

Welcome to java

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

Einfach?

Objektorientiert?

In Java ist alles in Klassen und Objekten

Verteilt?

Java interpretiert die Bytecodes in seiner
in seiner Netze

Interpretierend?

Secure?

Vor allem

Sicherheit

Java

richtet sich

(Sie sind

diesem

pa

Das

Nebenläufigkeit

innerhalb

eines

Javaprozesses

ist

durch

Architekturneutral?

Datentypen in c/c++ orientieren sich

An der zu Grunde liegenden Plattform

In Sachen Verarbeitungsbreite / Byteorder

Byteorder? Unklar? Mal recherchieren!

In java sind die Datentypen für alle

Plattformen einheitlich.

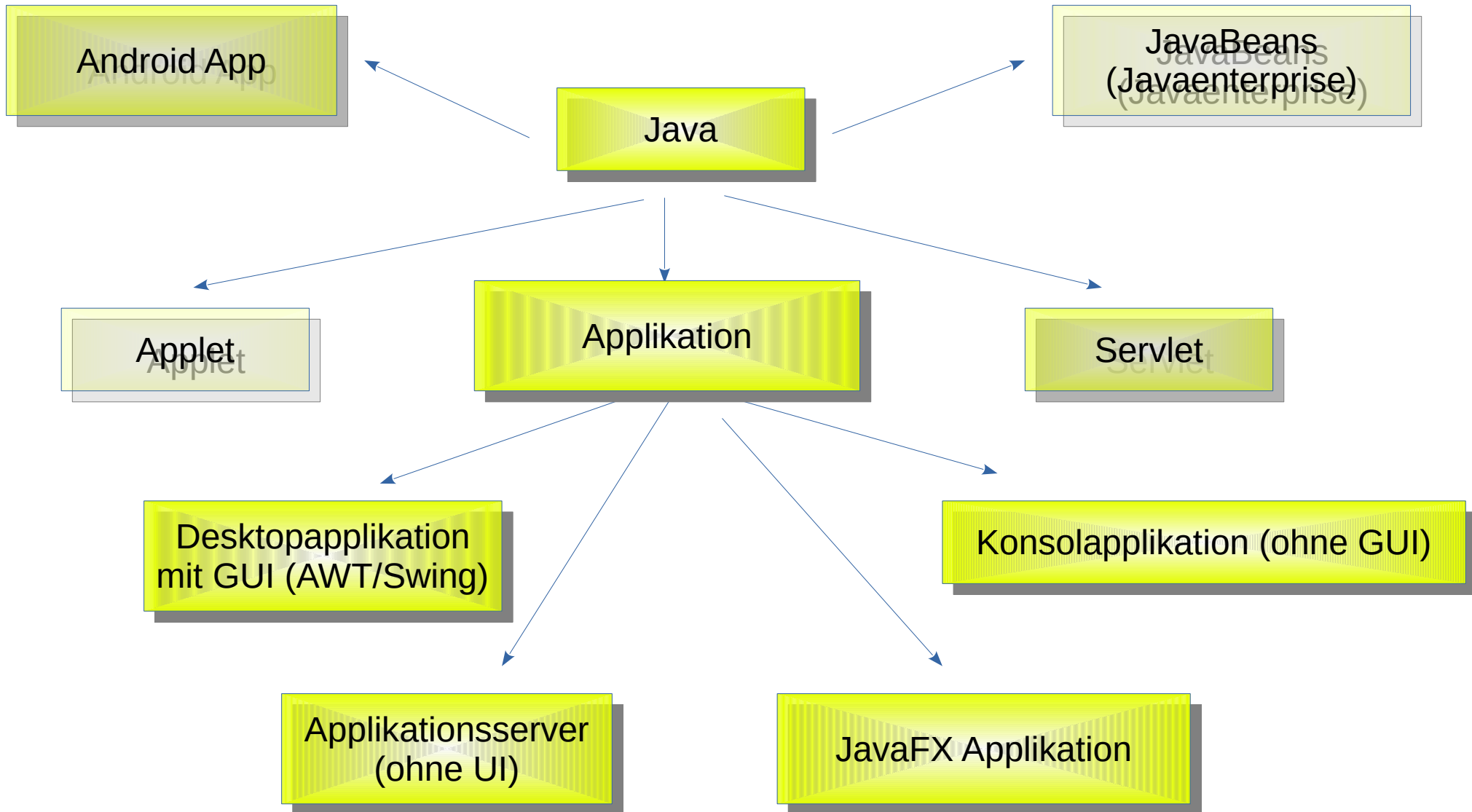
Das

Nebenläufigkeit innerhalb eines Javaprozesses

ist durch die Klassenbibliothek und die Sprache

Integriert.

Vielfalt von Java-Anwendungen



Es geht los!

- Javaquelltext ist immer in sogenannte Klassen eingeschlossen.
- Die Bedeutung einer solchen Klasse werden wir später ergründen.
- Zunächst gilt, dass Variablen und Funktionen von einem Rahmen umgeben sind, der mit `class` und einem Bezeichner eingeleitet wird.

```
class EinName
{
    // hier kommt jetzt alles rein
}
```

Es geht los!

Vieles kennen wir von c bereits:

- Syntax in Funktionen, Anweisungen und Ausdrücken
- Die meisten Operatoren werden ähnlich verwendet.
- Viele Statements sind ähnlich aufgebaut.
- Primitive Datentypen sind ähnlich denen von c.

Erstes Java-Programm

```
// 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);
    }
}
```

Javodateien 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

Erstes Java-Programm V2.0

Javodateien 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");
    }
}
```

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:

Führen Sie das Programm oben in einer
Konsole aus:

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

```
#include <stdio.h>
```

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

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 der immer 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 und liefert einen booleschen Wert.
- 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.
- Es gibt keine Pointer, nur Referenzen.
- 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 floating point	0.0	32 bit	$\pm 3.40282347E+38$... $\pm 1.40239846E-45$
double	IEEE 754 floating point	0.0	64 bit	$\pm 1.79769313486231570E+308$... $\pm 4.94065645841246544E-324$

Eingabekonvertierung

- Konvertierung erfolgt über Methoden der wrapper classes aus java.lang
- Umwandlung String → numerischen Wert
- In c: atoi, strtol, strtod
- In Java: Beispielsweise Integer.parseInt, Double.parseDouble, Long.parseLong usw.

```
int i;
```

i enthält als Ergebnis den Integerwert

```
// statische Methode
```

```
i=Integer.parseInt(args[0]);
```

```
// Member function Methode
```

```
i=new Integer(args[0]) . intValue();
```

String, der Ziffern enthält

wrapperclasses

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

Suchen Sie in der API-Dokumentation
[http://www.informatik.htw-dresden.de/~beck/PSPII_WI/](http://www.informatik.htw-dresden.de/~beck/PSPII_WI/API_Java8)
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);
```



printf, wie in C

Statements

Expressionstatement

if-statement

while-statement

do-while-statement

for-statement

switch-statement

break / continue -statement

Synchronized-statement

Exceptionhandling

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 (wird eher selten gebraucht)

```
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);
}
```

Die foreach-Konstruktion iteriert durch alle Element einer Datacollection. Außer Arrays gibt es weitere. Nachteilig kann sein, dass man den aktuellen Index nicht kennt.

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

Exceptionhandling dient dem Auffangen und Behandeln von Ausnahme- und Fehlersituationen. Diese Situationen führen normalerweise zum Programmabbruch. Mit Exceptionhandling können Maßnahmen zum sinnvollen Weiterlaufen des Programms ergriffen, oder der Fehler dokumentiert werden.

Exceptionhandling

- try / catch / finally

Try-catch bildet eine Klammer um eine Programmpassage, in der Fehlersituationen erwartet werden und die deshalb daraufhin überwacht wird. Tritt eine solche ein, wird die überwachte Region verlassen und eine geeignete catch-Klausel ausgeführt. Gibt es diese nicht, wird die Exception an die Virtuelle Maschine weitergeleitet, die in der Regel das Programm abbricht.

Exceptionhandling

- throws

Leitet eine Exception, wenn sie auftritt, an die aufrufende Funktion weiter. Die aktuelle Funktion wird dabei verlassen. Die Exception muss beim Aufrufer dann via try-catch behandelt oder wiederum weitergeleitet werden. Erfolgt dies nicht, landet die Exception in der virtuellen Maschine und führt dann zum Programmabbruch.

- throw

Erzeugt (wirft) eine Exception, um einen eingetretenen Fehlerzustand zu signalisieren.

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();
        }
    }
}
```

Überwacher
Bereich

Über die Kommandozeile kann ein Argument, das keine Zahl ist, übergeben worden sein. Die übergebene Zahl kann ungleich 0 oder 1 sein und damit zu einem Indexfehler führen

Weiterleiten von Exceptions

```
class throwsDemo
{
    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]);
    }
}
```