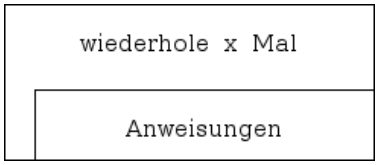
Zählschleifen – Wiederholung mit fester Anzahl

Auch die Wiederholung mit fester Anzahl kennst du bereits aus der 7. Klasse NuT-Informatik. Erinnere dich:

Zählschleife bzw. ***Wiederholung mit fester Anzahl***.

Ein Befehl oder auch eine Sequenz von Befehlen soll mehrmals wiederholt werden. Du schreibst den Befehl bzw. die Sequenz aber nicht mehrmals in dein Programm sondern gibst vorher an, wie oft wiederholt werden soll.

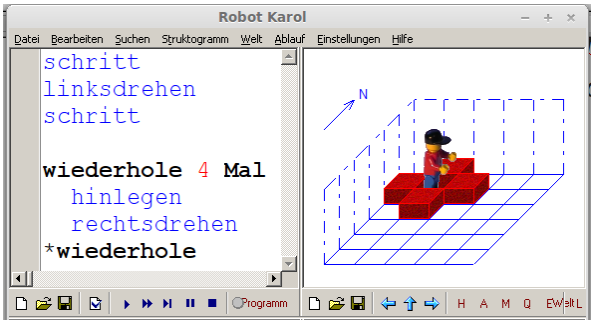


Alles, was im Struktogramm eingerückt ist, wird x Mal wiederholt.

35. **Robot Karol – Erinnerungen 2**

a) *Öffne die Umgebung „Robot Karol“.*

b) *Schreibe folgendes Programm:*



c) *Führe das Programm durch Klick auf die Start-Taste aus.*

d) *Zeichne ein Struktogramm zu diesem Programm.*

e) *Wie müssten all diese Methodenaufrufe aus Objekt-orientierter Sicht in Punktnotation lauten? Schreibe die Antwort in deinem Hefter auf.*

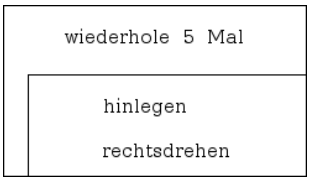
Nun übertragen wir dieses Wissen wieder in die Programmiersprache JAVA:

Zählschleife in JAVA:

```
for ( int z=1 ; z<=5 ; z=z+1) {
   Anweisungen;
}
```

- Die Zählschleife beginnt mit dem Schlüsselwort `for` .
- In runden Klammern musst du dann
  - zuerst eine **Zähl-Variable z deklarieren und sofort** mit einem Startwert **initialisieren**: `int z=1`
  - dann eine **Bedingung** formulieren, **unter welcher der Rumpf nochmal durchlaufen wird**: `z<=5`
  - zuletzt wird die **Schrittweite für das Erhöhen der Zähl-Variable** `z` angegeben: `z=z+1`
  - **Trennzeichen** ist hier ein **Strichpunkt**
- Die zu wiederholenden Anweisungen stehen dann innerhalb eines geschweiften Klammern-Paars.

Beispiel:



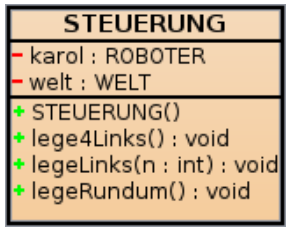

```
wiederhole 5 Mal          for ( int z=1 ; z<=5 ; z=z+1) {
   hinlegen                   this.Karol.Hinlegen();
   rechtsdrehen               this.Karol.RechtsDrehen();
*wiederhole               }
```

36. **JAVA-Karol – Zählschleife**

Öffne das BlueJ-Projekt JavaKarol und speichere es sofort unter dem Namen JavaKarol_Zaehlschleife.

a) *Erstelle wieder eine Klasse ROBOTERSTEUERUNG nach folgenden Vorgaben:*

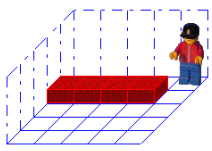
Als Welt wählst du die Datei welt-leer.kdw .



b) *Der Rumpf der Methode `legeRundum()` entspricht dabei der Aufgabe 32.*

c) *Schreibe eine weitere Methode `lege4Links()` , welche nebeneinander 4 Steine nach links legt.*

d) *Schreibe eine Methode `legeLinks(int n)` , welche dasselbe tut, aber die Anzahl der Steine als Übergabe-Parameter entgegen nimmt.*



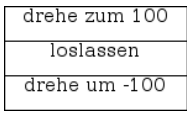
Teste die Methoden, indem du ein Objekt deiner ROBOTERSTEUERUNG erzeugst und die Methode durch Rechtsklick aufrufst.

37. Greif-Roboter – Projekt mit Zählschleife

Öffne das BlueJ-Projekt Greifroboter und speichere es sofort unter dem Namen Greifroboter_Zaehlschleife.

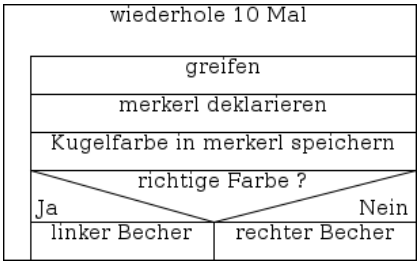
a) *Erstelle eine Klasse SPEZIALIST , welche von ROBOTER erbt.*

b) *Schreibe nun eine Methode* `linkerBecher()` *, welche den Greifarm zum linken Becher hin bewegt, dort die Kugel abwirft (wenn er eine hat) und den Greifarm dann wieder zurück zum Fließband bewegt.*



c) *Natürlich brauchen wir nun auch eine Methode* `rechterBecher() ...` *.*

d) *Schreibe nun eine Methode* `nimm10(String farbe)` *,*
welche nacheinander die ersten 10 Kugeln vom Fließband nimmt. Bei jeder Kugel soll zuerst die Farbe bestimmt und für den Moment des Methodenablaufs gespeichert werden.
Dazu brauchst du im Rumpf der Methode eine neue (lokale) Variable.
Anschließend soll die Kugel in den linken Becher geworfen werden, wenn sie die richtige Farbe (laut Übergabe-Parameter) hat, sonst wandert sie in den rechten Becher. Nutze für das Abwerfen der Kugeln die beiden Methoden aus b) und c) .



e) *Deklariere nun in der Klasse SPEZIALIST noch eine ganzzahlige Variable* `zaehler`*. Immer wenn in deiner Methode* `nimm10(...)` *eine Kugel in den linken Becher geworfen wird, dann soll der Wert diese Zählers um Eins erhöht werden.*

f) *Schreibe außerdem eine Methode* `nenneZaehler()` *, welche die Anzahl der Kugeln im Linken Becher zurück gibt.*

g) *Schreibe eine Methode* `zaehlerZuruecksetzen()` *, welche den Wert des Zählers wieder auf Null setzt.*

h) *Fertige nun eine ausführliche Klassenkarte der Klasse SPETIALIST.*

Teste die Methoden, indem du ein Objekt deiner ROBOTERSTEUERUNG erzeugst und die Methode durch Rechtsklick aufrufst.

Für besonders schlaue Köpfe

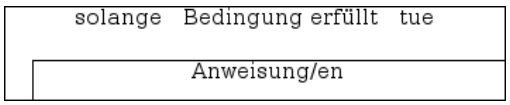
j) *Schreibe eine Methode* `zumFließband()` *, welche den Arm aus jeder beliebigen Position wieder zurück zum Fließband dreht.*

k) *Schaffst du das auch mit den Methoden* `zumLinkenBecher()` *und* `zumRechtenBecher()` *?*

Bedingte Wiederholung

Auch die bedingte Wiederholung kennst du bereits aus der 7. Klasse NuT-Informatik. Erinnere dich:

bedingte Wiederholung.



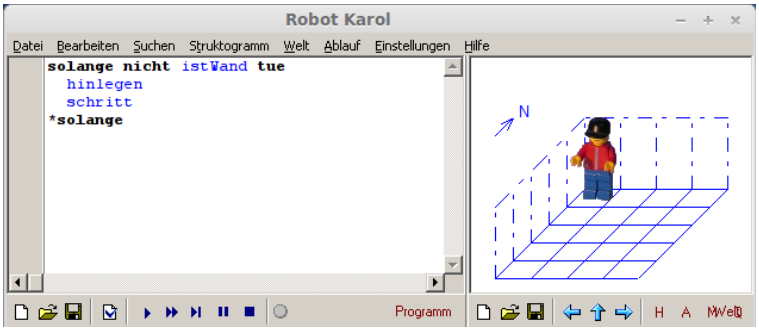
Ein Befehl oder auch eine Sequenz von Befehlen soll wiederholt werden solange eine gewisse Bedingung noch erfüllt ist.
Du gibst vor der Sequenz an, unter welcher Bedingung sie nochmal wiederholt werden soll.

Alles, was im Struktogramm eingerückt ist, wird solange wiederholt, bis die Bedingung nicht mehr erfüllt ist.

38. **Robot Karol – Erinnerungen 3**

a) *Öffne die Umgebung „Robot Karol“.*

b) *Schreibe folgendes Programm:*



c) *Führe das Programm durch Klick auf die Start-Taste aus.*

d) *Zeichne ein Struktogramm zu diesem Programm.*

e) *Wie müssten all diese Methodenaufrufe aus Objekt-orientierter Sicht in Punktnotation lauten? Schreibe die Antwort in deinem Hefter auf.*

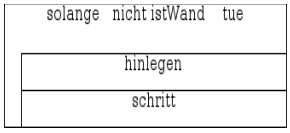
Nun übertragen wir dieses Wissen wieder in die Programmiersprache JAVA:

bedingte Wiederholung in JAVA:

```
while ( Bedingung ) {
    Anweisungen;
}
```

- Die Zählschleife beginnt mit dem Schlüsselwort `while` .
- In runden Klammern wird dann eine **Bedingung für das Durchlaufen** angegeben.
- Die zu wiederholenden Anweisungen stehen dann innerhalb eines geschweiften Klammern-Paars.

Beispiel:




```
solange nicht istWand tue
    hinlegen
    schritt
*solange
```

```
while ( ! karol.IstWand() ) {
    this.Karol.Hinlegen();
    this.Karol.Schritt();
}
```

39. **JAVA-Karol – bedingte Wiederholung**

a) *Öffne das BlueJ-Projekt JavaKarol und erstelle eine Klasse STEUERUNG mit den Referenz-Attributen* `welt` *und* `karol`*.*

b) *Schreibe eine Methode* `legeReihe()`*, welche in ihrem Rumpf den Code aus dem Beispiel „Robot Karol – Erinnerungen 3“ enthält.*
Übersetze deinen Code und teste die neue Methode.

c) *<u>Wiederholungs-Übung</u>*
Schreibe eine Methode `legeRunde()`*, welche von der Methode* `legeReihe()` *Gebrauch macht.*
Übersetze deinen Code und teste die neue Methode.

d) *Schreibe nun eine Methode* `legeRunde(...)`*, welche als Übergabe-Parameter eine ganze Zahl entgegen nimmt, die angibt, wie viele Runden gelegt werden sollen.*
Im Rumpf der Methode `legeRunde(...)` *soll die Methode* `legeRunde()` *verwendet werden.*
Übersetze deinen Code und teste die neue Methode.

Zusatzmaterial: Operatoren in JAVA

40. **Greif-Roboter – Projekt mit bedingter Wiederholung**

a) *Öffne das Projekt GreifRoboter und speichere es sofort unter dem Namen GreifRoboter_bedingte-Wiederholung.*

b) *Erstelle nun eine Klasse SPEZIALIST, welche von der Klasse Roboter erbt. Denke an einen Konstruktor, welcher den geerbten Superkonstruktor aufruft.*

c) *Schreibe in der Klasse SPEZIALIST eine Methode* `sucheNachBlau()`*, welche solange eine Kugel greift und weg wirft, bis eine blaue Kugel gegriffen wurde. Diese soll dann in den linken Becher geworfen werden und der Greifarm soll wieder zum Fließband zurück gedreht werden.*

d) *Schreibe eine weitere Methode* `sucheNach(...)`*, welche eine Farbe als Übergabe-Parameter entgegen nimmt. Ansonsten soll sich die Methode wie die aus Aufgabe c) verhalten.*

e) *Ändere die Methode aus d) folgendermaßen ab:*
Die Methode soll nun eine ganze Zahl als Antwort geben. Die Antwort soll angeben, wie viele Kugeln gegriffen wurden, bis die gewünschte Farbe erreicht wurde.
Tipp: *Dazu brauchst du einen lokalen Parameter (*`merker1`*) in der Methode, welcher bei jeder Kugel um Eins erhöht wird.*

f) *Zeichne ein Klassen-Diagramm, welches die Vererbung veranschaulicht, eine Klassen-Karte, welche deine Klasse SPEZIALIST darstellt sowie ein Struktogramm zur Methode aus Aufgabe e), beschrifte es mit deinem Namen und gib es beim Lehrer ab.*

Für Tüftler oder besonders Schnelle:

g) *Schreibe noch eine Methode* `sucheNach(... , ...)`*, welche neben dem Übergabe-Parameter für die Farbe noch einen weiteren ganzzahligen Übergabe-Parameter hat, welcher angibt, wie viele Kugeln der gewünschten Farbe gefunden werden sollen.*

Sicherung des bisher Gelernten

Beantworte folgende Fragen bevor du weiter machst:

1. Wie sieht das Struktogramm zu einer einfachen bedingten Anweisung bzw. zu einer bedingten Anweisung mit Alternative aus.
2. Wie wird eine bedingte Anweisung in JAVA formuliert?
3. Was ist ein lokaler Parameter?
   Wo wird er deklariert? Wo existiert er? Wann „stirbt“ er?
4. Wie sieht das Struktogramm zu einer Wiederholung mit fester Anzahl (Zählschleife) aus?
5. Wie wird eine Wiederholung mit fester Anzahl (Zählschleife) in JAVA formuliert?
   Welche Bedeutung haben insbesondere die drei Teile innerhalb der runden Klammern?
6. Wie sieht das Struktogramm zu einer bedingten Wiederholung aus?
7. Wie wird eine bedingte Wiederholung in JAVA formuliert?
8. Welche Werte kann eine Bedingung annehmen?

Zusatzmaterial: Test 3

Überladen von Methoden

- Es ist zulässig, z.B. mehrere Konstruktor-Methoden zu deklarieren:

  Beispiele:

```
public AMPEL() ...                                   und gleichzeitig
public AMPEL(String zustand) ...                     und gleichzeitig
public AMPEL(String zustand, boolean automatik) ...
```

- Diese müssen sich aber unbedingt
  in der Anzahl der Parameter oder
  deren Daten-Typen unterscheiden.

  (Sonst gibt es Fehler-Meldungen beim Übersetzen!)

- Hat man eine **Methode auf mehrere Arten implementiert**, so nennt man das ***Überladen*** dieser Methode.

- Im Rumpf eines Konstruktors kann natürlich jederzeit ein anderer Konstruktor aufgerufen werden um die Schreibarbeit zu reduzieren.
  Hierfür wird die Methode `this()` verwendet, um den Konstruktor ohne Parameter aufzurufen oder auch `this(zustand)` für den Konstruktor mit einem String als Übergabe-Parameter.

  Beispiel:

```
public AMPEL() {
    this.rot = new LAMPE();
    this.gruen = new LAMPE();
}
public AMPEL(String zustand) {
    this();
    if (zustand ==  „gruen“) {
        this.gruen();
    else if (zustand == „rot) {
        this.rot();
    }
}
public AMPEL(String zustand, boolean automatik) {
    this(zustand);
    if (automatik == false) {
        this.tickerAbmelden();
    }
}
```

41. **Engine-Alpha – Ampel Teil 3**

Öffne dein Projekt Ampel-02 und speichere es unter dem Namen Ampel-03.
Erstelle weitere zwei Konstruktoren wie oben in den Beispielen.

Teste deine Konstruktoren, ob sie wie gewollt funktionieren.

Arrays

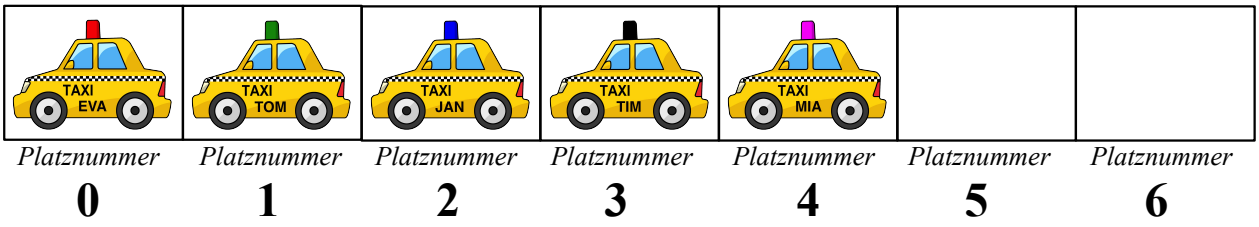
Gelegentlich braucht man für ein Programm **sehr viele Attribute desselben Daten-Typs**.

Beispiel:

In der HighScore-Liste eines Spiels werden von den 100 besten Spielern die Namen und Punktestände gespeichert.
Dazu bräuchte man 100 Attribute vom Typ `String` *und weitere 100 Attribute vom Typ* `int`.

In solchen Fällen wird man nicht 200 einzelne Attribute erst deklarieren und anschließend auch noch initialisieren … auch nicht mit copy & paste.

Hierfür gibt es eine Daten-Struktur in der (beliebig) viele Attribute desselben Daten-Typs gespeichert werden können. Diese Daten-Struktur nennt man ***Array*** oder auch Feld. Man kann es mit einem **Taxi-Stand vor dem Bahnhof** vergleichen:



- Es ist **eine gewisse Anzahl an Plätzen** vorhanden (hier 7 Stück).
- Auf so einem Platz darf **nur** ein **Taxi**, sonst nichts anderes stehen.
- **Es müssen nicht alle Plätze benutzt werden**, einige können auch leer sein.
- Aber mehr als die **maximale Anzahl** an Taxen kann dort nicht stehen.
- **Die Plätze werden – beginnend mit Null – durchnummeriert**.

Man kann Arrays von jedem nur denkbaren Daten-Typ erstellen – das sind dann im übertragenen Sinne „Parkplätze“ für Objekte dieses einen Daten-Typs.

arrays in JAVA

Array deklarieren:

```
int z;
```
eine ganze Zahl

```
int[ ] z;
```
*ein **Array von ganzen Zahlen***
*Array: **eckige Klammern** hinter dem Daten-Typ*

Array initialisieren:

```
z = new int[7];
```
*int-Array mit **7 Plätzen** **Nummern von 0 bis 6 !!!***

Zugriff auf Array-Plätze:

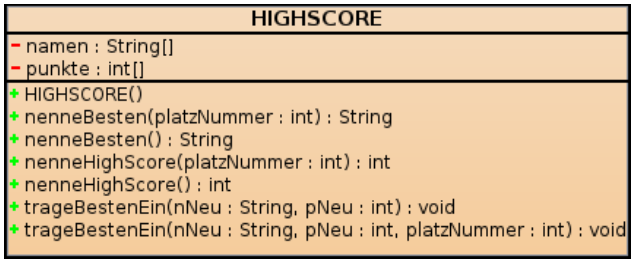
```
z[3] = 47;
```
dem vierten Platz den Wert 47 zuweisen

```
z[3] + z[5];
```
analog: x + y (bei 2 einzelnen int-Variablen)

42. Arrays – praktischer Einstieg

1. *Öffne Bluej und lege ein neues Projekt mit dem Namen HighScore an.*

2. *Erstelle darin eine **Klasse HIGHSCORE**.*
   *Die Klasse soll als **Referenz-Attribute** ein `int`-Array namens `punkte` und ein `String`-Array namens `namen` haben.*



3. *Ein erster **Konstruktor** soll keine Übergabe-Parameter haben.*
   Im Rumpf sind mehrere Initialisierungen vorzunehmen:
   - dem Referenz-Attribut `punkte` ein neues `int`-Array der Länge `10` übergeben
   - dem Referenz-Attribut `namen` ein neues `String`-Array der Länge `10` übergeben
   - in das String-Array einige Namen eintragen
   - in das int-Array einige Punktestände eintragen

Name	Punkte
Hans	12345
Susi	9876
Peter	8765

4. *Übersetze die Klasse, bis keine Fehler mehr auftreten und erzeuge ein HIGHSCORE-Objekt. Doppel-klicke auf die rote Objekt-Karte und begutachte das Objekt im **Objekt-Inspektor**. Folge durch weiteren Doppel-Klick der Referenz auf die Arrays.*
   *→ **Welche Werte sind in den Zahlen-Plätzen von 0 bis 9?***
   *→ **Was ist in den Text-Plätzen von 0 bis 9?***

5. *Doppel-klicke auf die Klassen-Karte und schreibe eine Methode **`nenneHighScore()`**, welche den Wert des punkte-Arrays mit Index 0 zurück gibt.*
   *→ **Welchen Rückgabe-Daten-Typ hat diese Methode?***
   Probiere an einem Objekt aus, ob die Methode wirklich `12345` als Antwort liefert.

6. *Schreibe auch eine Methode **`nenneBesten()`**, welche den Inhalt des Text-Platzes `0` des String-Arrays zurück gibt.*
   *→ **Welchen Rückgabe-Daten-Typ hat diese Methode?***
   Probiere an einem Objekt aus, ob die Methode wirklich `null` als Antwort liefert.

7. *Schreibe eine weitere Methode **`trageBestenEin(...)`**, welche zwei Übergabe-Parameter bekommt:*
   - einen `String`-Parameter für den Namen des neuen Besten
   - einen `int`-Parameter für den Punktestand des neuen Besten.
   *→ **Ist diese Methode verändernd oder sondierend?***
   Im Rumpf der Methode soll erst der neue Name dem String-Array auf Platz `0` zugewiesen werden und anschließend der neue Punktestand dem int-Array auf Platz `0`.
   Erzeuge ein Objekt und teste die neuen Methoden.
   *→ **Siehst du die neuen Werte im Objekt-Inspektor?***
   *→ **Geben die beiden ersten Methoden nun auch diese neuen Werte zurück?***

8. → *Schreibe eine Methode `nenneNamen(...)`, welche als Übergabe-Parameter den Index entgegen nimmt, von welcher Stelle im namen-Array der Name genannt werden soll.*
Ist diese Methode sondierend oder verändernd?
Von welchem Datentyp ist die Antwort?
Prüfe die ordnungsgemäße Funktion dieser Methode im Objekt-Inspektor.

→ *Schreibe analog eine Methode nennePunkte(...) …*
Von welchem Datentyp ist die Antwort?
Prüfe die ordnungsgemäße Funktion dieser Methode im Objekt-Inspektor.

9. → *Schreibe zuerst eine Methode `tragePunkteEin(...)`, welche als ersten Übergabe-Parameter den Punktestand und als zweiten Übergabe-Parameter den Index entgegen nimmt, an dem der neue Punktestand eingetragen werden soll.*
Prüfe die ordnungsgemäße Funktion dieser Methode im Objekt-Inspektor.

→ *Schreibe nun analog eine Methode `trageNameEin(...)`, welche als ersten Übergabe-Parameter den Namen und als zweiten Übergabe-Parameter den Index entgegen nimmt, an welcher Stelle der neue Name eingetragen werden soll.*
Prüfe die ordnungsgemäße Funktion dieser Methode im Objekt-Inspektor.

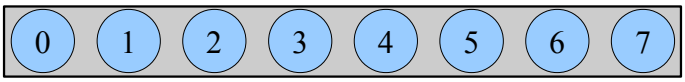
→ *Schreibe nun eine Methode `trageEin(...)`, welche als ersten Übergabe-Parameter den Namen, als zweiten Übergabe-Parameter den Punktestand und als dritten Übergabe-Parameter den Index entgegen nimmt, an welcher stelle in den den beiden Array eingetragen werden soll.*
Diese Methode kann sich der beiden vorherigen Methoden bedienen, um nicht denselben Code noch einmal schreiben zu müssen.
Prüfe die ordnungsgemäße Funktion dieser Methode im Objekt-Inspektor.

10. ***Nur für ganz Harte …***
Schreibe eine weitere Methode `trageEin(...)`, welche nur den Namen und den Punktestand als Übergabe-Parameter erwartet und einen `boolean` als Antwort zurück gibt. Die Methode soll aufgrund des übergebenen Punktestands selbst die richtige Position in den Arrays herausfinden und alle dahinter liegenden Spieler um einen Platz nach hinten verschieben. Der bisher Letzte wird dadurch von der Liste entfernt.
Wird der Spieler eingetragen, so gibt die Methode `true` zurück.
Sollte die Punktezahl nicht ausreichen, um eingetragen zu werden, so soll nichts geschehen, außer dass die Methode `false` zurück gibt.

Bisher war der Daten-Typ deiner Arrays einfach, es gab keine Referenzen.

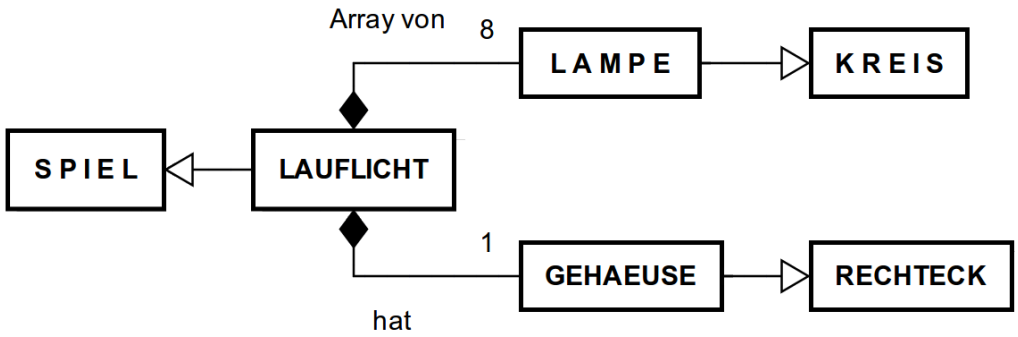
Das nächste Beispiel soll dir zeigen, worauf du noch achten musst, wenn es sich um Referenz-Typen handelt …

Projekt Lauflicht



Du sollst nun ein Lauflicht programmieren, wie du es z.B. aus einer Disco kennst. Unser Lauflicht soll aus 8 einzelnen Lampen bestehen, wie wir sie schon im Ampel-Projekt erstellt haben. Wir bauen hier also eine Art Ampel mit 8 Lampen.

Deshalb sieht das Klassen-Diagramm auch fast genauso aus wie bei dem Ampel-Projekt



Der Unterschied besteht darin, dass wir die 8 Lampen diesmal nicht als 8 einzelne Referenz-Attribute erstellen wollen. Wir verwenden diesmal ein Array von Lampen.

Wichtig ist dass du dir darüber klar wirst, dass das Array wie ein Zug mit Waggons funktioniert. Die 8 Plätze des Arrays sind die 8 Waggons des Zugs. Noch sind die Waggons leer. Wir müssen erst in jeden Waggon eine Lampe legen. Diese Lampen müssen natürlich vorher erst erzeugt werden.

Das Array und eine Zähl-Schleife werden uns dabei helfen, nicht 8 Mal quasi denselben Code schreiben zu müssen.

1) Das Grafik-Fenster ist 800 Pixel breit, so dass genau 8 Lampen vom Radius 50 hineinpassen.
Zeichne nun wieder zuerst in einem geeigneten Koordinaten-System dein Lauflicht.

2) Gib nun für das Gehäuse an, wie breit und wie hoch es in Pixeln ist und welche Pixel-Koordinaten sein Mittelpunkt hat.

Gehäuse	Breite = _____	Höhe = _____
Mittelpunkt	x = _____	y = _____

3) Gib für jede einzelne Lampe die Pixel-Koordinaten ihres Mittelpunkts und den Radius an.

Lampen [0]	x = _____	y = _____	r = _____
Lampen [1]	x = _____	y = _____	
Lampen [2]	x = _____	y = _____	
Lampen [3]	x = _____	y = _____	
Lampen [4]	x = _____	y = _____	
Lampen [5]	x = _____	y = _____	
Lampen [6]	x = _____	y = _____	
Lampen [7]	x = _____	y = _____	

Was fällt dir dabei auf?

4)
→ Öffne dein *Ampel-Projekt* in BlueJ und speichere es sofort unter dem Namen *Lauflicht*. Lösche die Klasse AMPEL.
→ Sieh in der Klasse LAMPE nach, ob der Radius dem Wert von oben entspricht. Passe den Wert gegebenenfalls an.
→ Sieh in der Klasse GEHAEUSE nach, ob Breite, Höhe und Mittelpunkt-Koordinaten den Werten von oben entsprechen und passe ggf. die Werte an.

5) → Schreibe eine neue Klasse LAUFLICHT, welche von SPIEL erbt.

→ Deklariere ein Referenz-Attribut vom Typ GEHAEUSE sowie ein Array von LAMPEN.

→ Schreibe nun den Konstruktor. Er braucht keine Übergabe-Parameter. Erzeuge das GEHAEUSE-Objekt und das LAMPE-Array der Länge 8.

Erzeuge ein Objekt deines Lauflichts und betrachte im Objekt-Inspektor das Array näher. ***Wo sind deine LAMPEN-Objekte?***

→ Folgender Code erzeugt mit Hilfe einer Zähl-Schleife deine 8 LAMPEN-Objekte am richtigen Ort:

```
for ( int i=0 ; i<8 ; i=i+1 )
{
   this.lampen[i] = new LAMPE ( "gelb" , 50+i*100 , 300 );
}
```

Versuche die Wirkung dieser 2 Zeilen zu verstehen.

Erzeuge ein Objekt deines Lauflichts. ***Sieht alles aus wie gewollt?***

6) Nun wollen wir für das Lauflicht eigene Lichteffekte programmieren.

→ Zunächst stört uns jedoch das Fenster mit dem tick … tack … .
Überschreibe in der Klasse LAUFLICHT die Methode `tick()` mit einem leeren Rumpf, damit es zu ticken aufhört und das weiße Fenster nicht wieder erscheint.

→ Schreibe eine Methode `aus()`, die mithilfe einer Zähl-Schleife alle Lampen des Arrays ausschaltet.
Schreibe analog eine Methode `an()`, die alle Lampen wieder anschaltet.
Schreibe auch eine Methode `setzeFarbe(String farbeNeu)`, welche die Farbe aller Lampen ändert.

→ Dein LAUFLICHT hat von der Klasse SPIEL die Methode `warte(int millisec)` geerbt. Sie veranlasst, dass das Programm etwas wartet bevor es den nächsten Befehl ausführt.
Schreibe eine Methode `laufe_nach_rechts()`, die erst alle Lampen ausschaltet. Anschließend soll mithilfe einer Zähl-Schleife nacheinander jede Lampe einschaltet, etwas gewartet und die Lampe dann wieder ausschaltet werden.
Schreibe analog eine Methode `laufe_nach_links()`.

7) **Sei kreativ! Erfinde selbst neue Lichteffekte.**
Es ist auch eindrucksvoll, in einer neuen Methode nacheinander unter Verwendung von `warte(...)` andere bereits selbst geschriebene Methode aufzurufen um länger andauernde und komplexe Lichtmuster zu erzeugen.