Loops

When writing a program, we often need to perform a certain operation multiple times, e.g. display a **Java** string three times or perform an operation until a condition is met, e.g. to withdraw coins from the wallet until we collect the required amount. To repeat the operations in Java, we can use a loop. There are several of them.

The for loop

The `for` loop is used to iterate (that is, to repeat statements or a group of statements).

Structure of the `for` loop

```
for (<initialization>; <logic-condition>; <step>) {
    // operations
}
```

Example of a loop that displays text **Java** four times

```
for (int i=1; i<=4; i++) {
    System.out.println("Java");
}
```

The result of running the program

```
Java
Java
Java
Java
```

The loop is executed as long as the `logical condition` is met; in this case, until `i` is less than or equal to 4. The `initialization` section is performed only once: at the beginning, before the loop is executed. The `step` section is

performed at the end of each turn of the loop; after each `operations` section.

To better illustrate this, I will write the previous code in a different form.

An example of writing the same loop in other way

```
int i=1;
for ( ; i<=4; ) {
    System.out.println("Java");
    i++
}
```

- the assignment (initialization) of the value to the variable takes place at the beginning, before the loop is started
- after each operation (in this case, after the text is displayed), the step is performed

> ## Tip
>
> For loops, it is accepted to use short variable names (e.g. `i`, `j`, `k` etc.) as *loop counters*. In other cases, it is recommended to use more descriptive variable names.

The `for` loops are used primarily when we know how many times the operations are to be performed. If necessary, we can also save the `for` loop so that it can be repeated endlessly.

Example of an infinite loop

```
for ( ; ; ) {
    System.out.println("Java");
}
```

Now, the word **Java** will be displayed infinitely on the screen. To end the operation of this program, press the red square on the right of ().

The `for` loop may not be executed even once if the first logical condition is false!

```
for (int i=5; i<3; i++) {
    System.out.println(i);
}
```

The advantage of the `for` loop is that during iteration we have access to the current value of the variable (the so-called *loop counter*), in our case it is variable `i`. For example, we can quickly enter all values from `1` to `10`.

Example of a loop that writes numbers from 1 to 10

```
for (int i=1; i<=10; i++) {
    System.out.println(i);
}
```

The result displayed on the screen will be as follows:

```
1
2
3
4
5
6
7
8
9
10
```

We can use the `print()` function (instead of `println()`) to display the values in one line.

Example of using the `print()` function

```
for (int i=1; i<=10; i++) {
    System.out.print(i + " ");
}
```

We will then see on the screen:

```
1 2 3 4 5 6 7 8 9 10
```

These values can also be displayed from the end, that is, from `10` to `1`.

Example of a loop with decrement

```
for (int i=10; i>=1; i--) {
    System.out.print(i + " ");
}
```

```
}
```

Result:

```
10 9 8 7 6 5 4 3 2 1
```

You can also perform some calculations for the `i` variable. For instance, you can display individual numbers and their squares.

Example of using a loop counter for additional calculations

```
for (int i=1; i<=10; i++) {
    System.out.println("i=" + i + "i^2=" + (i*i));
}
```

The result will be as follows:

```
i=1, i^2=1
i=2, i^2=4
i=3, i^2=9
i=4, i^2=16
i=5, i^2=25
i=6, i^2=36
i=7, i^2=49
i=8, i^2=64
i=9, i^2=81
i=10, i^2=100
```

With the loop, you can easily sum all numbers from `1` to `10`.

Example of the operation of adding numbers in a loop

```
int sum = 0;
for (int i=1; i<=10; i++) {
    sum += i;
}
System.out.println("Suma = " + sum);
```

Result:

```
Sum = 55
```

The use of conditional statements with loops opens up further possibilities. For example, you can add only even numbers from `1` to `10`.

Example of the operation of adding numbers in a loop using a conditional statement

```
int sum = 0;
for (int i=1; i<=10; i++) {
    if (i%2 == 0) {
        sum += i;
    }
}
System.out.println("Sum = " + sum);
```

- you can define any restrictions as needed by changing this condition

Result:

```
Sum = 30
```

A step does not need to change by `1`. You can, for example, write all multiples of digit `5` in the range from `5` to `100`.

Example of a loop with a step incremented by `5`

```
for (int i=5; i<=100; i=i+5) {
    System.out.print(i + " ");
}
```

The result will be as follows:

```
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
100
```

The `break` and `continue` commands

For loops, in addition to conditional statements, we can use statements that change the control flow.

To stop the iteration at a given moment, you can use the `break` statement that definitively terminates the loops. This can be useful, for example, in a situation where you sum up values and you want to stop the iteration after a certain value is exceed. You can additionally display the number of the summed values and

is exceed. You can additionally display the number of the summed values and the value itself.

Example of a loop using the `break` statement

```
int sum=0;
int count=0;
for (int i=1; i<=200; i++) {
    sum += i;
    count++;
    if (sum >= 50) {
        break;
    }
}
System.out.println("Sum = " + sum);
System.out.println("Number of summed values = " +
count);
```

- summing up the values
- increment of the number of summed values
- if a sum greater than or equal to `50` has been achieved, the entire loop is stopped

If, on the other hand, you would like to ignore several turns of the loop for the set values, then you can use the `continue` statement. Let us assume you want to display numbers from `1` to `20`, but without the values from `13` to `15`. You can achieve this in the following way:

Example of a loop using the `continue` statement

```
for (int i=1; i<=20; i++) {
    if (i>=13 && i<=15) {
        continue;
    }
    System.out.print(i + " ");
}
```

- if the i value is between 13 and 15
- it interrupts the **current** turn (iteration) of the loop and goes to the next one

The (enhanced) `for` loop

In Java, there is one more kind of `for` loop, sometimes also called an "improved" or "enhanced" loop, or a `for each` loop. It allows you to browse (iterate) arrays and collections without using the index (loop counter).

Structure of the `for each` loop

Structure of the **`for each`** loop

```
for (<variable> : <collection/array>) {
    // oparations
}
```

During each loop rotation, consecutive value from the *collection* or *array* is passed to the *variable* and you can access it in the individual iteration.

Example of iterating over a 10-element array

```
int[] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
for (int i : digits) {
    System.out.print(i + " ");
}
```

This loop type is very convenient when you want to view the entire collection or array and you do not care about controlling how to do it, i.e. on defining the scope and order of the elements to which you want to have access.

When you use the `for each` loop but you would like to have access to a counter, you need to program it by yourself.

Example of the `for each` loop with its own counter

```
int[] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int oddCount = 0;
int allCount = 0;
for (int i : digits) {
    allCount++;
    if (i%2 != 0) {
        oddCount++;
        System.out.println(i + " is odd");
    }
}
System.out.println("Count: all=" + allCount + ", odd="
+ oddCount);
```

The result will be as follows:

```
1 is odd
3 is odd
5 is odd
7 is odd
9 is odd
Count: all=10, odd=5
```

The while loop

The `while` loop allows you to execute specific statements for as long as the *logical condition* for the loop is met (its value is `true`).

Structure of the `while` loop

```
while (<logical-condition>) {
    // operations
}
```

Example that lists numbers from `1` to `10`

```
int max=10;
int i=1;
while (i<=max) {
    System.out.print(i + " ");
    i++;
}
```

The `while` loop statements can be never executed if the *logical condition* for the loop is false at the beginning.

The `while` loop is used most often when the number of repetitions is not known in advance, but you know the condition that must be met for the loop.

As for the `for` loop, the `while` loop also can be performed endlessly.

Example of an infinite `while` loop

```
while (true) {
    System.out.println("Java");
}
```

You can also apply conditional statements such as `break` and `continue`.

The `do while` loop

The `do while` loop (similarly as the `while` loop) allows you to execute specific instructions as long as the *logical condition* for the loop is met (its value is `true`).

Structure of the `do while` loop

```
do {
    // operations
} while (logical-condition>);
```

Unlike the `while` loop, the do while loop **will always be executed at least once**, even if the **logical condition** is not met at the beginning.

Example of the **`do while`** loop

```
do {
    System.out.println("Java is great!!!");
} while (false);
```

The result will be as follows:

```
Java is great!!!
```

As in the case of the `for` and `while` loops, the `do while` loop also can be performed endlessly, and you can apply conditional statements such as `break` and `continue`.

Nested loops

It is possible to enclose a loop in another loop's body.

Example of nested loops

```
for (int i=1; i<=10; i++) {
    for (int j=1; j<=20; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- **outer loop** – it loops `10` times; it is responsible for generating individual lines
- **internal loop** – it loops `20` times; it is responsible for generating individual characters in the line
- the `print()` method prints the `*` character as many times as the inner loop is executed
- in this place, after the execution of all rotations of the inner loop, a new line mark is printed for one rotation of the outer loop

Result:

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

Important

Analyze the operation of these two nested loops carefully to understand the code above. Change the values of the `i` and `j` variables. Replace the `print()` function with `println()` (and opposite) and see how the operation of the program will change. Add conditional statements or the `break` and `continue` statements.

Thanks to the nested loops you can, for example, draw a "checkerboard".

```
for (int i=1; i<=10; i++) {
    for (int j=1; j<=10; j++) {
        if ((i % 2 == 0 && j % 2 == 1) || (i % 2 == 1
&& j % 2 == 0)) {
            System.out.print("#");
        } else  {
            System.out.print(" ");
        }
    }
    System.out.println();
}
```

- the # sign is to be printed alternately; if you are in an even row and an odd column or in an odd row and an even column. In other cases, a space is to be printed.

The result will be as follows:

```
 # # # # #
# # # # #
 # # # # #
# # # # #
 # # # # #
# # # # #
 # # # # #
# # # # #
 # # # # #
# # # # #
```

> **Warning**
>
> In programming, you will often encounter situations where the loop counter starts from `0`. In IT, the `0` value is encountered much more often than in mathematics 🙂

Summary

What have you learned in this chapter?

What loops are there in Java?

There are several types of loops available:

- `for`
- (enhanced) `for`
- `while`
- `do while`

When should you use the `for` loop and when the `while` / `do while` loop?

We use the `for` loop most often when we know the number of elements or the number of repetitions of a given statement. In addition, we use it when we want to have an (easy) access to the loop counter. We use the `while` / `do while` loop when we do not know in advance how many times we will iterate since it depends on some factor.

What is the difference between the `for` loop and the (enhanced) `for` loop?

The `for` loops allows you to declare the iteration method by controlling the loop count. If we want only to go through a given set (without probing into the iteration method), we choose the `for` (enhanced loop that is also called the `foreach` loop).

What is the difference between the `while` loop and `do while` loop?

The `do while` loop will always be executed at least once, even if the logical condition is false at the beginning; the `while` loop will not be executed in this case.

Exercises

- List numbers from `-20` to `20` using a loop. Then write out:
  - first 6 numbers
  - last 6 numbers
  - all even numbers
  - all numbers except for number 5
  - all numbers up to and including 7
  - all numbers divisible by 3
  - sum of all numbers
  - sum of all numbers greater or equal 4
  - all numbers and their powers
  - all numbers and their value *modulo* `10`
- Using a class that offers operations on date and time, get the current time (hour, minute, and second) and write these values using the characters ∗, whose number equals the given value. To make it difficult, one line can contain up to 10 ∗ characters.Example for values `15:03:28`.`Current time: 15:03:28 Hour: ********** ***** Minute: *** Second: ********** ********** ********`
- A calendar page for January with January 1 being Wednesday, for the current date January 16, it might look like this:

```
Mo Tu We Th Fr Sa Su
--------------------
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15[16]17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

- display the above page using loops and additional instructions
- customize the code so January 1 can be any day of the week
- download data (using a class that offers operations on date and time) about the current date and display a page for it from the calendar

Complete Lesson