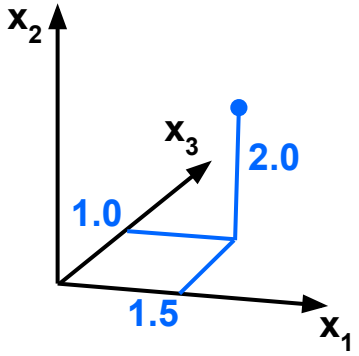
Objektorientierte und Formale Programmierung
-- Java Grundlagen --

5. Arrays und Strings

5.1. Arrays

- In einem Array kann man mehrere Variablen des gleichen Typs zusammenfassen
- Beispiel: Koordinaten eines Punkts im Raum:
  - Mathematisch: $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = (1.5, 2.0, 1.0)$
  - In Java: Array mit Elementen vom Typ double:

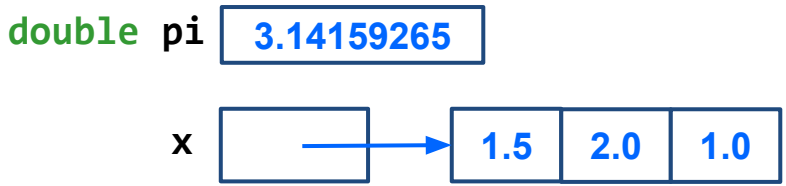
```
double[] x = new double[3];
x[0] = 1.5;
x[1] = 2.0;
x[2] = 1.0;
// oder kürzer:
double[] x = { 1.5, 2.0, 1.0 };
```



5.1. Arrays

- Arrays müssen dynamisch angelegt werden: **`new`**
- Das eigentliche Array wird über eine Referenz angesprochen:

```
double pi = 3.14159265;
double[] x = new double[3];
x[0] = 1.5;
x[1] = 2.0;
x[2] = 1.0;
```



- Ein Element eines Arrays wird über einen ganzzahligen Index ausgewählt
  - Z.B. **`x[1]`** oder **`x[i+j]`**
  - Das erste Element hat den Index **`0`**
  - Ausnahme bei Indexüberlauf: IndexOutOfBoundsException
- Arrays besitzen ein ”Attribut“ **`length`**: Anzahl der Elemente `x.length == 3`

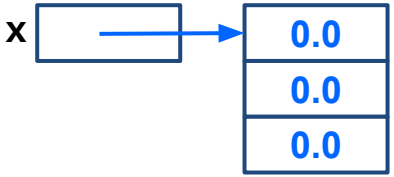
5.1. Arrays

- Beispiele (1)
  - Deklaration:

```
double[] x;   // Bevorzugte Schreibweise
double y[];   // Geht auch
```

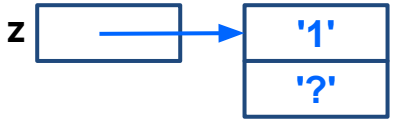
  - Erzeugung des (eigentlichen) Arrays

```
x = new double[3];  // 3 Elemente, Initialwerte 0.0
```



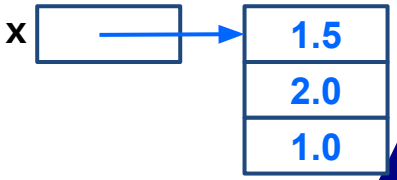
  - Deklaration, Erzeugung und Initialisierung

```
char[] z = { '1', '?' };  // automatisches new
```



  - Zugriff auf Array-Elemente:

```
for (int i = 0; i < x.length; i++)
   x[i] = 3.0 - i;
x[0] = 1.5;   // ersetzt alten Wert 3.0
```



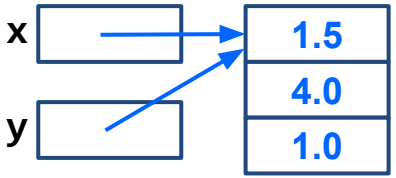
5.1. Arrays

- Beispiele (2)
  - Spezieller Wert **`null`** verweist nirgendwohin:

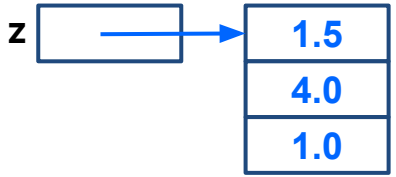
```
double[] y = null;
```

  - Zugriffe auf das ganze Array:

```
// erzeugt neue Referenzvariable y, die
// auf dasselbe Array zeigt wie x:
double[] y = x;
// aendert auch den Wert von x[2]:
y[1] = 4.0;
```

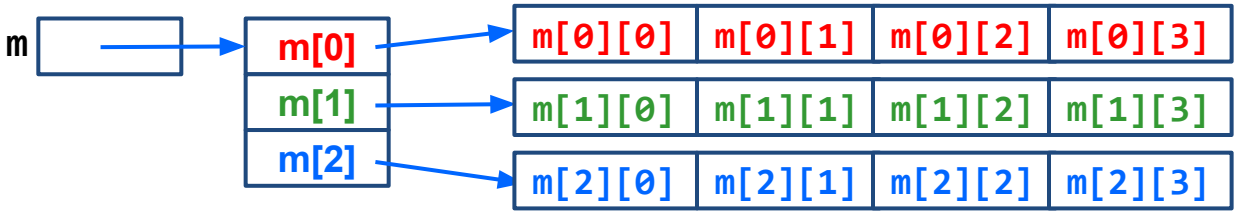



```
// So wird z eine echte Kopie von x:
double[] z = new double[x.length];
for (int j = 0; j < x.length; j++)
   z[j] = x[j];
```



5.1. Arrays

- Mehrdimensionale Arrays
  - Die Elemente eines Arrays können auch Referenzen auf Arrays enthalten:



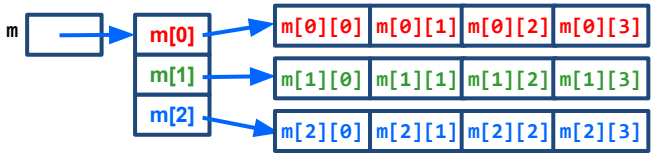
  - Deklaration der Referenzvariable:

```
int[][] m;          // 2-dimensionales Array: Matrix
int m[][];          // 2-dimensionales Array: Matrix, alternativ
int[][][] mat3d;    // 3-dimensionale Matrix
int m3d[][][];      // 3-dimensionale Matrix, alternativ
```

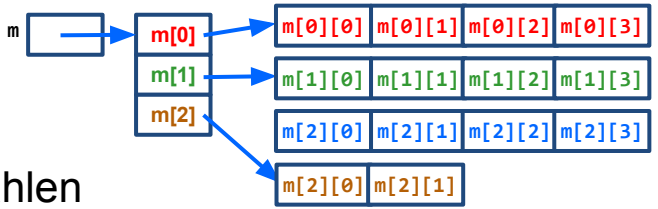
5.1. Arrays

- Mehrdimensionale Arrays
  - Erzeugung des Arrays:

```
                          //      | 0 0 0 0 |
m = new int[3][4];        // m =  | 0 0 0 0 |
                          //      | 0 0 0 0 |
```




```
m[2] = new int[2];        //      | 0 0 0 0 |
m[2][1] = 7;              // m =  | 0 0 0 0 |
                          //      | 0 7     |
```



  - `m[2]` ist eine Referenz auf ein Array von `int`-Zahlen
  - `m[2]` hat den Typ `int[]`
  - `m.length == 3`
  - Die Länge der Zeilen kann variieren:
    `m[0].length == 4`, aber `m[2].length == 2`

5.1. Arrays

- Mehrdimensionale Arrays
  - `m = new int[3][4];` ist gleichbedeutend mit:

```
m = new int[3][];          // Lege Array für 3 Zeilenverweise an
for (int i=0; i<3; i++)    // Initialisiere die
   m[i] = new int[4];      // Zeilenverweise
```

  - aber:

    `m = new int[][4];` führt zu Fehlermeldung, da

    - die Referenzen auf die Zeilen nirgends abgespeichert werden können
    - die Anzahl der zu erzeugenden Zeilen unbekannt ist

Beispiel (🌶️🌶️)

```
/**  Histogramm: Zeichnen Sie ein Histogramm:
  */
import java.util.Random;   // random number generator

class Histogram {
  public static void main( String[] str ) {
    int[] liste;  liste = new int[25];
    Random rand = new Random();
    // Fuelle liste mit beliebige Integer in [0,24]:
    for ( byte i = 0; i < liste.length; i++ ) {
      liste[i] = rand.nextInt( liste.length );
    }
    // Fuelle erst ein 2D-Array von Booleans
    // Zeichne dann ein Histogramm (pro-Tipp: benutze "\u2589"-Block)
  }
}
```

Beispiel

Beispiel

5.2. Strings

- In Java ist ein String (Zeichenkette) eine Folge von (Unicode-)Zeichen
  Beispiel:

```
String motto = "Wir lernen Java!";  // automatisches new
```

- **`motto`** ist eine Referenzvariable
  - speichert nicht den String, sondern nur die Referenz darauf
- jedes Zeichen hat eine Position (gezählt ab 0):

5.2. Strings

- Nach einer neuen Zuweisung an die Referenzvariable, z.B.

```
motto = "Carpe Diem";
```

ist der String `"Wir lernen Java!"` nicht mehr zugreifbar und wird vom Garbage Collector gelöscht

- Die folgenden Zuweisungen sind unterschiedlich:

```
motto = null;  // motto zeigt auf keinen String mehr
motto = "";    // motto zeigt auf den leeren String
```

5.2. Strings

Operationen auf Strings

- Zusammenfügen (Konkatenation):

```
String s = "Zahl 1" + "2";      // ”Zahl12”
s += " ist gleich 3";           // ”Zahl12 ist gleich 3”
s = "elf ist " + 1 + 1;         // ”elf ist 11”
s = 1 + 1 + " ist zwei";        // ”2 ist zwei”
s = 0.5 + 0.5 + " ist " + 1;    // ”1.0 ist 1”
```

- Der Operator **+** ist überladen:
  - **`int + int`**: Addition
  - **`String + String`**: Konkatenation
  - **`String + int`** und **`int + String`**: Umwandlung der Zahl in einen String und anschließende Konkatenation
  - analog für andere Datentypen

5.2. Strings

Operationen auf Strings

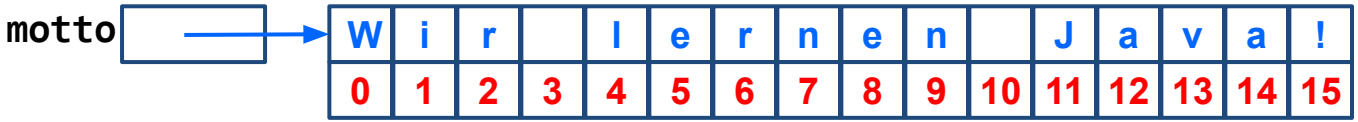
- Vergleichsoperatoren (nur `==` und `!=`):
  - der Operator == liefert true, wenn beide Operanden auf denselben String verweisen (Objektidentität)
- Vergleich:
  - `s1.equals(s2)` liefert true, wenn **`s1`** und **`s2`** zeichenweise übereinstimmen
  - `s1.compareTo(s2)`

    < 0, wenn **`s1`** alphabetisch vor **`s2`**

    = 0, wenn **`s1`** und **`s2`** zeichenweise übereinstimmen

    > 0, wenn **`s1`** alphabetisch nach **`s2`**
- Länge: `motto.length()`

5.2. Strings



Weitere Methoden

- Vergleich mit Anfang und Ende:

```
boolean a = motto.startsWith("Wir");    // true
boolean b = motto.endsWith(".");        // false
```

- Zugriff auf einzelne Zeichen:

```
char c = motto.charAt(5);      // 'e'
```

- Suche nach Zeichen:

```
int i = motto.indexOf('e', 0);          // ab Index 0: 5
int j = motto.lastIndexOf('e', 15);     // ab Index 15: 8
```

- Ausschneiden / Ersetzen:

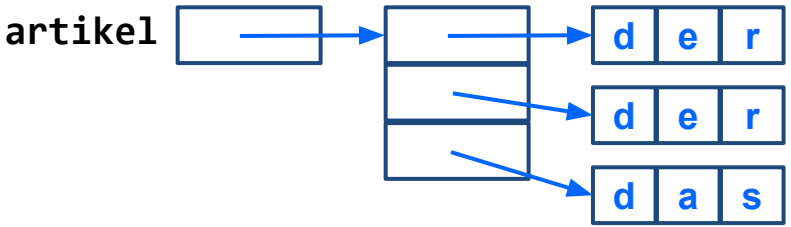
```
String s = motto.substring(11, 15);     // "Java!"
String t = motto.replace('a', 'A');     // "Wir lernen JAvA!"
```

5.3. Strings und Arrays

- String in Array von char umwandeln:
```
char[] textArray = motto.toCharArray();
```

- Array von char in String umwandeln:
```
String str = new String(textArray);
```

- Array von Strings:
```
String[] artikel = { "der", "die", "das" };
```



5.3. Strings und Arrays

- Strings können nicht verändert werden (”***immutable***“)
  - die Operation **+** oder z.B. die Methode **`replace`** erzeugen jeweils einen neuen String
  - dadurch verhalten sich Strings wie einfache Datentypen
    - z.B. *call-by-value-Semantik bei Parameter übergabe*
- Es gibt auch eine Klasse **`StringBuffer`**, deren Objekte auch verändert werden können
  - Geschwindigkeitsvorteil, wenn viele Manipulationen an Strings vorgenommen werden
- Dokumentation der Klasse String im WWW unter
  https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/String.html

Beispiel (🌶🌶🌶)

```
/**  SimpleWordle: Implementieren Sie ein einfaches Wordle-Spiel mit einer
     unendlichen Anzahl von Versuchen. Verwenden Sie diese Vorlage:
  */
import java.util.Random;   // random number generator
import java.util.Scanner;  // scan strings in der Konsole
class SimpleWordle {
  public static String dialog( String hint ) {  // Zeige Hinweis, Eingabe
    System.out.print( hint + ", what is your guess? ");
    Scanner scan = new Scanner(System.in);
    return scan.next();
  }
  public static void main( String[] str ) {
    String loesung = "weary"; // Loesunswort
    String hint = "-----";    // Hinweis mit gefundenen Buchstaben
    // Implementiere ein Schleife, die dialog() ausführt & aufhört
    // wenn das Wort geraten wurde:
  }
}
```

Beispiel (🌶️🌶️🌶️)

Beispiel (🌶️🌶️🌶️)