

MEHRDIMENSIONALE ARRAYS UND LISTEN

**Komplexere Datenstrukturen und die einfache Verwaltung
größerer Datenmengen**

Struktur und Inhalt des Kurses wurden 2012 von Markus Heckner entwickelt.
Im Anschluss haben Alexander Bazo und Christian Wolff Änderungen am
Material vorgenommen. Die aktuellen Folien wurden von Alexander Bazo
erstellt und können unter der [MIT-Lizenz](#) verwendet werden.

AKTUELLER SEMESTERFORTSCHRITT (WOCHE 11)

Kursabschnitt	Themen		
Grundlagen	Einführung	Einfache Programme erstellen	Variablen, Klassen & Objekte
	Kontrollstrukturen & Methoden	Arrays & komplexe Schleife	
Klassenmodellierung	Grundlagen der Klassenmodellierung		
	Vererbung & Sichtbarkeit		
Interaktive Anwendungen	Event-basierten Programmierung		String- & Textverarbeitung
Datenstrukturen	Listen, Maps & die Collections		Speicherverwaltung
	Umgang mit Dateien		
Software Engineering	Planhaftes Vorgehen bei der Softwareentwicklung		
	Qualitätsaspekte von Quellcode		

PINGO-QUIZ

 large-image

<https://pingo.coactum.de/792176> 

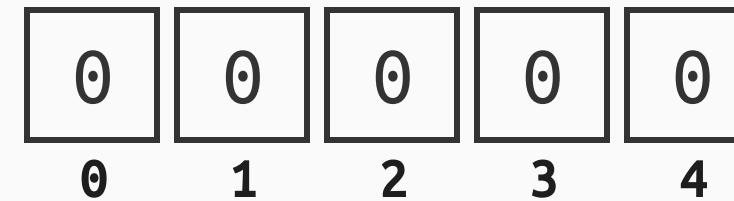
DAS PROGRAMM FÜR HEUTE

- Kurze Wiederholung zu Arrays
- Mehrdimensionale Datenstrukturen und Listen
- Bildverarbeitung mit der ArrayList

WIEDERHOLUNG: ARRAYS

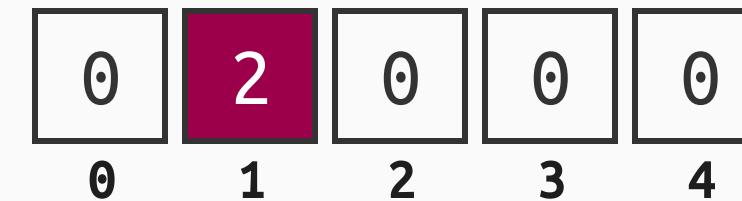
Arrays erstellen

```
1 int[] values = new int[5];
```



Arrays belegen und Werte auslesen

```
1 values[1] = 2;
```



```
1 int content = values[1]; // Returns 2
```

Länge eines Arrays bestimmen

```
1 int length = values.length; // Returns 5
```

WICHTIGES ZU ARRAYS (1/2)

- Arrays können für jeden Datentypen erzeugt werden.
- Bei der Initialisierung werden die Felder des Arrays mit *Default-Werten* belegt (primitive Datentypen) oder sind *leer* (null, bei Objekten).
- Auf die einzelnen Elemente kann über die Indizes (0 bis *Länge des Arrays - 1*) zugegriffen werden. Wird auf einen ungültigen Index zugegriffen, entsteht ein Fehler: `IndexOutOfBoundsException`

WICHTIGES ZU ARRAYS (2/2)

Arrays lassen sich auch als Literale definieren:

```
1 int[] values = {2,5,4,6};  
2  
3 /* entspricht */  
4  
5 int[] values = new int[4];  
6 values[0] = 2;  
7 values[1] = 5;  
8 values[2] = 4;  
9 values[3] = 6;
```

ARRAYS SORTIEREN

Algorithmen, die Listen von vergleichbaren Elementen nach gewünschten Kriterien sortieren, nennt man *Sortieralgorithmen*. Einer der einfachsten, in der Praxis aber kaum verwendeten, Sortieralgorithmen ist der **Bubble sort**-Algorithmus:

In der Bubble-Phase wird die Eingabe-Liste von links nach rechts durchlaufen. Dabei wird in jedem Schritt das aktuelle Element mit dem rechten Nachbarn verglichen. Falls die beiden Elemente das Sortierkriterium verletzen, werden sie getauscht. Am Ende der Phase steht bei auf- bzw. absteigender Sortierung das größte bzw. kleinste Element der Eingabe am Ende der Liste. Die Bubble-Phase wird solange wiederholt, bis die Eingabeliste vollständig sortiert ist. Dabei muss das letzte Element des vorherigen Durchlaufs nicht mehr betrachtet werden, da die restliche zu sortierende Eingabe keine größeren bzw. kleineren Elementen mehr enthält.

Je nachdem, ob auf- oder absteigend sortiert wird, steigen die größeren oder kleineren Elemente wie Blasen im Wasser (daher der Name) immer weiter nach oben, das heißt, an das Ende der Liste. Es werden stets zwei Zahlen miteinander in „Bubbles“ vertauscht.

Quelle: <https://de.wikipedia.org/wiki/Bubblesort>

BUBBLE SORT

```
1 public void bubbleSort(int[] array) {  
2     // Äußere Schleife  
3     for (int i = 1; i < array.length; i++) {  
4         // Innere Schleife: Vom ersten bis zum letzten, unsortierten Element  
5         for (int j = 0; j < array.length - i; j++) {  
6             // "Tauschen" der Plätze, bzw. Aufsteigen der Blasen  
7             if (array[j] > array[j + 1]) {  
8                 int tmp = array[j];  
9                 array[j] = array[j+1];  
10                array[j+1] = tmp;  
11            }  
12        }  
13    }  
14}
```

Hinweis: Es ist sehr sinnvoll, *Bubble Sort* und andere Algorithmen zu Übungszwecken zu implementieren. Weniger sinnvoll ist es, ohne guten Grund eigenen Sortier-Implementierungen in "richtiger" Software zu nutzen. Verwenden Sie dazu die vorgegebenen Funktionen, wie z.B. die Methode `Arrays.sort`.

MEHRDIMENSIONALE ARRAYS

VORBEMERKUNG

Wir sprechen heute von 2- oder mehrdimensionalen Arrays und meinen damit bestimmte Datenstrukturen, die eine tiefere Dimensionalität als "normale" Arrays haben. Technisch gesehen ist das nicht korrekt, weil ein Array in Java (!) immer nur eine Dimension hat. Der richtige Begriff für die gemeinten Strukturen wäre: *Array of an Array* (Ein Array aus Arrays). In der Praxis wird diese (sprachliche) Unterscheidung kaum angewandt.

ARRAYS ÜBER MEHRERE DIMENSIONEN

Ein Array kann aus beliebig vielen Dimensionen aufgebaut sein. D.h. jedes Element eines Arrays kann selbst ein Array sein. Auf der untersten Ebene stehen dann die Werte, die durch den Datentyp definiert werden. Eine solche Struktur eignet sich hervorragend, um 2-dimensionale Datenstrukturen (z. B. ein Schachbrett) abzubilden.

Hinweis: Java spezifiziert hier keine maximale Tiefe. Die *Java Runtimes* unterstützen jedoch meist nur Arrays mit einer maximalen Tiefe (Dimension) von 255: `int[][][]...[...]`

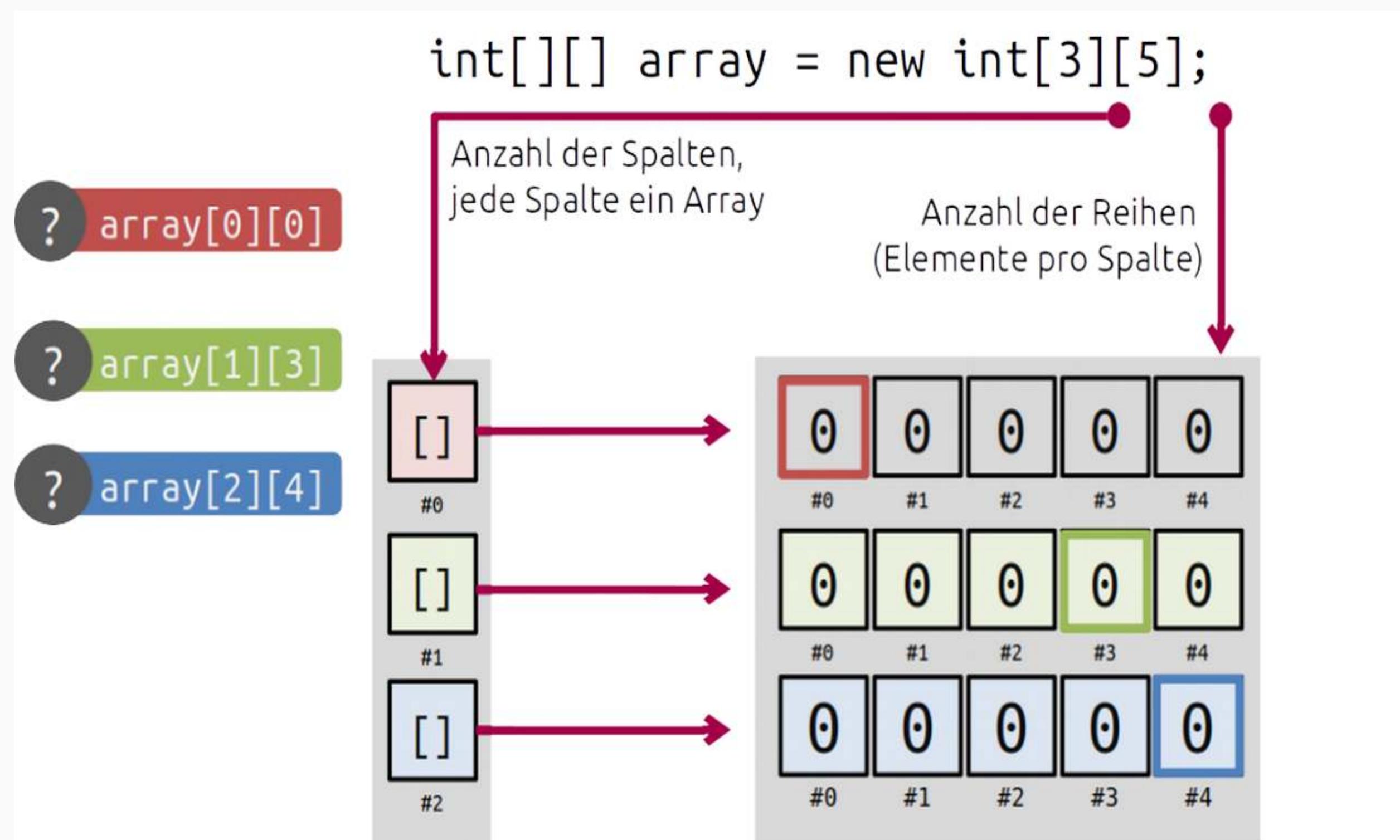
ANLEGEN EINES 2-DIMENSIONALEN ARRAYS

```
1 int[][] values = new int[x][y];
```

- Auch mehrdimensionale Arrays haben **einen** Datentyp und einen Namen.
- Die Länge der Dimensionen wird bei dem Erstellen der einzelne Arrays angegeben.
- Umgangssprachlich ausgedrückt wird hier "... ein Array der Länge x erstellt, in dem jedes Element ein eigenständiges int-Array der Länge y darstellt.".

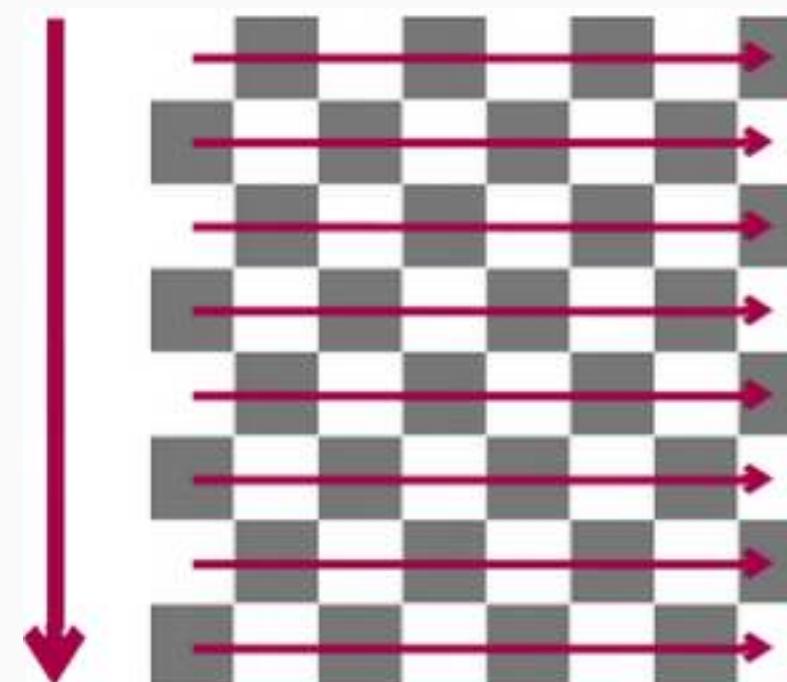
Hinweis: Ein 2-dimensionales Array besteht aus einem Array einer bestimmten Länge. Jedes Element dieses Array ist selbst wieder ein Array, mit den eigentlichen Werten.

AUFBAU EINES 2-DIMENSIONALEN ARRAYS



ITERATION ÜBER EIN 2-DIMENSIONALES ARRAY

```
1 int[][] checkerBoard = new int[8][8];
2
3 int numRows = checkerBoard.length;
4 int numCols = checkerBoard[0].length;
5
6 for (int rowIndex = 0; rowIndex < numRows; rowIndex++) {
7     for (int colIndex = 0; colIndex < numCols; colIndex++) {
8         int value = checkerBoard[rowIndex][colIndex];
9     }
10 }
```



DIE ARRAYLIST

EINSCHRÄNKUNGEN VON ARRAYS



+

In einem Array hat jedes Element seinen festen Platz: Arrays funktionieren dann gut, wenn wir im Vorfeld die Anzahl der Elemente kennen und zur Laufzeit keine Elemente entfernen müssen



+

Für dynamische Manipulation und Listen unterschiedlicher Länge benötigen wir etwas anderes: ArrayLists stellen mitwachsende Datenstrukturen dar, die ähnlich wie Arrays Objekte gleichen Typs bündeln und noch einige besondere Features haben. Sie funktionieren nur mit Objekten, nicht mit primitiven Datentypen.

VERWENDUNG DER ARRAYLIST (1/3)

Diese Klasse ist generisch formuliert. D.h. in der Klassendefinition ist nicht spezifiziert, was für Inhalte (Datentyp!) in der Liste gespeichert werden. Diese Information wird erst mit der Deklaration einer Variable bzw. des Aufruf des Konstruktors gegeben. Die Liste wird dann typisiert.

```
1 // Initialisierung einer ArrayList mit Datentyp (hier: String) und Name  
2 ArrayList<String> stringList = new ArrayList<String>();
```

VERWENDUNG DER ARRAYLIST (2/3)

```
1 // ArrayList erstellen
2 ArrayList<Ellipse> circles = new ArrayList<Ellipse>();
3
4 // Elemente hinzufügen (ans Ende der Liste)
5 Ellipse circle = new Ellipse(0,0,10,10);
6 circles.add(circle);
7
8
9 // Länge/Anzahl der Elemente feststellen
10 int numCircles = circles.size();
11
12 // Auslesen einzelner Elemente
13 Ellipse circle = circles.get(0);
14
15 // Element an Index 3 entfernen
16 circles.remove(3);
17
18 // Bestimmtes Objekt entfernen
19 Ellipse circle = circles.get(0);
20 circles.remove(circle)
```

VERWENDUNG DER ARRAYLIST (3/3)

`boolean add(E e)`

Appends the specified element to the **end** of this list.

`void add(int index, E e)`

Inserts the specified element at the specified position in this list.

`E remove(int index)`

Removes the element at the specified position in this list.

`boolean remove(Object o)`

Removes the first occurrence of the specified element from this list, if it is present.

`void clear()`

Removes all of the elements from this list.

`int size()`

Returns the number of elements in this list.

`E get(int index)`

Returns the element at the specified position in this list.

`E set(int index, Element e)`

Replaces the element at the specified position in this list with the specified element.

`int indexOf(Object o)`

Returns index of the first occurrence of the element (...), or -1 if this list does not contain the element.

`boolean contains(Object o)`

Returns true if this list contains the specified element.

`boolean isEmpty()`

Returns true if this list contains no elements.

ARRAYLIST: NUR MIT OBJEKten!

- Primitive Datentypen (int, double, boolean, char) sind in einer ArrayList nicht zulässig.
- Um trotzdem solche Daten in der Liste speichern zu können existieren *Wrapper-Klassen*. Instanzen davon kapseln einen primitiven Datentyp in einer Objekt-Repräsentation:

Datentyp	Wrapper-Klasse	Anmerkung
int	Integer	Erzeugung: new Integer(42);
double	Double	
boolean	Boolean	
char	Character	nicht mit characters verwechseln!

WRAPPER-KLASSEN MANUELL UND AUTOMATISCH NUTZEN

```
1 // Erzeugung einer Instanz von Integer
2 int numValue = 5;
3 Integer numObject = new Integer(num);
4
5 // Wert auslesen
6 int numValue = numObject.intValue();
7
8 // Automatisches Boxing und Unboxing
9 ArrayList<Integer> nums = new ArrayList<Integer>();
10
11 int numValue = 5;
12 nums.add(numValue);    // Boxing
13
14 int numValue = nums.get(0); // Unboxing
```

Hinweis: Genau wie Strings sind die *Wrapper-Klassen* Integer, Double, Boolean und Character *immutable!*

ARRAY VS. ARRAYLIST

Beim Implementieren muss abgewogen werden, ob für einen bestimmten Anwendungsfall ein Array oder eine ArrayList sinnvoller ist. Die zentralen Charakteristika der beiden Datenstrukturen können dabei als Entscheidungshilfe dienen.

Arrays: Immer dann verwenden, wenn die nötige (maximale) Größe im Vorfeld bekannt ist!

ArrayLists: Erlauben dynamische Größenveränderung und haben viele nützliche Operationen, die zur Laufzeit ausgeführt werden können, z.B. `contains`, `add`, `remove`, ... Diese Listen sind aber weniger effizient als "normale" Arrays.

KURZSCHREIBWEISE FÜR DIE FOR-SCHLEIFE

Oft möchten Sie (lesend) auf alle Elemente einer ArrayList zugreifen. Statt der normalen for-Schleife gibt es eine abkürzende Schreibweise für diese for each-Schleife:

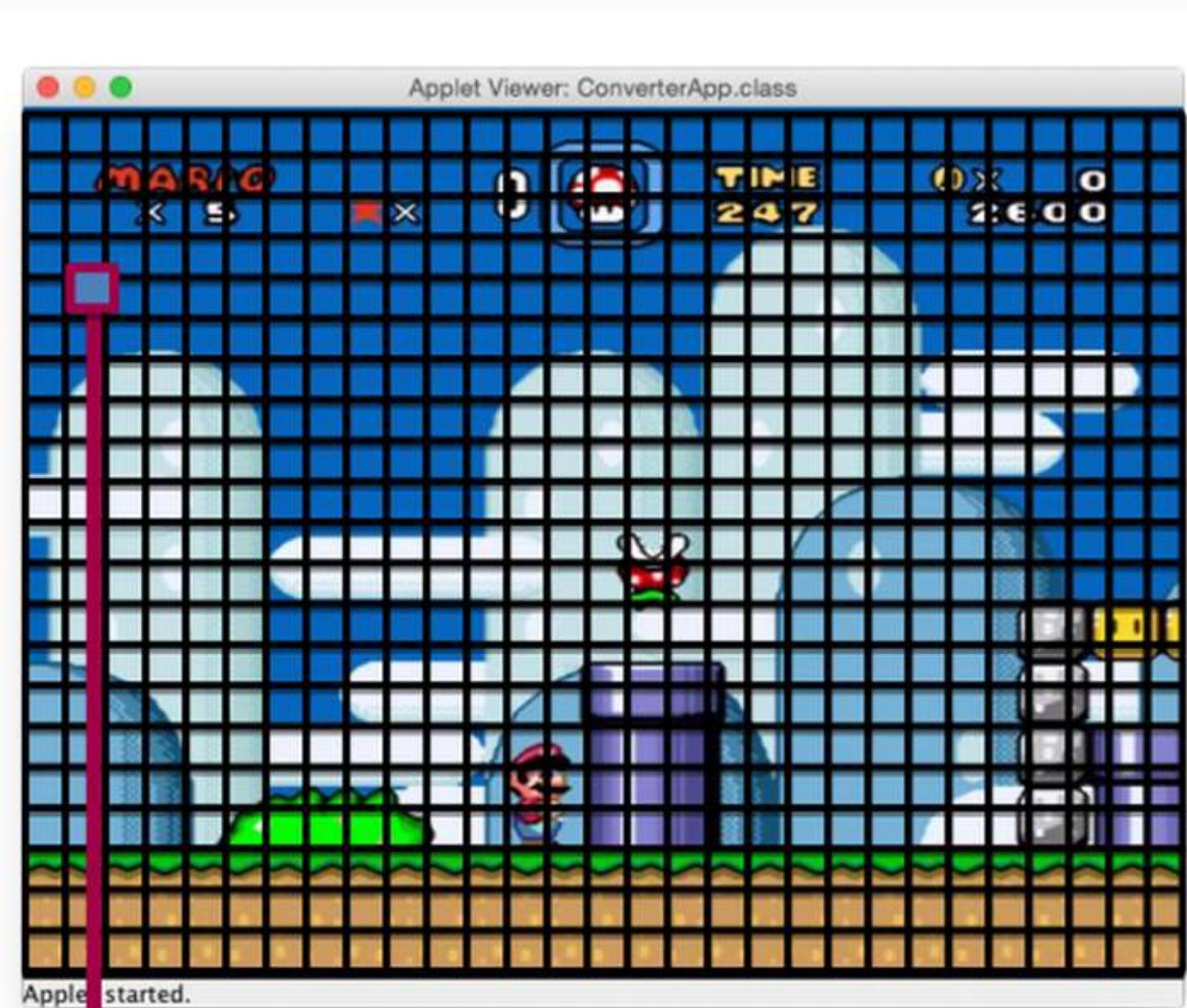
```
1 for(type element: arrayList) {  
2     // Zugriff auf die einzelnen Elemente  
3 }
```

Der Code im Block wird einmal pro Element in der Liste ausgeführt. Bei jeder Iteration ist das jeweils aktuell Element über den im Schleifenkopf vergebenen Variablennamen (hier: element) nutzbar.

Hinweis: Die for each-Schleife kann nur zum lesenden Zugriff auf die ArrayList verwendet werden. D.h. Innerhalb des Schleifenrumpfs können keine Einträge aus der Liste entfernt werden. Dafür können Sie herkömmliche Schleifen oder einen Iterator verwenden.

BILDVERARBEITUNG IN DER GRAPHICSAPP

RASTERDARSTELLUNG VON BILDERN (1/2)



Die Farbe eines Pixels wird durch die Mischung von Rot, Grün und Blau bestimmt. Die jeweilige Farbintensität in dieser Mischung wird durch Werte von 0 bis 255 ausgedrückt.

RASTERDARSTELLUNG VON BILDERN (2/2)

- Bilder werden in der GraphicsApp als Raster dargestellt (Vgl. EIMI).
- Jeder Pixel dieser Bilder hat einen bestimmten Farb/Zahl-Wert, der zur Darstellung auf dem Bild benutzt wird.
- Über die Image-Klasse können diese Pixel als mehrdimensionales Array ausgelesen werden.
- Jeder Pixel wird dabei durch einen kombinierten int-Wert beschrieben, der die Rot-, Grün- und Blau-Werte codiert.
- Die Pixel eines Bildes können mit einem neuen mehrdimensionalen Array überschrieben werden.

ARBEIT MIT DER IMAGE-KLASSE

Nachdem ein Bild erstellt (initialisiert) wurde, können die Pixelwerte ausgelesen oder gesetzt werden:

```
1 // Bild erstellen
2 Image image = new Image("data/image.png");
3
4 // Pixel auslesen
5 int[][] pixels = image.getPixelArray();
6
7 // Farbwert auslesen, Color-Repräsentation erstellen und manipulieren
8 int pixelColor = pixels[5][5];
9 Color color = new Color(pixelColor);
10 int red = color.getRed();
11 color.setGreen(red);
12
13 // Pixel manipulieren und zurück in das Image-Objekt schreiben
14 pixels[5][5] = color.toInt();
15 image.setPixelArray(pixels);
```

ZUSAMMENFASSUNG

- Mehrdimensionale Arrays werden als Arrays aus Arrays behandelt.
- Eine ArrayList bietet dem Client-Programmierer ähnliche Funktionalität wie ein Array und gleicht den Nachteil der festen Länge aus.
- Bilder werden als zweidimensionale Arrays behandelt, jeder Pixel wird durch einen Farbwert repräsentiert.

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT. WENN SIE MÖCHTEN, SEHEN WIR UNS IM ANSCHLUSS IN DER ZENTRALÜBUNG!

Fragen oder Probleme? In allgemeinen Angelegenheiten und bei Fragen zur Vorlesung wenden Sie sich bitte an Alexander Bazo (alexander.bazo@ur.de). Bei organisatorischen Fragen zu den Studienleistungen und Übungsgruppen schreiben Sie bitte Florin Schwappach (florin.schwappach@ur.de). Bei inhaltlichen Fragen zu den Übungen, Beispielen und Studienleistungen schreiben Sie uns unter mioop@mailman.uni-regensburg.de.

QUELLEN

Eric S. Roberts, *The art and science of Java: an introduction to computer science, New international Edition*, 1. Ausgabe, Pearson, Harlow, UK, 2014.