LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Informatik:
Programmierung und Softwareentwicklung

Wintersemester 2018/19

Kapitel 9: Arrays

Prof. Dr. David Sabel

Lehr- und Forschungseinheit für Theoretische Informatik

Institut für Informatik, LMU München

WS 2018/19

Stand der Folien: 13. Dezember 2018

Die Inhalte dieser Folien basieren – mit freundlicher Genehmigung – tlw. auf Folien von Prof. Dr. Rolf Hennicker aus dem WS 2017/18 und auf Folien von PD Dr. Ulrich Schöpp aus dem WS 2010/11

Ziele

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

Erweiterung zur Behandlung von Arrays: Überblick

Bisher		**Kapitel 9**
Klassendeklarationen		
Objekte und Heap	erweitert um	Arrayobjekte (kurz: Arrays)
Grunddaten- und Klassentypen	erweitert um	Arraytypen
Werte	erweitert um	Referenzen auf Arrayobjekte
Operationen	erweitert um	==, != für solche Referenzen
Ausdrücke	erweitert um	Arrayzugriff, Arrayerzeugung
Auswertung bezüglich Zustand (Stack + Heap)	erweitert um	Arrayobjekte auf dem Heap
Deklarationsanweisung	erweitert um	Arrayinitialisierung

Einführung

In vielen Anwendungen werden Tupel (Reihungen/Folgen) von Elementen mit einer bestimmten Länge verwendet.

Beispiel: Zahlenfolgen

- `[1.0, 1.0]`
- `[0.2, 1.2, 7.0]`

Beispiel: Zeichenfolgen

- `['L','M','U']`
- `['C','A','M','P','U','S']`

Solche Tupel können durch **Arrays** dargestellt werden.

Einführung (2)

- Ein **Array** ist ein Tupel von Elementen **gleichen Typs**
- Auf die einzelnen Elemente (Komponenten) kann über einen **Index** direkt zugegriffen werden.
- Mathematisch kann ein Array mit n Komponenten eines Typs T als eine Abbildung vom Indexbereich $\{0, \ldots, n-1\}$ in den Wertebereich von T aufgefasst werden.

Beispiel:

```
a:     ['C','A','M','P','U','S']
Index:   0   1   2   3   4   5
```

$$\texttt{a}: \{0, \ldots, 5\} \to \texttt{char}$$

$$\texttt{a[i]} = \begin{cases} \texttt{'C'} & \text{falls } \texttt{i} = 0 \\ \texttt{'A'} & \text{falls } \texttt{i} = 1 \\ \texttt{'M'} & \text{falls } \texttt{i} = 2 \\ \texttt{'P'} & \text{falls } \texttt{i} = 3 \\ \texttt{'U'} & \text{falls } \texttt{i} = 4 \\ \texttt{'S'} & \text{falls } \texttt{i} = 5 \end{cases}$$

Arraytypen

Syntax:

Type = PrimitiveType | ClassType | ArrayType
ArrayType = Type "[]"

Beispiele:

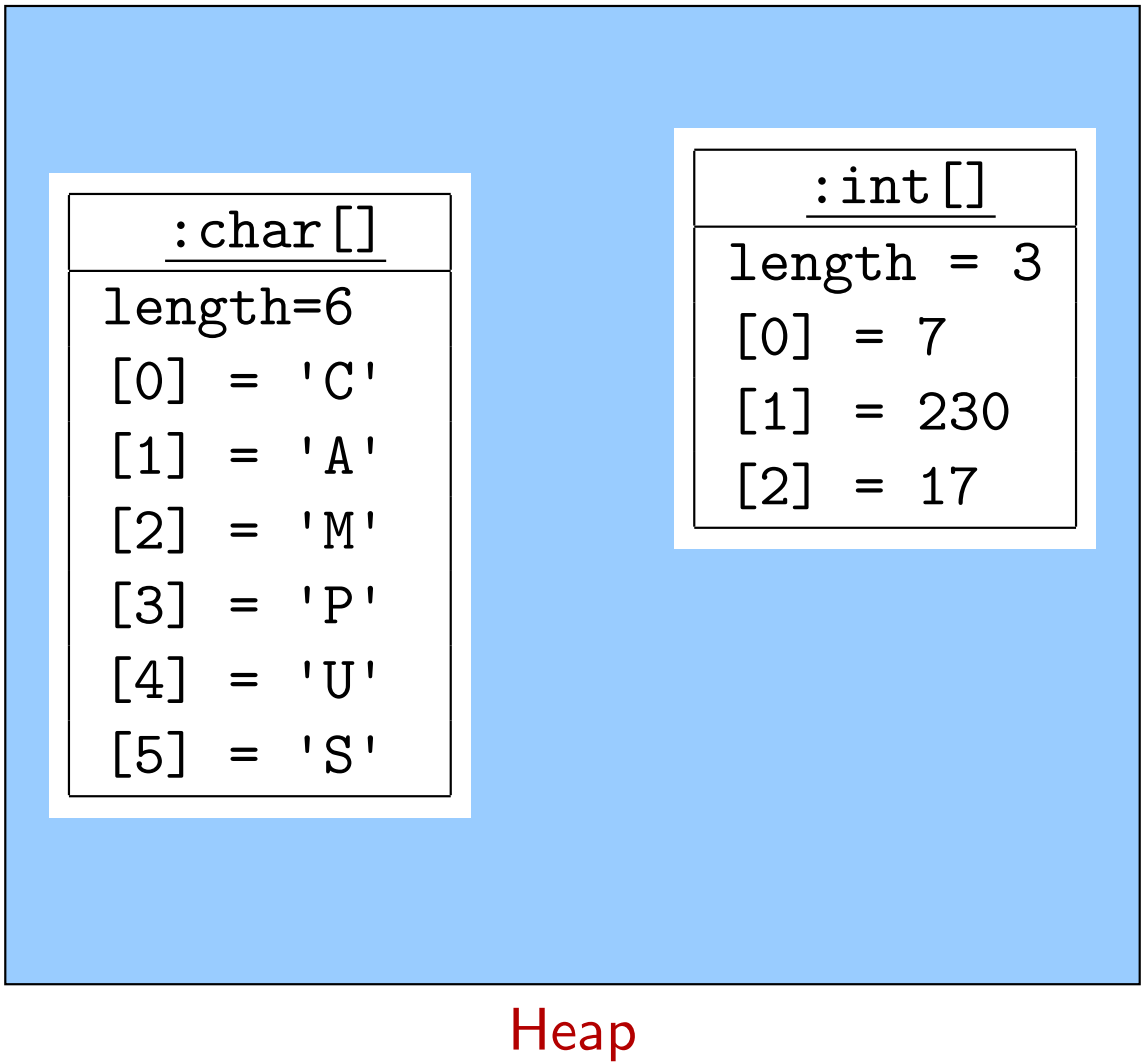
- `int[]`, `boolean[]`, `char[]`, `double[]`, `String[]`, `Point[]`
- `double[][]` 2-dimensionale Arrays mit `double`-Werten
- `Point[][][][]` 4-dimensionale Arrays von Punkten

Arrayobjekte

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`,. . . , `n-1` durchnummeriert sind.

Speicherdarstellung von Arrayobjekten

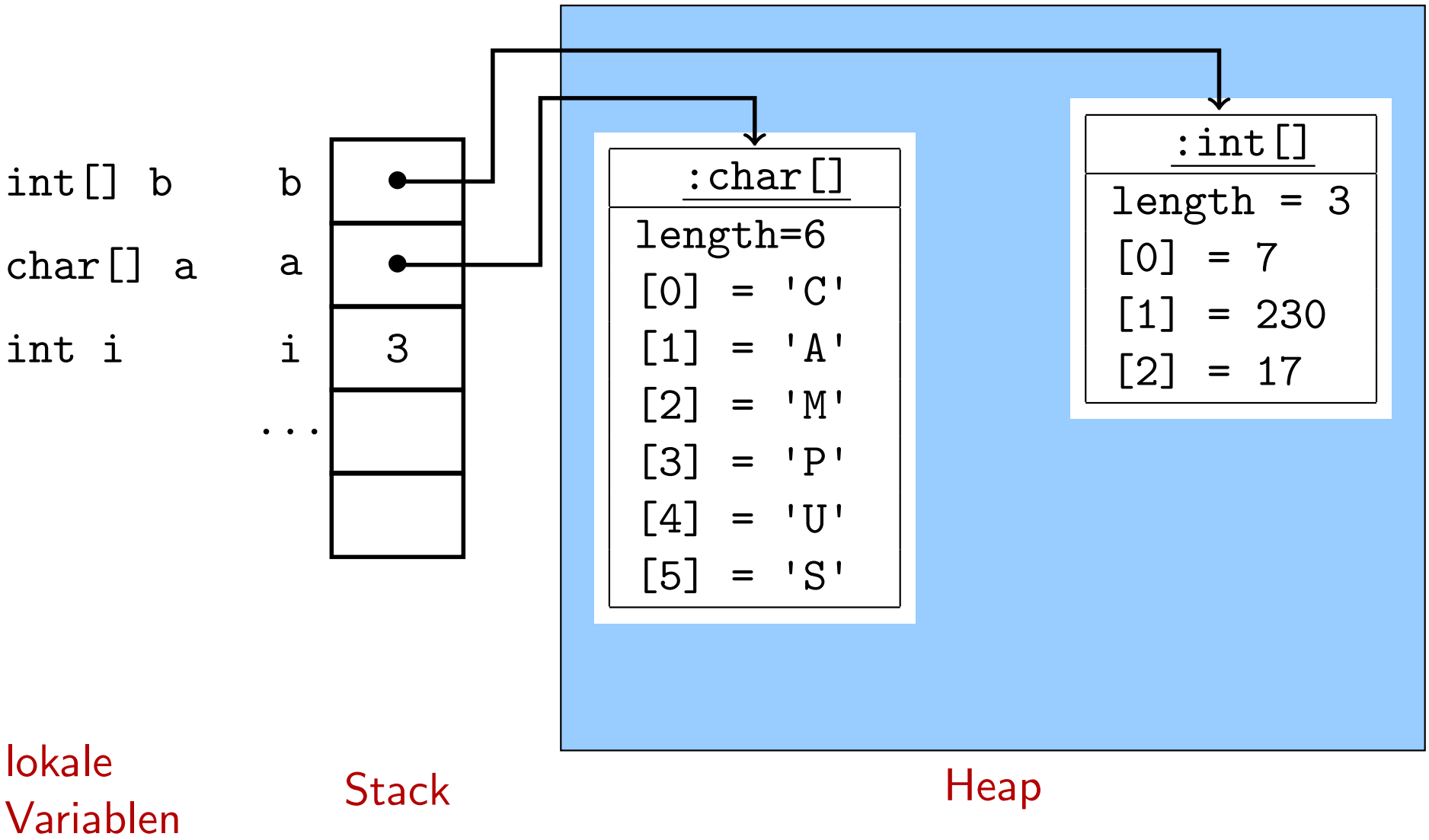


Werte von Arraytypen

- Die **Werte** eines Arraytypen `T[]` sind **Referenzen** auf Arrayobjekte des Typs `T[]` sowie (wie bei Klassentypen) die leere Referenz `null`
- **Lokale Variablen eines Arraytyps** speichern entsprechend Referenzen auf ein Arrayobjekt oder den Wert `null`
- Array-Referenzen können mit den Operatoren == und != auf Gleichheit bzw. Ungleichheit getestet werden

  (nicht empfehlenswert, da die Referenzen und nicht die einzelnen Komponenten der Arrays verglichen werden).

Zustand mit Arrays



Erweiterung zur Behandlung von Arrays: Überblick

Bisher		**Kapitel 9**
Klassendeklarationen		
Objekte und Heap	erweitert um	Arrayobjekte (kurz: Arrays)
Grunddaten- und Klassentypen	erweitert um	Arraytypen
Werte	erweitert um	Referenzen auf Arrayobjekte
Operationen	erweitert um	==, != für solche Referenzen
Ausdrücke	**erweitert um**	**Arrayzugriff**, **Arrayerzeugung**
Auswertung bezüglich Zustand (Stack + Heap)	erweitert um	Arrayobjekte auf dem Heap
Deklarationsanweisung	erweitert um	Arrayinitialisierung

Grammatik für Ausdrücke mit Arrays

```
Expression     = Variable | Value | Expression BinOp Expression
               | UnOp Expression | "(" Expression ")"
               | Expression "?" Expression ":" Expression
               | MethodInvocation | InstanceCreation
Variable       = NamedVariable | FieldAccess | ArrayAccess
NamedVariable  = Identifier
FieldAccess    = Expression "." Expression
ArrayAccess    = Expression "[" Expression "]"
Value          = IntegerValue | FloatingPointValue
               | CharacterValue | BooleanValue | "null"
```

Typisierung von Ausdrücken mit Arrayzugriffen

Ein Ausdruck ist, wie bisher, **typkorrekt**, wenn ihm ein Typ zugeordnet werden kann.

Der Arrayzugriff `"[]"` hat (wie der Attributzugriff `"."`) die höchste Präzendenz 15.

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

Ein Arrayzugriff ist daher von der Form $e\texttt{[}a\texttt{]}$.

- Der Ausdruck e (**Array-Referenzausdruck**) muss einen Arraytyp `T[]` haben und der Ausdruck a (**Indexausdruck**) muss den Typ `int` (oder einen kleineren Typ) haben
- $e\texttt{[}a\texttt{]}$ hat dann den Typ `T` der Arrayelemente.

Beispiele zur Typkorrektheit

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

- `a[3]`, `a[i]`, `a[-8+2*i]`, `a[a.length-1]`, `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`
- `a[4.5]`, `b[true]` `c[c[i][j]]` sind **nicht** typkorrekt

Auswertung von Arrayzugriffen

Die **Auswertung** eines Ausdrucks e erfolgt (weiterhin) unter einem **Zustand** (σ, η), d.h. wir berechnen $e =_{(\sigma,\eta)} \ldots$

Sei e`[`a`]` ein Arrayzugriffs-Ausdruck und 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 sei ref die erhaltene Arrayreferenz.

2. Der Wert v des Indexausdruck a wird berechnet. Falls v negativ oder größer gleich der Länge des mit ref referenzierten Arrays ist, erfolgt ein Laufzeitfehler (`IndexOutOfBoundsException`). Ansonsten wird das an der Postion v gespeicherte Element des mit ref referenzierten Arrays geliefert.

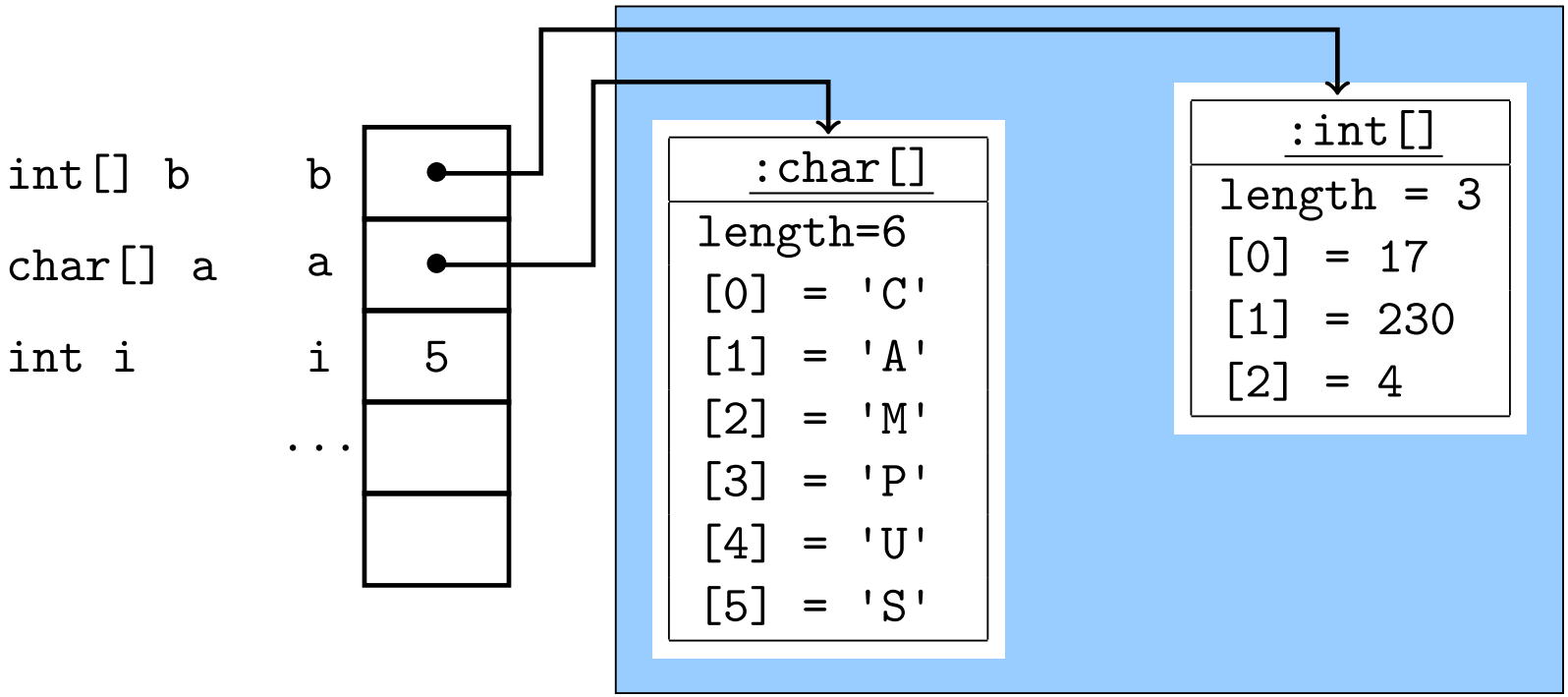
   (Beachte: Positionen werden von 0 beginnend gezählt.)

Arrayzugriffe: Bemerkungen

Falls bei der Auswertung von e keine Exception ausgelöst wird:

- e`[0]` liefert das erste Element des Arrays
- e`[`e`.length-1]` liefert das letzte Element des Arrays
- 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'` und `a[b.length]`$=_{(\sigma,\eta)}$`'P'`.
- `a[a.length]` und `a[b[i]-3]` führen jeweils zu `IndexOutOfBoundsException`

Syntax für Arrayerzeugung

MethodInvocation = Expression "." "(" [ActualParameters] ")"

ActualParameters = Expression { "," Expression }

InstanceCreation = ClassInstanceCreation | ArrayCreation

ClassInstanceCreation = "new" ClassType "(" [ActualParameters] ")"

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

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

Beispiele:

- `new char[6]` erzeugt ein eindimensionales Array von Zeichen mit 6 Plätzen.
- `new double[4][7]` erzeugt ein zweidimensionales Array von double-Werten mit 4 Zeilen und 7 Spalten
- `new int[4][]` erzeugt ein zweidimensionales Array von int-Werten mit 4 Zeilen und noch nicht festgelegter Anzahl an Spalten

Arrayerzeugung: Typkorrektheit

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

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

Typkorrektheit und Typisierung:

Für `newT[`e_1`]...[`e_n`]` $\underbrace{\texttt{[]}\ldots\texttt{[]}}_{m \text{ mal}}$ muss gelten

- Jeder der Ausdrücke e_i hat den Typ `int`.
- Dann hat der gesamte Ausdruck den Typ `T` $\underbrace{\texttt{[]}\ldots\texttt{[]}}_{n+m \text{ mal}}$

Beispiele: Sei `int i` definiert

- `new char[6]`, `new char[i]` haben den Typ `char[]`
- `new int[3]` hat den Typ `int[]`, `new String[i]` den Typ `String[]`
- `new double[4][i]`, `new double[10][]` haben den Typ `double[][]`

Arrayerzeugung: Auswertung

Sei `new T[`d_1`]...[`d_n`][]...[]` ein Arrayerzeugungs-Ausdruck.

1. Die Werte der Dimensionsausdrücke $d_1, \ldots, d_n$ werden ausgehend vom aktuellen Zustand von links nach rechts berechnet. Falls ein Wert negativ ist, erfolgt ein Laufzeitfehler (`NegativeArraySizeException`) und die Auswertung bricht ab.

Arrayerzeugung: Auswertung

Sei `new T[`$\mathtt{d}_1$`]...[`$\mathtt{d}_n$`][]...[]` ein Arrayerzeugungs-Ausdruck.

1. Die Werte der Dimensionsausdrücke $\mathtt{d}_1, \ldots, \mathtt{d}_n$ 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 $\mathtt{d}_1$ ist, und auf den Heap gelegt.

Arrayerzeugung: Auswertung

Sei `new T[`d_1`]...[`d_n`][]...[]` ein Arrayerzeugungs-Ausdruck.

1. Die Werte der Dimensionsausdrücke $d_1, \ldots, d_n$ 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)

Arrayerzeugung: Auswertung

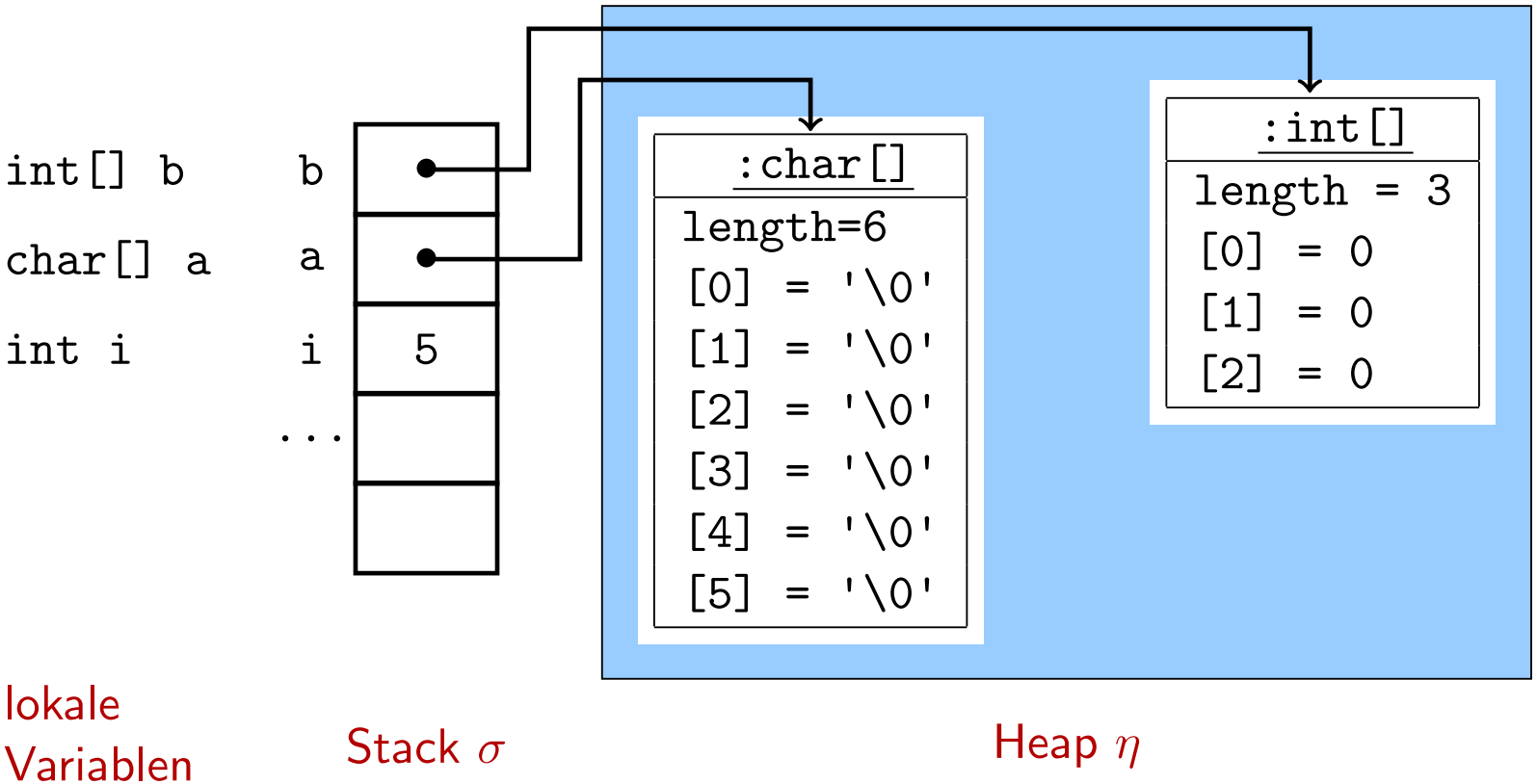
Sei `new T[`d_1`]...[`d_n`][]...[]` ein Arrayerzeugungs-Ausdruck.

1. Die Werte der Dimensionsausdrücke $d_1, \ldots, d_n$ 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,...,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];
```



lokale Variablen

Stack σ

Heap η

Erweiterung zur Behandlung von Arrays

Bisher		**Kapitel 9**
Klassendeklarationen		
Objekte und Heap	erweitert um	Arrayobjekte (kurz: Arrays)
Grunddaten- und Klassentypen	erweitert um	Arraytypen
Werte	erweitert um	Referenzen auf Arrayobjekte
Operationen	erweitert um	==, != für solche Referenzen
Ausdrücke	erweitert um	Arrayzugriff, Arrayerzeugung
Auswertung bezüglich Zustand (Stack + Heap)	erweitert um	Arrayobjekte auf dem Heap
Deklarationsanweisung	**erweitert um**	**Arrayinitialisierung**

Initialisierung von Arrays

Durch Einzelzuweisung an die Komponenten:

```
type[] arr = new type[n];
arr[0] = v_0;
...
arr[n-1]=v_{n-1};
```

Initialisierung von Arrays

Durch Einzelzuweisung an die Komponenten:

```
type[] arr = new type[n];
arr[0] = v_0;
...
arr[n-1]=v_{n-1};
```

Durch sofortige Initialisierung des gesamten Arrays:

```
type[] arr = {v_0,...,v_{n-1}};
```

Initialisierung von Arrays

Durch Einzelzuweisung an die Komponenten:

```
type[] arr = new type[n];
arr[0] = v_0;
...
arr[n-1]=v_{n-1};
```

Durch sofortige Initialisierung des gesamten Arrays:

```
type[] arr = {v_0,...,v_{n-1}};
```

Arrayinitialisierung ist eine Deklarationsanweisung.
Die Syntax von Deklarationsanweisungen muss dementsprechend erweitert werden.

Initialisierung von Arrays: Beispiel

Durch Einzelzuweisung an die Komponenten:

```
char[] a = new char[6];
a[0] = 'C';
a[1] = 'A';
a[2] = 'M';
a[3] = 'P';
a[4] = 'U';
a[5] = 'S';
```

Sofortige Initialisierung des gesamten Arrays:

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

Veränderung von Arrays

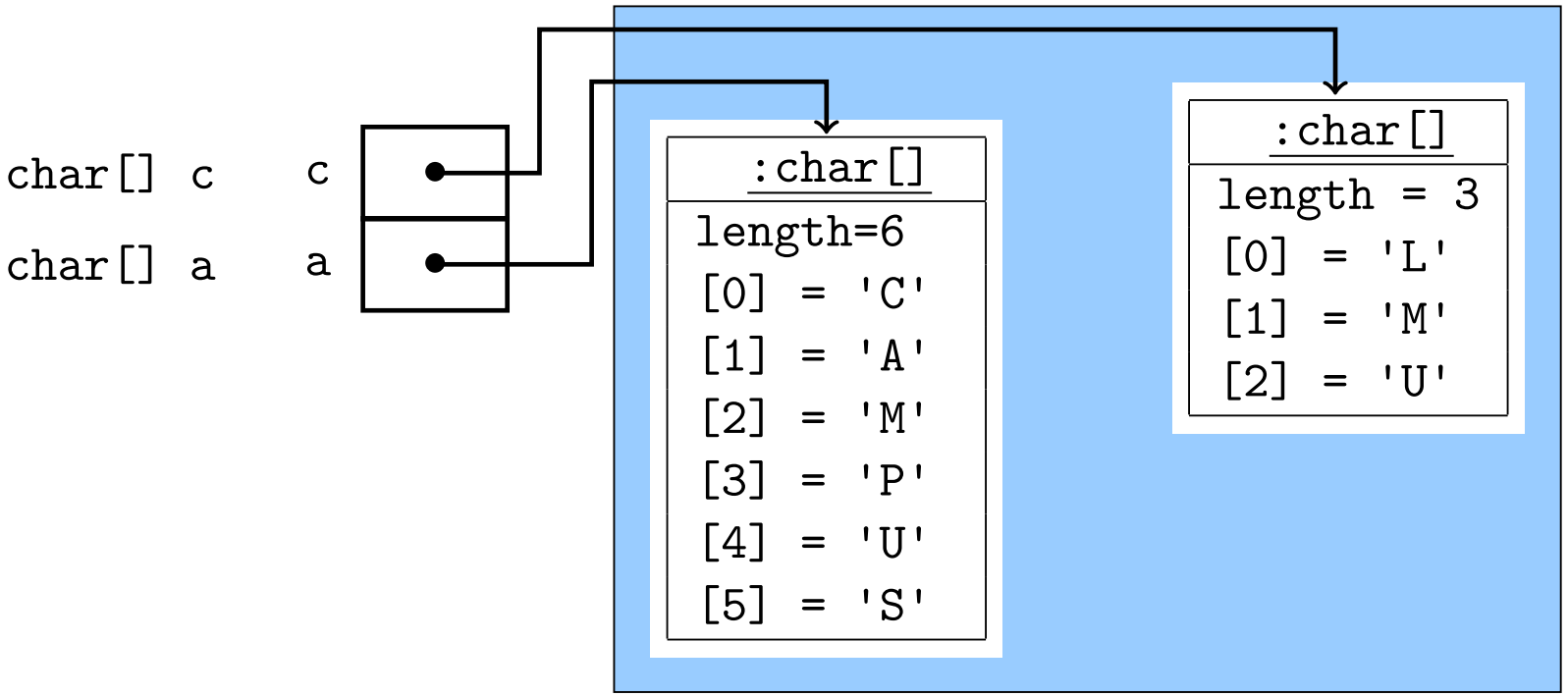
- Arrayzugriffs-Ausdrücke sind Variablen!
- Infolgedessen kann man ihnen Werte zuweisen und damit den Zustand eines Arrays ändern.
- Die Länge eines Arrays kann **nicht** verändert werden.

Beispiel: Man kann beliebige einzelne Buchstaben durch Zuweisungen ändern, z.B. für das Array a von vorher:

```
a[4] = 'E';
a[5] = 'R';
for (int i = 0; i < a.length; i++) {
  System.out.print(a[i]);
}
// druckt CAMPER
```

Zuweisungen und Arrays: Beispiel

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



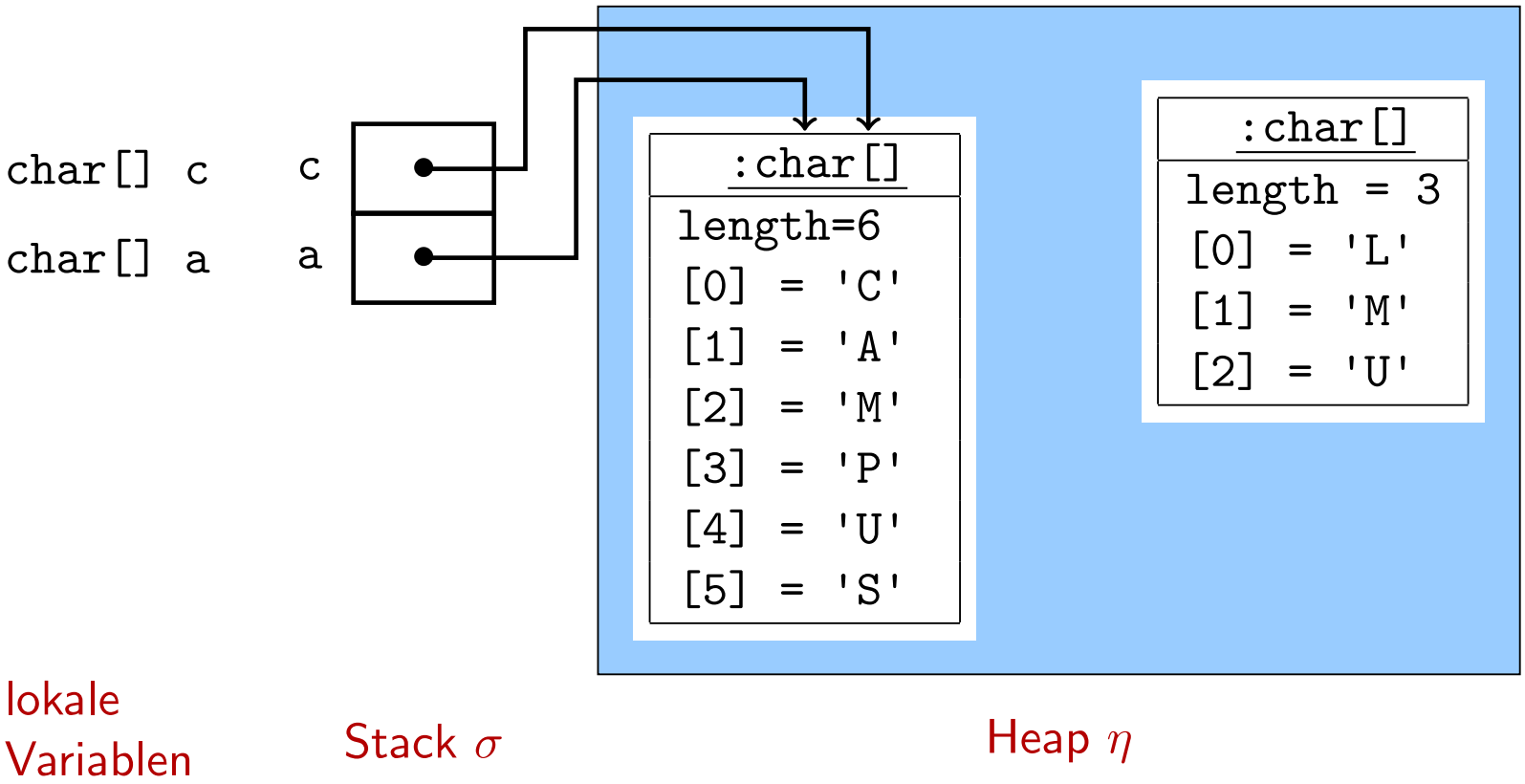
lokale Variablen

Stack σ

Heap η

Zuweisungen und Arrays: Beispiel (2)

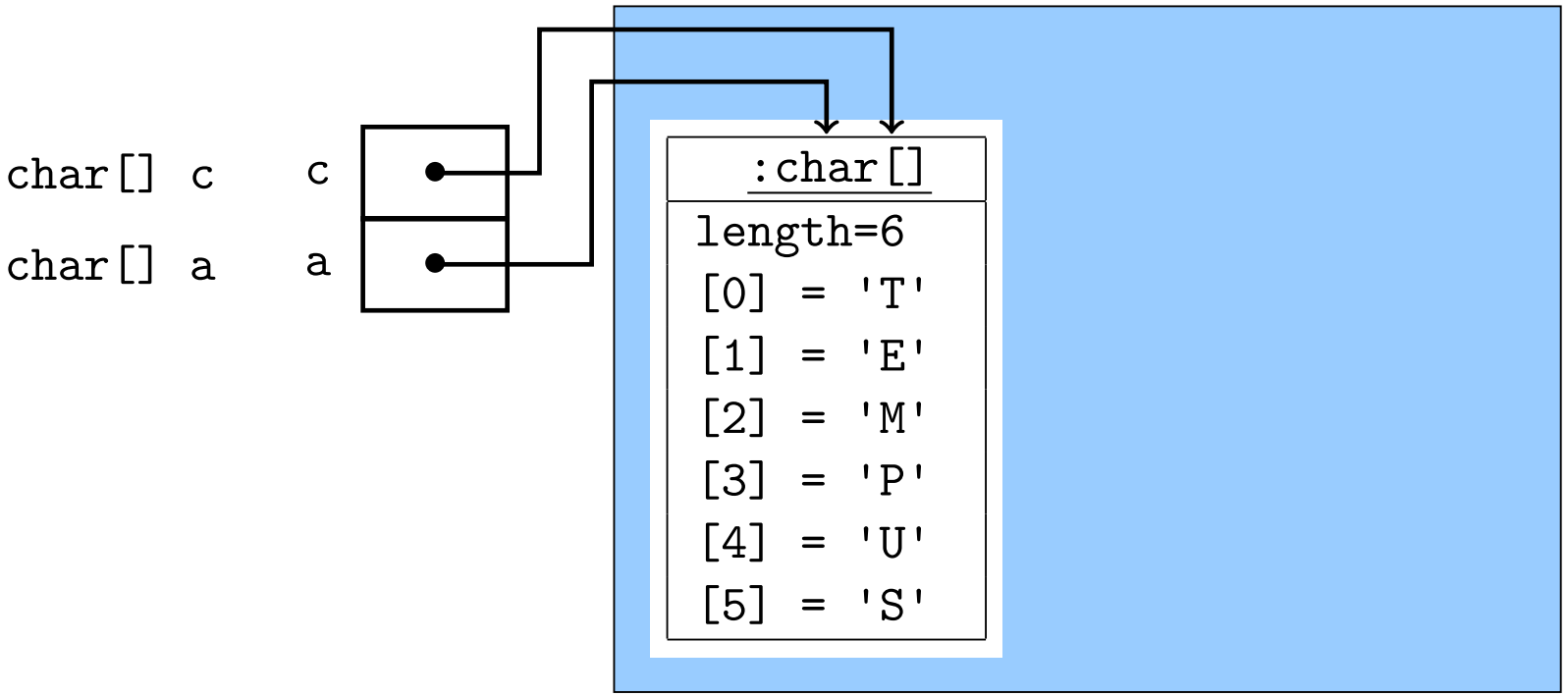
`c = a;` führt zu folgendem Zustand



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

Zuweisungen und Arrays: Beispiel (3)

```
c[0] = 'T';
c[1] = 'E'; führt zu:
```



Mehrdimensionale Arrays: Beispiele

Beispiel: TicTacToe-Spielbrett als (3×3)-Array

```
public class TicTacToe {
  public static final char SPIELER_X = 'X';
  public static final char SPIELER_O = 'O';
  public static final char LEER = ' ';
  private char[][] feld = new char[3][3];
  public TicTacToe() {
    for (int i=0;i<3;i++) {
      for (int j=0;j<3;j++) {
        this.feld[i][j] = LEER;
      }
   }
  }
  public void setze(char spieler,int x,int y){
   if (this.feld[x][y] == LEER) {
      this.feld[x][y] = spieler;
     }
   else {System.out.println("Fehler: Feld belegt");}
  }
```

Mehrdimensionale Arrays: Beispiele (2)

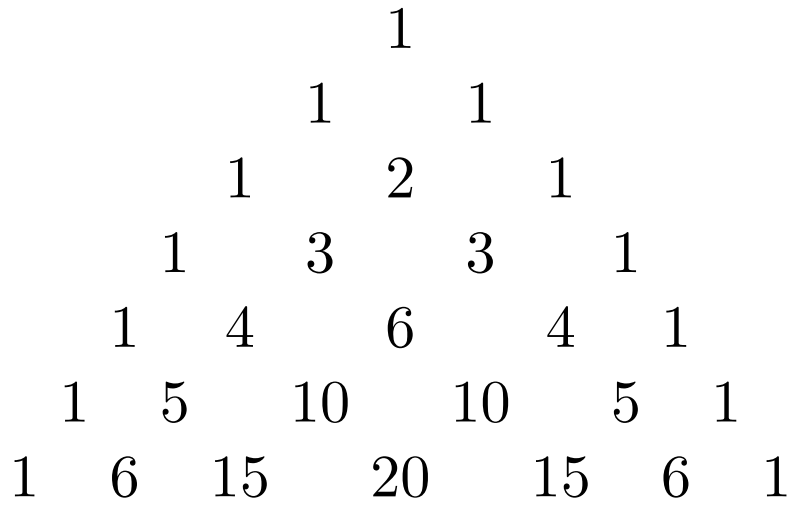
```
public char winner() {
  char winner = ' ';
  for(int i=0;i<3;i++)                                {
   // Pruefe Spalten
   if (   this.feld[i][0] != LEER
       && this.feld[i][0] == this.feld[i][1]
       && this.feld[i][0] == this.feld[i][2])
      {winner = this.feld[i][0];}
   // Pruefe Zeilen
   if (   this.feld[0][i] != LEER
       && this.feld[0][i] == this.feld[1][i]
       && this.feld[0][i] == this.feld[2][i])
      {winner = this.feld[0][i];}                      }
   // Pruefe Diagonalen
  if (this.feld[0][0] != LEER
       && this.feld[0][0] == this.feld[1][1]
       && this.feld[0][0] == this.feld[2][2])
       {winner = this.feld[0][0];}
  if (this.feld[2][0] != LEER
       && this.feld[2][0] == this.feld[1][1]
       && this.feld[2][0] == this.feld[0][2])
       {winner = this.feld[2][0];}
  return winner;
}
```

Mehrdimensionale Arrays: Beispiele (3)

```
  @Override
  public String toString() {
   String returnString = "";
   for(int i=0; i < 3; i++) {
     for (int j = 0; j < 3; j++) {
       returnString = returnString + (this.feld[i][j]);
     }
     returnString = returnString + "\n";
    }
    return returnString;
  }
}
public class TicTacToeMain{
 public static void main(String[] args) {
  TicTacToe ttt = new TicTacToe();
  ttt.setze(TicTacToe.SPIELER_X,1,1);
  ttt.setze(TicTacToe.SPIELER_O,0,1);
  ttt.setze(TicTacToe.SPIELER_X,0,2);
  ttt.setze(TicTacToe.SPIELER_O,1,2);
  ttt.setze(TicTacToe.SPIELER_X,2,0);
  System.out.println(ttt);
  System.out.println("Gewinner:" +  ttt.winner());
 }
}
```

Zweidimensionales Array mit unterschiedlicher Zeilenlänge

Beispiel: Pascalsches Dreieck




```
public class PascalschesDreieck {
 public static void main(String args[]) {
  final int ZEILEN = 20;
  int[][] dreieck = new int[ZEILEN][];      // ZEILEN viele Zeilen
  for (int i=0; i < dreieck.length; i++) { // durchlaufe die Zeilen
   dreieck[i] = new int[i+1];               // i+1 Eintr"age in Zeile
   for (int j =0; j <= i; j++) {            // durchlaufe die Eintr"age
    if (i == 0 || j == 0 || j == i) {dreieck[i][j] = 1;} // R"ander
    else {dreieck[i][j] = dreieck[i-1][j-1] + dreieck[i-1][j];}
   }
  };
 }
```

Suche in einem Array

Suche nach dem Index eines Minimums der Elemente eines Arrays

Beispiel:

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

Suche in einem Array

Suche nach dem Index eines Minimums der Elemente eines Arrays
Beispiel:

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

Algorithmus:

- Verwende eine Variable `minIndex` vom Typ `int` für das Ergebnis, d.h. den Index eines Minimums
- Initialisierung: `minIndex = 0;`
- Durchlauf das ganze Array von links nach rechts ab Index 1.
  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 Minimums der Elemente des Arrays.

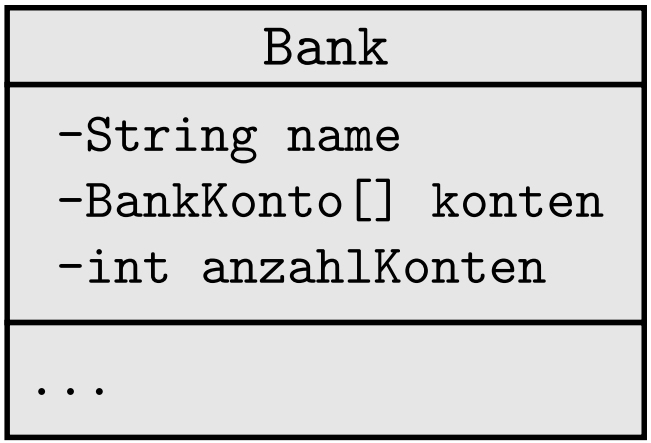
Java-Implementierung

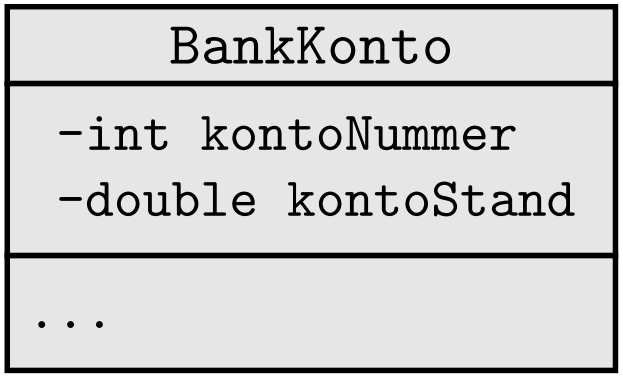
```
public class FindMinIndex {
 public static int findMinIndex(int[] array) {
  int minIndex = 0;
  for (int i = 1; i < array.length; i++) {
    if (array[i] < array[minIndex]) {
       minIndex = i;
    }
  }
  return minIndex;
 }
}

//Benutzung
public class MinIndexTest{
 public static void main(String[] args) {
   int[] array = {3, -1, 15, 2, -1};
   int min = FindMinIndex.findMinIndex(array);
   System.out.println("Index eines Minimums: " + min);
   System.out.println("Minimum: " + array[min]);
 }
}
```

Verdoppeln der Werte eines Arrays

```
class DoubleValues {
 public static void doubleValues(int[] array) {
  for (int i=0; i < array.length; i++) {
    array[i] = 2*array[i];
  }
 }
}
//Benutzung
public class DoubleValuesTest{
 public static void main(String[] args) {
   int[] array = {3, -1, 15, 2, -1};
   DoubleValues.doubleValues(array);
   for (int i=0; i < array.length; i++) {
    System.out.print(array[i]  + " ");
   }
   System.out.println();
 }
}
```

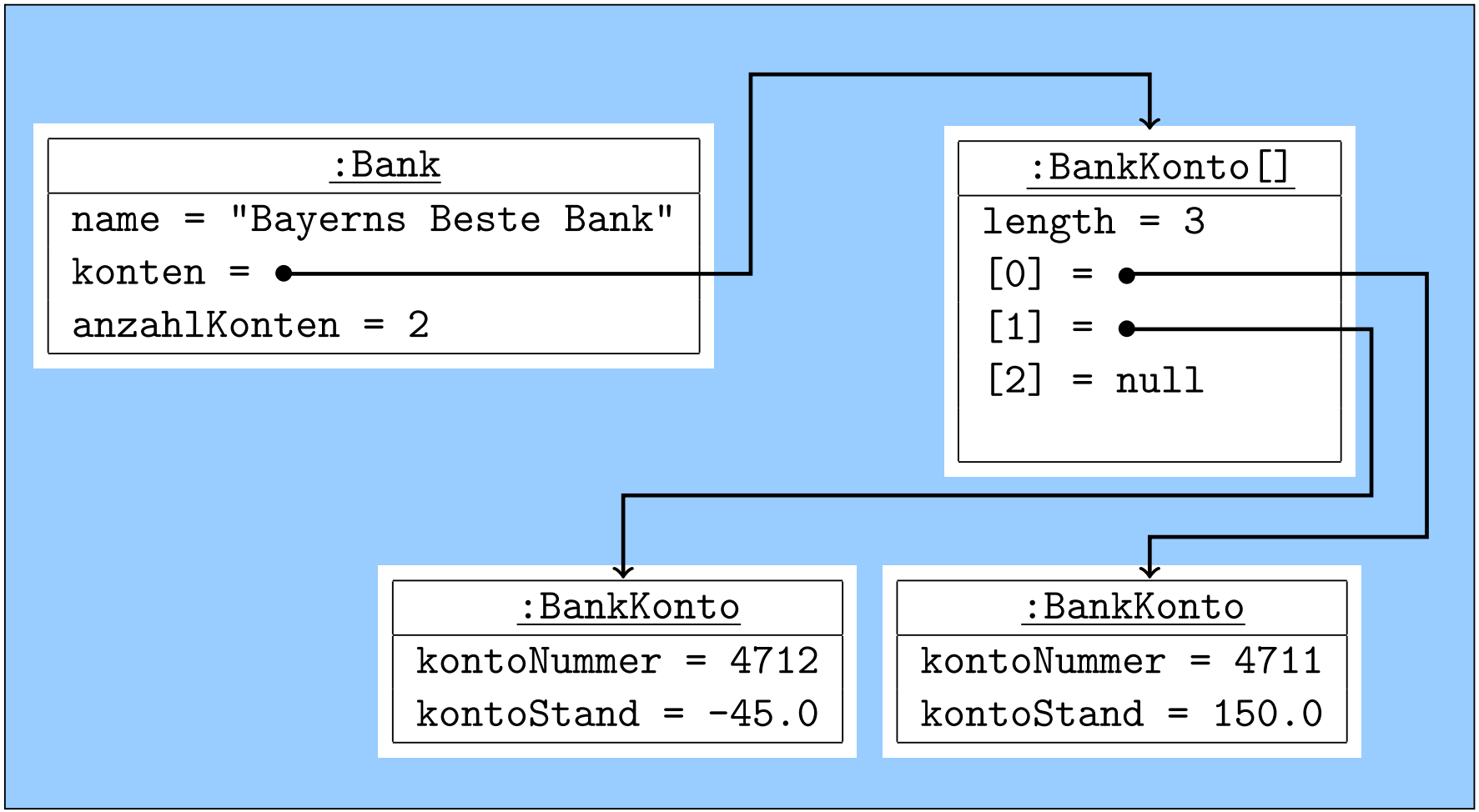
Arrays von Objekten, Beispiel: Bankkonten



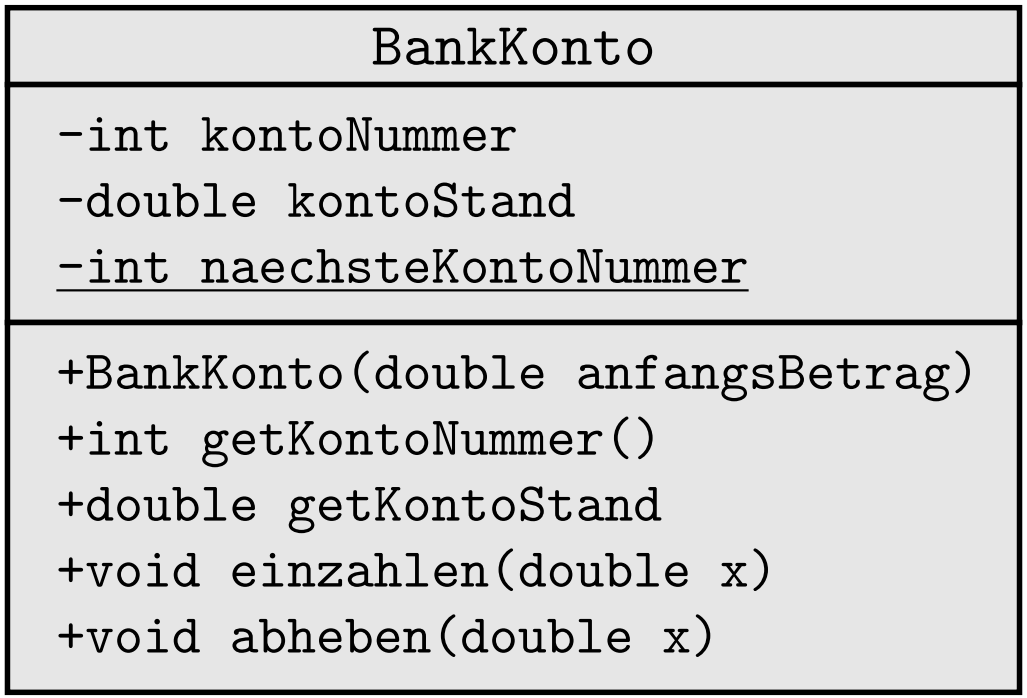


- Die Konten einer Bank werden in einem Array des Typs `BankKonto[]` gespeichert. Dafür wird das Attribut `konten` verwendet.
- Das Attribut `anzahlKonten` gibt an, wieviele Komponenten des Arrays aktuell mit Referenzen auf Objekte der Klasse `BankKonto` belegt sind.
- DIe Referenzen sin der Reihe nach 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.

Heap mit Bankkonten



Klasse BankKonto mit Konstruktor und Methoden



Bemerkung: Klassenattribute (`naechsteKontoNummer`) werden in UML-Diagrammen unterstrichen.

BankKonto in Java

```
public class BankKonto {
  private int kontoNummer;
  private double kontoStand;
  private static int naechsteKontoNummer=0;
  public BankKonto(double anfangsBetrag) {
     this.kontoNummer = BankKonto.naechsteKontoNummer;
     BankKonto.naechsteKontoNummer++;
     this.kontoStand = anfangsBetrag;
  }
  public double getKontoStand() {
     return this.kontoStand;
  }
  public int getKontoNummer() {
     return this.kontoNummer;
  }
  public void einzahlen (double x) {
     this.kontoStand += x;
  }
  public void abheben(double x) {
     this.kontoStand -= x;
  }
}
```

Klasse Bank mit Konstruktor und Methoden

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

Bemerkungen:

- kontoEroeffnen liefert die Kontonummer zurück
- einzahlen / abheben liefern true/false jenachdem, ob erfolgreich

Bank in Java

```
public class Bank {
 // Attribute
 String name;
 BankKonto[] konten;
 int anzahlKonten;

 // Konstruktor
 public Bank(String name, int maxAnzahlKonten) {
  this.name = name;
  this.konten = new BankKonto[maxAnzahlKonten];
  this.anzahlKonten = 0;
 }

 // Methoden
 public String getName() {
  return this.name;
 }
 ...
```

Bank in Java (2)

```
...
// Konto eroeffnen
public int kontoEroeffnen(double anfangsBetrag) {
 // Ist noch Platz?
 if (anzahlKonten >= konten.length) {
  // schaffe Platz, indem ein doppelt so
  // grosses Array angelegt wird und
  // die Daten kopiert werden.
  int maxAnzahlKonten = konten.length;
  BankKonto[] kontenNeu = new BankKonto[maxAnzahlKonten*2];
  for(int i=0;i < anzahlKonten;i++) {
     kontenNeu[i] = konten[i];
  }
  this.konten = kontenNeu;
 }
 // jetzt ist sicher genuegend Platz
 BankKonto neuesKonto = new BankKonto(anfangsBetrag);
 this.konten[this.anzahlKonten] = neuesKonto;
 this.anzahlKonten++;
 return neuesKonto.getKontoNummer();
}
...
```

Bank in Java (3)

```
...
// BankKonto suchen
public BankKonto sucheBankKonto(int kontoNummer) {
  for (int i = 0; i < anzahlKonten; i++) {
    BankKonto ergebnis = this.konten[i];     // lokale Variable
    if (ergebnis.getKontoNummer() == kontoNummer)
      { return ergebnis; }                   // gefunden
   }
  return null;                               // erfolglos
}
// einzahlen
public boolean einzahlen(int kontoNummer,double betrag) {
  BankKonto konto = this.sucheBankKonto(kontoNummer);
  if (konto == null) { return false; } // Konto existiert nicht
  else { konto.einzahlen(betrag); return true; }
}
// abheben
public boolean abheben(int kontoNummer,double betrag) {
  BankKonto konto = this.sucheBankKonto(kontoNummer);
  if (konto == null) { return false; } // Konto existiert nicht
  else { konto.abheben(betrag); return true; }
}
...
```

Bank in Java (4)

```
...
// Kontostand abfragen
public double kontoStand(int kontoNummer) {
  BankKonto konto = this.sucheBankKonto(kontoNummer);
  if (konto == null) {
     // kleinste Zahl zur"uck geben
     // besser w"are ein Fehler!
     return Integer.MIN_VALUE;
  }
  else {return konto.getKontoStand();}
}
// Gesamtsaldo berechnen
public  double gesamtSaldo() {
 int saldo = 0;                                  // lokale Variable
 for (int i = 0; i < anzahlKonten; i++) {
  // Kontostand von Konto zum saldo dazu addieren
  saldo += konten[i].getKontoStand();
  }
 return saldo;
}
}
```

Bank testen

```
public class BankTest {
 public static void main(String[] args) {
 Bank b = new Bank("Bayerns Beste Bank",2);
 int kontoNr1 = b.kontoEroeffnen(10);
 int kontoNr2 = b.kontoEroeffnen(20);
 int kontoNr3 = b.kontoEroeffnen(30);
 int kontoNr4 = b.kontoEroeffnen(40);
 int kontoNr5 = b.kontoEroeffnen(50);
 System.out.println(b.sucheBankKonto(kontoNr2));
 System.out.println(b.einzahlen(kontoNr2,150));
 System.out.println(b.einzahlen(kontoNr3,150));
 System.out.println(b.einzahlen(kontoNr4,150));
 System.out.println(b.einzahlen(7,150));
 System.out.println(b.sucheBankKonto(7));
 System.out.println(b.kontoStand(kontoNr1));
 System.out.println(b.kontoStand(kontoNr2));
 System.out.println(b.kontoStand(kontoNr3));
 System.out.println(b.kontoStand(kontoNr4));
 System.out.println(b.kontoStand(kontoNr5));
 System.out.println(b.kontoStand(7));
 System.out.println(b.gesamtSaldo());
 }
}
```

Beispiel: Stack (Stapel) mit Arrays

Wiederholung: ein Stack ist ein LIFO-Speicher (Last-in-first-out)

Operationen (für einen Stack von int-Werten):

- `void push(int i)` legt Zahl `i` oben auf den Stack
- `int pop()` entfernt die oberste Zahl vom Stack und liefert diese als Ergebnis.
- `int top()` liefert die oberste Zahl vom Stack als Ergebnis, ohne sie zu entfernen
- `boolean isEmpty()` prüft, ob der Stack leer ist

Beispiel: Stack (Stapel) mit Arrays (2)

Stack als Klasse in Java, mit Verwendung
eines Arrays zum Speichern der Elemente:

```
public class IntStack {
  private int[] arr; // Array f"uer die Elemente
  private int top;   // Index des obersten Elements

  public IntStack(int size) {
    this.arr = new int[size];
    this.top = -1;  //da keine Elemente
  }

  public int top() {
    // Achtung: Exception bei leerem Stack
    return this.arr[this.top];
  }

  ...
```

Beispiel: Stack (Stapel) mit Arrays (3)

```
  ...

  public int pop () {
    // Achtung: Exception bei leerem Stack
    int result = this.arr[this.top];
    this.top--;
    return result;
  }

  public void push(int i) {
    // Achtung: Exception, wenn Stack zu klein
    this.top++;
    this.arr[top]=i;
  }

  public boolean isEmpty() {
    return this.top == -1;
  }
}
```

Warteschlangen: FIFO-Queue

Eine Warteschlange dient zum Speichern von Elementen (hier `int`-Werte) und unterstützt die Operationen:

- `boolean enqueue(int i)` fügt Element `i` der Warteschlange hinzu (falls noch Platz ist) und liefert `true` bei Erfolg, und `false` sonst.
- `int dequeue()` entfernt und liefert das nächste Element der Warteschlange. Die Methode ist undefinert bei leerer Queue.

Dabei werden die Elemente in FIFO (first-in-first-out)-Reihenfolge abgespeichert.

Beispiel: FIFO-Queue mit Arrays (1)

```
public class FIFOQueue {
  private int[] arr;        // Array f"uer die Elemente
  private int maxSize;      // maximale Gr"oesse
  private int currentSize; // aktuelle Gr"oesse
  private int first;        // Index des ersten Elements
  private int last;         // Index des letzten Elements

  public FIFOQueue(int maxSize) {
    this.arr = new int[maxSize];
    this.maxSize = maxSize;
    this.first = -1;
    this.last  = -1;
  }

  public boolean isEmpty() {
    return (this.currentSize == 0);
  }
  ...
```

Beispiel: FIFO-Queue mit Arrays (2)

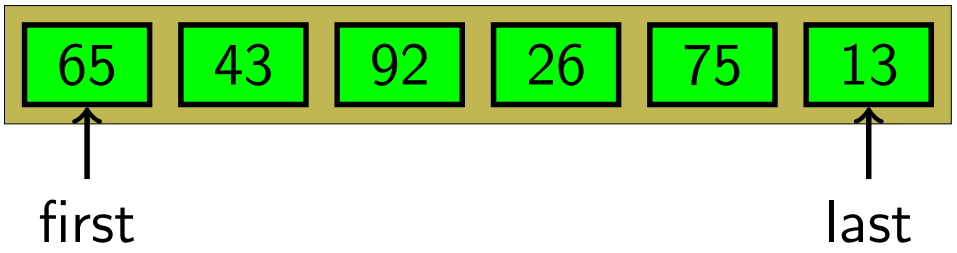
```
...
public int dequeue () {
  int returnValue = arr[this.first];
  if (currentSize <= 1) { // letztes Element wird entfernt
    this.last = -1 ;
    this.first = -1;
  }
  else { // first-Zeiger um 1 verschieben
    this.first = (this.first + 1) % maxSize;
  }
  currentSize = (currentSize > 0)?(currentSize - 1):0;
  return returnValue;
}
public boolean enqueue (int i) {
  if (currentSize == maxSize) {return false;}
  else {if (first == -1) {first++;}
        last = (last + 1) % maxSize;
        arr[last] = i;
        currentSize++;
        return true;}
}
```

Beispiel: FIFO-Queue mit Arrays (3)

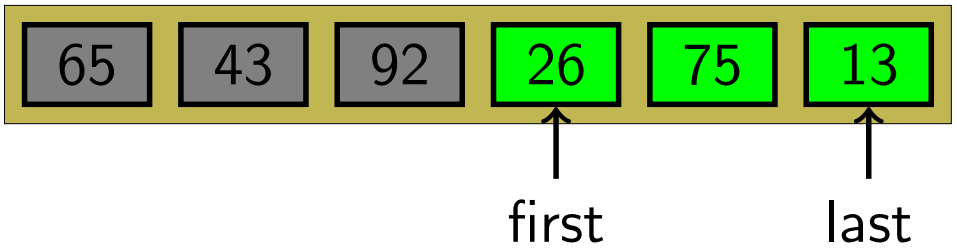
Beachte, dass die Warteschlange im Feld „wandert“:

Beispiel: Queue der Größe 6

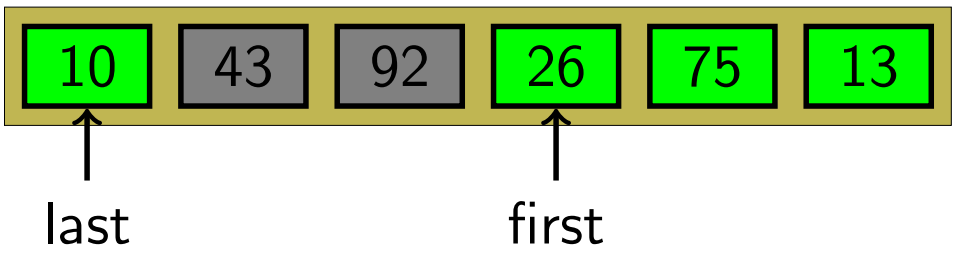
- `enqueue(65);enqueue(43);enqueue(92);enqueue(26);enqueue(75);enqueue(13);`



- `dequeue();dequeue();dequeue();`



- `enqueue(10);`



Bemerkung zu main

Wir kennen alle Konstrukte, um die `main`-Methode in Java zu verstehen

```
public class Classname {
  public static void main(String[] args) {
    ...
 }
}
```

Die Main-Methode ist eine Klassenmethode (`static`) ohne Ergebnis (`void`), die als Eingabeparameter ein Array von `Strings` erhält.

Bemerkung zu main

Wir kennen alle Konstrukte, um die `main`-Methode in Java zu verstehen

```
public class Classname {
  public static void main(String[] args) {
    ...
 }
}
```

Die Main-Methode ist eine Klassenmethode (`static`) ohne Ergebnis (`void`), die als Eingabeparameter ein Array von `Strings` erhält.

Die Strings kann man beim Programmaufruf übergeben:

```
public class TestMain {
  public static void main(String[] args) {
    for (int i=0; i < args.length; i++) {
     System.out.println("Argument " + i + ": " +  args[i]);
}}}
```

```
> javac TestMain.java
> java  TestMain hier kann man "mehrere Parameter uebergeben"
Argument 0: hier
Argument 1: kann
Argument 2: man
Argument 3: mehrere Parameter uebergeben
```

Zusammenfassung

- Datenstruktur Array zur Speicherung einer Reihe von Objekten gleichen Typs
- Beliebige aber feste Länge
- Mehrdimensionale Arrays
- Arrayinitialisierung, Arrayerzeugung und Arrayzugriff
- Algorithmen auf Arrays