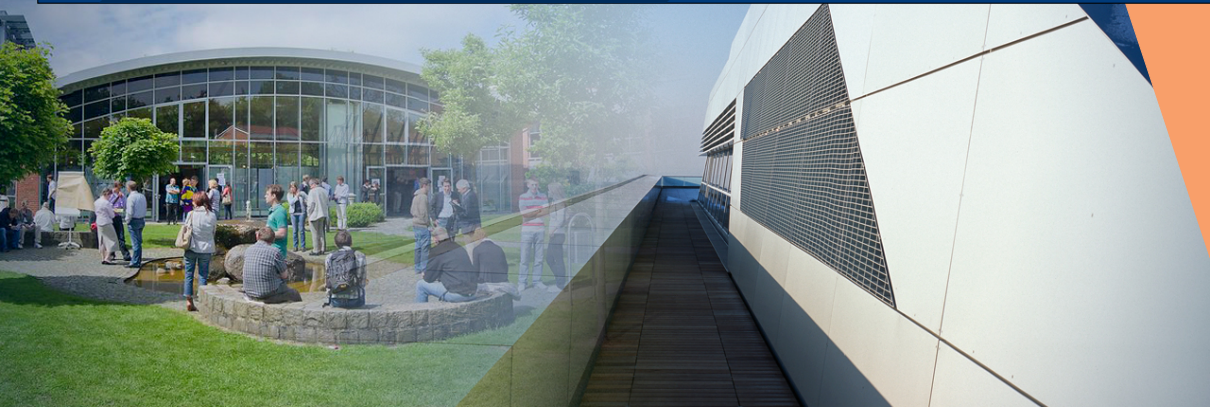
Kapitel 4.10, 4.12, 7

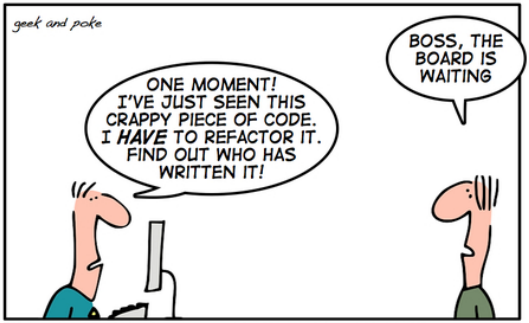
M. Brandenburg, M. Gohl, J. Kleimann, H.-W. Sehring

Studiengang Wirtschaftsinformatik

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT



Vorlesungsinhalt



Lerninhalte

Nach Beenden dieser Lektion werden Sie

- Mehrfachausführungen mittels der gängigen Schleifenkonstrukte durchführen können.
- Mengen von Objekten fester Größe mittels Arrays verwalten können.
- Das Konstrukt `null` mit den verbundenen Konsequenzen kennenlernen.
- Techniken zum Manipulieren von Mengentypen nutzen können.

Wiederholung Klassen

- Klassenrümpfe enthalten Exemplarvariablen, Konstruktoren und Methoden.
- Exemplarvariablen speichern Werte. Die Gesamtheit aller Werte der Exemplarvariablen machen den Status eines Objekts aus.
- Konstruktoren erzeugen und initialisieren Objekte.
- Methoden beschreiben das Verhalten eines Objekts.

Wiederholung Variablen

- Exemplarvariablen, Parameter und lokale Variablen sind Variablen zum Speichern von Informationen.
- Exemplarvariablen gibt es für die Lebensdauer des Objekts.
- Lokale Variablen werden für kurzfristiges Zwischenspeichern von Zwischenergebnissen verwendet.
- Parameter werden zur Übergabe von Informationen an Konstruktoren und Methoden verwendet.

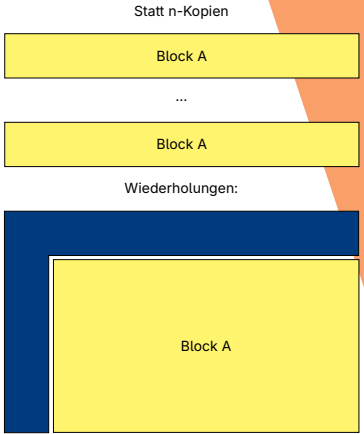
Wiederholung Methoden

- Methoden haben einen Rückgabetyp.
- Ist der Rückgabetyp `void`, dürfen die Methoden nichts zurückgeben.
- Ist der Rückgabetyp nicht `void`, müssen die Methoden immer einen Wert zurückgeben, der mittels der `return`-Anweisung festgelegt wird.
- Nach einem `return` wird die Methode verlassen.

Wiederholte Ausführung eines Blockes

Beim Berechnen, beim Arbeiten mit Sammlungen u.s.w. gibt es häufig die Notwendigkeit Anweisungen (Blöcke) mehrfach auszuführen. Dies kann man erreichen durch

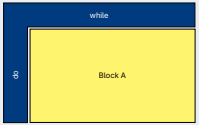
- entsprechend häufiges Wiederholen der Anweisung im Code oder
- durch spezielle Konstrukte, die dies systematisch und flexibel (Anzahl der Wiederholungen) ermöglichen.



Schleifenkonstrukte: while, do

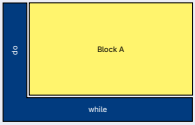
while-Schleife (abweisende Schleife)

```
int i=5;
while (i > 0) {
  System.out.println("Countdown:␣
      "+i);
  i=i-1;
}
```



do-Schleife (nicht abweisende Schleife)

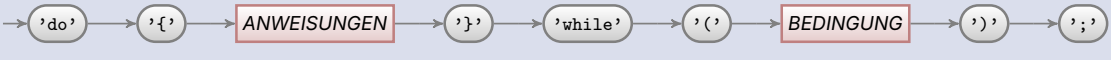
```
int i=1;
do {
  System.out.println("Jetzt␣zum␣"
      +i+".Mal!␣");
  i=i+1;
} while (i <= 10);
```



Schleifenkonstrukte (Syntax)

Abweisende Schleife `while`

Nicht abweisende Schleife `do...while`



Zählende Schleife

Anweisungsform

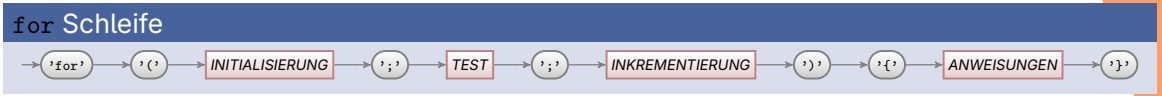
```
for (Initialisierung; Test; Inkrementierung)
  Anweisung;
```

Beispiel

```
for (int i = 0; i <= 10; i++) {
        System.out.println(i+"␣"+(10-i));
}
// Dasselbe aber mit zwei Variablen
for (int i = 0, j = 10; i <= 10; i++, j--) {
  System.out.println(i+"␣"+j);
}
```

- Anmerkung: Schleifen können (theoretisch) durch `break` verlassen werden.
- **Erinnerung:** Blöcke mit geschweiften Klammern fassen Anweisungen zusammen. Sie sollen verwendet werden, um die Lesbarkeit zu erhöhen.

Schleifenkonstrukte (Syntax)



Übungsaufgabe Test zu Foliensatz 3 Teil 1

- Schreiben Sie eine Methode `void printStars(int number)`, die in `number` übergebene Anzahl von „*“ auf die Console ausgibt.
- Schreiben Sie eine Methode, `void printTriangle(int number)`, die ein Dreieck von Sternen der Größe `number` ausgibt. Beispiel `number=4`:

```
*
**
***
****
```

- Schreiben Sie eine Methode `void printDiamond(int number)` die einen Diamanten der Breite 2*number+1 ausgibt. Beispiel `number=2`:

```
  *
 ***
*****
 ***
  *
```

Arithmetische Operatoren

Arithmetische Operatoren erlauben es Ausdrücke mit Zahlen zu berechnen.

Operator	Bedeutung	Gleitkomma	Festkomma
+	Addition	1.2+1.2=2.4	2+1=3
-	Subtraktion	1.2-0.5=0.7	2-1=1
*	Multiplikation	1.2*1.2=1.44	2*2=4
/	Division	1.2f/0.6f=2.0f	2/3=0
%	Rest	-	8%5=3
|	bitweises oder	-	5|3=7
∧	exklusives oder	-	5∧3=6
&	und	-	5&3=1

Operatoren: Ganzzahlige Division mit Rest

Operatoren (Geteilt / und Rest einer ganzzahligen Division %)

- Wenn der Divisionsoperator / auf `int`-Werte angewendet wird, dann ist das Ergebnis in Java auch ein `int`-Wert. `/` zwischen `int`-Werten ist also eine ganzzahlige Division.
- Der Rest einer Division wird mit dem Operator `%` berechnet.
- 17 geteilt durch 5 ergibt 3 Rest 2:
  $17/5 == 3$
  $17\%5 == 2$

Übungsaufgabe Tests zu Foliensatz 3 Teil 2

Schreiben Sie die folgenden Methoden, verwenden Sie jeweils einmal eine `while`- und einmal eine `for`-Schleife

- `void divisors(int n)` und `void properDivisors(int n)`, die für die Zahl n ihre (echten) Teiler ausgibt. Beispiel:
  Teiler von 15 sind 1, 3, 5, 15 Echte Teiler von 15 sind 3, 5
- Schreiben sie eine Methode `prime(int n)`, die ausgibt, ob eine Zahl Primzahl ist, oder nicht.

Weitere Übung: Kreiszahl π berechnen

Schreiben Sie eine Methode, mit deren Hilfe sich π berechnen lässt:

- `void pi(int genauigkeit)` berechnet π mit Durchläufen. Zur Erinnerung, π berechnet sich folgendermaßen:

$$\pi = 4 * (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ...)$$

- Schreiben Sie eine weitere Methode, mit der Sie Ihre errechnete Kreiszahl mit `Math.PI` vergleichen. Die Funktion soll ein `double` als Parameter für die Genauigkeit übergeben bekommen.

Sammlungen mit vordefinierter Größe

- Sehr häufig muss man in einem Programm identische Aufgaben mehrfach für dieselben Arten von Informationen durchführen. Etwa die Bestimmung eines mittleren Umsatzes oder den Gesamtbetrag aller Bestellpositionen in einem Warenkorb ermitteln. Zur Verwaltung dieser Informationen verwendet man in Java **Sammlungen**.
- Manchmal ist die maximale Anzahl an Elementen im Vorhinein bekannt.
- Dann steht eine spezielle Sammlung mit festlegbarer Größe bereit: ein **Array**.
- Arrays können sowohl primitive Datentypen als auch Objekte speichern.

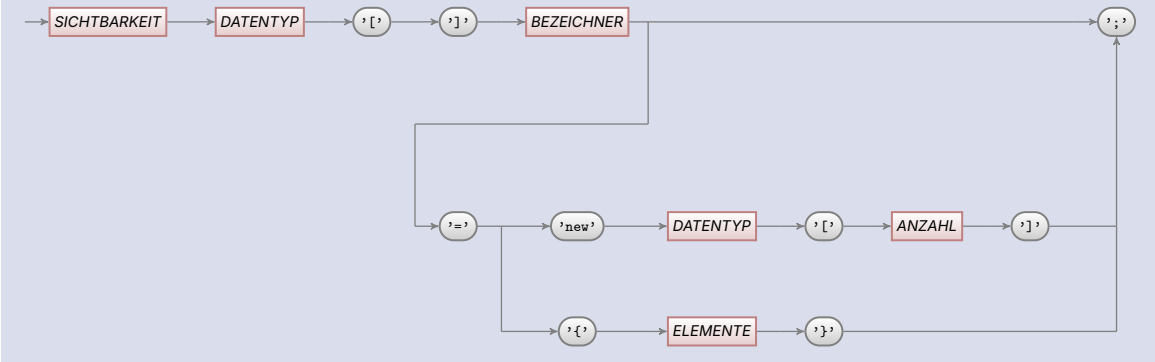
Eigenschaften von Arrays

Arrays

- haben eine **feste** Größe - dies unterscheidet Sie von den Klassen `ArrayList`, `HashSet`, `HashMap` etc. (siehe folgende Veranstaltungen), die eine beliebige Anzahl aufnehmen können und die wir später besprechen werden.
- erfordern eine spezielle Syntax. Aus historischen Gründen werden Arrays in Java durch eckige Klammern „[ ]“ angezeigt.
- sind Objekte ohne Methoden.

Erstellung eines Arrays (Syntax)

Verallgemeinert (Array Erstellung)



Übung

7.3

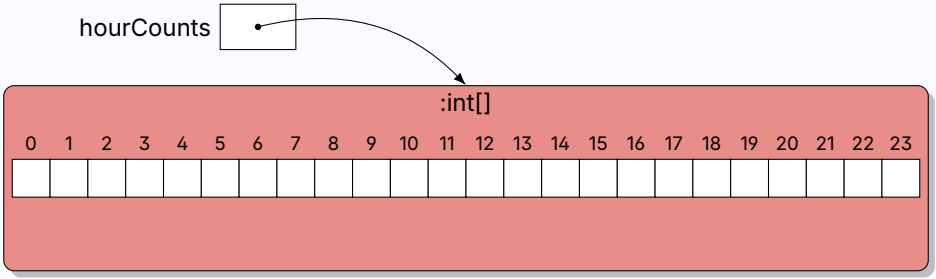
Schreiben Sie eine Deklaration für eine Array-Variable `available`, die auf ein Array von `boolean`-Werten verweisen kann.

7.5

Was ist falsch an den folgenden Array-Deklarationen? Korrigieren Sie.

```
[]int counter;
boolean[5000] occupied;
```

Das Array `hourCounts`



Verwendung eines Arrays

- Eine Notation basierend auf eckigen Klammern wird verwendet, um auf Elemente eines Arrays zuzugreifen:

```
hourCounts[...]
```

- Elemente eines Arrays werden wie „normale“ Variablen verwendet
  - Das Ziel einer Zuweisung:

```
hourCounts[hour] = 0;
```

  - In einem Ausdruck:

```
int incrementedHours = hourCounts[hour]+1;
hourCounts[hour]++;
if (hourCounts[hour] > 0) ...
```

  - Kombinierte Zuweisung auf einen Ausdruck:

```
hourCounts[hour] = hourCounts[hour]+1;
hourCounts[hour]++;
```

Normale Verwendung eines Arrays

1. Deklaration

```
private int[] hourCounts;
private String[] names;
...
```

2. Erstellung

```
...
hourCounts = new int[24];
...
```

3. Verwendung

```
hourCounts[hour] = 0;
hourCounts[hour]++;
System.out.println(hourCounts[i]);
...
```

Initialisierung von Arrays

- Die Größe wird automatisch aus den angegebenen Daten ermittelt

```
private int[] numbers = { 3, 15, 4, 5 };
```

  dies fasst Deklaration, Erstellung und Initialisierung zusammen
- Diese Form der Array-Erstellung kann nur in Deklarationen verwendet werden
- Spätere Initialisierungen mit einer vorgegebenen Datenmenge erfordern ein `new`

```
numbers = new int[] { 3, 15, 4, 5, 8, 6};
```

- Beachten sie, dass durch eine erneute Zuweisung die Größe des Arrays, das in der Variablen `numbers` gespeichert wird, eine andere ist. Es ist ein anderes Array!

Arrays mit Objekten

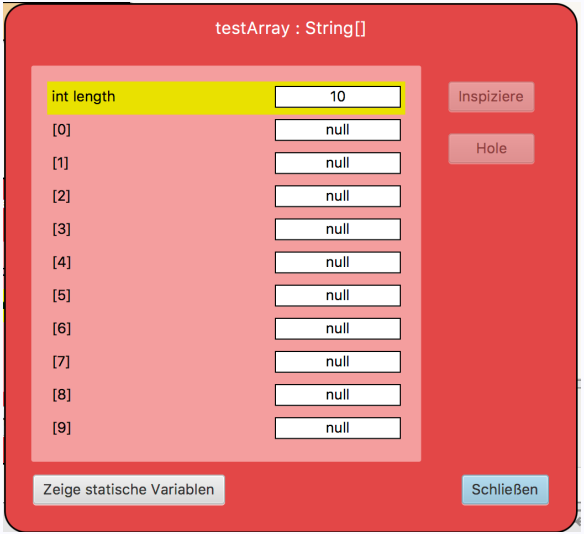
Arrays können auch Objekte einer Klasse enthalten.
Hier ein Beispiel eines String Arrays:

1. 1. Deklaration
   `String[] g7;`
2. Erstellung
   `g7 = new String[7];`
3. Verwendung
   `g7[3]=‘‘Great Britain‘‘;`

Bitte beachten:

- Nach der Erstellung enthält das Array zunächst nur `null` „Werte“. Eine Nutzung der Objekte ist erst nach einer initialen Zuweisung (Punkt 3 Verwendung) möglich.
- Auch eine deklarative Initialisierung ist möglich:
  `String[] g7={‘‘United States‘‘,...,‘‘Germany‘‘}`

Null Einträge in Objekt-Arrays



Inhalt von initialisierten Arrays

- Nachdem das Array mit einer Größenangabe initialisiert wird, hält es den Speicher für die darin zu speichernden Objekte vor.
- Gefüllt ist das Array mit Verweisen auf `null`, wenn es sich um Objekte (String, Rechteck, Student etc... handelt). Bei primitiven Typen (int, boolean, double, ...) richtet sich dies nach dem default für den jeweiligen Typen.
- Häufiger Fehler: Wenn nicht explizit auf null geprüft wird z.B. mittels `if(array[1] != null) ...` wird dies durch einen Fehler beim Zugriff auf das Objekt quittiert: **NullPointerException**.

Die Länge von Arrays

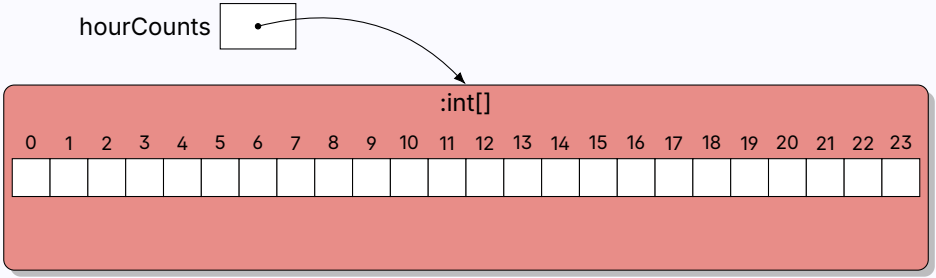
```
private int[] numbers = { 3, 15, 4, 5 };

int n = numbers.length;
```

`numbers.length` ist kein Methodenaufruf!

- `length` ist eine Exemplarvariable, keine Methode
- Der Wert wird bei Erstellung von Arrays festgelegt und kann nicht verändert werden: **feste Größe!**

Nummerierung der Elemente



- Die Nummerierung beginnt bei **0**.
- Die Nummerierung endet bei **length-1**.
- Häufiger Fehler: Zugriff `hourCounts[hourCounts.length]` spricht nicht das letzte Element des Arrays an, sondern führt zu einem Programmabbruch mit einer **ArrayIndexOutOfBoundsException**.

Übung

Initialisierung

Erzeugen Sie ein Array vom Typ `String` und initialisieren Sie dieses mit den Werten: `“ich”`,`“habe”`,`“keine”`,`“Lust”`. Initialisieren Sie das Array sowohl deklarativ als auch konkret unter Angabe der Größe und explizitem Setzen jedes einzelnen Wertes.

Iteration

Iterieren Sie über das Array mittels einer `for`-Schleife und geben Sie den Inhalt jeweils in der Konsole aus.

Veränderung

Entfernen Sie den dritten Parameter des Arrays (`“keine”`) und setzen Sie an dieser Stelle `null` ein.

Zugriff auf Objektinhalt

Geben Sie nun auch die Länge des jeweiligen Wortes aus. Dies können Sie mit der Methode `length()` des Strings realisieren.

Initialisierung von Arrays

Erzeugen Sie jeweils Arrays folgenden Typs und inspizieren Sie die default Belegungen:

```
int[] numbers = new int[3];
char[] chars = new char[3];
boolean[] booleans = new boolean[3];
float[] floats = new float[3];
double[] doubles = new double[3];
String[] strings = new String[3];
```

- Welches sind die jeweiligen default Belegungen?
- Provozieren Sie bewusst eine **NullPointerException**, es wird nicht Ihre letzte sein!

Die `for` Schleife

- Es existieren 2 Varianten von `for` Schleifen: `for` und `for-each`.
- Die `for` Variante wird verwendet, wenn man die Elemente eines Arrays nutzen und durch andere Elemente ersetzen will.
  - Diese Schleife haben wir bereits eben kennen gelernt
- Die `for-each` Variante erlaubt nur das Nutzen, aber nicht das Verändern aber das Ersetzen von Elementen.

Durchlaufen der Elemente eines Arrays

Wiederholung (`for`-Schleife)

Eine sehr häufig verwendete Art auf alle Elemente eines Arrays zuzugreifen, ist eine for-Schleife, die alle denkbaren Indices durchläuft.

`for` Schleife: Nutzen eines Elements

```
for (int hour = 0; hour < hourCounts.length; hour++) {
  System.out.println(hour + ":␣" + hourCounts[hour]);
}
```

`for` Schleife: Ersetzen eines Elements

```
for (int hour = 0; hour < hourCounts.length; hour++) {
  hourCounts[hour]=hourCounts[hour]+1;
}
```

Durchlaufen der Elemente eines Arrays

Alternativ verwendet man sehr häufig die `for-each`-Schleife, um auf alle Elemente eines Arrays zuzugreifen:

`for-each`-Schleife: Nutzen eines Elements

```
for (int hourCount: hourCounts) {
  System.out.println(hourCount); //Zugriff auf Index unmöglich
}
```

`for-each`-Schleife: Ersetzen eines Elements unmöglich

```
for (int hourCount: hourCounts) {
  hourCount++; //Verändert das Array nicht!
}
```

for-each-Schleife mit String Array

for-each-Schleife: Nutzen eines Elements

```
for (String state: g7) {
  System.out.println(state); //Zugriff auf Index unmöglich
}
```

for-each-Schleife: Ersetzen eines Elements unmöglich

```
for (String state: g7) {
  state+="(G7␣Member)"; //Verändert das Array nicht!
}
```

Schleifenkonstrukte (Syntax)

for-each Schleife

Übung

- Gegeben ein Array mit den Zahlen $4, 1, 22, 9, 14, 3, 9$. Drucken Sie, unter Verwendung einer `for`-Schleife alle Zahlen aus.
- Signatur der Methoden:
  `public void printNumbers(int[] numbers)`

Übung

Die for-Schleife ist eine sehr variable Art der Schleife. Sie erlaubt:

- Anpassen der Schrittweite
- Anpassen der Anfangswerte
- Anpassen der Abbruchbedingung (Endwerte)

Aufgabe: Geben Sie alle zweistelligen Vielfachen von 3 aus.

Übung Test zu Foliensatz 3 Teil 3

Das Pascal'sche Dreieck besteht aus den Binomialkoeffizienten $\binom{n}{k}$. Berechnen Sie ein Array von Binomialkoeffizienten $\binom{n}{k}$ mit $0 \leq k \leq n \leq 10$ mittels der Formel:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{falls } 0 < k < n \\ 1 & \text{falls } k = n \text{ oder } k = 0 \end{cases}$$

Geben Sie das Pascal'sche Dreieck aus:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
...
```

Übungsaufgabe Prüfziffer Mastercard Test Teil 3

Die Prüfziffer ist die letzte Ziffer einer Kreditkartenummer. Zum Prüfen einer Kreditkartennummer bzw. zum Erzeugen der Prüfziffer wird in der Praxis der Luhn-Algorithmus, den meisten eher als „Mod 10“-Algorithmus bekannt, eingesetzt. Die Vorgehensweise ist dabei relativ einfach:

1. Von der vorletzten Ziffer der Kreditkartennummer an wird diese nach links durchlaufen, wobei der Wert jeder zweiten Ziffer verdoppelt wird.
2. Von den einzelnen Ziffern bzw. Zahlen wird nun die Quersumme ermittelt (bei zweistelligen Ergebnissen werden beide Ziffern auseinander genommen und separat addiert!!!).
3. Von dieser Summe wird das Modulo 10 (Mod 10) berechnet, indem das nächstkleinere Vielfache der Quersumme von 10 berechnet wird (Beispiel: Quersumme 68, Modulo 10 von 68 = 6 Rest 8)
4. Im letzten Schritt wird der Modulo 10 der Quersumme (in unserem Beispiel 2) von 10 subtrahiert. Das Ergebnis ist die Prüfziffer (Beispiel: 10 – 8 = 2)

Beispielberechnung

Ziffer	Faktor	Ziffer*Faktor	Quersumme
4	2	8	8
6	1	6	6
8	2	16	7
3	1	3	3
4	2	8	8
5	1	5	5
7	2	14	5
8	1	8	8
2	2	4	4
9	1	9	9
3	2	6	6
7	1	7	7
6	2	12	3
5	1	5	5
2	2	4	4
		Summe:	88

Kreditkartennummer:
4683 4578 2937 652

Summe	88
Modulo 10	8 Rest 8
Differenz	10-8
Prüfziffer	***2***

Vollständige Kreditkartennummer:
*4683 4578 2937 652**2***

Zusammenfassung

- Arrays sind genau dann gut verwendbar, wenn eine Sammlung mit fester Größe benötigt wird.
- Arrays bringen eine spezielle Syntax mit.
- `for` Schleifen werden verwendet, wenn ein Index (eine Indexvariable) benötigt wird.
- `for` Schleifen stellen eine Alternative zu `while` Schleifen da, wenn die Anzahl der Iterationen vorher bekannt ist.
- `for` Schleifen arbeiten mit einer vordefinierten Schrittweite.

Methoden mit Bezug zu Arrays

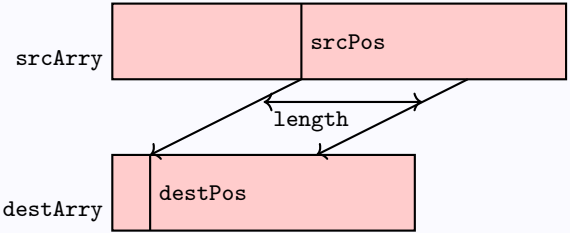
Array haben keine spezifischen Methoden und es ist in Java auch nicht möglich neue Methoden zu definieren. Deshalb gibt es „Werkzeugmethoden“, um Arrays zu manipulieren.

- `System` besitzt eine statische Methode `arraycopy`.
- `java.util.Arrays` enthält mehrere statische Hilfsmethoden, die bei der Verarbeitung von Arrays hilfreich sind:
  - `fill`
  - `sort`
  - `binarySearch`
  - u.v.a.m.

System.arraycopy

```
System.arraycopy(srcArr, srcPos, destArr, destPos, length);
```

- srcArry: Dies ist das Array, das kopiert werden soll.
- srcPos: Dies ist die Startposition. Bedenken Sie, dass ein Array bei 0 startet.
- dest: Dies ist ein Zielarray.
- destPos: An diese Position soll angehängt werden.
- length: Dies ist die Anzahl zu kopierender Array-Elemente.



Arrays.fill

Die Methode `Arrays.fill(int[], int)` füllt ein gegebenes Array mit einer festen Zahl auf:

Was ist die Ausgabe?

```
int arr[] = new int[] {1, 6, 3,
    2, 9};

System.out.println("Actual␣
    values:␣");
for (int value : arr) {
  System.out.println("Value␣=␣"
      + value);
}
```

```
Arrays.fill(arr, 18);

System.out.println("After␣fill()
    ␣method:␣");
for (int value : arr) {
  System.out.println("Value␣=␣"
      + value);
}
```

Arrays.sort

Sortieren eines int Arrays

```
int arr[] = new int[] {1, 6, 3, 2, 9};

System.out.print("Actual␣sequence:␣");
for (int value : arr) {
  System.out.print(value+",␣");
}
System.out.println();

Arrays.sort(arr);

System.out.print("After␣sort-method:␣");
for (int value : arr) {
  System.out.print(value+",␣");
}
```

Mehrdimensionale Arrays

- Die Syntax von Arrays erlaubt mehrdimensionale Arrays
  - z.B., 2D Arrays zum Repräsentieren eines Spielfeldes oder allgemeiner einer Anordnung von Zellen
- Vorstellen kann man sich dies als Array von Arrays.

Das Projekt *brain*

```
Cell[][] cells;
...
cells = new Cell[numRows][numCols];
...
for (int row = 0; row < numRows; row++) {
  for (int col = 0; col < numCols; col++) {
    cells[row][col] = new Cell();
  }
}
```

Alternative Iteration (*brain*)

`length` anstatt „externer“ Wert

```
for (int row = 0; row < cells.length; row++) {
  Cell[] nextRow = cells[row];
  for (int col = 0; col < nextRow.length; col++) {
    nextRow[col] = new Cell();
  }
}
```

- „Array von Arrays“
- Erfordert keinen Zugriff auf `numRows` und `numCols`
- Sie funktioniert sogar mit unregelmäßig geformten Arrays, die von Java ebenfalls unterstützt werden.

Übung Test Foliensatz 3 Teil 3

Füllen Sie ein zweidimensionales 8x8 Array schachbrettartig mit 0 en und 1 en und geben dies aus:

```
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
```

Übung

Füllen Sie ein zweidimensionales 8x8 Array zeilenweise von links nach rechts mit Zahlen. Fangen Sie in der linken oberen Ecke mit 1 an. Nach dem Ende der ersten Zeile machen Sie in der zweiten Zeile weiter. Geben Sie das Array aus:

```
1  2  3  4  5  6  7  8
9  10  11  12  13  14  15  16
...
57  58  59  60  61  62  63  64
```

Das Projekt *automaton*

- Ein Array von „Zellen“
- Jede Zelle verwaltet einen einfachen Zustand
  - Üblicherweise ist dies ein numerischer Wert
  - z.B., an/aus oder lebendig/tot
- Der Zustand verändert sich auf der Basis einfacher Regeln.
- Die Regeln betrachten dazu den Zustand der benachbarten Zellen.

Ein einfacher Automat

```
nextState[i] =
    (state[i-1] + state[i] + state[i+1]) % 2;
```

Step	Zustände der Zellen - leere Zellen sind im Zustand tot												
0							+						
1						+	+	+					
2					+		+		+				
3				+	+		+		+	+			
4			+				+				+		
5		+	+	+		+	+	+		+	+	+	
6	+		+				+				+		+

Wertauswahl bei gegebener Bedingung

Auswahl eines Wertes anhand einer Bedingung

```
Bedingung ? Wert1 : Wert2
```

Verwendung im Code

```
for (int cellValue : state) {
  System.out.print(cellValue == 1 ? '+' : '␣');
}
System.out.println();
```

Übung für Spielkinder

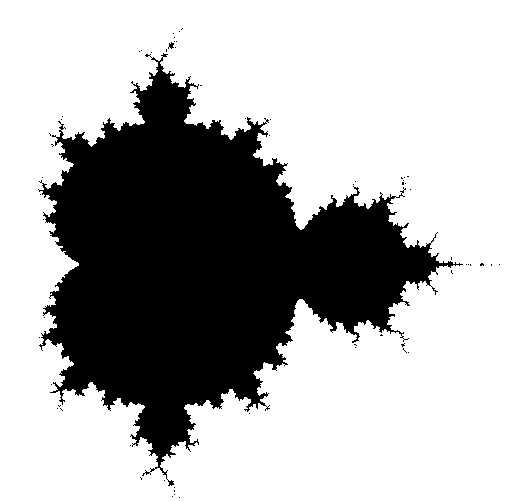
- Erstellen Sie eine Klasse `ComplexNumber`, die Exemplarvariablen **re** und **im** für den Realteil und den Imaginärteil hat: $z = a + b \cdot \mathrm{i}$. [1]
- Schreiben Sie getter-Methoden für den Real- und den Imaginärteil.
- Schreiben Sie einen Konstruktor, in dem der Real- und der Imaginärteil gesetzt werden.
- Schreiben Sie Methoden mit den Signaturen
  `public ComplexNumber add(ComplexNumber)`,
  `public ComplexNumber multiply(ComplexNumber)` und
  `public double abs()`.
  Nutzen Sie dafür folgende Formeln:

$$
\begin{aligned}
(a + b \cdot \mathrm{i}) + (c + d \cdot \mathrm{i}) &=_{def} (a + c) + (b + d) \cdot \mathrm{i} \\
(a + b \cdot \mathrm{i}) \cdot (c + d \cdot \mathrm{i}) &=_{def} (a \cdot c - b \cdot d) + (a \cdot d + b \cdot c) \cdot \mathrm{i} \\
|a + b \cdot \mathrm{i}| &=_{def} \sqrt{a^2 + b^2}
\end{aligned}
$$

[1] In dieser Aufgabe dürfen Sie ausnahmsweise `double` verwenden.

Mandelbrot

- Nutzen Sie die in der letzten Übung definierte Klasse (`ComplexNumber`) um die Mandelbrotmenge zu visualisieren.



Erklärung der Mandelbrotmenge

- Für eine komplexe Zahl c bildet man die Folge komplexer Zahlen:

$$\begin{aligned} z_0 &= c \\ z_{n+1} &= z_n \cdot z_n + c \end{aligned}$$

- Diese Folge kann je nach unterschiedlichem Startwert unterschiedliches Verhalten haben:
  1. Die z_n können alle einen Betrag kleiner gleich 2 haben. Das heißt, die Folge kann in einem Kreis mit Radius 2 um die 0 liegen.
  2. Die Folge kann ab einer Stelle einen Betrag größer als 2 haben. Dann wird sie von da an immer größere Beträge haben und gegen ∞ divergieren.
- Die Mandelbrotmenge ist die Menge aller der Startpunkte c, für die die Folge gutmütig bleibt, also betragsmäßig 2 niemals überschreitet.

Übung Mandelbrot

Immer noch für Spielkinder...

Man kann das Verhalten der Folge z_n mit dem Computer annähern, wenn man nicht Grenzwerte untersucht, sondern den Wert z_n für eine großes n, also etwa n=1000.

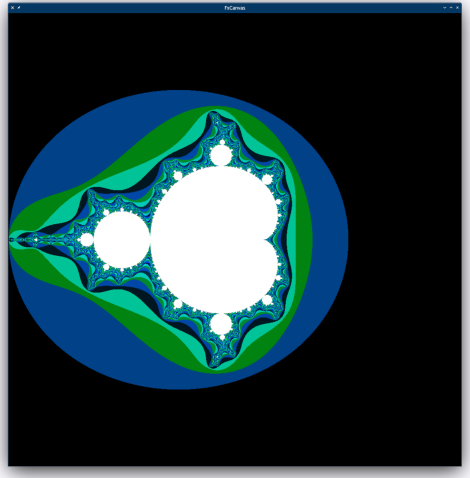
- Schreiben Sie eine Klasse Mandelbrot.
- Schreiben sie eine Methode mit der Signatur
  `private ComplexNumber sequence (int n, ComplexNumber c)`
  die das n-te Folgenglied zu dem Startpunkt c zu berechnen.
- Schreiben Sie eine Methode mit der Signatur `public void draw()` die für eine von Ihnen zu bestimmende Anzahl von Startpunkten der Gaußschen Zahlebene mit Real- und Imaginärteil zwischen -2 und 2 das 1000ste Folgenglied berechnet und es ausgibt, falls der Betrag des 1000sten Folgenglied kleiner als 2 ist.
- Zum Anzeigen steht Ihnen eine Hilfsklasse mit einer Hilfsmethode zur Verfügung: `FxCanvas.draw(double x, double y)`. Dabei sollen x und y Koordinaten in der Gaußschen Zahlenebene sein, die zwischen -2 und 2 liegen.

Übung Mandelbrot

Erweiterung

Die Darstellung wird interessanter, wenn man die Grafik einfärbt. Dann wird auch fraktale Natur (Selbstähnlichkeit) des Apfelmännchens deutlicher. Man kann die Anzahl der Iterationen, die bis zum Erkennen der Divergenz notwendig sind, in eine Farbe übersetzen.

Dann ergibt sich als Übungsergebnis beispielsweise:



Fragen?

Für weitere Fragen im Nachgang können Sie mich gerne über Moodle oder via E-Mail kontaktieren!

Konzepte: Zusammenfassung I

- **Schleife** Eine Schleife kann verwendet werden, um einen Block von Anweisungen wiederholt auszuführen zu lassen, ohne das diese dafür mehrfach in den Quelltext geschrieben werden müssen.
- **Array** Ein Array ist eine besondere Art von Sammlung, die eine festgelegte Anzahl von Elementen halten kann.

Schleife

Eine Schleife kann verwendet werden, um einen Block von Anweisungen wiederholt auszuführen zu lassen, ohne das diese dafür mehrfach in den Quelltext geschrieben werden müssen.

Array

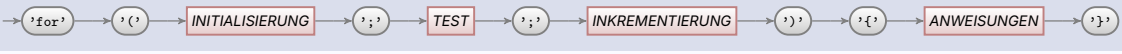
Ein Array ist eine besondere Art von Sammlung, die eine festgelegte Anzahl von Elementen halten kann.

Syntaxdiagramme I

Abweisende Schleife `while`

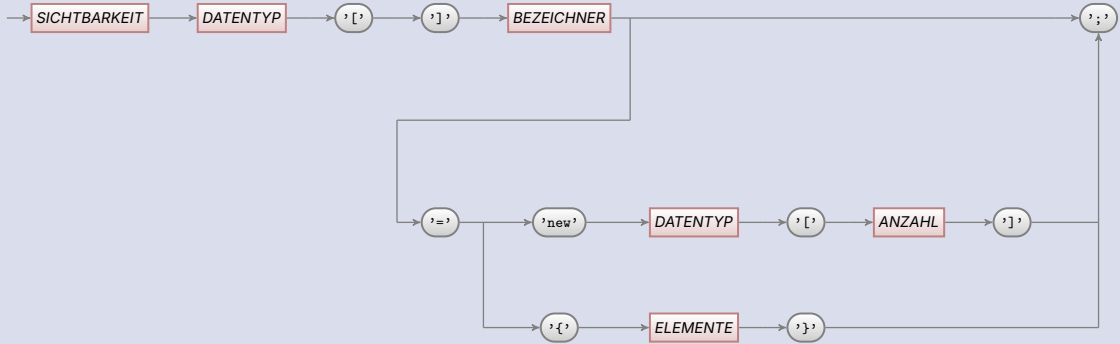
Nicht abweisende Schleife `do...while`

`for` Schleife



Syntaxdiagramme II

Verallgemeinert (Array Erstellung)



for-each Schleife