Java break Statement

In this tutorial, you will learn about the break statement, labeled break statement in Java with the help of examples.

While working with loops, it is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression.

In such cases, `break` and `continue` statements are used. You will learn about the Java continue statement in the next tutorial.

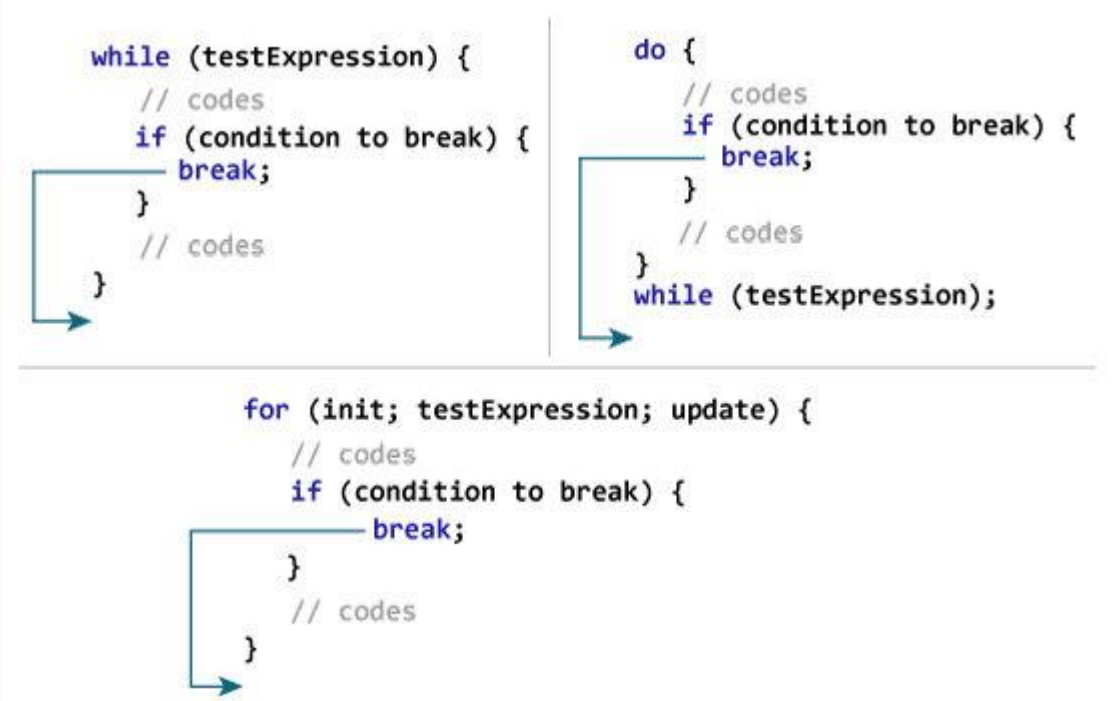
The `break` statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.

It is almost always used with decision-making statements (Java if...else Statement).

Here is the syntax of the break statement in Java:

```
break;
```

How break statement works?



Working of Java break Statement

Example 1: Java break statement

```
class Test {

    public static void main(String[] args) {


        // for loop

        for (int i = 1; i <= 10; ++i) {


            // if the value of i is 5 the loop terminates

            if (i == 5) {

                break;

            }

            System.out.println(i);

        }

    }

}
```

Output:

```
1

2

3

4
```

In the above program, we are using the `for` loop to print the value of `i` in each iteration. To know how `for` loop works, visit the Java for loop. Here, notice the statement,

```
if (i == 5) {

    break;

}
```

This means when the value of `i` is equal to 5, the loop terminates. Hence we get the output with values less than 5 only.

Example 2: Java break statement

The program below calculates the sum of numbers entered by the user until user enters a negative number.

To take input from the user, we have used the `Scanner` object. To learn more about `Scanner`, visit Java Scanner.

```
import java.util.Scanner;

class UserInputSum {

    public static void main(String[] args) {


        Double number, sum = 0.0;


        // create an object of Scanner

        Scanner input = new Scanner(System.in);


        while (true) {
```

```
            System.out.print("Enter a number: ");


            // takes double input from user

            number = input.nextDouble();


            // if number is negative the loop terminates

            if (number < 0.0) {

                break;

            }


           sum += number;

        }

        System.out.println("Sum = " + sum);

    }

}
```

Output:

```
Enter a number: 3.2

Enter a number: 5

Enter a number: 2.3

Enter a number: 0

Enter a number: -4.5

Sum = 10.5
```

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

```
if (number < 0.0) {

    break;

}
```

This means when the user input negative numbers, the while loop is terminated.

Java break and Nested Loop

In the case of nested loops, the `break` statement terminates the innermost loop.

```
while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (condition to break) {
            break;
        }
        // codes
    }
    // codes
}
```

Working of break Statement with Nested Loops

Here, the break statement terminates the innermost `while` loop, and control jumps to the outer loop.

Labeled break Statement

Till now, we have used the unlabeled break statement. It terminates the innermost loop and switch statement. However, there is another form of break statement in Java known as the labeled break.

We can use the labeled break statement to terminate the outermost loop as well.

```
label:
for (int; testExpresison, update) {
    // codes
    for (int; testExpression; update) {
        // codes
        if (condition to break) {
            break label;
        }
        // codes
    }
    // codes
}
```

Working of the labeled break statement in Java

As you can see in the above image, we have used the `label` identifier to specify the outer loop. Now, notice how the `break` statement is used (`break label;`).

Here, the `break` statement is terminating the labeled statement (i.e. outer loop). Then, the control of the program jumps to the statement after the labeled statement.

Here's another example:

```
while (testExpression) {
   // codes
   second:
   while (testExpression) {
      // codes
      while(testExpression) {
         // codes
         break second;
      }
   }
   // control jumps here
}
```

In the above example, when the statement `break second;` is executed, the `while` loop labeled as `second` is terminated. And, the control of the program moves to the statement after the second `while` loop.

Example 3: labeled break Statement

```
class LabeledBreak {
    public static void main(String[] args) {


        // the for loop is labeled as first
        first:
        for( int i = 1; i < 5; i++) {
```

```
            // the for loop is labeled as second
            second:
            for(int j = 1; j < 3; j ++ ) {
                System.out.println("i = " + i + "; j = " +j);

                // the break statement breaks the first for loop
                if ( i == 2)
                    break first;
            }
        }
    }
}
```

Output:

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

In the above example, the `labeled break` statement is used to terminate the loop labeled as first. That is,

```
first:for(int i = 1; i < 5; i++) {...}
```

Here, if we change the statement `break first;` to `break second;` the program will behave differently. In this case, `for` loop labeled as second will be terminated. For example,

```
class LabeledBreak {
    public static void main(String[] args) {

        // the for loop is labeled as first
        first:
        for( int i = 1; i < 5; i++) {

            // the for loop is labeled as second
```

```
            second:
            for(int j = 1; j < 3; j ++ ) {

                System.out.println("i = " + i + "; j = " +j);

                // the break statement terminates the loop labeled as second
                if ( i == 2)
                    break second;
            }
        }
    }
}
```

Output:

```
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**: The `break` statement is also used to terminate cases inside the `switch` statement.

Java continue Statement

In this tutorial, you will learn about the continue statement and labeled continue statement in Java with the help of examples.

While working with loops, sometimes you might want to skip some statements or terminate the loop. In such cases, `break` and `continue` statements are used.

To learn about the `break` statement, visit Java break. Here, we will learn about the `continue` statement.

Java continue

The `continue` statement skips the current iteration of a loop (`for`, `while`, `do...while`, etc).

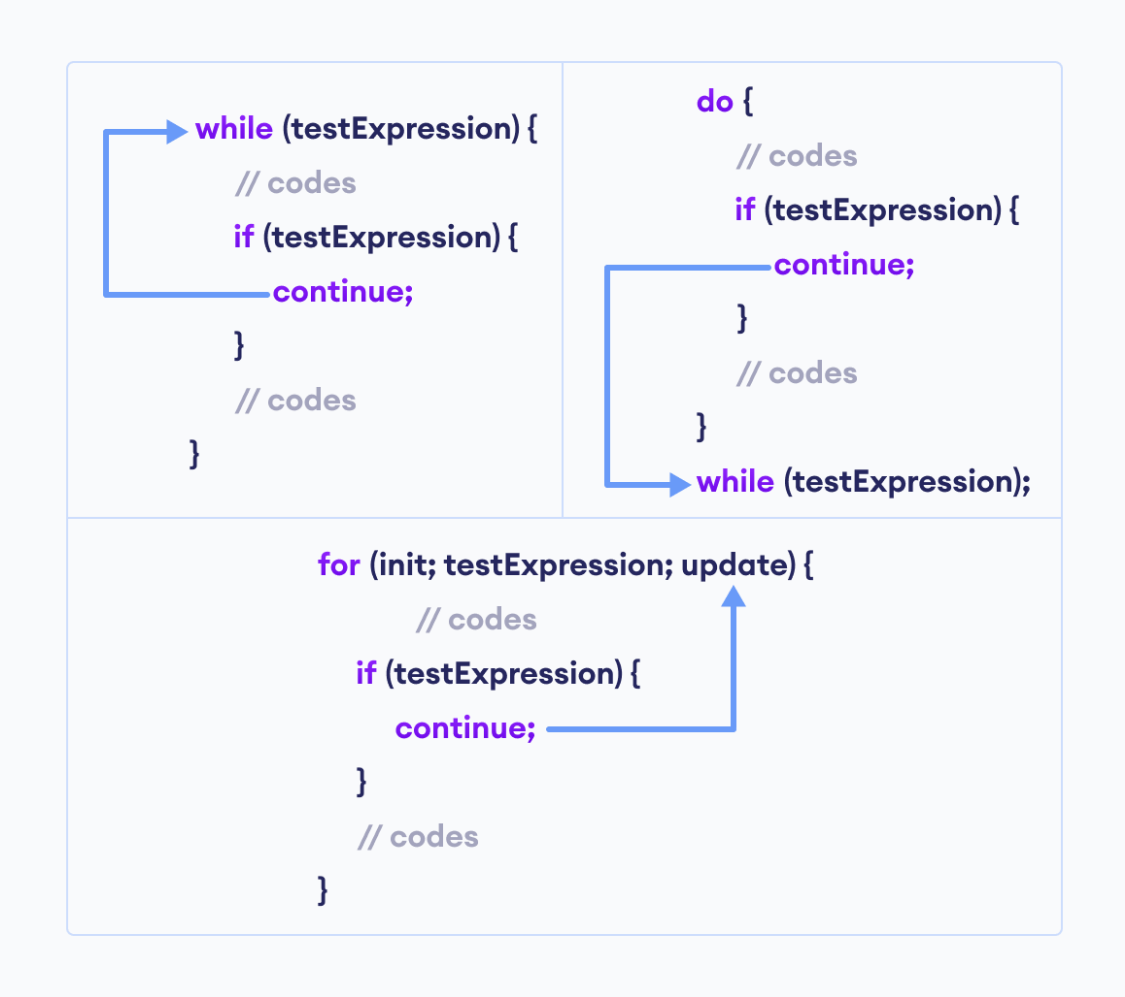
After the `continue` statement, the program moves to the end of the loop. And, test expression is evaluated (update statement is evaluated in case of the for loop).

Here's the syntax of the continue statement.

```
continue;
```

> **Note**: The continue statement is almost always used in decision-making statements (if...else Statement).

Working of Java continue statement



Working of Java continue Statement

Example 1: Java continue statement

```
class Main {

  public static void main(String[] args) {


    // for loop

    for (int i = 1; i <= 10; ++i) {


      // if value of i is between 4 and 9

      // continue is executed

      if (i > 4 && i < 9) {

        continue;
```

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

Output:

```
1
2
    3
4
9
10
```

In the above program, we are using the `for` loop to print the value of `i` in each iteration. To know how `for` loop works, visit Java for loop. Notice the statement,

```
if (i > 5 && i < 9) {
    continue;
}
```

Here, the `continue` statement is executed when the value of `i` becomes more than **4** and less than **9**.

It then skips the print statement inside the loop. Hence we get the output with values **5**, **6**, **7**, and **8** skipped.

Example 2: Compute the sum of 5 positive numbers

```
import java.util.Scanner;
class Main {
  public static void main(String[] args) {
```

```
    Double number, sum = 0.0;

    // create an object of Scanner

    Scanner input = new Scanner(System.in);


    for (int i = 1; i < 6; ++i) {

      System.out.print("Enter number " + i + " : ");

      // takes input from the user

      number = input.nextDouble();


      // if number is negative

      // continue statement is executed

      if (number <= 0.0) {

        continue;

      }


      sum += number;

    }

    System.out.println("Sum = " + sum);

    input.close();

  }

}
```

Output:

```
Enter number 1: 2.2

Enter number 2: 5.6

Enter number 3: 0

Enter number 4: -2.4

Enter number 5: -3

Sum = 7.8
```

In the above example, we have used the for loop to print the sum of 5 positive numbers. Notice the line,

---

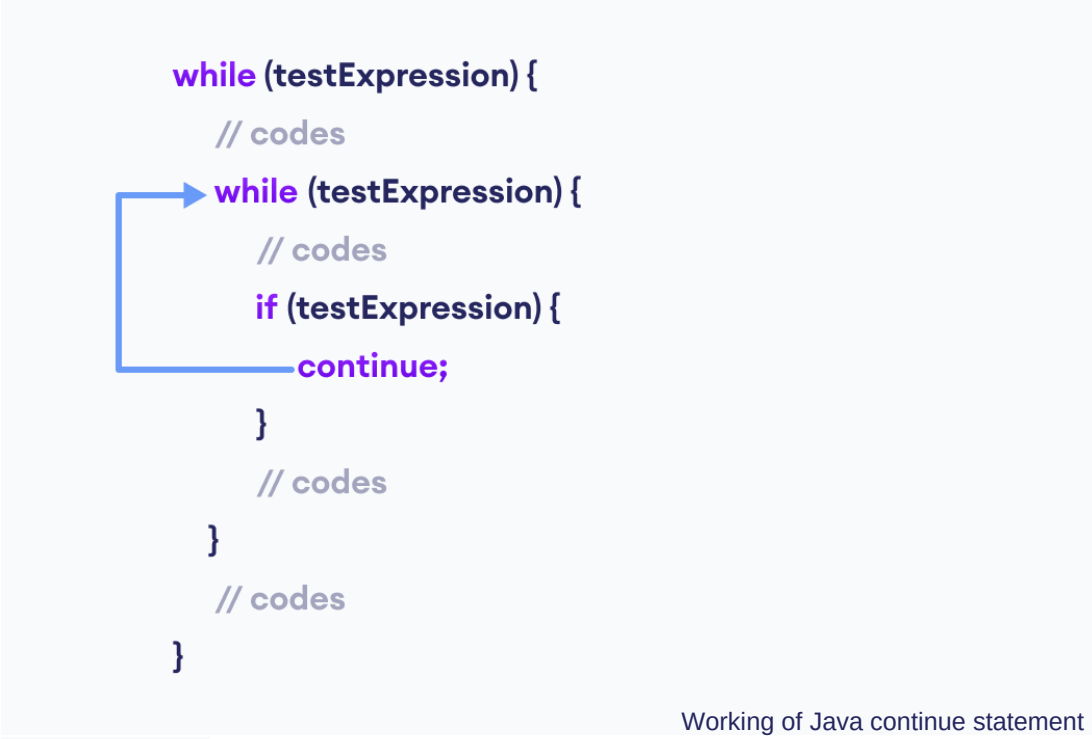
```
if (number < 0.0) {
    continue;
}
```

Here, when the user enters a negative number, the `continue` statement is executed. This skips the current iteration of the loop and takes the program control to the update expression of the loop.

> **Note**: To take input from the user, we have used the `Scanner` object. To learn more, visit Java Scanner.

Java continue with Nested Loop

In the case of nested loops in Java, the `continue` statement skips the current iteration of the innermost loop.



Working of Java continue statement with Nested Loops

Example 3: continue with Nested Loop

```
class Main {

  public static void main(String[] args) {


    int i = 1, j = 1;


    // outer loop

    while (i <= 3) {


      System.out.println("Outer Loop: " + i);


      // inner loop

      while(j <= 3) {


        if(j == 2) {

          j++;

          continue;

        }


        System.out.println("Inner Loop: " + j);

        j++;

      }

      i++;

    }

  }

}
```

Output

```
Outer Loop: 1

Inner Loop: 1

Inner Loop: 3
```

```
Outer Loop: 2

Outer Loop: 3
```

In the above example, we have used the nested while loop. Note that we have used the `continue` statement inside the inner loop.

```
if(j == 2) {

  j++;

  continue:

}
```

Here, when the value of `j` is **2**, the value of `j` is increased and the `continue` statement is executed.

This skips the iteration of the inner loop. Hence, the text `Inner Loop: 2` is skipped from the output.

Labeled continue Statement

Till now, we have used the unlabeled `continue` statement. However, there is another form of `continue` statement in Java known as **labeled continue**.

It includes the label of the loop along with the `continue` keyword. For example,

```
continue label;
```

Here, the `continue` statement skips the current iteration of the loop specified by `label`.

Working of the Java labeled continue Statement

We can see that the label identifier specifies the outer loop. Notice the use of the continue inside the inner loop.

Here, the `continue` statement is skipping the current iteration of the labeled statement (i.e. outer loop). Then, the program control goes to the next iteration of the labeled statement.

Example 4: labeled continue Statement

```
class Main {

  public static void main(String[] args) {


    // outer loop is labeled as first

    first:

    for (int i = 1; i < 6; ++i) {


      // inner loop

      for (int j = 1; j < 5; ++j) {
```

```
        if (i == 3 || j == 2)


          // skips the current iteration of outer loop

          continue first;

        System.out.println("i = " + i + "; j = " + j);

      }

    }

  }

}
```

Output:

```
i = 1; j = 1

i = 2; j = 1

i = 4; j = 1

i = 5; j = 1
```

In the above example, the `labeled continue` statement is used to skip the current iteration of the loop labeled as `first`.

```
if (i==3 || j==2)

    continue first;
```

Here, we can see the outermost `for` loop is labeled as `first`,

```
first:for (int i = 1; i < 6; ++i) {..}
```

Hence, the iteration of the outer `for` loop is skipped if the value of `i` is 3 or the value of `j` is 2.

> **Note**: 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.
