

# Welcome to java

# Welcome to java

- einfach

# Welcome to java

- einfach

Einfach?

Programmieren ist für viele nicht einfach!  
Aber ein Großteil der Syntax von Ausdrücken  
und Anweisungen orientiert sich an c.  
Techniken, an denen viele scheitern, sind in  
Java eliminiert (Pointer, malloc)  
Von daher vielleicht doch etwas einfacher  
als c/c++

# Welcome to java

- einfach

# Welcome to java

- einfach
- objektorientiert

# Welcome to java

- einfach
- objektorientiert

Objektorientiert?

In Java ist alles in Klassen und Objekten gekapselt. Was eine Klasse ist, werden wir in der nächsten Präsentation erarbeiten.

# Welcome to java

- einfach
- objektorientiert

# Welcome to java

- einfach
- objektorientiert
- verteilt



# Welcome to java

- einfach
- objektorientiert
- verteilt

Verteilt?

Java integriert Netzwerkprogrammierung in seine Bibliotheken. Somit können mit Java Netzanwendungen (Client-/Server) einfach gebaut werden.

Nicht nur Daten, sondern auch Code kann dazu über Netzwerke übertragen werden.

# Welcome to java

- einfach
- objektorientiert
- verteilt

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend

Interpretierend?

Der Javacompiler erzeugt Code für eine virtuelle Maschine. Javaprogramme werden immer mit Hilfe dieser virtuellen Maschine ausgeführt. Das Vor- und Nachteile. Die Verwendung der Virtuellen Maschine verlangsamt die Ausführung, Bringt aber Vorteile in Sachen Portabilität (Ausführbarkeit auf verschiedenen Plattformen) und Komfort zur Laufzeit.

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust

Robust?

Die virtuelle Maschine nimmt dem Programmierer viel ab, z.B. bei der Speicherverwaltung.

Überwachung von Ausnahmen gehört fest zur Javaprogrammierung. Darum wird Java gern für hochverfügbare Anwendungen genutzt.

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust



# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure

Secure?

Vor allem in Netzwerkanwendungen spielt Sicherheit eine große Rolle.

Java verfügt über eine Reihe von Klassen für sicherheitsrelevante Aspekte.

(Sie sind nicht Gegenstand unserer LV in diesem Semester)

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral

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.

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel

Portabel?

Javaprogramme können mit einem SDK (SoftwareDevelopmentKit) auf einer beliebigen Plattform gebaut werden und sind auf jeder anderen Plattform, für die es eine virtuelle Maschine gibt, ausführbar.



# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel
- schnell

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel
- schnell

Schnell?

Das ist sehr relativ. Javaprogramme mit einem Swing-GUI sind definitiv nicht schnell. Serveranwendungen mit java aber schon.

# Welcome to java

- einfach
- objektorientiert
- verteilt
- interpretierend
- robust
- secure
- architekturneutral
- portabel
- schnell

# Welcome to java

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

# Welcome to java

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

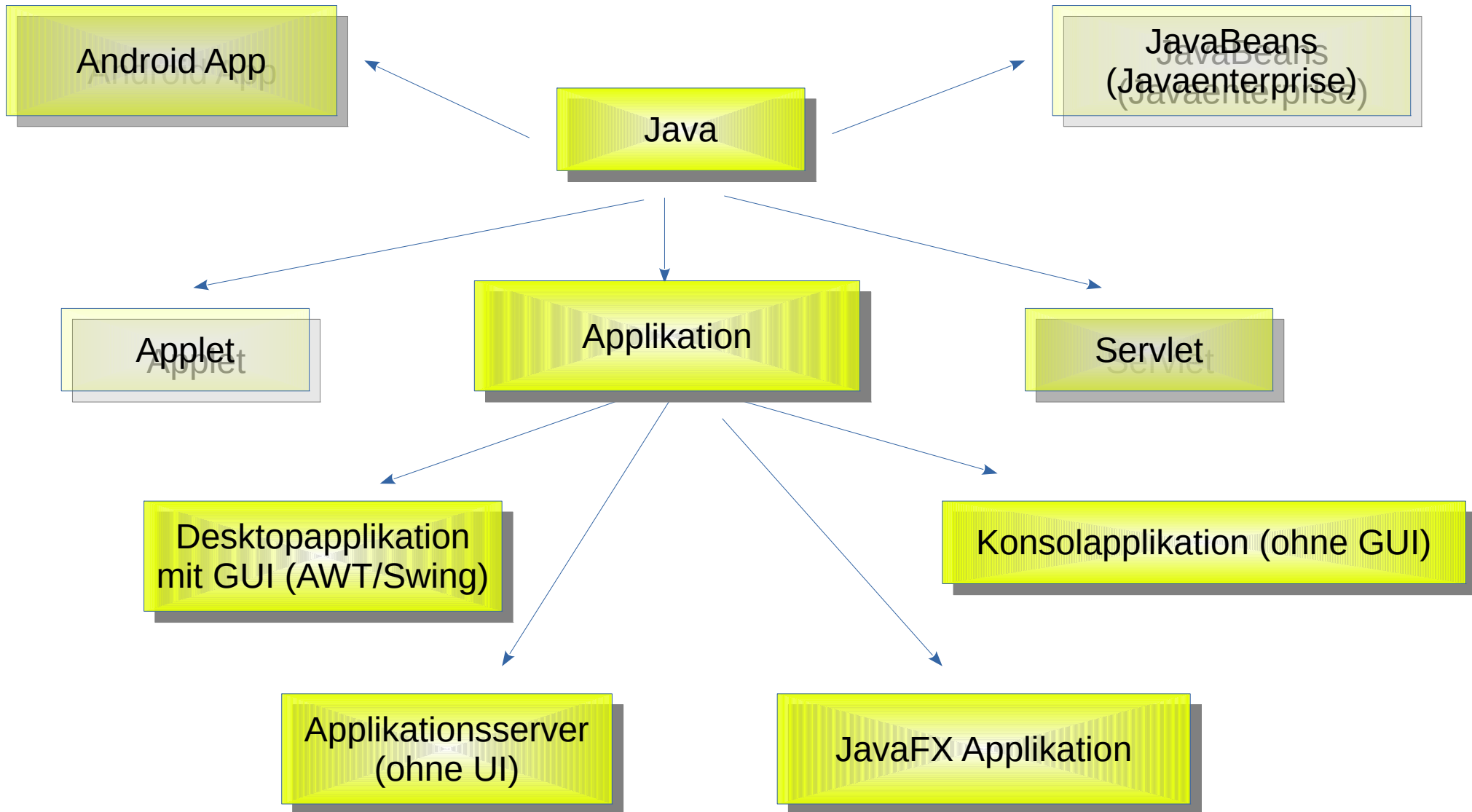
parallel?

Das Konzept „Threads“ zur Programmierung von Nebenläufigkeit innerhalb eines Javaprozesses ist durch die Klassenbibliothek und die Sprache integriert.

# Welcome to java

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

# Vielfalt von Java-Anwendungen





# 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**

# Die Werkzeuge

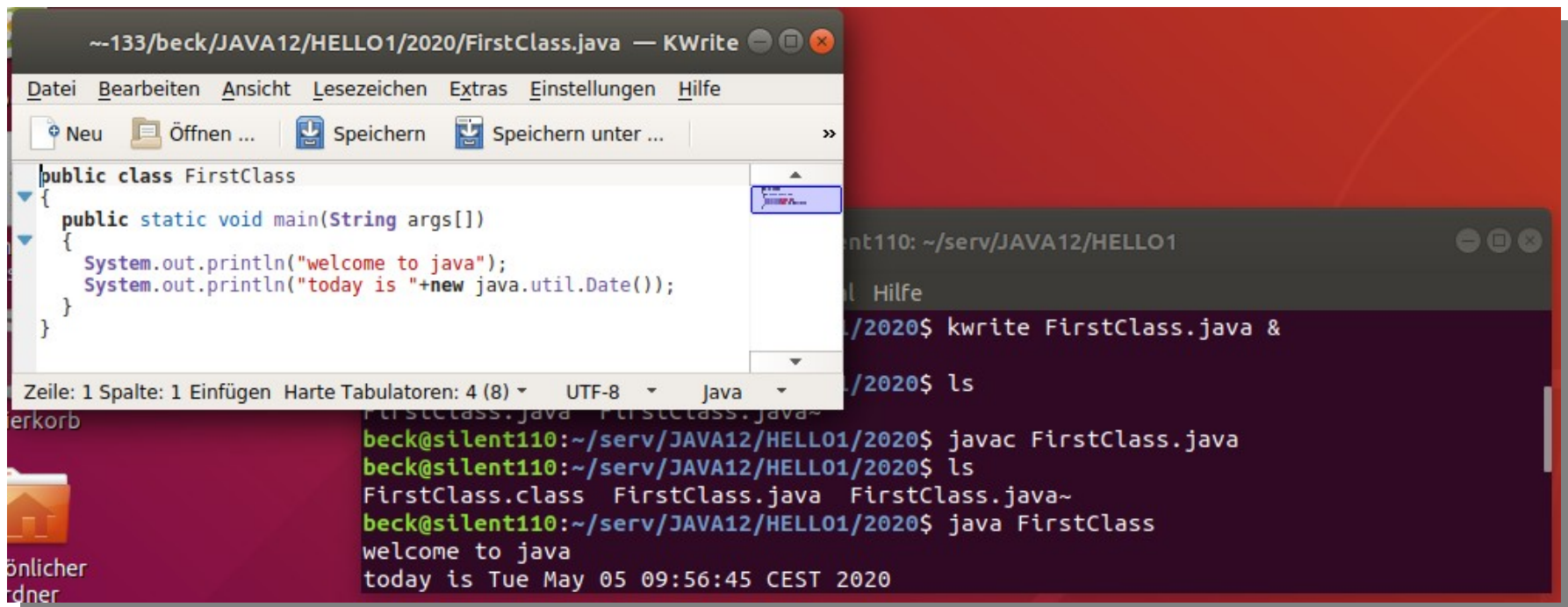
- In der Konsole:
  - Programmmeditor (kwrite, kate, atom ....)
  - Java Development Kit (JDK)
- IDE
  - IntelliJ IDEA, eclipse, netbeans ...

# 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 (FirstClass.class)



The screenshot shows a KWrite editor window titled '~133/beck/JAVA12/HELLO1/2020/FirstClass.java' containing the following Java code:

```
public class FirstClass
{
    public static void main(String args[])
    {
        System.out.println("welcome to java");
        System.out.println("today is "+new java.util.Date());
    }
}
```

Below the editor is a terminal window with the following commands and output:

```
beck@silent110: ~/serv/JAVA12/HELLO1
~/2020$ kwrite FirstClass.java &
~/2020$ ls
FirstClass.class FirstClass.java FirstClass.java~
~/2020$ javac FirstClass.java
~/2020$ ls
FirstClass.class FirstClass.java FirstClass.java~
~/2020$ java FirstClass
welcome to java
today is Tue May 05 09:56:45 CEST 2020
```

# 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 (hauptsächlich die Klassen) eines Javaprogramms in einer Archivdatei zusammen.

```
jar cfm FirstClass.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

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

```
class FirstClassV2
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        int i=2;
```

```
        System.out.println(
```

```
            "Willkommen im "+i+". Semester");
```

```
    }
```

```
}
```

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

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:

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 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 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 (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();
```

# 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();
```

String, der Ziffern enthält

# 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();
```

String, der Ziffern enthält

# 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.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

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.

# 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

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/](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);
```

printf, wie in C

Wie sprintf in c

```
String s;
```

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

# 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 Pendanten 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



```

public class BreakLabel {
    public static void main(String[] args)
    {

        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
            };

        int searchfor = Integer.parseInt(args[0]);

        int i;
        int j = 0;
        boolean foundIt = false;

```

Quelle:  
<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html>

### search:

```

    for (i = 0; i < arrayOfInts.length; i++)
    {
        for (j = 0; j < arrayOfInts[i].length; j++)
        {
            if (arrayOfInts[i][j] == searchfor)
            {
                foundIt = true;
                break search;
            }
        }
    }

    if (foundIt)
    {
        System.out.println("Found "
            + searchfor
            + " at " + i + ", " + j);
    } else
    {
        System.out.println(searchfor
            + " not in the array");
    }
}
}

```

# 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 catch-Block kann die Exception ausgegeben oder es kann versucht werden, die Situation so behandeln, dass eine sinnvolle Fortführung des Programms möglich ist.
- Zu einem try-Block kann es viele catch-Blöcke geben.
- Wurde eine Exception durch einen catch behandelt, 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 Beenden 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 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]);
    }
}
```