Verarbeitung von Zeichenketten

Character

- Die Character-Klasse bietet statische Prüffunktionen isXXX(), die bestimmen, ob ein char einer gewissen Kategorie angehört. (Ist der char eine Ziffer, ein Leerzeichen, …)
  - Character.isLetter(char ch)
  - Character.isLetterOrDigit(char ch)
  - Character.isLowerCase(char ch)
  - Character.isUpperCase(char ch)
  - Character.isWhitespace(char ch)

  Die Rückgabe der isXXX()-Funktionen ist boolean.
- Die statischen Funktionen toLowerCase(char ch) und toUpperCase(char ch) liefern ein char zurück.

Die Klasse java.lang.String

Die Klasse String

- Die Klasse java.lang.String kapselt die Implementierung für eine Zeichenkette.
  - Wird die Zeichenkette intern als Zeichenarray, als verkette Liste, als Baum, als ... gespeichert?
  - Wissen wir nicht, ist aber aus objektorientierter Sicht egal. (Aus Effizienzgründe interessiert das natürlich schon.)
- Alles, was in doppelten Anführungszeichen steht, ist als String-Literal ein String-Objekt.

```
System.out.println( "tutego" );
```

- Diese Referenz lässt sich in einer Referenzvariablen merken:

```
String s = "tutego";
```

Strings sind immutable

- Eine wichtige Eigenschaft von Strings ist, das sie unveränderlich **(immutable)** sind.
  - Im String ›Tutego‹ lässt sich das erste Zeichen nicht zu einem kleinen ›t‹ konvertieren.

- Jedoch kann die Referenzvariable jederzeit neu belegt werden.

```
String s = "TUTEGO";
s = "Tutego";
s = "tutego";
```

Einfache Abfragen

- Die Länge eines Strings: length()

```
int len = "tutego".length();          // 6
```

- Zeichen an der Stelle: charAt(int index)

```
char c = "tutego".charAt( 1 );        // ›u‹
```

- Suchen im String: indexOf(int), indexOf(String), lastIndexOf()
  - Die Methoden liefern -1, wenn nichts gefunden wurde.

```
int pos = "tutego".indexOf( 'u' );    // 1
```

- Ob ein String mit einem anderen String beginnt oder endet zeigt startsWith() und endsWith().

```
"http://tutego.com".startsWith("http"); // true
```

String-Vergleiche

- Vergleiche sollten nicht mit == gemacht werden, da so nur Referenzen verglichen werden.
- Die Klasse String implementiert equals() so, dass Zeichenketten auf Gleichheit überprüft werden.

```
String s1 = "Foo", s2 = "Foo";
out.println( s1.equals(s2) );    // true
boolean b = "Foo".equals( s );   // true
```

- Vergleiche unabhängig der Groß-/Kleinschreibung macht boolean equalsIgnoreCase(String).
- Mit regionMatches() lässt sich vergleichen, ob ein Teilstring mit dem String übereinstimmt.
- Lexikografisch vergleicht compareTo(). Die Rückgabe ist kein boolean, sondern <0, =0 oder >0.

Konvertierungen in neue Objekte

- Da String-Objekte immutable sind, können Funktionen, die nach Veränderung aussehen, nur neue Objekte liefern.
- Leerraum vorne und hinten schneidet ab:
  - String trim(String)
- Den String in Klein- bzw. Großbuchstaben konvertieren
  - String toLowerCase(String)
  - String toUpperCase(String)
- Zum Suchen/Ersetzen:
  - String replace(char, char)
  - String replaceAll(CharSequence, CharSequence)

    String ist eine CharSequence.

Strings und Zeichenfelder

- toCharArray() wandelt ein String in ein Array um:

```
char[] vokale = "aeiouäöü".toCharArray();
```

- Um ein Feld von chars in einen String umwandeln, wird der Konstruktor von String genutzt:

```
char[] cs = { 'F', 'o', 'o' };
String two = new String( cs );
boolean b = "Foo".equals( two );        // true
```

StringBuilder und StringBuffer

Die Klasse StringBuffer/StringBuilder

- Die Klasse java.lang.StringBuffer – und seit Java 5.0 StringBuilder – behandelt veränderbare Strings.
  - Beim StringBuffer sind alle Methoden synchronisiert, also gegen nebenläufige Veränderungen durch Threads sicher.
  - Dem StringBuilder fehlt die Synchronisation; er ist daher bei nur einem Thread etwas schneller.
- Einen StringBuffer/StringBuilder kann man mit dem Standard-Konstruktor oder parametrisierten Konstruktor anlegen.
  - Mit einem übergebenen String wird der StringBuffer bzw. StringBuilder gleich gefüllt.

```
StringBuilder sb = new StringBuilder( "Nun" );
```

Anhängen und Löschen von Teilen

- Die Zeichenfolgen können größer werden, in dem Teile mit append() angehängt werden.

```
StringBuilder sb = new StringBuilder( "Es" );
sb.append( " wird warm." );
```

- Die append() Methode gibt eine Referenz auf das aktuelle StringBuilder Objekt zurück. Somit kann man append() Aufrufe auch hintereinander schreiben.

```
StringBuilder sb = new StringBuilder( "Es" );
sb.append( " wird " ).append( "warm." );
```

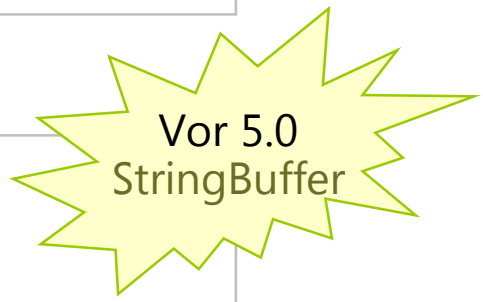
- Zeichen können gelöscht werden: delete(), deleteCharAt().

+ und StringBuffer/StringBuilder

- Die in Java so praktische Konkatenation (+) wird vom Compiler umgebaut.

```
double pi = Math.PI; int n = 9;
String bum = "pi" + pi + ' ' + n;
```

```
String bum = new StringBuilder()
                .append( "pi" )
                .append( pi )
                .append( ' ' )
                .append( n ).toString();
```



- Das Problem ist: Es entstehen besonders in Schleifen viele temporäre Objekte.

Von String nach StringBuilder

- Erzeuge das Alphabet »abc...z« im String s.

```
String s = "";
for ( char c = 'a'; c <= 'z'; ++c )
  s += c;
```

Das Problem ist: Hier entstehen viele Objekte. Wie viele?

- Besser ist eine Lösung mit StringBuilder.

```
StringBuilder sb = new StringBuilder( 26 );
for ( char c = 'a'; c <= 'z'; ++c )
  sb.append( c );
String s = sb.toString();
```

Konvertierungen, Zerlegungen

Von Etwas in String

- Die überladenen statischen Funktionen String.valueOf(val) liefern eine String-Repräsentation von val.

```
String s = String.valueOf( 1 );  // "1"
```

- Ein beliebtes Java-Idiom ist auch

```
String s = "" + i;    // i ist ein int oder irgendwas
```

- Jedes Objekt besitzt die Methode toString(), die eine Stringrepräsentation des Objektzustandes zurückgibt.
  - Sie wird von Object geerbt und oft überschrieben.
- Für Referenzen ruft String.valueOf(o) die Methode toString() des Objekts o auf. Ein Beispiel:

```
String date = String.valueOf( new Date() );
```

Tokenisierung durch split()

- Oftmals stellt sich die Aufgabe, einen String in eine Reihe von Teilstrings (Tokens) zu zerlegen.
  - Ein **Token** ist eine zusammenhängende Sequenz von Zeichen, die durch Trennzeichen (engl. **delimiter**) getrennt sind.
- Ist das Leerzeichen der Delimiter, so entstehen fünf einzelne Strings: ›Gestern‹, ›war‹, ›heute‹, ›noch‹, ›morgen‹.

```
String s = "Gestern war heute noch morgen";
```

String#split()

- Die Objektmethode split() liefert ein Feld von String-Objekten zu einem gegebenen Delimiter.

```
String s = "Gestern war heute noch morgen";
String[] tokens = s.split( " " );
for ( String token : tokens )
  System.out.println( token );
```

- Der Delimiter bei split() ist ein regulärer Ausdruck!
  - Zeichen mit Sonderfunktion, wie ›.‹ oder ›?‹, müssen passend ausmaskiert werden.

Die Klasse Scanner

- java.util.Scanner ist eine flexible Klasse zum Zerlegen von Eingaben.
- Ein Scanner-Objekt kann mit unterschiedlichen Konstruktoren aufgebaut werden:
  - Scanner(String)
  - Scanner(File)
  - Scanner(InputStream), …
- Zum Auslesen gibt es eine Reihe von hasNextXXX()-und nextXXX()-Methoden.
  - Die nextXXX()-Methode liefert die Information, ob ein weiteres Token gelesen werden kann.
  - nextXXX() liefert das nächste Token.

Alle Zeilen mit der Scanner-Klasse

- Mit dem Pärchen hasNextLine() und nextLine() der Scanner-Klasse lässt sich einfach eine Textdatei Zeile für Zeile durchlaufen:

```
File f = new File( dateiname );
Scanner scnr = new Scanner( f );

while ( scnr.hasNextLine() )
  System.out.println( scnr.nextLine() );

scnr.close();
```