Staatliches Gymnasium
"Karl Theodor Liebe "
Gera

Fachbereich Informatik

Informatik

Strukturierte Datentypen

F. Möckel

24. März 2020

Inhaltsverzeichnis

1 Das Array

1.1 Einführung Array (Feld)

Bis jetzt wurde nur mit einfachen Datentypen programmiert. Nun soll ein Simulationsprogramm für das Lottospie 6 aus 49 erstellt werden.
Dies bedeutet, das 6 Zufallszahlen aus einem Bereich von 1 bis 49 ermittelt werden sollen.
Bis jetzt hätte man sechs Variablen mit einem Zufallswert belegt.

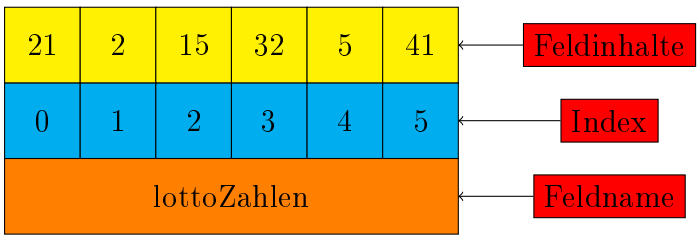
zahl1	Zufallszahl
zahl2	Zufallszahl
zahl3	Zufallszahl
zahl4	Zufallszahl
zahl5	Zufallszahl
zahl6	Zufallszahl

Zum Glück heißt das Spiel nicht 255 aus 6549!

Im Quelltext würde die so aussehen:

```
1 int lottoZahl1, lottoZahl2, lottoZahl3,
     lottoZahl4, lottoZahl5, lottoZahl6;
2
3 lottoZahl1 = (int) random(1,50);
4 lottoZahl2 = (int) random(1,50);
5 lottoZahl3 = (int) random(1,50);
6 lottoZahl4 = (int) random(1,50);
7 lottoZahl5 = (int) random(1,50);
8 lottoZahl6 = (int) random(1,50);
```

Dies ist natürlich recht umständlich. Günstiger wäre es wenn man diese zusammengehörenden Zahlen unter einen Namen speichern könnte.



Dies wird als Reihung oder Array bezeichnet.

Unter einer Reihung (Feld engl. Array) versteht man die Zusammenfassung mehrerer Variablen des gleichen Typs (z.B. INTEGER, CHAR) unter einem Namen. Ein einzelnes Feldelement kann über seinen Index aufgerufen werden, der den Platz des Elementes im Feld angibt.

Die Länge eines Arrays ist die Anzahl seiner Elemente. Die Elemente werden durch Indizes bezeichnet, die ganze Zahlen im Bereich 0 bis Arraylänge – 1 sind.

Das Deklarieren eines Arrays erfolgt in zwei Schritten. Als erstes wird dem Compiler durch die eckigen Klammern nach dem Datentyp mitgeteilt, dass es sich um eine Referenzvariable handelt, mit der später ein Feld erzeugt werden soll.

```
1 int[] lottoZahlen; // Arrayvariable deklarieren
```

Bis jetzt steht noch nicht fest, wie lang das Feld ist und womit die einzelnen Elemente gefüllt werden.

Im Arbeitsspeicher wurde bis jetzt ein Speicherplatz mit dem Namen `lottoZahlen` erstellt. Da aber die Länge noch fehlt ist diese Speicherzelle noch leer. Processing/Java setzt daher eine Markierung namens **NULL** ein.

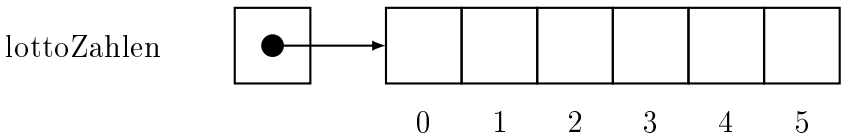
In der grafischen Notation sieht das für folgende zwei Beispiele folgendermaßen aus.

```
1 int   a = 45;
2 int[] lottoZahlen;
```

Die Festlegung der Länge des Feldes erfolgt in folgender Form.

```
1 lottoZahlen = new int [6]; // Array mit 6 Elementen
```

Daraus ergibt sich die folgende grafische Notation.



Die allgemeine Speicherbelegung würde wie folgt aussehen.

Arbeitsspeicher

symbolische Adresse	Adresse im Speicher	Inhalt der Speicherzelle	Typ des Inhalts
	⋮	⋮	
a	8135	45	Integer
	⋮	⋮	
lottoZahlen	2725	4376	Referenz
	⋮	⋮	
	4376		
	⋮	⋮	

Häufig benutzt man beide Vereinbarungsanweisungen in einem Schritt:

```
int[] lottoZahlen = new int[6];
```

Die Länge des Array kann mit

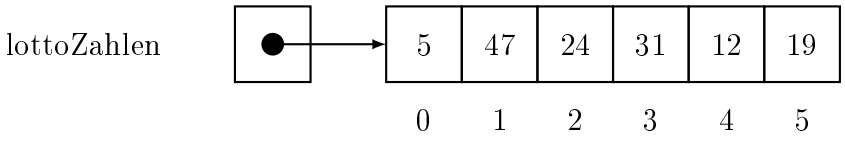
```
println(lottoZahlen.length);
```

ermittelt werden.

Das Programm könnte jetzt folgendermaßen aussehen.

```
int[] lottoZahlen = new int[6];
lottoZahlen[0] = (int) random(1,50);
lottoZahlen[1] = (int) random(1,50);
lottoZahlen[2] = (int) random(1,50);
lottoZahlen[3] = (int) random(1,50);
lottoZahlen[4] = (int) random(1,50);
lottoZahlen[5] = (int) random(1,50);
```

Die grafische Notation könnte dann folgendes Aussehen besitzen.



Dies ist natürlich ein ganz schlechter Programmierstil. Besser sind die folgenden Programmzeilen.

```
int[] lottoZahlen = new int[6];

for (int i = 0; i < 6; i++){
  lottoZahlen[0] = (int) random(1,50);
}
```

Die Ausgabe kann in Processing wie folgt aussehen.

```
println(lottoZahlen);
print(lottoZahlen);
```

Dies funktioniert nur in Processing!
Deshalb folgendes angewöhnen.

```
int[] lottoZahlen = new int[6];

for (int i = 0; i < 6; i++){
  lottoZahlen[i] = (int) random(1,50);
}
for (int i = 0; i < 6; i++){
  print("Zahl " + i + ": " + lottoZahlen[i]);
}
```

Testen Sie die obigen Codebeispiele und führen Sie Übung 1 und 2 aus.

Übung 1:
Erstellen Sie ein int-Array namens feld1 mit den Zahlen 5, 45, -10, 20 (in dieser Reihenfolge). Addieren Sie die ersten zwei Zahlen und legen Sie das Ergebnis in einer neuen int-Variablen a ab. Addieren Sie die letzten zwei Zahlen und legen Sie das Ergebnis einer neuen int-Variablen b ab.
Printen sie a und b auf die Konsole und überprüfen Sie die Werte.

Übung 2:
Erstellen Sie einen int-Array namens feld2 mit den Zahlen 2, 4, 6 (in dieser Reihenfolge). Multiplizieren Sie die ersten zwei Zahlen und legen Sie sie das Ergebnis in der dritten Zelle des Arrays ab. Sie sollten also 2, 4, 8 in feld2 haben. Schreiben Sie feld2 mit println() auf die Konsole, um dies zu prüfen.

Führen Sie die folgenden Beispiele aus

Beispiele:

Animation vieler Bälle
Jetzt ein praktisches Beispiel: Wir lassen drei Bälle über den Bildschirm fliegen. Dazu verwenden wir einen Array für die x-Koordinate der drei Bälle. Zunächst mal ohne Schleifen.

```
int[] xarray = new int[3]; // erzeugen

void setup() {
  // initialisieren
  xarray[0] = 0;
  xarray[1] = 20;
  xarray[2] = 40;
}

void draw() {
  background(255);

  // Werte verwenden
  ellipse(xarray[0], 50, 10, 10);
  ellipse(xarray[1], 50, 10, 10);
  ellipse(xarray[2], 50, 10, 10);

  // Werte hochzählen
  xarray[0]++;
  xarray[1]++;
  xarray[2]++;
}
```

Sie haben jetzt drei Bälle und ein Array mit drei Elementen. Sie hätten genauso schreiben können:

```
int x1;
int x2;
int x3;

void setup() {
  x1 = 0;
  x2 = 20;
  x3 = 40;
}

void draw() {
  background(255);
  ellipse(x1, 50, 10, 10);
  ellipse(x2, 50, 10, 10);
  ellipse(x3, 50, 10, 10);
  x1++;
  x2++;
  x3++;
}
```

Der Vorteil der Array-Lösung? Sie können die einzelnen Elemente mit Hilfe einer Schleife durchlaufen. Im Ball-Beispiel von oben kann man zwei Schleifen einfügen. In der ersten Schleife wird die x-Startposition mit i * 20 berechnet. In drei Durchläufen ergibt sich jeweils:
0 * 20,
1 * 20 und
2 * 20,
also 0, 20, 40.

```
int[] xarray = new int[3];

void setup() {
  for (int i = 0; i < 3; i++) {
    xarray[i] = i * 20; // berechne x-
       Startposition
  }
}

void draw() {
  background(255);
  for (int i = 0; i < 3; i++) {
    ellipse(xarray[i], 50, 10, 10); //
       zeichnen
    xarray[i]++; // hochzählen
  }
}
```

Diese Lösung ist nicht nur kürzer, sondern auch mächtiger. Sie können nämlich die Anzahl der Bälle erhöhen, z.B. auf 5. Dazu müssen Sie an drei Stellen im Code die 3 durch 5 ersetzen.

1.2 Aufgabenabschnitt 1

Aufgabe 1

In den folgenden Aufgaben sollte immer eine For-Schleife zum Einsatz kommen.
Erstellen Sie ein Array für 10 ganze Zahlen und befüllen Sie die Elemente mit 1, 2, ..., 10.
Ihr Programm soll so allgemein geschrieben sein, dass Sie auch Arrays mit 20 oder 100 Elementen befüllen können.
Varianten: Befüllen Sie das Array mit

- -10, -9, ..., -2, -1
- 2, 4, 6, ..., 20
- 100, 110, 120, ..., 190

Tipp
Verwenden Sie length, um Ihre Schleife unabhängig von der konkreten Länge des Arrays zu machen. Falls Ihr Array feld heißt, verwenden Sie feld.length für die Länge.

Aufgabe 2

Befüllen Sie ein Array der Länge 10 mit den Fibonacci-Zahlen: 1, 1, 2, 3, 5, 8, 13, ...
Beachten Sie, dass diese Variante etwas anders funktioniert (einfacher!) als die Aufgabe, mit Hilfe einer Schleife Fibonacci-Zahlen zu erzeugen. Hier können Sie auf die Vorgängerzahlen im Array zugreifen.
Besetzen Sie die ersten zwei Elemente im Array direkt mit je 1 und starten Sie in der Schleife ab dem dritten Element (Index 2).

Aufgabe 3

Befüllen Sie zwei Arrays gleichzeitig in einer Schleife. Beide Arrays haben Länge 10. Das eine Array wird mit 1, 2, 3, ... befüllt, das zweite Array mit -10, -9, -8, ...

Aufgabe 4

Verändern Sie das Array dahingehend, dass jedes Element verdoppelt wird.

```
1 int[] feld1  = { 2, 1, 3, 5 };
2
3 // Hier Code schreiben
4
5 println(feld1);
```

Sie sollten das hier auf der Konsole sehen:

```
[0] 4
[1] 2
[2] 6
[3] 10
```

Aufgabe 5

Berechnen Sie die Summe von allen Elementen des Array und geben Sie diese auf der Konsole aus. Der Code sollte ohne weitere Änderungen funktionieren, wenn Sie Zahlen zu `feld` hinzufügen oder entfernen.

```
1 int[] feld = { 22, -10, 8, 10 };
2
3 // Hier Code schreiben
```

Sie sollten das hier auf der Konsole sehen:

30

Tipp
Verwenden Sie eine neue int-Variable, um die Summe schrittweise zu summieren.

Aufgabe 6

Gegeben sei Array a. Drucken Sie alle negativen Elemente untereinander auf der Konsole aus.

```
1 int[] a  = { 1, -2, -25, 6, -3, 5
      };
```

Tipp
Sie benötigen eine if - Anweisung.

Aufgabe 7

Gegeben sind drei Arrays a, b und c. Verändern Sie Array c dahingehend, dass im Element c[0] die Summe a[0] und b[0] steht und entsprechend für die Elemente 1, 2, 3, ...

```
1 int[] a  = { 1, 2, 25, 6 };
2 int[] b  = { 9, 18, 5, 34 };
3 int[] c = new int[4];
4
5 // Hier Code schreiben
6
7 println(c);
```

Sie sollten das hier auf der Konsole sehen:

```
[0] 10
[1] 20
[2] 30
[3] 40
```

Aufgabe 8

Gegeben sind zwei Arrays a und b. Erzeugen Sie einen neuen Array c, der so lang ist wie a und b zusammengenommen und auch die Werte von a und b (in dieser Reihenfolge) enthält.

```
int[] a  = { 1, 2, 25 };
int[] b  = { 9, 18 };

// Hier Code schreiben

println(c);
```

Sie sollten das hier auf der Konsole sehen:

```
[0] 1
[1] 2
[2] 25
[3] 9
[4] 18
```

Verwenden Sie nicht die Processing-Funktionen concat() oder splice(). Ihr Code sollte auch funktionieren, wenn Sie bei a und/oder b Elemente hinzufügen oder entfernen.

Tipp Zerlegen Sie das Problem in zwei Schritte. In Schritt 1 befüllen Sie den ersten Teil von c mit den Werten von a. In Schritt 2 füllen Sie den zweiten Teil von c mit den Werten von b.
Jeden Schritt lösen Sie mit einer eigenen For-Schleife. Schritt 1 ist leicht, denn die Indexzahlen von a und c sind gleich (c[0] bekommt Wert von a[0] usw.). Schritt 2 erfordert eine kleine Änderung. Wie lautet die Indexzahl in c für den ersten Wert von b?

Aufgabe 9

Gegeben sei ein Array:

```
int[] feld1 = { 1, 2, 3, 4 };
```

Erstellen Sie ein neues Array `feld2`, wo die Elemente in umgekehrter Reihenfolge abgelegt sind. Wenn Sie `feld2` ausgeben, sehen Sie also:

```
[0] 4
[1] 3
[2] 2
[3] 1
```

Verwenden Sie nicht die Processing-Funktion reverse(). Wie immer soll Ihr Code auch für andere Arrays funktionieren.

Tipp
Sie laufen mit einer For-Schleife und Laufvariablen i durch 0, 1, 2, 3 und befüllen das neue Array `feld2`. Welchen Wert weisen Sie `feld2[i]` zu? Notieren Sie das per Hand. Es geht um einen Zusammenhang zwischen den Indexzahlen von `feld1` und `feld2`.

Aufgabe 10

Gegeben sei ein Array:

```
1 int[] feld = { 10, 20, 30, 40, 50
      };
```

Verschieben Sie alle Elemente ab Index 1 um eine Stelle nach rechts. Als Ergebnis sollte hinterher das folgende in feld stehen:

```
[0] 10
[1] 10
[2] 20
[3] 30
[4] 40
```

Aufgabe 11

Gegeben sei ein Array mit einer geraden Anzahl von Elementen:

```
1 int[] feld1 = { 1, 2, 3, 4, 5, 6
    };
```

Erstellen Sie ein neues Array `feld2`, das halb so lang ist wie feld1 und wo jeweils zwei benachbarte Elemente von `feld1` addiert wurden, also hier 1+2 und 3+4 und 5+6. Wenn Sie `feld2` ausgeben, sehen Sie also:

```
[0] 3
[1] 7
[2] 11
```

1.3 Aufgabenabschnitt 2

Aufgabe 1

Es soll die Häufigkeit der Augensumme beim Werfen eines Würfels bestimmt werden. Dazu sollen die Würfelversuche durch den Computer simuliert werden. Die Anzahl der Würfe soll durch den Anwender eingegeben werden.
Entwerfen und implementieren Sie ein Programm, das nach Eingabe der Anzahl der Würfe die entsprechende Anzahl der Augen ausgibt.

Aufgabe 2

Belegen Sie ein ein Feld von 1000 Feldelementen mit zufälligen ganzen Zahlen von -100 bis 100.

Ermitteln Sie die Anzahl der negativen Zahlen , der positiven Zahlen und der Nullen.

Geben Sie diese Anzahlen auf dem Bildschirm aus.

1. Erarbeiten Sie sich einen Algorithmus indem Sie per Hand die Aufgabe für 6 von Ihnen gewählten Zahlen auf Papier lösen. Schreiben Sie sich dabei Ihr Vorgehen in Stichpunkten auf.
2. Überführen Sie Ihre Stichpunkte in ein Struktogramm und schreiben Sie sich auch die Datenstruktur auf.
3. Implementieren Sie jetzt Ihren Entwurf in ein Java-Programm.
4. Führen Sie Funktionstests aus und dokumentieren Sie diese.

Aufgabe 3

Es ist ein Programm zu erstellen, das in ein Feld „quad“ die ersten 100 ganzzahligen Quadratzahlen (ohne 0) ablegt und dann alle geraden Quadratzahlen in das Feld „gerade“ und alle „ungeraden“ in das Feld ungerade abspeichert.

Danach sollen alle Feldinhalte auf dem Bildschirm ausgegeben werden.

1. Erarbeiten Sie sich einen Algorithmus indem Sie per Hand die Aufgabe für 6 Quadratzahlen auf Papier lösen. Schreiben Sie sich dabei Ihr Vorgehen in Stichpunkten auf.
2. Überführen Sie Ihre Stichpunkte in ein Struktogramm und schreiben Sie sich auch die Datenstruktur auf.
3. Implementieren Sie jetzt Ihren Entwurf in ein Java-Programm.
4. Führen Sie Funktionstests aus und dokumentieren Sie diese.
5. Erweitern Sie Ihr Programm, so dass eine frei wählbare Anzahl von Quadratzahlen in gerade und ungerade Zahlen sortiert werden.

Aufgabe 4

Es sollen alle Primzahlen zwischen 2 und 100 bestimmt werden. Dazu verwenden wir das sogenannte Sieb des Eratosthenes:
Zunächst wird auf die Zahl 2 positioniert, und dann werden alle Vielfache von 2 herausgestrichen:
2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ ...
Im nächsten Schritt wird um eine Zahl weiterpositioniert, in unserem Fall also auf die 3. Ist diese Zahl nicht herausgestrichen, so handelt es sich um eine Primzahl, von der wiederum alle Vielfachen zu streichen sind:
2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ...
Die nächste Primzahl nach 3 wäre dann 5, von der wieder alle Vielfachen zu streichen sind, usw. Dieses Verfahren wird solange wiederholt, bis 100 erreicht ist.
Entwerfen und implementieren Sie für diesen Algorithmus in ein Java-Programm.