Theory: The while and do-while loops

30 minutes 0 / 5 problems solved

Skip this topic Start practicing

There are a number of approaches to repeat a fragment of code while a certain condition is `true`. In this lesson, we will learn how to do it using two kinds of loops. The difference between them is the order of the repetitive fragment execution and condition evaluation.

The while loop

The **while** loop consists of a block of code and a condition (a Boolean expression). If the condition is `true`, the code within the block is executed. This code repeats until the condition becomes `false`. Because this loop checks the condition before the block is executed, the control structure is often also known as a **pre-test loop**. You can think of the **while** loop as a repeating conditional statement.

The basic syntax of the **while** loop is the following:

```
while (condition) {
    // body: do something repetitive
}
```

A loop's body can contain any correct Java statements including conditional statements and even other loops (nested loops).

It is also possible to write an **infinite loop** if the condition is invariably `true`

```
while (true) {
    // body: do something indefinitely
}
```

The use of infinite loops will be considered in the following topics.

Example 1. The following loop prints integer numbers while a variable is less than 5.

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
// a next statement
```

Let's explain how this loop works. First, the value 0 is assigned to the variable `i`. Before the first execution of the loop's body, the program checks if `i < 5`. That is true (because `i` is 0), so the body of the loop starts executing. The body has two statements: displaying the current value of `i` and incrementing it by 1. After this, the expression `i < 5` is evaluated again. Now `i` equals 1, so the conditional is `true` again, and the

loop's body is repeated again. This is repeated until `i` has taken the value 5, after which the expression `i < 5` ceases to be `true`, and the execution of this loop terminates. The program proceeds to the next statement after the loop.

The output:

```
0
1
2
3
4
```

Example 2. The following program displays English letters in a single line.

```
public class WhileDemo {

    public static void main(String[] args) {
        char letter = 'A';
        while (letter <= 'Z') {
            System.out.print(letter);
            letter++;
        }
    }
}
```

The program takes the first letter `'A'` and then repeats:

- if the letter is less or equal to `'Z'` the program go to the loop's body
- inside the body, it prints the current character and gets the next letter.

The program prints:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Remember that it is possible to get the next character (according to the Unicode table) using the increment operator.

The do-while loop

In the **do-while** loop, the body is executed first and the condition is tested afterwards. If the condition is `true`, statements within the block are executed again. This repeats until the condition becomes `false`. Because **do-while** loops check the condition after the block is executed, the control structure is often also known as a **post-test loop**. In contrast to the **while** loop, which tests the condition before the code within the block is executed, the **do-while** loop is an exit-condition loop. So, the code within the block is always executed at least once.

This loop contains three parts: the `do` keyword, a body, and `while(condition)`:

```
do {
    // body: do something
} while (condition);
```

The following program reads an integer number from the standard input and displays the number. If the number 0 is entered, the program prints it and then stops. It demonstrates the **do-while** loop.

```
public class DoWhileDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int value;
        do {
            value = scanner.nextInt();
            System.out.println(value);
        } while (value != 0);
    }
}
```

Input numbers:

```
1 2 4 0 3
```

The program prints:

```
1
2
4
0
```

Note that, like the **while** loop, the **do-while** loop can be infinite.

In practice, the **do-while** loop is used less than the **while** loop. A good example of using it is a program that reads data from the standard input until a user enters a certain number or string. It is assumed that the program will be executed at least once, and repeated execution is optional.

Reading a sequence with an unknown length

The `while` loop can be used to read a sequence of characters of an arbitrary length if it invokes `hasNext()` method of `Scanner` inside the condition. The method returns `true` if the next element exists and otherwise, `false` .

Here is code that calculates the sum of all elements from the given numbers:

```
Scanner scanner = new Scanner(System.in);

int sum = 0;
while (scanner.hasNext()) {
    int elem = scanner.nextInt();
    sum += elem;
}

System.out.println(sum);
```

If the input sequence is `1 2 3`, the code prints `6`, but if the input sequence is `5 18 9 23 4`, the code prints `59`.

As you see, the **while** loop can be used to do many interesting things in your programs.