11 Speicherplätze für einzelne Zeichen (char)

DOWNLOAD Alle Java-Quelltexte der Beispiele, Übungsaufgaben und Lösungen dieses Kapitels können von http://www.w-g-m.de/java.htm durch Anklicken von Dateien für Kapitel 11 heruntergeladen werden. Das weitere Vorgehen erfolgt so, wie auf Seite 55 geschildert. Die Bildschirm-Abzüge basieren auf JOE – in diesem Kapitel werden alle Beispiele damit behandelt.

11.1 Vereinbarung und Belegung

11.1.1 Speicherplätze für einzelne Zeichen

Tritt in einem Java-Programm im Vereinbarungsteil eine char-Vereinbarung auf:

```
// Vereinbarungsteil
     char c;
```

dann ist damit ein Speicherplatz angefordert, der *genau ein einzelnes Zeichen* aufnehmen kann – einen Groß- oder Kleinbuchstaben, eine Ziffer, ein Sonderzeichen von Plus bis Schrägstrich, von Punkt über Komma bis zu Doppelpunkt und Semikolon, dazu auch alle Zeichen über den Zifferntasten oben am Tastaturrand.

Soll ein solcher char-Speicherplatz mit einem ganz konkreten Zeichen belegt werden, muss es *in einfache Apostrophe* ' '(Hochkommas) eingeschlossen werden, die sich meist über dem Zeichen # befinden:

```
c='$';
System.out.println("Inhalt von c="+c);
```

11.1.2 char-Felder

Mengen nummerierter Zeichen-Speicherplätze können in gleicher Weise wie Zahlen-Felder in den bekannten zwei Stufen vereinbart werden:

Zuerst erfolgt die Compiler-Information

```
char[] wort;
```

mit der der Bezeichner wort dem Java-Compiler javac als übergreifender Feldname für eine (noch unbekannte) Anzahl nummerierter Einzelzeichen-Speicherplätze mitgeteilt wird.

Mit der zweiten Vereinbarungsaktivität

```
wort=new char[100];
```

werden dann die einhundert char-Speicherplätze mit den Namen wort[0] bis wort[99] angefordert. Sie sind *alle anfangs als leer* zu betrachten.

Mit einem char-Feld ist es in diesem Buch erstmalig möglich, auch einen *Text*, d. h. eine ganze *Zeichenfolge*, zu speichern. Das Beispiel 11.1 (im Download verfügbar in der Datei Bsp11_1.java) zeigt den Weg dazu:

```
// Vereinbarungsteil
     char[] wort; wort=new char[100];
     int n;
//Ausführungsteil
     wort[0]='V';  wort[1]='i'; wort[2]='e'; wort[3]='w'; wort[4]='e';
     wort[5]='g';  wort[6]='-'; wort[7]='V'; wort[8]='e'; wort[9]='r';
     wort[10]='l'; wort[11]='a'; wort[12]='g';
     n=13;
     for(int i=0; i<=n-1; i++){
        System.out.print(wort[i]);
        }
     System.out.println();
     for(int i=n-1; i>=0; i--){
        System.out.print(wort[i]);
        }
```

Mit den beiden Zählschleifen wird beispielhaft vorgeführt, wie aus einzeln abgespeicherten Zeichen eine *Zeichenfolge*, ein *Wort*, sogar vorwärts und rückwärts gelesen, auf dem Bildschirm erzeugt werden kann. Der Ausgabe-Befehl System.out.println(); mit dem leeren Klammernpaar dazwischen sorgt für den Zeilenwechsel – Bild 11.1 zeigt die Ausgabe.

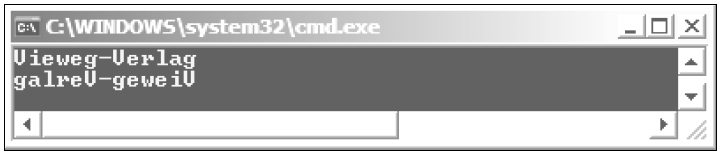

Bild 11.1: Texte aus Einzelzeichen zusammengesetzt

Es ist seltsam – die Methode Keyb.nextChar() gibt es in Java 5 nicht. Also kann kein Nutzerdialog programmiert werden, um einen char-Speicherplatz mit einem einzelnen Zeichen zu belegen. Warum?

Der Grund liegt darin, dass es ohnehin nicht üblich ist, nach jedem Buchstaben des eigenen Namens die Enter -Taste zu drücken.

Mit der komfortablen *Zeichenkettenarbeit* werden wir uns bald, im Abschnitt 12 ab Seite 149, beschäftigen.

11.2 char und int

11.2.1 Zusammenhänge

Es ist selbstverständlich, dass in einem Quelle-Ziel-Befehl ein char-Speicherplatz den Inhalt eines anderen char-Speicherplatzes zugewiesen bekommen kann:

```
// Vereinbarungsteil
    char c1, c2;
//Ausführungsteil
    c1='#'; c2=c1;
    System.out.println("c2="+c2);
```

Selbstverständlich ist auch, dass der Inhalt eines char-Speicherplatzes mit einem in Hochkommas eingeschlossenen Einzelzeichen oder mit dem Inhalt eines anderen char-Speicherplatzes verglichen werden kann (Bsp11_2.java):

```
    if(c2 != 'A') System.out.println("c2 hat nicht das grosse A");
```

Sehr erstaunlich jedoch ist, dass ein char-Speicherplatz *ohne Fehlermeldung* mit einer *ganzen Zahl* belegt werden darf

```
// Vereinbarungsteil
    char c;
    int x;
//Ausführungsteil
    c=65;                                          //ganze Zahl in char!
```

und dass auch umgekehrt ein char-Speicherplatz die Quelle für einen int-Speicherplatz bilden darf:

```
    x=c;                                           //char nach int!
```

In beiden Fällen gibt es, wie man durch Ausprobieren oder mit Hilfe der Download-Datei Bsp11_3.java überzeugen kann, *keine Fehlermeldung*, obwohl Java doch sonst so penibel die Verhältnisse zwischen Quelle und Ziel prüft.

Ganz im Gegenteil, es gibt eine *scheinbar sehr überraschende Ausgabe*, wie Bild 11.2 zeigt.

Im char-Speicherplatz c entstand aus der Zahl 65 das Zeichen A, und im int-Speicherplatz x entstand rückwirkend aus dem Zeichen A wieder die Zahl 65.

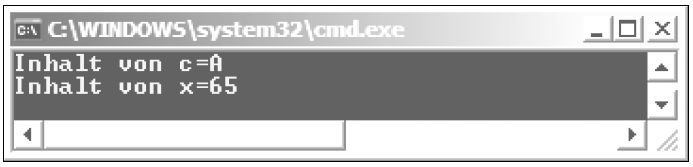

Bild 11.2: Seltsames Wechselspiel zwischen `char` *und* `int`

Des Rätsels Lösung liegt in der so genannten *ASCII-Tabelle*:

32			58	:		84	T		110	n	
33	!		59	;		85	U		111	o	
34	"		60	<		86	V		112	p	
35	#		61	=		87	W		113	q	
36	$		62	>		88	X		114	r	
37	%		63	?		89	Y		115	s	
38	&		64	@		90	Z		116	t	
39	'		65	A		91	[		117	u	
40	(		66	B		92	\		118	v	
41	)		67	C		93	]		119	w	
42	*		68	D		94	^		120	x	
43	+		69	E		95	_		121	y	
44	,		70	F		96	`		122	z	
45	-		71	G		97	a		123	{	
46	.		72	H		98	b		124	|	
47	/		73	I		99	c		125	}	
48	0		74	J		100	d		126	~	
49	1		75	K		101	e		127		
50	2		76	L		102	f				
51	3		77	M		103	g				
52	4		78	N		104	h				
53	5		79	O		105	i				
54	6		80	P		106	j				
55	7		81	Q		107	k				
56	8		82	R		108	l				
57	9		83	S		109	m				

Denn anders als bei den Zahlen, wo durch die *Binärdarstellung* (`an-aus` in unserem Lampenmodell von Seite 67) eine *natürliche Form der internen Verarbeitung* gegeben ist, gibt es für die einzelnen *Zeichen* keine naturgegebene Form der internen Verarbeitung. Es sind *willkürliche Tabellen*, die sich inzwischen durchgesetzt haben und heute einen Standard darstellen. Die obige *ASCII-Tabelle* enthält die weltweit meistgenutzte Codierung.

Beim *Codewert 32* beginnen die *darstellbaren Zeichen* mit dem *Leerzeichen*.

Die Codewerte davor werden abgespeichert, wenn ein Nutzer andere Tasten drückt. Zum Beispiel entspricht der ASCII-Wert 27 der Taste Esc. Der ASCII-Wert 13 entspricht der Enter -Taste usw.

Die Codewerte 48 *bis* 57 sind den Zeichen *Null bis Neun* vorbehalten. Von 65 *bis* 90 finden sich die *Großbuchstaben,* im Bereich von 97 *bis* 122 finden sich die *Kleinbuchstaben* des englischen Alphabets.

Die Belegung der ASCII-Werte 0 bis 127 ist auf jedem Computer der Welt *einheitlich*. Für die weiteren Belegungen kann es große Unterschiede geben, hier können entsprechend dem spezifischen Zeichensatz der jeweiligen Landessprache *Sonderregelungen* getroffen werden.

Die türkische Sprache beispielsweise benutzt ein i ohne Punkt – dieses Zeichen ist aber links nicht enthalten. Dafür braucht die deutsche Sprache ihre Umlaute und das so genannte *scharfe S*, das ß.

Folglich werden auf Computern in der Türkei die ASCII-Werte von 128 bis 255 anders belegt sein, den Gegebenheiten der Sprache entsprechend. So ist es in allen Ländern, die mit den Standard-ASCII-Zeichen nicht auskommen.

Mit Blick auf die ASCII-Tabelle lässt sich nun auch erklären, wie die *Kleiner-größer-Beziehungen zwischen Zeichen* zustande kommen: *Ein Zeichen wird kleiner als ein anderes angesehen, wenn der zugehörige ASCII-Wert kleiner ist.* Aus der ASCII-Tabelle können wir ablesen:

```
'#' < '(' < '0' < ... < '9'<...<'A'<...<'Z'< '\' <...<'a'<...< 'z'< '~'
```

Soll ein bestimmtes Zeichen in einen char-Speicherplatz gebracht werden, ist es in einfache Hochkommas '' zu setzen. Der Zuweisungsbefehl z='0' belegt den char-Speicherplatz z mit dem *Zeichen Null*. Die *Zahl Null* dagegen kann nur in einen der int-Speicherplätze gebracht werden, z=0 mit char z; wäre also grundfalsch!

Nach diesen Überlegungen und dem Kennenlernen der ASCII-Tabelle können wir uns die vorhin geschilderten, scheinbar eigenartigen Vorgänge in den Speicherplätzen c und x erklären:

```
char c;
int x;
c=65;                                          //ganze Zahl in char!
x=c;                                               //char nach int!
```

Durch die Zuweisung c=65; wurde der Befehl formuliert, dass der char-Speicherplatz c das zum ASCII-Wert 65 gehörende Zeichen bekommen soll. Also wurde es mit dem großen A belegt.

Umgekehrt bewirkte der Befehl x=c; dass der int-Speicherplatz x den ASCII-Wert des Inhalts von c bekommen sollte – dort kam also die Zahl 65 an.

11.2.2 Anwendungen

Im Beispiel 11.1 hatten wir erlebt, wie die Zeichenfolge *Vieweg-Verlag* mit Hilfe eines `char`-Feldes gespeichert und auch schon bearbeitet werden kann.

Nun wollen wir lernen, wie mit Hilfe der bekannten ASCII-Codewerte genau analysiert werden kann, aus welcher Art von Zeichen sich diese Zeichenfolge zusammensetzt (Download: `Bsp11_4.java`).

```
anz_gross=0; anz_klein=0; anz_ziffer=0; anz_sonst=0;
for(int i=0; i<=n-1; i++){
   if((wort[i]>=48)&&(wort[i]<=57))anz_ziffer++;
   if((wort[i]>=65)&&(wort[i]<=90))anz_gross++;
   if((wort[i]>=97)&&(wort[i]<=122))anz_klein++;
   }
anz_sonst=n - (anz_gross + anz_klein + anz_ziffer);
System.out.println("Anzahl der Ziffern="+anz_ziffer);
System.out.println("Anzahl der Grossbuchstaben="+anz_gross);
System.out.println("Anzahl der KLeinbuchstaben="+anz_klein);
System.out.println("Anzahl sonstiger Zeichen="+anz_sonst);
```

Drei *Zählwerksspeicherplätze* werden genutzt, um die Groß- und Kleinbuchstaben und die Ziffern zu zählen. Ihre Summe und deren Differenz zur Gesamtzeichenzahl ergibt die Anzahl der Sonderzeichen.

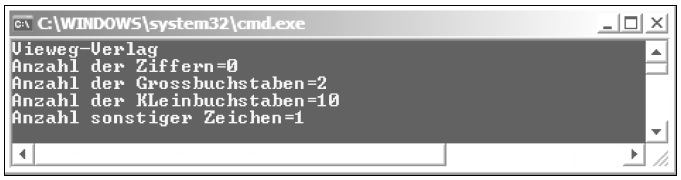

Bild 11.3: Textanalyse

ÜBUNG *Übung 11.1:* Benutzen Sie den Programmrahmen früherer Beispiele oder arbeiten Sie mit der Download-Datei `Uebg11_1.java` im Download-Ordner `WGMKap11`. Vereinbaren Sie ein `char`-Feld `x` mit 100 Elementen. Belegen Sie dessen erste `26` Elemente mit der folgenden Zählschleife:

```
for(int i=0; i<=25; i++){
     wort[i]=(char)(65+i);
     }
```

Überlegen Sie, welche Zeichen damit in `wort[0]` bis `wort[25]` erzeugt werden. Die Lösung finden Sie auf Seite 372. **ÜBUNG**