for loop

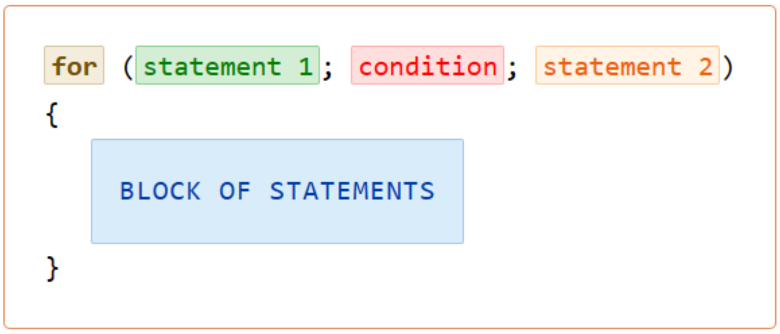
1. Comparing loops: `for` vs `while`

A `while` loop can be used anywhere a statement or group of statements needs to be performed several times. But among all the possible scenarios, one is worth highlighting.

We're talking about the situation when the programmer (the program's creator) knows in advance how many times the loop should be executed. This is usually handled by declaring a special counter variable, and then increasing (or decreasing) the variable by `1` with each iteration of the loop.

Everything seems to work as it should, but it isn't very convenient. Before the loop, we set the counter variable's initial value. Then in the condition we check whether it has already reached the final value. But we usually change the value at the very end of the loop body.

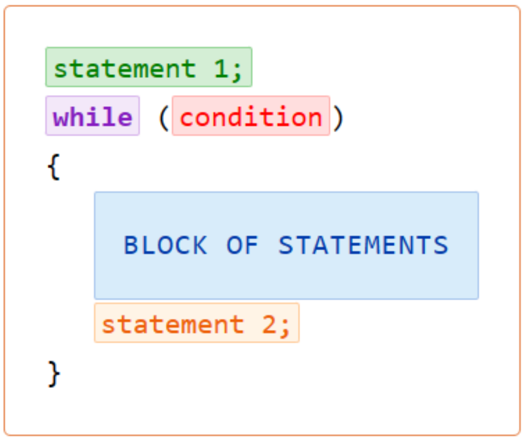
And what if the body of the loop is large? Or if we have several nested loops? In general, in these cases it's desirable to collect all this information about counter variables in one place. And that's why we have the `for` loop in Java. It also doesn't look very complicated:



```
for (statement 1; condition; statement 2)
{
    BLOCK OF STATEMENTS
}
```

A `while` loop has just a condition in parentheses, but a `for` loop adds two statements, separated by semicolons.

The reality is simpler than it sounds: the compiler converts a `for` loop into an ordinary `while` loop like this:



```
statement 1;
while (condition)
{
    BLOCK OF STATEMENTS
    statement 2;
}
```

Or better yet, let's demonstrate this with an example. The two code snippets below are identical.

Option 1	Option 2

`for (int i = 0; i < 20; i++)`
`{`
`   System.out.println(i);`
`}`
	`int i = 0;`
	`while (i < 20)`
	`{`
	`   System.out.println(i);`
	`   i++;`
	`}`

We just gathered into one place all the code that pertains to the `i` counter variable.

In a `for` loop, statement 1 is executed just once, before the loop itself begins. This can be seen clearly in the second code snippet

statement 2 is executed the same number of times as the body of the loop, and each time it is executed after the entire body of the loop has been executed

---

2. Where the `for` loop is used

The `for` loop is probably the most used type of loop in Java. It is used everywhere, for programmers it is just clearer and more convenient than a `while` loop. Virtually any `while` loop can be converted into a `for` loop.

Examples:

while loop	for loop

`int i = 3;`
`while (i >= 0)`
`{`
`   System.out.println(i);`
`   i--;`
`}`
	`for (int i = 3; i >= 0; i--)`
	`{`
	`   System.out.println(i);`
	`}`

`int i = 0;`
`while (i < 3)`
`{`
`   System.out.println(i);`
`   i++;`
`}`
	`for (int i = 0; i < 3; i++)`
	`{`
	`   System.out.println(i);`
	`}`

`boolean isExit = false;`
`while (!isExit)`
`{`
`   String s = console.nextLine();`
`   isExit = s.equals("exit");`
`}`
	`for (boolean isExit = false; !isExit; )`
	`{`
	`   String s = console.nextLine();`
	`   isExit = s.equals("exit");`
	`}`

`while (true)`
`   System.out.println("C");`
	`for (; true; )`
	`   System.out.println("C");`

`while (true)`
`{`
`   String s = console.nextLine();`
`   if (s.equals("exit"))`
`      break;`
`}`
	`for (; true; )`
	`{`
	`   String s = console.nextLine();`
	`   if (s.equals("exit"))`
	`      break;`
	`}`

Pay attention to the last example. The statements for working with the loop counter are missing. There is no counter and no statement.

In a `for` loop, Java lets you omit the "statement to initialize the counter" and the "statement to update the counter". Even the expression that defines the loop condition can be omitted.

3. Nuances of using the `for` loop

An important point about using `for` loops and `break` and `continue` statements.

A `break` statement in a `for` loop works the same as in a `while` loop — it terminates the loop immediately. A `continue` statement skips the loop body, but not `statement 2` (which changes the loop counter).

Let's take another look at how `for` and `while` loops are related.

```
for (statement 1; condition; statement 2)
{
   block of statements
}
```

```
statement 1;
while (condition)
{
   block of statements
   statement 2;
}
```

If a `continue` statement is executed in a `for` loop, then the rest of the block of statements is skipped, but statement 2 (the one that works with the `for` loop's counter variable) is still executed.

Let's return to our example with skipping numbers that are divisible by 7.

This code will loop forever	This code will work fine
`int i = 1;`	`for (int i = 1; i <= 20; i++)`
`while (i <= 20)`	`{`
`{`	`   if ( (i % 7) == 0) continue;`
`   if ( (i % 7) == 0) continue;`	`   System.out.println(i);`
`   System.out.println(i);`	`}`
`   i++;`	
`}`	

The code that uses the `while` loop will not work — i will never be greater than 7. But the code with the `for` loop will work fine.

4. Comparing for loops: Java vs Pascal

By the way, Pascal also has a `For` loop. In general, essentially every programming language has one. But in Pascal is it super clear. Examples:

Pascal	Java

`For i := 1 to 10 do`
`Begin`
`   Writeln(i);`
`End;`
	`for (int i = 1; i <= 10; i++)`
	`{`
	`   System.out.println(i);`
	`}`

`For i := 1 to 10 do step 2`
`Begin`
`   Writeln(i);`
`End;`
	`for (int i = 1; i <= 10; i = i + 2)`
	`{`
	`   System.out.println(i);`
	`}`

`For i := 10 downto 0 do step 2`
`Begin`
`   Writeln(i);`
`End;`
	`for (int i = 10; i >= 0; i = i - 2)`
	`{`
	`   System.out.println(i);`
	`}`