Wiederholung mit Schleifen

Neben der ***Anweisung*** (z.B. Zuweisung mit dem =-Operator oder Aufruf einer Methode auf einem Objekt mit der Punktnotation) und der ***bedingten Anweisung bzw. Fallunterscheidung*** bilden sie mit der ***Wiederholung mit Schleifen*** die drei ***Kontrollstrukturen*** bei Programmiersprachen. Alle Programme, die mit einer imperativen Programmiersprache (wie Java, C, C++, C#, Pascal, Assembler, Visual Basic und viele mehr…) geschrieben wurden, bestehen ausschließlich aus diesen drei Bausteinen.

Wiederholung mit fester Anzahl mit der Zählschleife

Für die Kontrollstruktur der Zählschleife wird durch das Schlüsselwort ist ***for*** eingeleitet, gefolgt von einem ***runden Klammernpaar*** und einem ***geschweiften Klammernpaar***.

```
for( int i = startwert; i < maximalwert; i = i + 1 ){
        // ToDo – hier deine Befehle
}
```

In den runden Klammern stehen 3 Informationen (**mit Strichpunkt getrennt**):

Zählvariable und Startwert

int i = 0	Meine Zählvariable heißt i und startet mit dem Wert 0.	(vorne im Array)
int i = 10	Meine Zählvariable heißt i und startet mit dem Wert 10.	(hinten im Array)

Bedingung für das Ausführen der Befehle

Die Befehle in den geschweiften Klammern werden nur ausgeführt, wenn diese Bedingung ***WAHR*** ist.

i <= 5	„Platznummer“ von 0 bis 5	(nur sinnvoll beim Erhöhen der Zählvariablen)
i >= 0	„Platznummer“ von 10 bis 0	(nur sinnvoll beim Verringern der Zählvariablen)

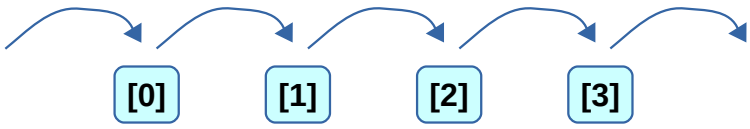
Wert der Zählvariablen ändern

Wenn die Befehle in den geschweiften Klammern einmal ausgeführt wurden, dann muss der Wert der Zählvariablen verändert werden.

i = i + 1	Wert um 1 erhöhen	(Array von links nach rechts durchlaufen)
i = i – 2	Wert um 2 verringern	(von rechts nach links; nur jedes zweite Element)

Iteration von Arrays mit der Zählschleife

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



Das kommt recht häufig vor, z.B. wenn du alle gespeicherten Kreise anders färben möchtest, d.h. auf alle Objekte die Methode `setzeFarbe(farbwert)` aufrufen möchtest.

Iteriere ein Array, indem du die bekannte ***for-Schleife*** verwendest.

```
for( int i = 0; i < kreise.length; i = i + 1 ){
        kreise[i].setzeFarbe("blau");
}
```

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. Dieser Wert kann als Maximalwert bei der for-Schleife angegeben werden. Dies klappt aber nur, wenn ***alle „Plätze" im Array*** befüllt wurden. Ansonsten nullPointerException

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

Aufgabe 2:

Erweitere die Klasse HIGHSCORE im Projekt Array um eine weitere Methode `nenneAlleRaenge() : void`*, die mithilfe einer Schleife alle Spieler samt ihrer Punktzahl ausgibt.*

(Hinweis: zurückgeben ≠ ausgeben; Zurückgeben: über den Rückgabedatentyp der Methode; Ausgeben: auf die Konsole schreiben mit ***System.out.println(...)****)*

Aufgabe am Projekt Moorhuhn:

a) *Öffne dein Projekt Moorhuhn. Nutze beim Erstellen der Moorhühner, bei klickReagieren(…) und beim Neustart des Spiel jeweils eine Schleife. Überarbeite deinen Code bis das Spiel wieder fehlerfrei funktioniert.*

b) *Dem User soll bei Moorhuhn die Möglichkeit gegeben werden, dass die Anzahl an Moorhühnern selber zu bestimmen. Gib dazu dem Konstruktor von Moorhuhn ein Übergabeparameter* `int zahl`*. Initialisiere das Array mit dem*

Übergabeparameter.

Sofern du nun bei allen Schleifen und beim Stoppen des Spiels die Länge des Arrays genutzt hast, passt sich das Spiel entsprechend des übergebenen Werts an. Teste dein Spiel!

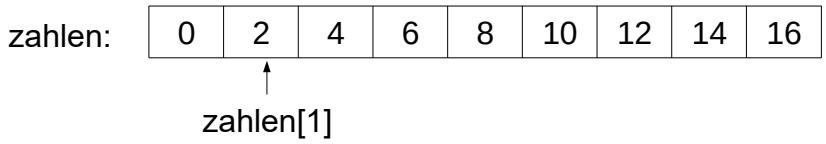
Nutzung der Zählvariable

Im Gegensatz zu der vorangegangenen Aufgabe erzeugt man in der Regel die Objekte nicht an zufälligen Positionen. Aber jedes Objekt einzeln zu erstellen und zu „konfigurieren“ ist – wie bereits bekannt – sehr mühsam. Deshalb kann es hier sinnvoll sein, ***die Zählvariable geschickt zu verwenden***, um sich viel Schreibarbeit zu sparen.

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 = i + 1 ){
        zahlen[i] = i*2;
}
```

Hier ist das Array zahlen mit den zugehörigen Belegungen veranschaulicht:



Aufgabe 3:

a) *Erstelle im Projekt Array eine neue Klasse FIBONACCI, welche im Konstruktor ein Zahlenarray einer übergebenen Länge initialisiert und mit den Fibonacci-Zahlen befüllt wird. Die ersten beiden Fibonacci-Zahlen sind 0 und 1. Jede weitere Zahl wird durch Addition der beiden vorangegangen Zahlen gebildet. Somit ergibt sich die Zahlenfolge: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …*

b) *Öffne im Projekt Array die Klasse MEINSPIEL. Hier sollen im Konstruktor alle 50 Plattformen mithilfe einer Schleife erzeugt werden. Die Plattformen sollen jeweils 6m auseinander liegen und die Plattformen sollen abwechselnd auf der Höhe -5m und -3m erstellt werden. Die erste Plattform soll bei -10m in x-Richtung erzeugt werden. Vergiss nicht die Plattformen passiv zu machen. Nutze hierfür geschickt die Laufvariable.*

Die Koordinaten der ersten Plattformen sind: (-10,-5),(-4,-3),(2,-5),(8,-3),(14,-5),…

> ***Tipp:*** *Schau dir für das abwechselnde Oben-Unten-Erzeugen die Modulo-Rechnung etwas später im Skript bei „Exkurs: Kurzschreibweisen von zum Verändern von Variablen“ an.*

Exkurs: lokale Variablen

Bisher: Variablen, ***die außerhalb von Methoden*** angelegt werden, sind ***Attribute***.

Neu: Variablen kann man auch ***innerhalb von Methoden*** anlegen. Man nennt sie ***lokale Variablen***. Sie sind solange „nutzbar“, bis der dazugehörige Block (ein Block beginnt und endet mit geschweiften Klammern{ }) endet.

Beispiele:

```
public void methode1(double d){
    d = 2.7;
    String s;
    s = “Hallo“;
    int zahl;
    zahl = 5;
    …
}
```

Die lokalen Variablen s, d und zahl wurden in der Methode methode1() erstellt und sind bis zum Ende der Methode gültig!

```
public void methode2(){
    for(int i = 0; i<10; i=i+1){
    …
    }
    for(int i = 5; i>=0; i=i-1){
    …
    }
}
```

Die Variable i würde für den Block der for-Schleife erzeugt und ist bis hier nutzbar.

Da die Variable i von der vorherigen Schleife hier bereits gelöscht ist, darf ich als Namen für die Zählvariable der zweiten Schleife wieder i verwenden.

Exkurs: Kurzschreibweisen von zum Verändern von Variablen

Mit Variablen muss häufig und viel gerechnet werden. Daher gibt es in Java Kurzschreibweisen, um schneller programmieren zu können.

Im folgenden Beispiel wird die Variable `int z` um den Wert `a` verändert:

	Standard	Kurzversion	Veränderung um genau 1
Addition	z = z + a	z += a	z++
Subtraktion	z = z - a	z -= a	z--
Multiplikation	z = z * a	z *= a	
Division	z = z / a	z /= a	
Modulo	z = z % a	z %= a	

In der Informatik gibt es fünf Grundrechenarten. Die fünfte Grundrechenart ist die Modulo-Rechnung (=Divisionsrest bei einer Division mit Rest).

Beispiele: 17 % 5 = 2 13 % 10 = 3 66 % 3 = 0

Bedingte Wiederholung mit der while-Schleife

Nicht immer weiß man vorher (also beim Programmieren), wie oft eine Wiederholung durchlaufen werden soll. Sondern man möchte bestimmten Code ***solange wiederholen, bis ein bestimmtes Ereignis eintritt.***

Beispiel Methode tick(): Sobald der Ticker gestartet wurde, wird die Methode tick() solange mit einem gewissen Zeitabstand aufgerufen, bis der Ticker wieder gestoppt wird.

```
while( Bedingung ){
        // ToDo – hier deine Befehle
}
```

Die Bedingung bei der while-Schleife funktioniert genauso wie bei der If-Anweisung.
Solange die ***Bedingung zu true ausgewertet*** wird, wird weiter der Code in der while-Schleife wiederholt.

Aufgabe 4:

Die while-Schleife ist ein mächtigeres Werkzeug als die for-Schleife (=> manche Algorithmen lassen sich nur mit der while-Schleife und nicht mit der for-Schleife lösen). Schreibe eine while-Schleife, die wie eine for-Schleife funktioniert.

Eine besondere Schleife ist die ***while(true){ … }*** Schleife. Da hier die Bedingung sowieso immer true ist, läuft diese Schleife so lange, bis das Programm geschlossen wird.

Aufgabe 5: Iterative Nullstellensuche

Öffne im Projekt Array die Klasse NULLSTELLENSUCHE und vervollständige die Methode `sucheNullstelleImIntervall(double x1, double x2) : double`.

Lege dir eine lokale Variable `mitte` *an. Wiederhole die folgenden Schritte solange bis der Betrag (Math.abs(…)) von* `f(x1)` *den Abbruchsfall von epsilon unterschreitet:*

- Berechne die neue Mitte des Intervalls zwischen `x1` *und* `x2`.

- Überprüfe anhand des Vorzeichenwechsels, ob die Nullstelle zwischen den Intervall `x1` *und* `mitte` *oder zwischen* `mitte` *und* `x2` *liegt.*

- Setze für den nächsten Schleifendurchlauf das Intervall `x1` *und* `x2` *auf die beiden Werte, die beim vorherigen Schritte die Nullstelle enthalten haben.*

Gib nach der Schleife den Wert `x1` *als Antwort der Methode zurück.*

Iteration von Arrays mit der ForEach-Schleife (Alternative zur Zählschleife)

Die ForEach-Schleife ist ganz ähnlich zu der Zählschleife. Die Besonderheit ist, dass es hier nach außen keine sichtbare Zählvariable genutzt wird. Bei jeder Iteration bekommt man hier direkt den Wert aus dem Array geliefert. Somit ist die ForEach-Schleife ***nur bei Objekt-Arrays sinnvoll einsetzbar***, weil man aufgrund der fehlenden Zählvariable die neu berechnete Werte nicht im Array zurückspeichern kann.

```
for ( Datentyp name : array ){
    //name beinhaltet nacheinander die
    //Werte des Arrays
}
```

Beispiele:

```
private KREIS[] kreise;
kreise = new KREIS[5];
…
for( KREIS k : kreise ){
    k.setzeFarbe("rot");
}
```

```
private String[] namen;
namen = new String[9];
…
for( String s : namen ){
    System.out.println(s);
}
```