- Einzelne Zeichen (char)
- Strings und String-Methoden

(Einzelne) Zeichen

- In Java (wie in den meisten Programmiersprachen) gibt es nicht nur *Strings*, d.h. Zeichenketten, sondern auch einzelne Zeichen vom Datentyp *char* (engl. character)
- Einzelne Zeichen werden in einfache Hochkommata (= Apostroph; nicht: Akzent-Zeichen!) eingeschlossen
- Beispiele:

  'a'

  '$'

  '='

  'Ö'

  usw.

Zugriff auf einzelne Zeichen im String

- Die Zeichen innerhalb eines Strings können mit der Methode

  String.charAt(*int*)

  abgefragt werden; der Parameter gibt die Position des Zeichens im String an; die Positionsnummern beginnen bei 0
- Diese Methode liefert das Zeichen an der angegebenen Position in dem String, an den sich die Anfrage richtet
- Beispiele
  - String s = "abcde";
    s.charAt(3)

    →
  - "a".charAt(0)

    →
  - "".charAt(3)

    →
  - ("abc" + "123").charAt(4)

    →

Escape-Darstellungen 1

- Problem:
  Der Apostroph kann nicht „normal“ dargestellt werden:

```
char ch = 'a';

ch = ''';
```

- Die Lösung in Java (andere Sprachen, andere Lösungen):
  Um den Apostroph in einem Zeichen darzustellen, wird ihm das Zeichen \ (Backslash) vorangestellt:

```
char ch = 'a';

ch = '\'';
```

- Das ist die Schreibweise für *ein einzelnes Zeichen*!

Escape-Darstellungen 2

- Problem Nummer 2:
  Der Backslash kann jetzt nicht mehr „normal“ dargestellt werden:

```
char ch = 'a';

ch = '\';
```

- Die Lösung:
  Um den Backslash in einem Zeichen darzustellen, wird auch ihm das Zeichen \ (Backslash) vorangestellt:

```
char ch = 'a';

ch = '\\';
```

- Das ist wiederum die Schreibweise für *ein einzelnes Zeichen*!

Escape-Darstellungen 3

- Weitere spezielle, nicht-schreibbare *char*s werden ebenfalls mittels Escape-Sequenz dargestellt; bekannte Beispiele:

  \t horizontal tab HT, Tabulator

  \n linefeed LF, Zeilenumbruch und -vorschub

  \r carriage return CR, Zeilenumbruch

- Verwendung

```
println("Hallo");

// Entsprechung in Linux/MacOS:
print("Hallo\n");

// Entsprechung in Windows:
print("Hallo\r\n");
```

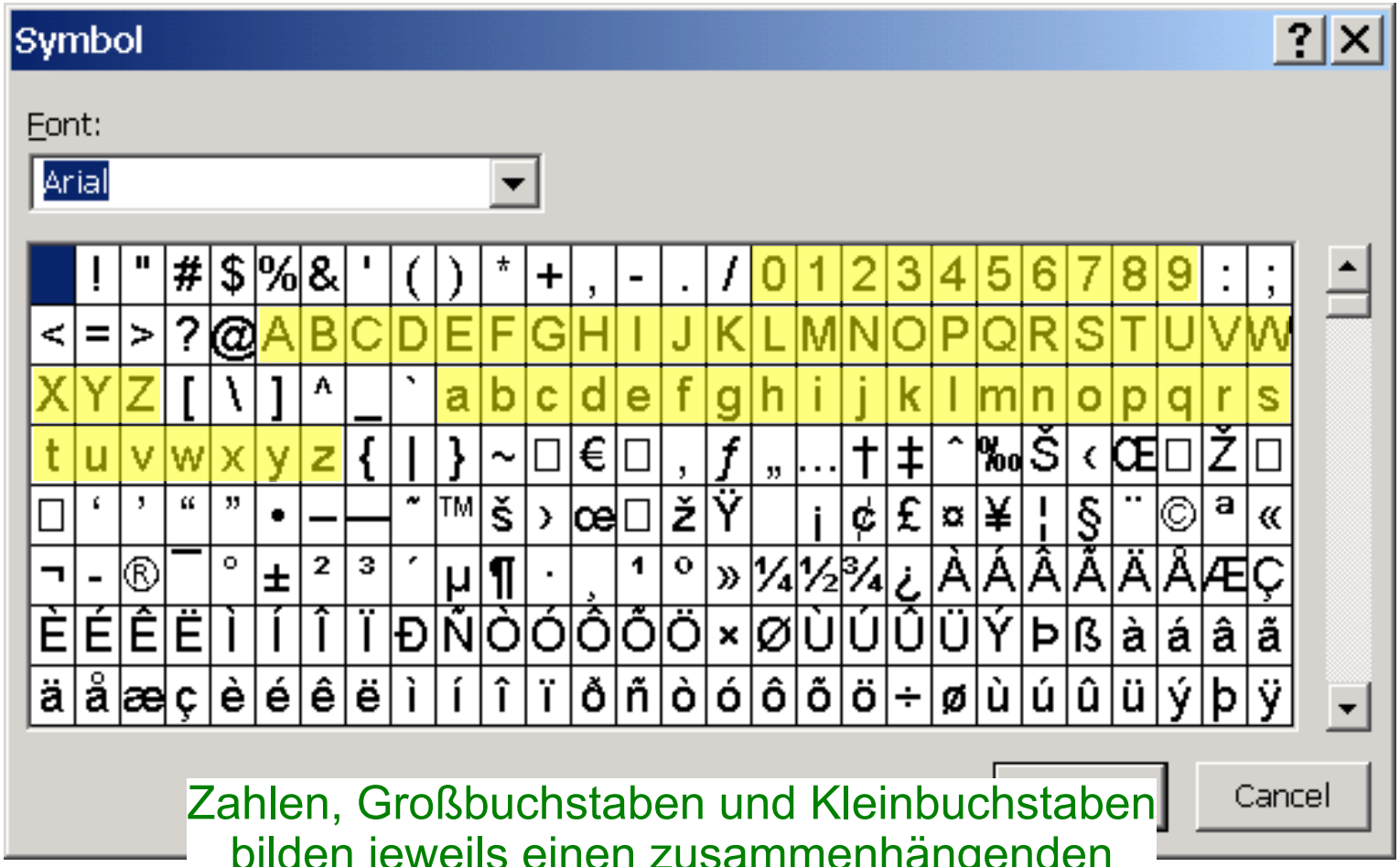
Der Unterschied wird oft (nur) in Textdateien bemerkt

Unicode

- Java stellt intern jedes Zeichen in *Unicode* dar, wofür 1 bis 4 Bytes verwendet werden
- In vielen *anderen* Programmiersprachen wird die *ASCII-Darstellung* verwendet, die mit einem Byte, also 8 Bit, auskommt (auskommen muss)
  (Nur) Die ersten 128 Zeichen in ASCII und Unicode entsprechen einander
- Vorteil von Unicode:
  Es können viel mehr Zeichen, z.B. Umlaute oder andere nationale Sonderzeichen, auch deutsche, französische, russische, asiatische etc. dargestellt werden

- Mehr Informationen zu Unicode: http://www.unicode.org

Zeichen-Anordnung



Zahlen, Großbuchstaben und Kleinbuchstaben bilden jeweils einen zusammenhängenden geordneten Bereich

Rechnen mit Zeichen

- Java behandelt auch Werte vom Typ *char* als ganze Zahlen, daher kann mit *char*s ähnlich wie mit Zahlen gerechnet werden
- Wertebereich von *char*:
  - Von 0 bis 65.535
- Typische Anwendung:

```
ch = readString().charAt(0);

while (ch < '0' || ch > '9')
    ch = readString().charAt(0);

int digit = ch - '0';

println(digit * 10);
```

Zugehörige Ein-/Ausgabe:

```
1
10
```

Rechnen mit Zeichen

- Nur zur Illustration:
  Der Unterschied zwischen Zeichen und Ziffern zeigt sich beim Rechnen

- Typische Anwendung:

```
ch = readString().charAt(0);

while (ch < '0' || ch > '9')
    ch = readString().charAt(0);

int digit = ch - '0';

println(digit * 10);

println(ch * 10);
```

Zugehörige Ein-/Ausgabe:

```
1
10
490
```

Merk-Regel:

Es ist ganz schlechter Stil,

Zeichen durch ihren

Dezimalcode (z.B. 97 statt 'a') darzustellen

Zeichen und ihr Dezimalcode

```
print("Haben Sie heute Geburtstag (j/n)? ");
ch = readString().charAt(0);

if (ch == 106)
    println("Herzlichen Glückwunsch!");
```

schlechtes **Beispiel**

Zeichen und ihr Dezimalcode

```
print("Haben Sie heute Geburtstag (j/n)? ");
ch = readString().charAt(0);

if (ch == 106)
    println("Herzlichen Glückwunsch!");


if (ch == 'j')
    println("Herzlichen Glückwunsch!");
```

schlechtes **Beispiel**

richtiges **Beispiel**

HOCHSCHULE DER WIRTSCHAFT FÜR MANAGEMENT
UNIVERSITY OF APPLIED MANAGEMENT STUDIES

Strings und String-Methoden

- Strings werden in Java durch doppelte Hochkommata begrenzt
- Problem:
  Die doppelte Hochkommata können in einem String nicht „normal“ dargestellt werden:

```
String s = "Ein String mit " Hochkommata";
```

- Die Lösung:
  Es wird das Zeichen \ (Backslash) vorangestellt:

```
String s1 = "Ein String mit \" Hochkommata";

String s2 = "Ein String mit \\ Backslash";
```

- Das ist die Schreibweise für *ein einzelnes Zeichen* im String!

Längenbestimmung von Strings

- Wir kennen bereits die Methode

  *String.*length()

  um die Länge eines Strings zu erfragen
- Die Methode liefert die Anzahl der Zeichen in dem String, an den sich die Anfrage richtet, als *int*-Wert
- Beispiele
  - "a".length()

    →
  - ("abc" + "123").length()

    →
  - "".length()

    →
  - "ab\"c\'d\"ef".length()

    →

Korrektes Vergleichen von Strings

- Wir kennen bereits die Methode

  *String.*equals(*String*)

  um den Inhalt zweier Strings zu vergleichen
- Die Methode liefert dann und nur dann *true*, wenn die beiden Strings gleich lang und Zeichen für Zeichen gleich sind, sonst liefert sie *false*

Strings vs. Array of Character

- Anders als in anderen Sprachen (C, Pascal etc.) sind in Java Array of Character, d.h. char[ ], *nicht* gleichbedeutend mit Strings
- Beispiel:
  Ausgehend von folgenden Vereinbarungen

```
char[ ] ca = new char[3];                    // Array
ca[0] = 'a';
ca[1] = 'b';
ca[2] = 'c';
String s = "abc";                            // String
```

  sind folgende Zuweisungen *nicht* zulässig

```
ca = s;                                      // Array = String
s = ca;                                      // String = Array
```

Exkurs: Umwandlung String → char[ ]

- Strings können einfach in Character-Arrays überführt werden
- Die Umwandlung erfolgt mittels einer *Methode* der *Klasse String*

```
String s = "text";
char[ ] ca;

ca = s.toCharArray();
ca = "dies ist auch ein String".toCharArray();
```

- Das Array mit den *char*s wird in der passenden Länge angelegt

Exkurs: Umwandlung char[ ] → String

- Character-Arrays können mittels *Konstruktor-Aufruf* mit dem Array als Parameter in einen String überführt werden:

```
char[ ] ca = new char[4];
ca[0] = 'T';
ca[1] = 'e';
ca[2] = 'x';
ca[3] = 't';

String s = new String( ca );

println( s );                    → "Text"
```

Unveränderliche Strings

- Strings in Java haben einen konstanten Wert, sie können *nicht verändert* werden, also ihre Zeichen können *nicht* einzeln modifiziert werden
  Es gibt also keine Methode setCharAt(...) oder ähnliche

- Strings können nicht über ein Zeilenende fortgesetzt werden, lange Strings, die schlecht in einer Zeile notiert werden können, können durch die String-Konkatenation zusammengebaut werden

F R A G E N