Control structures: while and for

In a recipe, sometimes you find repetition instructions, for example: "add salt and pepper until seasoned." When reading this, a cook understands that he/she must start adding a little salt and pepper, taste the result, then if the result is not yet satisfactory, add a little more and taste again, and so on.

The corresponding construct in imperative language is the *iteration statement*. There are actually several of them.

Simple "while loop"

The simplest iteration statement is defined as follows:

Iteration statement, variant 1 (also called "while loop"):

Syntax:

`while (` <expression> `)` <block-or-statement>

(the keyword "`while`" followed by an expression between parentheses, followed by either a statement or block between "`{`" and "`}`". The statement-or-block part is also called the *body* of the iteration statement.)

Semantics:

The expression between parentheses, also called *conditional expression*, is evaluated first. If its value is equivalent to `true`, the body is executed. Then the conditional expression it is evaluated again. If it is still true, the body is executed again. The construct continues to repeat the test and execution until the condition becomes false.

For example:

```
n = 1;
while (n <= 100)
{
   System.out.println(n);
   n = n + 1;
}
System.out.println("bye " + n);
```

This program fragment prints the numbers from 1 to 100 on the standard output, followed by "bye 101".

Note that the condition is evaluated at the beginning. *If the condition is false when the while loop is first reached, the body is not executed at all.* For example:

```
n = 10;
while (n < 5)
{
   System.out.println(n);
   n = n + 1;
}
System.out.println("bye " + n);
```

This program fragment only prints "bye 10", because n is greater than 5 when the while loop is reached during execution.

Condition at the end, the "do-while loop"

The second iteration statement is defined as follows:

Iteration statement, variant 2 (also called "do-while loop"):

Syntax:

> `do` <block-or-statement> while `(` <expression> `)` `;`
> (the keyword "`do`", followed by either a statement or block between "`{`" and "`}`", followed by the keyword "`while`", followed by an expression between parentheses, followed by a semicolon. The statement-or-block part is also called the *body* of the iteration statement.)

Semantics:

> The body is executed first. The expression between parentheses, also called *conditional expression*, is then evaluated. If its value is equivalent to `true`, the body is executed again. The construct continues to repeat the execution and test until the condition becomes false.

For example:

```
n = 1;
do
{
   System.out.println(n);
   n = n + 1;
}
while (n <= 100);
System.out.println("bye " + n);
```

This program fragment prints the numbers from 1 to 100 on the standard output, followed by "bye 101".

Note that the body is always executed once. For example:

```
n = 10;
do
{
   System.out.println(n);
   n = n + 1;
}
while (n < 5);
System.out.println("bye " + n);
```

This program fragment prints 10, then "bye 11".

When to choose “do-while” over “while”?

When writing your own code, if you hesitate between “while” and “do-while” in some situation, you can apply the following technique:

Start writing the code using “while”; then if you recognize the following pattern:

```
<action X> ;
while ( <condition> ) {
    <action X> ;
}
```

In other word, you see a part of your program is duplicated just before the `while` and within its body; then you can rewrite the code to use “do-while” instead:

```
do {
    <action X> ;
} while ( <condition> );
```

In practice, “do-while” is used much less frequently than just “while”. Many programming languages (like Python) even do not offer it at all.

Compact iteration: the “for loop”

There is a pattern that often occurs in programs:

```
// Example 1:
i = 0;
while (i <= 100)
{
   System.out.println(i);
   i = i + 1;
}

// Example 2:
i = 100;
while (i > 0)
{
   System.out.println(i);
   i = i - 1;
}

// Example 3:
i = 1;
while (i < 500)
{
   System.out.println(i);
   i = i * 2;
}
```

The common pattern in these examples is “an iteration whose condition depends a counter variable initialized once before the loop starts and updated in the loop body”. We can abstract it as follows:

<initialization>

`while (` <condition> `) {`

<something...>
<update-counter> `;`

`}`

This pattern is so often used that the designers of the C language have created a shorthand construct, that is also available in C++ and Java. It is defined as follows:

Iteration statement, variant 3 (also called "for loop"):

Syntax:

`for (` <initialization> `;` <condition> `;` <update-counter> `)`
<block-or-statement>

(the keyword "`for`", followed by an opening parenthesis, followed by a first expression or declaration called the *initializer*, followed by a semicolon "`;`", followed by a 2nd expression called the *condition*, followed by another semicolon "`;`", followed by a third expression called the *update expression*, followed by a closing parenthesis, followed by a statement, or block between "`{`" and "`}`". The statement-or-block part is also called the *body* of the iteration statement.)

Semantics:

Equivalent to:

```
{
    <initialization> ;
    while ( <condition> ) {
      <block-or-statement>
      <update> ;
    }
}
```

The initialization is executed first. The condition is then evaluated. If its value is equivalent to `true`, the body is executed, then the update expression. Then the condition is evaluated again; if it is still true, the body and update are executed again. The construct continues to repeat the test and execution until the condition becomes false.

For example:

```
for (i = 0; i <= 100; i = i + 1)
{
   System.out.println(i);
}

for (i = 100; i > 0; i = i - 1)
{
```

```
    System.out.println(i);
}

for (i = 1; i < 500; i = i * 2)
{
    System.out.println(i);
}
```

Like for the "while loop", the condition is evaluated at the beginning, so If the condition is false when the for loop is first reached, the body is not executed at all.

When to choose "for" over "while"?

When writing your own code, if you hesitate between "for" and "while" in some situation, you can apply the following technique:

- if you know how many iterations there will be before the loop starts, use "for";
- if you do not know how many iterations there will be, try writing your code using both "while" and "for", and choose the shortest of the two.

Examples

Check out the examples Alphabet, Alphabet2 and AlphabetZonderK in the accompanying source code repository.

Important concepts

- *iteration statement*, definition of the 3 variants
- the terms "while loop", "do-while loop" and "for loop"
- the parts of an iteration statement: the *condition expression* and the *body*;
- when to choose "do-while" over "while";
- when to choose "for" over "while".

Further reading

- Think Java, sections 7.2 (pp. 76-78), 12.4 (pp. 152-153)
- Programming in Java, sections 3.3, 3.3.1, 3.3.2 (pp. 82-86) and 3.4 (pp. 88-96)
- Absolute Java, section 3.3 with exercises 22-39 (pp. 132-147)

Copyright and licensing