

Referenztypen

Prof. Dr. Helmut G. Folz



Referenztypen und Referenzvariablen

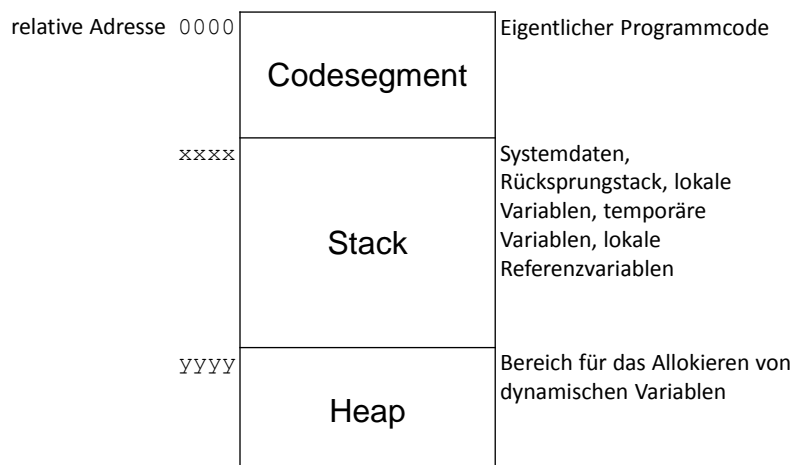
- Nicht-einfache Datentypen sind generell **Referenztypen**.
- **Referenztypen** können sein:
 - ⇒ **Klassen**, denn jede Klasse stellt einen eigenen Datentyp dar.
 - ⇒ **Interfaces**, eine spezielle Art von abstrakten Klassen (hierzu später mehr)
 - ⇒ **Arrays**, also Felder von anderen Datentypen (später mehr dazu)
- Generell müssen Variablen von einem Klassen-, Interface- oder Array-Typ dynamisch mit **new** allokiert werden.

Referenztypen und Referenzvariablen

- **Dynamische Variablen** werden daher nicht explizit definiert und haben auch keinen Namen.
- Der Zugriff auf eine dynamische Variable kann nur über eine Referenzvariable erfolgen.
- **Referenzvariablen** sind Variablen, die als Wert
 - ⇒ eine Referenz auf ein Objekt einer Klasse oder auf ein Array
 - ⇒ oder die sogenannte Null-Referenz null haben kann.

```
String s = null;      // s hat den Wert null
s = new String("Hallo Welt");
    // s zeigt auf ein dynamisch erzeugtes
    // String-Objekt mit dem Inhalt
    // "Hallo Welt"
```

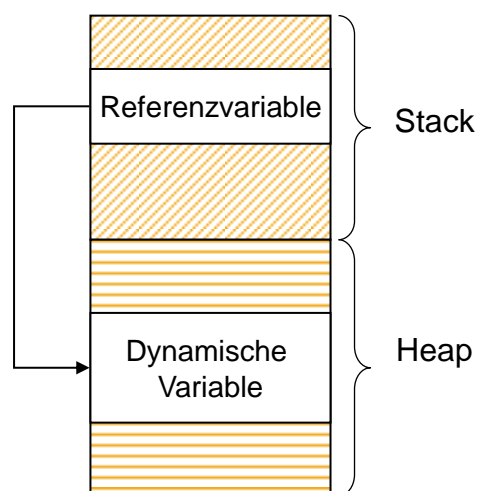
Stack und Heap



Stack und Heap

- Im **Codesegment** wird der eigentliche auszuführende Programmcode abgespeichert.
- Das **Stacksegment** ist für alle automatisch angelegten und automatisch entfernten Variablen zuständig.
 - ⇒ Hier werden alle Variablen von einfachem Datentyp angelegt, die als lokale Variablen innerhalb von Methoden benötigt werden.
 - ⇒ Desweiteren werden hier die lokalen Referenzvariablen abgelegt.
- Aufgabe des **Heaps** ist es, Speicherplatz für die Erzeugung dynamischer Variablen bereitzuhalten, d. h. alle Objekte von Klassen oder Interfaces und alle Arrays werden im Heap angelegt.

Referenzvariable



Referenzvariablen

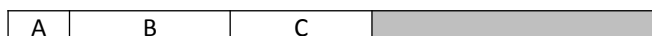
- Der new-Operator, der vom Anwendungsprogramm aufgerufen wird, gibt eine Referenz auf die im Heap erzeugte dynamische Variable zurück.
- An welcher Stelle des Heaps die dynamische Variable angelegt wird, entscheidet die virtuelle Maschine.
- Die Referenz kann in einer Referenzvariablen abgelegt werden.
- Der Zugriff auf die dynamische Variable kann nun über die Referenzvariable erfolgen.
- Es kann mehrere Referenzen auf die selbe dynamische Variable geben.
- Eine Referenz kann entfernt werden durch explizites Setzen auf den Wert null oder durch das automatische Entfernen der Referenzvariablen.

Der Garbage Collector

- Der Heap ist in seiner Größe begrenzt. Mit zunehmender Anzahl an dynamischen Variablen wird der Heap immer mehr belegt.
- Durch das ständige Allokieren und Freigeben von Bereichen aus dem Heap wird dieser im Laufe der Zeit immer mehr zerstückelt.



- Der Garbage Collector hat die Aufgaben,
 - ⇒ nicht mehr referenzierte dynamische Variablen freizugeben,
 - ⇒ den Speicher neu ordnen, so dass wieder größere unbenutzte Speicherbereiche entstehen können.



Felder (Arrays)

- Ein **Feld** (Vektor, engl. *array*) ist ein Objekt, das aus mehreren Datenelementen des gleichen Datentyps besteht.
- Die einzelnen Datenelemente heißen **Feldelemente**. Sie können über einen sogenannten **Index** angesprochen werden.
- Definition einer Referenzvariablen auf ein eindimensionales Feld

Typname[] **fieldName** ;

- ⇒ fieldName ist eine Referenzvariable, die ein Feld aus Elementen des Typs *typname* referenzieren kann.
- ⇒ Die Länge des Feldes kann erst beim Allokieren des Felds angegeben werden.

Felder (Arrays)

Allokieren eines Felds

```
int [ ] tab = new int[10];
```

Die Referenzvariable tab zeigt auf ein int-Feld mit 10 Elementen.

tab[0]	0
tab[1]	0
tab[2]	0
tab[3]	0
tab[4]	0
tab[5]	0
tab[6]	0
tab[7]	0
tab[8]	0
tab[9]	0

Beispiel: Feld eingeben (1)

```
public class FeldEingabe {
    private Scanner input = new Scanner(System.in);

    public void start() {
        final int MAXANZ = 10;
        int[] tab = new int[MAXANZ]; // Felddefinition
        int zahl, i = 0, anzahl; // Zahl, Index, Anzahl

        // Zahlen einlesen:
        System.out.println("Bis zu 10 Zahlen eingeben "
            + "(Abbruch mit einer negativen Zahl:");
        zahl = input.nextInt();
        while (zahl >= 0) {
            tab[i++] = zahl;
            zahl = input.nextInt();
        }
        anzahl = i; // tatsaechlich eingegebene Anzahl an Zahlen
    }
}
```

Beispiel: Feld eingeben (2)

```
System.out.println("Eingegeben wurden die folgenden "
    + anzahl + " Zahlen:");

for( i = 0; i < anzahl; ++i)
    System.out.print(tab[i] + "\t");
System.out.println();
}

public static void main(String[] args) {
    new FeldEingabe().start();
}
}
```

Initialisierungsliste

Implizites Erzeugen eines Feldes über eine Initialisierungsliste

Ein Feld kann auch mit einer Initialisierungsliste erzeugt und gleichzeitig initialisiert werden

```
double[] vec = { 1.5, 2.5, 3.5 };
```

vec[0]	1.5
vec[1]	2.5
vec[2]	3.5

Auch ein derart angelegtes Feld wird dynamisch auf dem Heap angelegt.

Beispiel: Elementares zu Feldern (1)

```
/** Elementares zu Feldern */
public class FeldTest1 {
    private Scanner input = new Scanner(System.in);

    /**
     * Initialisiere ein Feld mit Zufallszahlen zwischen
     * 0 und 100
     */
    public void initFeld (int[] tab) {
        for (int i = 0; i < tab.length; i++)
            tab[i] = (int)Math.round(Math.random() * 100);
    }
}
```

Beispiel: Elementares zu Feldern (2)

```
/** Bestimme das Minimum in einem Feld */  
public int min (int[] tab) {  
    int minWert = tab[0];  
    for (int i = 1; i < tab.length; i++)  
        if (tab[i] < minWert)  
            minWert = tab[i];  
    return minWert;  
}
```

Beispiel: Elementares zu Feldern (3)

```
/** Mittelwert in einem Feld bestimmen*/  
public double mittelwert (int[] tab) {  
    double summe = 0.0;  
    for (int i = 0; i < tab.length; i++)  
        summe += tab[i];  
    return summe / tab.length;  
}
```


Beispiel: Elementares zu Feldern (4)

```
/** Gebe ein Feld auf die Standardausgabe aus */
public void ausgabeFeld(int[] tab) {
    for (int i = 0; i < tab.length; i++) {
        System.out.print(tab[i] + "\t");
        if ((i+1) % 10 == 0)
            System.out.println();
    }
    System.out.println();
}
```

Beispiel: Elementares zu Feldern (5)

```
/** Ein Feld allokalieren und einlesen*/
public int[] readFeld() {
    System.out.print("Feldgroesse: ");
    int groesse = input.nextInt();
    int[] tab = new int[groesse];
    for (int i=0; i < tab.length; i++) {
        System.out.print("tab[" + i + "] = ");
        tab[i] = input.nextInt();
    }
    return tab;
}
```

Beispiel: Elementares zu Feldern (6)

```
public void start() {
    int[] tab1 = new int[100],
        tab2 = readFeld();

    initFeld(tab1);
    System.out.println("tab1:");
    ausgabeFeld(tab1);
    System.out.println("tab2:");
    ausgabeFeld(tab2);

    System.out.println("Minimum tab1: " + min(tab1));
    System.out.println("Minimum tab2: " + min(tab2));

    System.out.println("Mittelwert tab1: "
        + mittelwert(tab1));
    System.out.println("Mittelwert tab2: "
        + mittelwert(tab2));
}
```

Beispiel: Elementares zu Feldern (7)

```
public static void main(String[] args) {
    new FeldTest1().start();
}
/*
Feldgroesse: 5
tab[0] = 2
tab[1] = 4
tab[2] = 6
tab[3] = 8
tab[4] = 10
tab1:
52  71  63  43  81  58  33  71  85  81
tab2:
2    4    6    8    10
Minimum tab1: 33
Minimum tab2: 2
Mittelwert tab1: 63.8
Mittelwert tab2: 6.0 */
```

Felder von Referenztypen

Was wird mit der folgenden Definition erzeugt?

```
String[] stab = new String[5];
```

Antwort: Ein Feld aus Referenzvariablen auf String-Objekte mit dem jeweiligen Initialwert null.

0	null
1	null
2	null
3	null
4	null

Felder von Referenztypen

Explizites Allokieren von Feldelementen von Referenztyp

```
StringBuffer[] sbtab = new StringBuffer[10];  
for (int i=0; i < sbtab.length; i++)  
    sbtab[i] = new StringBuffer();  
  
sbtab[0].append("Jetzt geht es!");
```

Felder von Referenztypen

Implizites Erzeugen über eine Initialisierungsliste

```
String hallo = "hallo";  
String[] stab = { hallo,  
                  new String("HTW"),  
                  "Informatik"};
```

Mehrdimensionale Felder

- Mehrdimensionale Felder werden einfach als Felder von Feldern definiert.

```
double[][] matrix = new double[3][5];  
// 3 x 5 - Matrix  
// 3 Felder mit je 5 double-Werten
```

	0	1	2	3	4
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0

Beispiel: Mehrdimensionale Felder (1)

```
public class FeldTest2 {
    private Scanner input = new Scanner(System.in);
    private int[][] matrix1;
    private int[][] matrix2;
    /**
     * Allgemeine Funktion zum Ausgeben von beliebigen
     * int-Matrizen
     */
    public void matrixAusgabe(int[][] mat) {
        int i, j;
        for (i = 0; i < mat.length; i++) {
            for (j = 0; j < mat[i].length; j++) {
                System.out.print(mat[i][j] + " ");
                System.out.println();
            }
            System.out.println();
        }
    }
}
```

Beispiel: Mehrdimensionale Felder (2)

```
/**
 * Allgemeine Funktion zum Einlesen von beliebigen
 * int-Matrizen
 */
public void matrixEingabe(int[][] mat) {
    int i, j;
    for (i = 0; i < mat.length; i++) {
        System.out.print("mat[" + i + "]: "
            + mat[i].length + " Werte eingeben: ");
        for (j = 0; j < mat[i].length; j++) {
            mat[i][j] = input.nextInt();
        }
    }
}
```

Beispiel: Mehrdimensionale Felder (3)

```
public void start() {
    matrix1 = new int[2][3];
    matrixEingabe(matrix1);
    matrixAusgabe(matrix1);

    matrix2 = new int[4][5];
    matrixEingabe(matrix2);
    matrixAusgabe(matrix2);
}

public static void main(String[] args) {
    new FeldTest2().start();
}

/*
mat[0]: 3 Werte eingeben: 1 2 3
mat[1]: 3 Werte eingeben: 4 5 6
1 2 3
4 5 6 ... */
```

Mehrdimensionale Felder

- Initialisierung von mehrdimensionalen Feldern

```
int[][] mat = { { 1, 2, 3 },
                { 4, 5, 6 } };
```

- Beispiel: 3-dimensionales Feld:

```
int[][][] matrix3D = new int[3][4][5];
// 3 * 4 * 5 Elemente
```

- Offen allokierte Felder

Was bedeutet `int [][] mat = new int[5][]` ?

Beispiel: Dreieckige Matrizen (1)

```
public class FeldTest3 {
    public void start () {
        int i, j;
        // dreieckiges 2-dimensionales Array

        int[][] dreieck = new int[5][]; // Zeilen allokiert
        for (i = 0; i < dreieck.length; i++) {
            dreieck[i] = new int[2*i+1];
            // 0-te Zeile: 1 Spalte
            // 1-te Zeile: 3 Spalten
            // ...
            // 4-te Zeile: 9 Spalten
            for (j = 0; j < dreieck[i].length; j++)
                dreieck[i][j] = i + j;
        }
        FeldTest2 f = new FeldTest2();
        f.matrixAusgabe(dreieck);
    }
}
```

Beispiel: Dreieckige Matrizen (2)

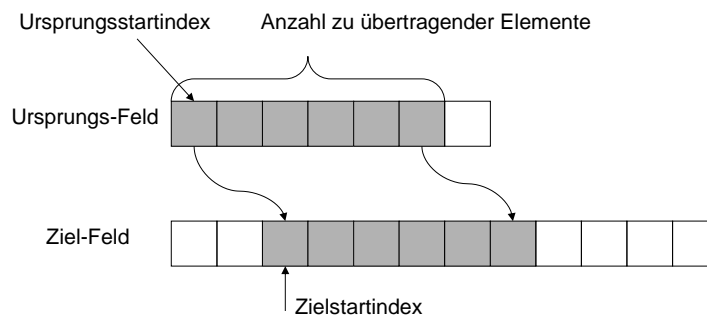
```
public static void main(String[] args) {
    new FeldTest3().start();
}
/*
0
1 2 3
2 3 4 5 6
3 4 5 6 7 8 9
4 5 6 7 8 9 10 11 12 */
```

Beispiel: Kommandozeilenparameter

```
/** Echo.java: Ausgeben der Kommandozeilenargumente */  
public class Echo {  
    public static void main (String[] args) {  
        for (int i=0; i < args.length; i++)  
            System.out.println("args["+i+"]= |"  
                               + args[i] + "| ");  
        System.out.println();  
    }  
}
```

Kopieren von Arrays

```
System.arraycopy(Ursprungs-Feld, Ursprungsstartindex,  
                 Ziel-Feld      , Zielstartindex,  
                 Anzahl-zu-übertragender-Elemente);
```



Kopieren von Arrays

```
String[] namen = { "Adam", "Berta", "Caesar"};
String[] neueNamen = new String[2 * namen.length];

System.arraycopy (namen,      0,
                  neueNamen, 0, namen.length);
namen = neueNamen;
// Das Ursprungs-Feld wird dem Garbage Collector
// übergeben
```

Algorithmen mit Feldern

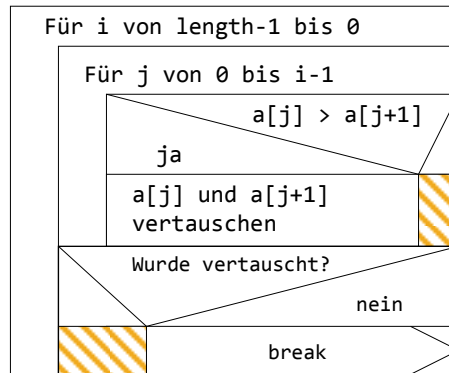
Bubblesort (Sortieren durch Austauschen)

Das einfachste und gleichzeitig eines der langsamsten Verfahren, um ein Feld zu sortieren ist der sogenannte BubbleSort-Algorithmus, der im Folgenden vorgestellt wird.

1. Gegeben sei ein unsortiertes Feld beliebiger Größe.
2. Im ersten Durchgang werden jeweils zwei benachbarte Elemente verglichen und vertauscht, wenn das erste Element größer ist. Nach dem Ende des ersten Durchgangs steht das größte Element am Ende des Felds.
3. Im zweiten Durchgang wird das Verfahren wiederholt allerdings nur bis zum vorletzten Element.
4. Das wird wiederholt bis man entweder keine Vertauschungen vorgenommen hat oder bis man am Anfang des Feldes angelangt ist.

Algorithmen mit Feldern

Bubble Sort



BubbleSort

```
public void bubbleSort(int[] tab) {
    int i, j, tmp;
    for (i = tab.length-1; i >= 0; i--) {
        boolean swapped = false;
        for (j = 0; j < i; j++) {
            if (tab[j] > tab[j+1]) {
                tmp = tab[j];
                tab[j] = tab[j+1];
                tab[j+1] = tmp;
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}
```

SelectionSort

- Zunächst wird das kleinste Feldelement gesucht und mit dem ersten Element vertauscht.
- Danach wird ab dem 2. Element das zweitkleinste Element gesucht usw.
- Gegenüber dem Bubblesort werden immer nur length Vertauschungen benötigt.

Selection Sort

Für i von 0 bis length-1

min = i

Für j von i+1 bis length

a[j] < a[min]

ja

min = j

a[i] und a[min]
vertauschen

Selection Sort

```
public void selectionSort(int[] tab) {
    int i, j, min, tmp;
    for (i = 0; i < tab.length-1; i++) {
        min = i;
        for (j = i+1; j < tab.length; j++) {
            if (tab[j] < tab[min])
                min = j;
        }
        if (min != i) {
            tmp = tab[min];
            tab[min] = tab[i];
            tab[i] = tmp;
        }
    }
}
```

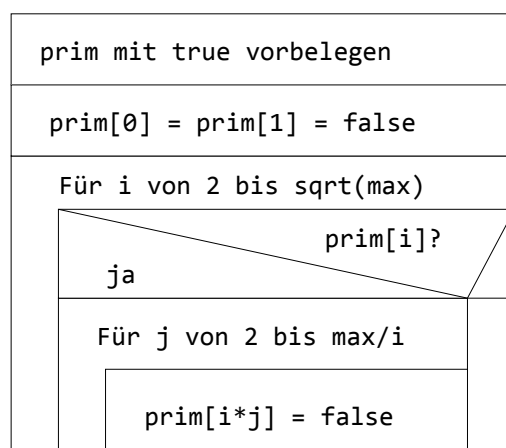
Test auf Sortierung

```
public boolean isSorted(int[] tab) {
    boolean sorted = true;
    for (int i = 0; i < tab.length-1; i++) {
        if (tab[i] > tab[i+1]) {
            sorted = false;
            break;
        }
    }
    return sorted;
}
```

Das Sieb des Eratosthenes

- Ein klassischer Algorithmus zum Bestimmen der ersten n Primzahlen ist das Sieb des Eratosthenes (276 - 195 v. Chr):
 - ⇒ Belege ein boolean-Feld **prim** vollständig mit **true** vor.
 - ⇒ Streiche die Zahl 1, d. h. besetze **prim[1]** mit **false**
 - ⇒ Nehme die nächstgrößere noch nicht gestrichene Zahl, dies ist automatisch eine Primzahl und streiche alle Vielfachen dieser Zahl unterhalb der Grenze.
 - ⇒ Wiederhole dies solange bis man die Quadratwurzel der oberen Grenze erreicht hat.

Das Sieb des Eratosthenes



Das Sieb des Eratosthenes

```
public void sieben () {
    int i, j;
    int max = prim.length - 1;
    // maximal zu untersuchende Zahl
    for (i = 0; i < prim.length; i++)
        prim[i] = true;

    prim[0] = prim[1] = false;

    int grenze = (int)Math.round(Math.sqrt(max));
    for (i = 2; i <= grenze; i++) {
        if (prim[i]) {
            for (j = 2; j <= max/i; j++)
                prim[i*j] = false;
        }
    }
}
```

Zeichenketten

- In Java gibt es für die Darstellung von Zeichenketten drei Klassen:
 - ⇒ `java.lang.String` nicht veränderliche Zeichenkette
 - ⇒ `java.lang.StringBuffer` veränderliche Zeichenkette (thread-sicher)
 - ⇒ `java.lang.StringBuilder` veränderliche Zeichenkette (nicht thread-sicher aber schneller als `StringBuffer`)
- Generell sind Zeichenketten auch Zeichenkettenlitterale anders als bei C/C++ vollwertige Objekte und außerdem aus 16-Bit-Zeichen zusammengesetzt.
- Auch das Nullzeichen '\0', das bei C/C++ üblicherweise Zeichenketten abschließt, ist bei Java nicht vorhanden.
- Für `java.lang.String` gibt es die zusätzlichen Sprachkonstrukte '+' und '+=' zum Konkatenieren von Zeichenketten.

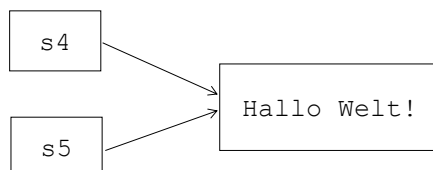
Beispiel: Alphabet

```
public class Alphabet {  
    public void start() {  
        String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
        System.out.println(alphabet);  
        System.out.println("Laenge dieser Zeichenkette: "  
            + alphabet.length());  
        System.out.println("Das 4. Zeichen ist ein "  
            + alphabet.charAt(4));  
        System.out.println("Das Zeichen Z steht an der Stelle "  
            + alphabet.indexOf('Z'));  
        System.out.println("Das String NO steht an der Stelle "  
            + alphabet.indexOf("NO"));  
    }  
    ...  
}
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
Laenge dieser Zeichenkette: 26  
Das 4. Zeichen ist ein E  
Das Zeichen Z steht an der Stelle 25  
Das String NO steht an der Stelle 13
```

Konstruktion von String-Objekten

- `String s1;`
Definiert eine Referenzvariable, die auf ein String-Objekt verweisen kann
- `String s2 = new String("Hallo Welt!");`
Kopie der Stringkonstante "Hallo Welt!"
- `String s3 = new String(s2);`
Kopie des String-Objektes s2
- Was genau passiert hier?:
`String s4 = "Hallo Welt!";`
`String s5 = "Hallo Welt!";`



Vergleichen von Strings

- Wenn man zwei Referenzvariablen, die auf String-Objekte zeigen mit `==` vergleicht, so werden nur deren Werte, d. h. die Referenzen verglichen und nicht die Inhalt der String-Objekte.

```
String x = "abc"; // Referenz auf Literal!  
String y = new String("abc");  
           // Objekt mit gleichem Inhalt allokiert
```

```
if (x == y)  
    // Vergleich der Referenzen! - ist false!
```

- Warum funktioniert das Folgende trotzdem?

```
String z = "abc";  
if (z == x)    // true, warum?
```

Vergleichen von Strings

- Zum inhaltlichen Vergleich zweier String-Objekte muss die Methode `equals()` der Klasse `String` verwendet werden:

```
if (x.equals(y)) // inhaltlicher Vergleich!
```


Strings sind unveränderlich, oder?

Was genau passiert hier?

```
String s1 = "Hallo ";  
String s2 = "Welt";  
  
s1 = s1 + s2; // wird s1 verändert?  
s1 += "!";    // und hier?
```

Die Klasse String

<i>Methode</i>	<i>Beschreibung</i>
char charAt (int i)	i-tes Zeichen des Strings (0,...,length()-1) Bei Zugriff auf $i < 0$ oder $i \geq \text{length}()$, Ausnahme <code>IndexOutOfBoundsException</code>
int compareTo (String s)	< 0 falls <code>String < s</code> $= 0$ falls <code>String = s</code> > 0 falls <code>String > s</code>
int compareToIgnoreCase (String str)	Unabhängig von Groß- oder Kleinschreibung vergleichen < 0 falls <code>String < s</code> $= 0$ falls <code>String = s</code> > 0 falls <code>String > s</code>
boolean equals (Object o)	Inhaltlicher Vergleich
boolean equalsIgnoreCase (Object o)	Inhaltlicher Vergleich ohne Berücksichtigung von Groß- und Kleinschreibung

Die Klasse String

<i>Methode</i>	<i>Beschreibung</i>
int <code>length()</code>	Länge in Zeichen
int <code>indexOf</code> (int ch)	erstes Auftreten des Zeichen ch oder -1
int <code>indexOf</code> (String str)	erstes Auftreten des Strings str oder -1
int <code>lastIndexOf</code> (int ch)	letztes Auftreten des Zeichen ch oder -1
int <code>lastIndexOf</code> (String str)	letztes Auftreten des Strings str oder -1
boolean <code>startsWith</code> (String prefix)	Beginnt String beginnt mit einem bestimmten Präfix?
boolean <code>endsWith</code> (String suffix)	Endet String beginnt mit einem bestimmten Suffix?

Die Klasse String

<i>Methode</i>	<i>Beschreibung</i>
String <code>substring</code> (int anf)	Teilzeichenkette von Position anf (inkl.) bis zum Ende der Zeichenkette ggf Ausnahme <code>StringIndexOutOfBoundsException</code>
String <code>substring</code> (int anf, int ende)	Teilzeichenkette von Position anf (inkl.) bis Position ende (exkl.) ggf. Ausnahme <code>StringIndexOutOfBoundsException</code>
char[] <code>toCharArray()</code>	String in ein char-Array umwandeln
String <code>toLowerCase()</code>	alle Großbuchstaben des String in Kleinbuchstaben umwandeln.
String <code>toUpperCase()</code>	alle Kleinbuchstaben des String in Großbuchstaben umwandeln.
String <code>trim()</code>	Leerzeichen am String-Anfang und -Ende entfernen

Die Klasse String

<i>Methode</i>	<i>Beschreibung</i>
boolean contains (CharSequence s)	Enthält das String eine bestimmte Zeichenkette?
boolean contentEquals (CharSequence cs)	Inhaltliche Gleichheit zu Zeichenketten überprüfen.
static String format (String format, Object... args)	Zeichenkette nach Vorgabe formatieren
static String format (Locale l, String format, Object... args)	Zeichenkette nach Vorgabe formatieren (internationalisiert)

Die Klasse String

<i>Methode</i>	<i>Beschreibung</i>
String replace (char oldChar, char newChar)	Ersetze alle Auftreten von oldChar durch newChar
String replace (CharSequence target, CharSequence replacement)	Ersetze alle Substrings target durch replacement
String replaceAll (String regex, String replacement)	Ersetze alle Substrings, die den regulären Ausdruck regex erfüllen durch replacement
String replaceFirst (String regex, String replacement)	Ersetze den ersten Substring, der regex erfüllt durch replacement

Die Klasse String

<i>Methode</i>	<i>Beschreibung</i>
boolean matches (String regex)	Erfüllt das String den regulären Ausdruck?
String[] split (String regex) String[] split (String regex, int limit)	String mit Hilfe eines regulären Ausdruck aufsplitten

Beispiel: Substrings

```
public class Substrings {
    public void start() {
        String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        System.out.println(alphabet);
        System.out.println("Substring von 4 bis 8: "
            + alphabet.substring(4,8));
        System.out.println("Substring von 0 bis 8: "
            + alphabet.substring(0,8));
        System.out.println("Substring von 8 bis zum Ende: "
            + alphabet.substring(8));
    }
    public static void main(String[] args) {
        new Substrings().start();
    }
} /*
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Substring von 4 bis 8: EFGH
Substring von 0 bis 8: ABCDEFGH
Substring von 8 bis zum Ende: IJKLMNOPQRSTUVWXYZ */
```

Einlesen von Zeichenketten

Zusätzliche Methoden der Klasse Scanner:

<i>Methode</i>	<i>Beschreibung</i>
String <code>next()</code>	Liefert das nächste Token als String zurück
String <code>nextLine()</code>	Liest eine komplette Zeile von der Standardeingabe ein.

Beispiel: Einlesen Zeichenketten

```
public class StringEingabe {
    private Scanner input = new Scanner(System.in);

    public void start() {
        String wort;
        System.out.println(
            "Woerter eingeben bis ende eingegeben");

        do {
            wort = input.next();
            System.out.println(wort.toUpperCase());
        } while (!wort.equals("ende"));

        System.out.print("Zeile eingeben: ");
        String zeile = input.nextLine();
        System.out.println("Laenge: " + zeile.length());
        System.out.print("Noch eine Zeile: ");
        zeile = input.nextLine();
        System.out.println("Laenge: " + zeile.length());
    }
}
```

Beispiel: URLs analysieren

```
/** Einlesen und Analysieren von URLs */
public class URLAnalyse {
    private Scanner input = new Scanner(System.in);
    public void start () {
        String url = null;
        int pos1, pos2;
        do {
            System.out.print("URL eingeben: ");
            url = input.next();
            pos1 = url.indexOf(":/");
            System.out.println("Protokoll: " + url.substring(0,pos1));
            pos2 = url.indexOf('/', pos1+3);
            if (pos2 == -1)
                System.out.println("Name: " + url.substring(pos1+3 ));
            else {
                System.out.println("Name: "
                                   + url.substring(pos1+3, pos2));
                System.out.println("File: " + url.substring(pos2));
            }
        } while (weitermachen());
    }
}
```

Zeichenketten und Typumwandlungen

- Für alle einfachen Datentypen gibt es Standardkonvertierungen in Zeichenketten, nämlich die jeweilige Methode `toString()` der zugehörigen Hüllenklasse.
`int i = 17;`
`String s = new Integer(i).toString();`
- Alternativ gibt es auch für jeden einfachen Datentyp Klassenmethoden in der Klasse `String`, die das selbe leisten
`String s = String.valueOf (i);`
- Eine spezielle Anwendung dieser Konvertierungen haben wir schon mehrmals gesehen:

```
System.out.println("i = " + i);
```

Zeichenketten und Typumwandlungen

Hüllenklasse	Methode
Boolean	static Boolean valueOf(String s) static boolean parseBoolean(String s)
Byte	static Byte valueOf(String s) static byte parseByte(String s)
Short	static Short valueOf(String s) static short parseShort(String s)
Integer	static Integer valueOf(String s) static int parseInt(String s)
Long	static Long valueOf(String s) static long parseLong(String s)
Double	static Double valueOf(String s) static double parseDouble(String s)
Float	static Float valueOf(String s) static float parseFloat(String s)

Beispiel: ParseTest (1)

```
public class ParseTest {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        String eingabe = null;

        do {
            try {
                System.out.print("Zahl eingeben: ");
                eingabe = input.next();

                double d = Double.parseDouble(eingabe);
                System.out.println("double-Wert: " + d);

                float f = Float.parseFloat(eingabe);
                System.out.println("float-Wert : " + f);

                long l = Long.parseLong(eingabe);
                System.out.println("Long-Wert : " + l);
            }
        } while (eingabe != null);
    }
}
```

Beispiel: ParseTest (2)

```
int i = Integer.parseInt(eingabe);
System.out.println("int-Wert : " + i);

short s = Short.parseShort(eingabe);
System.out.println("short-Wert : " + s);

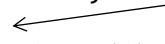
byte b = Byte.parseByte(eingabe);
System.out.println("byte-Wert : " + b);

} catch (NumberFormatException e) {
    System.out.println("Falsches Zahlenformat: " + e);
} catch (Exception e) {
    System.out.println("Sonstige Ausnahme: " + e);
}
} while (!eingabe.equals("0"));
}
```

Autoboxing und Autounboxing

- Automatisches Ein- und Auspacken von einfachen Typen in und aus Wrapper-Klassen.
- An vielen Stellen wird automatisch zwischen einfachen Typen und den zugehörigen Wrapper-Objekten konvertiert.
- Erwartet eine Methode beispielsweise einen Integer-Wert als Argument, kann außer einem Integer auch direkt ein `int` übergeben werden; und er wird ohne Zutun des Entwicklers in einen gleichwertigen Integer konvertiert.
- Auch in umgekehrter Richtung funktioniert das, etwa wenn in einem arithmetischen Ausdruck ein `double` erwartet, aber ein `Double` übergeben wird.

Autoboxing und Autounboxing

```
public class AutoboxingTest {  
    public static void main(String[] args) {  
        int i = 5;  
        Integer ii = i;  
        System.out.println("i = " + i  
                             + ", ii = " + ii);  
  
        Double dd = new Double(3.14);  
  
        double d = dd * 5;  
        System.out.println("d = " + d  
                             + ", dd = " + dd);  
  
        Integer jj = null;  
        int j = jj;    
        System.out.println(j);  
    }  
}
```

Was passiert hier?

StringBuffer bzw. StringBuilder

- Veränderliche Zeichenketten werden mit der Klasse StringBuffer (ab JDK 1.5 auch StringBuilder) realisiert.
- StringBuffer hat dabei genau die gleiche Schnittstelle wie StringBuilder ist aber thread-sicher programmiert, d. h. StringBuffer ist weniger performant als StringBuilder aber bei Multithreading zu bevorzugen.
- Die Länge der Zeichenkette ist nicht festgelegt. Sie vergrößert sich automatisch, wenn Zeichen hinzugefügt werden und der Platz nicht ausreicht.
- Der Inhalt eines StringBuffer-Objektes lässt sich verändern.
- Es gibt keinen + und += Operator zum Verknüpfen der Objekte

StringBuffer bzw. StringBuilder

- Konstruktion:

```
StringBuffer s = null;           // leere Referenz
```

```
StringBuffer s2 = new StringBuffer (10);  
                // leeres String der Länge 10
```

```
StringBuffer s3 = new StringBuffer ("abc");
```

StringBuffer bzw. StringBuilder

Methoden	Beschreibung
<code>int length();</code>	Länge
<code>void setLength (int newLength)</code>	Länge neu setzen (abschneiden / verlängern)
<code>int capacity()</code>	aktuelle Speicherkapazität des StringBuffers
<code>char charAt (int i);</code>	i-tes Zeichen
<code>void setCharAt (int i, char ch)</code>	setze i-tes Zeichen auf ch
<code>StringBuffer append (String str)</code>	hänge str an , analoge Varianten von append gibt es auch für char, byte, short, int, long, float, double, char[], Object
<code>StringBuffer insert (int offset, String str)</code>	füge str an hinter Stelle offset ein, analoge Varianten von insert gibt es auch für char, byte, short, int, long, float, double, char[], Object

StringBuffer bzw. StringBuilder

Methoden	Beschreibung
StringBuffer delete (int start, int end)	lösche ab Position start bis zu Position end (exklusive)
StringBuffer deleteCharAt (int index)	Zeichen an der Stelle index entfernen
StringBuffer replace (int start, int end, String str)	Ersetze die Positionen start, .. end-1 durch das String str
StringBuffer reverse ()	Reihenfolge der Zeichen umdrehen
String substring (int anf)	Teilzeichenkette von Position anf (inkl.) bis zum Ende der Zeichenkette ggf Ausnahme StringIndexOutOfBoundsException
String substring (int anf, int ende)	Teilzeichenkette von Position anf (inkl.) bis Position ende (exkl.) ggf. Ausnahme StringIndexOutOfBoundsException
String toString ()	Inhalt des StringBuffer-Objektes in ein String-Objekt umwandeln

StringBuffer bzw. StringBuilder

Methoden	Beschreibung
int indexOf (String str)	erstes Auftreten des Strings str oder -1
int indexOf (String str, int fromIndex)	erstes Auftreten des Strings str ab der Position fromIndex oder -1
int lastIndexOf (String str)	letztes Auftreten des Strings str oder -1
int lastIndexOf (String str, int fromIndex)	letztes Auftreten des Strings str ab der Position fromIndex oder -1

Beispiel: StringBufferTest (1)

```
/** Test der Klasse StringBuffer */
public class StringBufferTest {
    public void start() {
        final StringBuffer sb =
            new StringBuffer("Hallo Welt");
        System.out.println(sb);
        // sb = new StringBuffer("HTW");
        // geht nicht, da sb final

        sb.delete(6,10);
        // geht, da das Objekt selbst nicht final

        System.out.println(sb); // Hallo
        System.out.println("Laenge: " + sb.length());
        System.out.println("Kapazitaet: " + sb.capacity());
        sb.append("Fachbereich M");
        System.out.println(sb + " " + sb.length() + " "
                           + sb.capacity());
        int pos = sb.indexOf("M");
    }
}
```

Beispiel: StringBufferTest (2)

```
        sb.replace(pos, pos+1,
                    "Grundlagen, Informatik, Sensortechnik");
        System.out.println(sb + " " + sb.length() + " "
                           + sb.capacity());

        sb.append('!');
        System.out.println(sb + " " + sb.length() + " "
                           + sb.capacity());
    }
    public static void main(String[] args) {
        StringBufferTest sb = new StringBufferTest();
        sb.start();
    }
}
```

```
Hallo Welt
Hallo
Laenge: 6
Kapazitaet: 26
Hallo Fachbereich M 19 26
Hallo Fachbereich Grundlagen, Informatik, Sensortechnik 55 55
Hallo Fachbereich Grundlagen, Informatik, Sensortechnik! 56 112
```

Anwendung in der Bank-Klasse

```
class Bank { // bisherige Implementierung von toString
    ...
    public String toString() {
        String s = "Bank: " + name + '\n';
        for (int i = 0; i < anzKonten; i++) {
            s += i + ": " + kontoTab[i] + '\n';
        }
        return s;
    }
}
```

Anwendung in der Bank-Klasse

```
class Bank { // bessere Implementierung von toString
    ...
    public String toString() {
        StringBuilder sb = new StringBuilder("Bank: ");
        sb.append(name).append("\n");
        for (int i = 0; i < anzKonten; i++) {
            sb.append(i).append(": ")
              .append(kontoTab[i]).append("\n");
        }
        return sb.toString();
    }
}
```

Beispiel: MethodenUndReferenzen (1)

```
/** Besonderheiten von Referenztypen bei Methodenaufrufen
 *  aufzeigen */
public class MethodenUndReferenzen {
    public void start() {
        StringBuffer s1 = new StringBuffer(
            "Praktische Informatik");
        System.out.println("s1 vor f1(): " + s1);
        f1(s1);
        System.out.println("s1 nach f1(): " + s1);
        f2(s1);
        System.out.println("s1 nach f2(): " + s1);
        StringBuffer s2 = f3(s1);
        System.out.println("s1 nach f3(): " + s1);
        System.out.println("s2 nach f3(): " + s2);

        if (s1 == s2)
            System.out.println("s1 und s2 sind identisch");
    }
}
```

Beispiel: MethodenUndReferenzen (2)

```
public void f1(StringBuffer sb) {
    sb = new StringBuffer("Sensortechnik");
}

public void f2(StringBuffer sb) {
    sb.delete(0, 11);
}

public StringBuffer f3(StringBuffer sb) {
    return sb.replace(0, 5, "Mathe");
}

public static void main(String[] args) {
    new MethodenUndReferenzen().start();
}
```

Beispiel: MethodenUndReferenzen (3)

```
/*  
s1 vor f1(): Praktische Informatik  
s1 nach f1(): Praktische Informatik  
s1 nach f2(): Informatik  
s1 nach f3(): Mathematik  
s2 nach f3(): Mathematik  
s1 und s2 sind identisch  
*/
```