11876 users solved this topic. Latest completion was **40 minutes ago**.

Theory: The while and do-while loops

37 minutes 5 / 13 problems solved

Start practicing

There are a number of approaches to repeat a fragment of the code while a certain condition is `true`. In this lesson, we will learn how to do it by using two kinds of loops. They differ in the order of the repeated fragment execution and condition evaluation.

§1. 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`. Since this loop checks the condition before the block is executed, the control structure is also known as a **pre-test loop**. You can think of the **while** loop as a repetitive 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, the latter being called nested loops.

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

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

The application 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++;
}
// 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 the condition `i < 5` is true. In our case, `i` is 0, so the condition is true and 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 is done, the expression `i < 5` is evaluated again. Now `i` equals 1, so the condition is still `true`, 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
```

Note, that the last value of `i`, that is 5, is not printed.

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

Current topic:

- ✓ The while and do-while loops — Stage 2

Topic depends on:

- ✓ Increment and decrement — Stage 1
- ✓ Ternary operator — Stage 2

Topic is required for:

- ✓ Branching statements — Stage 2

Table of contents:

```
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 goes on like this:

- 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 `letter` takes the next alphabet letter.

The program prints:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Remember that it is possible to get the next character according to the Unicode table by using the increment operator. After the code execution, the `letter` will equal `[`.

§2. The do-while loop

In the **do-while** loop, the body is executed first, while the condition is tested afterward. 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);
```

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. The following program reads integer numbers from the standard input and displays them. If the number 0 is entered, the program prints it and then stops. The following example 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 as well as the **while** loop, the **do-while** loop can be infinite.

In practice, the **do-while** loop is used less than the **while** loop. It is assumed that the program will be executed at least once, repeated execution is optional.

§3. Reading a sequence with an unknown length

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

Here is a code that calculates the sum of all elements from the provided sequence:

```
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`, if it is `5 18 9 23 4`, the code prints `59`.

As you see, the **while** loop can be used to solve different interesting tasks in your programs.

§4. Conclusion

There are different ways to perform some fragment of your code several times. In this topic, we've discussed two alternative ways to use loops that are based on the conditional statement evaluation. If you want to check the condition first, and based on the result perform the operations or ignore them at all, **while** loop is your choice. If you want to do one iteration of the loop in any case and then evaluate the condition for repetition, then choose **do-while**. Both types of loops can be used to read the sequence from the standard input: for **do-while**, you may use some stop-value to terminate the loop, for **while**, use the `hasNext()` to check that the input is over.

Report a typo

965 users liked this theory. **40** didn't like it. **What about you?**

Start practicing

Comments (38) Hints (6) Useful links (0) Show discussion