Array-Formalia und Reste

- Deklaration und Zugriff
- Initialisierung mit Werten
- Formalia
- Kopieren von Arrays
- Array als Rückgabewert einer Funktion

Vereinbarung von Array-Variablen

Array-Variable werden vereinbart, indem nach dem Elementtyp *eckige Klammern [ ]* notiert werden

Beispiele

- Konkret:

```
int[ ]      werte;      // array of int
boolean[ ]  b1, b2, b3; // b1, b2 und b3 sind array of boolean
String[ ]   arrayOfString;
```

- Formale Schreibweise:

Typ[ ] v*ariable1*, v*ariable2*;

- Wichtig:
  Das ist nur die Vereinbarung einer Variablen,
  bisher gibt es das Array selbst noch nicht!

Array-Werte

- Arrays werden mittels **Konstruktor-Aufruf** in einer bestimmten Länge erzeugt; Beispiele:

```
new boolean[15]
new int[5]
new String[10]
new int[0]
```

- Wie bei anderen Typen auch können Array-Variablen bei der *Initialisierung* oder in einer *Zuweisung* mit einem Wert versehen werden; Beispiele:

```
boolean [ ] bc = new boolean[15];

int [ ] ia;
ia = new int[5];
```

- Formal
  ArrayVariable = **new** *Element-Typ*[*Länge*];

Initialisierung bei der Vereinbarung

- int [ ] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };
  - Die Variable factorial ist ein Integer-Array mit 8 Werten
  - Die Indizes laufen von 0 bis 7
- boolean [ ] bc = { true, false, false, true, true, true };
  - Die Variable bc ist ein Boolean-Array mit 6 Werten
  - Die Indizes laufen von 0 bis 5
- String [ ] as = { "array", "of", "String" };
  - Die Variable as ist ein String-Array mit 3 Werten
  - Die Indizes laufen von 0 bis 2
- int [ ] empty = { };
  - Die Variable empty ist ein leeres int-Array
  - Das ist ein gültiger Wert für ein Array (wie z.B. ein leerer String)
  - Es gibt keinen gültigen Index-Wert

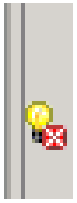
Undefinierte Array-Variable

- Arrays verhalten sich wie Variablen von anderen Typen: Solange einer Array-Variablen nichts zugewiesen wurde, ist ihr Wert undefiniert
- Es kann also weder auf ihre Komponenten zugegriffen werden, noch kann ihre Länge abgefragt werden
- Das bedeutet, der *Compiler* meldet (wie bei anderen Variablen) jeden Zugriffsversuch als Fehler; Beispiele:

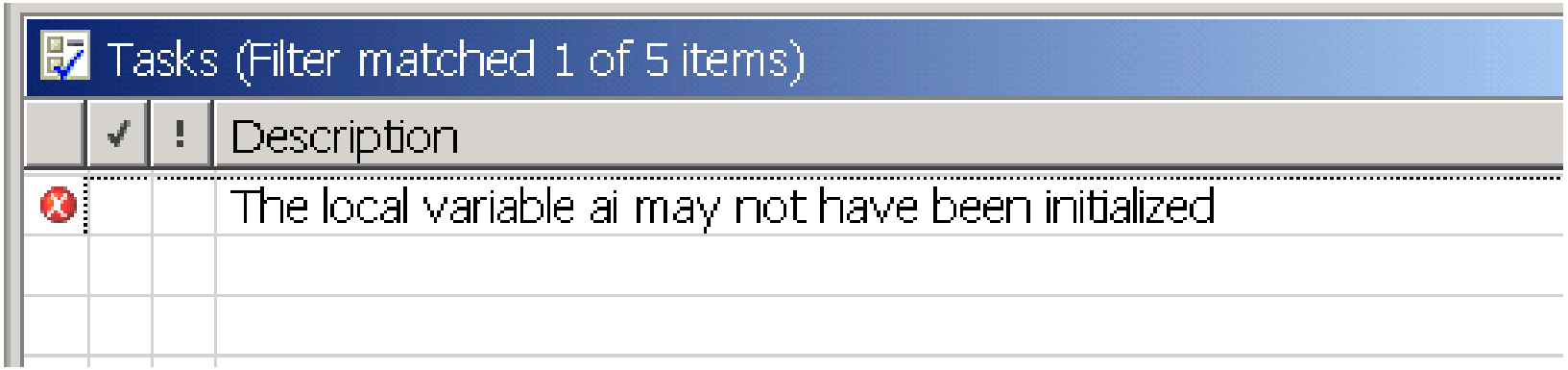
```
int [ ] ai;
...
println( ai[0] );      → Variable ai wurde noch nicht initialisiert
println( ai[1] );      → Variable ai wurde noch nicht initialisiert
println( ai[...] );    → Variable ai wurde noch nicht initialisiert

println( ai.length ); → Variable ai wurde noch nicht initialisiert
```

Fehler beim Zugriff auf Arrays

- Erfolgt ein Zugriff auf eine Komponente eines Arrays, das noch *nicht initialisiert* wurde, meldet der *Compiler* einen Fehler



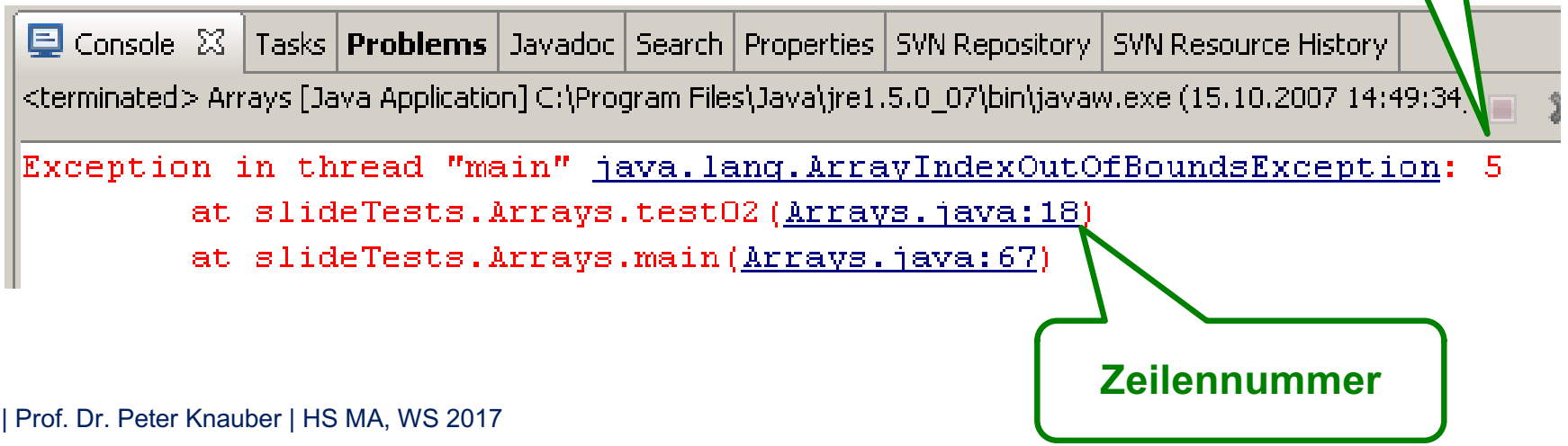
```
int [] ai;
ai[3] = 12;
```



Fehler beim Zugriff auf Array-Elemente

- Beim Zugriff auf eine *ungültige Index-Position* liefert die *Virtuelle Maschine* einen Laufzeitfehler
  ArrayIndexOutOfBoundsException
  und meldet den Index und die Position (Datei und Zeilennummer) im Quelltext

```
int [] ai = new int[5];
ai[5] = 12;
```



Zwischenfrage

- Diskutieren Sie kurz mit Ihrem Nebenmann

1. Warum kommt eine der besprochenen Fehlermeldungen von Compiler, die andere von der VM?
2. Könnten beide vom Compiler kommen?
3. Könnten beide von der VM kommen?

- Haben Sie Fragen?

Typisches Missverständnis

- Bei der Zuweisung eines Array-Wertes werden keine Inhalte (des alten Wertes) übernommen!
- Beispiel

```
int [ ] zahlen = new int[5];
zahlen[2] = 42;
int i = 0;
while ( i < zahlen.length ) {
        print( zahlen[i] + "," );
        i++;
}                                   →

zahlen = new int[7];
int i = 0;
while ( i < zahlen.length ) {
        print( zahlen[i] + "," );
        i++;
}                                   →
```

Diese Arrays (diese beiden Array-Werte) haben nichts miteinander zu tun! Daher muss u.a. die Länge bei einer Zuweisung nicht mit der vorherigen Länge übereinstimmen!

Ist die ***Länge*** eines Arrays einmal festgelegt, kann sie ***nie*** wieder ***geändert*** werden!

Man kann ein *neues Array* (= einen neuen Array-Wert) anlegen und zuweisen, das aber nichts mit dem alten zu tun hat und daher auch keine der bisherigen Inhalte "übernimmt"...

Haben Sie Fragen?

Kopieren von Arrays: gleichlange Arrays

```
int[] alt, neu;

alt = new int[16];
neu = new int[16];

int i=0;
while ( i < alt.length ) {
    alt[i] = i;
    i++;
}

i=0;
while ( i < alt.length && i < neu.length ) {
    neu[i] = alt[i];
    i++;
}
```

das Array wird erstmalig mit Werten belegt

hier wird kopiert

Ausgabe der Arrays:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Kopieren von Arrays: neues Array kürzer

```
int[] alt, neu;

alt = new int[18];
neu = new int[16];

int i=0;
while ( i < alt.length ) {
    alt[i] = i;
    i++;
}

i=0;
while ( i < alt.length && i < neu.length ) {
    neu[i] = alt[i];
    i++;
}
```

das Array wird erstmalig mit Werten belegt

hier wird kopiert

Ausgabe der Arrays:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Kopieren von Arrays: neues Array länger

```
int[] alt, neu;

alt = new int[16];
neu = new int[18];

int i=0;
while ( i < alt.length ) {
    alt[i] = i;
    i++;
}

i=0;
while ( i < alt.length && i < neu.length ) {
    neu[i] = alt[i];
    i++;
}
```

das Array wird erstmalig mit Werten belegt

hier wird kopiert

Ausgabe der Arrays:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 0]
```

Kopieren von Arrays: elementweise

- Anderes Beispiel

```
int [ ] ia1 = new int[3], ia2 = new int[3];        2 Arrays mit 3 Elementen

int i = 0;
while ( i < ia2.length )                           ia2 wird initialisiert mit ...
        ia2[i] = 6-i++;                            → ?


i = 0;
while ( i < ia1.length && i < ia2.length )         elementweises Kopieren
        ia1[i] = ia2[i++];                         von ia2 nach ia1


i = 0;
while ( i < ia1.length )
        print( ia1[i++] + ", " );                  →

println( ia1.length );                             →

ia1[1] = 12;                                       Ändern eines Wertes in ia1...
println( ia1[1] );                                 →

println( ia2[1] );                                 →
```

Haben Sie Fragen?

Arrays als Parameter

- Bisher haben wir nur einfache Typen als Parameter und Ergebnistyp von Methoden kennen gelernt
- Oft ist es jedoch notwendig oder wünschenswert, ein ganzes Array als Parameter zu übergeben oder als Ergebnis zu liefern
  Beispiel: Arbeiten mit Adressen

  Es wäre praktisch, zu schreiben

  ```
  int pos = searchPosition( name, names );
  ```

  wobei *name* als String und *names* als String-Array vereinbart ist

Übergabe von Arrays als Parameter

- Funktionen können ein oder mehrere Arrays als Parameter erhalten
- Die *Übergabe* von Arrays als aktuelle Parameter an formale Parameter *entspricht* einer *Array-Zuweisung*; das bedeutet;
- Dem formalen Parameter wird nur der *Deskriptor* des aktuellen Parameters übergeben (aus Effizienzgründen *keine* Kopie des Arrays!), was zur Aliasierung führt
- Damit sind so genannte *Seiteneffekte* möglich (dazu später mehr)

Ein Array als Rückgabewert einer Funktion

- Funktionen können ein *Array als Rückgabewert* liefern

- Bemerkung:
  Wird das gelieferte Array innerhalb der Funktion angelegt, so kann keine Aliasierung eintreten

Arrays als Rückgabewerte von Funktionen: keine Aliasierung möglich

In der Funktion wird ein neues Array angelegt

```
static String[] readStrings( int anzahl, String string ) {
    String[] res = new String[anzahl];
    int i = 0;
    while (i < anzahl) {
        print("Geben Sie den " + (i + 1) + ". " + string + " ein: ");
        res[i] = readLine();
        i++;
    }
    return res;
}
```

Haben Sie Fragen?

Übler Programmierstil: Operatoren & und &&

```
/* Die Methode tut etwas Unschönes und
 * gibt etwas Unsinniges zurück
 */
static boolean changeArray( int[] numbers ) {
    numbers[2] = 4712;
    return true;
}

public static void main( String[] args ) {
    int[] numbers = { 1, 1, 1, 1 };

    boolean b = false;
    if (b && changeArray(numbers))
        println("Bedingung erfüllt");
    println("Wert an der Stelle 2 == " + numbers[2]);

    if (b & changeArray(numbers))
        println("Bedingung erfüllt");
    println("Wert an der Stelle 2 == " + numbers[2]);
}
```

Beispiel:
Wir wollen das Maximum einer *beliebig großen* Anzahl von Zahlen bestimmen

```
static int maxAlt( int[] values ) {
    int max = Integer.MIN_VALUE;
    for ( int i : values )
        if ( i > max )
            max = i;
    return max;
}
```

Vorbereitung und Aufruf für 3 Zahlen

```
int[] param = new int[3];
param[0] = 3;
param[1] = 1;
param[2] = 5;
println(maxAlt(param));
```

Vorbereitung und Aufruf für 2 Zahlen

```
param = new int[2];
param[0] = -1;
param[1] = -2;
println(maxAlt(param));
```

Beispiel

```
static int maxNeu( int... values ) {
    int max = Integer.MIN_VALUE;
    for ( int i : values )
        if ( i > max )
            max = i;
    return max;
}
```

```
int[] param = new int[3];
param[0] = 3;
param[1] = 1;
param[2] = 5;
println(maxAlt(param));
```

```
param = new int[2];
param[0] = -1;
param[1] = -2;
println(maxAlt(param));
```

neuer, einfacher Aufruf

```
println(maxNeu(3, 1, 5));
println(maxNeu(-1, -2));
```

Formalia

- Ein Rest-Parameter erlaubt es, eine *beliebige Anzahl von aktuellen Parametern* eines (einzigen) Typs an eine Methode zu übergeben
- Ein *Rest*-Parameter steht immer *als letzter* in der formalen Parameterliste (weil er *alle restlichen* aktuellen Parameter übernimmt)
- Folge: Eine Methode kann *maximal einen* Rest-Parameter akzeptieren
- Alle aktuellen Parameter müssen zuweisungskompatibel zum Typ des formalen Parameters sein
- Der Compiler erzeugt automatisch Code, um ein Array der passenden Länge zu erzeugen und die aktuellen Parameter dort einzutragen;
  dieses Array wird an die Methode übergeben
- In der Methode wird auf dieses Array zugegriffen