Loops in Java

1. Loops in our life

Very often our lives require us to perform the same actions many times. For example, suppose I need to scan a document consisting of lots of pages. That we repeat the same procedure over and over again:

- Put the first page on the scanner
- Press the scan button
- Put the next page on the scanner

This is difficult to do manually. It would be nice if this process could be somehow be automated.

Or consider another example: let's say I want to mark all unread emails in my inbox as spam. Once upon a time I would have to select each email one at a time and mark it as spam.

But programmers are lazy, so they automated this process long ago: now you simply select any list of letters and click "mark as spam", and then your email client runs through the list and moves each email to the spam folder.

What can we say here? It's super convenient when a computer or program can execute hundreds or thousands of monotonous operations with one click. And now you will learn how to do this too.

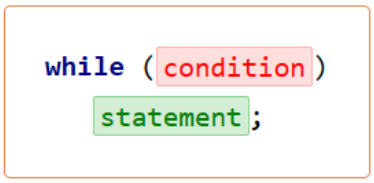
---

2. `while` loop

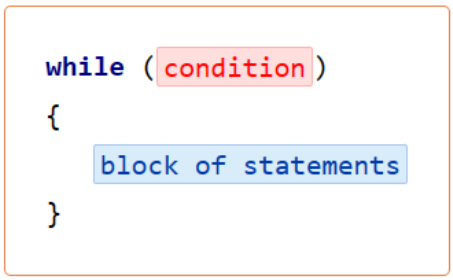
The if-else statement significantly expanded our programming capabilities, making it possible to write programs that perform different actions in different situations. But there is one more thing that will make our programs an order of magnitude more powerful — **loops**.

Java has 4 kinds of loops: **`while`**, **`for`**, **`for-each`** and **`do-while`**. We will now dig into the very first of these.

A **`while`** loop is very simple. It consists of only two parts: a *condition* and a *loop body*. The loop body is executed over and over again as long as the condition is `true`. In general, a `while` loop looks like this:



Notation for a `while` loop with a single statement



Notation for a `while` loop with a block of statements

It's very simple. The *statement* or *block of statements* is executed over and over again as long as the *loop condition* equals `true`.

This is how it works: first, the *condition* is checked. If it is true, then the *loop body* is executed (the *statement* or *block of statements*). Then the *condition* is checked again and the *loop body* is executed again. And so on until the *condition* becomes false.

If the condition is **always true**, then the program will never stop running. It will be permanently stuck in the loop.

If the condition is **false the very first time it is checked**, then the body of the loop will not be executed even once.

---

3. Examples of loops

Here are some practical examples of loops in action.

Code	Explanation

`int n = 5;`
`while (n > 0)`
`{`
`  System.out.println(n);`
`  n--;`
`}`
	5 lines will be displayed on the screen:
	5
	4
	3
	2
	1

Code	Explanation

`int  n = 0;`
`while (n < 10)`
`{`
`  System.out.println(n);`
`  n++;`
`}`
	10 lines will be displayed on the screen:
	0
	1
	...
	8
	9

Code	Explanation

`Scanner console = new Scanner(System.in);`
`while(console.hasNextInt())`
`{`
`  int x = console.nextInt();`
`}`
	The program reads numbers from the keyboard as long as numbers are entered.

Code	Explanation
`while (true)` `  System.out.println("C");`	The program will endlessly print the letter `C` on the screen.

In the previous example, the `equals()` method is used to compare strings. If the strings are equal, the function will return `true`. If the strings are not equal, then it will return `false`.

4. Loop within a loop

As you learned about conditional statements, you saw that you can use them to implement complex logic by combining multiple conditional statements. In other words, by using an `if` statement inside an `if` statement.

You can do the same thing with loops. To write a loop within a loop, you need to write the second loop inside the body of the first loop. It will look something like this:

```
while (condition for outer loop)
{
   while (condition for inner loop)
   {
     block of statements
   }
}
```

`while` loop (with a `block of statements`) inside another `while` loop

Let's look at three tasks.

Task 1. Let's say we want to write a program that displays the word `Mom` on the screen 4 times. A loop is exactly what we need. And our code will look something like this:

Code	Explanation

`int n = 0;`
`while (n < 4)`
`{`
`  System.out.println("Mom");`
`  n++;`
`}`
	4 lines will be displayed on the screen:
	Mom
	Mom
	Mom
	Mom

Task 2. We want to write a program that displays 5 letter `A`s on a single line. To do this, we need a loop once again. This is what the code will look like:

Code	Explanation

`int n = 0;`
`while (n < 5)`
`{`
`  System.out.print("A");`
`  n++;`
`}`
	Instead of `println()`, we will use `print()`. Otherwise, each letter `A` would end up on a separate line.
	
	The screen output will be:
	AAAAA

Task 3. We want to display a rectangle comprised of letter `A`s. The rectangle should consist of 4 rows by 5 columns. To accomplish this, we now need a nested loop. We'll simply take our first example (the one where we output 4 lines) and replace the code for outputting one line with the code from the second example.

Code	Explanation

`int n = 0;`
 	
`while (n < 4)`
`{`
 	
`  int m = 0;`
 	
`  while (m < 5)`
`  {`
`  System.out.print("A");`
`  m++;`
`  }`
 	
`  System.out.println();`
 	
`  n++;`
 	
`}`
	The *outer loop* is purple. It uses the `n` variable to count the number of iterations of the loop.
	
	The *inner loop* is green. It uses the `m` variable to count the number of loop iterations.
	
	We have to explicitly *move the cursor* to the next line after the inner loop is complete. Otherwise, all the letters that the program prints will end up on one line.
	
	The screen output will be:
	AAAAA
	AAAAA
	AAAAA
	AAAAA

The outer and inner loops must use different variables to count the number of loop iterations. We also had to add the `System.out.println()` command after the inner loop, since that loop displays letter `A`s on the same line. Once the letters on a line are displayed, someone has to move the cursor to a new line.

5. Comparing loops Java vs Pascal

Many of you studied Pascal in high school. To make it easier for you to understand the material here, take a look at this comparison of `while` loops written in Pascal and Java. If you don't know Pascal, then just skip this part.

Pascal	Java

`i := 3;`
`While i >= 0 Do`
`  Begin`
`  WriteLn(i);`
`  i := i - 1;`
`  End;`
	`int i = 3;`
	`while (i >= 0)`
	`{`
	`  System.out.println(i);`
	`  i--;`
	`}`

`i := 0;`
`While i < 3 Do`
`  Begin`
`  WriteLn(i);`
`  i := i + 1;`
`  End;`
	`int i = 0;`
	`while (i < 3)`
	`{`
	`  System.out.println(i);`
	`  i++;`
	`}`

`IsExit := False;`
`While Not isExit Do`
`  Begin`
`  ReadLn(s);`
`  isExit :=  (s = 'exit');`
`  End;`
	`boolean isExit = false;`
	`while (!isExit)`
	`{`
	`  String s = console.nextLine();`
	`  isExit = s.equals("exit");`
	`}`

`While True Do` `  WriteLn('C');`	`while (true)` `  System.out.println("C");`

`While True Do`
`  Begin`
`  ReadLn(s);`
`  If s = 'exit' Then`
`  Break;`
`  End;`
	`while (true)`
	`{`
	`  String s = console.nextLine();`
	`  if (s.equals("exit"))`
	`  break;`
	`}`