Result of a method

1. The `return` statement

Think you've already learned all about Java methods? Whatever you think you know, you still don't know the half of it.

Let's start with something simple. For example, Java has a **return** statement that lets you instantly terminate the method that calls it. Here's the statement:

```java
return;
```

It's simple: the solitary word `return` followed by a semicolon. As soon as the program executes this statement, the current method exits immediately, and the calling continues.

If `return` is called in the `main` method, then the `main` method will immediately end, and with it the entire program.

Example:

```java
class Solution
{
   public static void fill(int[] data, int from, int to, int value)
   {


     if (from < 0 || to > data.length)
       return;

     for (int i = from; i < to; i++)
     {
       data[i] = value;
     }
   }

   public static void main(String[] args)
   {
     int[] months = {1, 2, 3, 4, 5, 6, 7, 8 ,9, 10, 11, 12};
     fill(months, 2, 10, 8);
   }
}
```

The `fill` method fills part of the passed array with `value`.
The part of the array to be filled is defined by the indices `from` and `to`.
If `from` is less than `0` or if `to` is greater than the length of the array, then the method terminates immediately.

The above program has a `fill` method that fills the array passed to it with `value`. It does not fill the entire array, only the part specified by the indices `from` and `to`.

At the beginning of the `fill` method, the passed values are checked to ensure that they are valid. If `from` is less than 0, or if `to` is greater than the length of the array, then the `fill` method terminates immediately (executes a `return` statement).

This `return` statement is useful: practically every method in Java has one, and here's why.

2. Methods with a result, `void`

Remember we once figured out that there are statements, and there are expressions. An expression, unlike a statement, has a value that can be used somewhere.

And, in Java, methods can have a value. And this is very good news: methods are not only able to do something based on the input parameters, but also, for example, to evaluate something and **return the result of the calculation**.

By the way, you have already encountered such methods:

`double delta = Math.abs(d1 - d2);`	The `abs()` method returns a double

`Scanner console = new Scanner(System.in);`
`int x = console.nextInt();`
	The `nextInt()` method returns an `int`

`String str = "Hello";`
`String s2 = str.toUpperCase();`
	The `toUpperCase()` method returns a `String`

`int[] data = {1, 4, 5, 6, 7, 8, 11};`
`int[] array = Arrays.copyOf(data, 4);`
	The `copyOf()` method returns an `int[]`

Each method can only return one value of one predetermined type. The return type is determined when the method is declared:

```
public static Type name(parameters)
{
  method body
}
```

Where `name` is the name of the method, `parameters` is the list of method parameters, and `type` is the type of the result that the method returns.

For methods that return nothing, there is a special placeholder type: `void`.

Are you writing your own method and don't want to return anything to the calling method? Just declare the method's type as `void`, and the problem is solved. There are also lots of methods like this in Java.

---

3. Returning a result

We just figured out how to declare a method that returns the result of a calculation, but how do we result this result in the method itself?

The `return` statement helps us out here once again. Passing a result from a method looks like this:

```
return value;
```

Where `return` is a statement that terminates the method immediately. And `value` is the value that the method returns to the calling method when it exits. The type of `value` must match the `type` specified in the method declaration.

Example 1. The method calculates the minimum of two numbers:

```
int min(int a, int b)
{
   if (a < b)
      return a;
   else
      return b;
}
```

The method returns the minimum of two numbers.

If `a < b`
return `a`
Otherwise
return `b`

Example 2. The method duplicates the string passed to it `n` times:

`String multiple(String str, int times)` `{`	The method takes two parameters — a string and the number of times that the string should be repeated.
`String result = "";`	An empty string is created for the future result.
`for (int i = 0; i < times; i++)` `result = result + " "+ str;`	In a loop with `times` iterations, a space and the `str` string is added to the `result` string.
`return result;` `}`	The string `result` is returned as the result of the method.

Example 3: The method calculates the maximum of two numbers using the ternary operator:

`int max(int a, int b)` `{`	The method returns the maximum of two numbers.
`return (a > b ? a : b);` `}`	return (if `a > b`, then `a`, otherwise `b`)