Theory: Arrays as parameters

19 minutes 0 / 5 problems solved

Skip this topic Start practicing

5276 users solved this topic. Latest completion was about 1 hour ago.

§1. Passing arrays to methods

A method can have parameters of any types including arrays, strings, primitive types and so on.

Here is an example, the method `processArray` has a single parameter of the type `int[]`:

```
public static void processArray(int[] array) { /* do something */ }
```

In the body of the method, we can process the input array in any way.

A parameter of an array type looks like a primitive type parameter. But there is one important difference related to the fact that an array is a reference type.

When you pass a value of a primitive type to a method, a copy of the value is created. When you pass an array to a method, a copy of the reference is created but the value is the same. It means if you change the actual value (elements of an array) in the body of a method, you will see these changes outside the method.

The following method swaps the first and the last elements of its parameter (array).

```
public static void swapFirstAndLastElements(int[] nums) { // nums is an array
    if (nums.length < 1) {
        return; // it returns nothing, i.e. just exits the method
    }

    int temp = nums[nums.length - 1]; // save the last element in temporary local variable
    nums[nums.length - 1] = nums[0];  // now, the last element is the first
    nums[0] = temp;                   // now, the first element is the previous last
}
```

Calling the method from the main method:

```
public static void main(String[] args) {

    int[] numbers = { 1, 2, 3, 4, 5 }; // numbers

    System.out.println(Arrays.toString(numbers)); // before swapping

    swapFirstAndLastElements(numbers); // swapping

    System.out.println(Arrays.toString(numbers)); // after swapping

}
```

The output is:

```
[1, 2, 3, 4, 5]
[5, 2, 3, 4, 1]
```

So, in the body of the main method, an array is visible as modified.

§2. Varargs

Current topic:

Arrays as parameters

Topic depends on:

✓ Declaring a method Stage 3

✓ Array Stage 2

Topic is required for:

Command-line arguments

Table of contents:

It's possible to pass an arbitrary number of the same type arguments to a method using the special syntax named **varargs (variable-length arguments)**. These arguments are specified by three dots after the type. In the body of the method, you can process this parameter as a regular array of the specified type.

The following method takes an integer **vararg** parameter and outputs the number of arguments in the standard output using the **length** property of arrays.

```
public static void printNumberOfArguments(int... numbers) {
    System.out.println(numbers.length);
}
```

As you can see, here is a special syntax **...** is used to specify a **vararg** parameter.

Now, you can invoke the method passing several integer numbers or an array of ints.

```
printNumberOfArguments(1);
printNumberOfArguments(1, 2);
printNumberOfArguments(1, 2, 3);
printNumberOfArguments(new int[] { }); // no arguments here
printNumberOfArguments(new int[] { 1, 2 });
```

This code outputs:

```
1
2
3
0
2
```

This example also demonstrates the difference between the arguments and parameters of a method. The method has only a single parameter but it can be called with several arguments.

§3. Varargs and other parameters

If a method has more than one parameter, a `vararg` parameter must be the last parameter in the declaration of the method.

Here is an incorrect example:

```
public static void method(double... varargs, int a) { /* do something */ }
```

The correct version of the method is:

```
public static void method(int a, double... varargs) { /* do something */ }
```

Report a typo

501 users liked this theory. 11 didn't like it. **What about you?**

Start practicing

Comments (10) Hints (0) Useful links (0) Show discussion