dem Betriebssystem
- Dadurch ist Java Plattformunabhängig

2. Hello World!

5.1. Das erste Programm

```
public class Main {

    public static void main(String args[]) {
        System.out.println("Hello World!");
    }

}
```

2. Hello World!

5.2. Einzeilige Kommentare:

```
int aNumber = 10; // „aNumber“ wird 10 zugewiesen
```

5.3. Mehrzeilige Kommentare

```
int aNumber = 10; /* Dies ist ein Kommentar über
                     mehrere Zeilen: Der Variablen
                     „aNumber“ wird   das Literal 10
                     zugewiesen... */
```

2. Hello World!

5.4. Dokumentationskommentare

```
/**
 * Dies ist eine „Methode“, oder auch „Funktion“.
 * Man könnte sagen ein „Unterprogramm“, das mehrmals im
 * Hauptprogramm aufgerufen werden kann.
 *
 * @var         int          ein Parameter
 * @return      int          der Rückgabe-Wert
 */
int square(int aNumber)     {
    return aNumber * aNumber;
}
```

Was das genau heißt, wissen wir spätestens nach Teil 2 dieses Kurses...

2. Hello World!

5.5. Eingaben

Mit folgendem Aufruf ist es möglich, Eingaben entgegenzunehmen:

```
Scanner scanner = new Scanner(System.in);

String name  = scanner.nextLine();
int input     = scanner.nextInt();
```

Der Aufruf von „**`nextLine()`**“ ist auch notwendig, um *unverarbeitete Eingaben* von nextInt() zu „vergessen“

3. Bezeichner, Variablen und Datentypen

3.1. Bezeichner

- **Was sind Bezeichner?**

  Kurz*:* *Wann immer der Programmierer selbst Namen vergibt, handelt es sich dabei um einen Bezeichner…*

- **Die Regeln für Bezeichner:**

  - Beliebig viele ***Unicode*** *(engl. Zeichensatz)* Zeichen und Zahlen
  - Erstes Zeichen = ***Buchstabe***, ***Unterstrich*** oder ***Dollarzeichen***
  - ***Keine Schlüsselwörter*** (im Kurs durchgängig blau dargestellt)
  - ***CaseSensitive*** = `anumber` ist etwas anderes als `aNumber` !

3. Bezeichner, Variablen und Datentypen

3.2. Bezeichner - Konventionen

- Es ist anzuraten, ausschließlich **englische** Bezeichner zu nutzen!
- Zusammengesetzte Wörter im „**CamelCase**“ – Format
- ***Variablen*** (und ***Methoden***) beginnen mit einem **Kleinbuchstaben**

„Gute Bezeichner erzählen was sie sind und was sie tun!“

3. Bezeichner, Variablen und Datentypen

3.3. Variablen zuweisen/definieren

- Definition:
- Initialisierung:

```
int aNumber;
aNumber = 10;
```

- Explizite Initialisierung:

```
int anotherNumber = 20;
```

- Mehrere Variablen eines selben Datentyps gleichzeitig definieren:

```
int age, size, weight;
```

- Und auch explizit:

```
int height = 10, width = 20;
```

3. Bezeichner, Variablen und Datentypen

3.4. Die einfachen („primitiven“) Datentypen

- Wahrheitswerte

```
boolean isReady = true;
```

- Ganzzahlen

```
int pos = 123, neg = -987;
```

- Fließkommazahlen *(englischer Punkt statt deutschem Komma!)*

```
float number = 12.34f;
```

- Zeichen *(nur genau EIN Zeichen in „einfachen“ Anführungszeichen)*

```
char letter = 'A';
```

3. Bezeichner, Variablen und Datentypen

3.4. Die 8 einfachen („primitiven“) Datentypen - Übersicht

Bezeichnung	Datentyp/Schlüsselwort	Wertebereich
Wahrheitswerte	`boolean`	`false, true`
Zeichen	`char`	`a…z, A…Z, 0…9, !"§$%&/()...` Alle möglichen Zeichen
Ganze Zahlen	`byte`	`-128 … 127`
	`short`	`-32.768 … 32.767`
	`int`	`-2.147.483.648 … 2.147.483.647`
	`long`	`-9.223.372.036.854.775.808 … 9.223.372.036.854.775.807`
Fließkommazahlen/ Gleitkommazahlen	`float`	`~1,4*10^-45 … 3,4*10^38` *Genauigkeit ca. 7 Stellen*
	`double`	`~4,9*10^-324 … 1,8*10^308` *Genauigkeit ca. 15 Stellen*

3. Bezeichner, Variablen und Datentypen

3.5. String und Ausgabe *(„Auf den Schirm!“)*

String ist eine **Standard-Klasse**. Es ist KEIN „einfacher/primitiver“ Datentyp. Trotzdem wird sie so oft genutzt, dass Sie hier erwähnt wird.

```
String string1 = „Hello“;
String string2 = „World“;
String message = string1 + „ “  + string2;
System.out.println(message);  // Ausgabe!
```

WICHTIG: Vergleiche erfolgen nicht mit == sondern mit einer Methode Namens „equals“:

```
if (string1.equals(string2)) ...
```

4. Ausdrücke und Operatoren

4.1. Ausdrücke

- Ein **Ausdruck** ist eine Kombination von **Operatoren** und **Operanden**

- Das **Ergebnis** des Ausdrucks wird „nach links geschrieben“ (***zugewiesen***)

```
int aNumber      =    2          +          8;
Ergebnis         =  Operand1     Operator   Operand2
```

- Jeder Operator hat eine **Priorität**, wodurch die Reihenfolge mit der sie innerhalb eines Ausdrucks abgearbeitet werden, festgelegt ist.

4. Ausdrücke und Operatoren

4.2. Arithmetische Operatoren

Bezeichnung	Operator		Priorität
Positives Vorzeichen	`+`	10 (das Vorzeichen wird weggelassen)	**1**
Negatives Vorzeichen	`-`	-10	
Inkrementieren	`++`	`a++;` äquivalent zu: `a = a + 1;`	
Dekrementieren	`--`	`a--;` äquivalent zu: `a = a - 1;`	
Multiplikation	`*`	`int multiplied = a * b;`	**2**
Division	`/`	`float divided = a / b;`	
Rest	`%`	`int modulo = a % b;`	
Addition	`+`	`int addition = a + b;`	**3**
Subtraktion	`-`	`int subtraction = a - b;`	

4. Ausdrücke und Operatoren

4.3. Relationale Operatoren

Bezeichnung	Operator		Priorität
Kleiner	`<`	`boolean isSmaller = a < b;`	5
Kleiner oder gleich	`<=`	`boolean isSmallerOrEuqal = a <= b;`	
Größer	`>`	`boolean isGreater = a > b;`	
Größer oder gleich	`>=`	`boolean isGreaterOrEqual = a >= b;`	
Gleich	`==`	`boolean isEqual = a == b;`	6
Ungleich	`!=`	`boolean isUnequal = a != b;`	

4. Ausdrücke und Operatoren

4.4. Logische Operatoren

Bezeichnung	Operator		Priorität
NICHT	`!`	`boolean isTrue = !isFalse;`	1
UND („vollständige“ Ausw.)	`&`	`boolean isTrue = justTrue & trueToo;`	7
ODER („vollständige“ Ausw.)	`|`	`boolean isTrue = justTrue | justFalse;`	9
UND („kurze“ Auswertung)	`&&`	`boolean isTrue = justTrue && trueTrue;`	10
ODER („kurze“ Auswertung)	`||`	`boolean isTrue = justTrue || justFalse;`	11
exklusive ODER (XOR)	`^`	`boolean isTrue = justTrue ^ justFalse;` `boolean isFalse = justTrue ^ trueTrue;`	8

Der Unterschied von `&` *zu* `&&` *wird später erläutert*

4. Ausdrücke und Operatoren

4.5. Bitoperatoren

Beispiele: **8 << 2 = 32** **8 >> 2 = 2**

0000 1000 (also 8)	**<<**	**2**	**=**	**0010 0000 (also 32)**
0000 1000 (also 8)	**>>**	**2**	**=**	**0000 0010 (also 2)**

Was ist das binäre Zahlensystem?

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2^7	**2^6**	**1^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^1**
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Beispiele:	**Die Zahl**	**9**	**(= 8 + 1)**	**schreibt man binär**	**0000 1001**
	Die Zahl	**123**	**(= 64 + 32 + 16 + 8 + 2 + 1)**	**schreibt man binär**	**0111 1011**

…eine Erwähnung der Vollständigkeit halber - Thema kommt nicht weiter vor…

4. Ausdrücke und Operatoren

4.6. Zuweisungsoperatoren

Bezeichnung	**Operator**		**Priorität**
Einfache Zuweisung	**=**	Das Ergebnis des Ausdrucks (rechts vom Zuweisungsoperator) wird einer Variablen (links vom Zuweisungsoperator) zugeordnet	**13 (immer die letzte Aktion)**

Kombinierte Zuweisung	*op*=	`int aNumber = aNumber + anotherNumber;`
		Vom Ergebnis her dasselbe wie:
		`int aNumber += anotherNumber;`
		
		Mögliche bei: **`* / % + -`** **und Bitoperatoren…**	

4. Ausdrücke und Operatoren

4.7. Cast-Operator (Priorität = 1)

```
int x = 13;
int y = 5;
long bigResult = x * y;
int result = bigResult; // FEHLER!
int result = (int) bigResult; // OK
```

Problemlos, wenn Ergebnistyp „größer“

Absicht bei „kleineren“ Ergebnissen angeben

```
float f = 10.123
int i = f;          // FEHLER !!!
int i = (int) f;    // OK
```

Umwandlung der Fließkommazahl **10.123** in die Ganzzahl **10** erfolgt NICHT automatisch.

5. Kontrollstrukturen

5.1. Das Ende eines Ausdrucks ;

- Mit einem **Semikolon** wird das Ende eines Ausdrucks definiert

5.2. Block {…}

- Die Grenzen von Klassen, Methoden und anderen Kontrollstrukturen
- Zusammenfassung mehrerer aufeinander folgenden Ausdrücke.
- Alle Variablen innerhalb eines Blocks, sind nur dort gültig.

5. Kontrollstrukturen

5.3. Verzweigungen mit Bedingung (1/2)

Einfachste Form der Bedingung – der Block wird nur ausgeführt, wenn Bedingung „wahr“:

```
if (a < b) {
        System.out.println(„a is smaller than b“);
}
```

Alternative – was soll ich „sonst“, wenn der erste Ausdruck „falsch“ ist:

```
if (a > b) {
        System.out.println(„a is bigger than b“);
} else {
        System.out.println(„a is NOT bigger than b“);
}
```

5. Kontrollstrukturen

5.3. Verzweigungen mit Bedingung (2/2)

```
if (a < b) {
    System.out.println(„a is smaller“);
} else if (a > b) {
    System.out.println(„a is bigger“);
} else {
    System.out.println(„a & b are equal“);
}
```

5. Kontrollstrukturen

5.4. Verzweigung mit dem Bedingungsoperator (Priorität = 12)

Bedingung ? Ausdruck1 : Ausdruck2

Ist die **Bedingung wahr**, wird **Ausdruck1** ausgewertet.

Ansonsten wird **Ausdurck2** ausgewertet.

Verschachteln vermeiden!!!

```
b3 ? b2 ? b1 ? a1 : a2 : a3 : a4;          // ??????
```

5. Kontrollstrukturen

5.6. Verzweigungen abhängig von immer derselben Variable

Die Ausdrücke links und rechts sind äquivalent:

```
if (aNumber == 1) {
        aNumber += 2;
} else if (aNumber == 4) {
         aNumber--;
} else {
        aNumber = 3;
}
```

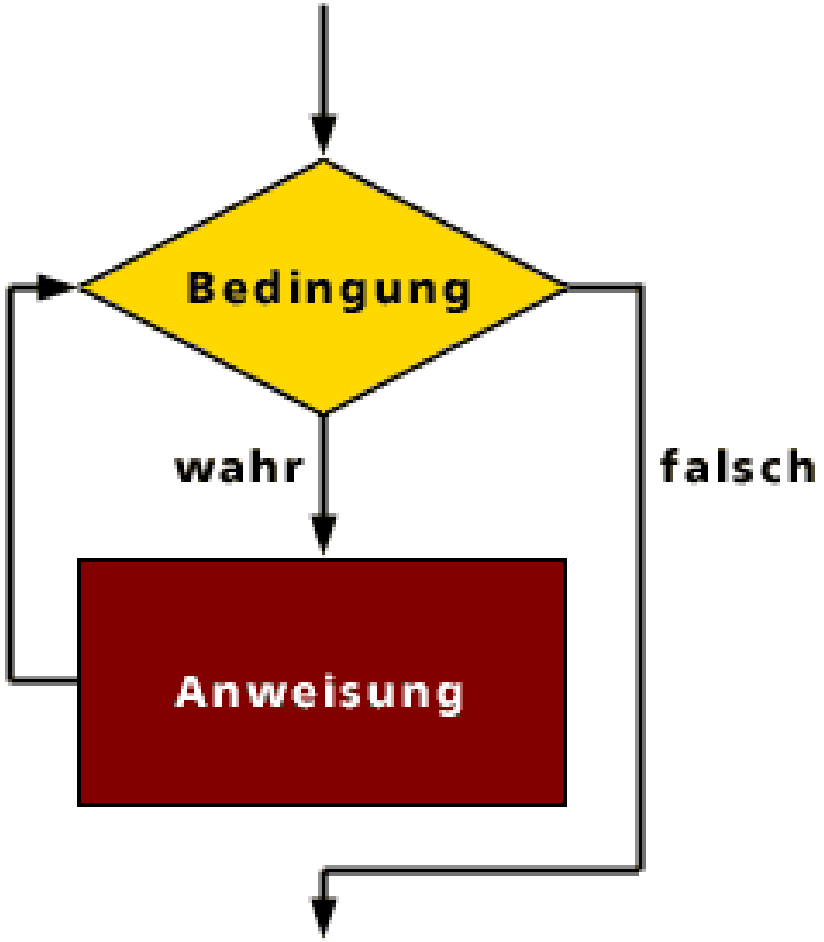
```
switch (aNumber) {
        case 1:
                aNumber += 2;
                break;
        case 4:
                aNumber--;
                break;
        default:
                aNumber = 3;
                break;
}
```

5. Kontrollstrukturen

5.7. Schleifen (1/4) „while“

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

```
while (Bedingung) {
      Anweisung
}
```

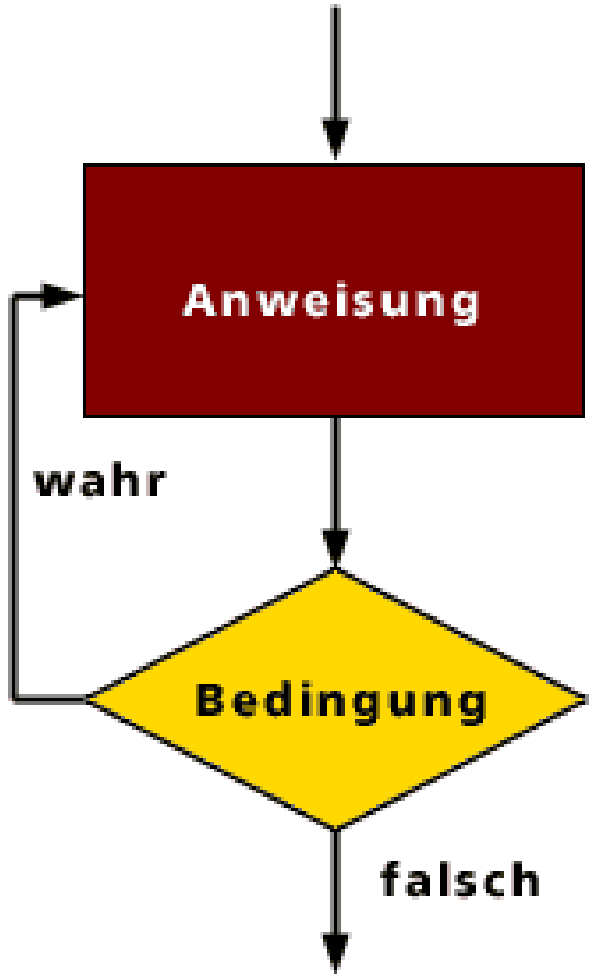


5. Kontrollstrukturen

5.7. Schleifen (2/4) „do“

```
int i = 1;
do {
      System.out.println(i++);
} while (i < 10);
```

```
do {
     Anweisung
} while (Bedingung);
```

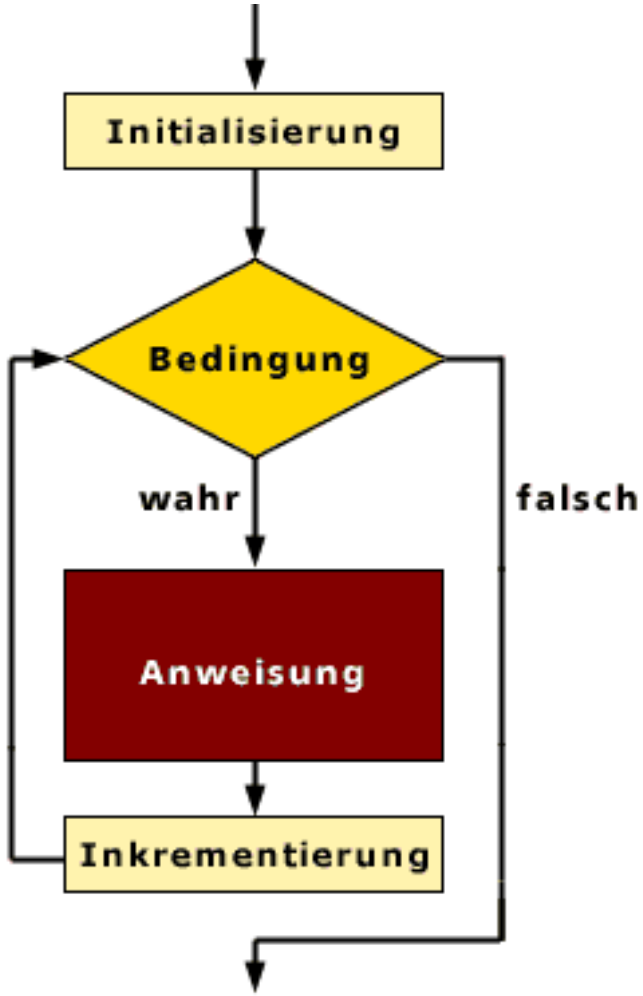


5. Kontrollstrukturen

5.7. Schleifen (3/4) „for“

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

```
for (Init; Bedingung; Update){
      Anweisung
}
```



5. Kontrollstrukturen

5.7. Schleifen (4/4) *Sprunganweisungen*

```
boolean justOddNumbers;
int externalMaxValue;
// [...]
for (int i = 1; i < 10; i++) {
        if (justOddNumbers && i % 2 != 0) {
                continue;          // naechster Schleifendurchlauf
        }
        if (i > externalMaxValue) {
                break;             // Schleife komplett abbrechen!
        }
        System.out.println(i);
}
```

Der Unterschied von & und &&

```
boolean x = trueOrFalse;  // „trueOrFalse“ ist wahr ODER falsch...
boolean isUnsafe = x    &&    a > b++;
boolean isSafe   = x    &     a > b++;
```

`&&` = Ausdruck wird **beim ersten false abgebrochen**. `b` wird NICHT inkrementiert!

`&` = Ausdruck wird **vollständig ausgewertet** und `b` ist danach um `1` höher!

```
int x = 0;
while (x < 3) {
        if (false & x++) {
                System.out.println(x);
        }
}
```

```
int x = 0;
while (x < 3) {
        if (false && x++) {
                System.out.println(x);
        }
}
```

Die linke Schleife wird beendet, die rechte läuft endlos weiter…