7. Arrays

Gelegentlich braucht man für ein Programm mehrere Attribute desselben Datentyps oder derselben Klasse.

Beispiel:

In der Highscore-Liste eines Spiels werden von den 10 besten Spielern die Namen und die Punktestände gespeichert.

Dazu bräuchte man 10 Attribute vom Typ *String*

und weitere 10 Attribute vom Typ *int*.

Name	Punkte
Hans	12345
Susi	9876
Peter	8765
...	...

Beim Deklarieren und Initialisieren der Liste bräuchte man oft zueinander sehr ähnlichen Code:

String spielerPlatz1; String spielerPlatz2; ...

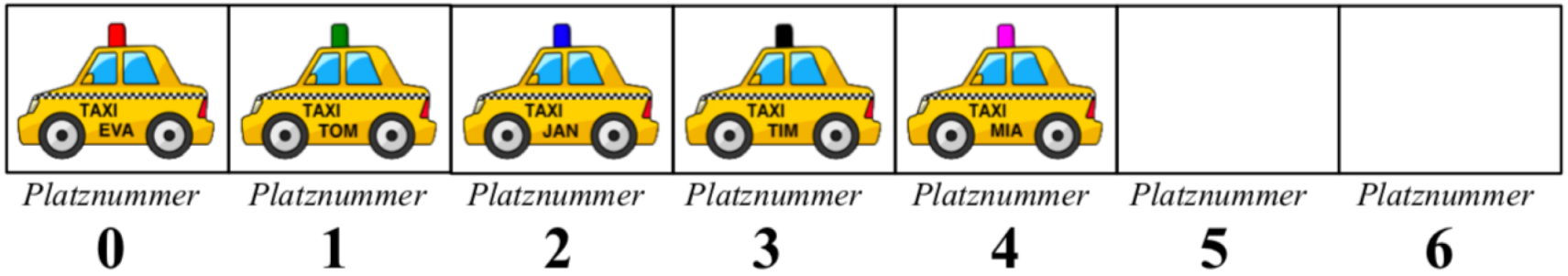
int punktePlatz1; int punktePlatz2; ...

...

Ähnlich verhält es sich beim Verwalten der Liste. Wird z.B. ein neuer erster Platz eingefügt, müsste zum Nachrücken der anderen unnötig viel Code geschrieben werden.

Dafür gibt es eine Datenstruktur, in der beliebig viele Attribute desselben Datentyps oder derselben Klasse gespeichert werden können. Diese Datenstruktur heißt ***Array*** oder auch ***Feld***.

Man kann es mit einem Taxistand vor dem Bahnhof vergleichen:



- Es ist eine **bestimmte Anzahl an Plätzen** vorhanden (hier 7 Stück).
- Auf einem Platz (Index) darf **nur** ein **Taxi** und nicht etwas anderes stehen.
- **Es müssen nicht alle Plätze belegt sein**, einige können auch leer (null) sei.
- Mehr als die **maximale Anzahl** an Taxen kann dort nicht stehen.
- Die Plätze werden von ***0*** bis ***Länge des Feldes – 1*** durchnummeriert.

Im Unterschied zu einem realen Taxistand gilt bei einem Array:

- Man muss nicht in einer bestimmten Reihenfolge auffüllen oder Plätze leeren. Im obigen Beispiel könnte also auch das Taxi auf Platz 2 wegfahren oder ein anderes Taxi könnte auf Platz 6 fahren.

MERKE

Arrays in Java

Array deklarieren:

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

Array initialisieren:

```
namen = new String [10];
punkte = new int [10];
```

Plätze mit Werten oder Objekten füllen:

```
namen[0] =“Hans“ ;
namen[1] =“Susi“ ;
...
punkte[0] = 12345;
punkte[1] = 9876
```

Zugriff auf Array-Plätze:

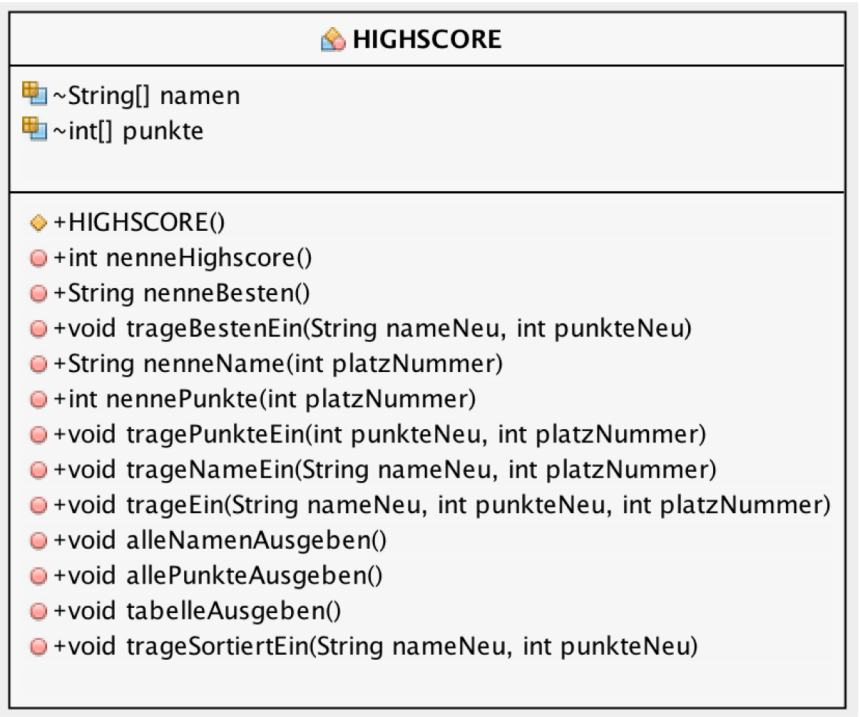
```
public String nenneBesten(){
    return namen[0];
}
```

Übung 1 – Arrays, praktischer Einstieg

a)
Starte BlueJ und lege ein neues Projekt mit dem Namen *HighScore* an.
b)
Erstelle darin eine Klasse **HIGHSCORE**.
Deklariere das *String*-Array ***namen*** und
das *int*-Array ***punkte***.
c)
Erstelle einen Konstruktor ohne Übergabeparameter und führe folgende Initialisierungen durch:

- das Array *namen* vom Typ *String* und der Länge 10
- das Array *punkte* vom Typ *int* und der Länge 10
- trage in *namen* einige Namen ein
- trage in *punkte* einige Punktestände ein; lass in beiden Arrays noch Plätze frei.

d)
Erzeuge ein Objekt von HIGHSCORE und untersuche die Arrays im Objektinspektor. Welche Werte sind auf den Plätzen 0 bis 9?



Name	Punkte
Hans	12345
Susi	9876
Peter	8765
…	…

Übung 1 – Arrays, praktischer Einstieg

e)
Schreibe die beiden Methoden ***nenneHighscore()*** und
nenneBesten(), die jeweils den Inhalt des Arrays am Platz (Index) 0 zurückgeben.

f)
Schreibe die Methode ***trageBestenEin(String nameNeu, int punkteNeu***).
In dieser Methode wird in den beiden Arrays jeweils der Platz mit Index 0 mit dem Wert des entsprechenden Übergabeparameters überschrieben.
Teste deine Methode sowohl mit dem Objektinspektor als auch mit den Methoden aus e).

g)
Schreibe die Methoden ***nenneName(int platzNummer***) und ***nennePunkte(int platzNummer),***
die jeweils den Wert des Objekts an der übergebenen Platznummer zurückgeben.

h)
Schreibe die Methoden
tragePunkteEin(int punkteNeu, int platzNummer)
trageNameEin(String nameNeu, int platzNummer)
trageEin(String nameNeu, int punkteNeu, int platzNummer),
in denen jeweils die übergebenen Werte an der passenden Stelle eingetragen werden.
Für die letzte dieser Methoden solltest du dich bei den beiden vorherigen bedienen, damit du nicht denselben Code mehrmals schreiben musst.

Übung 1 – Arrays, praktischer Einstieg

i)
Schreibe die Methoden ***alleNamenAusgeben()***, welche in einem Textfenster alle Namen in einer Zeile ausgibt.

Anleitung:

Man verwendet hierzu eine Zählschleife
Die Länge der Zählschleife ist hier 10 (von 0 bis 9), besser verwendet man aber *namen.length* , da dann der Quelltext nicht angepasst werden muss, falls man die Feldlänge verändert.

```
public void alleNamenAusgeben(){
    for (int i = 0; i<=namen.length-1; i++){
        System.out.print(namen[i]+" ");
    }
}
```

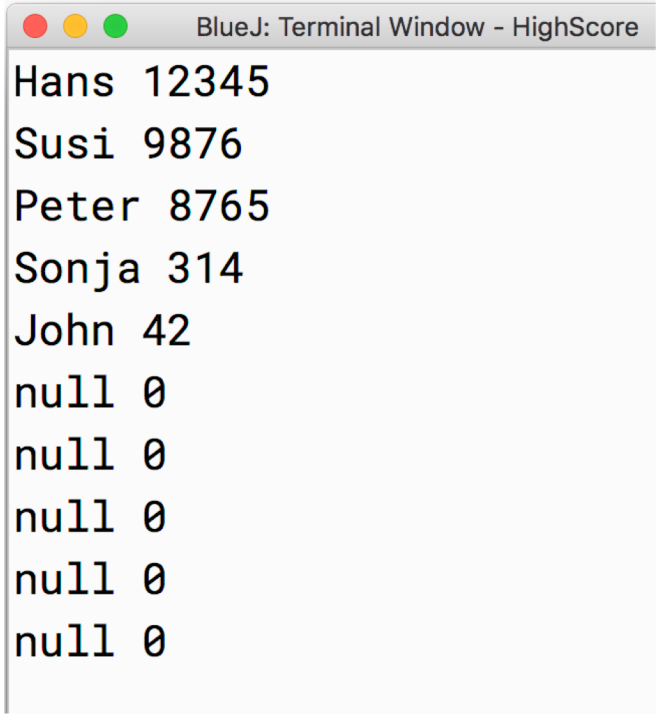
Zur Ausgabe in einem Textfenster dient die Methode *System.out.print(String s*).
Man kann auch mehrere Strings getrennt durch das Zeichen + in die Klammer schreiben.
In unserem Fall besteht der zweite String einfach aus einem oder mehreren Leerzeichen: “ “:

Teste die Methode.

Beobachte auch die Fehlermeldung, die erscheint, wenn du die Zählschleife länger als die Feldlänge programmierst, also z.B. von 0 bis *namen.length*.
Dies ist ein sehr häufiger kleiner Programmierfehler, deshalb ist es gut, sich diese Fehlermeldung zu merken.

Übung 1 – Arrays, praktischer Einstieg

j)
Schreibe analog die Methode ***allePunkteAusgeben()***,
welche in einem Textfenster alle Punkte in einer Zeile ausgibt.

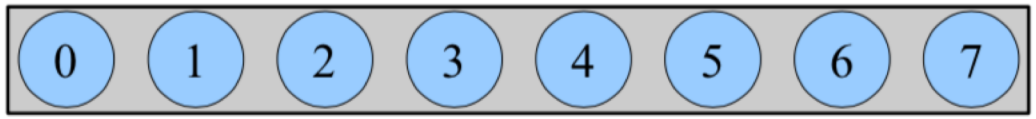


k)
Schreibe die Methode ***tabelleAusgeben()***,
die pro Zeile jeweils den Namen und den Punktestand ausgibt.
Hinweis:
Die Methode *System.out.println(String s)*
erzeugt nach der Ausgabe eine neue Zeile.

l)* (etwas herausfordernd)
Schreibe die Methode **trageSortiertEin(String nameNeu, int punkteNeu),**
die eine nach den Punktetänden sortierte Liste erzeugt,
vorausgesetzt, die bisherigen Einträge sind bereits sortiert.

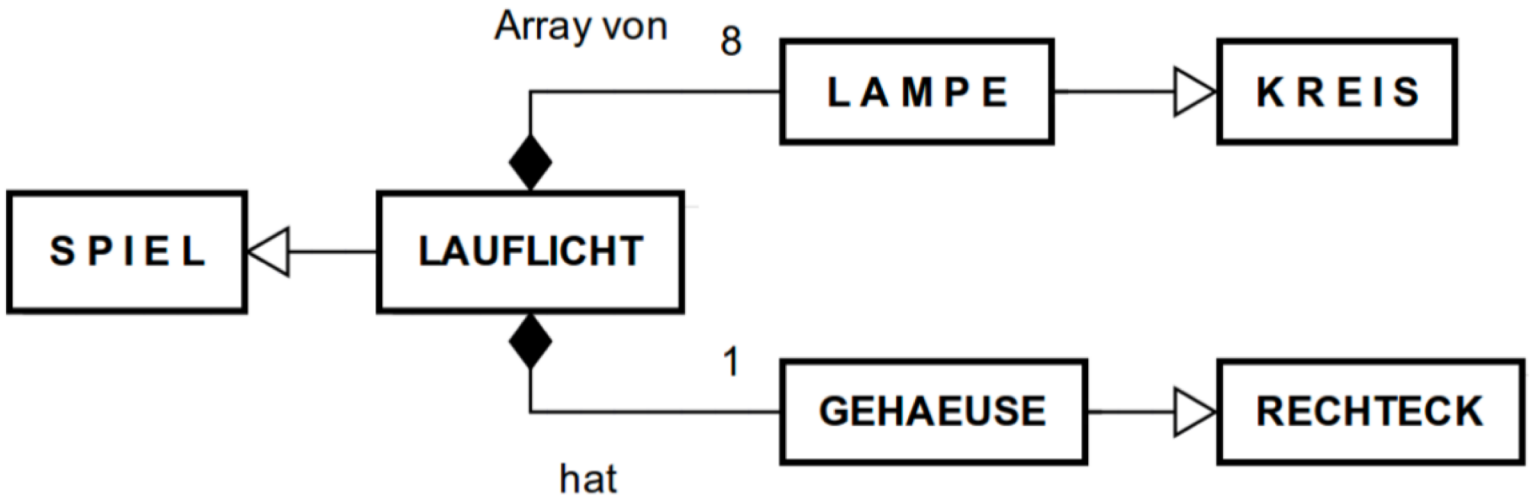
In der Methode soll zuerst die passende passende Platznummer gesucht werden.
Alle Spieler und ihre Punktestände ab diesem Platz werden dann um eins nach hinten gerückt.
Der neue Name und der Punktestand werden an den so erzeugten freien Stellen eingetragen

Übung 2 – Lichtorgel

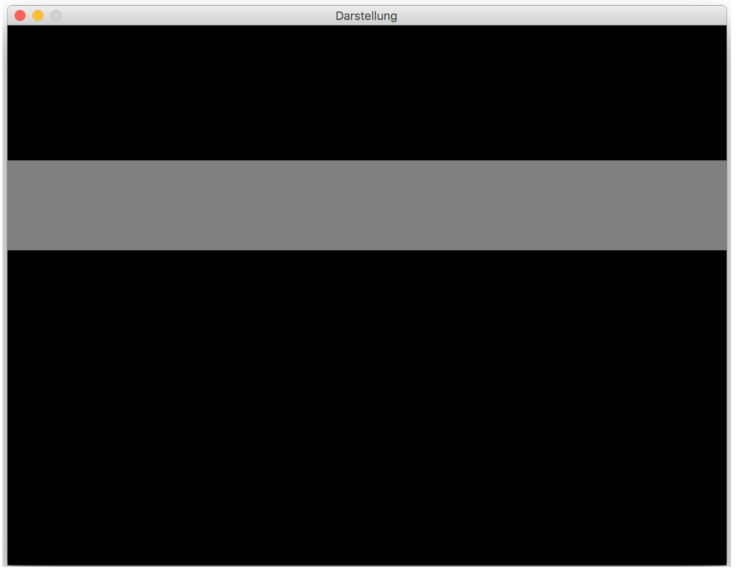


Du sollst ein Lauflicht, ähnlich einer einfachen Lichtorgel programmieren.
Die Lichtorgel besteht aus 8 Lampen in einem Gehäuse.
Die Lampen werden durch Kreise, das Gehäuse durch ein Rechteck modelliert.

Klassendiagramm:

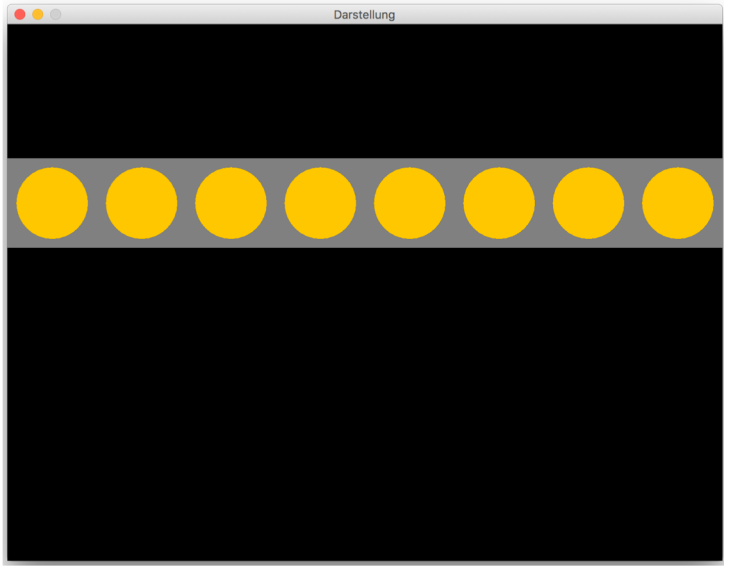


Übung 2 – Lichtorgel



Öffne das BlueJ-Projekt **Lichtorgel_Vorlage .**
Erzeuge ein Objekt der Klasse **LICHTORGEL.**
Die Klasse erbt von **SPIEL** und erzeugt ein Objekt der Klasse **GEHAEUSE**, die von **RECHTECK** erbt.
In dieses Gehäuse sollen die 8 Lampen als Array gesetzt werden.
Anschließend wirst du einige Lichteffekte programmieren.

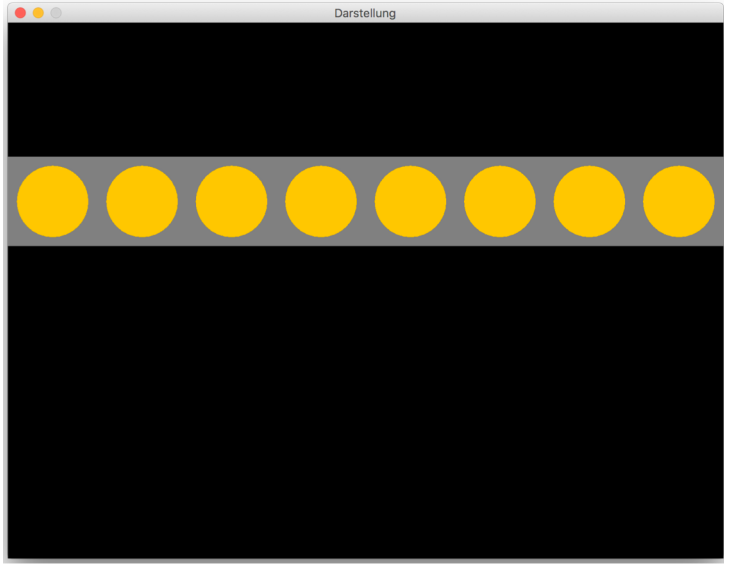
a)
Finde anhand des Quelltextes heraus:
Welche Maße hat das Fenster?
Welche Maße und welche Position hat das Gehäuse?



b)
Erzeuge ein Objekt von **LAMPE** und teste die Methoden ***an()*** und ***aus()***.
Informiere dich im Quelltext, wie sie realisiert werden.

Die mögliche Farben bei der engine alpha findest du im Dokument Farben_TastenCode.pdf im Ordner Skripten_und_Programme.

Übung 2 – Lichtorgel



c)
Bevor du die Klasse LICHTORGEL implementierst,
solltest du dir die Positionen der 8 Lampen überlegen.

Stelle insbesondere eine Formel auf, welche die x-Koordinate des Mittelpunkts der Lampe mit dem Index i (i läuft von 0 bis 7) berechnet.

Für den Radius der Lampen ist 40px ein sinnvoller Wert.

d)
Deklariere in der Klasse **LICHTORGEL** das Array ***lampen*** vom Typ LAMPE und initialisiere es im Konstruktor.
Besetze jeden Platz des Arrays mit einem Objekt der Klasse LAMPE.
Verwende dazu eine Zählschleife und setze die Formel aus Aufgabe c) an der passenden Stelle im Konstruktor von LAMPE ein.

Teste deine Klasse. Sind alle Lampen sichtbar und an der korrekten Position?

Übung 2 – Lichtorgel

Nun kann es endlich an das Programmieren von Lichteffekten gehen.
In der Klasse LICHTORGEL erkennst du, dass die geerbte Methode ***tick()*** mit einem leeren Rumpf überschrieben wurde. Dadurch erreicht man, dass das Textfenster mit tick...tack nicht erscheint.
Eine nützliche von SPIEL geerbte Methode ist ***warte(int millisec)***.
Sie veranlasst, dass das Programm etwas wartet, bevor es den nächsten Befehl ausführt.

e)
Schreibe die Methoden ***alleAus()*** und ***alleAn()***, die jeweils mithilfe einer Zählschleife alle Lampen ein- bzw. ausschalten.

f)
Schreibe die Methode ***setzeFarbeAlleLampen(String farbeNeu)***, die alle Lampen auf die Farbe des Übergabeparameters setzt.

g)
Schreibe die Methode ***laufeNachRechts()***.
Die Methode schaltet zuerst alle Lampen aus. Anschließend wird nacheinander jede Lampe eingeschaltet, etwas gewartet und die Lampe dann wieder ausgeschaltet.
Schreibe analog eine Methode ***laufeNachLinks()***.

Übung 2 – Lichtorgel

h)

Erfinde eigene Lichteffekte!

Vielleicht benötigst du auch die Möglichkeit, eine Lampe auf eine zufällig ausgewählte Farbe zu setzen.
Dazu ist es sinnvoll, in der Klasse LAMPE eine Methode setzeZufallsFarbe() zu schreiben.
In dieser Methode kannst du dann eine Zufallszahl erzeugen und mit einer Mehrfachauswahl (if – else if oder auch mit switch, case; siehe Kap. 03_Bedingte_Anweisungen) die Farbe der Lampe zufällig setzen.

Wie man Zufallszahlen in Java erzeugt, ist auf der nächsten Seite beschrieben.

MERKE

Zufallszahlen in Java

Möglichkeit 1 - mithilfe der Klasse Random:

- *Importiere dazu am Anfang der Klasse LAMPE noch vor dem Klassenkopf das benötigte Java-Package:*
  import java.util.*;

- *Deklariere dann ein Objekt zgen der Klasse Random und initialisiere es im Konstruktor:*
  zgen = new Random(); *zgen ist dann so etwas wie ein Zufallsgenerator.*

- *Nun kannst du die Methoden der Klasse Random nutzen:*
  zgen.nextInt(9) *erzeugt z.B. eine ganzzahlige Zufallszahl zwischen 0 und 8.*

Möglichkeit 2 - mithilfe der Klasse Math:

Ein Import eines Packages oder das Erzeugen eines Objekts ist hier nicht nötig.
Du kannst mit ***Math.random()*** *direkt eine Zufallszahl erzeugen. Sie ist vom Typ* ***double*** *und liegt im Intervall* ***]0;1[.***

Um ganzzahlige Zufallszahlen in einem bestimmten Intervall zu erzeugen, muss man etwas tricksen:
Beispiel:

int zzahl = (int) (1000*Math.random());	*erzeugt eine ganzzahlige Zufallszahl im Intervall [0;1000[.*
zzahl%6;	*berechnet den Rest bei der Division durch 6, die Zahl liegt also im Intervall]0;5[.*
1+ zzahl%6;	*erzeugt eine ganzzahlige Zufallszahl im Intervall [1;6].*

In beiden Fällen werden die Zufallszahlen mithilfe eines Algorithmus erzeugt, was der eigentlichen Vorstellung von Zufall widerspricht. Man nennt solche Zahlen deshalb **Pseudozufallszahlen**.