Technische Universität Braunschweig

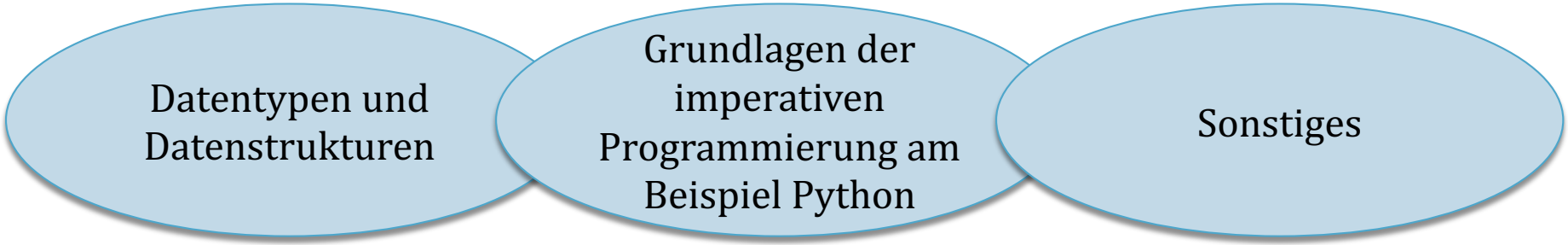
Programmieren 1 – Vorlesung #3

Arne Schmidt

Wiederholung

Rückblick

Was haben wir bisher gelernt?



Integer
Float
Boolean
Strings
Arrays

Kontrollstrukturen
- Anweisung
- Verzweigung
- Schleifen
- Methoden

Unäre / binäre Operatoren
Binärzahlen
Scope
Pass by Value

Heute

Einführung in Java
- Lexik
- Syntax
- Datentypen

Kapitel 3 – Einführung in Java

Java

Im Gegensatz zu Python muss Java kompiliert und dann interpretiert werden.

Dies geschieht mit den Befehlen
Kompilieren: javac MyClass.java
Interpretieren: java MyClass

Der Dateiname ist identisch mit dem *Klassen*bezeichner in der Datei , welche public ist.

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

Java – Was?

```
public class MyClass {
   public static void main(String args[]) {
      System.out.println(“Hello Audimax!”);
   }
}
```

class?

Static?

Public?

void?!

System?

args[]?

Keine Sorge, wir gehen dort Schrittweise durch.
Nehmt zunächst einfach an: die ersten und letzten zwei Zeilen werden benötigt, damit es kompiliert

Kapitel 3.1 – Lexik

Javas Lexik

Kommentare:
Einzeilig: // Text
Mehrzeilig: /* Text */
Dokumentkommentar: /** Text */

Trennzeichen:
Leerzeichen, Zeilenendzeichen (ENTER-Taste), Tabulatorzeichen (TAB-Taste)

Operatoren:
+, -, *, /, %, ==, +=, -=, >=, <=, !, >>, <<, |, &, ^, ...

Interpunktion:
., (,), {, }, [,]

** als Exponentiation gibt es nicht.
^ ist immer noch bitweises xor.

Javas Lexik - Datentypen

Typ	Größe	Wertebereich	Beispiel
byte	1 Byte	-128 bis 127	byte b = 42
short	2 Byte	-32 768 bis 32 767	short s = 4414
int	4 Byte	-2^{31} bis $2^{31} - 1$	int i = 12345
long	8 Byte	-2^{63} bis $2^{63} - 1$	long l = 18_293_194_530L
float	4 Byte	nach IEEE 754	float f = 3.14f
double	8 Byte	nach IEEE 754	double d = 9.3
boolean	??	true / false	boolean b = false
char	4 Byte	‘\u0000’ bis ‘\uffff’	char c = ‘p’

Zeichenkodierung

Eben gesehen, ein Char kann den Wert ‘\u0000’ annehmen. Das wirft Fragen auf!

- Was bedeutet das?
- Wie wird überhaupt der Quellcode codiert? Letztlich stehen dort nur 0en und 1en!
- Wie muss eine Datei interpretiert werden?

Beispiel ASCII (American Standard Code for Information Interchange) mit 7 Bits pro Zeichen:

Binärzahl: 101 0111 Zeichen: W

Unicode benutzt 16 Bit pro Zeichen und umfässt knapp 150 000 Zeichen.

Java liest Dateien per Unicode aus (default).

ASCII Tabelle

$b_7 b_6 b_5$ / Bits					000	001	010	011	100	101	110	111
b_4	b_3	b_2	b_1	Row / Column	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	’	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Javas Lexik - Literale

Typ	Größe	Wertebereich	Beispiel (typ bez. = literal)
byte	1 Byte	-128 bis 127	byte b = 42
short	2 Byte	-32 768 bis 32 767	short s = 4414
int	4 Byte	-2^{31} bis $2^{31} - 1$	int i = 12345
long	8 Byte	-2^{63} bis $2^{63} - 1$	long l = 18_293_194_530L
float	4 Byte	nach IEEE 754	float f = 3.14f
double	8 Byte	nach IEEE 754	double d = 9.3
boolean	??	true / false	boolean b = false
char	4 Byte	‘\u0000’ bis ‘\uffff’	char c = ‘p’

Java – Schlüsselwörter (bereits bekannt)

assert	enum	native	throw
boolean	extends	new	throws
break	final	package	transient
byte	finally	private	try
case	**float**	protected	*void*
catch	for	public	volatile
char	goto	return	while
class	if	**short**	
const	implements	static	
continue	import	strictfp	
default	instanceof	super	
do	**int**	switch	
double	interface	synchronized	
else	**long**	this	

Java – Schlüsselwörter (Kapitel 4 – Objekte)

assert	enum	native	throw
boolean	**extends**	**new**	throws
break	**final**	**package**	transient
byte	finally	**private**	try
case	float	**protected**	void
catch	for	**public**	volatile
char	goto	return	while
class	if	short	
const	**implements**	**static**	
continue	import	strictfp	
default	**instanceof**	**super**	
do	int	switch	
double	**interface**	synchronized	
else	long	**this**	

Java – Schlüsselwörter (Kapitel 6 – Testen)

assert
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
enum
extends
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
try
void
volatile
while

Kapitel 3.2 – Datentypen und Operatoren

Datentypen - Typecast

Da die Typisierung in Java nicht automatisch stattfindet, muss man immer darauf achten, kompatible Datentypen für Operatoren zu nutzen.

Das geschieht per **Typecast:**

Explizit:
```
int i = (int) 3.0;     //Aus dem double-Wert 3.0 wird ein Integer-Wert 3

int i = Integer.parseInt("483");   //Die Zahl 483 wird aus dem String gelesen
String s = String.valueOf(48.03); //Zahlenwert wird zu String konvertiert
```

Implizit:
```
double d = i / 3;    //Das Ergebnis wir implizit zu double gecastet
String s2 = i + "";
```

Operatoren und Reihenfolgen

Operatoren für Zahlentypen sind sehr ähnlich wie in Python.

Aber: int / int ergibt int (Ergebnis wird abgerundet)

Sollte man sich nicht sicher sein, welche Priorität Operatoren besitzen: Ausdrücke klammern!

Beispiele:

```
double x = 3.0 / 2.0;
double y = 3 / 2;
double z = x+y ;
String s = “x+y = “ + z;
```

x = 1.5
y = 1.0
z = 2.5
s = “x+y = 2.5”

Welchen Wert besitzt s?

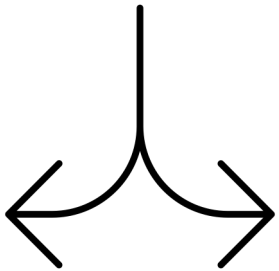
Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

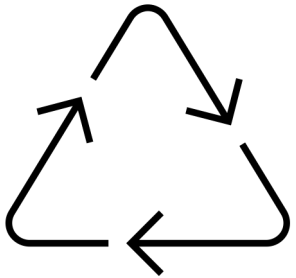
Kapitel 3.3 – Kontrollstrukturen

Kapitel 3.3.1 – Anweisungen, Verzweigungen und Schleifen

Anweisungen

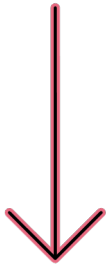


Verzweigungen

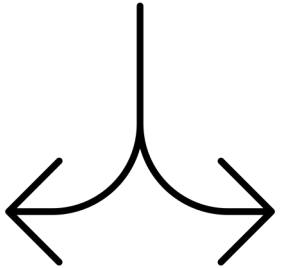


Schleifen

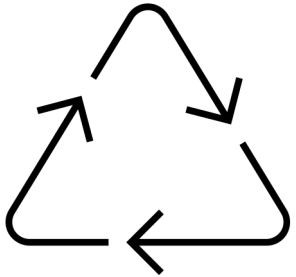
Kontrollstrukturen



Anweisungen



Verzweigungen



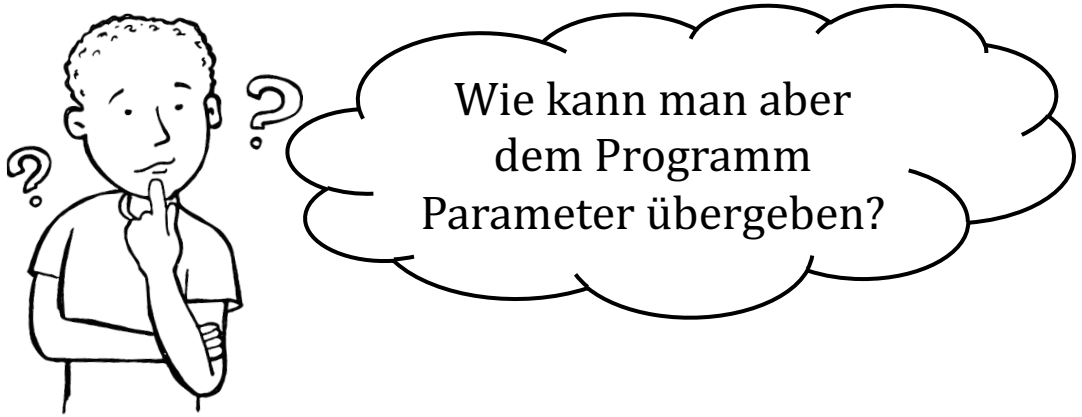
Schleifen

Anweisungen

In Java enden Anweisungen immer mit einem Semikolon (;).

Zur Deklaration einer Variablen **muss** der Datentyp mit angegeben werden.

Zur Ausgabe auf die Konsole kann die Methode System.out.println(String) genutzt werden.
Anweisungen zur Eingabe während der Laufzeit lassen wir zunächst aus.



Eingabe von Parametern

Um dem Programm Input zu geben, können über die Kommandozeile Parameter übergeben werden, welche über **args[i] in der main-Methode** abgerufen werden können

```
public class MyClass {
   public static void main(String[] args) {
      System.out.println(args[0]);
   }
}
```

```
(base) Arnes-MacBook-Pro:Prog1 schmidt$ javac MyClass.java
(base) Arnes-MacBook-Pro:Prog1 schmidt$ java MyClass 333
333
```

Eingabe von Parametern – Zahlen

Um die Eingabe, welche immer ein String ist, in eine Zahl umzuwandeln, stehen bspw. folgende Methoden zur Verfügung:

Integer.parseInt(String s): Konvertiert den String s in einen int.

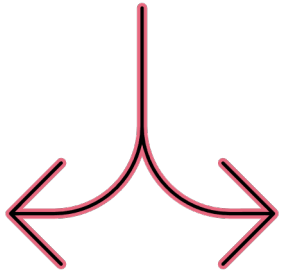
Double.parseDouble(String s): Konvertiert den String s in einen double.

Beispiel mit Aufruf `java MyClass 12 3.141`:

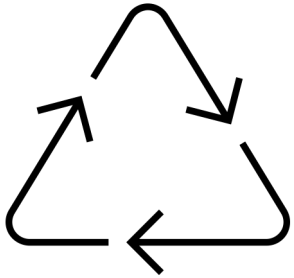
```
int a = Integer.parseInt(args[0]);
double d = Double.parseDouble(args[1]);
```

Kontrollstrukturen

Anweisungen



Verzweigungen



Schleifen

If-Else und Switch-Case

Der Test bzw. die Variable, über welche verzweigt wird, steht stets in Klammern.

Die Anweisungen danach werden mit geschweiften Klammern umschlossen.

Damit ist Einrückung nicht nötig, aber trotzdem empfohlen für bessere Lesbarkeit!

Die Anweisungen für einen case folgenden nach einem Doppelpunkt und enden immer mit break;

default wird ausgeführt, wenn kein anderer Fall eintrat.

```
if (Test) {
   Anweisungen;
} else {
   Anweisungen;
}
```

```
if (Test) {
   Anweisungen;
} else if (Test 2) {
   Anweisungen;
} else {
   Anweisungen;
}
```

```
switch (variable){
   case Wert_1:
      Anweisungen;
      break;
   case Wert_2:
      Anweisungen;
      break;
   case Wert_3:
      Anweisungen;
      break;
   default:
      Anweisungen;
      break;
}
```

If-Else-Alternative: Ternärer Operator

Java bietet einen ternären Operator (?:), welcher einen bedingten Wert zurück gibt.

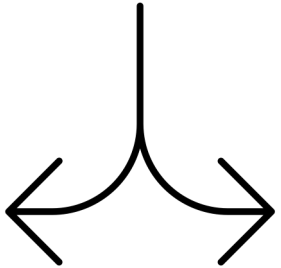
Syntax ternärer Operator:
(Test) ? (Wert bei true) : (Wert bei False)

Beispiel:
int a = -5
String s = “Ist a negativ?” + (**a < 0 ? “Ja” : “Nein”**);

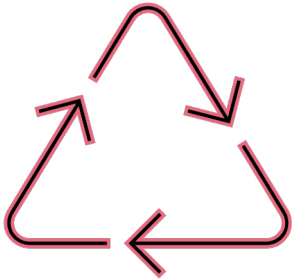
Dieser Operator ist nur mit Bedacht zu nutzen!
Nur für sehr einfache und unkomplizierte Anweisungen, ansonsten wird es schnell unübersichtlich.

Kontrollstrukturen

Anweisungen



Verzweigungen



Schleifen

For-Schleife

Java bietet zwei Möglichkeiten für for-Schleifen an.

For-Schleife:

```
for (Init; Test; Inkrement){
   Anweisungen;
}
```

ForEach-Schleife:

```
for (type var : arrayname){
   Anweisungen;
}
```

Beispiele:

```
int n = 10;
int sum = 0;
for (int i = 1; i <= 10; ++i){
   sum += i;
}
```

```
String[ ] obst = {“Apfel”, “Birne”, “Banane”};
for (String s : obst){
   System.out.println{s};
}
```

(Arrays schauen wir uns nächste Woche genauer an)

While- und Do-While-Schleifen

While-Schleife
```
while (Bedingung){
   Anweisung;
}
```

```
int x = 8;
int f = 1;
while (x > 0){
   f *= x;
   x -= 1;
}
System.out.println("8! = " + f);
```

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

```
int x = 8;
int f = 1;
do {
   f *= x;
   x -= 1;
} while (x > 0);
System.out.println("8! = " + f);
```

Wichtig: Do-while führt mind. eine Iteration aus!

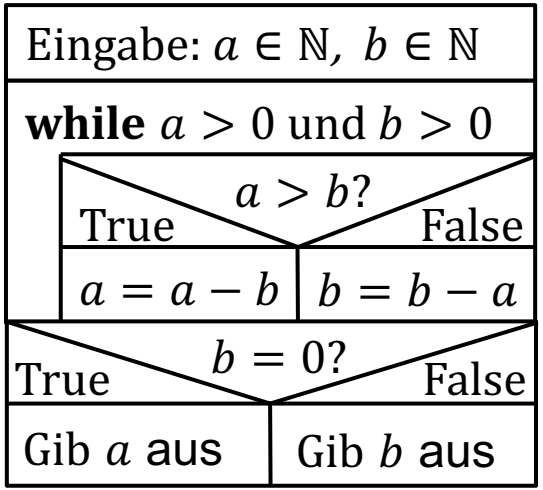
Break und Continue

Break und Continue besitzen die gleiche Bedeutung wie in Python!

Was bewirken noch einmal break und continue?

Gesamtbeispiel

Als Struktogramm:



Als Java Programm:

```
public class MyClass {
   public static void main(String[] args) {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);

      while (a > 0 && b > 0){
         if (a > b){
            a -= b;
         } else {
            b -= a;
         }
      }
      System.out.println(b == 0 ? a : b);
   }
}
```

Kapitel 3.3.2 –Methoden

Methoden

Eine Methode in Java wird allgemein definiert durch *Rückgabetyp Name(Inputparameter).*

Dazu ist zu beachten:

- Parameter werden als Pass-by-Value übergeben (wie bei Python).
- Rückgabetyp ist void, wenn kein Rückgabewert vorgesehen ist.
- Methoden, die von einer Methode mit static-*Modifizierer* aufgerufen werden und kein Teil eines *instanziierten Objektes* sind, benötigen den Modifizierer static.

Methoden – Beispiel

```
public class MyClass {

   static int pow(int x, int p){
      int res = 1;
      for (int i = 0; i < p; ++i){
         res *= x;
      }
      return res;
   }

   public static void main(String[] args) {
      int base = Integer.parseInt(args[0]);
      int exponent = Integer.parseInt(args[1]);

      System.out.println(pow(base, exponent));
   }
}
```

Methoden – Signatur

Die **Signatur** eine Methode ist die Kombination von Methodennamen und der Ordnung der Parametertypen.
Zwei Methoden sind unterscheidbar, wenn sich ihre Signaturen unterscheiden (d.h. der Methodenname von zwei Methoden darf identisch sein).

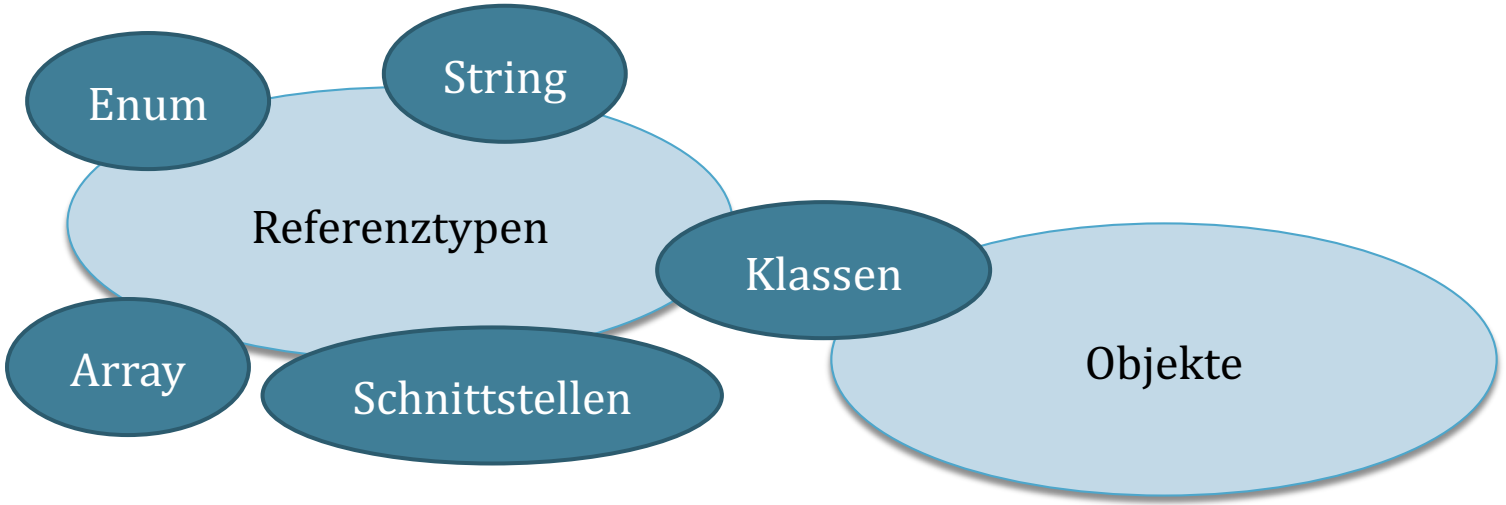
```
public class MyClass {

   static int pow(int x, int p){
      int res = 1;
      for (int i = 0; i < p; ++i){
         res *= x;
      }
      return res;
   }

   public static void main(String[] args) {
      int base = Integer.parseInt(args[0]);
      int exponent = Integer.parseInt(args[1]);
      System.out.println(pow(base, exponent));
   }
}
```

Signatur von pow: pow(int, int)

Ausblick


Enum
String
Referenztypen
Klassen
Objekte
Array
Schnittstellen