

Lektion 2

Variablen im Speicher, primitive Datentypen, Ausgabe

Wie werden in Java
Daten im Speicher gehalten?

Für eine ganze Zahl
werden in Java 32 Bits vorgesehen,
d.h. 32 Stellen,
wobei jede Stelle unabhängig voneinander
zwei Zustände einnehmen kann: 0 oder 1.

Darstellung ganzer Zahlen in versch. Zahlensystemen

in der Mathematik

Dezimal	Binär	Hexadezimal	Oktal
0	0 ₂	0 ₁₆	0 ₈
1	1 ₂	1 ₁₆	1 ₈
2	10 ₂	2 ₁₆	2 ₈
3	11 ₂	3 ₁₆	3 ₈
4	100 ₂	4 ₁₆	4 ₈
5	101 ₂	5 ₁₆	5 ₈
6	110 ₂	6 ₁₆	6 ₈
7	111 ₂	7 ₁₆	7 ₈
8	1000 ₂	8 ₁₆	10 ₈
9	1001 ₂	9 ₁₆	11 ₈
10	1010 ₂	A ₁₆	12 ₈
11	1011 ₂	B ₁₆	13 ₈
12	1100 ₂	C ₁₆	14 ₈
13	1101 ₂	D ₁₆	15 ₈
14	1110 ₂	E ₁₆	16 ₈
15	1111 ₂	F ₁₆	17 ₈

in Java

Dezimal	Binär	Hexadezimal	Oktal
0	0b0	0x0	00
1	0b1	0x1	01
2	0b10	0x2	02
3	0b11	0x3	03
4	0b100	0x4	04
5	0b101	0x5	05
6	0b110	0x6	06
7	0b111	0x7	07
8	0b1000	0x8	010
9	0b1001	0x9	011
10	0b1010	0xA	012
11	0b1011	0xB	013
12	0b1100	0xC	014
13	0b1101	0xD	015
14	0b1110	0xE	016
15	0b1111	0xF	017

Zahlenumwandlungsbeispiele

$$\begin{aligned} 57_{10} &= 32 + 16 + 8 + 1 \\ &= 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 111001_2 \end{aligned}$$

$$\begin{aligned} 111001_2 &= 0011|1001_2 \\ &= 39_{16} \end{aligned}$$

$$1011100_2 = 0101|1100_2 = 5C_{16}$$

primitive Datentypen für ganze Zahlen

Datentyp	Größe	Zahlenbereich	Zahlenbereich
byte	8 Bits	-2^7 bis $2^7 - 1$	-128 ... 127
short	16 Bits	-2^{15} bis $2^{15} - 1$	-32768 ... 32767
int	32 Bits	-2^{31} bis $2^{31} - 1$	-2147483648 ... 2147483647
long	64 Bits	-2^{63} bis $2^{63} - 1$	-9223372036854775808 ... 9223372036854775807

8 Bits entsprechen 1 Byte

char	16 Bits	0x0000 ... 0xFFFF	Zur Speicherung einzelner Buchstaben. Wird durch Zahlencodes repräsentiert
------	---------	-------------------	-------------------------------------------------------------------------------

Darstellung ganzer Zahlen: Zweierkomplement

- Ganze Zahlen (byte, short, int, long) werden in Java im Zweierkomplement dargestellt.
- Beim Zweierkomplement legt das erste Bit einer Zahl fest, ob die Zahl als negative Zahl interpretiert wird, d.h. bei einem byte:

Binär	Dezimal	Umrechnung
0000 0110	6	$0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ $= 4 + 2 = 6$
1000 0110	-122	$-1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ $= -128 + 6 = -122$

Was passiert, wenn ich die 6 in eine -6 umwandeln will?

Darstellung ganzer Zahlen: Zweierkomplement

- Um eine positive Zahl in eine negative Zahl umzuwandeln, muss die Binärzahl invertiert werden und danach „1“ addiert.

Binär	Dezimal	Umrechnung
0000 0110	6	$0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ $= 4 + 2 = 6$
1111 1001	-7	$-1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ $= -128 + 64 + 32 + 16 + 8 + 1$ $= -128 + 121 = -7$
+0000 0001	1	$0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ $= 1$
1111 1010	-6	$-1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ $= -128 + 64 + 32 + 16 + 8 + 2$ $= -128 + 122 = -6$

invertieren

+1 addieren



Quelle: <http://dilbert.com/fast/1992-09-08>

Zurück zu der Summe der ersten n Quadratzahlen:

Wir können übersichtshalber das Ergebnis in einer separaten Variablen speichern:

```
public class SummeNQuadratZahlen4
{
    public static void main(String[] args)
    {
        int n = 8;
        int ergebnis = (1.0/6.0)*n*(n+1)*(2*n+1);
        System.out.println(ergebnis);
    }
}
```



Bei der Berechnung tritt eine Kommazahl auf.

Schauen wir uns die erst einmal an!

Kommazahlen

Kommazahlen

- Es gibt zwei primitive Datentypen für Kommazahlen:
 - float: Fließkommavariablen einfacher Genauigkeit
 - double: Fließkommavariablen doppelter Genauigkeit
- Vor- und Nachkommastellen werden durch einen Punkt getrennt:
 - 1.0
 - 1500.23
- Java interpretiert obige Beispiele als double-Werte.
- Wenn eine Kommazahl als **float** interpretiert werden soll, muss ihr ein „f“ angehängt werden:
 - 1.0f

Kommazahlen – Beispiel

```
public class Application {  
    public static void main(String[] args) {  
  
        float kapital = 10000f; //Kommazahl deklarieren  
        float zinssatz = 3.5f;  
        float kapitalNachEinemJahr;  
        kapitalNachEinemJahr = kapital * (1.0f+(zinssatz/100.0f));  
        System.out.println("Verzinstes Kapital:" + kapitalNachEinemJahr);  
    }  
}
```

Kommazahlen - Zahlenbereiche

Datentyp	Größe	Zahlenbereich
float	32 Bits	$1,40239846 \cdot 10^{-45} \dots 3,40282347 \cdot 10^{38}$
double	64 Bits	$4,94065645841246544 \cdot 10^{-324} \dots 1,79769131486231570 \cdot 10^{308}$

Der Zahlenbereich ist riesig,
obwohl nur 32/64 Bits zur Verfügung stehen.



Nicht jede Zahl im Zahlenbereich kann exakt getroffen werden

Bsp: `System.out.println(2345678.88f);` gibt `2345679.0` aus

Wissenschaftliche Notation

- Wissenschaftliche Notation (wie beim Taschenrechner) wird in Java auch unterstützt:

Standardnotation	wissenschaftliche Notation
1234.5f	1.2345E3f
0.000012345	

```
double d = 0.000012345;  
System.out.println(d);
```

```
float f = 1.2345E3f;  
System.out.println(f);
```

???



Darstellung von Kommazahlen

- interne Darstellung für eine 32 bit float erfolgt nach der IEEE 754 Spezifikation.



Kommazahl zwischen 1 und 2

- Berechnungsprinzip:** Vorzeichen $\cdot 2^{\text{Exponent}}$ \cdot Mantisse

Datentyp	Vorzeichen (Bits)	Exponenten (Bits)	Mantisse (Bits)
float	1	8	23
double	1	11	52

Zur Vertiefung: <http://www.youtube.com/watch?v=H79PNQ4Z9HE>

http://openbook.galileocomputing.de/javainasel9/javainasel_12_002.htm#mj619c77e97f365a03e7bfea2ba0c64530

© Prof. Dr. Steffen Heinzl

Typumwandlungen

Da bspw. ganze Zahlen und Kommazahlen unterschiedlich repräsentiert werden, müssen ab und an Typumwandlungen vorgenommen werden.

Explizite
Typumwandlungen

Implizite
Typumwandlungen

Explizite Typumwandlungen

- Eine Variable (oder der Wert eines Ausdrucks) soll explizit in einen anderen Datentyp umgewandelt werden.
- Dazu stellt man diesen Typ geklammert einer Variablen oder einem Ausdruck voran.
- Folgende hervorgehobene Anweisung wandelt einen float-Wert in einen Integer-Wert um:

```
float f = 3.5f;
```

```
int i;
```





```
i = (int) f;
```

(<type>)
heißt auch
cast-Operator

- Obige Umwandlung hat zur Folge, dass die Nachkommastellen einfach **abgeschnitten** werden.

Implizite Typumwandlung

Werden zwei Operanden durch einen Operator verknüpft, erfolgen folgende Typumwandlungen implizit:

- sonst  ▪ Ein Operand ist vom Typ **double** -> der andere Operand wird ebenfalls in **double** umgewandelt.
- sonst  ▪ Ein Operand ist vom Typ **float** -> der andere Operand wird ebenfalls in **float** umgewandelt.
- sonst  ▪ Ein Operand ist vom Typ **long** -> der andere Operand wird ebenfalls in **long** umgewandelt.
-  ▪ Beide Operanden werden in Typ **int** umgewandelt (d.h. byte, char und short werden ebenfalls in **int** umgewandelt)

➤ Der „kleinste“ Datentyp innerhalb eines Ausdrucks ist **int**.

Implizite Typumwandlung - Beispiel

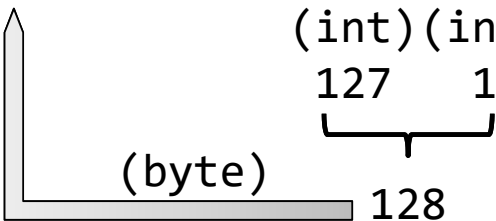
```
byte b = 0x2a;  
short s = 32;  
int i = 5;  
float f = 10.25f;  
double d;
```

```
d = (b - s) + i * f;
```

The diagram illustrates the implicit type conversions for the expression `(b - s) + i * f` in the assignment `d = (b - s) + i * f;`. The variable `d` is of type `double`. The expression is evaluated as follows:

- `b` (byte) and `s` (short) are promoted to `int` for the subtraction operation.
- `i` (int) and `f` (float) are promoted to `float` for the multiplication operation.
- The results of the subtraction and multiplication are promoted to `float` for the addition operation.
- The final result is promoted to `double` to match the type of `d`.

Explizite Typumwandlung - Beispiel

`erg = (byte) (b1 + b2);`


Zur besseren Anschaulichkeit wird ein **int** hier mit „nur“ 16 Bits dargestellt anstelle von 32 Bits.

0000 0000 0111 1111 int (Zweierkomplement)	127	$ \begin{aligned} &0 \cdot 2^{15} + 0 \cdot 2^{14} + 0 \cdot 2^{13} + 0 \cdot 2^{12} + 0 \cdot 2^{11} + 0 \cdot 2^{10} + 0 \cdot 2^9 + 0 \cdot 2^8 \\ &+ 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127 \end{aligned} $
0000 0000 1000 0000 int (Zweierkomplement)	128	$ \begin{aligned} &0 \cdot 2^{15} + 0 \cdot 2^{14} + 0 \cdot 2^{13} + 0 \cdot 2^{12} + 0 \cdot 2^{11} + 0 \cdot 2^{10} + 0 \cdot 2^9 + 0 \cdot 2^8 \\ &+ 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \\ &= 128 \end{aligned} $
1000 0000 byte (Zweierkomplement)	erg	$ \begin{aligned} &-1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \\ &= -128 \end{aligned} $

Zurück zu der Summe der ersten n Quadratzahlen:

Was heißt das für die Speicherung unseres Zwischenergebnisses?

```
public class SummeNQuadratZahlen5
{
    public static void main(String[] args)
    {
        int n = 8;
        double ergebnis = (1.0/6.0)*n*(n+1)*(2*n+1);
        System.out.println(ergebnis);
    }
}
```

Bei der Berechnung tritt
eine Kommazahl auf.



Das Ergebnis ist auch eine
Kommazahl.



Wir speichern das Ergebnis
in einer double Variablen.

Alternativ können wir das Ergebnis vor dem Zwischenspeichern casten.

```
public class SummeNQuadratZahlen6
{
    public static void main(String[] args)
    {
        int n = 8;
        int ergebnis = (int) ((1.0/6.0)*n*(n+1)*(2*n+1));
        System.out.println(ergebnis);
    }
}
```

Die Nachkommastellen sind dann weg!

Einzelzeichen

Im Speicher wurden Zahlen mit Hilfe einer Aneinanderreihung von 0ern und 1ern unterschieden.

Aber wie legt man Buchstaben und Texte ab?

Genauso!

Einzelzeichen/Buchstaben

char	16 Bits	0x0000 ... 0xFFFF	Zur Speicherung einzelner Buchstaben. Wird durch Zahlencodes repräsentiert
------	---------	-------------------	-------------------------------------------------------------------------------

- Mit dem Datentyp **char** kann ein einzelner Buchstabe (engl. Character) gespeichert werden.

```
char c = 'A'; //beachten Sie die einfachen Anführungsstriche!
```

Durch den Datentypen `char` wird festgelegt, dass der Speicherbereich als Buchstabe interpretiert werden soll!

Aber welche Kombination aus 0ern und 1ern entspricht welchem Buchstaben?

Dies wird festgelegt durch den ASCII Code.

ASCII - American Standard Code for Information Interchange

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Quelle: <http://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/800px-ASCII-Table-wide.svg.png>

ASCII - American Standard Code for Information Interchange

Großbuchstaben

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Quelle: <http://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/800px-ASCII-Table-wide.svg.png>

© Prof. Dr. Steffen Heinzl

ASCII - American Standard Code for Information Interchange

Kleinbuchstaben

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Quelle: <http://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/800px-ASCII-Table-wide.svg.png>

© Prof. Dr. Steffen Heinzl

ASCII - American Standard Code for Information Interchange

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Quelle: <http://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/800px-ASCII-Table-wide.svg.png>

© Prof. Dr. Steffen Heinzl

Einzelzeichen/Buchstaben – ctd.

- Java verwendet Unicode. Jedes Zeichen wird durch 2 Bytes (=16 Bits) repräsentiert.
- Die ersten 128 Zeichen des Unicode entsprechen den ersten 128 Zeichen des ASCII Codes.
- Es gibt folgende Möglichkeiten einen Buchstaben darzustellen:

```
char c = 'A'; //beachten Sie die einfachen Anführungsstriche!
```

```
char c = 65;
```

```
/*durch \uXXXX wird ein Unicode-Zeichen definiert, wobei X jeweils eine  
Hexadezimalziffer darstellt*/
```

```
char c = '\u0041';
```


Es gibt einige Zeichen, die schwierig auszugeben sind, bspw.
Zeilenumbrüche.

```
System.out.println("Hallo  
Welt!");
```

```
System.out.println("Hallo  
Welt!");
```

Wie unterscheidet der Compiler, was ausgegeben
werden soll und was zur Anordnung des Codes dient?

Wie schreibe ich im Quellcode, dass ich

- einen Zeilenumbruch ausgeben möchte
- ein Zeichen zurückgehen möchte (Backspace)
- einen ' ausgeben möchte
- einen " ausgeben möchte
- einen \ ausgeben möchte?

Dafür werden sogenannte **Escape-Sequenzen** verwendet.

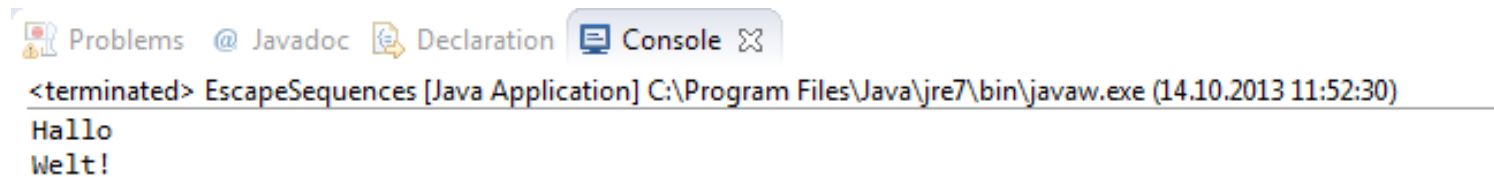
Escape-Sequenzen

- Eine Escape-Sequenz wird durch einen Backslash (\) eingeleitet.

Zeichen	Java-Kürzel	Bedeutung
Line Feed/New Line	\n	gibt einen Zeilenumbruch aus
Backspace	\b	geht mit dem Cursor einen Schritt zurück (sieht man nur in einem echten Konsolen/Terminal-Fenster).
einfacher Anführungsstrich	\'	gibt einen einfachen Anführungsstrich aus
doppelte Anführungsstriche	\"	gibt doppelte Anführungsstriche aus
Backslash	\\	gibt einen Backslash aus

Escape-Sequenzen – Beispiele

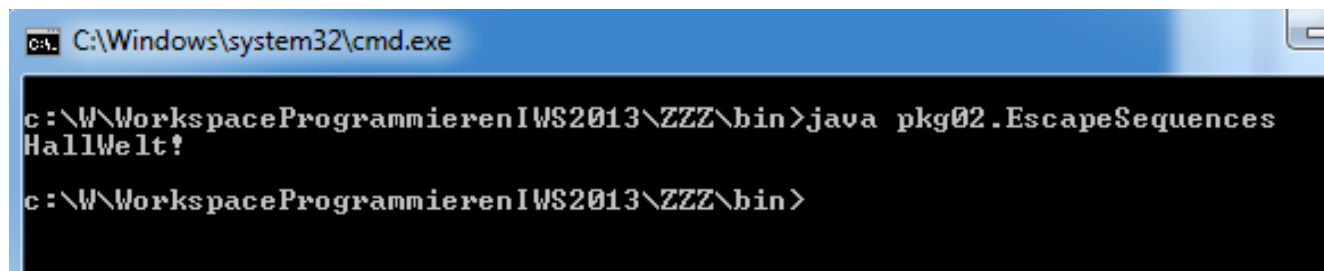
```
System.out.println("Hallo\nWelt!");
```



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a Java application. The output consists of two lines: "Hallo" followed by "Welt!" on the next line, demonstrating the effect of the escape sequence \n.

```
<terminated> EscapeSequences [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (14.10.2013 11:52:30)  
Hallo  
Welt!
```

```
System.out.println("Hallo\bWelt!");
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "c:\W\WorkspaceProgrammierenIWS2013\ZZZ\bin>". The user has entered the command "java pkg02.EscapeSequences". The output of the command is "HallWelt!", where the backslash escape sequence \b has been used to insert a space between "Hall" and "Welt!".

```
C:\Windows\system32\cmd.exe  
c:\W\WorkspaceProgrammierenIWS2013\ZZZ\bin>java pkg02.EscapeSequences  
HallWelt!  
c:\W\WorkspaceProgrammierenIWS2013\ZZZ\bin>
```

Whitespaces

- **Whitespaces** bezeichnen Zeichen, die einen vertikalen oder horizontalen Abstand repräsentieren.
- Darunter fallen in unserem Sprachraum (ISO-LATIN-1) bspw.:

Zeichenbezeichnung	Java-Kürzel
Leerzeichen (Space)	'\u0020', ' '
horizontaler Tabulator	'\u0009', '\t'
Zeilenumbruch (New Line/Line Feed)	'\u000A', '\n'
Wagenrücklauf (Carriage Return)	'\u000D', '\r'
Non-breaking spaces	'\u00A0', '\u2007', '\u202F'
Form Feed	'\u000C'
File Separator	'\u001C'
Group Separator	'\u001D'
Record Separator	'\u001E'
Unit Separator	'\u001F'

Whitespaces im ASCII-Code

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Strings

Mit einzelnen Zeichen zu hantieren ist umständlich.

Geht das nicht einfacher?

Ja! Mit Zeichenketten (Strings).

Strings - Zeichenketten

- Ein String (Zeichenkette) kann man als Aneinanderreihung von Einzelzeichen interpretieren.
- Ein String ist kein primitiver Datentyp, sondern hat eigene Methoden (dazu später mehr).

```
/*Deklariert eine Variable vom Typ String  
und weist ihr den Wert Hallo Welt zu*/
```

```
String str = "Hallo Welt"; //Beachten Sie die doppelten Anführungsstriche  
System.out.println(str); //gibt den String aus
```

Konkatenationsoperator +


- Der + Operator ist in Java **überladen**.
 - + wird für die Addition von Zahlen benutzt.
 - + wird auch für das Aneinanderhängen (die Konkatenation) von Strings (Zeichenketten) verwendet.
- Beispiel für Stringkonkatenation:
"Hal" + "lo" wird ausgewertet zu "Hallo"

Konkatenationsoperator + - Beispiel

```
int x = 5;  
int ergebnis = x*x;  
System.out.println("Das Ergebnis lautet: " + ergebnis);
```

Implizite Typumwandlung - Ergänzung

Werden zwei Operanden durch einen + Operator verknüpft, erfolgen folgende Typumwandlungen implizit:

- sonst 
- Einer der Operanden ist ein String -> der andere Operand wird auch in einen String umgewandelt.
 - gelten die bisherigen Regeln für implizite Typumwandlung

Beispiel: Strings und Konkatination

```
String s = "Hallo";
```

```
String t = "Welt!";
```

```
/*Konkateniere die beiden String s und t und gebe den konkatenierten String aus*/
```

```
System.out.println(s+t);
```

```
System.out.println(s + " " + t);
```

Beispiel: Strings und Konkatenation

```
String s = "Hallo";  
String t = "Welt!";
```

```
/*Konkateniere die beiden String s und t und gebe den konkatenierten String  
aus*/
```

```
System.out.println(s+t);
```

```
System.out.println(s + " " + t);
```

 = String-Literale

Leerer String

- "" ist ein leerer String (ähnlich der 0 bei einer ganzen Zahl)
- An den leeren String können Inhalte angehängt werden:

```
String s;  
s = "" + 2015;
```

Mischung Arithmetische Operatoren und Konkatenationsoperator

```
System.out.println(x + " - " + y + " = " + (x-y));  
System.out.println(x + " * " + y + " = " + (x*y));  
System.out.println(x + " / " + y + " = " + (x/y));  
System.out.println(x + " % " + y + " = " + (x%y));  
System.out.println(x + " + " + y + " = " + (x+y));
```

**+ als Operator zur
Stringkonkatenation**



+ als Rechenoperator



Ausgabe für
x=14 und y=4:

```
14 - 4 = 10  
14 * 4 = 56  
14 / 4 = 3  
14 % 4 = 2  
14 + 4 = 18
```

Ausgabe auf Standardoutput

- `System.out.println(<argument>)`: gibt das übergebene Argument aus, gefolgt von einem Zeilenumbruch
- `System.out.print(<argument>)`: gibt das übergebene Argument aus
- `System.out.printf(<arguments>)`: für eine formatierte Ausgabe, z. B. um eine bestimmte Anzahl Nachkommastellen auszugeben.

Konstanten

(symbolische) Konstanten

- Wenn eine Variable im Programmablauf nicht mehr geändert werden soll, kann man diese durch Voranstellung des Schlüsselworts **final** als Konstante deklarieren.

```
final float PI = 3.1415f;
```

- Konstanten werden per Konvention komplett in **Großbuchstaben** geschrieben.
- Bei Schreibzugriffen auf eine Konstante wirft bereits der Compiler eine Fehlermeldung.

```
3 public class Konstante
4 {
5     public static void main(String[] args)
6     {
7         final float PI = 3.1415f;
8         PI = 5;
9     }
10 }
11
```

```
3 public class Konstante
4 {
5     public static void main(String[] args)
6     {
7         final float PI = 3.1415f;
8         PI = 5;
9     }
10 }
11
```

The final local variable PI cannot be assigned. It must be blank and not using a compound assignment