
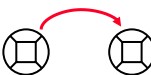
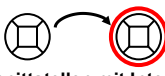



Weihnachten steht vor der Tür...



Level 4: Hinter den Kulissen von Sammlungen

Inhaltliche Gliederung von SE1

Stufe	Titel	Themen u.a.	Woche
1	 „Simple Klasse, simple Objekte“	Klasse, Objekt, Methode, Parameter, Feld, Variable, Sequenz, Zuweisung, Ausdruck, Syntax in EBNF, bedingte Anweisung, primitiver Typ	1 - 4
2	 „Objekte benutzen Objekte“	Typ, Referenz, UML: Klassen- und Objektdiagramme, Schleife, Rekursion, Sichtbarkeit und Lebensdauer, reguläre Ausdrücke	5 - 7
3	 „Schnittstellen mit Interfaces“	Black-Box-Test, Testklasse, Interface, Spezifikation, API, Sammlungen benutzen	8 - 9
4	 „Hinter den Kulissen von Sammlungen“	Arrays, Sammlungen implementieren: Array-Liste, verkettete Liste, Hashing; Sortieren; Stack; Graphen	10 - 14

SE1 - Level 4

3

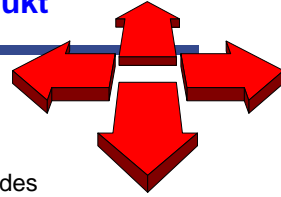
Einige Worte zur Modulprüfung

- Die **Modulprüfung** findet in schriftlicher Form (Klausur) statt.
- Es gibt zwei Klausurtermine für das aktuelle SE1-Modul:
 - 1. Termin: **Mo 6. Februar 2011, 10-12 Uhr**
 - 2. Termin: **Mo 19. März 2011, 10-12 Uhr**
- Dauer jeweils **zwei Stunden**, keine Hilfsmittel außer einem Stift erlaubt.
- Wir führen die Klausur als **Multiple-Choice-Prüfung** durch, d.h., zu jeder gestellten Frage werden mehrere mögliche Antworten vorgegeben.
 - Entweder ist **genau eine Antwort** richtig (meist 1 aus 4)
 - Oder es gibt **n richtige Antworten** ($n > 0$, meist n aus 4 oder n aus 6)
- Eine **Probeklausur** steht im CommSy bereit.
- Die SE1-Klausur ist sehr gut für diejenigen schaffbar, die einen **Übungsschein** haben!

SE1 - Level 4

4

Arrays als speichernahes Sammlungskonstrukt



- Nachdem wir uns die **Benutzung** von Sammlungen exemplarisch angesehen haben, werden wir für den Rest des Semesters untersuchen, wie dynamische Sammlungen **implementiert** werden können.
- Dazu müssen wir uns zuerst eine grundlegende Datenstruktur zur Implementierung von Sammlungen ansehen: das **Array**.
 - Wir sehen uns zuerst die **Deklaration**, **Erzeugung**, **Initialisierung** und **Benutzung** von **Arrays in Java** an.
 - Wir diskutieren dann die allgemeinen Eigenschaften von Arrays.

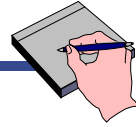
Ein Beispiel

- Vorgabe: ein Temperaturmessgerät in einer hochseetauglichen Boje.
- Sensoren messen den ganzen Tag über immer wieder die Temperatur.
- Der Speicher ist so begrenzt, dass wir nur 1000 Messwerte speichern können.
- Wir wollen
 - die letzten 1000 Messungen speichern,
 - Maximum und Minimum finden.
- Für die Realisierung in Java bietet sich ein Array an.

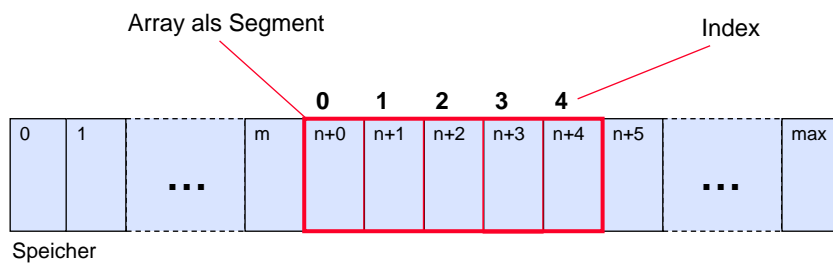


Level 4: Hinter den Kulissen von Sammlungen

Arrays sind ein speichernahes Konzept



- Arrays sind ein klassisches imperatives Sprachkonzept zur **Sammlung gleichartiger Elemente**. Diese Elemente werden über einen **Index** zugegriffen.
- Arrays dienen meist zur Realisierung von **Listen mit fester Größe**.
- Sie abstrahieren von einem **zusammenhängenden Speicherbereich** samt **indiziertem** Zugriff auf die Speicherzellen.



SE1 – Level 4

7

Anwendungsbeispiel: Arrays für Bilddaten



- Bilder sind digital zweidimensionale Strukturen einzelner **Bildpunkte**.
- Die Bildgröße (Bildzeilen und –spalten) ist relativ statisch.
- Informationen über Bildpunkte müssen für Bildverarbeitung **schnell** zugreifbar sein.

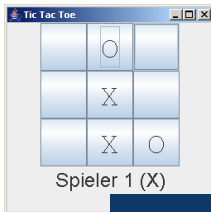
SE1 – Level 4

8

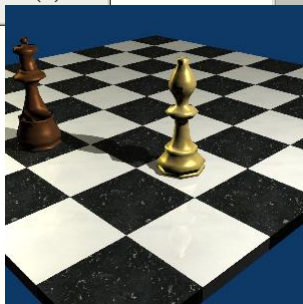
Level 4: Hinter den Kulissen von Sammlungen

Anwendungsbeispiel: Spielfelder fester Größe

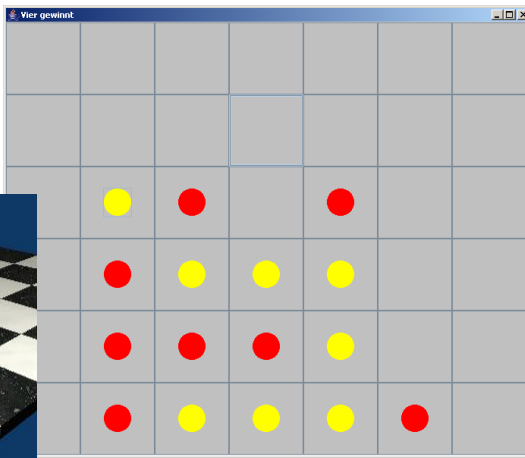
Tic-Tac-Toe: 3x3



Schach: 8x8



Vier gewinnt: 6x7



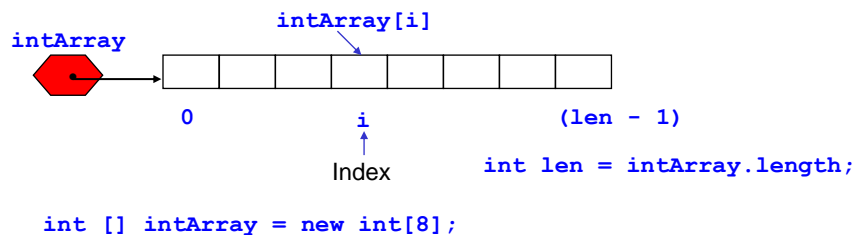
SE1 - Level 4

9

Übersicht: Arrays in Java

Arrays in Java:

- Eine **geordnete Reihung gleichartiger Elemente**.
- Elementtypen können **Basistypen** oder **Referenztypen** sein (auch Referenzen auf andere Arrays).
- Die **Länge eines Array** wird erst beim Erzeugen festgelegt.
- Jeder **Zugriff** über den Index wird **automatisch geprüft**.

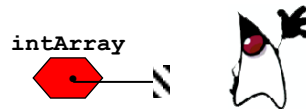


SE1 - Level 4

10

Arrays sind (fast) Objekte: Array-Variablen

- Ein **Array** ist in Java immer ein **Objekt** (Arrays haben alle Eigenschaften, die in der Klasse **Object** definiert sind).
- Eine **Array-Variable** ist immer eine **Referenzvariable**. Der Typ dieser Variablen ist als „**Array von Elementtyp**“ definiert.
- Beispiel einer **Deklaration** eines Arrays von Integer-Werten:
`int[] intArray;`
- Die **Länge/Größe** des Arrays wird in der Deklaration **nicht** angegeben.



Syntax: Array-Deklaration (eindimensional) in Java

Type [] Identifier

Beispiele:

```
int[] numbers; // eine Array-Variable mit primitivem Elementtyp („Array von int“)
```

```
Person[] people; // eine mit einem Objekttyp als Elementtyp („Array von Person“)
```

Erzeugen von Arrays in Java

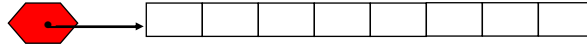
- **Array-Objekte** müssen explizit **erzeugt** werden (wie alle Objekte mit **new**). Dabei kann die Länge definiert oder berechnet werden. Ein einmal erzeugtes Array kann in seiner Länge nicht mehr verändert werden.

```
intArray = new int [8];
```

- Deklaration und Erzeugung können, wie sonst auch, in einem Schritt zusammengefasst werden.

```
int[] intArray = new int [8];
```

intArray



Syntax: Array-Erzeugung (eindimensional) in Java

Wie bei einer „normalen“ Objekterzeugung werden auch Arrays durch einen Ausdruck mit dem Schlüsselwort **new** erzeugt.

```
new Type [ LengthExpression ]
```

Beispiele für Ausdrücke:

```
new int[10]           // Erzeugung eines Arrays mit primitivem Elementtyp
```

```
new Person[x]        // Erzeugung eines Arrays mit einem Objekttyp als Elementtyp
```

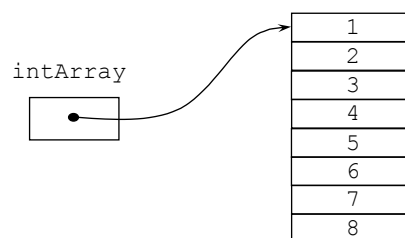
Level 4: Hinter den Kulissen von Sammlungen

Initialisieren von Array-Zellen in Java

- Beim Erzeugen erhalten die **Zellen** eines Arrays in Java die Default-Werte des Elementtyps (für **int** den Wert 0, für **boolean** den Wert **false**, für Referenztypen den Wert **null**, etc.).
- Neben der normalen Zuweisung von Werten kann ein Array auch **implizit erzeugt** und **direkt initialisiert** werden.

```
int[] intArray = {1,2,3,4,5,6,7,8};
```

Diese implizite Erzeugung und Initialisierung mit **geschweiften Klammern** ist ausschließlich bei der Deklaration erlaubt!



SE1 – Level 4

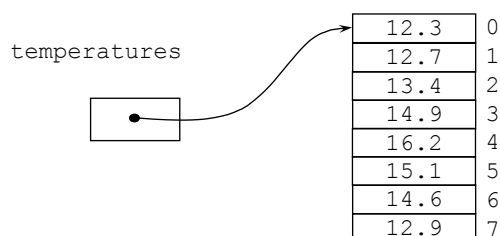
15

Indizierung

- Arrays werden **beginnend bei 0** indiziert!!!
- Gültige Indizes eines Arrays sind: 0 ... length - 1
- Beispiel: Ein Array der Größe 8 hat als gültige Indizes 0..7

Gültige Namen für die Elemente des Arrays sind:

- `temperatures[0]`
- `temperatures[1]`
- `temperatures[2]`
- `temperatures[3]`
- ...



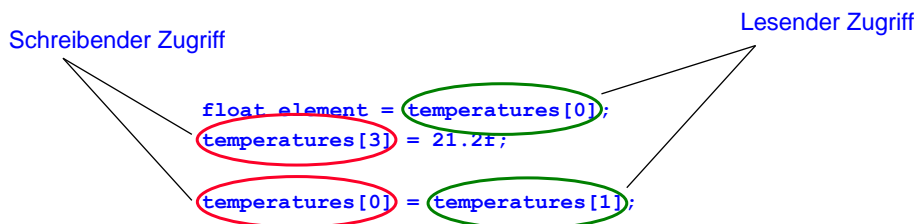
SE1 – Level 4

16

Level 4: Hinter den Kulissen von Sammlungen

Schreibender und lesender Zugriff auf Array-Zellen

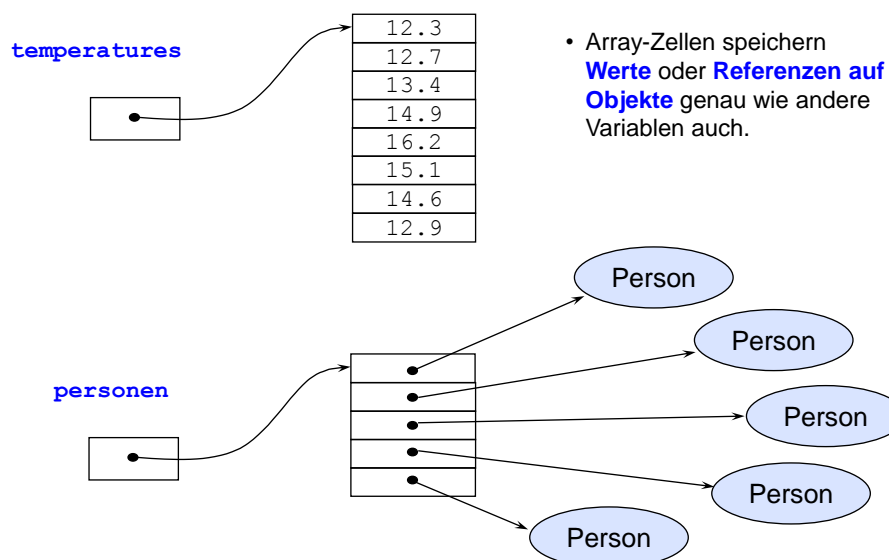
- Auf die Array-Zellen wird mit eckigen Klammern `[]` zugegriffen:



SE1 - Level 4

17

Werte vs. Objekte als Elemente



SE1 - Level 4

18

Typischer Fehler: Ungültiger Index (Out of Bounds)

- Angenommen, wir haben ein Array der Größe 8.
- Dann schreiben wir:

```
System.out.println(temperatures[8]);
```

temperatures



12.3	0
12.7	1
13.4	2
14.9	3
16.2	4
15.1	5
14.6	6
12.9	7

- Es kommt zu einer Fehlermeldung:

ArrayIndexOutOfBoundsException: 8

Typischer Fehler: Array-Objekt nicht erzeugt

- Angenommen, wir haben ein Array als Exemplarvariable deklariert:
- Im Konstruktor schreiben wir als Erstes:
- Es kommt zu einer Fehlermeldung:

```
private int[] _numbers;
```

```
_numbers[0] = 42;
```

NullPointerException

_numbers

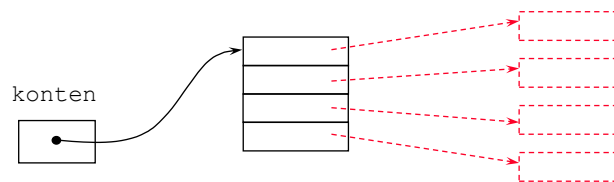


- Es fehlt die **Erzeugung** des Arrays:

```
_numbers = new int[8];
```

Typischer Fehler: Array-Inhalt nicht erzeugt

- Angenommen, wir haben ein Array mit einem Objekttyp als Elementtyp deklariert und initialisiert:
`Konto[] konten = new Konto[4];`
- Unmittelbar danach schreiben wir:
`konten[0].einzahlen(123);`
- Es kommt zu einer Fehlermeldung:
`NullPointerException`



- Es fehlt die **Erzeugung der Elemente** des Arrays, also der Konto-Objekte.

For-Schleifen und Arrays

- Typischerweise werden For-Schleifen eingesetzt, um alle Elemente eines Arrays zu bearbeiten. Dabei wird die **öffentliche Exemplarkonstante** `length` benutzt.
- Beispiel: Das Ausgeben der Werte eines Arrays.

```
public void printArray(int[] intArray)
{
    for (int i = 0; i < intArray.length; ++i)
    {
        System.out.println(intArray[i]);
    }
}
```

Eine solche Standardbenutzung einer For-Schleife für Arrays wird auch als **Programmiermuster** bezeichnet. Im Englischen wird oft der Begriff **idiom** verwendet.

Die erweiterte For-Schleife für Arrays

- Die erweiterte For-Schleife kann auch für Arrays verwendet werden. Dies erspart den Zugriff auf `length`.
- Gleiches Beispiel: Das Ausgeben der Werte eines Arrays.

```
public void printArray(int[] intArray)
{
    for (int k : intArray)
    {
        System.out.println(k);
    }
}
```

Lies auch hier: für jeden int k
im intArray tue...

Diese Schleifenart ist nicht geeignet, wenn
am Array selbst (also an der Belegung der
Zellen) etwas geändert werden soll.
Außerdem steht im Schleifenrumpf **kein
Schleifenindex** zur Verfügung.

Beispiel: Den minimalen Wert finden

```
/**
 * Liefere den minimalen Wert im gegebenen Array.
 */
public int findeMinimum(int[] intArray)
{
    int min = Integer.MAX_VALUE;
    for (int i=0; i < intArray.length; i++)
    {
        if (intArray[i] < min)
        {
            min = intArray[i];
        }
    }
    return min;
}
```

Alternativ: neue For-Schleife

```
{
    int min = Integer.MAX_VALUE;
    for (int k : intArray)
    {
        if (k < min)
        {
            min = k;
        }
    }
    return min;
}
```

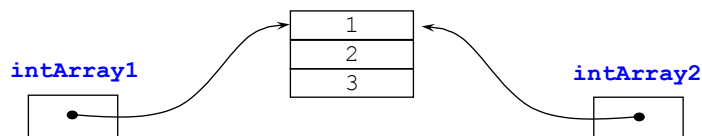
Benutzung:

```
int[] myArray = new int[10];
myArray[0] = 20;
myArray[1] = 40;
myArray[2] = 30;
int mini = findeMinimum(myArray);
System.out.println(mini);
```

Zuweisungen mit Arrays

- Die Zuweisung einer Array-Variablen kopiert **nur eine Referenz!**
- Beispiel:

```
int[] intArray1 = { 1, 2, 3 };  
int[] intArray2 = intArray1;
```
- Beide Referenzen verweisen nun auf **dasselbe Array-Objekt**:



- Wenn man eine **Kopie des Array-Objektes** benötigt, dann gibt es zwei Möglichkeiten:
 - Elementweise in ein neues Array-Objekt kopieren.
 - Die Operation `clone` verwenden.

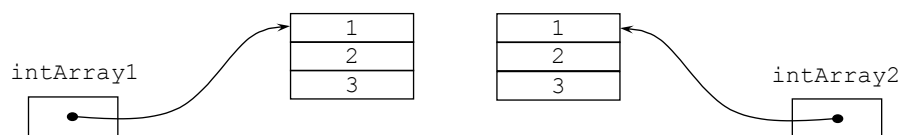
Kopieren von Array-Objekten

- Neues Array-Objekt selbst erzeugen und elementweise kopieren:

```
int[] intArray1 = { 1, 2, 3 };  
int[] intArray2 = new int[intArray1.length];  
for (int i=0; i < intArray1.length; i++)  
{  
    intArray2[i] = intArray1[i];  
}
```

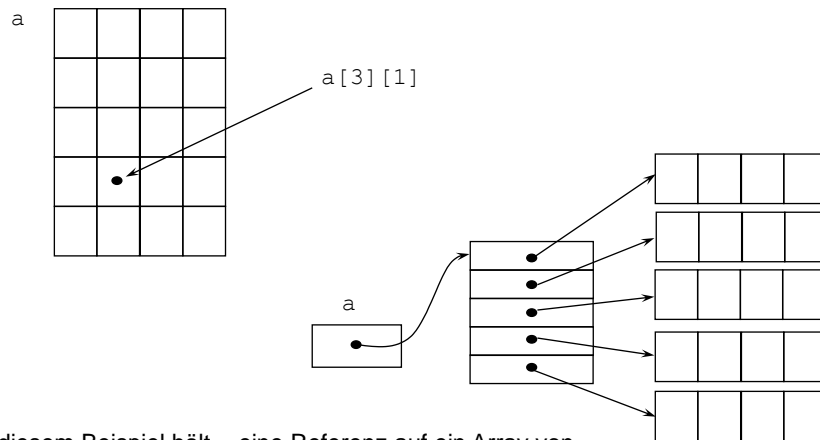
- Die Operation `clone` verwenden:

```
int[] intArray1 = { 1, 2, 3 };  
int[] intArray2 = intArray1.clone();
```



Zweidimensionale Arrays

Zweidimensionale Arrays in Java sind Arrays von eindimensionalen Arrays.



In diesem Beispiel hält **a** eine Referenz auf ein Array von Zeilen; der erste Index benennt die Zeile, der zweite die Spalte.

SE1 – Level 4

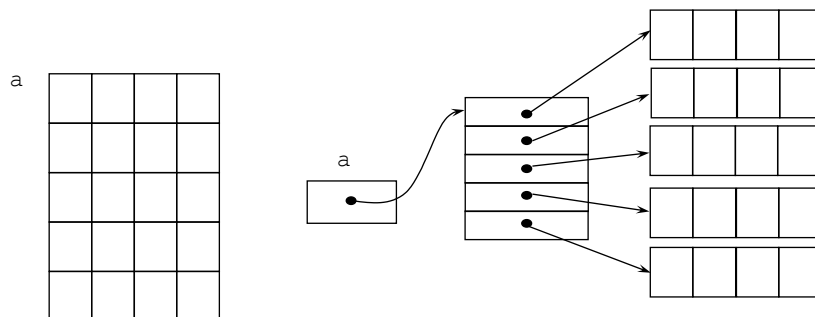
27

Zweidimensionale Arrays erzeugen

```
int[][] a;  
a = new int[5][];  
for (int i=0; i<5; i++)  
{  
    a[i] = new int[4];  
}
```

Mit **new** ist es auch möglich, ein zweidimensionales Arrays direkt zu erzeugen.

Also: **new int[5][4]**



SE1 – Level 4

28

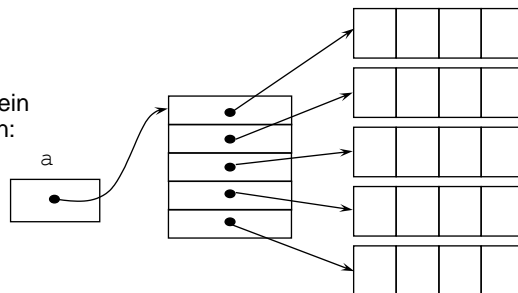
Zugriff auf zweidimensionale Arrays

```
// Alle Array-Elemente auf den Wert 7 setzen

for (int row=0; row < a.length; row++)
{
    for (int column=0; column < a[row].length; column++)
    {
        a[row][column] = 7;
    }
}
```

Mit Hilfe von Initializern kann auch direkt ein zweidimensionales Array angelegt werden:

```
int[][] a = { {7,7,7,7},
               {7,7,7,7},
               {7,7,7,7},
               {7,7,7,7},
               {7,7,7,7} };
```

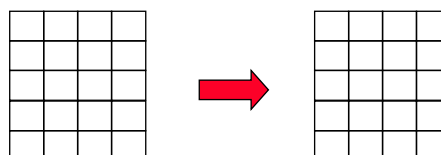


SE1 – Level 4

29

Kopieren von mehrdimensionalen Arrays

- Bei mehrdimensionalen Arrays ist es prinzipiell wie bei eindimensionalen:
 - Es kann „von Hand“ kopiert werden, indem für jede Dimension die notwendigen Arrays erzeugt und deren Inhalte kopiert werden.
 - Oder es wird die Operation `clone` benutzt. Dabei ist zu beachten:
 - `clone` ist für Arrays nicht rekursiv implementiert!
 - D.h., bei einem Aufruf von `clone` auf einer Array-Variablen für ein mehrdimensionales Array wird nur das Array auf der obersten Ebene kopiert.
 - Wenn eine vollständige Kopie gewünscht ist, dann muss „von Hand“ für alle Dimensionen geklont werden.
- In jedem Fall muss die Objektstruktur von Arrays gut verstanden sein!



SE1 – Level 4

30

Vorteile von Arrays

- Aufgrund der speichernahen Modellierung eines zusammenhängenden Speicherbereichs haben Arrays vor allem **Effizienzvorteile**:
 - Der Zugriff auf ein Element kann direkt auf einen **Index-Zugriff** der unterliegenden Rechnerarchitektur abgebildet werden; dies ist sehr effizient.
 - Auch das **Kopieren eines ganzen Arrays** kann direkt auf unterliegende Maschinenbefehle abgebildet werden und ist damit sehr effizient.
- **Arrays in Java** haben gegenüber den dynamischen Sammlungen außerdem den Vorteil, dass sie **auch elementare Typen** als Elementtyp zulassen.



Nachteile von Arrays

- In klassischen imperativen Sprachen (wie Pascal) ist die Größe eines Arrays bereits **zur Übersetzungszeit festgelegt**. Dies muss kein Nachteil sein, denn bei etlichen Anwendungen ist die Größe einer Datenstruktur fachlich festgelegt (Beispiel: **Schachbrett**).
- Für ein Array in Java muss die Größe erst zur Laufzeit festgelegt werden; dennoch bilden Java-Arrays gegenüber dem Typ **List** aus dem JCF eine schwächere Realisierung des Listenkonzeptes:
 - Ein Array, einmal erzeugt, hat eine **feste Maximalkapazität**.
 - Auf einem Array gibt es außer dem indizierten Zugriff **keine höherwertigen Operationen** wie beispielsweise (zwischen zwei Elementen) **Einfügen**, **Entfernen** (mit Aufrücken), am Ende **Anfügen** oder den **Test auf Enthaltensein**.



Vergleich (dynamische) Sammlung und Array

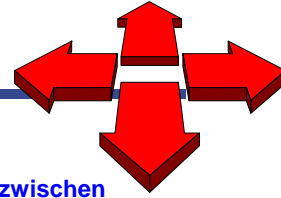
- Die Java-Sammlungen wie **List** oder **Set** sind beschränkt auf **Sammlungen von Referenzen**, d.h. es können nur Referenztypen als Elementtypen definiert werden; Arrays erlauben hingegen beide Typfamilien als Elementtyp.
- Nach seiner Erzeugung kann ein Array seine **Größe nicht mehr verändern**; **List** und **Set** hingegen sind dynamische Sammlungen und können **beliebig viele Elemente** aufnehmen.
- Ein Array modelliert **zusammenhängende Speicherzellen**; der schreibende Zugriff auf eine Position verschiebt nicht alle nachfolgenden Elemente, sondern ersetzt das Element an der Position. Ein Einfügen muss durch ein Verschieben „von Hand“ realisiert werden.
- Arrays werden in der Implementierung von dynamischen Sammlungen verwendet; sie stehen auf einem **niedrigeren Abstraktionsniveau**.

Zusammenfassung



- Ein **Array** ist eine elementare imperative Datenstruktur, die sehr speichernah konzipiert ist.
- Arrays sind geordnete Reihungen gleichartiger Elemente, auf die über einen Index zugegriffen wird.
- Arrays stehen auf einem **niedrigeren Abstraktionsniveau** als Listen und Mengen.
- Für Java gilt:
 - Die Elemente können von elementarem Typ oder Referenztypen sein (auch Referenzen auf andere Arrays).
 - Die Größe eines Arrays wird erst beim Erzeugen festgelegt.
 - Jeder Zugriff über den Index wird zur Laufzeit geprüft.

Klassen und Objekte – revisited



- Im klassischen objektorientierten Modell ist der **Unterschied zwischen Klasse und Objekt** klar:
 - **Klassen** sind die Einheiten des **statischen Programmtextes**. Sie beschreiben die Erzeugung und das Verhalten von Objekten.
 - **Objekte** sind die Einheiten des **laufenden Programms**. Sie haben einen veränderbaren Zustand und ein prinzipiell festgelegtes Verhalten. Alle Operationen beziehen sich immer auf ein Objekt.
- Wenn **Klassen** selbst vollständig **im Laufzeitsystem verfügbar** sind, verschiebt sich diese klare Unterteilung:
 - Über den Zugriff auf eine Klasse kann zur Laufzeit das Verhalten ihrer Objekte verändert werden.
 - Klassen werden zu eigenständigen Objekten mit einem eigenen Zustandsraum.
- Java wählt einen „Mittelweg“ zwischen diesen beiden Positionen.

Klassen in Java sind auch selbst Objekte



- Klassen in Java definieren nicht nur das Verhalten und die Struktur ihrer Exemplare zur Laufzeit; sie existieren auch selbst als Objekte zur Laufzeit.
- Ein solches **Klassenobjekt** kann wie alle Objekte einen Zustand haben (über **Klassenvariablen**) und Methoden anbieten (**Klassenmethoden**).
- Klassenvariablen und Klassenmethoden werden mit dem Modifikator **static** deklariert.

```
class Konto
{
    private static int exemplarzaehler = 0;

    public Konto()
    {
        exemplarzaehler++;
    }

    public static int anzahlErzeugteExemplare()
    {
        return exemplarzaehler;
    }
}
```

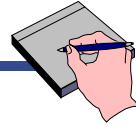
Klassenvariable

Auch hier gilt: Klassenvariablen sollten privat deklariert werden.

Klassenmethode



Klassenoperationen



- Die öffentlichen Klassenmethoden bilden die Operationen eines Klassenobjektes.
- Die Operationen eines Klassenobjektes sind für Klienten in der Punktnotation aufrufbar:

`<Klassenname>.<Klassenoperation> (<aktuelle Parameter>);`



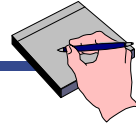
```
class Kontenverwalter
{
    public void statusPruefen()
    {
        int anzahlKonten = Konto.anzahlErzeugteExemplare();
        ...
    }
}
```

Klassenoperationen als Dienstleistungen

- Die statischen Methoden in Java werden häufig zur Realisierung von (einfachen) **Dienstleistungen** benutzt, die sich nicht auf den Zustand des gerufenen Klassenobjekts beziehen, sondern **ausschließlich auf den übergebenen Parametern** arbeiten.
- Vorteil: Zum Abrufen dieser Dienstleistungen muss kein Exemplar einer Klasse erzeugt werden; das Klassenobjekt steht unmittelbar zur Verfügung.
- Beispiele:
 - Die Klasse **Arrays** aus dem Paket **java.util**, die ausschließlich statische Methoden anbietet, mit denen Arrays manipuliert werden können. Alle Arrays werden dabei als Parameter an die Methoden übergeben.
 - Die Klassenoperation **arraycopy** in der Klasse **java.lang.System**. Sie bietet eine dritte Möglichkeit zum Kopieren von Array-Inhalten (neben dem expliziten Traversieren und **clone**).
 - Mathematische Funktionen in **java.lang.Math**.



Die spezielle Klassenoperation main



- Eine Klasse kann eine Klassenmethode mit einer ganz speziellen Signatur anbieten:

```
public static void main(String[] args)
```



- Die auf **genau diese Weise** definierte Klassenoperation wird in der Laufzeitumgebung von Java gesondert behandelt: Sie bildet die Schnittstelle zum Betriebssystem, indem nur genau sie von außerhalb der Java-Welt aufgerufen werden kann.
- Sie bildet somit den **Einstiegspunkt für Java-Programme**: In dieser Methode werden üblicherweise die ersten Exemplare erzeugt, mit denen eine Java-Anwendung gestartet wird.
- Die Möglichkeit, beliebige Objekte interaktiv erzeugen und manipulieren zu können, wird ausschließlich von BlueJ geboten. Andere IDEs bieten einen **Startknopf**, mit dem eine main-Methode aufgerufen wird.

Ein weiteres Mysterium entzaubert: System.out

- Wir haben bisher Ausgaben auf die Konsole mit Anweisungen der Form

```
System.out.println("Hello World!");
```

vorgenommen.



- Diese ungewöhnliche Anweisung ist nun etwas besser erklärbar: Die Klasse java.lang.System verfügt über eine öffentliche Klassenkonstante **out**.
- Diese Konstante ist vom Typ **PrintStream** und somit eine **konstante Referenz** auf ein Exemplar der Klasse **java.io.Printstream**.
- Ein **PrintStream** ermöglicht mit seinen Operationen (u.a. **println**) die Ausgabe von Zeichenströmen.

Initialisierung von Klassenobjekten in Java

- Jede Klasse in Java definiert nur genau ein Klassenobjekt.
- Dieses Klassenobjekt wird nicht über einen Konstruktoraufruf erzeugt, sondern wird automatisch erzeugt, sobald eine Klasse in die Virtual Machine geladen wird (weil eine Klassenmethode aufgerufen wird oder weil ein Exemplar der Klasse erzeugt wird).
- Es gibt deshalb auch keine aufrufbaren Konstruktoren für Klassenobjekte. In einer Klassendefinition können aber **Klassen-Initialisierer** angegeben werden, die nach dem Laden der Klasse ausgeführt werden.

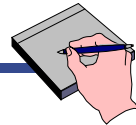


```
class Konto
{
    static {
        exemplarzaehler = 42;
        ...
    }
}
```

SE1 - Level 4

41

Klassenkonstanten



- Auch Konstanten (gekennzeichnet durch den Modifikator **final**) können mit dem Modifikator **static** deklariert werden.
- Sie werden dadurch zu **Klassenkonstanten**.
- Klassenkonstanten werden öffentlich (**public**) deklariert, wenn sie als globale Konstanten dienen sollen.
- Beispiele:

```
public static final int TAGE_PRO_WOCHE = 7;
public static final float PI = 3.141592654f;
public static final int ANZAHL_SPALTEN = 80;
```



Hinweis zur Pragmatik: Fast immer sollten im Quelltext solche **symbolischen Konstanten** verwendet werden, anstatt an allen benutzenden Stellen jeweils das gewünschte Literal direkt anzugeben.

SE1 - Level 4

42

Nicht alle Objekte sind Exemplare



- Nach der Einführung von Klassenobjekten können wir eine Unterscheidung zwischen **Exemplar** und **Objekt** für Java vornehmen:
 - Alle Exemplare einer Klasse sind Objekte.
 - Auch eine Klasse ist ein Objekt, sie ist aber in Java nicht das Exemplar einer weiteren Klasse (es gibt keine so genannten „Metaklassen“).
 - Exemplare werden explizit mit **new** erzeugt, während Klassen automatisch geladen und initialisiert werden, sobald sie benutzt werden.

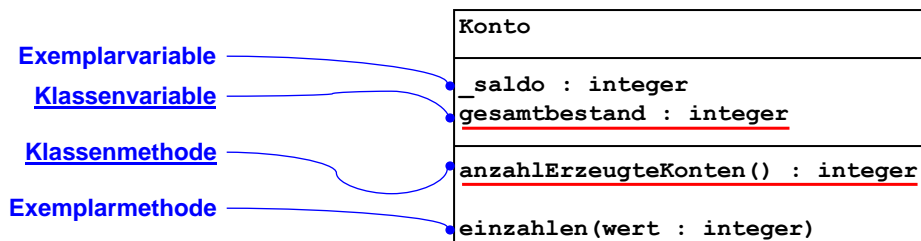


Die Rolle der Klasse in anderen Sprachen



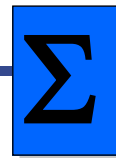
- Eine Klasse wird generell verstanden als eine Schablone, ein Bauplan für Objekte.
- **Smalltalk** geht weiter als Java: In Smalltalk ist eine Klasse selbst wieder ein Exemplar einer Metaklasse. Es ist in Smalltalk sogar möglich, die Methoden einer Klasse „im laufenden Betrieb“ zu verändern.
- In **Eiffel** ist eine Klasse hingegen ein Konzept der Organisation von Programmtexten und zu deren Übersetzung. Zur Laufzeit gibt es kein Klassenobjekt.
- Es gibt auch objektorientierte Programmiersprachen ohne eingebautes Klassenkonzept (**Self**, **Cecil**). In diesen so genannten **exemplarbasierten Sprachen** werden Einzelobjekte definiert, weitere Objekte können durch **Klonen** bereits bestehender Objekte erzeugt werden.

Klassenmethoden und -variablen in der UML-Notation



Klassenvariablen und Klassenmethoden werden in den Klassen-Diagrammen der UML **unterstrichen**, um sie von Exemplarvariablen und -methoden zu unterscheiden.

Zusammenfassung



- **Klassen** sind in verschiedenen Sprachen sehr unterschiedlich modelliert.
- In Java sind **Klassen auch Objekte** mit einem eigenen Zustand, der zur Laufzeit verändert werden kann; die dazu notwendigen **Klassenvariablen** werden mit dem Schlüsselwort **static** deklariert.
- Die Operationen eines Klassenobjektes werden mit **Klassenmethoden** realisiert, die ebenfalls mit dem Schlüsselwort **static** deklariert werden. Für ihren Aufruf muss kein Exemplar erzeugt werden.
- Öffentliche **Klassenkonstanten** (**public static final**) werden verwendet, um global gültige Konstanten zu definieren.

Level 4: Hinter den Kulissen von Sammlungen

Entspannte Weihnachtstage und einen guten Rutsch!

SE1 - Level 4

47