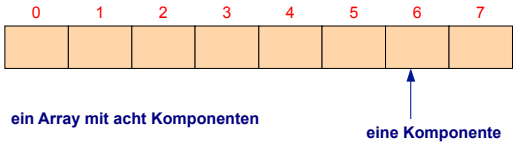
Arrays

- wie in C/C++ sind Arrays (Felder) Dinge, die sich aus gleichartigen Komponenten zusammensetzen
  - man veranschaulicht sich Arrays am besten so:



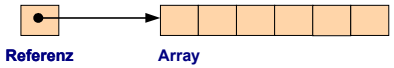
  - die Komponenten dienen zur Aufnahme von Daten
    - in jeder Komponente genau ein Wert

- mit „Gleichartigkeit“ der Komponenten ist gemeint, dass alle Komponenten denselben Datentyp haben
  - er wird Komponentendatentyp des Arrays genannt
- welche Datentypen sind als Komponentendatentyp von Arrays zulässig?
  - alle, die einen Namen haben, d.h. alle primitiven Datentypen (`byte`, `short`, `int`, `long`, `char`, `float`, `double`, `boolean`) und alle Referenzdatentypen (Klassen, `enum`-Klassen, Arrays, Interfaces)

    Achtung: für Arrays, deren Komponentendatentyp ein Arraydatentyp ist, gibt es eine spezielle Notation

- welchen Datentyp haben Arrays?
  - ist der Komponentendatentyp eines Arrays `int`, hat das Array den Datentyp `int[]`
  - ist der Komponentendatentyp eines Arrays `double`, hat das Array den Datentyp `double[]`
- allgemein:
  - ist der Komponentendatentyp eines Arrays `A`, hat das Array den Datentyp `A[]`
  - das gilt auch, wenn der Komponentendatentyp eine Klasse, eine `enum`-Klasse oder ein Interface ist
- es gibt daher unendlich viele Arraydatentypen

- in Java ist jeder Arraydatentyp ein Referenzdatentyp
- dies hat folgende Konsequenzen:
  - Arrays (= die Werte von Arraydatentypen) werden – wie alle Objekte – immer auf dem Heap gespeichert
  - der Zugriff auf Arrays ist immer nur über Referenzen (= Zeiger) möglich



- wie sind Referenzen auf Arrays zu deklarieren?
  - zum Beispiel so:

```
int[] x;
```

`x` ist eine Referenz auf Arrays, deren Komponentendatentyp `int` ist (kurz: eine Referenz auf `int`-Arrays)

```
double[] messwert;
```

`messwert` ist eine Referenz auf `double`-Arrays

```
String[] menuePunkt;
```

`menuePunkt` ist eine Referenz auf `String`-Arrays

```
PapierFormat[] format;
```

`format` ist eine Referenz auf `PapierFormat`-Arrays

- **Achtung:** wie bei allen Referenzdatentypen werden dadurch lediglich Referenzen (= Zeiger) deklariert
  - die eigentlichen Arrays (= Objekte) müssen erst noch mit Hilfe des **`new`**-Operators erzeugt werden!
  - um das zu verdeutlichen, ist es auch bei Arraydatentypen sinnvoll, die Referenzen zunächst mit **`null`** zu initialisieren:

```
int[] x = null;
double[] messwert = null;
String[] menuePunkt = null;
PapierFormat[] format = null;
```


- **Hinweis:** als Reminiszens an C/C++ könnten Referenzen auf Arrays auch so deklariert und initialisiert werden

```
int x[] = null;
double messwert[] = null;
String menuePunkt[] = null;
PapierFormat format[] = null;
```

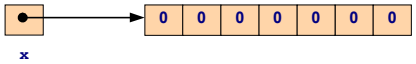
  - das ist aber äußerst unüblich und sollte daher vermieden werden

- die Arrays selbst können mit einer Variante des **`new`**-Operators zum Beispiel so erzeugt werden:

```
int[] x = null;
...
x = new int[7];      // erzeugt int-Array mit 7 Komponenten
```

- diese (neue) Form des **new**-Operators hat folgende Auswirkungen:
  - auf dem Heap wird Speicherplatz für ein **int**-Array mit sieben Komponenten allokiert
  - die Anfangsadresse des **int**-Arrays wird in **x** gespeichert
  - die sieben Komponenten des **int**-Arrays werden jeweils mit **0** initialisiert

- die dadurch entstandene Situation veranschaulichen wir uns so:



- bei dieser Variante des **new**-Operators für Arrays sind mehrere Dinge zu beachten:
- bei primitiven Datentypen als Komponentendatentyp muss der hinter dem Schlüsselwort **new** angegebene Datentyp identisch sein mit dem in der Deklaration der Referenz angegebenen Datentyp


```
int[] x = null;
x = new byte[7];                    // falscher Datentyp!
```

führt zu folgender Fehlermeldung:

```
incompatible types
found   : byte[]
required: int[]
  x = new byte[7];
      ^
1 error
```

- ist aber der Komponentendatentyp ein Referenzdatentyp, genügt es, wenn der hinter dem Schlüsselwort **new** angegebene Datentyp lediglich zuweisungskompatibel ist zu dem in der Deklaration der Referenz angegebenen Datentyp

- beim Erzeugen von Arrays mit **new** muss die Anzahl der Komponenten des Arrays angegeben werden
  - sie kann während der Ausführung des Programms nicht mehr geändert werden, d.h. sie ist statisch

```
int[] x = null;
x = new int[];          // Anzahl der Komponenten fehlt!
```

führt zu folgender Fehlermeldung:

```
array dimension missing
  x = new int[];
              ^
1 error
```

- wie in C/C++ wird die Anzahl der Komponenten eines Arrays als Länge des Arrays bezeichnet
- wie lang dürfen Arrays sein?
  - die Längenangabe hat den Datentyp **`int`**
  - ein Array kann daher maximal (2^{31} - 1) Komponenten haben
  - Arrays dürfen aber nie eine negative Länge haben:

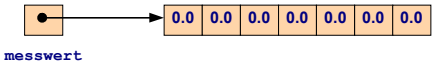
```
int[] x = null;
x = new int[-3];       // negative Länge nicht zulässig!
```

diese Anweisungen werden zwar vom Compiler übersetzt, führen aber bei der Ausführung durch die JVM zu einer **`NegativeArraySizeException`**

- es ist aber zulässig, Arrays der Länge 0 zu erzeugen
  - derartige leere Arrays haben keine Komponenten
  - wie wir bald sehen werden, gibt es Situationen, in denen sich diese Möglichkeit als nützlich erweist
- der **`new`**-Operator für Arrays sorgt stets dafür, dass die Komponenten der erzeugten Arrays initialisiert werden
  - zur Initialisierung werden dieselben Werte benutzt, wie bei der Initialisierung von Attributen

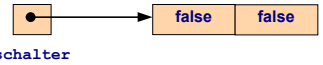

```
double[] messwert = null;
messwert = new double[7];
```

führt zu:




```
boolean[] schalter = null;
schalter = new boolean[2];
```

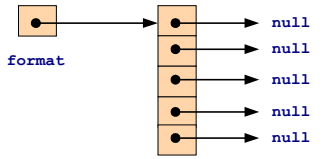
führt zu:



- **Achtung:** wenn der Komponentendatentyp ein Referenzdatentyp ist, enthält ein mit dem **`new`**-Operator erzeugtes Array zunächst keine Objekte, sondern lediglich Referenzen auf **`null`**


```
PapierFormat[] format = null;
format = new PapierFormat[5];
```

führt zu:



- die Objekte müssen erst noch in einem zweiten Schritt mit Hilfe des **`new`**-Operators erzeugt werden!