

Kapitel 8

Arrays

Ziele

- Die Datenstruktur der Arrays kennenlernen
- Grundlegende Algorithmen auf Arrays in Java implementieren können
- Mit Arrays von Objekten arbeiten können

Erweiterungen zur Behandlung von Arrays: Überblick

Bisher

Klassendeklarationen

Objekte und Objekthalde (Heap)

Grunddaten- und Klassentypen

Werte

Operationen

Ausdrücke

Typisierung

Auswertung bzgl.

Zustand (Stack + Heap)

Deklarationsanweisung

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

Kapitel 8

Arrayobjekte (kurz: Arrays)

Arraytypen

Referenzen auf Arrayobjekte

==, != für solche Referenzen

Arrayzugriff, Arrayerzeugung

Arrayobjekte auf dem Heap

Arrayinitialisierung

Arrays: Einführung (1)

In vielen Anwendungen werden Tupel (Reihungen/Folgen von Elementen) verschiedener Länge benutzt.

Beispiel: ~~Vektoren~~ *Zahlenfolgen*

- [1.0, 1.0] *Länge 2*
- [0.2, 1.2, 7.0] *Länge 3*

Beispiel: Zeichenfolgen

- ['L', 'M', 'U'] *Länge 3*
- ['C', 'A', 'M', 'P', 'U', 'S']

Solche Tupel können durch Arrays dargestellt werden.

Arraytypen und Arrayobjekte

Type = *PrimitiveType* | *ClassType* | ***ArrayType*** (← neu)

ArrayType = *Type* "[]"

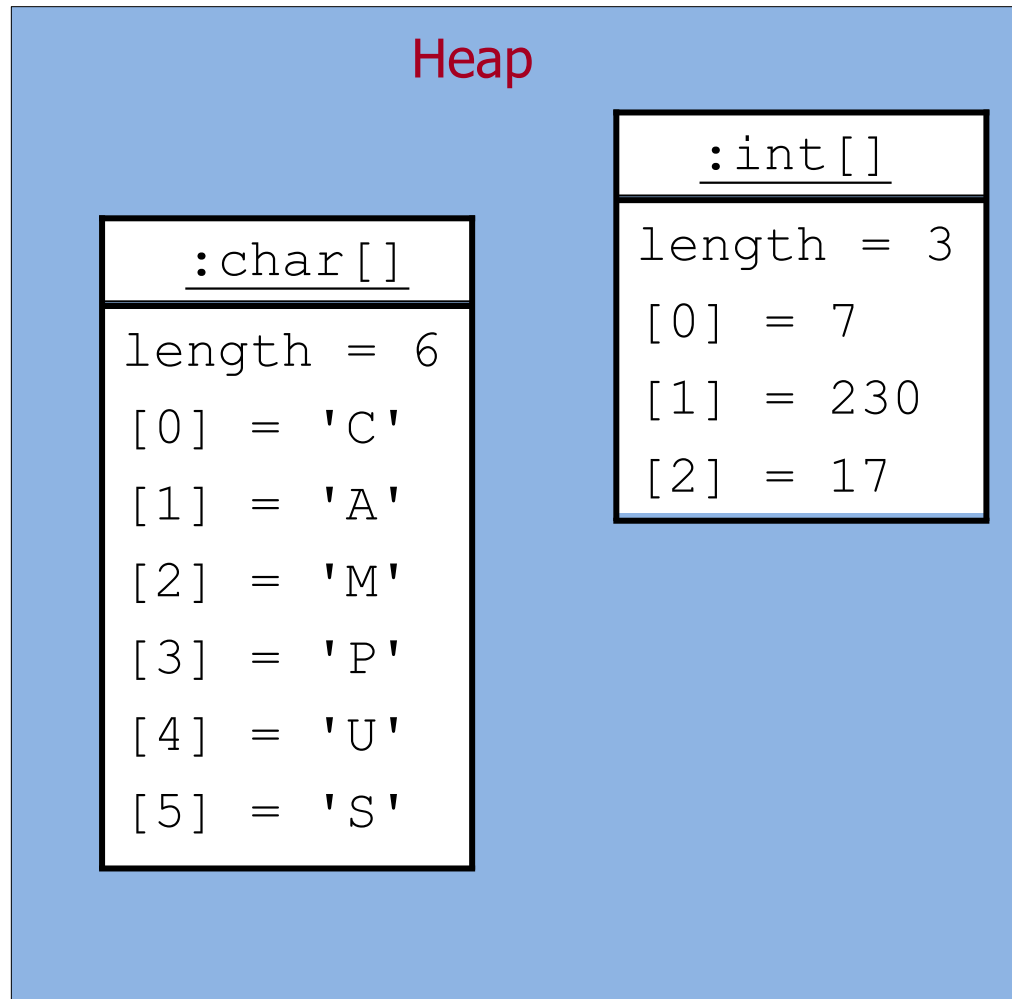
Beispiele:

```
int[], boolean[], char[], double[], String[], Point[],  
double[][] //2-dimensionale Arrays mit double-Werten  
Point[][][][] // 4-dimensionale Arrays von Punkten
```

Arrayobjekte eines Arraytyps **T[]** besitzen

- ein unveränderbares Attribut `length`, das die Anzahl n der Komponenten des Arrays angibt, und
- eine der Reihe nach angeordnete Folge von n Elementen des Typs **T**, die mit den Indizes $0, \dots, n-1$ durchnummeriert sind.

Speicherdarstellung von Arrayobjekten



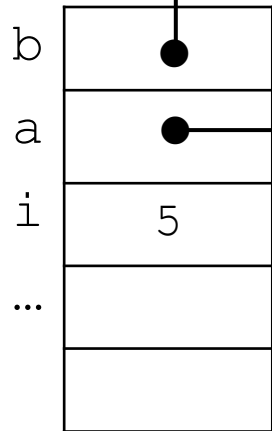
Werte von Arraytypen

- Die **Werte** eines Arraytyps `T[]` sind **Referenzen** auf Arrayobjekte des Typs `T[]` sowie (wie bei Klassentypen) die leere Referenz `null`.
- Dementsprechend speichern lokale Variablen eines Arraytyps Referenzen auf ein Arrayobjekt oder den Wert `null`.
- Array-Referenzen können mit den Operationen `==` und `!=` auf Gleichheit bzw. Ungleichheit getestet werden (nicht empfehlenswert, da die Referenzen und nicht die einzelnen Komponenten der Arrays verglichen werden).

Zustand mit Arrays

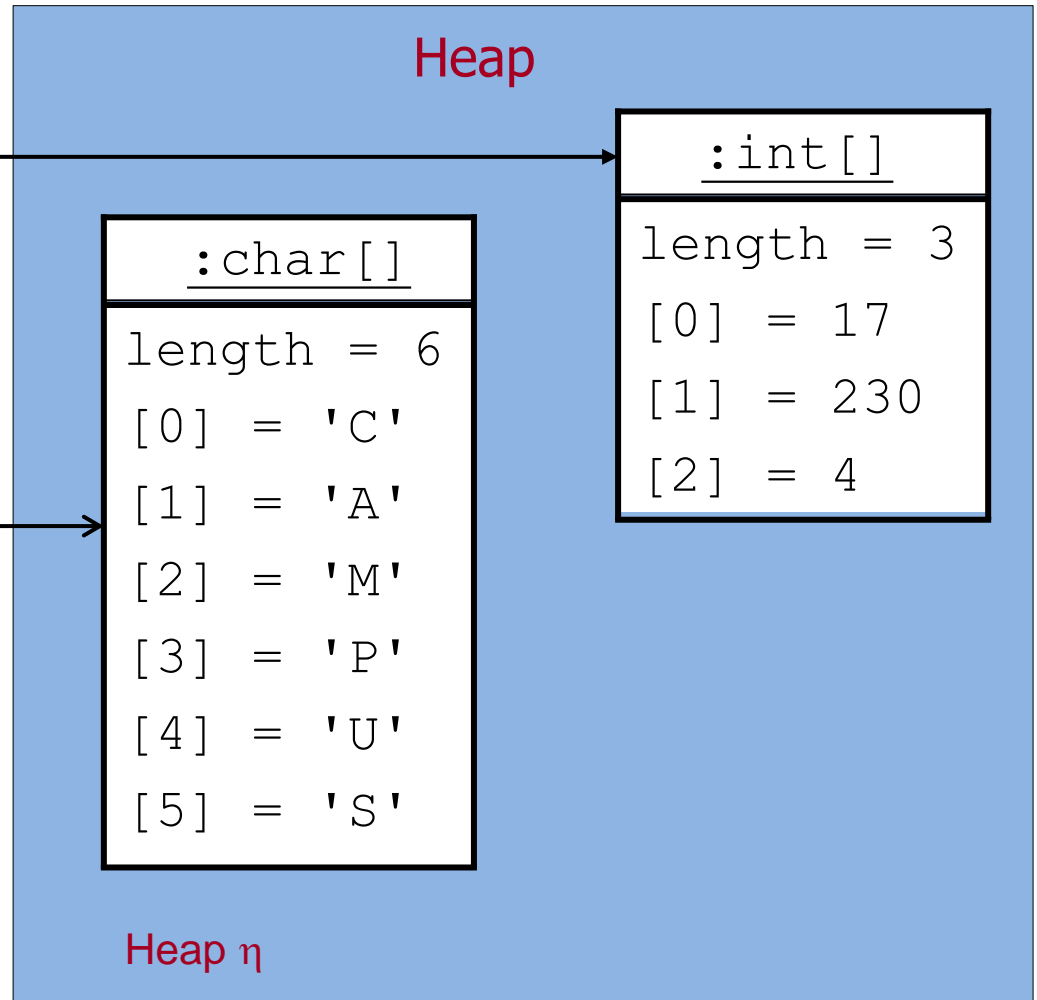
b.length = (b, n) 3
b[1] = (b, n) 230
b[b.length - 1] = (b, n) 4

```
int[] b;
char[] a;
int i;
...
```



Lokale Variablen

Stack σ



Heap η

Erweiterungen zur Behandlung von Arrays (Wdh.)

Bisher

Klassendeklarationen

Objekte und Objekthalde (Heap)

Grunddaten- und Klassentypen

Werte

Operationen

Ausdrücke

Typisierung

Auswertung bzgl.

Zustand (Stack + Heap)

Deklarationsanweisung

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

Kapitel 8

Arrayobjekte (kurz: Arrays)

Arraytypen

Referenzen auf Arrayobjekte

==, != für solche Referenzen

Arrayzugriff, Arrayerzeugung

Arrayobjekte auf dem Heap

Arrayinitialisierung

Grammatik für Ausdrücke mit Arrays

Expression = *Variable* | *Value* | *Expression BinOp Expression* |
UnOp Expression | "(" *Expression* ")" |
MethodInvocation | *InstanceCreation*

Variable = *NamedVariable* | *FieldAccess* |
ArrayAccess (← neu)

NamedVariable = *Identifier*

FieldAccess = *Expression* "." *Identifier*

ArrayAccess = *Expression* "[" *Expression* "]" (← neu)

Value = *IntegerValue* | *FloatingPointValue* | *CharacterValue* | *BooleanValue* | "null"

Grammatik für Methodenaufruf, Objekt- und Arrayerzeugung

MethodInvocation =

Expression "." *Identifer* "(" [*ActualParameters*] ")" "

ActualParameters = *Expression* {"," *Expression*}

InstanceCreation = *ClassInstanceCreation* | ***ArrayCreation*** (← neu)

ClassInstanceCreation =

"new" *ClassType* "(" [*ActualParameters*] ")" "

ArrayCreation = "new" *Type* *DimExprs* {"[" "]"} (← neu)

DimExprs = "[" *Expression* "]" {"[" *Expression* "]" }

Typ und Auswertung der Array-Ausdrücke

- Ein Ausdruck ist, wie bisher, **typkorrekt**, wenn ihm ein Typ zugeordnet werden kann.
- Die **Auswertung** eines Ausdrucks e erfolgt (weiterhin) unter einem **Zustand** (σ, η) , d.h. wir berechnen $e \stackrel{=}{=}_{(\sigma, \eta)} \dots$
- Der Arrayzugriff " $[]$ " hat (wie der Attributzugriff ".") die höchste Präzedenz 15.

Wir bestimmen nun Regeln für Typkorrektheit und Auswertung für die ²neu hinzugekommenen Array-Ausdrücke.

Arrayzugriff: Typkorrektheit

$ArrayAccess = Expression \text{ "[" } Expression \text{ "]"}$

~~7[4]~~

- Der erste *Expression*-Ausdruck (Array-Referenzausdruck) muss einen Arraytyp $T[]$ haben und der zweite *Expression*-Ausdruck (Indexausdruck) muss den Typ `int` (oder einen kleineren Typ) haben.
- ArrayAccess* hat dann den Typ T der Arrayelemente.

~~a[A.3]~~

Beispiel:

Seien char[] a; int[] b; double[][] c; int i, j; lokale Variable.

a[3], a[i], a[-8+2*i], letztes Element a[a.length-1], a[b[i]-3], a[b[i-3]]
haben den Typ char.

b[0], b[a.length], b[b[i]+7] haben den Typ int.

c[i] hat den Typ double[], c[i][j] hat den Typ double.

Arrayzugriff: Auswertung

Sei $e[a]$ ein Arrayzugriffs-Ausdruck.

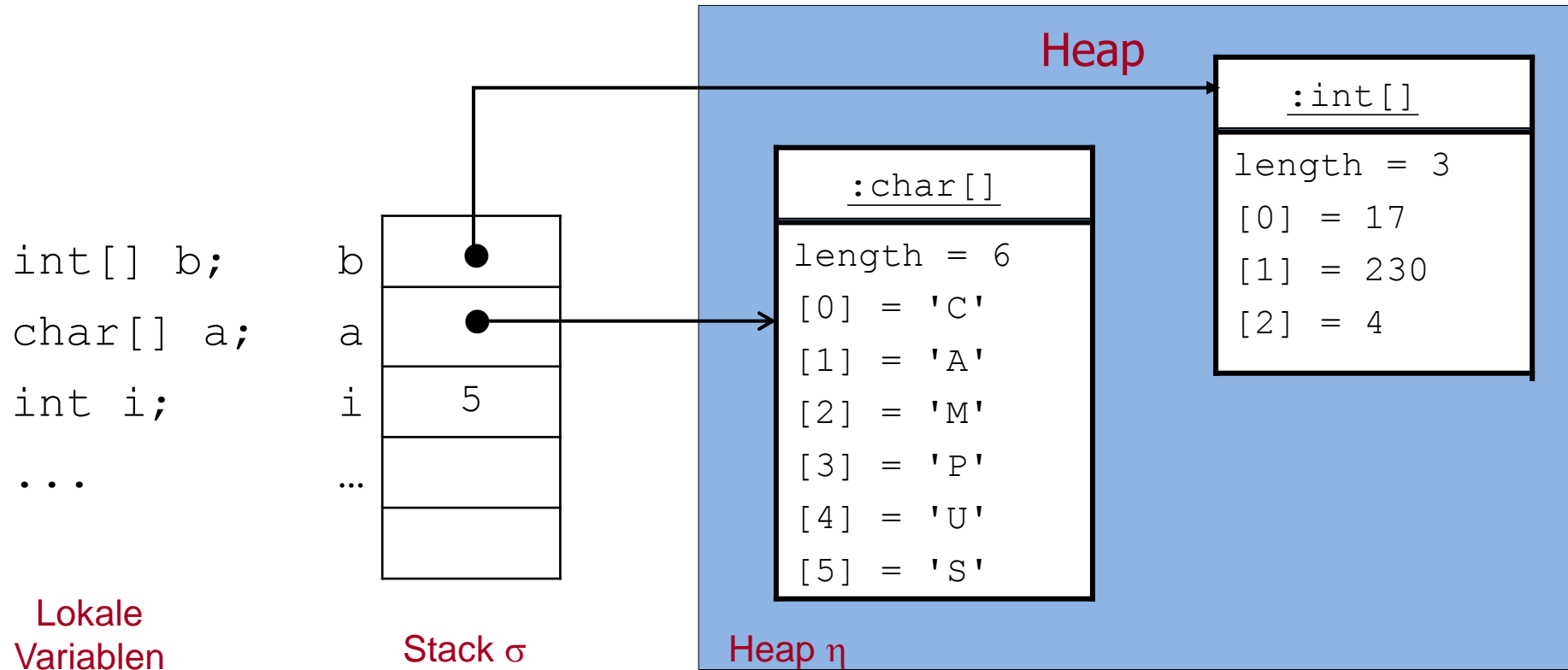
Sei e ein Ausdruck mit Arraytyp $T[]$.

1. Der Referenzausdruck e wird im aktuellen Zustand (σ, η) ausgewertet.
Falls der Wert `null` ist, erfolgt ein Laufzeitfehler (`NullPointerException`),
andernfalls wird die erhaltene Arrayreferenz p gemerkt. *null[0]*
2. Der Wert v des Indexausdrucks a wird berechnet.
Falls v negativ ist oder größer gleich der Länge des mit p referenzierten Arrays
ist, erfolgt ein Laufzeitfehler (`IndexOutOfBoundsException`).
Ansonsten wird das an der Position v gespeicherte Element des mit p
referenzierten Arrays geliefert.

Beachte: Falls bei der Auswertung von e keine Exception ausgelöst wird:

- $e[0]$ liefert das erste Element des Arrays und $e[e.length-1]$ das letzte.
- $e[e.length]$ führt zu einer `IndexOutOfBoundsException`.

Arrayzugriff: Beispiel



- $a[3]_{(\sigma,\eta)} = 'P'$, $a[i]_{(\sigma,\eta)} = 'S'$, $a[-8+2*i]_{(\sigma,\eta)} = 'M'$, $a[a.length-1]_{(\sigma,\eta)} = 'S'$,
- $a[b[i-3]]_{(\sigma,\eta)} = 'U'$, $a[b.length]_{(\sigma,\eta)} = 'P'$,
- $a[a.length]$ **und** $a[b[i]-3]$ -> `IndexOutOfBoundsException`.

Arrayerzeugung: Typkorrektheit

ArrayCreation = "new" *Type* *DimExprs* {"**[]**"}

DimExprs = "[" *Expression* "]" {"[" *Expression* "]" }

- Für jede Dimension muss der *Expression*-Ausdruck den Typ `int` (oder einen kleineren Typ) haben.
- *ArrayCreation* hat dann den Typ *Type* `[]...[]` mit so vielen Klammerpaaren, wie Dimensionen angegeben wurden.

Beispiel:

`new char[6]`, `new char[22]` haben den Typ `char[]`.

`new int[3]` hat den Typ `int[]`, `new String[8]` hat den Typ `String[]`.

`new double[4][7]`, `new double[4][]` haben den Typ `double[][]`.

DimExprs

Arrayerzeugung: Auswertung

Sei `new T[d1][d2][d3]` ein Arrayerzeugungs-Ausdruck.

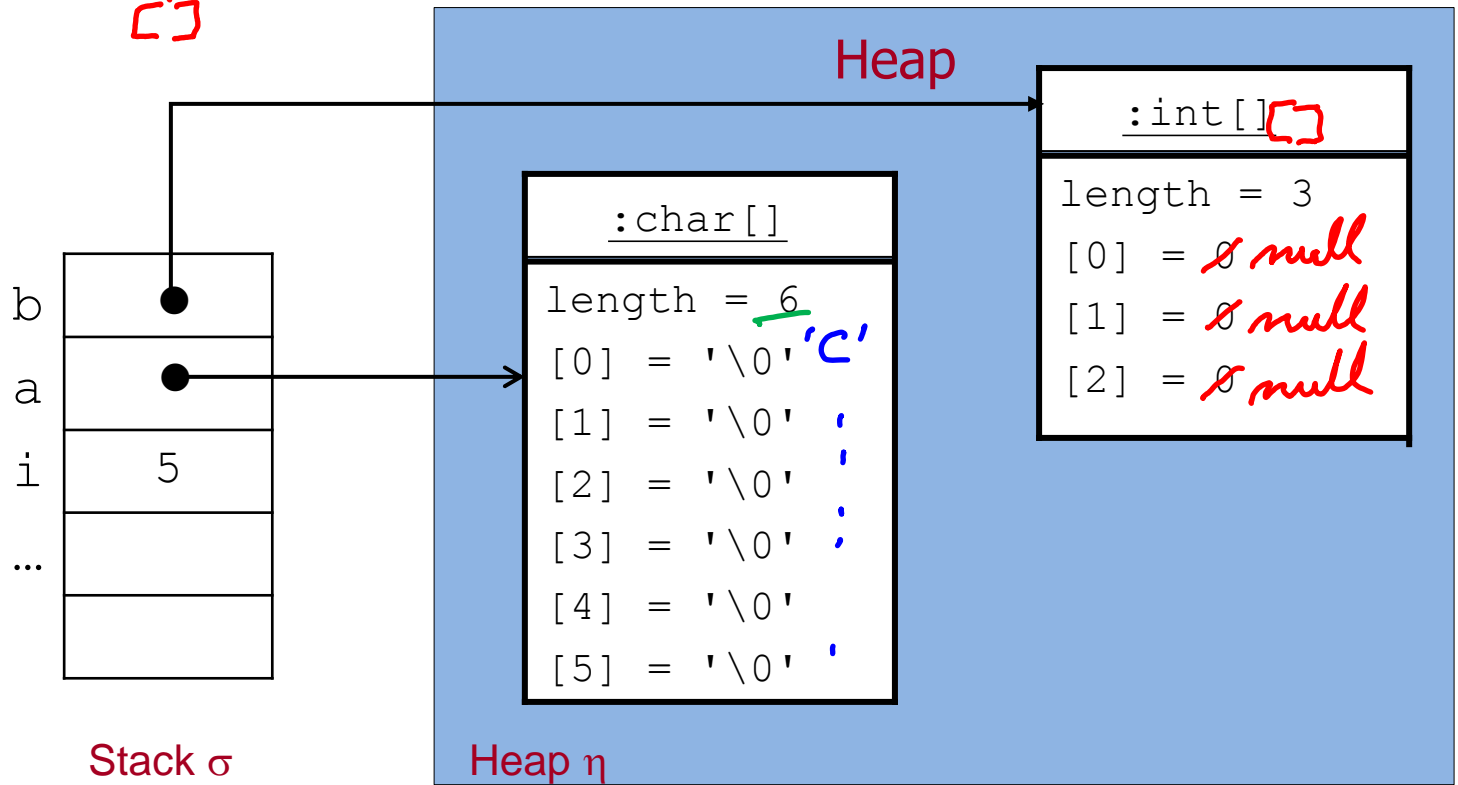
1. Die Werte der Dimensionsausdrücke d_1, \dots, d_n werden ausgehend vom aktuellen Zustand von links nach rechts berechnet. Falls ein Wert negativ ist, erfolgt ein Laufzeitfehler (`NegativeArraySizeException`) und die Auswertung bricht ab.
2. Ein neues Array-Objekt für den Typ `T[][][]` wird erzeugt, dessen Länge der Wert von d_1 ist, und auf den Heap gelegt.
3. Die Komponenten des Array-Objekts werden mit Default-Werten initialisiert. (0 bei `int`, `false` bei `boolean`, `'\0'` bei `char`, `null` bei Klassen- und Arraytypen).
4. Solange n noch nicht erreicht ist, wird dieser Vorgang für die einzelnen Komponenten des gerade angelegten Arrays wiederholt.

Z.B. wird bei $n = 2$ für jede Komponente `a[i]` ($i = 0, \dots, d_1 - 1$) des zuletzt erzeugten Arrays `a` ein Array der Länge d_2 angelegt.

Arrayerzeugung: Beispiel

```
...  
int i = 5;  
char[] a = new char[6];  
int[] b = new int[3];
```

[Handwritten red annotations: a bracket under 'a' and another under 'b' in the code above]



Erweiterungen zur Behandlung von Arrays (Wdh.)

Bisher

Klassendeklarationen

Objekte und Objekthalde (Heap)

Grunddaten- und Klassentypen

Werte

Operationen

Ausdrücke

Typisierung

Auswertung bzgl.

Zustand (Stack + Heap)

Deklarationsanweisung

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

erweitert um

Kapitel 8

Arrayobjekte (kurz: Arrays)

Arraytypen

Referenzen auf Arrayobjekte

==, != für solche Referenzen

Arrayzugriff, Arrayerzeugung

Arrayobjekte auf dem Heap

Arrayinitialisierung

Intialisierung von Arrays

Durch Einzelzuweisungen an die Komponenten:

```
type[] arr = new type[n];  
arr[0] = v0;  
⋮  
arr[n-1] = vn-1;
```

Durch sofortige Initialisierung des gesamten Arrays:

```
type[] arr = {v0, ..., vn-1};
```

Arrayinitialisierung ist eine Deklarationsanweisung. Die Syntax von Deklarationsanweisungen wird dementsprechend erweitert.

Initialisierung von Arrays: Beispiel

- Einzelzuweisungen an die Komponenten:

```
char[] a = new char[6];
```

Deklarationsanweisung

```
a[0] = 'C';  
a[1] = 'A';  
a[2] = 'M';  
a[3] = 'P';  
a[4] = 'U';  
a[5] = 'S';
```

*Variablen
(Array
Access)*

Zuweisungen

- Sofortige Initialisierung des gesamten Arrays:

```
char[] a = { 'C', 'A', 'M', 'P', 'U', 'S' };
```

0 1 2 3 4 5 length = 6

Der Typ von a ist char[], d.h. der Typ eines eindimensionalen Arrays mit Elementen aus char.

Veränderung von Arrays

- Arrayzugriffs-Ausdrücke sind Variablen!
- Infolgedessen kann man ihnen Werte zuweisen und damit den Zustand eines Arrays ändern.
- Die Länge eines Arrays kann nicht verändert werden.

Beispiel:

Man kann beliebige einzelne Buchstaben durch Zuweisungen ändern, z.B. für den Array `a` von oben:

```
a[4] = 'E';
```

```
a[5] = 'R';
```

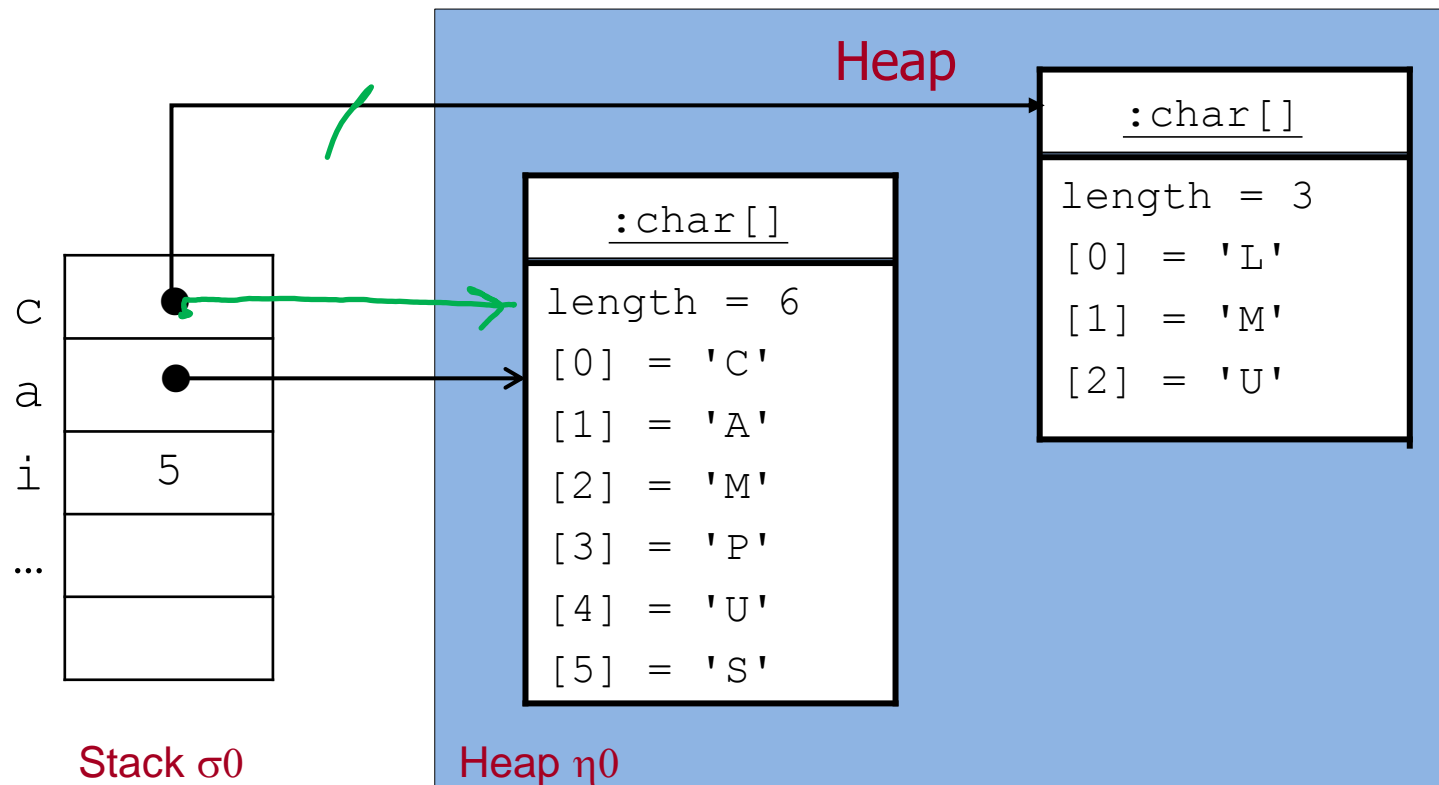
```
for (int i = 0; i < a.length; i++) {  
    System.out.print(a[i]);  
}
```

*Stem ist
 $i \leq a.length - 1$*

druckt dann CAMPER

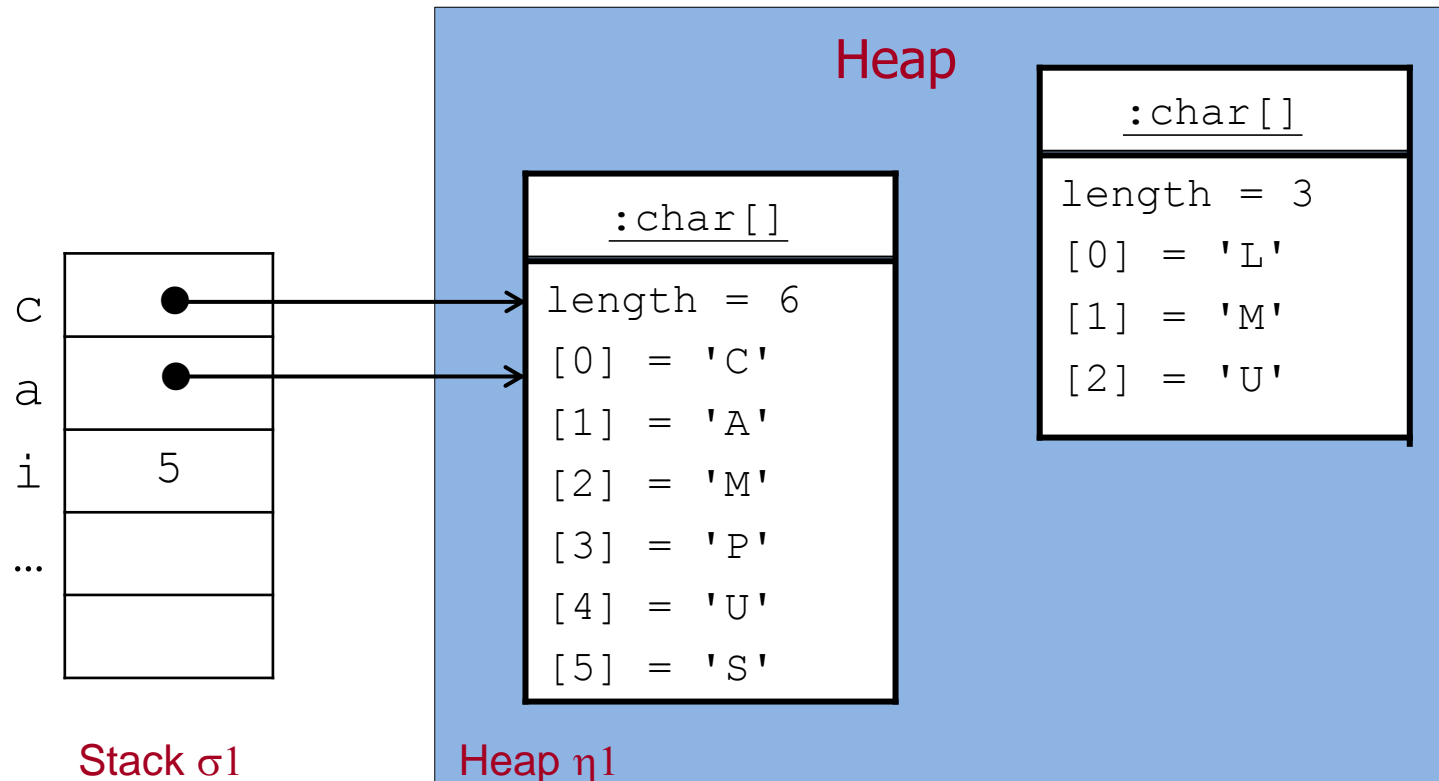
Zuweisungen und Arrays (1)

```
char[] a = {'C', 'A', 'M', 'P', 'U', 'S'};  
char[] c = {'L', 'M', 'U'};
```



Zuweisungen und Arrays (2)

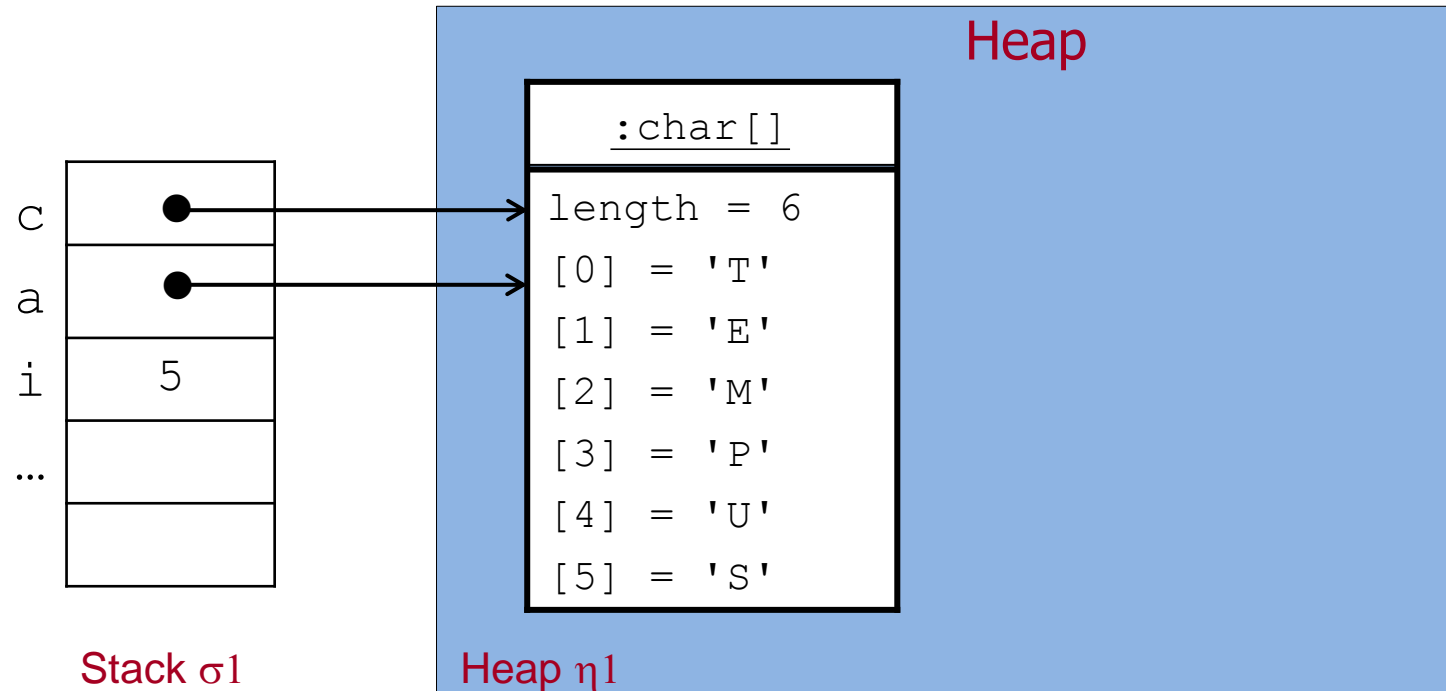
`c = a;` führt zu folgendem Zustand:



Nach Speicherbereinigung (Garbage Collection) wird das nicht mehr zugreifbare Arrayobjekt entfernt.

Zuweisungen und Arrays (3)

`c[0] = 'T'; c[1] = 'E';` führt zu:



```
for (int i = 0; i < a.length; i++) {
    System.out.print(a[i]);
}
```

druckt TEMPUS

Suche nach dem Index eines minimalen Elements eines Arrays

Beispiel:

a:	[3,	-1,	15,	2,	-1]
Index:		0	1	2	3	4	

minIndex i
minIndex

Algorithmus:

- Verwende eine Variable `minIndex` vom Typ `int` für den Index eines minimalen Elements.
- Initialisierung: `minIndex = 0;`
- Durchlaufe das ganze Array von links nach rechts.
Im i -ten Schritt vergleiche das Arrayelement mit Index `minIndex` (d.h. `a[minIndex]`) mit dem Wert des aktuellen Elements (d.h. `a[i]`).
Falls `a[i] < a[minIndex]` setze `minIndex = i`.
- Danach ist der Wert von `minIndex` der Index eines minimalen Elements des Arrays (und `a[minIndex]` ein minimales Element).

Java Implementierung

```
public static void main(String[] args) {  
  
    int[] a = {3, -1, 15, 2, -1};  
  
    int minIndex = 0;  
    for (int i = 1; i < a.length; i++) {  
        // Wir fangen gleich bei i = 1 an, da a[0] < a[0] falsch ist.  
        if (a[i] < a[minIndex]) {  
            minIndex = i;  
        }  
    }  
  
    System.out.println("Index eines minimalen Elements: " + minIndex);  
    System.out.println("Minimales Element: " + a[minIndex]);  
}
```

Statische Methode `findMinIndex`

```
public static int findMinIndex(int[] arr) {  
    int minIndex = 0;  
    for (int i = 1; i < arr.length; i++) {  
        if (arr[i] < arr[minIndex]) {  
            minIndex = i;  
        }  
    }  
    return minIndex;  
}
```

Benutzung:

```
public static void main(String[] args) {  
    int[] a = {3, -1, 15, 2, -1};  
    int min = findMinIndex(a);  
    System.out.println("Index eines minimalen Elements: " + min);  
    System.out.println("Minimales Element: " + a[min]);  
}
```

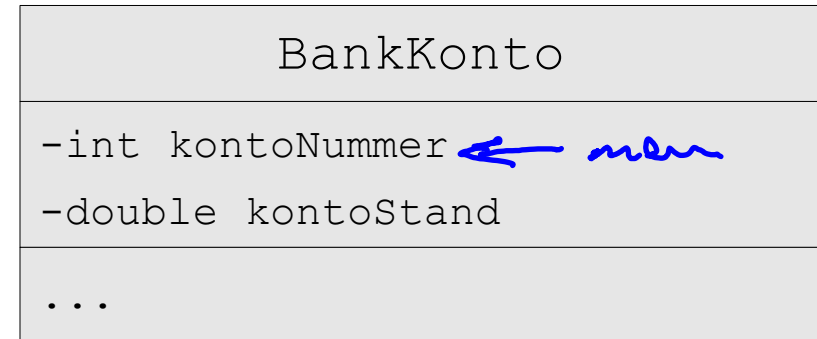
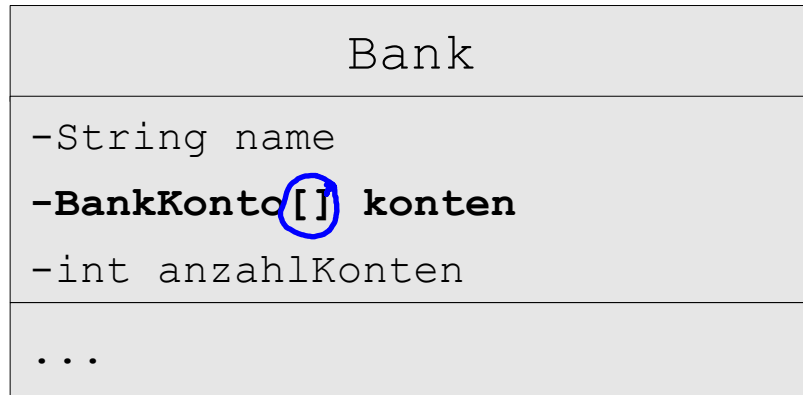
Verdoppeln der Werte eines Arrays

```
public static void doubleValues(int[] a){
    for (int i = 0; i < a.length; i++) {
        a[i] = 2*a[i];
    }
}
```

Benutzung:

```
public static void main(String[] args){
    int[] a = {3, -1, 15, 2, -1};
    doubleValues(a);
    for (int i = 0; i < a.length; i++) {
        System.out.print(a[i]);
    }
}
```

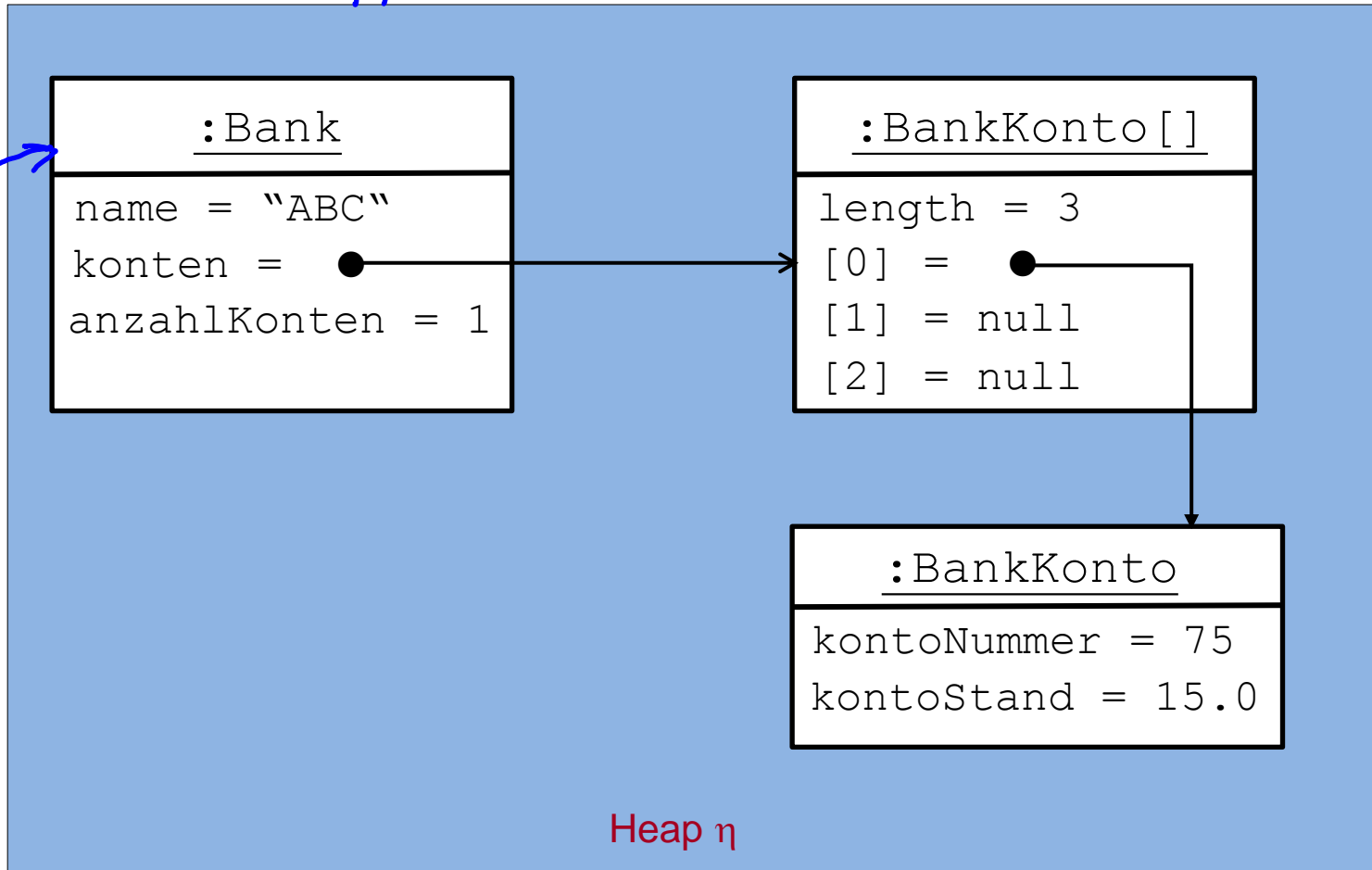
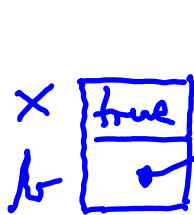
Arrays von Objekten: Bank mit Bankkonten



- Die Konten einer Bank werden in einem Array des Typs `BankKonto[]` gespeichert. Dafür wird das Attribut `konten` verwendet.
- Das Attribut `anzahlKonten` gibt an, wieviele Komponenten des Arrays aktuell mit Referenzen auf Objekte der Klasse `BankKonto` belegt sind.
- Diese Referenzen sind der Reihe nach bis zum Index `anzahlKonten-1` im Array `konten` gespeichert.
- Ein neues Konto muss beim Index `anzahlKonten` eingefügt werden.
- Wir sprechen von einem **partiellen Array** zur Verwaltung der Konten.

Bank b = new Bank("ABC", 3); Heap mit einer Bank

boolean x = b.kontoEröffnen(75, 15.0);



Klasse BankKonto mit Konstruktor und Methoden

BankKonto
<pre>-int kontoNummer ← -double kontoStand</pre>
<pre>+BankKonto(<u>int kontoNummer</u>, double anfangsBetrag) +int getKontoNummer() ← +double getKontoStand() +void einzahlen(double x) +void abheben(double x)</pre>

Klasse BankKonto in Java

```
public class BankKonto {
    private int kontoNummer; ←
    private double kontoStand;
    public BankKonto(int kontoNummer, double anfangsBetrag) {
        this.kontoNummer = kontoNummer;
        this.kontoStand = anfangsBetrag;
    }
    ↑ public int getKontoNummer() {
        return this.kontoNummer;
    }
    public double getKontoStand() {
        return this.kontoStand;
    }
    public void einzahlen(double x) {
        this.kontoStand = this.kontoStand + x;
    }
    public void abheben(double x) {
        this.kontoStand = this.kontoStand - x;
    }
}
```

Klasse Bank mit Konstruktor und Methoden

Bank
-String name
-BankKonto[] konten
-int anzahlKonten
+Bank(String name, int maxAnzahlKonten)
+String getName()
+boolean kontoEroeffnen(int kontoNummer, double anfangsBetrag)
+BankKonto sucheBankkonto(int kontoNummer)
+boolean einzahlen(int kontoNummer, double betrag)
+boolean abheben(int kontoNummer, double betrag)
+double kontoStand(int kontoNummer)
+double gesamtSaldo()

Klasse Bank in Java (1)

```
/**
 * Repräsentation einer Bank mit einem Namen und einer Liste von Konten.
 * @author Annabelle Klarl
 */
public class Bank {
    private String name;
    private BankKonto[] konten;
    private int anzahlKonten;
    /**
     * Konstruktor
     * @param name
     * @param maxAnzahlKonten
     */
    public Bank(String name, int maxAnzahlKonten) {
        this.name = name;
        this.konten = new BankKonto[maxAnzahlKonten];
        this.anzahlKonten = 0;
    }
}
```

Klasse Bank in Java (2)

```
/**
 * Diese Methode liefert den Namen der Bank
 *
 * @return Name der Bank
 */
public String getName() {
    return this.name;
}
```

Klasse Bank in Java (3)

```
/**
 * Diese Methode eröffnet ein Konto mit der gegebenen Kontonummer und dem
 * gegebenen Anfangsbetrag. Dazu wird zunächst ein neues Objekt der Klasse
 * {@link BankKonto} erzeugt, dieses der Bank an der nächsten freien Position
 * hinzugefügt und true zurückgegeben. Ist die Bank schon voll (d.h. wird die
 * Maximalanzahl an Konten für diese Bank überschritten), wird das Konto nicht
 * eröffnet und false zurückgegeben.
 * @param kontoNummer
 * @param anfangsBetrag
 * @return false falls die Maximalanzahl an Konten überschritten würde,
 *         true sonst
 */
public boolean kontoEroeffnen(int kontoNummer, double anfangsBetrag) {
    if (this.anzahlKonten < this.konten.length) [ ]
        this.konten[this.anzahlKonten] =
            new BankKonto(kontoNummer, anfangsBetrag);
        this.anzahlKonten++;
        return true; [ ]
    else return false;
}
```

Klasse Bank in Java (4)

```
/**
 * Diese Methode sucht in der Liste der Konten der Bank das Konto mit der
 * gegebenen Kontonummer. Wird ein Konto gefunden, wird dieses zurückgegeben.
 * Falls kein Konto mit dieser Kontonummer existiert, wird null zurückgegeben.
 *
 * @param kontoNummer
 * @return das Objekt der Klasse {@link BankKonto} mit der gegebenen
 *         Kontonummer; null falls kein Konto mit dieser Kontonummer
 *         existiert.
 */
private BankKonto sucheBankkonto(int kontoNummer) {
    for (int i = 0; i < this.anzahlKonten; i++) {
        BankKonto aktuellesKonto = this.konten[i];
        if (aktuellesKonto.getKontoNummer() == kontoNummer) {
            return aktuellesKonto;
        }
    }
    return null;
}
```

Klasse Bank in Java (5)

```
/**
 * Diese Methode zahlt auf ein Konto mit einer gegebenen Kontonummer einen
 * gegebenen Betrag ein. Falls kein Konto mit dieser Kontonummer existiert,
 * wird false zurückgegeben, sonst true.
 *
 * @param kontoNummer
 * @param betrag
 * @return false falls kein Konto mit dieser Kontonummer existiert,
 *         true sonst
 */
public boolean einzahlen(int kontoNummer, double betrag) {
    BankKonto aktuellesKonto = this.sucheBankkonto(kontoNummer);
    if (aktuellesKonto != null) {
        aktuellesKonto.einzahlen(betrag);
        return true; }
    else return false;
}

//Methode abheben wird analog implementiert
```


Klasse Bank in Java (6)

```
/**
 * Diese Methode gibt den Kontostand des Kontos mit der gegebenen
 * Kontonummer aus. Falls kein Konto mit dieser Kontonummer existiert, wird
 * {@link Integer#MIN_VALUE} zurückgegeben.
 *
 * @param kontoNummer
 * @return der Kontostand des Kontos oder {@link Integer#MIN_VALUE}, falls
 *         kein Konto mit der gegebenen Kontonummer existiert
 */
public double kontoStand(int kontoNummer) {
    BankKonto aktuellesKonto = this.sucheBankkonto(kontoNummer);
    if (aktuellesKonto != null) {
        return aktuellesKonto.getKontoStand();
    }
    else {
        return Integer.MIN_VALUE;
    }
}
```

Klasse Bank in Java (7)

```
/**
 * Diese Methode gibt die Gesamtsumme aller Kontostände zurück.
 *
 * @return der Gesamtsaldo aller Konten dieser Bank
 */
public double gesamtSaldo() {
    double gesamtSaldo = 0.0;
    for (int i = 0; i < this.anzahlKonten; i++) {
        BankKonto aktuellesKonto = this.konten[i];
        gesamtSaldo = gesamtSaldo + aktuellesKonto.getKontoStand();
    }
    return gesamtSaldo;
}

} //Ende Klasse Bank
```

Akkumulator

~~BankKonto aktuellesKonto = this.konten[i];~~

this.konten[i]