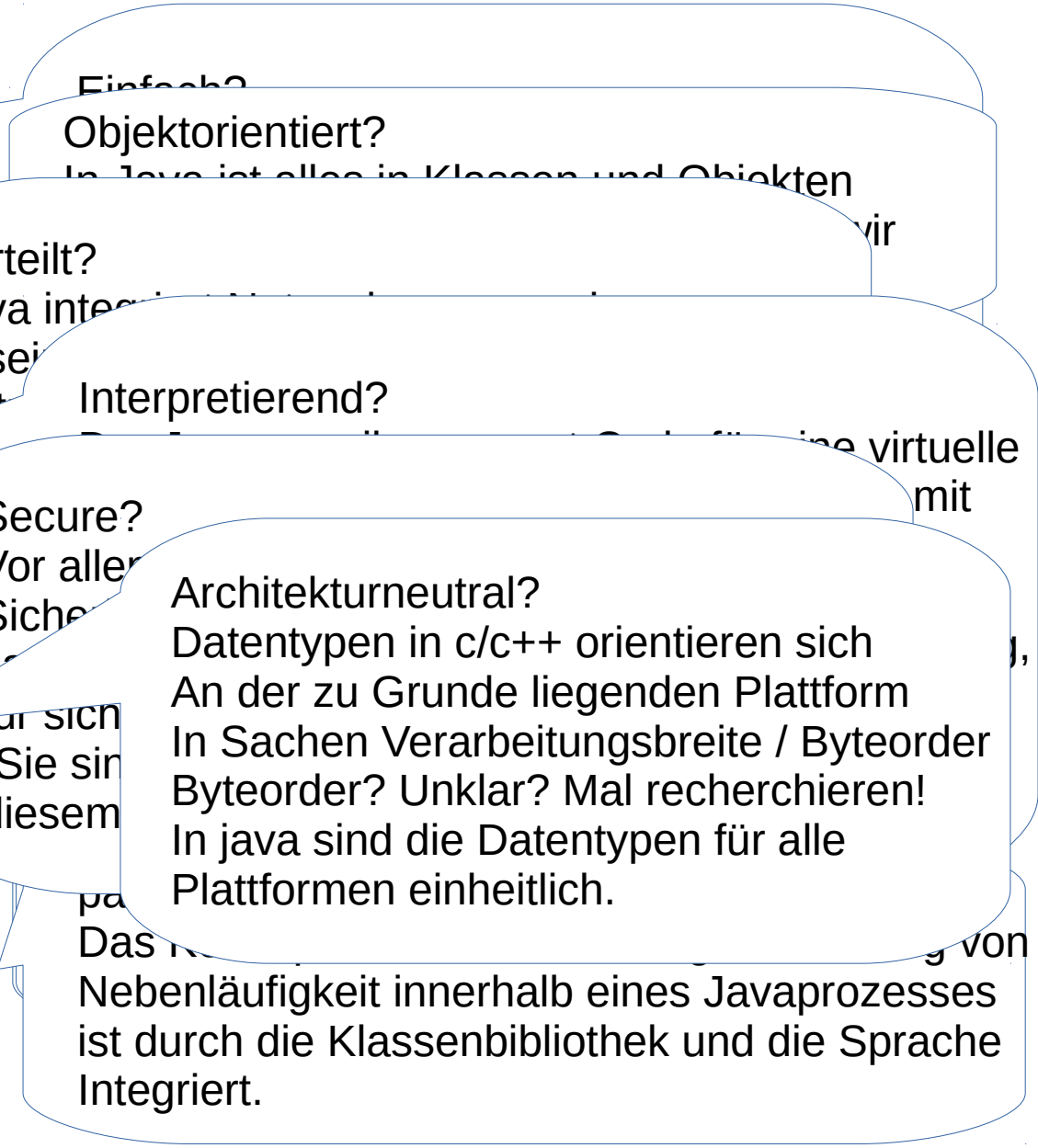
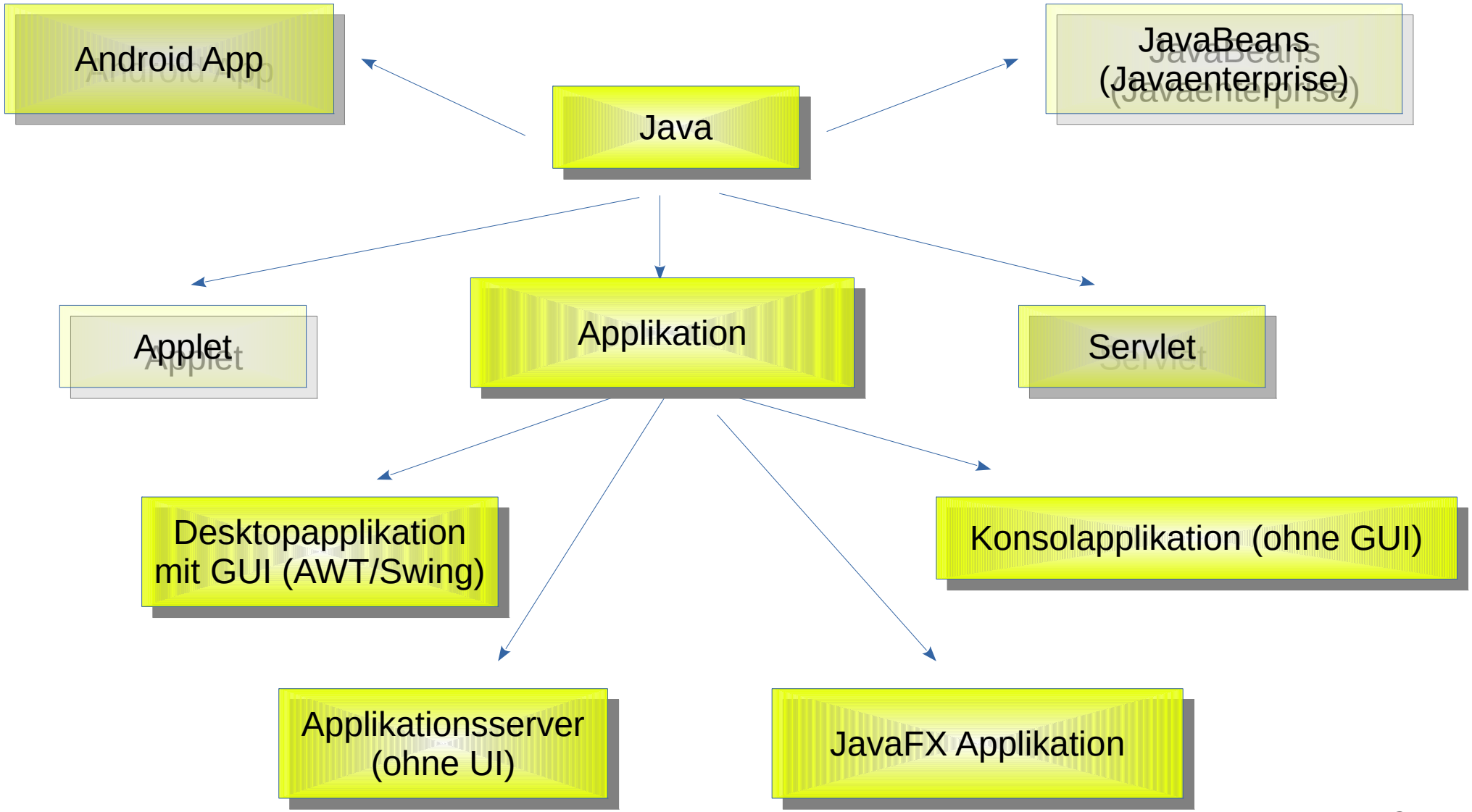
Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel
- schnell
- parallel(multitheded)



Vielfalt von Java-Anwendungen



Erstes Java-Programm

Javadateien enthalten (zunächst) immer
Nur eine Klasse. Der Name der Klasse steht hinter
Dem Schlüsselwort „class“. Der **Dateiname muss mit Diesem Namen, ergänzt um „.java“ übereinstimmen**.

In diesem Fall:
FirstClass.java

```
// Beispiel 1
import java.io.*;

class FirstClass
{
  public static void main(String args[])
  {
    int i=2;
    System.out.printf(
     "Willkommen im %d. Semester\n",i);
  }
}
```

Die Werkzeuge

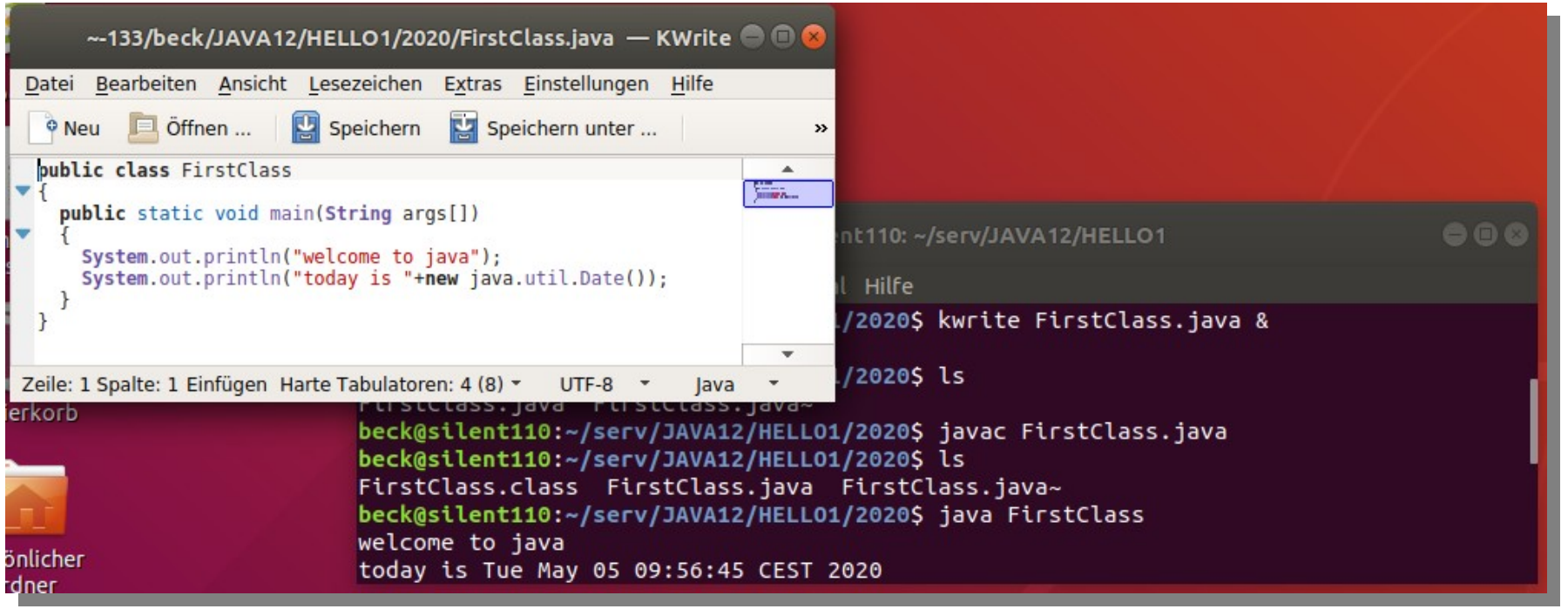
- In der Konsole:
  - Programmeditor (kwrite, kate, atom ….)
  - Java Development Kit (JDK)
- IDE
  - InelliJ IDEA, eclipse, netbeans ...

Die Werkzeuge des jdk - Compiler

Name der Java-Klasse (class FirstClass) muss mit dem Dateinamen übereinstimmen!!! (**class FirstClass – Datei: FirstClass.java**)

```
javac FirstClass.java
```

Aus dem java-Quellfile werden ein oder mehrere .class-Files erzeugt (FistClass.class)



Die Werkzeuge des jdk - Virtuelle Maschine

Der java-Compiler erzeugt sogenannten ByteCode. Es ist ein Code für eine hypothetische, also nicht real existierende Maschine. Diese virtuelle Maschine wird durch ein Programm „nachgebildet“. Java-Programme können also nicht, wie übersetzte c-Programme direkt auf der Maschine ausgeführt werden.

Mit dem Aufruf von java wird die virtuelle Maschine gestartet.

```
java FirstClass
```

führt das Javaprogramm aus, in dem die main-funktion der angegebenen Klasse aufgerufen wird.

Die Werkzeuge des jdk – Archiv
(optional)

Ein Javaprogramm kann aus sehr vielen Klassen, die jede in einer .class Datei nach dem Compilieren vorliegen, bestehen.

- Jar Fasst alle Bestandteile (hautsächlich die Klassen) eines Javaprogramms in einer Archivdatei zusammen.

```
jar cfm FistClass.jar MANIFEST.MF FirstClass.class
```

- In der Datei `MANIFEST.MF` sind Festlegungen zu dem Programm getroffen, im einfachsten Fall, die Klasse, mit der das Programm starten soll (die die aufzurufende main-Funktion enthält).
- MANIFEST.MF enthält im einfachsten Fall die Zeile:

```
Main-Class: FirstClass
```

- Die Zeile muss mit einem Zeilenvorschub (Enter) abgeschlossen sein!
- Aufruf eines Javaprogramms aus einem jar-File:

```
java -jar FirstClass.jar
```

Erstes Java-Programm V2.0

Javadateien enthalten (zunächst) immer
Nur eine Klasse. Der Name der Klasse steht hinter
Dem Schlüsselwort „class“. Der Dateiname muss mit
Diesem Namen, ergänzt um „.java“ übereinstimmen.

In diesem Fall:
FirstClassV2.java

```
// Beispiel 1
import java.io.*;

class FirstClassV2
{
  public static void main(String args[])
  {
    int i=2;
    System.out.println(
        "Willkommen im "+i+". Semester");
  }
}
```

Strings können sehr komfortabel,
auch mit Zahlen u.a. verkettet werden

Kommandozeilenargumente

```
public class HelloEcho
{
  public static void main(String args[])
  {
    for(int i=0;i<args.length;i++)
                System.out.println(args[i]);
    System.exit(0);
  }
}
```

Zum Vergleich als C-Programm:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i;
    for (i=0; i<argc; i++) puts(argv[i]);
    return 0;
}
```

Führen Sie das Programm oben in einer Konsole aus:

```
java HelloEcho hans anna peter sylvia
```

Operatoren

Java stellt nahezu alle Operatoren von C in gewohnter Weise zur Verfügung
Ausnahmen bilden:

- Während >> und << vorzeichenbehaftet arbeiten, führt der Operator >>> eine Rechtsverschiebung aus, bei immer der Nullen von links aufgefüllt werden.
- Den Kommaoperator gibt es in Java nur in Verbindung mit der for-Anweisung.
- Den Operator sizeof gibt es nicht.
- Der Operator instanceof stellt fest, ob ein Objekt ein Objekt einer bestimmten Klasse ist.
- Die Operatoren == und != liefern eine Aussage, ob es sich um ein und das selbe Objekt handelt, wenn sie auf Referenzvariablen angewandt werden, auf primitive Datentypen angewandt, wird ein gewöhnlicher Vergleich ausgeführt.
- Keine Pointer
- Der Operator + auf Stringoperanden angewandt, verkettet zwei Strings und bildet einen neuen String.
- Die Operanden && und || bewirken die Verknüpfung nach dem Kurzschlußverfahren, es können auch die Operatoren & und | zur Anwendung kommen, dann werden beide Teilausdrücke immer bewertet (getestet).

Eingebaute Datentypen (primitive types)

Type	Inhalt	Default	Size	Min ... Max
boolean	true | false	false	1 bit	false ... true
char	unicode char	\u0000	16 bit	\u0000 ... \uffff
byte	signed int	0	8 bit	-128 ... 127
short	signed int	0	16 bit	-32768 ... 32767
int	signed int	0	32 bit	-2147483648 ... 2147483647
long	signed int	0	64 bit	-9223372036854775808 ... 9223372036854775807
float	IEEE 754 floting point	0.0	32 bit	±3.40282347E+38 ... ±1.40239846E-45
double	IEEE 754 floting point	0.0	64 bit	±1.79769313486231570E+308 ... ±4.94065645841246544E-324

Eingabekonvertierung

Konvertierung erfolgt über Methoden der wrapper classes aus java.lang (nächste Folie)

Umwandlung String → numerischen Wert

In c: atoi, strtol, strtod

Beispielsweise Integer.parseInt, Double.parseDouble, Long.parseLong usw.

```
  int i;
// statische Methode
i=Integer.parseInt(args[0]);
// Member function Methode
i=new Integer(args[0]) . intValue();
```

i enthält als Ergebnis den Integerwert

String, der Ziffern enthält

wrapperclasses

Typ	wrapperclass
byte	java.lang.Byte
boolean	java.lang.Bool
char	java.lang.Character
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double

Wrapperclasses enthalten zu den primitiven Datentypen jeweilige Funktionalität, zum Beispiel Konvertierungsfunktionen, aber auch viele weitere interessante Funktionen, wie beispielsweise das Vertauschen, Rotieren … von Bytes in Int-Werten.

Suchen Sie in der API-Dokumentation
http://www.informatik.htw-dresden.de/~beck/PSPII_WI/
API Java8
Nach dem Package java.lang und darin nach
Den links angegebenen Wrapperclasses.
Finden Sie in den Wrapperklassen die
Konvertierungsfunktionen parse….

Ausgabe / Konvertierung numerischer Werte

Aufruf von toString der zugehörigen wrapperclass

```
String s= new Integer(i).toString();
```

oder implizit

Operator + in Stringverkettung

```
int i=11;

String s=""+i;
```

Nutzung von PrintStream / PrintWriter

```
System.out.println(i);
```

Fomatierte Ausgabe / Konvertierung numerischer Werte

```
System.out.printf("0x%04x %d\n",i,i);


String s;
s=String.format("0x%04x %d\n",i,i);
```

printf, wie in C

Wie sprintf in c

Statements

Expressionstatement

if-statement

while-statement

do-while-statement

for-statement

switch-statement

break / continue -statement

synchronized-statement

Die meisten Statements sind syntaktisch und in ihrer Funktionsweise, den Pendants in c sehr ähnlich. Wichtiger Unterschied besteht in den Wahrheitswerten. Die Wahrheitswerte in Bedingungen müssen in java vom Typ boolean sein.

Eingabe von Standardeingabe (wird eher selten gebraucht)

```
import java.io.*;
class StdIo
{
  public static void main(String args[]) throws Exception
  {
    int i;
    BufferedReader br= new BufferedReader
            (new InputStreamReader(System.in));
    System.out.println("Input:");
    String s= br.readLine();
    System.out.println("Input:"+s);
    if (Character.isDigit(s.charAt(0)))
    {
      i=Integer.parseInt(s);
      System.out.println("i    :"+i);
    }
  }
}
```

Eingabe von Standardeingabe eher selten

```
import java.io.*;
import java.util.*;
class StdIoScanner
{
 public static void main(String args[]) throws Exception
 {
  int i=0;
  Scanner sc = new Scanner(System.in);
  while(true)
  {
   if(sc.hasNextInt()){ i=sc.nextInt();System.out.println(i);}
   else
   if (sc.hasNext("Quit")) break;
   else
   {
     System.out.println("falsche Eingabe, ….");
     sc.nextLine();
   }
  {
 }
}
```

Boolsche Werte

In if-, while-, do/while- statements müssen die Bedingungen auch vom Typ boolean sein

```
while (true) ...

if (i != 0) ...
```

for-statement

Vereinbarung von Schleifenvariablen kann im Initialisierungsausdruck erfolgen

```
for (int j=0; j<5; j++)System.out.println(j);
```

```
for (int j=0; j<5; j++)System.out.println(j);
```

Kommaoperator

```
for (int j=0, i=1; j<5; j++,i++)
```

Oder

```
for (j=0, i=1; j<5; j++,i++)
```

Aber nicht:

~~`for (j=0, int i=1; j<5; j++,i++)`~~

for-statement (for each)

```
class foreach
{
  public static void main(String args[])
  {
     for(String x:args)
        System.out.println(x);
  }
}
```

Klassische Variante:

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

break / continue

Bedeutung wie in C **aber**

Schleifen können mit Labels markiert werden

break/continue können über diese Labels auch umgebende Schleifen steuern

Die böse sieben, continue mit Label

```
1  import java.math.*;
2  import java.io.*;
3  class ContinueLabel
4  {
5    public static void main(String args[])
6    {
7      int array[][]=new int[3][4];
8      int i,j;
9      for (i=0; i<array.length;i++)
10       for(j=0; j<array[i].length;j++)
11         array[i][j]=(int)(Math.random()*10);
12     for (i=0; i<array.length;i++)
13       for(j=0; j<array[i].length;j++)
14         System.out.printf("%3d%s",array[i][j],
15                                   j<(array[i].length-1)?", ":"\n");
16
17     loopi:
18     for (i=0; i<array.length;i++)
19       for(j=0; j<array[i].length;j++)
20         if (array[i][j]==7)
21         {
22         System.out.println("7 found in Line "+i);
23         continue loopi;
24         }
25   }
26 }
```

Ein 2-dimensionales Array anlegen

Mit Zufallszahlen füllen

Zeilenweise ausgeben

Finde Zeilen, in denen die 7 vorkommt

```
$ java ContinueLabel
  8,   1,   7,   9
  0,   8,   7,   3
  0,   6,   6,   9
7 found in Line 0
7 found in Line 1
```

Exception handling

- try

  öffnet einen Block, in dem Ausnahmesituationen auftreten können

- catch
  - fängt eine in einem try-Block aufgetretene Exception auf. Mit dem Code im cacht-Block kann die Exception ausgegeben oder es kann verscuht werden, die Situation so behandeln, dass eine sinnvolle Forführung des Programms nöglich ist.
  - Zu einem try-Block kann es viele catch-Blöcke geben.
  - Wurde eine Exception durch einen catch behanelt, wird die try catch-Konstruktion verlassen oder finally ausgeführt.

Exception handling

- finally
  - Finally-Block am Ende von try-catch wird immer ausgeführt, wenn der try-Block betreten worden ist.
- throws
  - Leitet die angegebenen Exceptions an den Aufrufer weiter.
- throw
  - Wirft eine Exception, das führt zum Verlassen des try-Blocks und Verzweigung zu einem passenden catch oder zum Verlassen der Funktion oder zum Beeden des Programms.

Try / catch

```
class tryDemo
{
  public static void main(String args[])
  {
    try
    {
      int array[]={1,2};
      int i=Integer.parseInt(args[0]);
      System.out.println("Array["+i+"]="+array[i]);
    } catch (IndexOutOfBoundsException e1)
    {
       System.out.println("myException: "+e1);
       e1.printStackTrace();
    } catch (NumberFormatException e2)
    {
       System.out.println("myException: "+e2);
       e2.printStackTrace();
    }
  }
}
```

Weiterleiten von Exceptions

```
class thowsDemo
{
 public static void main(String args[])throws Exception
  {
      int array[]={1,2};
      int i=Integer.parseInt(args[0]);
      System.out.println("Array["+i+"]="+array[i]);
  }
}
```