

Java for Loop

In computer programming, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then rather than typing the same code 100 times, you can use a loop.

In Java, there are three types of loops.

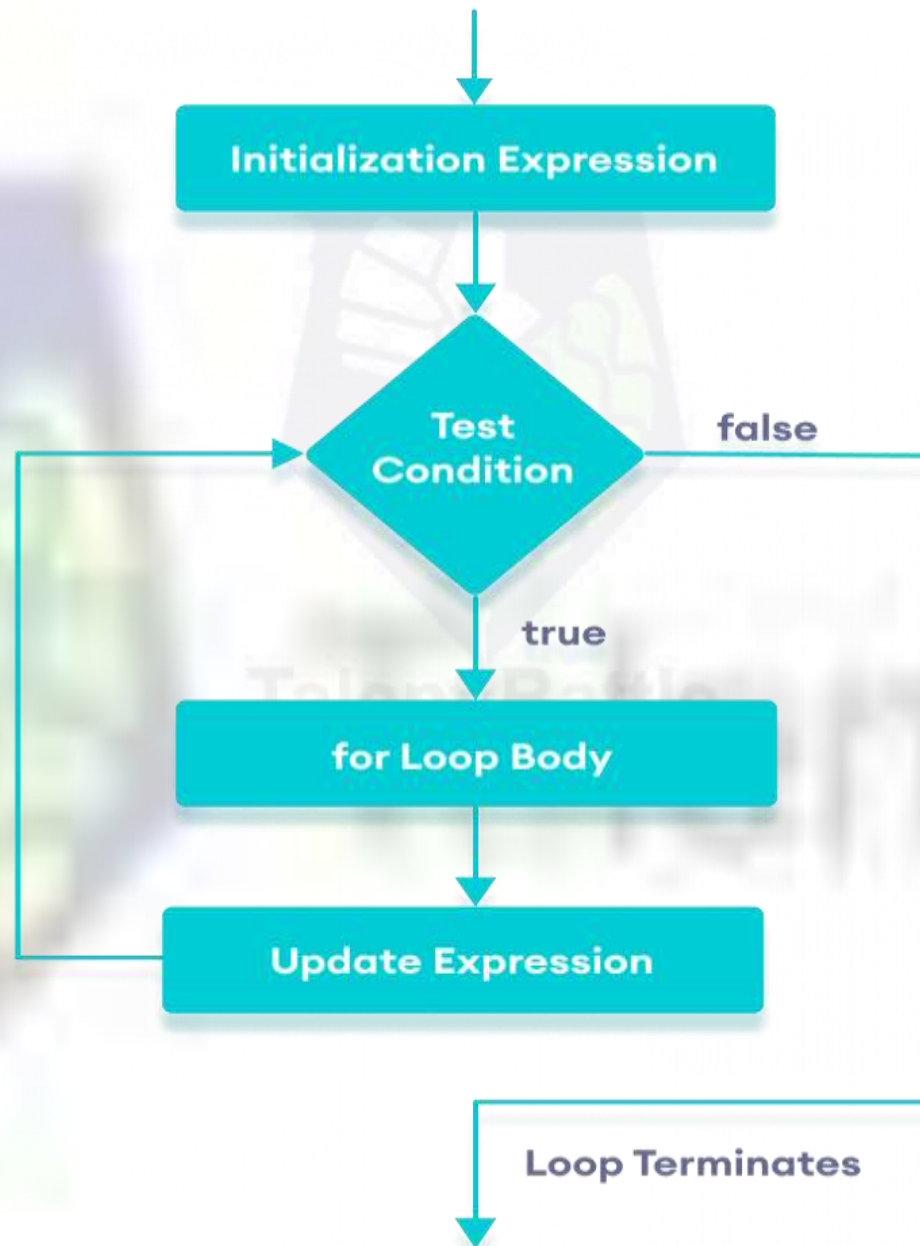
1. for loop
2. while loop
3. do...while loop

Java for Loop

Java for loop is used to run a block of code for a certain number of times.

The syntax of for loop is:

```
for (initialExpression; testExpression; updateExpression) {  
    // body of the loop  
}
```



// Program to print numbers from 1 to 5

```
class Main {  
    public static void main(String[] args) {  
  
        int n = 5;  
        // for loop  
        for (int i = 1; i <= n; ++i) {  
            System.out.println(i);  
        }  
    }  
}
```

// Program to find the sum of natural numbers from 1 to 1000.

```
class Main {  
    public static void main(String[] args) {  
  
        int sum = 0;  
        int n = 1000;  
  
        // for loop  
        for (int i = 1; i <= n; ++i) {  
            // body inside for loop  
            sum += i;    // sum = sum + i  
        }  
  
        System.out.println("Sum = " + sum);  
    }  
}
```

Java for-each Loop

The Java for loop has an alternative syntax that makes it easy to iterate through arrays and collections. For example,

```
// print array elements
```

```
class Main {  
    public static void main(String[] args) {  
        // create an array  
        int[] numbers = {3, 7, 5, -5};  
        // iterating through the array  
        for (int number: numbers) {  
            System.out.println(number);  
        }  
    }  
}
```

for-each Loop Syntax

The syntax of the Java for-each loop is:

```
for(dataType item : array) {  
    ...  
}
```

Here,

array - an array or a collection

item - each item of array/collection is assigned to this variable

dataType - the data type of the array/collection

// Calculate the sum of all elements of an array

```
class Main {  
    public static void main(String[] args) {  
  
        // an array of numbers  
        int[] numbers = {3, 4, 5, -5, 0, 12};  
        int sum = 0;  
  
        // iterating through each element of the array  
        for (int number: numbers) {  
            sum += number;  
        }  
  
        System.out.println("Sum = " + sum);  
    }  
}
```


for loop Vs for-each loop

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

```
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
```

```
        // iterating through an array using a for  
        loop
```

```
        for (int i = 0; i < vowels.length; ++ i) {  
            System.out.println(vowels[i]);
```

```
        }  
    }  
}
```

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

```
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
```

```
        // iterating through an array using the for-  
        each loop
```

```
        for (char item: vowels) {  
            System.out.println(item);
```

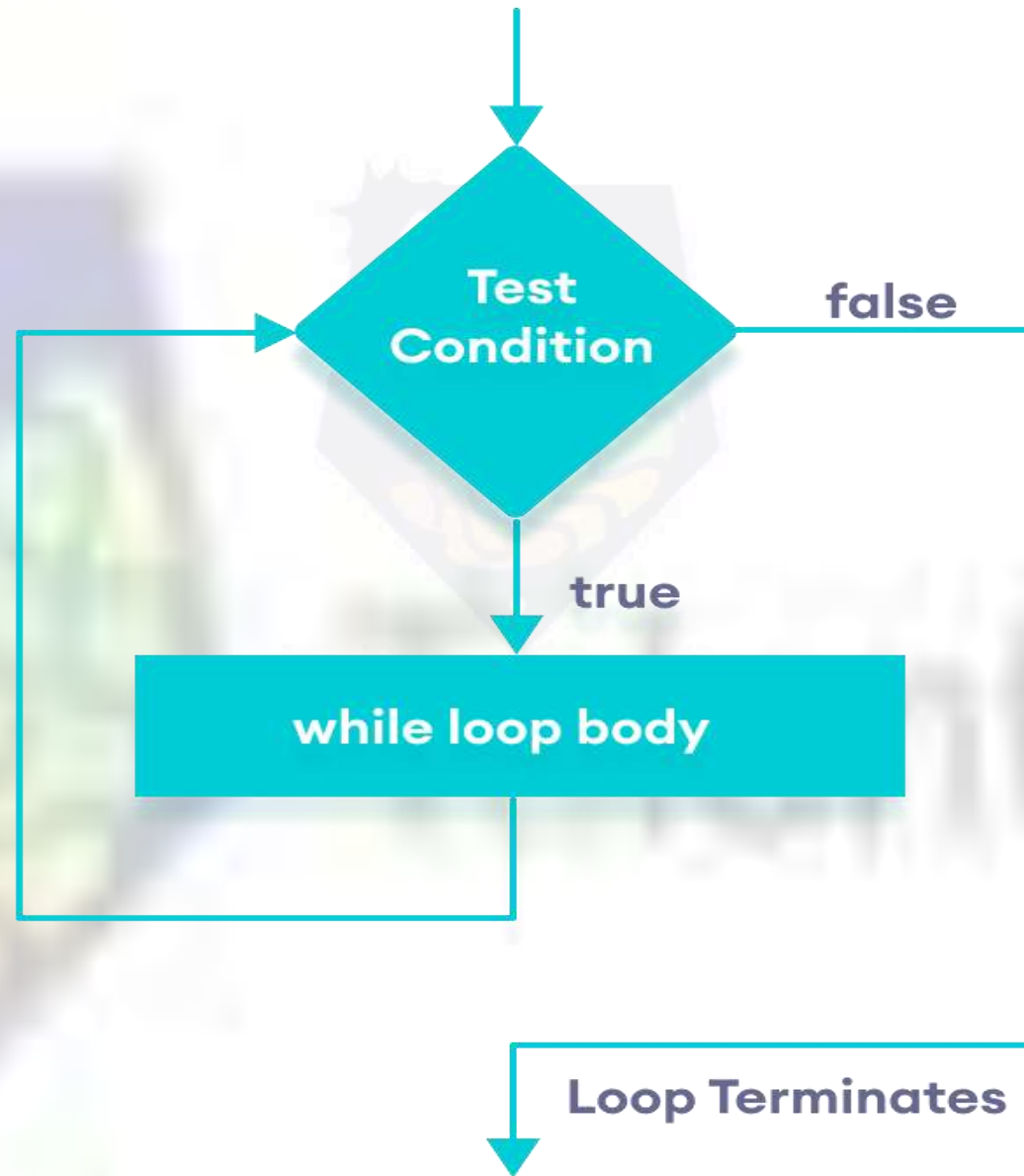
```
        }  
    }  
}
```

Java while and do...while Loop

Java while loop

Java while loop is used to run a specific code until a certain condition is met.
The syntax of the while loop is:

```
while (testExpression) {  
    // body of loop  
}
```



// Program to display numbers from 1 to 5

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

```
        // declare variables  
        int i = 1, n = 5;
```

```
        // while loop from 1 to 5  
        while(i <= n) {  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

```
// Java program to find the sum of positive numbers
```

```
import java.util.Scanner;
```

```
class Main {
```

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

```
        int sum = 0;
```

```
        // create an object of Scanner class
```

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

```
        // take integer input from the user
```

```
        System.out.println("Enter a number");
```

```
        int number = input.nextInt();
```

```
        // while loop continues
```

```
        // until entered number is positive
```

```
        while (number >= 0) {
```

```
            // add only positive numbers
```

```
            sum += number;
```

```
            System.out.println("Enter a number");
```

```
            number = input.nextInt();
```

```
        }
```

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

```
        input.close();
```

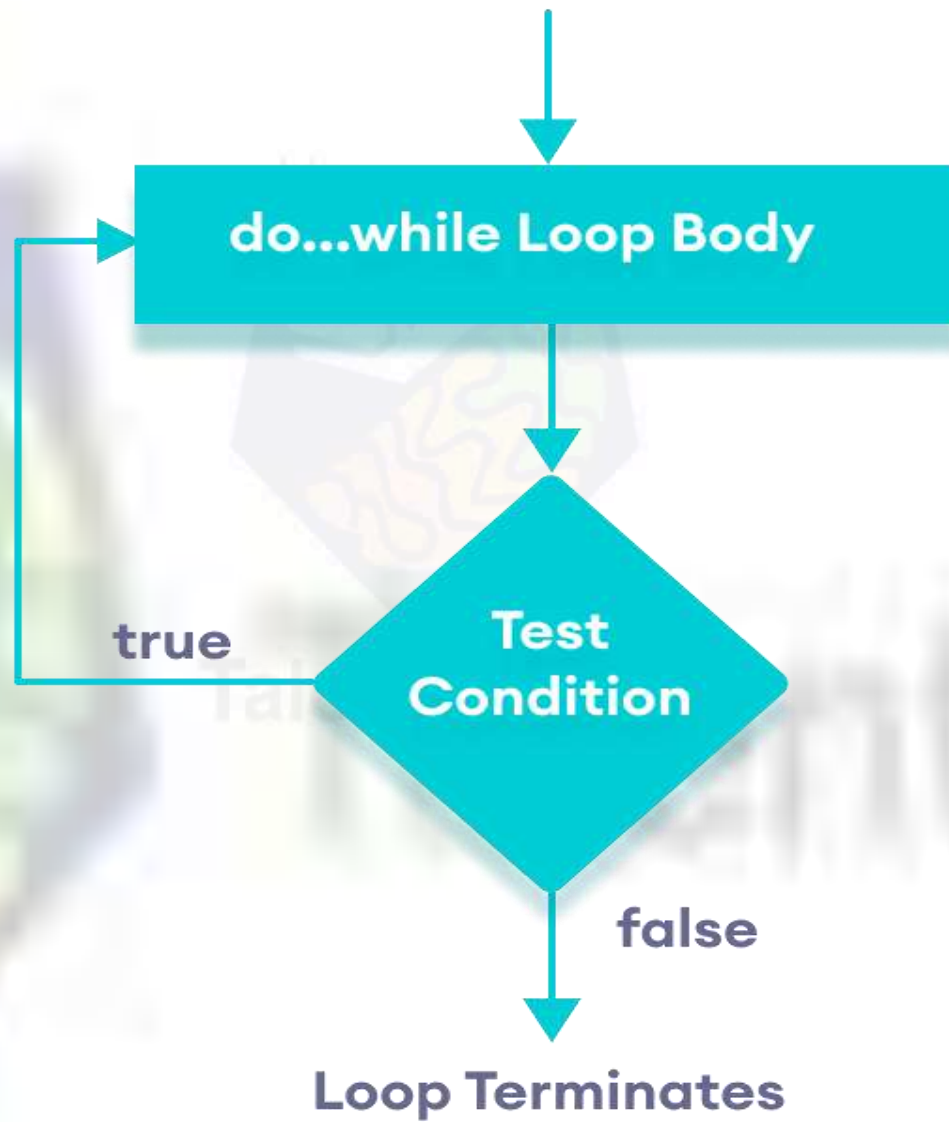
```
    }
```

```
}
```

Java do...while loop

The do...while loop is similar to while loop. However, the body of do...while loop is executed once before the test expression is checked. For example,

```
do {  
    // body of loop  
} while(textExpression)
```



// Java Program to display numbers from 1 to 5

```
import java.util.Scanner;
```

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

```
        int i = 1, n = 5;
```

```
        // do...while loop from 1 to 5
```

```
        do {
```

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

```
            i++;
```

```
        } while(i <= n);
```

```
    }
```

```
}
```

for and while loops

The for loop is used when the number of iterations is known. For example,

```
for (let i = 1; i <=5; ++i) {  
    // body of loop  
}
```

And while and do...while loops are generally used when the number of iterations is unknown. For example,

```
while (condition) {  
    // body of loop  
}
```

Java break Statement

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.


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;
```

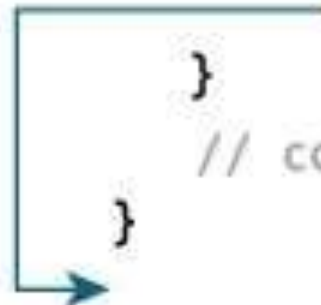
```
while (testExpression) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```



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




```
for (init; testExpression; update) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```



```
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);  
        }  
    }  
}
```

Labeled break Statement

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



The diagram illustrates the execution of the 'break label;' statement. A teal arrow originates from the 'break label;' line within the inner 'for' loop and points to the 'label:' line, which is the start of the outer 'for' loop. This visualizes how the break statement immediately exits the inner loop and jumps to the beginning of the labeled outer loop.

```
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.


```
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;  
            }  
        }  
    }  
}
```

Java continue Statement

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

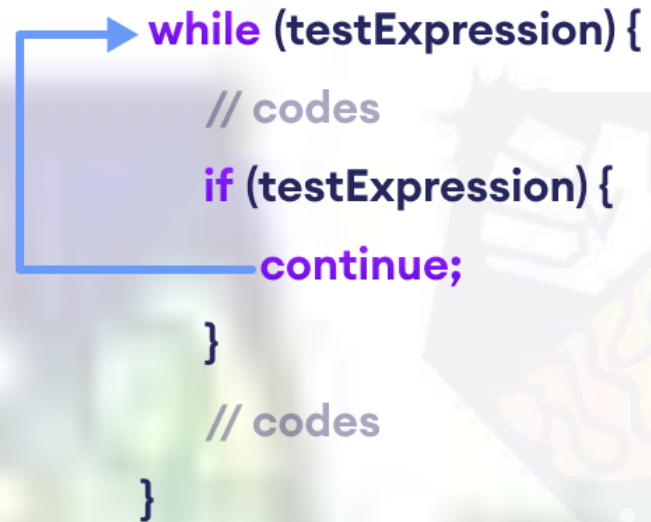
After the continue statement, the program moves to the end of the loop. And, test expression is evaluated (update statement is evaluated in case of the for loop).

Here's the syntax of the continue statement.

```
continue;
```

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

```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```



```
for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



```
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);  
        }  
    }  
}
```

Java continue with Nested Loop

```
while (testExpression) {
```

```
    // codes
```

```
    while (testExpression) {
```

```
        // codes
```

```
        if (testExpression) {
```

```
            continue;
```

```
        }
```

```
        // codes
```

```
    }
```

```
    // codes
```

```
}
```



```
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++;  
        }  
    }  
}
```

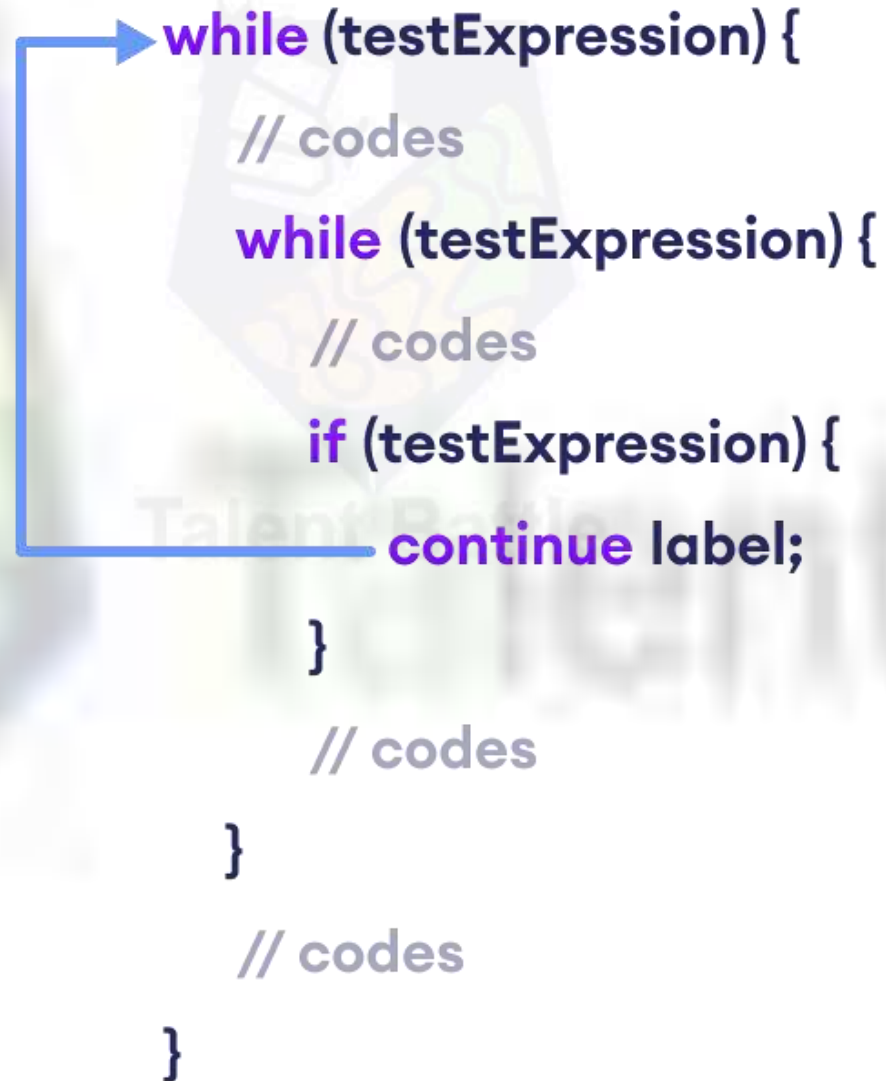
Labeled continue Statement

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.

label:



```
while (testExpression) {  
    // codes  
    while (testExpression) {  
        // codes  
        if (testExpression) {  
            continue label;  
        }  
        // codes  
    }  
    // codes  
}
```

The diagram illustrates a loop structure. A blue arrow originates from the **continue** statement inside the nested **while** loop and points back to the start of the outer **while** loop, indicating that the loop restarts from the beginning when the **continue** statement is executed.

```
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);  
            }  
        }  
    }  
}
```

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.

1. Java Program to Check Leap Year
2. Java Program to Check Whether a Number is Positive or Negative
3. Java Program to Check Whether a Character is Alphabet or Not
4. Java Program to Calculate the Sum of Natural Numbers
5. Java Program to Find Factorial of a Number
6. Java Program to Generate Multiplication Table
7. Java Program to Display Fibonacci Series
8. Java Program to Find GCD of two Numbers
9. Java Program to Find LCM of two Numbers
10. Java Program to Display Alphabets (A to Z) using loop
11. Java Program to Count Number of Digits in an Integer
12. Java Program to Reverse a Number
13. Java Program to Calculate the Power of a Number
14. Java Program to Check Palindrome
15. Java Program to Check Whether a Number is Prime or Not
16. Java Program to Display Prime Numbers Between Two Intervals
17. Java Program to Check Armstrong Number
18. Java Program to Display Armstrong Number Between Two Intervals
19. Java Program to Display Factors of a Number
20. Java Program to Make a Simple Calculator Using switch...case
21. Java Program to Count the Number of Vowels and Consonants in a Sentence
22. Java Program to Sort Elements in Lexicographical Order (Dictionary Order)
23. Java Code To Create Pyramid and Pattern

Which of the following is not a decision making statement?

- a) If else
- b) Switch
- c) If
- d) Do-while

In java, __ can evaluate any type of Boolean expression and __ can only test for equality.

- a) if, break
- b) If, continue
- c) If, switch
- d) Break, if

Literal can be of which of these data types?

- a) Boolean
- b) Float
- c) Integer
- d) All of these

Which of the following loops will execute the body of loop even when condition controlling the loop is initially false?

- a) While
- b) For
- c) Switch
- d) Do-while

The __ loop is especially useful when you process a menu selection.

- a) While
- b) For
- c) Do-while
- d) None of these

Which of these jump statements can skip processing the remaining of the code in its body for a particular iteration?

- a) Return
- b) Break
- c) Continue
- d) exit