4. Arrays

Prof. Dr. Charles Wüthrich
B.Sc. Francesco Andreussi,
CoGVis/MMC, Faculty of Media
Bauhaus-University Weimar

Introduction

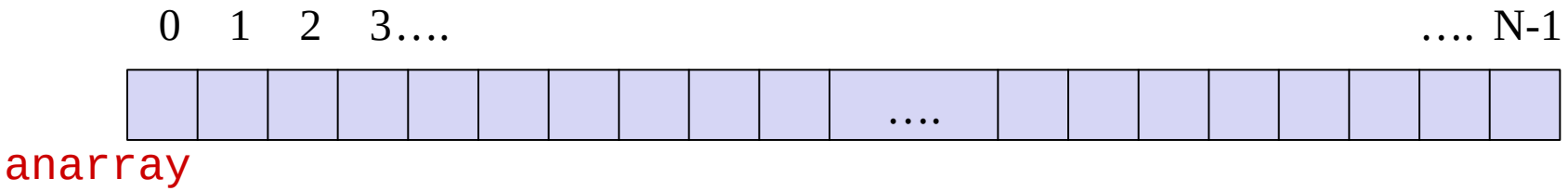
- Now we know how to change the program flow
- We also know how to make programs with individual variables
- Listing, even declaring, single variables is lengthy and tedious when we have lots of data

- On top, many operations in programming are done on groups of variables
- For example, repetitive operations on different data…

- …. This is why *arrays* were invented

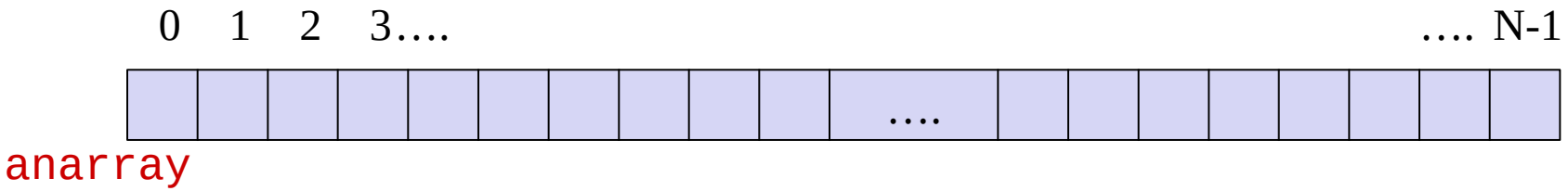
Arrays

- An *array* is a sequence of consecutive cells of a single data type
- An array has a size, which is the number N of its cells

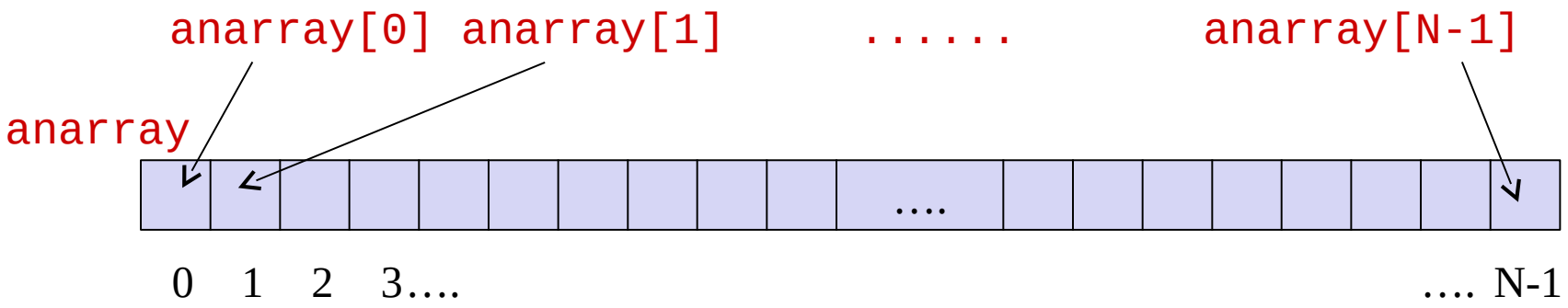


Arrays

- An *array* is a sequence of consecutive cells of a single data type
- An array has a size, which is the number N of its cells



- Its cells are indicated with the name of the array and the cell position, starting from 0:



Arrays

- Initiating arrays so they are ready to use is a multiple step procedure:
  - Declare the type of the array
    ```
    float[] anarray;
    ```
    so the compiler knows that `anarray` is an array of floats
  - Reserve the right amount of memory for the array
    ```
    anarray = new float[9]
    ```
    this reserves an array of 9 floating point numbers (Allocation)
  - From now on, the elements of the array can be addressed as `anarray[i]`, where `i` is any integer between `0` and `8`.
  - If necessary, initialize the array:
    ```
    for(i=0;i<9,i++){
       anarray[i]=0.0f;
    }
    ```

Arrays: Example

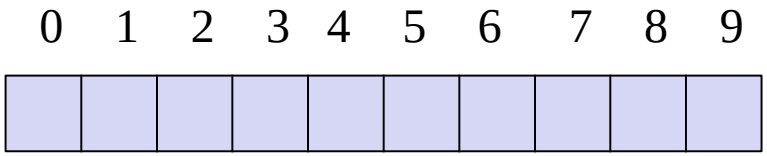
- Let us program arrays together: Exercise!
- We want to write a program that does the following:
  - Allocate an array of integers with 10 components
  - Fill the array with numbers from 1 to 10 in growing order
    1,2,3,4,5,6,7,8,9,10
  - Exchange
    - first and the last number
    - Second and one before the last
    - Until the whole array is “inverted”:
      10,9,8,7,6,5,4,3,2,1

Arrays: Example

- Let us program arrays together: Exercise!
- We want to write a program that does the following:
  - Allocate an array of integers with 10 components
  - Fill the array with numbers from 1 to 10 in growing order
    1,2,3,4,5,6,7,8,9,10
  - Exchange
    - first and the last number
    - Second and one before the last
    - Until the whole array is “inverted”:
      10,9,8,7,6,5,4,3,2,1
- HOW?

Arrays: Example

- Work and think together!
- We want to create this:

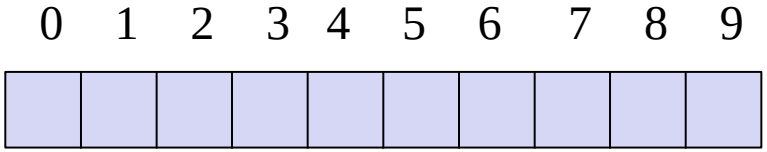


- …. And then fill it up

Arrays: Example

- Step ONE: create program with its name:

- We want to create this:

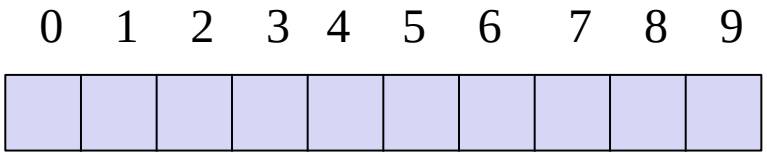


Arrays: Example

- Step ONE: create program with its name:

```
public class ReorderArray {
   public static void main(String[] args) {
      Placeholder;
   }
}
```

- We want to create this:

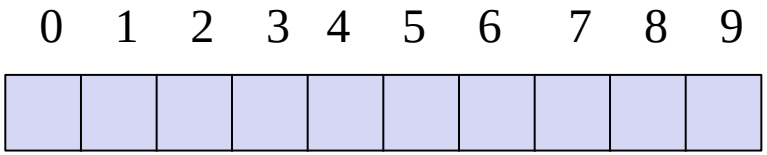


Arrays: Example

- Step TWO: which variables do I need?
  - An array of integers:
    - Declare name
    - Allocate space: how much? 10

```
public class ReorderArray {
   public static void main(String[] args) {
      int[] numbersarray;

      numbersarray = new float[10];
   }
  }
```

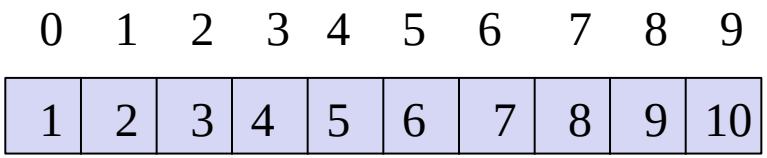


Arrays: Example

- Step THREE: Fill the array with 1,2,3,4,5,6,7,8,9,10
  - This we can do with FOR
  - we need an index variable `i`:

```
public class ReorderArray {
   public static void main(String[] args) {
      int[] numbers;
      int i;

      numbers = new float[10];
      for(i=0;, i<9; i++){
         numbers[i]=i+1;
      }
   }
}
```

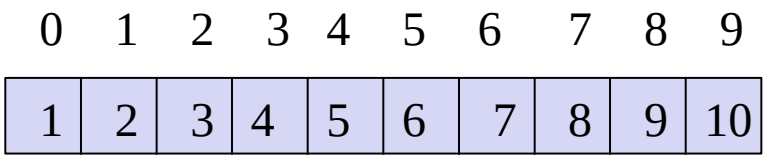


Arrays: Example

- Step FOUR: swap elements:
  - first element with last,
  - 2nd with forelast…
  - Looks like another FOR!
  - UNTIL?

```
public class ReorderArray {
   public static void main(String[] args) {
      int[] numbers;
      int i;

      numbers = new float[10];
      for(i=0;, i<9; i++){
         numbers[i]=i+1;
      }
   }
}
```

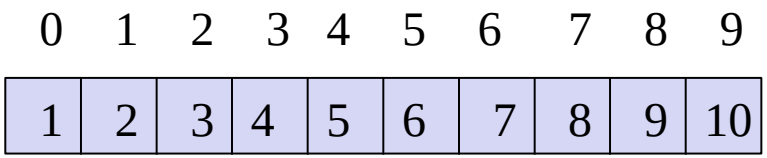


Arrays: Example

- Step FOUR: swap elements:
  - first element with last,
  - 2nd with forelast…
  - Looks like another FOR!
  - UNTIL? Not end of array, otherwise we swap twice!

```
public class ReorderArray {
   public static void main(String[] args) {
      int[] numbers;
      int i;

      numbers = new float[10];
      for(i=0;, i<9; i++){
         numbers[i]=i+1;
      }
   }
}
```

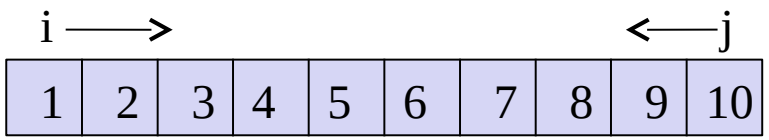


Arrays: Example

- Step FOUR: swap elements:
  - For the FOR I need two indexes! Front (`i`) and Back (`j`)!
  - If I swap I need a swapping integer, and `j` must start from back

```
public class ReorderArray {
   public static void main(String[] args) {
      int[] numbers;
      int i,j,swap;

      numbers = new float[10];
      for(i=0;, i<9; i++){
         numbers[i]=i+1;
      }
      j=10;
      for(i=0;i<(9/2);i++){
         j=j-1;swap=numbers[i];
         numbers[i]=numbers[j];numbers[j]=swap;
      }
   }
}
```



Arrays: Example

```
public class ReorderArray {
  public static void main(String[] args) {
    int[] numbers; // the array
    int size=10; // size of the array

    int i,j,swap; // indexes and swap variable

    // initialize array
    numbers = new int[size];

    // fill array
    for(i=0; i<size; i++){
       numbers[i]=i+1;
    }

    // print array
    for(i=0; i<size; i++){
       System.out.print(numbers[i]+" ");
    }
    System.out.printf("%n");

    // swap cell contents
    // only to half array length!
    j=size;
    for(i=0; i<(size/2); i++){
       j=j-1;
       swap=numbers[i];
       numbers[i]=numbers[j];
       numbers[j]=swap;
    }

    // print array again to check result
    for(i=0; i<size; i++){
       System.out.print(numbers[i]+" ");
    }
    System.out.printf("%n");
  } // main
} // ReorderArray
```

Arrays: Execution

This is a CLI execution of the program:

```
> java ReorderArray
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
>
```

Arrays: Execution

- This is a CLI execution of the program:

```
> java ReorderArray
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
>
```

Before

After

Arrays: 2-dimensional

- I can also have two indexes ⇒ matrix, table

```
int a2array[][] = new int[8][5];
```

- Here we condensed declaration and allocation in one line

10	2	22	13	15
17	-9	11	22	11
15	31	117	-45	21
24	33	10	12	10
-112	21	39	63	91
26	48	953	19	23
31	38	73	15	54
72	21	33	-127	-112

Arrays: 2-dimensional

- In a two-dimensional array, one can also assign values during the variable declaration:

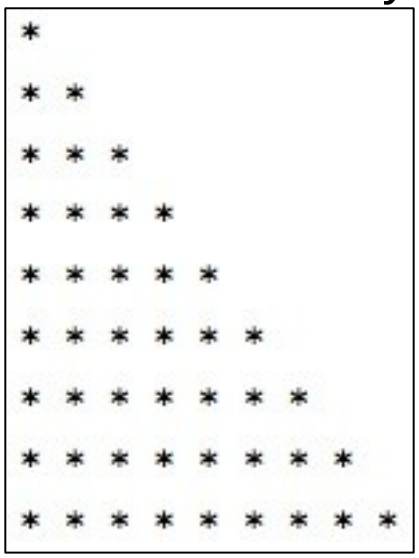
```
int anintarray[][] = {
    {12, 5, 3, 7},
    {3, 2, 17, 24},
    {15, 7, 18, 22},
    {18, 11, 23, 27},
};
```

results into the following array:

12	5	3	7
3	2	17	24
15	7	18	22
18	11	23	27

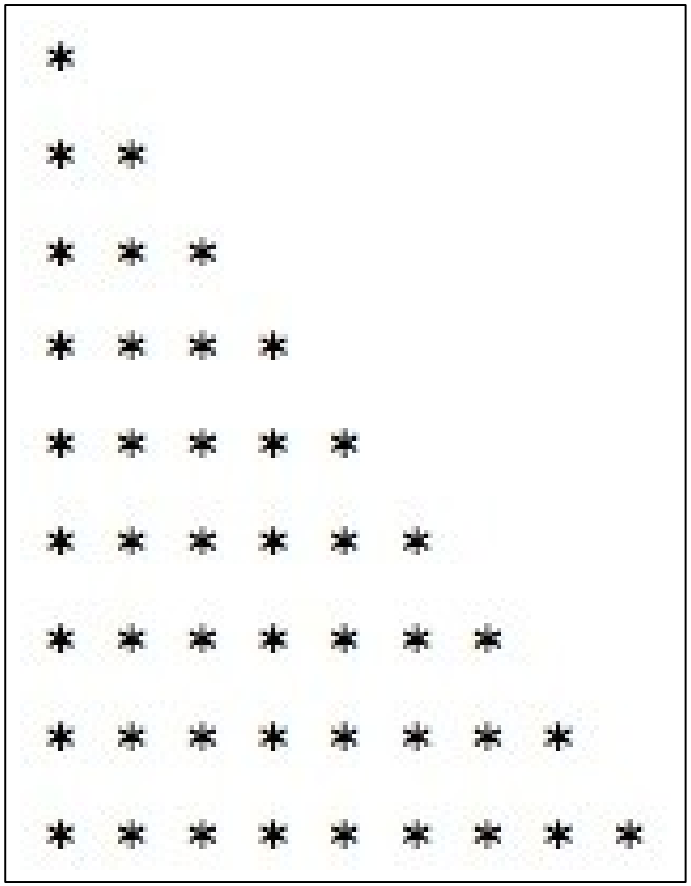
Arrays: 2-dimensional

- Let us write the following program:
  - Create a 2-dim `char` array with 10 rows and 10 columns
  - Fill it it with blank characters
  - Now replace the blanks with * characters so that
  - 1st row has one * at the beginning
  - 2nd row has ** at the beginning
    and so on, till the last line only has all * characters, like this:



Arrays: 2-dimensional

- Help me to do this:



Arrays: 2-dimensional

```
public class FillWithStars {
  public static void main(String[] args) {
    char[][] arrchar; // the array
    int size=10; // size of the array rows and columns
    int i,j; // indexes

    // allocate array
    arrchar = new char[size][size];

    // fill array
    for(i=0; i<size; i++){
       for(j=0;j<size;j++){
         arrchar[i][j]=' ';
       }
    }

    // now fill with *
    for(i=0; i<size; i++){
       for(j=0; j<i; j++){
         arrchar[i][j]='*';
       }
    }

    // print array
    for(i=0; i<size; i++){
       for(j=0;j<size;j++){
         System.out.printf("%s ",arrchar[i][j]);
       }
       System.out.printf("%n");
    }
  } // main
} // FillWithStars
```

Arrays: 3-dimensional

- Java does not restrict only to two coordinates:

```
float[][][]3darray = new float[5][5][5];
```

declares a 3d array with 125 cells of floats

- In this case, the individual cells have to be addressed with 3 indexes

```
3darray[i][j][k]
```

Strings as arrays

- In case you did not notice, an array of `char` is a String
- Moreover, the Operating System makes sure that when you type after the program name in the CLI strings, then
  - Each string after the program name will be stored in a vector called `args`, so
    - `args[0]` contains the first word after the program name when executing the program
    - `args[1]` contains the 2nd word after the program name when executing the program
    - … and so on…
- … which explains a lot of what looked cryptic till now

End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++