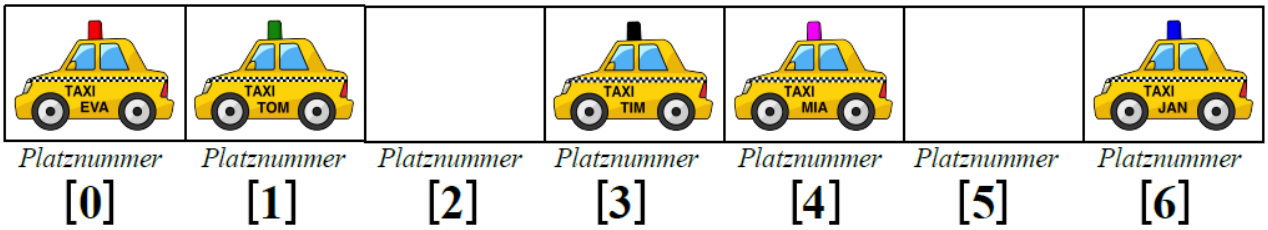
Arrays / Felder

Die Datenstruktur ***Array*** speichert (beliebig) ***viele Objekte desselben Datentyps***. Man kann Arrays mit einem Taxistand vor dem Bahnhof vergleichen:



- Es ist immer eine ***festgelegte Größe*** an Plätzen vorhanden (hier: 7 Stück).
- Auf einem Taxistand dürfen nur Taxis stehen, nichts anderes. → Ein Array verwaltet auch nur Objekte ***eines gemeinsamen Datentyps***.
- Es müssen nicht alle Plätze benutzt werden, einige können auch leer sein.
- Es können nicht mehr Objekte als Felder vorhanden gespeichert werden.
- Die Plätze werden – ***beginnend mit [0]*** – durchnummeriert.

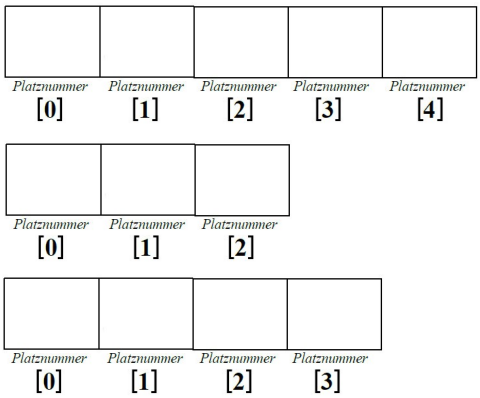
Programmierung in Java

Deklaration mit eckigen Klammern hinter dem Datentyp:

```
FIGUR[ ] nüsse;
int[ ] punkte;
String[ ] namen;
```

Das Array (der Parkplatz, der Container) selbst ist ein ***eigenständiges Objekt*** und muss mit einer ***festen Größe initialisiert*** werden:

```
nüsse = new FIGUR[5];

punkte = new int[3];

namen = new String[4];
```

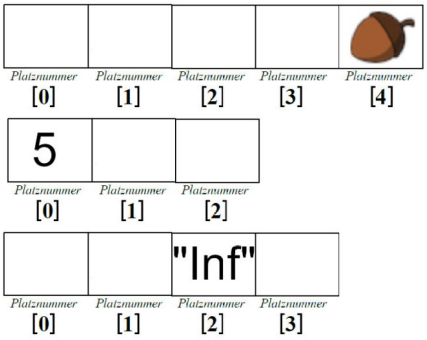


Die Arrays sind nach dem Erzeugen allerdings noch leer und speichern noch nichts!

Man kann jeden Platz im Array einzeln ansprechen und darin Werte/Objekte speichern.

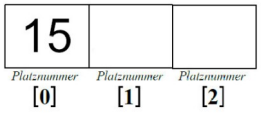
Beachte, dass bei Arrays von Objekten jedes einzelne Objekt erst noch erzeugt werden muss.

```
nüsse[4] =  new FIGUR("nuss.png");

punkte[0] = 5;

namen[2] = "Inf";
```



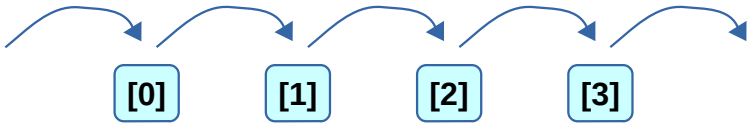
Man mit den gespeicherten Inhalten (primitive Datentypen als auch Referenzobjekten) arbeiten, indem man sie über ihren Platz im Array anspricht:

```
nüsse[4].setzeMittelpunkt(10,10);
punkte[0] = punkte[0] + 10;
namen[2] = "Informatik";
```



Iteration über ein Array mit der Zählschleife

Ein Array zu ***iterieren*** (= durchlaufen) bedeutet, für alle Mitglieder dieser Sammlung eine (ähnliche) Aktion auszuführen.




```
for( int i = 0; i < hühner.length; i++ ){
        hühner[i] = new FIGUR("moorhuhn.png");
        hühner[i].setzeMittelpunkt(0,0);
}
```

```
for( int i = 0; i < zahlen.length; i++ ){
        zahlen[i] = zahlen[i] * 5;
}
```

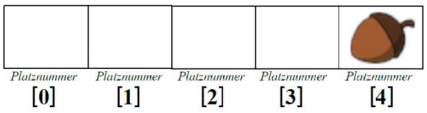
Das Array selbst ist ein Objekt und somit ist das Array durch die Punktnotation „ansprechbar". Mittels des Befehls ***array.length*** nennt das Array seine Länge, mit welcher es initialisiert wurde.

Hinweis: length ist keine Methode, sondern ein Attribut (eine Eigenschaft) des Arrays. Deshalb werden nach length auch keine Klammern () benötigt.

Mögliche Fehlerausgaben (Exceptions)

Neben ***Syntaxfehlern***, die durch den Übersetzer/Compiler ***beim Programmieren*** direkt rot angestrichen werden, gibt es noch ***Laufzeitfehler***. Diese treten erst ***beim Ausführen des Programms*** auf und führen zum Absturz.

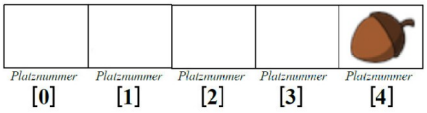
NullPointerException: Es wird eine Methode auf einer Referenz aufgerufen an deren Speicheradresse kein Objekt initialisiert wurde.



nüsse[2].setzeMittelpunkt(0,0);

```
java.lang.NullPointerException: Cannot invoke "FIGUR.setzeMittelpunkt(double, double)" because "this.nüsse[2]" is null
```

ArrayIndexOutOfBoundsException: Es wird auf eine Platznummer/Index im Array zugegriffen, welcher im Array nicht existiert.



nüsse[10].setzeMittelpunkt(0,0);
nüsse[-1].setzeMittelpunkt(0,0);

```
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
```

Geschickte Nutzung der Zählvariable

Falls Objekte oder Zahlen ein bestimmtes Muster aufweisen sollen, kann man die Zählvariable geschickt verwenden, um dieses Muster ***in Abhängigkeit der Zählvariable*** darzustellen:

Beispiel: Das Array soll aufsteigend mit geraden Zahlen befüllt werden.

```
private int[ ] zahlen;
zahlen = new int[9];
for( int i = 0; i < zahlen.length; i++ ){
        zahlen[i] = i*2;
}
```

Hier ist das Array zahlen mit den zugehörigen Belegungen veranschaulicht: