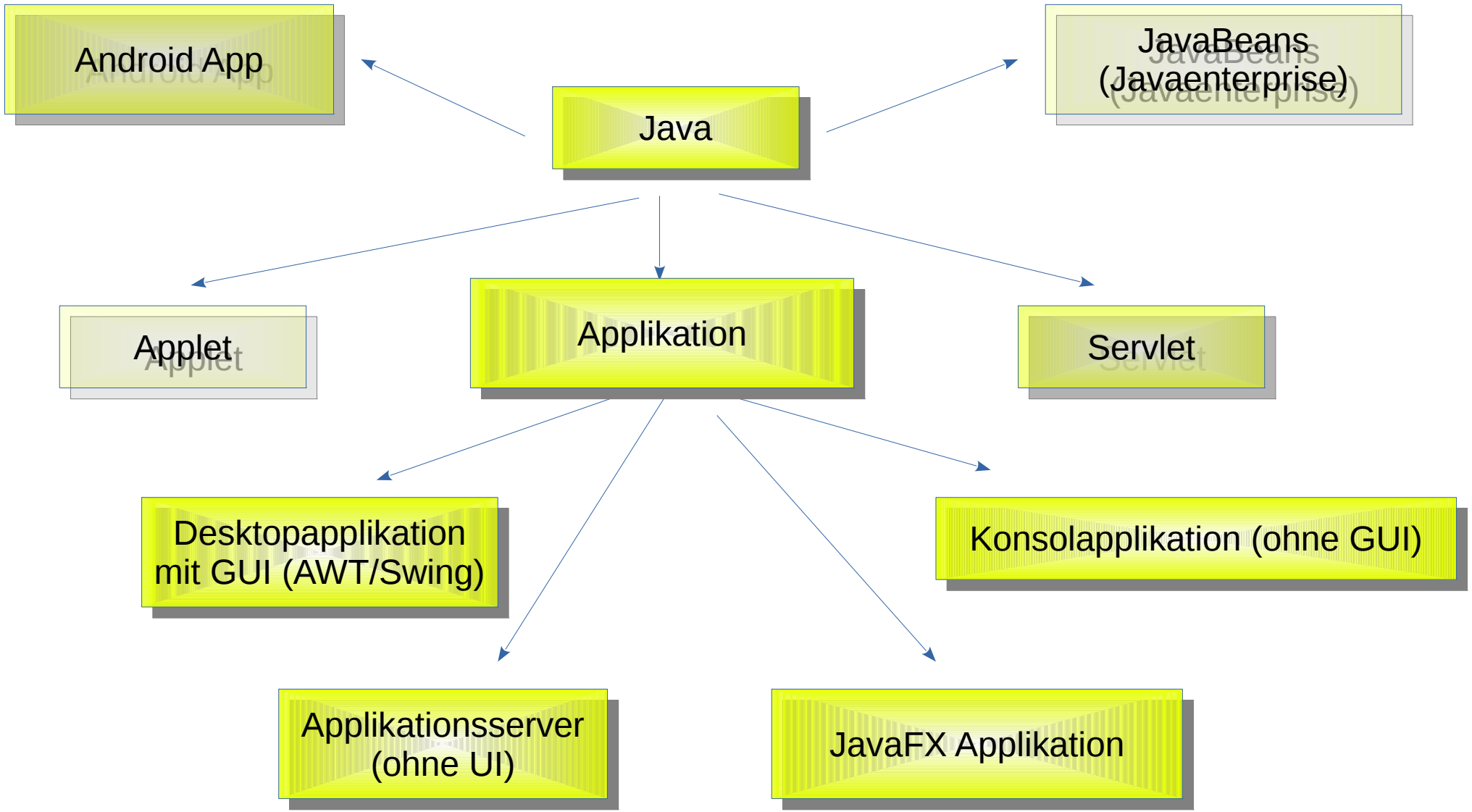
Welcome to java

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

Vielfalt von Java-Anwendungen



Erstes Java-Programm

Dateiname: Klassenname.java
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);
  }
}
```

Erstes Java-Programm V2.0

Dateiname: Klassenname.java
In diesem Fall:
FirstClass.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:

```
#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 Nullen 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 eingebaute Datentypen angewandt, wird ein gewöhnlicher Vergleich ausgeführt.
- Keine Poiner (&, *, ->)
- 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
- 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();
```

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

Ausgabe / Konvertierung numerischer Werte

Aufruf von toString der zugehörigen wrapperclass

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

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);
```

Statements

Expressionstatement

if-statement

while-statement

do-while-statement

for-statement

switch-statement

break / continue -statement

synchronized-statement

Eingabe von Standardeingabe

```
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

```
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 im Initialisierungsausdruck

```
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;

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 / catch / finally
- throws
- throw

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]);
  }
}
```