# Control structures

## Iterative structures

Iterative structures, also known as *loops*, can repeat a set of instructions a given number of times, or as long as a condition is true. This way, we avoid writing this set of instructions more than once if we want to repeat it. Now, we are going to see the main iterative structures provided by Java, although we will learn some more in later documents.

### 1. The "while" clause

This clause is used to repeat a given set of instructions while a given condition (or set of conditions) is true. For instance, this example counts from 1 to 10:

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

As soon as the condition inside the *while* clause is false, the program exits the *loop* and runs next instruction beyond this loop. Take into account that this instruction can be simple or complex:

```
int n = 1;
while (n >= 1 && n <= 10)
{
    ...
}
```

### 2. The "do..while" clause

This clause si similar to the previous one, but the condition is evaluated at the end of the loop, instead of the beginning. If we do the same loop than in previous example with a `do..while` structure, it would look like this.

```
int n = 1;
do
{
    System.out.println(n);
    n++;
} while (n <= 10);
```

We will use this loop when we don't know how many iterations are expected, but we know that there will be (at least) one iteration. It is very usual when we ask the user to type something and then check the input and ask the user again. On the contrary, *while* loop is better when we don't even know if there will be one iteration.

## 3. The "for" clause

We will use this loop when we know how many iterations are expected. It has 3 parts on it:

- The declaration of a counter
- The condition to repeat the bucle (similar to *while* or *do..while* condition)
- The increment or decrement for the counter (generally to reach the end of the loop when the condition turns into *false*)

The counter from 1 to 10 should be done with this structure preferably, and it would look like this:

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

Note that we can declare variables in `for` loops (and in the middle of other code, as in other languages such as C#).

### 3.1. Another "for"

There is another way of using the `for` clause, applied to collections or arrays. It consists in using a variable with the same type, this way:

```
for (int number: numbers)
    System.out.println("" + number);
```

where `numbers` is expected to be a collection or array of integers. This structure is equivalent to the `foreach` structure of other languages such as C#, and is expected to be used in a read-only way (only to check the values, but not to modify them).

**Exercise 1**:

Create a program called **GroupPeople** that asks the user to enter how many people is going to attend to a conference. The program must create groups of (preferably) 50 people. Whenever this is not possible, then it will attempt to create groups of 10 people, and finally it will create groups of 1 person. The program must output how many groups of each category are necessary. For instance, if 78 people are going to attend to the conference, then we need 1 group of 50 people, 2 groups of 10 people and 8 groups of 1 people.

**Exercise 2**:

Create a program called **SumDigits** that asks the user to enter numbers (integer values) until he enters 0. The program must sum up all the numbers entered by the user and then show the final result, and how many digits it has. For instance, if the user types 12, 20, 60, 33, 99 and 0, then the program must output: "*The result is 224, and it has 3 digits*".