Zeichenketten und Felder

- Die Klasse `String`
- Die Klasse `StringBuffer`
- Felder
- Referenztypen
- Die Klasse `Vector`

Zeichenketten

Die Klasse `String`

Die Klasse `String` umfasst Zeichenketten, die sich nach ihrer Erzeugung nicht ändern, und unterstützt Operationen auf diesen.

Neue Objekte der Klasse `String` können *implizit* entweder durch in Anführungszeichen eingeschlossenen Zeichenketten (wie z. B. `"Java"`) oder durch Anwendung der Operatoren + und += auf zwei Objekte der Klasse `String` erzeugt werden.

Man kann Objekte der Klasse `String` auch *explizit* durch Anwendung des Operators `new` erzeugen.

Beispiel:

Die Deklaration

```
String str = "abc";
```

ist äquivalent zu

```
char[] data = {'a', 'b', 'c'};
String str = new String(data);
```

Zeichenketten

Die Klasse `String`

wichtige Konstruktoren der Klasse `String`

`public String()`

- Erzeugt eine Zeichenkette, die keine Zeichen enthält.

`public String(char[] feld)`

- Erzeugt eine Zeichenkette, die die Buchstaben des `char`-Feldes `feld` enthält; das ganze Feld wird übernommen.

`public String(char[] feld, int offset, int count)`

- Erzeugt eine Zeichenkette, die die Buchstaben des `char`-Feldes `feld` enthält; aus dem Feld werden `count` Elemente ab Index `offset` übernommen.

`public String(String text)`

- Erzeugt einen String mit der angegebenen Zeichenkette `text`.

`public String(StringBuffer buffer)`

- Wandelt den Puffer `buffer` vom Typ `StringBuffer` in einen String um.

Zeichenketten

Die Klasse `String`

`String`-Methoden für Vergleiche

```
public boolean equals(Object o)
public boolean equalsIgnoreCase(String text)
```

- Liefern `true`, falls das Argument denselben Inhalt wie das aktuelle Objekt (`this`) hat.
- Die zweite Methode liefert auch `true`, wenn die Strings bis auf Groß-/Kleinschreibung übereinstimmen.

```
public int compareTo(String text)
```

- Vergleicht zwei Zeichenketten und liefert:
  - 0, wenn die beiden Strings gleich sind,
  - einen Wert < 0, wenn das aktuelle Objekt lexikographisch kleiner ist als `text`,
  - einen Wert > 0, wenn das aktuelle Objekt lexikographisch größer ist als `text`.

```
public boolean startsWith(String prefix)
public boolean startsWith(String prefix,
   int toffset)
public boolean endsWith(String suffix)
```

- Liefern `true`, wenn das aktuelle Objekt (ab dem Index 0 bzw `toffset`) mit `prefix` beginnt bzw. mit `suffix` endet.

Zeichenketten

Die Klasse `String`

`String`-Methoden für Vergleiche

```
public boolean regionMatches(
   boolean ignoreCase, int toffset,
   String other, int ooffset, int len)
public boolean regionMatches(int toffset,
   String other, int ooffset, int len)
```

- Vergleichen den Teilstring des aktuellen Objekts (`this`) ab dem Index `toffset` mit dem Teilstring von `other`, beginnend ab `ooffset` mit der Länge `len`.
- Wird in der ersten Variante für den Parameter `ignoreCase` der Wert `true` übergeben, wird beim Vergleich nicht zwischen Groß-/Kleinschreibung unterschieden.

Zeichenketten

Die Klasse `String`

Beispiel: `StringDemo1`

```
public class StringDemo1
{
   public static void main(String[] argv)
   {
   // Konstruktoraufrufe von String
   // ----------------------------
      String leererString = new String();
      String stringRef;     // lediglich Referenz
      String s1 = new String("Das ist ein String");
      String s2 = new String("zyx");
      boolean test;

      System.out.println("leererString: \""
         + leererString + "\"");
      System.out.println("s1    : \""
         + s1 + "\"");
      System.out.println("s2    : \""
         + s2 + "\"");
      stringRef = s2;
      // stringRef zeigt auf dasselbe Objekt wie s2

      System.out.println("\"stringRef\" hat Laenge "
         + stringRef.length());
```

Zeichenketten

Die Klasse `String`

Beispiel: `StringDemo1` (Fortsetzung)

```
// Vergleiche
// ----------
   System.out.print("\"" + stringRef
      + "\" kommt lexikographisch ");
   if (stringRef.compareTo(s1) < 0)
      System.out.print("vor");
   else
      System.out.print("nach");
   System.out.println(" \"" + s1 + "\"");

   test = s2.equals("ZYX");    // false
   // s2=="zyx" ungleich "ZYX"
   System.out.println(test);
   test = s2.equalsIgnoreCase("ZYX");
   // Groß-/Kleinschreibung wird ignoriert,
   // also jetzt sind dann beide gleich
   System.out.println(test);   // true
   test = stringRef == s2;
   // liefert true, da beide Variablen auf
   // dasselbe Objekt zeigen
   System.out.println(test);   // true
   stringRef = "zyx";  // ein NEUES String-Objekt
   test = stringRef == s2;
   // liefert jetzt false,
   // da verschiedene Referenzen
   System.out.println(test);   // false
```

Zeichenketten

Die Klasse `String`

Beispiel: `StringDemo1` (Fortsetzung)

```
      // Konkatenation, startsWith
      // -------------------------
         stringRef = stringRef + s1;
         // Es wird ein NEUES String-Objekt erzeugt und
         // der Referenzvariablen stringRef zugewiesen.
         // Es wird nicht das String-Objekt verändert!!
         System.out.println("stringRef = " + stringRef);
         test = stringRef.startsWith("zyx");   // true
         System.out.println(test);
      }
   }
```

Ausgabe des Programms:

```
leererString: ""
s1    : "Das ist ein String"
s2    : "zyx"
"stringRef" hat Laenge 3
"zyx" kommt lexikographisch nach "Das ist ein String"
false
true
true
false
stringRef = zyxDas ist ein String
true
```

Zeichenketten

Die Klasse `String`

Zugriffsmethoden der Klasse `String`

```
public int length()
```

- Liefert die Länge der Zeichenkette.

```
public char charAt(int index)
```

- Liefert das Zeichen an der Position `index` zurück. Das vorderste Zeichen hat die Position 0.

```
public void getChars(int srcBegin, int srcEnd,
   char[] dst, int dstBegin)
```

- Kopiert die Zeichen des aktuellen Objekts von `srcBegin` bis einschließlich `srcEnd-1` in das `char`-Feld `dst`, beginnend ab der Position `dstBegin`.

```
public int indexOf(int ch)
public int indexOf(int ch, int fromIndex)
public int indexOf(String str)
public int indexOf(String str, int fromIndex)
```

- Liefern den (Anfangs-)Index des *ersten* Auftretens des Buchstabens `ch` bzw. des Strings `str`, beginnend am Anfang bzw. ab Position `fromIndex`.
- Liefern -1, falls nicht gefunden.

Zeichenketten

Die Klasse `String`

Zugriffsmethoden der Klasse `String`

```
public int lastIndexOf(int ch)
public int lastIndexOf(int ch, int fromIndex)
public int lastIndexOf(String str)
public int lastIndexOf(String str,
   int fromIndex)
```

- Liefern den (Anfangs-)Index des *letzten* Auftretens des Buchstabens `ch` bzw. des Strings `str`, beginnend am Anfang bzw. ab Position `fromIndex`.
- Liefern -1, falls nicht gefunden.

```
public String substring(int beginIndex)
public String substring(int beginIndex,
   int endIndex)
```

- Liefern den Teilstring des aktuellen Objekts (`this`) ab der Position `beginIndex` bis zum Ende bzw. bis einschließlich der Position `endIndex-1`.

Zeichenketten

Die Klasse `String`

Beispiel: `Reverse`

```
// Programm, das die Kommandozeilen-Argumente
// in umgekehrter Reihenfolge und die Zeichen
// jedes Arguments wiederum rueckwaerts ausgibt.

public class Reverse
{
  public static void main(String[] args)
  {
    // rueckwaerts das Argument-Array durchlaufen
    for(int i = args.length - 1; i >= 0; i--)
    {
      // rueckwaerts alle Zeichen jedes Arguments
      // durchlaufen
      for(int j = args[i].length()-1; j >= 0; j--)
      {
        // Zeichen j von Argument i ausgeben
        System.out.print(args[i].charAt(j));
      }
      // nach jedem Argument ein Leerzeichen ausgeben
      System.out.print(" ");
    }
    // und am Ende die Zeile abschliessen
    System.out.println();
  }
}
```

Zeichenketten

Die Klasse `String`

Beispiel: `StringDemo2`

```
class StringDemo2 {
   public static void main(String[] arg) {
      String ooSprachen = "C++, Ada, " + "Java";
      String bearbeiter = new String("Meyer");

      String aktuell = ooSprachen.substring(10, 14);
      // TeilString von ooSprachen von 10 inklusive
      // bis 14 exklusive, also "Java"
      char c = aktuell.charAt(2);         // 'v'
      // Buchstabe an Stelle 2 von Java (gezählt ab 0)
      System.out.println("char c = aktuell.charAt(2);"
         + " // 'v'  : " + c);

      int pos = ooSprachen.indexOf("Java");   // 10
      System.out.println("pos = ooSprachen.indexOf"
         + "(\"Java\"): " + pos);

      pos = ooSprachen.indexOf(',', 5);        // 8
      // Index des erste Kommas ab Index 5
      System.out.println("int pos = ooSprachen.indexOf"
         + "(',') ab 5: " + pos);

      boolean b = ooSprachen.regionMatches(true, 10,
                     "Die JAVA-Sprache", 4, 4);
      System.out.println("b = " + b);
   }
}
```

Zeichenketten

Die Klasse `String`

Methoden zur Manipulation von `String`-Objekten

`public String replace(char oldChar, char newChar)`

- Liefert eine neue Zeichenkette, in der alle Auftreten von `oldChar` in der aktuellen Zeichenkette (`this`) durch `newChar` ersetzt sind.

`public String toLowerCase()`
`public String toUpperCase()`

- Liefert eine Zeichenkette, in der alle Buchstaben im aktuellen String in Klein- bzw. Großbuchstaben gewandelt sind.

Beispiel: Systemeigenschaften

```
public class SystemEigenschaften
{
  static String prop =
    System.getProperties().toString();

  public static void main(String[] args)
  {
    String Eigenschaften;
    Eigenschaften = prop.replace(',' , '\n');
    Eigenschaften = Eigenschaften.replace('{' , ' ');
    Eigenschaften = Eigenschaften.replace('}' , ' ');
    System.out.println(Eigenschaften);
  }
}
```

Zeichenketten

Beispiel: `SystemEigenschaften` (Fortsetzung)

Das Programm liefert etwa folgende Ausgabe:

```
user.language=de
java.home=C:\PROGRA~1\JDK11~1.5\BIN\..
java.vendor.url.bug=
   http://java.sun.com/cgi-bin/bugreport.cgi
awt.toolkit=sun.awt.windows.WToolkit
file.encoding.pkg=sun.io
java.version=1.1.5
file.separator=\
line.separator=

user.region=DE
file.encoding=8859_1
java.vendor=Sun Microsystems Inc.
user.timezone=GMT
user.name=kuenkler
os.arch=x86
os.name=Windows NT
java.vendor.url=http://www.sun.com/
user.dir=V:\kuenkler\Java\src\bsp\String
java.class.path=.;..;..\..;..\..\..;
java.class.version=45.3
os.version=4.0
path.separator=;
user.home=C:/
```

Zeichenketten

Die Klasse `String`

weitere Methoden der Klasse `String`

`public char[] toCharArray()`

- Wandelt die aktuelle Zeichenkette in ein neu erzeugtes `char`-Feld und liefert dieses zurück.

`public static String valueOf(`*<Typ>* `p)`

- Für alle einfachen Datentypen, `char`-Felder und für die Klasse `Object` gibt es eine Klassenmethode `valueOf()`, die den angegebenen Parameter in eine Zeichenkette wandelt.

`public String trim()`

- Entfernt alle "white space"-Zeichen (Zeichen kleiner oder gleich `'\u0020'`) vorn und hinten vom aktuellen String und liefert diesen zurück.

`public String concat(String str)`

- Hängt das Argument `str` an den aktuellen String an.

Zeichenketten

Die Klasse `StringBuffer`

Objekte der Klasse `StringBuffer` werden für Zeichenketten benutzt, von denen angenommen wird, dass sie wachsen, kleiner werden, oder ihre Buchstaben ändern.

Der Java-Compiler verwendet Objekte dieser Klasse zur Realisierung der Verkettungsoperatoren + und +=.

Konstruktoren der Klasse `StringBuffer`

```
public StringBuffer()
```

- Liefert ein leeres `StringBuffer`-Objekt mit der Anfangskapazität 16.

```
public StringBuffer(int length)
```

- Liefert ein leeres `StringBuffer`-Objekt mit der Anfangskapazität `length`.

```
public StringBuffer(String str)
```

- Liefert ein `StringBuffer`-Objekt mit dem Inhalt von `str` und einer Anfangskapazität, die um 16 größer ist als die Länge von `str`.

Zeichenketten

Die Klasse `StringBuffer`

Methoden für Zugriffe und Manipulationen

`public int length()`

- Liefert die Länge der Zeichenkette.

`public char charAt(int index)`

- Liefert das Zeichen an der Position `index` zurück. Das vorderste Zeichen hat die Position 0.

```
public void getChars(int srcBegin, int srcEnd,
   char[] dst, int dstBegin)
```

- Kopiert die Zeichen des aktuellen Objekts von `srcBegin` bis einschließlich `srcEnd-1` in das `char`-Feld `dst`, beginnend ab der Position `dstBegin`.

Zeichenketten

Die Klasse `StringBuffer`

Methoden für Zugriffe und Manipulationen

`public StringBuffer append(<`*Typ*`> p)`

- Hängt die String-Darstellung des Parameters `p` an das Objekt an und gibt den so erweiterten `StringBuffer` zurück.
  <*Typ*> kann ein einfacher Datentyp, ein `char`-Feld, `String` oder `Object` sein.

`public StringBuffer insert(int offset, <`*Typ*`> p)`

- Fügt die String-Darstellung des Parameters `p` ab der Position `offset` ein und gibt den so erweiterten `StringBuffer` zurück.
  <*Typ*> kann ein einfacher Datentyp, ein `char`-Feld, `String` oder `Object` sein.

`public StringBuffer reverse()`

- Liefert einen `StringBuffer`, in dem die Buchstaben in umgekehrter Reihenfolge stehen.

`public void setCharAt(int index, char ch)`

- Ersetzt den Buchstaben an Position `index` durch `ch`.

`public void deleteCharAt(int index)`

- Löscht den Buchstaben an Position `index`.

Zeichenketten

Die Klasse `StringBuffer`

weitere Methoden

`public int capacity()`

- Liefert die momentane Kapazität des `StringBuffer`-Objekts. Die Kapazität ist die Anzahl der Zeichen, die ohne Speicherplatzanforderung eingefügt werden kann.

`public void ensureCapacity(int minimumCapacity)`

- Setzt die Kapazität des Objekts auf den größeren Wert von
  - `minimumCapacity` und
  - 2 * (alte Kapazität) + 2.

`public void setLength(int newLength)`

- Falls die aktuelle Länge größer als `newLength` ist, wird der Inhalt auf diese Länge abgeschnitten. Andernfalls wird die Länge auf `newLength` gesetzt und es werden gegebenenfalls Null-Zeichen (`'\u0000'`) angehängt.

`public String toString()`

- Liefert ein neues `String`-Objekt, das mit dem Inhalt des aktuellen `StringBuffer`-Objekts initialisiert ist.

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: Realisierung des Verkettungsoperators +

Die Anweisung

```
String s = "a" + 4 + "c";
```

wird vom Compiler übersetzt in

```
String s =
   new StringBuffer().append("a")
       .append(4).append("c").toString();
```

Alternativ könnte der Compiler auch den folgenden Code erzeugen

```
String s =
   new String("a").concat(String.valueOf(4))
       .concat("c");
```

der allerdings weniger effizient ist.

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `DreiChinesen`

```
import utilities.TastaturEingabe;

public class DreiChinesen
{
   private String lied;

   DreiChinesen()
   {
     lied = "Drei Chinesen mit dem Kontrabass\n" +
            "sassen auf der Strasse und " +
            "erzaehlten sich was.\n" +
            "Da kam die Polizei: \"" +
            "Ei was ist denn das?\"\n" +
            "Drei Chinesen mit dem Kontrabass.\n";
   }

   DreiChinesen(String s)
   {  lied = s; }

   /**
    * Bestimmt, ob eingegebenes Zeichen Vokal ist.
    * @param c Zeichen
    * @return  ist zeichen ein Vokal?
    */
   private boolean istVokal(char c)
   {
      c = Character.toLowerCase(c);
      // c ist jetzt Kleinbuchstabe
      return (c=='a') || (c=='e') || (c=='i') ||
             (c=='o') || (c=='u');
   }
```

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `DreiChinesen` (Fortsetzung)

```
/*
 * So lange einlesen, bis Vokal eingegeben wurde.
 * @return eingelesener Vokal als Kleinbuchstabe!
 */
public char vokalEinlesen()
{
   char c;
   do
   {
      c = TastaturEingabe.readChar(
             "Geben Sie einen Vokal ein: ");
   } while (!istVokal(c));

   c = Character.toLowerCase(c);
   // c ist jetzt Kleinbuchstabe

   return c;
}
```

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `DreiChinesen` (Fortsetzung)

```
/**
 * ersetzt Vokale im Text durch angegebenen Vokal
 * @param vokal Ersatzvokal
 */
public String singe(char vokal)
{
   // Hilfsvariable liedHilf vom Typ StringBuffer
   // anlegen und mit lied initialisieren
   StringBuffer liedHilf = new StringBuffer(lied);

   // alle Buchstaben von liedHilf betrachten
   for(int i = 0; i < liedHilf.length(); i++)
   {
      char c = liedHilf.charAt(i);

      // falls aktueller Buchstabe Vokal ist,
      // diesen durch den eingelesenen Vokal
      // ersetzen
      if (istVokal(c))
         // Grossbuchstaben beruecksichtigen
         if (Character.isUpperCase(c))
            liedHilf.setCharAt(
               i, Character.toUpperCase(vokal));
         else
            liedHilf.setCharAt(i, vokal);
   }

   return liedHilf.toString();
}
```

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `DreiChinesen` (Fortsetzung)

```
    public static void main(String[] argv)
    {
       DreiChinesen lied = new DreiChinesen();
       char vokal;
       System.out.println("Umwandlung des Liedes:");
       System.out.println(lied.lied);
       System.out.println();
       vokal = lied.vokalEinlesen();
       System.out.println(
          "Das geaenderte Lied singen:");
       System.out.println();
       System.out.println(lied.singe(vokal));

       String text = TastaturEingabe.readString(
                        "Geben Sie ein Lied ein: ");
       DreiChinesen eigen = new DreiChinesen(text);
       System.out.println();
       vokal = eigen.vokalEinlesen();
       System.out.println(
          "Das geaenderte Lied singen:");
       System.out.println();
       System.out.println(eigen.singe(vokal));
    }
}
```

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `Palindrom`

```
import utilities.TastaturEingabe;

public class Palindrom
{
   String text;
   String palindrom;

   public Palindrom()
   {  text = "ein neger mit gazel";  }

   public Palindrom(String txt)
   {  text = txt;  }

   void zeigePalindrom()
   {  System.out.println(palindrom);  }
```

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `Palindrom` (Fortsetzung)

```
void bildePalindrom()
{
   StringBuffer spiegel = new StringBuffer(text);
   spiegel = spiegel.reverse();

   StringBuffer pal = new StringBuffer();
   pal.ensureCapacity(2*text.length()+3);
   // Kapazität von pal festlegen
   // + 3 wegen der Trennzeichen
   // Alternativ könnte man Konstruktor verwenden:
   // ...pal = new StringBuffer(2*text.length()+3);

   // Texte anhängen
   pal.append(text);
   pal.append(spiegel);
   // Buchstaben einfügen an angegebener Stelle
   pal.insert(text.length(), '|');
   pal.insert(0, '|');
   pal.append('|');

   // Wandlung nach String
   palindrom = pal.toString();
}
```

Zeichenketten

Die Klasse `StringBuffer`

Beispiel: `Palindrom` (Fortsetzung)

```
   public static void main(String[] argv)
   {
      Palindrom p = new Palindrom();
      p.bildePalindrom();
      p.zeigePalindrom();

      String text = TastaturEingabe.readString(
                        "Geben Sie einen Text ein: ");
      Palindrom p2 = new Palindrom(text);
      p2.bildePalindrom();
      p2.zeigePalindrom();
   }
}
```

Das Programm liefert etwa den folgenden Dialog:

```
|ein neger mit gazel|lezag tim regen nie|
Geben Sie einen Text ein: SunJava
|SunJava|avaJnuS|
```

Felder

In Java kann man eine *Folge gleichartiger Objekte* unter einem Namen zusammenfassen. Auf die einzelnen Elemente dieser Folge kann man über Indizes zugreifen.

Die Anzahl der Elemente eines Feldes wird bei seiner Erzeugung festgelegt; danach kann man die Anzahl der Feldelemente nicht mehr verändern.

eindimensionale Felder

Felder (engl. *arrays*) sind - wie Klassen und Schnittstellen - Referenztypen, d. h. eine Feldvariable enthält eine Referenz auf ein Feld.

Die Deklarationen

```
int[] feld;
Punkt[] bildpunkte;
String[] texte;
```

vereinbaren drei Feldvariablen, die Referenzen auf Felder aufnehmen können.

Felder

eindimensionale Felder

Die Erzeugung eines Feldobjekts erfolgt mit dem Operator new:

```
// Feld mit 5 int-Werten
feld = new int[5];
// Feld mit 100 Punkten
bildpunkte = new Punkt[100];
// Feld mit 5 Strings
texte = new String[5];
```

Der Ausdruck in den eckigen Klammern muss einen ganzzahligen Wert liefern.

Die Elemente der Feldobjekte werden implizit alle mit dem Wert 0 bzw. einem zu 0 äquivalenten Wert initialisiert.

Felder können direkt bei ihrer Deklaration durch Angabe der Werte in geschweiften Klammern initialisiert werden.

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

Felder

eindimensionale Felder

Die Elemente eines Feldes werden beginnend mit 0 durchnumeriert.

Jedes Feldobjekt kennt seine Länge; diese kann über das konstante Attribut `length` abgefragt werden.

Auf einzelne Feldelemente wird wie folgt zugegriffen:

```
feld[0] = 5;
System.out.println(bildpunkte[63]);
texte[3] = "Otto";
```

Bei der Deklaration einer Feldvariablen können wir die eckigen Klammern - wie in C und C++ - auch hinter den Variablennamen schreiben. Die folgenden beiden Deklarationen sind äquivalent:

```
int[] feld;
int feld[];
```

ebenso die folgenden:

```
byte[] f(int n);
byte f(int n)[];
```

Felder

eindimensionale Felder

Beispiel:
eindimensionale Felder mit und ohne Initialisierung

```
import utilities.TastaturEingabe;

public class EindimFelder
{
   private static void eindimAusgeben(Object[] o)
   {
      for(int i=0; i<o.length; i++)
         System.out.print(o[i] + ", ");
      System.out.println();
   }

   public static void main(String[] argv)
   {
      int[] feld = new int[5];
      char[] text = {'J','a','v','a'};
      String[] texte = {"Java", "C++",
                        "Pascal", "Assembler"};

      // Feldelemente einlesen
      System.out.println(
         "Geben Sie die Feldelemente ein : ");
      for(int i=0; i<feld.length; i++)
         feld[i] = TastaturEingabe.readInt(i + ": ");
```

Felder

eindimensionale Felder

Beispiel: eindimensionale Felder (Fortsetzung)

```
        // int-Feld ausgeben
        System.out.print("Feld: ");
        for(int i=0; i<feld.length; i++)
           System.out.print(i + ":" + feld[i] + ",");
        System.out.println();

        // char-Feld ausgeben
        for(int i=0; i<text.length; i++)
           System.out.print(text[i] + ", ");
        System.out.println();

        // Texte ausgeben
        eindimAusgeben(texte);
     }
  }
```

Felder

eindimensionale Felder

Beispiel: Kommandozeilen-Argumente

```
public class Echo
{
  public static void main(String [] args)
  {
    for(int i = 0; i <= args.length; i++)
      System.out.print(args[i] + " ");
    System.out.print("\n");
  }
}
```

Der Aufruf

```
java Echo Das Echo ist doof.
```

liefert die folgende Ausgabe:

```
Das Echo ist doof.
java.lang.ArrayIndexOutOfBoundsException: 4
        at Echo.main(Echo.java:6)
```

Jeder Feldzugriff wird zur Laufzeit überprüft. Liegt der Index außerhalb des zulässigen Bereichs, wird eine Ausnahme ausgelöst.

Felder

mehrdimensionale Felder

sind in Java als Felder von Feldern implementiert.

Referenzvariablen für mehrdimensionale Felder deklariert man durch Angabe mehrerer Paare eckiger Klammern. Die Anzahl der Klammerpaare gibt die Dimension des Feldes an.

Bei der Felderzeugung muss mindestens die erste Dimension festgelegt werden; die Längen der anderen Dimensionen können später festgelegt werden.

Auch mehrdimensionale Felder können initialisiert werden.

Felder

mehrdimensionale Felder

Beispiel: Matrizen

```
import utilities.TastaturEingabe;

public class Matrix
{
   private static void matrixAusgeben
      (String titel, int[][]m) {
      System.out.println(titel);
      for(int i=0; i<m.length; i++) {
         for(int j=0; j<m[i].length; j++)
            System.out.print(m[i][j] + ",");
         System.out.println();
      }
   }

   public static void main(String[] argv)
   {
      int[][] matrix1 = new int[3][3],
              matrix2 = {{ 0, 1, 2, 3, 4},
                         {10,11,12,13,14},
                         {20,21,22,23,24}
                        };

      for(int i=0; i<matrix1.length; i++)
         for(int j=0; j<matrix1[i].length; j++)
            matrix1[i][j] =
               TastaturEingabe.readInt(
                  "matrix1[" + i + "][" + j + "]: ");

      matrixAusgeben("Matrix1", matrix1);
      matrixAusgeben("Matrix2", matrix2);
   }
}
```

Felder

mehrdimensionale Felder

Beispiel: nicht-rechteckige Felder

```
import utilities.*;

public class Dreiecksmatrix
{
   static int [][] dreieck;

   static private void matAusgeben
      (String titel, int[][]m)
   {
      System.out.println(titel);
      for(int i=0; i<m.length; i++)
      {
         for(int j=0; j<m[i].length; j++)
           FormatierteAusgabe.ausgabeInt(m[i][j], 5);
         System.out.println();
      }
   }

   public static void main(String[] argv)
   {
      int [][] zweidimFeld = {{ 0, 1, 2, 3},
                              {10,11,12},
                              {20,21},
                              {31,32,33,34,35}
                             };
      int laenge = TastaturEingabe.readInt(
                      "Wie gross soll die " +
                      "Dreiecksmatrix sein? ");
```

Felder

mehrdimensionale Felder

Beispiel: nicht-rechteckige Felder (Fortsetzung)

```
        // dreieck instanziieren
        dreieck = new int[laenge][];

        // Jetzt werden die einzelnen Zeilen
        // instanziiert; mit abnehmender Laenge
        for(int i=0; i<dreieck.length; i++)
           dreieck[i] = new int[laenge-i];

        // Belegung der Feldelemente mit Zufallszahlen
        for(int i=0; i<dreieck.length; i++)
           for(int j=0; j<dreieck[i].length; j++)
           {
              dreieck[i][j] = (int)(1000*Math.random());
              if (Math.random() >= 0.5)
                 dreieck[i][j] *= -1;
           }

        // Ausgabe der 2-dim Matrix
        matAusgeben("2-dim. Feld", zweidimFeld);

        // Ausgabe der Dreiecksmatrix
        matAusgeben("dreieck", dreieck);
     }
  }
```

Felder

einige Beispiele

sequenzielle Suche

```
// Durchsuchen eines Feldes nach einem "Schlüssel"
// der Index des ersten Auftretens des Schlüssels
// wird zurückgeliefert; suchen() liefert -1, falls
// der Schlüssel nicht im Feld enthalten ist

import utilities.TastaturEingabe;

class SeqSearch
{
   // Durchsuchen des Bereichs start .. end nach key
   public static int suchen
      (int[] a, int start, int end, int key)
   {
      for(int i = start; i <= end; i++)
         if(a[i] == key)
            return i;
      return -1;
   }

   // Durchsuchen des gesamten Feldes
   public static int suchen(int[] a, int key)
   {
      return suchen(a, 0, a.length-1, key);
   }
```

Felder

einige Beispiele

sequenzielle Suche (Fortsetzung)

```
      public static void main(String[] args)
      {
         int key, firstIndex;
         int[] intFeld = new int[10];

         // intFeld mit Pseudo-Zufallszahlen
         // im Bereich 0 .. 100 initialisieren
         for(int i = 0; i < intFeld.length; i++)
            intFeld[i] = (int)(100*Math.random());

         System.out.print("Feld: ");
         for(int i = 0; i < intFeld.length; i++)
            System.out.print(intFeld[i] + " ");
         System.out.println();

         key = TastaturEingabe.readInt(
                  "Bitte Schluessel eingeben: ");

         firstIndex = suchen(intFeld, key);

         System.out.print("Der Schluessel " + key);
         if(firstIndex == -1)
            System.out.println(" wurde nicht gefunden.");
         else
            System.out.println(" tritt an Position "
               + firstIndex + " erstmals auf.");
      }
   }
```

Felder

einige Beispiele

Duplikate löschen

```
class RemoveDuplicates
{
   // Durchsuchen des Bereichs start .. end nach key
   public static int suchen
      (int[] a, int start, int end, int key)
   {
      for(int i = start; i <= end; i++)
         if(a[i] == key)
            return i;
      return -1;
   }

   // Loeschen des Elements mit dem Index pos;
   // alle nachfolgenden Elemente muessen um eine
   // Position nach links verschoben werden.

   public static int loeschen
      (int[] a, int size, int pos)
   {
      // nur wenn pos == size-1 ist,
      // muessen wir nichts verschieben
      for(int i = pos; i < size-1; i++)
         a[i] = a[i+1];
      return size-1;
   }
```

Felder

einige Beispiele

Duplikate löschen (Fortsetzung)

```
public static int duplikateEntfernen
  (int[] a, int size)
{
  int i = 0, key, index;

  // wenn i == size geworden ist, sind wir fertig
  while(i < size)
  {
     // Feldelement mit Index i wird Suchschluessel
     key = a[i];

     // suche und entferne alle Duplikate des
     // Schluessels beginnend beim Index i+1
     // wir sind fertig, wenn suchen() -1
     // zurueckliefert
     while((index = suchen(a, i+1, size-1, key)) != -1)
        size = loeschen(a, size, index);

     i++;
  }

  return size;
}
```

Felder

einige Beispiele

Duplikate löschen (Fortsetzung)

```
      public static void main(String[] args)
      {
         int size;
         int[] intFeld = new int[10];

         // intFeld mit Pseudo-Zufallszahlen
         // im Bereich 0 .. 10 initialisieren
         for(int i = 0; i < intFeld.length; i++)
            intFeld[i] = (int)(10*Math.random());

         System.out.print("Originalzustand:   ");
         for(int i = 0; i < intFeld.length; i++)
            System.out.print(intFeld[i] + " ");
         System.out.println();

         size = duplikateEntfernen
                   (intFeld, intFeld.length);

         System.out.print("nach dem Loeschen: ");
         for(int i = 0; i < size; i++)
            System.out.print(intFeld[i] + " ");
         System.out.println();
      }
   }
```

Felder

einige Beispiele

Bubble Sort - Sortieren durch Austauschen

```
class BubbleSort
{
   public static void bubble(int[] a)
   {
      for(int i = a.length-1; i > 0; i--)
         for(int j = 0; j < i; j++)
            if(a[j] > a[j+1])
            {
               int tmp = a[j];
               a[j] = a[j+1];
               a[j+1] = tmp;
            }
   }
```

Felder

einige Beispiele

Bubble Sort (Fortsetzung)

```
      public static void main(String[] args)
      {
         int[] intFeld = {8, 64, 13, 9, 2,
                          27, 3, 123, 17, -1};

         System.out.print("vor dem Sortieren:  ");
         for(int i=0; i<intFeld.length; i++)
            System.out.print(intFeld[i] + " ");
         System.out.println();

         bubble(intFeld);

         System.out.print("nach dem Sortieren: ");
         for(int i=0; i<intFeld.length; i++)
            System.out.print(intFeld[i] + " ");
         System.out.println();
      }
   }
```

Ausgabe des Programms:

```
vor dem Sortieren:  8 64 13 9 2 27 3 123 17 -1
nach dem Sortieren: -1 2 3 8 9 13 17 27 64 123
```

Einrücken!!!

```
class BubbleSort
{
   public static void bubble(int[] a)
   {
      for(int i = a.length-1; i > 0; i--)
         for(int j = 0; j < i; j++)
            if(a[j] > a[j+1])
            {
               int tmp = a[j];
               a[j] = a[j+1];
               a[j+1] = tmp;
            }
   }

   public static void main(String[] args)
   {
      int[] intFeld = {8, 64, 13, 9, 2,
                       27, 3, 123, 17, -1};

      System.out.print("vor dem Sortieren:  ");
      for(int i=0; i<intFeld.length; i++)
         System.out.print(intFeld[i] + " ");
      System.out.println();

      bubble(intFeld);

      System.out.print("nach dem Sortieren: ");
      for(int i=0; i<intFeld.length; i++)
         System.out.print(intFeld[i] + " ");
      System.out.println();
   }
}
```

Einrücken??? Wozu denn das???

```
class BubbleSort
{
public static void bubble(int[] a)
{
for(int i = a.length-1; i > 0; i--)
for(int j = 0; j < i; j++)
if(a[j] > a[j+1])
{
int tmp = a[j];
a[j] = a[j+1];
a[j+1] = tmp;
}
}

public static void main(String[] args)
{
int[] intFeld = {8, 64, 13, 9, 2,
27, 3, 123, 17, -1};

System.out.print("vor dem Sortieren:  ");
for(int i=0; i<intFeld.length; i++)
System.out.print(intFeld[i] + " ");
System.out.println();

bubble(intFeld);

System.out.print("nach dem Sortieren: ");
for(int i=0; i<intFeld.length; i++)
System.out.print(intFeld[i] + " ");
System.out.println();
}
}
```

Einrücken??? So ein Quatsch!!!

```
class BubbleSort
{
public static void bubble(int[] a)
{
for(int i = a.length-1; i > 0; i--)
for(int j = 0; j < i; j++)
if(a[j] > a[j+1])
{
int tmp = a[j];
a[j] = a[j+1];
a[j+1] = tmp;
}
}

public static void main(String[] args)
{
int[] intFeld = {8, 64, 13, 9, 2,
27, 3, 123, 17, -1};

System.out.print("vor dem Sortieren:  ");
for(int i=0; i<intFeld.length; i++)
System.out.print(intFeld[i] + " ");
System.out.println();

bubble(intFeld);

System.out.print("nach dem Sortieren: ");
for(int i=0; i<intFeld.length; i++)
System.out.print(intFeld[i] + " ");
System.out.println();
}
}
```

Einrücken??? Alles Unfug!!!

```
                                class BubbleSort
                                                {
                        public static void bubble(int[] a)
                                                {
                      for(int i = a.length-1; i > 0; i--)
                                for(int j = 0; j < i; j++)
                                        if(a[j] > a[j+1])
                                                {
                                          int tmp = a[j];
                                           a[j] = a[j+1];
                                            a[j+1] = tmp;
                                                }
                                                }

                   public static void main(String[] args)
                                                {
                         int[] intFeld = {8, 64, 13, 9, 2,
                                       27, 3, 123, 17, -1};

               System.out.print("vor dem Sortieren:  ");
                     for(int i=0; i<intFeld.length; i++)
                     System.out.print(intFeld[i] + " ");
                                    System.out.println();

                                        bubble(intFeld);

               System.out.print("nach dem Sortieren: ");
                     for(int i=0; i<intFeld.length; i++)
                     System.out.print(intFeld[i] + " ");
                                    System.out.println();
                                                }
                                                }
```

Referenztypen

einfache Datentypen:

- `boolean`
- `char`
- `byte`
- `short`
- `int`
- `long`
- `float`
- `double`

Referenztypen:

- Felder
- Klassen
- Schnittstellen

Eine Variable eines primitiven Datentyps enthält einen Wert dieses Typs.

Eine Variable eines Referenztyps enthält eine Referenz auf auf das eigentliche Objekt (den eigentlichen Wert).

- Konsequenz: Die Zuweisung (=) kopiert die *Referenz* (und nicht das Objekt!), die Vergleichsoperationen (`==` und `!=`) vergleichen *Referenzen* (und keine Objekte!).

- Referenz auf nichts: `null`

Referenztypen

Wertzuweisung und Vergleich bei einfachen Datentypen

Beispiel: Variablen vom Typ `int`

```
int i = 123;
int j = i;
System.out.println(i == j);
```

Wertzuweisung bei einfachen Datentypen:
Kopie des **Werts**

Vergleich bei einfachen Datentypen:
Vergleich der **Werte**

Referenztypen

Wertzuweisung und Vergleich bei Referenztypen

Beispiel: Objekte der Klasse `Kreis`

```
Kreis k1 = new Kreis(10.0);
Kreis k2 = new Kreis(10.0);
Kreis k3 = k1;
System.out.println(k1 == k2);
System.out.println(k1 == k3);
```

Wertzuweisung bei Referenztypen:
Kopie der **Referenz**

Vergleich bei Referenztypen :
Vergleich der **Referenzen**

Referenztypen

Wertzuweisung und Vergleich bei Referenztypen

Beispiel: Objekte der Klasse `String`

```
String s1 = "gleich";
String s2 = "ch";
String s3 = new String("gleich");
String s4 = new String("gleich");
String s5 = s1;
String s6 = "gleich";

System.out.println(s1 == "gleich");
System.out.println(s1 == "glei" + s2);
System.out.println(s1 == s3);
System.out.println(s3 == s4);
System.out.println(s1 == s5);
System.out.println(s1 == s6);
System.out.println(s1.equals("glei" + s2));
System.out.println(s3.equals(s4));
```

Referenztypen

Wertzuweisung und Vergleich bei Referenztypen

Beispiel: Felder von einfachen Typen (hier Typ `int`)

```
int [] feld1 = {1, 2, 3};
int [] feld2 = {1, 2, 3};
int [] feld3 = feld1;

System.out.println(feld1 == feld2);
System.out.println(feld1 == feld3);

feld3[1] = 4812;
for (int i = 0; i < feld1.length; i++)
   System.out.print(feld1[i] + "   ");
System.out.println();
```

Referenztypen

Wertzuweisung bei Referenztypen

Beispiel: Kopieren von Feldern:

1. Sogenannte „flache“ Kopie: Die *Referenz* wird kopiert.

```
int [] original = {1, 2, 3};
int [] kopie = original;

for (int i = 0; i < kopie.length; i++)
   kopie[i] *= 2;
for (int i = 0; i < original.length; i++)
   System.out.print(original[i] + "   ");
System.out.println();
```

2. Sogenannte „tiefe“ Kopie:

   Mittels new wird ein neues `int`-Feld `kopie` angelegt, das die gleiche Länge wie `original` hat.

   Anschließend werden in einer Schleife die Feldelemente von `original` nach `kopie` kopiert.

```
int [] original = {1, 2, 3};
int [] kopie = new int[original.length];
for (int i = 0; i < kopie.length; i++)
   kopie[i] = original[i];

for (int i = 0; i < kopie.length; i++)
   kopie[i] *= 2;
for (int i = 0; i < original.length; i++)
   System.out.print(original[i] + "   ");
System.out.println();
```

Referenztypen

Wertzuweisung bei Referenztypen

Beispiel: Felder von Objekten (hier Typ `Integer`)

```
class NullPointer
{
   public static void main (String[] args)
   {
      int a = 0;
      Integer[] feld = new Integer[5];
      a = feld[3].intValue();
   }
}
```

Ausgabe des Programms:

```
java.lang.NullPointerException
  at NullPointer.main(NullPointer.java:7)
Exception in thread "main"
```

Problem: Die Deklaration

```
Integer[] feld = new Integer[5];
```

liefert kein Feld von `Integer`-Objekten, sondern lediglich ein Feld von Elementen, die auf `Integer`-Objekte verweisen können.

Referenztypen

Wertzuweisung bei Referenztypen

Beispiel: Felder von Objekten (hier Typ `Integer`)

```
class ReferenzUndKopie
{
   public static void main (String[] args)
   {
      Integer[] beer;
      Integer[] lager;

      beer = new Integer[5];

      beer[3] = new Integer(3);


      // Der Wert der Referenzvariablen beer wird
      // in die Referenzvariable lager kopiert.
      lager = beer;

      if (lager[3].intValue() == 3)
         System.out.println(
            "lager[3] hat jetzt den Wert 3.");


      beer[3] = new Integer(256);

      if (lager[3].intValue() == 256)
         System.out.println(
            "Die Referenz beer wurde kopiert" +
            " -- beide Referenzen beer und lager" +
            " zeigen auf dasselbe Objekt.");
```

Referenztypen

Wertzuweisung bei Referenztypen

Beispiel: Felder von Objekten (Fortsetzung)

```
        // Ein Feld fuer lager erzeugen ...
        lager = new Integer[5];

        // ... und die Feldelemente
        // von beer nach lager kopieren.
        for (int i = 0; i < lager.length; i++)
           lager[i] = beer[i];


        beer[3] = new Integer(1234);

        if (lager[3].intValue() != 1234)
           System.out.println(
              "Eine Kopie des Feldes, auf das beer" +
              " verweist, wurde erzeugt" +
              " -- die Referenzen beer und lager" +
              " zeigen auf unterschiedliche Objekte.");
     }

  }
```

Referenztypen

Wertzuweisung und Vergleich bei Referenztypen

Die Klasse `java.lang.Object` enthält die Methoden

```
public boolean equals(Object obj)
protected Object clone()
   throws CloneNotSupportedException
```

Die Methode **`equals()`** der Klasse `Object` führt einen Referenzvergleich durch:

- Für zwei Referenzvariablen `x` und `y` gilt `x.equals(y)` genau dann, wenn `x` und `y` auf *dasselbe* Objekt verweisen, d.h. wenn `x == y` gilt.

Die Methode **`clone()`** der Klasse `Object` erzeugt ein neues Objekt derselben Klasse wie dieses (`this`) Objekt:

- Die Attribute des neuen Objekts werden mit den Werten der entsprechenden Attribute dieses Objekts initialisiert; ein Konstruktor wird nicht aufgerufen.
- Bei Attributen, die von einem Referenztyp sind, wird *die Referenz* kopiert und nicht für jedes interne Objekt die `clone()`-Operation wiederholt („flache“ Kopie).

Zeichenketten und Felder

Die Klasse `Vector`

Das Modul `java.util` enthält die Klasse `Vector`.
Die Klasse `Vector` implementiert Felder, deren Länge nach Bedarf wachsen und abnehmen kann.

Konstruktoren der Klasse `Vector`

Jeder der drei Konstruktoren erzeugt ein zunächst leeres `Vector`-Objekt. Intern wird bereits Speicherplatz für eine bestimmte Zahl von Elementen bereitgestellt, angegeben durch die Kapazität.

```
public Vector()
```

- Die Anfangskapazität wird auf 10 festgelegt.

```
public Vector(int initialCapacity)
```

- Die Anfangskapazität wird auf `initialCapacity` festgelegt.

```
public Vector(int initialCapacity,
              int capacityIncrement)
```

- Die Anfangskapazität wird auf `initialCapacity` festgelegt, und mit `capacityIncrement` wird festgelegt, um welchen Wert die Kapazität erhöht wird, wenn der Vektor verlängert werden muss.

Die Klasse `Vector`

Methoden für Kapazität und Größe

`public int capacity()`

- Liefert die momentane Kapazität des `Vector`-Objekts. Dies ist mindestens die Zahl der Elemente.

`public void ensureCapacity(int minCapacity)`

- Die Kapazität des `Vector`-Objekts wird vergrößert gemäß dem *Attribut* `capacityIncrement`, aber mindestens auf den Wert `minCapacity` gesetzt.

`public int size()`

- Liefert die aktuelle Zahl der Vektor-Elemente.

`public void setSize(int newSize)`

- Falls die Zahl der Elemente größer als `newSize` ist, wird der Inhalt auf diese Länge abgeschnitten. Andernfalls wird die Zahl der Elemente auf `newSize` gesetzt und es werden gegebenenfalls neue Elemente mit dem Wert `null` am Ende des Vektors angehängt.

`public void trimToSize()`

- Setzt die Kapazität des `Vector`-Objekts auf die aktuelle Zahl der Elemente.

Die Klasse `Vector`

Beispiel: Vektor-Größe und -Kapazität

```
import java.util.Vector;

public class VectorCapacity {
  public static void main(String[] argv) {
    Vector v1 = new Vector(),
           v2 = new Vector(50),
           v3 = new Vector(100,20),
           v4 = new Vector(0);

    System.out.println("Anfangskapazitaet von v1: "
                       + v1.capacity());
    System.out.println("Anfangskapazitaet von v2: "
                       + v2.capacity());
    System.out.println("Anfangskapazitaet von v3: "
                       + v3.capacity());
    System.out.println("Anfangskapazitaet von v4: "
                       + v4.capacity() + "\n");
    v4.ensureCapacity(33);
    System.out.println("nach v4.ensureCapacity(33): "
                       + v4.capacity());
    System.out.println("Groesse von v4: "
                       + v4.size() + "\n");
    v3.trimToSize();
    System.out.println("Kapazitaet von v3 nach trimToSize: "
                       + v3.capacity() + "\n");
    v1.setSize(333);
    System.out.println("Groesse von v1 nach setSize(333): "
                       + v1.size());
    v1.trimToSize();
    System.out.println("Groesse von v1 nach trimToSize: "
                       + v1.size());
  }
}
```

Die Klasse `Vector`

Beispiel: Vektor-Größe und -Kapazität

Ausgabe des Programms:

```
Anfangskapazitaet von v1: 10
Anfangskapazitaet von v2: 50
Anfangskapazitaet von v3: 100
Anfangskapazitaet von v4: 0

nach v4.ensureCapacity(33): 33
Groesse von v4: 0

Kapazitaet von v3 nach trimToSize: 0

Groesse von v1 nach setSize(333): 333
Groesse von v1 nach trimToSize: 333
```

Die Klasse `Vector`

Methoden zur Arbeit mit Vektor-Elementen

```
public void addElement(Object obj)
```

- Fügt das Objekt `obj` am Ende des Vektors an. Die Größe des Vektors erhöht sich dadurch um 1.

```
public void insertElementAt(Object obj,
                            int index)
```

- Fügt das Objekt `obj` an der Position `index` ein; `index` muss = 0 und = der aktuellen Größe des Vektors sein. Die Indizes der Elemente hinter `index` werden um 1 erhöht.

```
public void setElementAt(Object obj,
                         int index)
```

- Überschreibt das Element an der Position `index` mit `obj`.

```
public void removeAllElements()
```

- Entfernt alle Elemente aus dem Vektor; die Vektor-Größe wird 0.

```
public void removeElementAt(int index)
```

- Entfernt das Element an der Position `index`. Die Indizes der Elemente hinter `index` werden um 1 erniedrigt.

Die Klasse `Vector`

Methoden zur Arbeit mit Vektor-Elementen

```
public boolean removeElement(Object obj)
```

- Sucht nach dem *ersten* Vorkommen von `obj`.
  Ist es vorhanden, so wird es aus dem Vektor entfernt, und die Methode gibt `true` zurück; die Indizes der dahinter liegenden Elemente werden um 1 erniedrigt.
  Andernfalls wird `false` zurückgegeben.

```
public Object elementAt(int index)
```

- Liefert das Element an der Position `index`.

```
public int indexOf(Object elem)
public int indexOf(Object elem, int index)
```

- Liefern den Index des *ersten* Vorkommens des Elements `elem` zurück oder aber -1, falls nicht vorhanden.
  Die Suche beginnt beim Index 0 bzw. `index`.

```
public int lastIndexOf(Object elem)
public int lastIndexOf(Object elem, int index)
```

- Liefern den Index des *letzten* Vorkommens des Elements `elem` zurück oder aber -1, falls nicht vorhanden.
  Die Suche beginnt am Ende des Vektors bzw. beim Index `index`.

Die Klasse `Vector`

Methoden zur Arbeit mit Vektor-Elementen

```
public Object firstElement()
public Object lastElement()
```

- Liefern das erste bzw. letzte Element des Vektors.

```
public boolean contains(Object elem)
```

- Liefert `true`, falls `elem` im Vektor enthalten ist, sonst `false`.

```
public boolean isEmpty()
```

- Liefert `true`, falls der Vektor kein Element enthält, sonst `false`.

Die Klasse `Vector`

Beispiel: Vektor mit Elementen verschiedener Typen

```
import java.util.Vector;

class Konto
{
   private int nr, stand;   // Kontostand in €

   public Konto(int nr, int stand)
   {
      this.nr = nr;
      this.stand = stand;
   }

   public void drucken()
   {
      System.out.println(nr + ", Kontostand: " + stand);
   }
}

class Kunde
{
   private String name, vorname, ort;

   public Kunde(String n, String v, String o)
   {
      name = n;
      vorname = v;
      ort = o;
   }

   public void drucken()
   {
      System.out.println(vorname + " " + name +
                         " wohnt in " + ort);
   }
}
```

Die Klasse `Vector`

Beispiel: Vektor mit Elementen verschiedener Typen

```
import java.util.Vector;

public class Elemente1
{
   public static void main(String[] argv)
   {
      Vector v = new Vector(5, 3);

      // Kunden-Elemente an v anfuegen
      v.addElement(new Kunde("Maier", "Franz", "Ulm"));
      v.addElement(new Kunde("Schmidt", "Fritz", "Essen"));
      v.addElement(new Kunde("Mueller", "Paul", "Berlin"));
      System.out.println("Kapazitaet nach 3 Elementen: "
                         + v.capacity());

      v.addElement(new Kunde("Schmidt", "Otto", "Kiel"));
      v.addElement(new Kunde("Adel", "Suse", "Ulm"));
      v.addElement(new Kunde("Magg", "Eva", "Kiel"));
      System.out.println("Kapazitaet nach 6 Elementen: "
                         + v.capacity());

      v.addElement(new Kunde("Zimmer", "Edith", "Ulm"));

      // Konto-Elemente einfügen
      v.insertElementAt(new Konto(4711, 1234), 0);
      v.insertElementAt(new Konto(79033, 43210), 3);
      System.out.println("Kapazitaet nach 9 Elementen: "
                         + v.capacity() + "\n");

      v.insertElementAt(new Konto(1248, -134209), 7);
```

Die Klasse `Vector`

Beispiel: Vektor mit Elementen verschiedener Typen

```
      System.out.println(
         "Alles ausgeben\n--------------");
       for (int i = 0; i < v.size(); i++)
       {
          Object o = v.elementAt(i);
          System.out.print(i + ": ");
          if (o instanceof Kunde)
             ((Kunde)o).drucken();
          else
             ((Konto)o).drucken();
       }
   }
}
```

Ausgabe des Programms:

```
Kapazitaet nach 3 Elementen: 5
Kapazitaet nach 6 Elementen: 8
Kapazitaet nach 9 Elementen: 11

Alles ausgeben
--------------
0: 4711, Kontostand: 1234
1: Franz Maier wohnt in Ulm
2: Fritz Schmidt wohnt in Essen
3: 79033, Kontostand: 43210
4: Paul Mueller wohnt in Berlin
5: Otto Schmidt wohnt in Kiel
6: Suse Adel wohnt in Ulm
7: 1248, Kontostand: -134209
8: Eva Magg wohnt in Kiel
9: Edith Zimmer wohnt in Ulm
```

Die Klasse `Vector`

Beispiel: Vektor mit Elementen vom Typ `Druckbar`

```
import java.util.Vector;

interface Druckbar
{
   public void drucken();
}

class Konto implements Druckbar
{
   // wie im vorigen Beispiel
}

class Kunde implements Druckbar
{
   // wie im vorigen Beispiel
}

public class Elemente2
{
   public static void main(String[] argv)
   {
      // wie im vorigen Beispiel

      System.out.println(
         "Alles ausgeben\n--------------");
      for (int i = 0; i < v.size(); i++)
      {
         Druckbar d = (Druckbar)v.elementAt(i);
         System.out.print(i + ": ");
         d.drucken();
      }
   }
}
```

Die Klasse `Vector`

Die Suchmethoden wie

- `removeElement()`
- `indexOf()`
- `contains()`

haben alle einen Parameter `elem` vom Typ `Object`.

Zum Auffinden des Elements `elem` in dem Vektor rufen diese Methoden die Methode `equals()` auf, die in der Klasse `Object` wie folgt definiert ist

```
public boolean equals(Object obj)
```

Die Methode `equals()` der Klasse `Object` führt einen Referenzvergleich durch:

- Für zwei Referenzvariablen `x` und `y` gilt `x.equals(y)` genau dann, wenn `x` und `y` auf *dasselbe* Objekt verweisen, d.h. wenn `x == y` gilt.

Die Klasse `Vector`

Beispiel: doppelte Vektor-Einträge löschen (Version 1)

```
import java.util.*;
import utilities.TastaturEingabe;

class Kunde
{
   private String name, vorname, ort;

   public Kunde(String n, String v, String o)
   {
      name = n; vorname = v; ort = o;
   }

   public void drucken()
   {
      System.out.println(
         vorname + " " + name + " wohnt in " + ort);
   }
}

public class DoppelteLoeschen
{
   static void kundenAusgeben(String Titel, Vector v)
   {
      System.out.println(Titel);
      for (int i = 0; i < v.size(); i++)
      {
         Kunde k = (Kunde)v.elementAt(i);
         System.out.print(i + ": ");
         k.drucken();
      }
      System.out.println();
   }
```

Die Klasse `Vector`

Beispiel: doppelte Vektor-Einträge löschen (Version 1)

```
public static void loescheDuplikate(Vector v)
{
   int index;

   for (int i = 0; i < v.size(); i++)
   {
      while (true)
      {
         index = v.indexOf(v.elementAt(i), i+1);
         System.out.println("index= " + index);

         if (index != -1)
            v.removeElementAt(index);
         else
            break; // kein Duplikat von i-tem Element

         kundenAusgeben(
            "geloescht wird Element [" + i + "]", v);

         TastaturEingabe.warte();
      }
   }
}
```

Die Klasse `Vector`

Beispiel: doppelte Vektor-Einträge löschen (Version 1)

```
   public static void main(String[] argv)
   {
      Vector v = new Vector(5, 3);

      // Kunden-Elemente an v anfuegen
      Kunde p1 = new Kunde("Schmidt", "Otto", "Kiel");
      Kunde p2 = new Kunde("Adel", "Suse", "Ulm");
      Kunde p3 = new Kunde("Magg", "Eva", "Kiel");

      v.addElement(p1);
      v.addElement(p2);
      v.addElement(new Kunde("Magg", "Eva", "Kiel"));
      v.addElement(p3);
      v.addElement(new Kunde("Magg", "Eva", "Kiel"));

      // Kunden-Elemente einfuegen
      v.insertElementAt(p2, 1);
      v.insertElementAt(p1, 3);
      v.insertElementAt(p3, 2);
      v.insertElementAt(
         new Kunde("Adel", "Suse", "Ulm"), 4);

      kundenAusgeben("Alles", v);

      // alle doppelten Eintraege loeschen
      loescheDuplikate(v);

      kundenAusgeben("ohne Duplikate", v);
   }
}
```

Die Klasse `Vector`

Beispiel: doppelte Vektor-Einträge löschen (Version 1)

Ausgabe des Programms:

```
Alles
0: Otto Schmidt wohnt in Kiel
1: Suse Adel wohnt in Ulm
2: Eva Magg wohnt in Kiel
3: Suse Adel wohnt in Ulm
4: Suse Adel wohnt in Ulm
5: Otto Schmidt wohnt in Kiel
6: Eva Magg wohnt in Kiel
7: Eva Magg wohnt in Kiel
8: Eva Magg wohnt in Kiel

...

ohne Duplikate
0: Otto Schmidt wohnt in Kiel
1: Suse Adel wohnt in Ulm
2: Eva Magg wohnt in Kiel
3: Suse Adel wohnt in Ulm
4: Eva Magg wohnt in Kiel
5: Eva Magg wohnt in Kiel
```

Die Klasse `Vector`

Beispiel: doppelte Vektor-Einträge löschen (Version 2)

```
class Kunde
{
   private String name, vorname, ort;

   public Kunde(String n, String v, String o)
   {
      name = n; vorname = v; ort = o;
   }

   public void drucken()
   {
      System.out.println(
         vorname + " " + name + " wohnt in " + ort);
   }

   // Die Methode equals() der Klasse Object wird
   // überschrieben: zwei Kunden sind gleich,
   // wenn ihre Attribute gleich sind.
   public boolean equals(Object o)
   {
      if (o instanceof Kunde)
      {
         Kunde kunde = (Kunde)o;

         return    name.equals(kunde.name)
                && vorname.equals(kunde.vorname)
                && ort.equals(kunde.ort);
      }
      else
         return false;
   }
}
```

Die Klasse `Vector`

Beispiel: doppelte Vektor-Einträge löschen (Version 2)

Ausgabe des Programms:

```
Alles
0: Otto Schmidt wohnt in Kiel
1: Suse Adel wohnt in Ulm
2: Eva Magg wohnt in Kiel
3: Suse Adel wohnt in Ulm
4: Suse Adel wohnt in Ulm
5: Otto Schmidt wohnt in Kiel
6: Eva Magg wohnt in Kiel
7: Eva Magg wohnt in Kiel
8: Eva Magg wohnt in Kiel

...

ohne Duplikate
0: Otto Schmidt wohnt in Kiel
1: Suse Adel wohnt in Ulm
2: Eva Magg wohnt in Kiel
```

Die Klasse `Vector`

weitere Methoden der Klasse `Vector`

```
public String toString()
```

- Wandelt den Vektor in eine Zeichenkette.

```
public void copyInto(Object[] anArray)
```

- Kopiert die Elemente des Vektors in das Feld `anArray`.

```
public Object clone()
```

- Liefert eine Kopie des `Vector`-Objekts zurück.

```
public Enumeration elements()
```

- Liefert ein Objekt vom Typ `Enumeration`, welches eine Aufzählung aller Elemente des Vektors darstellt. Die Aufzählung liefert zuerst das Element mit dem Index 0, dann das Element mit dem Index 1 usw..

Die Klasse `Vector`

Die Schnittstelle `Enumeration`

Die Schnittstelle `Enumeration` aus dem Modul `java.util` enthält zwei abstrakte Methoden, mit denen man Aufzählungselemente bearbeiten kann:

```
public Interface Enumeration
{
  public abstract boolean hasMoreElements();
  public abstract Object nextElement();
}
```

- `hasMoreElements()` prüft, ob die Aufzählung noch mindestens ein weiteres Element besitzt.
- `nextElement()` gibt das nächste Element zurück, falls ein solches Element existiert.

Die Klasse `Vector` implementiert die Schnittstelle `Enumeration`.

Die Klasse `Vector`

Beispiel: Ausgabe als Aufzählungselemente

```
import java.util.*;

class Kunde
{
   // wie gehabt
}

public class Aufzaehlung
{
   public static void main(String[] argv)
   {
      Vector v = new Vector(5, 3);
      Kunde p1 = new Kunde("May", "Otto", "Kiel");
      // usw.

      // Enumeration-Objekt anlegen
      // und initialisieren
      Enumeration alles = v.elements();
      int i = 0;

      // Aufzaehlung durchlaufen
      // und jedes Element ausgeben
      while (alles.hasMoreElements())
      {
         Object o = alles.nextElement();

         System.out.print(i + ":");
         ((Kunde)o).drucken();

         i++;
      }
   }
}
```