Theory: Iterating over arrays

8558 users solved this topic. Latest completion was about 1 hour ago.

53 minutes 5 / 20 problems solved

Start practicing

§1. Processing arrays using loops

Often, it's needed to perform some kind of algorithms on the elements of an array. For instance: sort them, find the maximum element, print only positive numbers, reverse the order, calculate the arithmetic average of numbers and so on.

A convenient way to process an array is to iterate over the array using a loop. The property `length` of an array can help us to avoid `ArrayIndexOutOfBoundsException`.

Example 1. Filling an array with the squares of indexes of its elements.

```
int n = 10; // the size of an array
int[] squares = new int[n]; // creating an array with the specified size


System.out.println(Arrays.toString(squares)); // [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

/* iterating over the array */
for (int i = 0; i < squares.length; i++) {
    squares[i] = i * i; // set the value by the element index
}



System.out.println(Arrays.toString(squares)); // [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In this code, an array with the size 10 and filled with zeros is created. Then the value of each element of the array is set to the square of the element's index. Then the program converts the array to the string representation that lists all elements inside square brackets, and prints it to the standard output.

Example 2. Checking the order of elements.

The following program checks if the given array is sorted ascending order and prints "OK", otherwise it prints "BROKEN".

```
int[] numbers = { 1, 2, 3, 4, 5, 10, 6 }; // the order is broken

boolean broken = false; // suppose the array is well-ordered

/* iterating over the array */
for (int i = 1; i < numbers.length; i++) {

    if (numbers[i] < numbers[i - 1]) { // if the order is broken
        broken = true; // write a result

        break;         // terminate the loop

    }

}



if (broken) {

    System.out.println("BROKEN");

} else {

    System.out.println("OK");

}
```

Current topic:

- ✓ Iterating over arrays

Topic depends on:

- ✓ Branching statements — Stage 2
- ✓ Array — Stage 2

Topic is required for:

- Multi-dimensional array
- Algorithms in Java
- Enums in Java
- Dynamic array in Java
- ArrayList

Table of contents:

For the given array the program prints `"BROKEN"`.

To iterate over arrays while and do-while loops are also admissible, but they are used less often.

§2. Reading an array from the standard input

Using a loop we can read all elements of an array from the standard input.

For example, the input consists of two lines. The first line contains the length of an array, the second line contains all elements of the array.

```
5
101 102 504 302 881
```

Let's read these numbers using the `Scanner` (you can use another tool for reading) and then output all read numbers.

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

int len = scanner.nextInt(); // reading a length

int[] array = new int[len];  // creating an array with the specified length

for (int i = 0; i < len; i++) {
    array[i] = scanner.nextInt(); // read the next number of the array
}


System.out.println(Arrays.toString(array)); // output the array


```

The program outputs:

```
[101, 102, 504, 302, 881]
```

§3. Using for-each loop

Since Java 5 there is a special form of the for-loop called for-each. It is used to iterate through each element of an array, string or a collection (we will learn them in next lessons) without indexes.

Here's how it looks:

```
for (type var : array) {
    //statements using var
}
```

It can be read as: for each element `var` of type `type` in the array `array` do `{` some statements in the body `}`.

Let's consider the constituents in more detail. `type` specifies the type of variable that will store one element of the array in each iteration. Usually, that type equals the type of the array. `var` is the name of that variable. You can choose any name you want here, remember to stick to variable naming conventions though. On the first iteration, it stores the first element of the array, on the second iteration it stores the second element of the array and so on.

Let's write some code for calculating the number of `'a'` letters in the given character array with **for-each loop**:

```
char[] characters = { 'a', 'b', 'c', 'a', 'b', 'c', 'a' };

int counter = 0;
for (char ch : characters) {
    if (ch == 'a') {
        counter++;
    }
}


System.out.println(counter); // it outputs "3"
```

The same thing we can do with for-loop:

```
char[] characters = {'a', 'b', 'c', 'a', 'b', 'c', 'a'};

int counter = 0;
for (int i = 0; i < characters.length; i++) {
    if (characters[i] == 'a') {
        counter++;
    }
}


System.out.println(counter); // it outputs "3"
```

The for-each loop has some limitations. First of all, you cannot use it if you want to modify an array, because the variable we use for iterations doesn't store the array element itself, only its copy. It is also not possible to obtain an element by its index since we have no track of them. Finally, as it follows from the name, we cannot move through an array with the step more than once: we iterate over each and every element, so we do it one by one.

As you can see, the absence of indexes makes the code more readable. The for-each loop also allows you to avoid `ArrayIndexOutOfBoundsException`. That's why it is commonly used for iterating over an array.

Report a typo

836 users liked this theory. **12** didn't like it. **What about you?**

Start practicing

Comments (14) Hints (0) Useful links (0) Show discussion