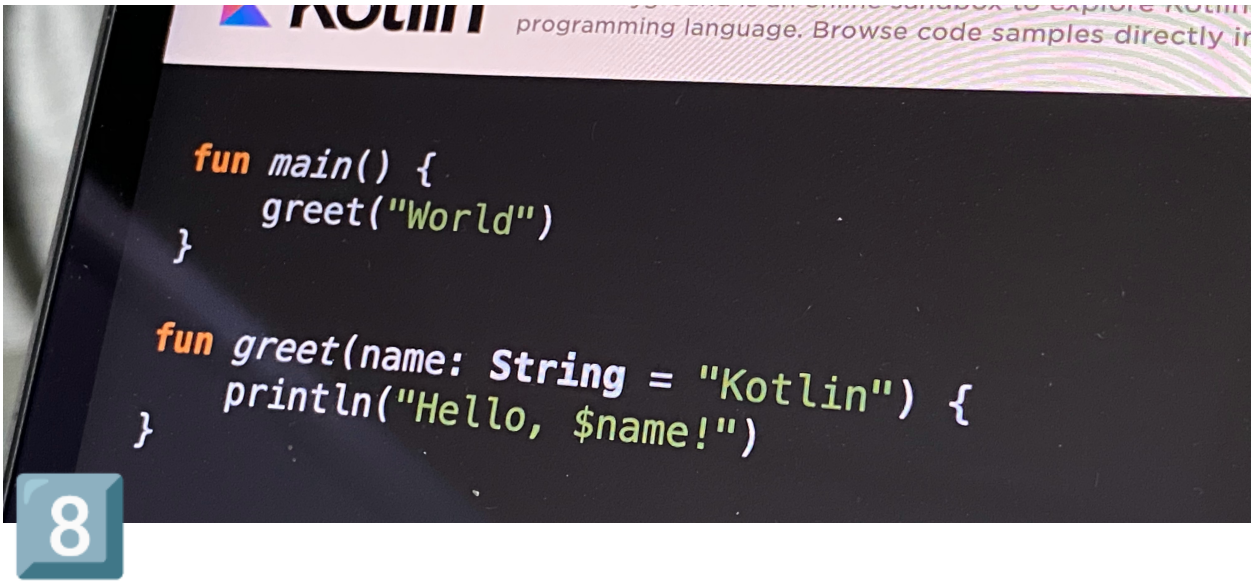


Class 8(Loop, Break, Continue)

Kotlin while and do...while Loop

Loop is used in programming to repeat a specific block of code until certain condition is met (test expression is `false)`.

Loops are what makes computers interesting machines. Imagine you need to print a sentence 50 times on your screen. Well, you can do it by using print statement 50 times (without using loops). How about you need to print a sentence one million times? You need to use loops.

You will learn about two loops `while` and `do..while` in this article with the help of examples.

If you are familiar with while and do...while loops in Java, you are already familiar with these loops in Kotlin as well.

Kotlin while Loop

The syntax of `while` loop is:

```
while (testExpression) {
    // codes inside body of while loop
```

```
}
```

How while loop works?

The test expression inside the parenthesis is a Boolean expression.

If the test expression is evaluated to `true`,

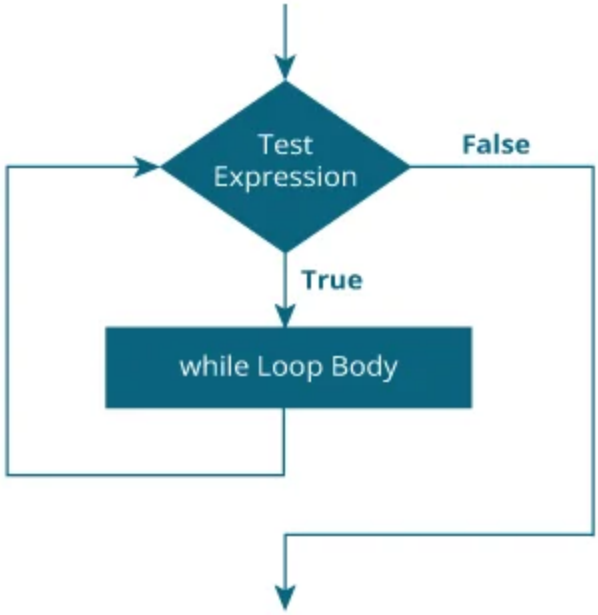
- statements inside the while loop are executed.
- then, the test expression is evaluated again.

This process goes on until the test expression is evaluated to `false`.

If the test expression is evaluated to `false`,

- while loop is terminated.

Flowchart of while Loop



Example: Kotlin while Loop

```
// Program to print line 5 times

fun main(args: Array<String>) {

    var i = 1

    while (i <= 5) {
        println("Line $i")
        ++i
    }
}
```

When you run the program, the output will be:

```
Line 1
Line 2
Line 3
Line 4
Line 5
```

Notice, `++i` statement inside the `while` loop. After 5 iterations, variable i will be incremented to 6. Then, the test expression `i <= 5` is evaluated to `false` and the loop terminates.

If the body of loop has only one statement, it's not necessary to use curly braces `{ }`.

Example: Compute sum of Natural Numbers

```
// Program to compute the sum of natural numbers from 1 to 100.
fun main(args: Array<String>) {

    var sum = 0
    var i = 100

    while (i != 0) {
        sum += i     // sum = sum + i;
        --i
    }
    println("sum = $sum")
}
```

When you run the program, the output will be:

```
sum = 5050
```

Here, the variable sum is initialized to 0 and i is initialized to 100. In each iteration of while loop, variable sum is assigned `sum + i`, and the value of i is decreased by 1 until i is equal to 0. For better visualization,

```
1st iteration: sum = 0+100 = 100, i = 99
2nd iteration: sum = 100+99 = 199, i = 98
3rd iteration: sum = 199+98 = 297, i = 97
... .. ...
... .. ...
99th iteration: sum = 5047+2 = 5049, i = 1
100th iteration: sum = 5049+1 = 5050, i = 0 (then loop terminates)
```

To learn more about test expression and how it is evaluated,
visit comparison and logical operators.

Kotlin do...while Loop

The `do...while` loop is similar to `while` loop with one key difference. The body of `do...while` loop is executed once before the test expression is checked.

Its syntax is:

```
do {
   // codes inside body of do while loop
} while (testExpression);
```

How do...while loop works?

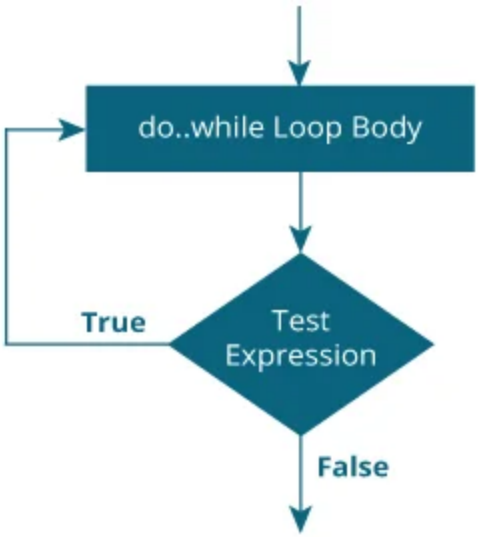
The codes inside the body of `do` construct is executed once (without checking the testExpression). Then, the test expression is checked.

If the test expression is evaluated to `true`, codes inside the body of the loop are executed, and test expression is evaluated again. This process goes on until the test

expression is evaluated to `false`.

When the test expression is evaluated to `false`, `do..while` loop terminates.

Flowchart of do...while Loop



Example: Kotlin do...while Loop

The program below calculates the sum of numbers entered by the user until user enters 0.

To take input from the user, `readline()` function is used. **Recommended Reading:** Kotlin Basic Input

```
fun main(args: Array<String>) {

    var sum: Int = 0
    var input: String

    do {
        print("Enter an integer: ")
        input = readLine()!!
        sum += input.toInt()

    } while (input != "0")
```

```
    println("sum = $sum")
}
```

When you run the program, the output will be something like:

```
Enter an integer: 4
Enter an integer: 3
Enter an integer: 2
Enter an integer: -6
Enter an integer: 0
sum = 3
```

Kotlin for Loop

There is no traditional for loop in Kotlin unlike Java and other languages.

In Kotlin, `for` loop is used to iterate through ranges, arrays, maps and so on (anything that provides an iterator).

The syntax of `for` loop in Kotlin is:

```
for (item in collection) {
    // body of loop
}
```

Example: Iterate Through a Range

```
fun main(args: Array<String>) {

    for (i in 1..5) {
        println(i)
    }
}
```

Here, the loop iterates through the range and prints individual item.

Output

```
1
2
3
4
5
```

If the body of the loop contains only one statement (like above example), it's not necessary to use curly braces `{ }`.

```
fun main(args: Array<String>) {
    for (i in 1..5) println(i)
}
```

It's possible to iterate through a range using `for` loop because ranges provides an iterator. To learn more, visit *Kotlin iterators*.

Example: Different Ways to Iterate Through a Range

```
fun main(args: Array<String>) {

    print("for (i in 1..5) print(i) = ")
    for (i in 1..5) print(i)

    println()

    print("for (i in 5..1) print(i) = ")
    for (i in 5..1) print(i)             // prints nothing

    println()

    print("for (i in 5 downTo 1) print(i) = ")
    for (i in 5 downTo 1) print(i)

    println()

    print("for (i in 1..5 step 2) print(i) = ")
    for (i in 1..5 step 2) print(i)

    println()

    print("for (i in 5 downTo 1 step 2) print(i) = ")
```

```
    for (i in 5 downTo 1 step 2) print(i)
}
```

Output

```
for (i in 1..5) print(i) = 12345
for (i in 5..1) print(i) =
for (i in 5 downTo 1) print(i) = 54321
for (i in 1..5 step 2) print(i) = 135
for (i in 5 downTo 1 step 2) print(i) = 531
```

Iterating Through an Array

Here's an example to iterate through a `String` array.

```
fun main(args: Array<String>) {

    var language = arrayOf("Ruby", "Kotlin", "Python" "Java")

    for (item in language)
        println(item)
}
```

Output

```
Ruby
Kotlin
Python
Java
```

It's possible to iterate through an array with an index. For example,

```
fun main(args: Array<String>) {

    var language = arrayOf("Ruby", "Kotlin", "Python", "Java")

    for (item in language.indices) {
```

```
        // printing array elements having even index only
        if (item%2 == 0)
            println(language[item])
    }
}
```

Output

```
Ruby
Python
```

Note: Here, `language.indices` returns all all indices of each array elements.

If you want to learn more about arrays, visit *Kotlin arrays*.

Iterating Through a String

```
fun main(args: Array<String>) {

    var text= "Kotlin"

    for (letter in text) {
        println(letter)
    }
}
```

Output

```
K
o
t
l
i
n
```

Similar like arrays, you can iterate through a `String` with an index. For example,

```
fun main(args: Array<String>) {

    var text= "Kotlin"

    for (item in text.indices) {
        println(text[item])
    }
}
```

Output

```
K
o
t
l
i
n
```

Kotlin break Expression

Suppose you are working with loops. It is sometimes desirable to terminate the loop immediately without checking the test expression.

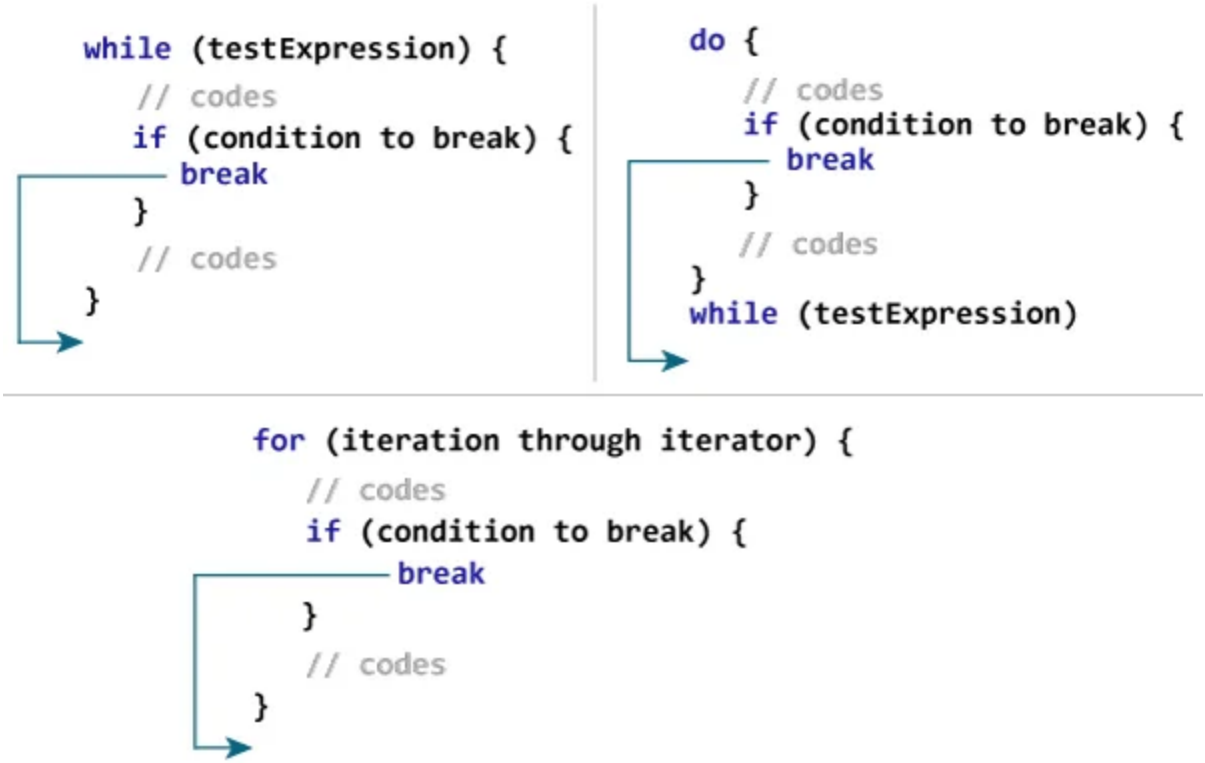
In such case, `break` is used. It terminates the nearest enclosing loop when encountered (without checking the test expression). This is similar to how break statement works in Java.

How break works?

It is almost always used with if..else construct. For example,

```
for (...) {
    if (testExpression) {
        break
    }
}
```

If testExpression is evaluated to `true`, `break` is executed which terminates the `for` loop.



Example: Kotlin break

```
fun main(args: Array<String>) {

    for (i in 1..10) {
        if (i == 5) {
            break
        }
        println(i)
    }
}
```

When you run the program, the output will be:

```
1
2
```

```
3
4
```

When the value of i is equal to 5, expression `i == 5` inside `if` is evaluated to `true`, and `break` is executed. This terminates the for loop.

Example: Calculate Sum Until User enters 0

The program below calculates the sum of numbers entered by the user until user enters 0.

Visit Kotlin Basic Input Output to learn more on how to take input from the user.

```
fun main(args: Array<String>) {

    var sum = 0
    var number: Int

    while (true) {
        print("Enter a number: ")
        number = readLine()!!.toInt()

        if (number == 0)
            break

        sum += number
    }

    print("sum = $sum")
}
```

When you run the program, the output will be:

```
Enter a number: 4
Enter a number: 12
Enter a number: 6
Enter a number: -9
Enter a number: 0
sum = 13
```

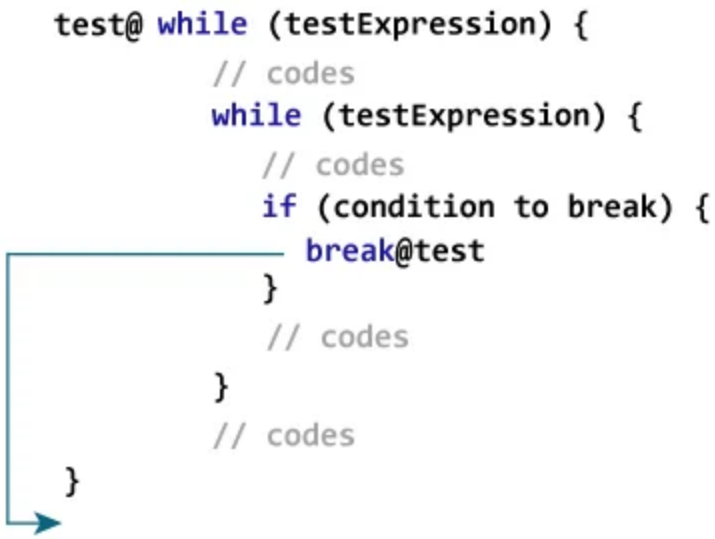
In the above program, the test expression of the `while` loop is always `true`.

Here, the `while` loop runs until user enters 0. When user inputs 0, `break` is executed which terminates the `while` loop.

Kotlin Labeled break

What you have learned till now is an unlabeled form of `break`, which terminates the nearest enclosing loop. There is another way `break` can be used (labeled form) to terminate the desired loop (can be an outer loop).

How labeled break works?



Label in Kotlin starts with an identifier which is followed by `@`.

Here, test@ is a label marked at the outer while loop. Now, by using `break` with a label (`break@test` in this case), you can break the specific loop.

Here's an example:

```
fun main(args: Array<String>) {

    first@ for (i in 1..4) {

        second@ for (j in 1..2) {
            println("i = $i; j = $j")

            if (i == 2)
```

```
                break@first
        }
    }
}
```

When you run the program, the output will be:

```
i = 1; j = 1
i = 1; j = 2
i = 2; j = 1
```

Here, when `i == 2` expression is evaluated to `true`, `break@first` is executed which terminates the loop marked with label `first@`.

Here's a little variation of the above program.

In the program below, break terminates the loop marked with label `@second.`

```
fun main(args: Array<String>) {

    first@ for (i in 1..4) {

        second@ for (j in 1..2) {
            println("i = $i; j = $j")

            if (i == 2)
                break@second
        }
    }
}
```

When you run the program, the output will be:

```
i = 1; j = 1
i = 1; j = 2
i = 2; j = 1
i = 3; j = 1
i = 3; j = 2
i = 4; j = 1
i = 4; j = 2
```

Note: Since, `break` is used to terminate the innermost loop in this program, it is not necessary to use labeled break in this case.

Kotlin continue Expression

Suppose you are working with loops. It is sometimes desirable to skip the current iteration of the loop.

In such case, `continue` is used. The `continue` construct skips the current iteration of the enclosing loop, and the control of the program jumps to the end of the loop body.

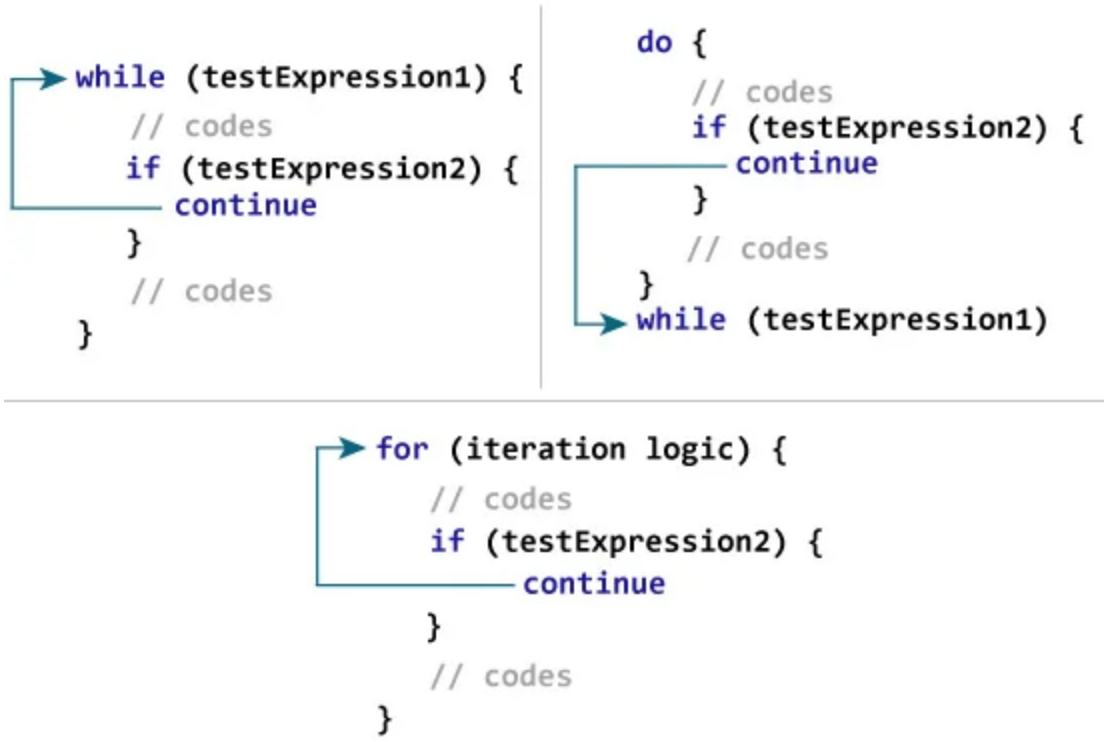
How continue works?

It is almost always used with if...else construct. For example,

```
while (testExpression1) {

    // codes1
    if (testExpression2) {
        continue
    }
    // codes2
}
```

If the testExpression2 is evaluated to `true`, `continue` is executed which skips all the codes inside `while` loop after it for that iteration.



Example: Kotlin continue

```
fun main(args: Array<String>) {

    for (i in 1..5) {
        println("$i Always printed.")
        if (i > 1 && i < 5) {
            continue
        }
        println("$i Not always printed.")
    }
}
```

When you run the program, the output will be:

```
1 Always printed.
1 Not always printed.
2 Always printed.
3 Always printed.
4 Always printed.
```

```
5 Always printed.
5 Not always printed.
```

When the value of i is greater than 1 and less than 5, `continue` is executed, which skips the execution of

```
println("$i Not always printed.")
```

statement.

However, the statement

```
println("$i Always printed.")
```

is executed in each iteration of the loop because this statement exists before the `continue` construct.

Example: Calculate Sum of Positive Numbers Only

The program below calculates the sum of maximum of 6 positive numbers entered by the user. If the user enters negative number or zero, it is skipped from calculation.

Visit Kotlin Basic Input Output to learn more on how to take input from the user.

```
fun main(args: Array<String>) {

    var number: Int
    var sum = 0

    for (i in 1..6) {
        print("Enter an integer: ")
        number = readLine()!!.toInt()

        if (number <= 0)
            continue

        sum += number
    }
    println("sum = $sum")
}
```

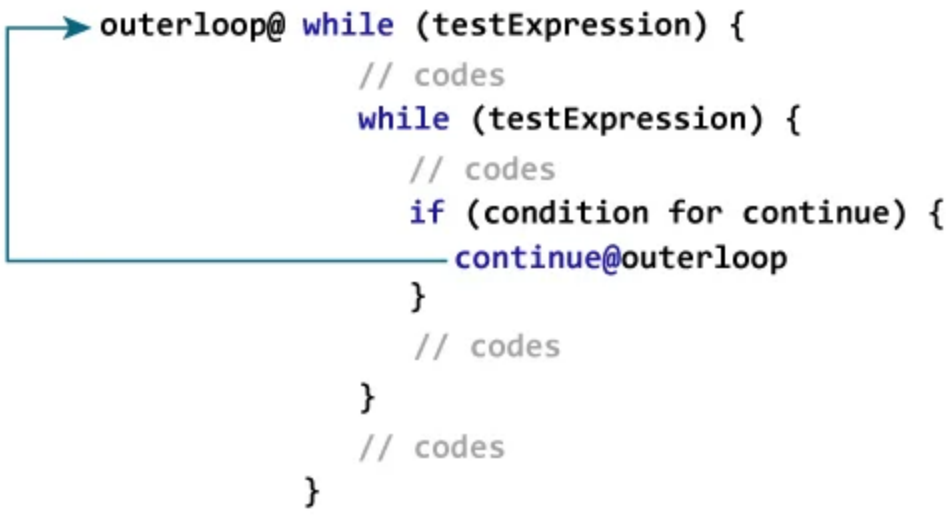
When you run the program, the output will be:

```
Enter an integer: 4
Enter an integer: 5
Enter an integer: -50
Enter an integer: 10
Enter an integer: 0
Enter an integer: 12
sum = 31
```

Kotlin Labeled continue

What you have learned till now is unlabeled form of `continue`, which skips current iteration of the nearest enclosing loop. `continue` can also be used to skip the iteration of the desired loop (can be outer loop) by using `continue` labels.

How labeled continue works?



Label in Kotlin starts with an identifier which is followed by `@`.

Here, outerloop@ is a label marked at outer while loop. Now, by using `continue` with the label (`continue@outerloop` in this case), you can skip the execution of codes of the specific loop for that iteration.

Example: labeled continue

```
fun main(args: Array<String>) {

    here@ for (i in 1..5) {
        for (j in 1..4) {
            if (i == 3 || j == 2)
                continue@here
            println("i = $i; j = $j")
        }
    }
}
```

When you run the program, the output will be:

```
i = 1; j = 1
i = 2; j = 1
i = 4; j = 1
i = 5; j = 1
```

The use of labeled `continue` is often discouraged as it makes your code hard to understand. If you are in a situation where you have to use labeled `continue`, refactor your code and try to solve it in a different way to make it more readable.