Arrays

Learning objectives

- You know what an Array is and how to use it.
- You can create an Array, assign a value to a given index and iterate over it.
- You recognize that an Array is a reference type and know how to use an array as a parameter of a method

We've gotten familiar with the ArrayList, which has a lot of functionality to make the life of a programmer easier. Perhaps the most important is about adding elements. From the point of view of the programmer, the size of the ArrayList is unlimited. In reality there are no magic tricks in the ArrayList — they have been programmed like any other programs or tools offered by the programming language. When you create a list, a limited space is reserved in the memory of the computer. When the ArrayList runs out of space, a larger space is reserved and the data from the previous space is copied to the new one.

Even though the ArrayList is simple to use, sometimes we need the ancestor of the ArrayList, the Array.

An Array contains a limited amount of numbered spots (indices) for values. The length (or size) of an Array is the amount of these spots, i.e. how many values can you place in the Array. The values in an Array are called **elements**.

Creating an Array

There are two ways to create an Array. In the first one you have to explicitly define the size upon the creating. This is how you create an Array to hold three integers:

```
int[] numbers = new int[3];
```

An array is declared by adding square brackets after the type of the elements it contains (typeofelements[]). A new Array is created by calling `new` followed by the type of the elements, square brackets and the number of the elements in the square brackets.

An Array to hold 5 Strings can be created like this:

```
String[] strings = new String[5];
```

Assigning and accessing elements

An element of an Array is referred to by its index. In the example below we create an Array to hold 3 integers, and then assign values to indices 0 and 2. After that we print the values.

```
int[] numbers = new int[3];
numbers[0] = 2;
numbers[2] = 5;

System.out.println(numbers[0]);
System.out.println(numbers[2]);
```

Assigning a value to a specific spot of an Array works much like assigning a value in a normal variable, but in the Array you must specify the index, i.e. to which spot you want to assign the value. The index is specified in square brackets. The ArrayList `get`-method works very similarly to accessing an element of an Array. Just the syntax, i.e. the way things are written, is different.

The index is an integer, and its value is between [0, length of the Array - 1]. For example an Array to hold 5 elements has indices 0, 1, 2, 3, and 4.

```
Scanner reader = new Scanner(System.in);

int[] numbers = new int[5];
numbers[0] = 42;
numbers[1] = 13;
numbers[2] = 12;
numbers[3] = 7;
numbers[4] = 1;

System.out.println("Which index should we access? ");
int index = Integer.valueOf(reader.nextLine());

System.out.println(numbers[index]);
```

The value held in an Array can also be assigned to be the value of another variable.

```
Scanner reader = new Scanner(System.in);

int[] numbers = new int[5];
numbers[0] = 42;
numbers[1] = 13;
numbers[2] = 12;
numbers[3] = 7;
numbers[4] = 1;
```

```
System.out.println("Which index should we access? ");
int index = Integer.valueOf(reader.nextLine());

int number = numbers[index];
System.out.println(number);
```

Quiz:
Swapping array values

Points:
1/1

Below there is an array that contains integers:

```
int[] numbers= new int[5];
numbers[0] = 42;
numbers[1] = 13;
numbers[2] = 12;
numbers[3] = 7;
numbers[4] = 1;

// code here
```

Which of the following snippets of code swaps the values at indices 0 and 1 (the value originally at index 0 is now at index 1, and vice versa)?

Select the correct answer

```
numbers[0] = numbers[1];
```

```
numbers[1] = numbers[0];
```

```
numbers[0] = numbers[1];
numbers[1] = numbers[0];
```

```
int helper = numbers[0];
numbers[0] = numbers[1];
numbers[1] = helper;
```

```
int helper = numbers[0];
numbers[1] = numbers[0];
numbers[0] = helper;
```

The answer is correct

Answered Tries remaining: 1

Programming exercise:

Swap

Points 1/1

The exercise template already contains a program, that creates an array and prints the values of the array twice. Modify the program to do following: After the first printing, the program should ask for two indices from the user. The values in these two indices should be swapped, and in the end the values of the array should be printed once again.

Sample output

```
1
3
5
7
9

Give two indices to swap:
2
4

1
3
9
```

```
7
5
```

Sample output

```
1
3
5
7
9

Give two indices to swap:
0
1

3
1
5
7
9
```

You can assume that the array contains the given indices. Tip! You'll need an additional variable to store one of the values for a little while.

Exercise submission instructions

How to see the solution

Size of an array and iterating

You can find the size of the array through the associated variable `length`. You can access this associated variable by writing name of the array dot name of the variable, i.e. `numbers.length`. Note, that this is not a method call, so `numbers.length()` doesn't work.

You can iterate over the array, i.e. go through each element of the array with a while loop.

```
int[] numbers = new int[4];
numbers[0] = 42;
numbers[1] = 13;
numbers[2] = 12;
```

```
numbers[3] = 7;

System.out.println("The array has " + numbers.length + " elements.");

int index = 0;
while (index < numbers.length) {
    System.out.println(numbers[index]);
    index = index + 1;
}
```

Sample output

```
The array has 4 elements.
42
13
12
7
```

The example above iterates over indices 0, 1, 2 and 3, and at each index prints the value held in the index. So first it prints `numbers[0]`, then `numbers[1]` etc. The iteration stops, once the condition of the loop `index < number.length` is false, i.e. once the index variable is greater or equal to the length of the array. NB! The iteration doesn't stop the moment the index variable grows too big; the condition is evaluated only in the beginning of the while loop.

```
1  public class Example {
2      public static void main(String[] args) {
3          int[] numbers = new int[4];
4          numbers[0] = 42;
5          numbers[1] = 13;
6          numbers[2] = 12;
7          numbers[3] = 7;
8
9          System.out.println("There are " + numbers.length + " elements in the array.");
10
11         int index = 0;
12         while (index < numbers.length) {
13             System.out.println(numbers[index]);
14             index++;
15         }
16     }
17 }
```

Output

Programming exercise:

Index was not found

Points 1/1

The exercise template already has an array containing numbers. Complete the program so that it asks the user for a number to search in the array. If the array contains the given number, the program tells the index containing the number. If the array doesn't contain the given number, the program will advise that the number wasn't found.

Sample output

```
Search for? 3
3 is at index 4.
```

Sample output

```
Search for? 7
7 is at index 7.
```

Sample output

```
Search for? 22
22 was not found.
```

Exercise submission instructions

How to see the solution

If the index is pointing outside the Array, i.e. the element doesn't exist, we get an **ArrayIndexOutOfBoundsException**. This error tells, that the Array doesn't contain the given index. You cannot access outside of the Array, i.e. index that's less than 0 or greater or equal to the size of the Array.

The next example is a program that initially asks the user to enter how many numbers, and then enter the numbers. Finally it prints back the numbers in the same

order. The numbers are stored in an Array.

```
System.out.print("How many numbers? ");
int howMany = Integer.valueOf(reader.nextLine());

int[] numbers = new int[howMany];

System.out.println("Enter the numbers:");

int index = 0;
while (index < numbers.length) {
    numbers[index] = Integer.valueOf(reader.nextLine());
    index = index + 1;
}


System.out.println("Here are the numbers again:");

index = 0;
while (index < numbers.length) {
    System.out.println(numbers[index]);
    index = index + 1;
}
```

An execution of the program might look like this:

Sample output

```
How many numbers? 4
Enter the numbers:
4
8
2
1
Here are the numbers again:
4
8
2
1
```

Type of the elements

You can create an array stating the type of the elements of the array followed by square brackets (typeofelements[]). Therefore the elements of the array can be of any type. Here's a few examples:

```
String[] months = new String[12];
double[] approximations = new double[100];
```

```
months[0] = "January";
approximations[0] = 3.14;
```

Indices and the structure of the memory

Every programmer should know a bit about the structure of the memory used by a computer program. Each variable — let it be primitive or reference type — is saved in the memory. Each variable has size i.e. a defined number of bits (zeros and ones) it takes in the memory. The value of the variable is also represented in bits.

The reference of the array object is actually information about the location of the data. By stating `array[0]` we're referring to the first element of the array. The statement `array[0]` can also be read "Go to the beginning of the array and move forward 0 times the size of the variable contained in the array — and return a chunk of data the size of the variable.

The size of an int variable in java is 32 bits. One bit is reserved for the sign, so the largest possible number to present in int is 2^{31}-1. When you create an int array of 4 elements, 4 * 32 bits of memory is allocated to hold the integers. When you access `array[2]`, 32 bits are read starting from beginning of the array + 2 * 32 bits. Some programming languages try to make sure that the programmer doesn't go "in the wrong area". If java didn't cause the exception when we say `array[-1]`, we would find the data located just before the array in the memory of the program. In such case there wouldn't be anything preventing us from writing a program reading the whole memory reserved for the program.

Array as a parameter of a method

You can use arrays as a parameter of a method just like any other variable. As an array is a reference type, the value of the array is a reference to the information contained in the array. When you use array as a parameter of a method, the method receives a copy of the reference to the array.

```
public static void listElements(int[] integerArray) {
    System.out.println("the elements of the array are: ");
    int index = 0;
    while (index < integerArray.length) {
        int number = integerArray[index];
        System.out.print(number + " ");
        index = index + 1;
    }

    System.out.println("");
}
```

```
int[] numbers = new int[3];
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;

listElements(numbers);
```

Sample output

```
the elements of the array are: 1 2 3
```

As noticed earlier, you can freely choose the name of the parameter inside the method, the name doesn't have to be the same as the name of the variable when you call the method. In the example above, we call the array `integerArray`, meanwhile the caller of the method has named the same array `numbers`.

Array is an object, so when you change the array inside the method, the changes persist after the execution of the method.

Programming exercise:

Sum of array

Points

1/1

The class SumOfArray has a method `public static int sumOfNumbersInArray(int[] array)`. Complete the method so that it computes and returns the sum of the numbers in the array it receives as parameter.

You can try out the computation of the sum with this example:

```
int[] numbers = {5, 1, 3, 4, 2};
sumOfNumbersInArray(numbers);
```

Sample output

```
15
```

Exercise submission instructions

How to see the solution

Programming exercise:

Print neatly

Points
1/1

Complete the method `public static void printNeatly(int[] array)` in the class named 'ArrayPrinter' to make it print the numbers of the array it receives more neatly. There should be a whitespace and a comma between each number. don't put a comma after the last number.

Print the numbers on one line using `System.out.print`.

You can try out your printing with this example:

```
int[] array = {5, 1, 3, 4, 2};
printNeatly(array);
```

Sample output

```
5, 1, 3, 4, 2
```

Exercise submission instructions

How to see the solution

Programming exercise:

Print in stars

Points
1/1

Complete the method `public static void printArrayInStars(int[] array)` in the class named 'Printer' to make it print a row of stars for each number in the array. The amount of stars on each row is defined by the corresponding number in the array.

You can try out the printing with this example:

```
int[] array = {5, 1, 3, 4, 2};
printArrayInStars(array);
```

Sample output

```
*****
```

```
*
***
****
**
```

The 0th element of the array is 5, so the first line has 5 stars. The next one has 1 etc.

Exercise submission instructions

How to see the solution

The shorter way to create an array

Just like for Strings, there's also a shortcut to create an array. Here we create an array to hold 3 integers, and initiate it with values 100, 1 and 42 in it:

```
int[] numbers = {100, 1, 42};
```

So apart from calling for `new`, we can also initialize an array with a block, that contains comma-separated values to be assigned in the array. This works for all the types: below we initialize an array of strings, then an array of floating-point numbers. Finally the values are printed.

```
String[] arrayOfStrings = {"Matti L.", "Matti P.", "Matti V."};
double[] arrayOfFloatingpoints = {1.20, 3.14, 100.0, 0.6666666667};

for (int i = 0; i < arrayOfStrings.length; i++) {
    System.out.println(arrayOfStrings[i] + " " +  arrayOfFloatingpoints[i]);
}
```

Sample output

```
Matti L. 1.20
Matti P. 3.14
Matti V. 100.0
```

When you initialize an array with a block, the length of the array is precisely the number of the values specified in the block. The values of the block are assigned to the array in the order, eg. the first value is assigned to index 0, the second value to index 1 etc.

```
// index           0   1    2    3   4   5     6     7
int[] numbers = {100,  1,  42,  23,  1,  1, 3200, 3201};

// prints the number at index 0, i.e. number 100
System.out.println(numbers[0]);
// prints the number at index 2, i.e. number 42
System.out.println(numbers[2]);
```

Just like in ArrayLists, you can't access an index outside of the array. You can try out the following example on your own computer, for example in the sandbox.

```
String[] arrayOfStrings = {"Matti L.", "Matti P.", "Matti V."};
double[] arrayOfFloatingpoints = {1.20, 3.14, 100.0, 0.6666666667};

for (int i = 0; i < arrayOfFloatingpoints.length; i++) {
    System.out.println(arrayOfStrings[i] + " " +  arrayOfFloatingpoints[i]);
}
```

You have reached the end of this section! Continue to the next section:

→ 4. Using strings

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.




HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI


MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI