

Arrays

- Nutzen: zur komfortablen Verwaltung einer sehr großen Menge an Variablen
- Motivation zur Nutzung von Array
 - Speichere die Namen von 5 Mitarbeitern in einem Programm in den Variablen name1..name5
 - Speichere die Namen von 100.000 Mitarbeitern in einem Programm
 - Sehr großer Aufwand bei der Programmerstellung
 - Sehr unflexibel bei Änderungen der Anzahl von Mitarbeitern

Arrays

- Initialisierung eines Arrays
- Speicherbelegung
- Zugriff auf Elemente des Arrays
- Call by Value VS Call by Reference

Arrays

- werden genutzt, um mehrere Variablen gleichen Datentyps komfortabel zu speichern
- die Größe von Arrays muss bei Definition der Variable bekannt sein
 - `int[] meinArray = new int[10];`
 - ein Array ist ein komplexer Datentyp
- die Länge eines Arrays ist in der Eigenschaft `length` gespeichert
 - `System.out.println(meinArray.length);`

Speicherbelegung

Index	Adresse	Name	Datentyp	Wert	Referenz
-------	---------	------	----------	------	----------

100

101

0	102		:int	0
1	103		:int	0
2	104		:int	0
3	105		:int	0
4	106		:int	0
5	107		:int	0
6	108		:int	0
7	109		:int	0
8	110		:int	0
9	111		:int	0

meinArray

112

113

114	meinArray	:int[]	102
-----	-----------	--------	-----

115

116

117

118

Zugriff auf Elemente

- Das i-te Element wird mit `array[i]` lokalisiert
 - Schreiben an i-te Position `meinarray[i]=10`
 - Lesen aus i-ter Position
`System.out.println(meinarray[i])`
- Task: erzeugen Sie ein Array mit den ersten 100 natürlichen Zahlen und geben sie diese auf der Console aus

Call by Value vs Call by Reference

- Call by Value: Übergabe eines Variablenwertes an Funktionseingabevariablen (bisher der Standard in der Vorlesung)
 - Lokale Manipulationen sind außerhalb der Funktion nicht sichtbar
- Call by Reference: Übergabe eines Arrays an Funktionseingabevariablen
 - Lokale Manipulationen an den Slots des Arrays bleiben außerhalb der Funktion erhalten

Call by Value VS Call by Reference

//call by reference

```
public static void myFunction(int[] aArray)
{
    aArray[0]=10;
}
```

//call by value

```
public static void myFunction(int a)
{
    a=10;
}
```

```
public static void main(String[] args) {
    //initialise 2 int-Variablen und setze diese auf 100
    int[] meinArray=new int[1];
    meinArray[0]=100;

    int meineVariableA=100;

    //call functions
    myFunction(meinArray); //changes meinArray[0] to 10
    myFunction(meineVariableA); //does nothing
}
```

Task Array 1

- Erstellen Sie eine Funktion
 - `function createArray():int[]`
 - die ein Int-Array erzeugt und mit aufsteigend gefüllten Zahlen befüllt
 - beginnend bei -10000 bis +10000
- Erstellen Sie eine Funktion
 - `function showArray(array:int[]):void`
 - die alle Elemente eines Arrays auf der Konsole ausgibt
- erzeugen sie in der Main das Array und geben sie es dann auf der Konsole aus

Task Array 2

- Schreiben Sie eine Funktion
 - `function getMaxNumber(array: int[]):int`
 - welche die größte Zahl in einem gegebenem Array findet
- Schreiben Sie eine Funktion
 - `function getMaxNumberIndex(array: int[]):int`
 - welche die Indexposition der größte Zahl in einem gegebenem Array zurückgibt
- testen sie die Funktionen mit von ihnen befüllten Testarrays

Task Array 3

- Schreiben Sie eine Funktion
 - `function invertArray(array:int[]):int[]`
 - die ein beliebig langes Int-Array als Eingabe erhält
 - und ein neues Array mit invertierter Reihenfolge erzeugt
- Schreiben Sie eine Funktion
 - `function invertArray2(array:int[]):void`
 - die ein beliebig langes Int-Array als Eingabe erhält
 - und die Reihenfolge der Elemente im Array umkehrt

Task Typkonvertierung & Array

- Schreiben Sie eine Funktion
 - `function createRandArray(length:int):float[]`
 - welche ein array der Länge length mit Zufallszahlen zwischen 0.0 und 100.0 zufällig füllt.
 - Mit der Funktion `Math.random()` wird ein double-Zufallswert z erzeugt im Bereich $0.0d \leq z < 1.0d$
- Testen Sie die obige Funktion mit Hilfe der bereits bekannten Funktion `showArray` (diese muss auf float-Array angepasst werden)