Inhaltsverzeichnis

String, Texte verarbeiten

Strings definieren und verwenden

```
java.lang.String

String s;
s= "Hallo";

System.out.print(s);
System.out.println(", Welt!");

s += ", Welt!";
System.out.print(s);
```

Frage:
Was gibt das obige Programm aus?

Antwort:

Konvertierungen: String zu int,float,... und umgekehrt

```
static String valueOf(boolean b)
static String valueOf(char c)
```

```
static String valueOf(char[] c)
static String valueOf(double d)
static String valueOf(float f)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(Object obj)
```

```
int x=17;
double pi= 3.14;
String ziffern= "4711";
int iZahl;
String dblString="5.67"
double dblZahl;
String s1,s2;

s1= String.valueOf(pi);
s2= String.valueOf(x);

iZahl= Integer.parseInt(ziffern);
dblZahl= Double.parseDouble(dblString);
```

Konvertierung: Strings in char-Arrays mit toCharArray()

```
String str= "HALLO, WELT";

char[] b= str.toCharArray();

int len= str.length();
int key= 10;

for (int i =0 ; i < len; i++){
     if (str.charAt(i) >= 'A' && str.charAt(i) <= 'Z'){
          b[i] += key;
          if (b[i] > 'Z') b[i]-=26;
     }
}

String str_chiper= String.valueOf(b);
```

Stringverkettung mit +

```
String s;
s= "Hallo";
s += ", Welt!";

System.out.println(s); // Ausgabe: Hallo, Welt!
```

Stringlänge ermitteln

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

```
int len;

len= s.length();
```

Auf einzelne Stringbereiche zugreifen

```
char charAt(int index) throws
     StringIndexOutOfBoundsException

String substring(int begin, int end) throws
     StringIndexOutOfBoundsException

String[] split(String s)
```

```
String s= "Hallo, Welt!";
int len;
int grossbuchstabe=0;
char zeichen;

len= s.length();
for (int i=0; i < len; i++){
     zeichen= s.charAt(i);
     if (zeichen <= 'A' && zeichen >= 'Z')
          grossbuchstabe++;
     ...

...
```

```
String s= "Hallo, Welt!";
String s1;
String s2;

s1= s.substring(0,5);
s2= s.substring(7,11);
```

Frage:
Was steht in s1 und was in s2?
Antwort:
s1 .... hallo
s2 ....

Merke:

substring(begin,end) liefert als Ergebnis die Zeichen von begin (einschliesslich) bis **end** (**ausschliesslich**)

```
String s ="Hallo:Welt";
String[] t;

t= s.split(":");
```

```
System.out.println(t[0]); // "Hallo"
System.out.println(t[1]); // "Welt"



String tutorials = "www.tutorials.de";
String[] splittArray = tutorials.split(".");

Ergebnis:

splittArray[0] = "www";
splittArray[1] = "tutorials";
splittArray[2] = "de";
```

Vergleichen von Strings

```
boolean equals(Object anObject)
boolean equalsIgnoreCase(String s)

boolean startsWith(String s)
boolean endsWith(String s)

int compareTo(String s)
```

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

if (s.equals("HALLO, WELT!")){
     System.out.println("ungleich);
}
else if (s.equalsIgnoreCase("HALLO, WELT!")){ //liefert hier true
     System.out.println("gleich);
}

if (s.startsWith("Hallo")){
     .....
if (s.endsWith("Welt!")){
     .....
```

Suchen in Strings

```
int indexOf(String s)
int indexOf(String s, int fromIndex)
int lastIndexOf(String s)
```

Die Methode indexOf(s) sucht das erste Vorkommen der Zeichenkette s innerhalb des String-Objekts. Wird s gefunden, liefert die Methode den Index des ersten übereinstimmenden Zeichens zurück, andernfalls wird -1 zurückgegeben.

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

```
int index= s.indexOf("a");

Frage:
Was steht nun in der Variablen index?
1
```

Die zweite Variante von indexOf arbeitet wie die erste, beginnt mit der Suche aber erst ab Position fromIndex.

Die Methode lastIndexOf sucht nach dem letzten Vorkommen des Teilstrings s im aktuellen String-Objekt. Wird s gefunden, liefert die Methode den Index des übereinstimmenden Zeichens, andernfalls -1.

```
String s ="AA:BB:CC";
int index= s.lastIndexOf(":");

Frage:
Was steht nun in der Variablen index?
5
```

Ersetzen in Strings

```
String toLowerCase()
String toUpperCase()
String replace(char oldchar, char newchar)
```

```
String s ="Hallo, Welt!";
String klein;
String neu;

klein= s.toLowerCase();

neu= s.replace('a', 'e'); // Hello, Welt
```

StringBuffer: Einfügen,Löschen,Verändern von Strings

Wenn Strings zur Laufzeit des Programmes ihre **Länge** **verändern** können sollen, muss man aus einem String einen **StringBuffer** machen.

```
java.lang.StringBuffer

StringBuffer()
StringBuffer(String s)
```

```
String s= "Dies ist eine Text";
```

```
StringBuffer sb= new StringBuffer(s);
// sb ist nun ein StringBuffer mit dem Text: Dies ist ....


StringBuffer str= new StringBuffer();
// ist auch möglich. str ist ein leerer StringBuffer
```

Einfügen von Elementen

```
StringBuffer append(String s)
StringBuffer insert(int offset, String s)
```

Mit **append** wird der String s an das Ende des StringBuffer-Objekts angehängt. Zurückgegeben wird das auf diese Weise verlängerte StringBuffer-Objekt s. Zusätzlich gibt es die Methode append in Varianten für das Anhängen aller Arten von primitiven Typen. Anstelle eines String-Objekts wird hier der entsprechende primitive Typ übergeben, in einen String konvertiert und an das Ende des Objekts angehängt.

insert fügt den String s an der Position offset in den aktuellen StringBuffer ein. Zurückgegeben wird das auf diese Weise verlängerte StringBuffer-Objekt s. Auch diese Methode gibt es für primitive Typen. Der anstelle eines String übergebene Wert wird zunächst in einen String konvertiert und dann an der gewünschten Stelle eingefügt.

```
String s= "Hallo, ";
StringBuffer sb= new StringBuffer(s);

sb= sb.append("Welt!");
sb= sb.append(4711);
sb= sb.append(3.14);

Frage:
Welchen Text enthält nun sb?
//sb enthält nun folg. Text: Hallo, Welt!47113.14
```

```
String s= "Des ist eine Text";
StringBuffer sb= new StringBuffer(s);

sb= sb.insert(1, "i");

Frage:
Welchen Text enthält nun sb?
Dies ist .....


sb= sb.insert(0, "Hallo");

Frage:
Welchen Text enthält nun sb?
HalloDies ist .....
```

Löschen von Elementen

```
public StringBuffer deleteCharAt(int index)
public StringBuffer delete(int start, int end)
```

Mit deleteCharAt wird das an Position index stehende Zeichen entfernt und der StringBuffer um ein Zeichen verkürzt.

delete entfernt den Teilstring, der von Position start bis end reicht, aus dem StringBuffer und verkürzt ihn um die entsprechende Anzahl Zeichen.

Verändern von Elementen

```
void setCharAt(int index, char c) throws StringIndexOutOfBoundsException
StringBuffer replace(int start, int end, String str)
```

Mit der Methode setCharAt wird das an Position index stehende Zeichen durch das Zeichen c ersetzt. Falls der StringBuffer zu kurz ist (also index hinter das Ende des StringBuffer-Objekts zeigt), löst die Methode eine Ausnahme des Typs StringIndexOutOfBoundsException aus.

Seit dem JDK 1.2 gibt es zusätzlich eine Methode replace, mit der ein Teil des StringBuffer-Objekts durch einen anderen String ersetzt werden kann. Dabei wird das von Position start bis Position end - 1 gehende Teilstück durch den String str ersetzt. Falls erforderlich, wird der ursprüngliche StringBuffer verkürzt oder verlängert.

Längeninformationen

```
int length()
int capacity()
```

length liefert die Länge des Objekts, also die Anzahl der Zeichen, die im StringBuffer enthalten sind.
Mit capacity wird dagegen die Größe des belegten Pufferspeichers ermittelt. Dieser Wert ist typischerweise größer als der von length zurückgegebene Wert.

Konvertierung in einen String

```
String toString()
```

Ein StringBuffer-Objekt kann mit Hilfe dieser Methode effizient in einen String verwandelt werden.

+StringTokenizer: Strings zerlegen

```
java.util.StringTokenizer
```

Die Klasse StringTokenizer ist eine nützliche Hilfsklasse, mit der Strings in einzelne **Tokens** zerlegt werden können.

Ein Token wird dabei als **zusammenhängende Sequenz** von Zeichen angesehen, die durch **Trennzeichen**

oder durch das Ende der Zeichenkette begrenzt ist.

Die Klasse StringTokenizer implementiert das Interface Enumeration, so dass sie genauso benutzt werden kann wie die Iteratoren in den Klassen Vector oder Hashtable.

Anlegen eines StringTokenizers

Die Klasse StringTokenizer besitzt drei Konstruktoren:

```
public StringTokenizer(String str)

public StringTokenizer(String str, String delim)

public StringTokenizer(String str,
                       String delim,
                       boolean returnTokens)
```

Die erste Variante übergibt den String str, der tokenisiert werden soll, und bereitet das Objekt für die nachfolgenden Zugriffe auf die einzelnen Tokens vor.

Die zweite Variante erwartet zusätzlich die Übergabe einer Zeichenkette delim, die alle Zeichen enthält, die als Trennzeichen zwischen zwei aufeinanderfolgenden Tokens angesehen werden sollen. In der Variante ohne delim werden die Zeichen '\n', '\r', '\t' und das Leerzeichen als Begrenzer verwendet.

Der dritte Konstruktor enthält einen weiteren Parameter, returnTokens. Wird er auf true gesetzt, geben die Funktionen zur Extraktion der Tokens auch die Trennzeichen zwischen zwei Tokens zurück. Falls der Parameter false ist, werden die Trennzeichen lediglich als Begrenzer angesehen, ohne an den Aufrufer zurückgegeben zu werden.

Zugriff auf Tokens

```
public boolean hasMoreTokens()

public String nextToken()
  throws NoSuchElementException
```

Beispiel:

```
/* Listing1701.java */

import java.util.*;

public class Listing1701
{
     public static void main(String[] args){
          String s = "Dies ist nur ein Test";
          StringTokenizer st = new StringTokenizer(s);
          while (st.hasMoreTokens()) {
               System.out.println(st.nextToken());
          }
     }
}
```

Die Programmausgabe ist:

Dies
ist
nur
ein
Test

Hinweis:
In der Standardbibliothek des JDK gibt es noch einen zweiten Tokenizer, nämlich die Klasse StreamTokenizer aus dem Paket java.io. Im Gegensatz zum StringTokenizer arbeitet sie nicht mit Strings, sondern mit Streams (siehe Kapitel Streams) und läßt sich umfangreicher konfigurieren. Sie kann zwischen Zahlen und Wortsymbolen unterscheiden und auf Wunsch Kommentare (in C/C++-Syntax) ignorieren. Mit ihrer Hilfe lassen sich Tokenizer für einfache Sprachen aufbauen.

Aufgaben

Aufgabe: StringCGI

Aufgabe: StringCGI
Bei der CGI (CommonGatewayProgrammierung) werden oft folg. Strings verarbeitet:
Feld1=Wert1&Feld2=Wert2&Feld3=Wert3

Schreibe ein Programm, das
☑ 2 StringArrays definiert (sFeld und sWert)
☑ Lies einen String von der Konsole ein (Aufbau des Strings: s.o.)
☑ Zerlege den String derart, dass in sFeld die Feldnamen und in sWert die entsprechenden Werte sind.
☑ Anschließend soll der Benutzer eine Zahl eingeben können. Das Programm gibt dann den an dieser Stelle befindlichen Feldnamen und Wert aus.

Zusammenfassung

Liste für jedes Kapitel die Methoden auf und gib eine kurze Beschreibung/Beispiel:

1. String, Texte verarbeiten
1.1. Strings definieren und verwenden
String s;
System.out.println(s);

1.2. Konvertierungen: String zu int,float,... und umgekehrt
s= String.valueOf(3.14);
zahl= Integer.parseInteger(s);

1.3. Stringverkettung mit +

1.4. Stringlänge ermitteln

Ausblick

Im nächsten Kapitel wollen wir das Anwenden der Strings und StringBuffer beim Zugriff auf Textfiles kennen lernen.