Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

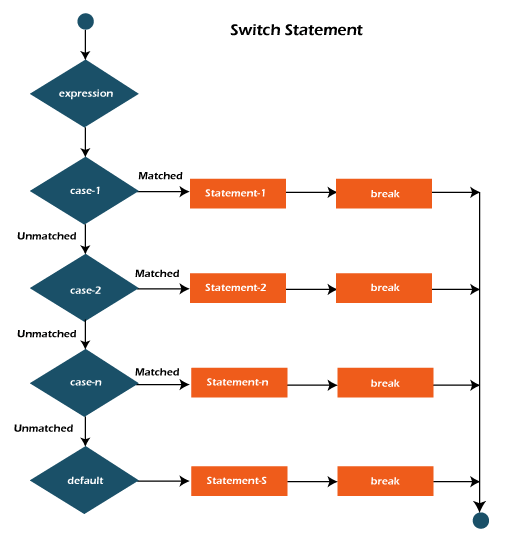
Points to Remember

* There can be *one or N number of case values* for a switch expression.
* The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
* The case values must be *unique*. In case of duplicate value, it renders compile-time error.
* The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
* Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
* The case value can have a *default label* which is optional.

**Syntax:**

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;  //optional
5. **case** value2:
6. //code to be executed;
7. **break**;  //optional
8. ......
10. **default**:
11. code to be executed **if** all cases are not matched;
12. }

**Flowchart of Switch Statement**



**Example:**

**SwitchExample.java**

1. **public** **class** SwitchExample {
2. **public** **static** **void** main(String[] args) {
3. //Declaring a variable for switch expression
4. **int** number=20;
5. //Switch expression
6. **switch**(number){
7. //Case statements
8. **case** 10: System.out.println("10");
9. **break**;
10. **case** 20: System.out.println("20");
11. **break**;
12. **case** 30: System.out.println("30");
13. **break**;
14. //Default case statement
15. **default**:System.out.println("Not in 10, 20 or 30");
16. }
17. }
18. }

**Output:**

20

**Finding Month Example:**

**SwitchMonthExample.javaHTML**

1. //Java Program to demonstrate the example of Switch statement
2. //where we are printing month name for the given number
3. **public** **class** SwitchMonthExample {
4. **public** **static** **void** main(String[] args) {
5. //Specifying month number
6. **int** month=7;
7. String monthString="";
8. //Switch statement
9. **switch**(month){
10. //case statements within the switch block
11. **case** 1: monthString="1 - January";
12. **break**;
13. **case** 2: monthString="2 - February";
14. **break**;
15. **case** 3: monthString="3 - March";
16. **break**;
17. **case** 4: monthString="4 - April";
18. **break**;
19. **case** 5: monthString="5 - May";
20. **break**;
21. **case** 6: monthString="6 - June";
22. **break**;
23. **case** 7: monthString="7 - July";
24. **break**;
25. **case** 8: monthString="8 - August";
26. **break**;
27. **case** 9: monthString="9 - September";
28. **break**;
29. **case** 10: monthString="10 - October";
30. **break**;
31. **case** 11: monthString="11 - November";
32. **break**;
33. **case** 12: monthString="12 - December";
34. **break**;
35. **default**:System.out.println("Invalid Month!");
36. }
37. //Printing month of the given number
38. System.out.println(monthString);
39. }
40. }

**Output:**

7 - July

**Program to check Vowel or Consonant:**

If the character is A, E, I, O, or U, it is vowel otherwise consonant. It is not case-sensitive.

**SwitchVowelExample.java**

1. **public** **class** SwitchVowelExample {
2. **public** **static** **void** main(String[] args) {
3. **char** ch='O';
4. **switch**(ch)
5. {
6. **case** 'a':
7. System.out.println("Vowel");
8. **break**;
9. **case** 'e':
10. System.out.println("Vowel");
11. **break**;
12. **case** 'i':
13. System.out.println("Vowel");
14. **break**;
15. **case** 'o':
16. System.out.println("Vowel");
17. **break**;
18. **case** 'u':
19. System.out.println("Vowel");
20. **break**;
21. **case** 'A':
22. System.out.println("Vowel");
23. **break**;
24. **case** 'E':
25. System.out.println("Vowel");
26. **break**;
27. **case** 'I':
28. System.out.println("Vowel");
29. **break**;
30. **case** 'O':
31. System.out.println("Vowel");
32. **break**;
33. **case** 'U':
34. System.out.println("Vowel");
35. **break**;
36. **default**:
37. System.out.println("Consonant");
38. }
39. }
40. }

**Output:**

Vowel

Java Switch Statement is fall-through

The Java switch statement is fall-through. It means it executes all statements after the first match if a break statement is not present.

**Example:**

**SwitchExample2.java**

1. //Java Switch Example where we are omitting the
2. //break statement
3. **public** **class** SwitchExample2 {
4. **public** **static** **void** main(String[] args) {
5. **int** number=20;
6. //switch expression with int value
7. **switch**(number){
8. //switch cases without break statements
9. **case** 10: System.out.println("10");
10. **case** 20: System.out.println("20");
11. **case** 30: System.out.println("30");
12. **default**:System.out.println("Not in 10, 20 or 30");
13. }
14. }
15. }

**Output:**

20

30

Not in 10, 20 or 30

Java Switch Statement with String

Java allows us to use strings in switch expression since Java SE 7. The case statement should be string literal.

**Example:**

**SwitchStringExample.java**

1. //Java Program to demonstrate the use of Java Switch
2. //statement with String
3. **public** **class** SwitchStringExample {
4. **public** **static** **void** main(String[] args) {
5. //Declaring String variable
6. String levelString="Expert";
7. **int** level=0;
8. //Using String in Switch expression
9. **switch**(levelString){
10. //Using String Literal in Switch case
11. **case** "Beginner": level=1;
12. **break**;
13. **case** "Intermediate": level=2;
14. **break**;
15. **case** "Expert": level=3;
16. **break**;
17. **default**: level=0;
18. **break**;
19. }
20. System.out.println("Your Level is: "+level);
21. }
22. }

**Output:**

Your Level is: 3

Java Nested Switch Statement

We can use switch statement inside other switch statement in Java. It is known as nested switch statement.

**Example:**

**NestedSwitchExample.java**

1. //Java Program to demonstrate the use of Java Nested Switch
2. **public** **class** NestedSwitchExample {
3. **public** **static** **void** main(String args[])
4. {
5. //C - CSE, E - ECE, M - Mechanical
6. **char** branch = 'C';
7. **int** collegeYear = 4;
8. **switch**( collegeYear )
9. {
10. **case** 1:
11. System.out.println("English, Maths, Science");
12. **break**;
13. **case** 2:
14. **switch**( branch )
15. {
16. **case** 'C':
17. System.out.println("Operating System, Java, Data Structure");
18. **break**;
19. **case** 'E':
20. System.out.println("Micro processors, Logic switching theory");
21. **break**;
22. **case** 'M':
23. System.out.println("Drawing, Manufacturing Machines");
24. **break**;
25. }
26. **break**;
27. **case** 3:
28. **switch**( branch )
29. {
30. **case** 'C':
31. System.out.println("Computer Organization, MultiMedia");
32. **break**;
33. **case** 'E':
34. System.out.println("Fundamentals of Logic Design, Microelectronics");
35. **break**;
36. **case** 'M':
37. System.out.println("Internal Combustion Engines, Mechanical Vibration");
38. **break**;
39. }
40. **break**;
41. **case** 4:
42. **switch**( branch )
43. {
44. **case** 'C':
45. System.out.println("Data Communication and Networks, MultiMedia");
46. **break**;
47. **case** 'E':
48. System.out.println("Embedded System, Image Processing");
49. **break**;
50. **case** 'M':
51. System.out.println("Production Technology, Thermal Engineering");
52. **break**;
53. }
54. **break**;
55. }
56. }
57. }

**Output:**

Data Communication and Networks, MultiMedia

Java Enum in Switch Statement

Java allows us to use enum in switch statement. Java enum is a class that represent the group of constants. (immutable such as final variables). We use the keyword enum and put the constants in curly braces separated by comma.

**Example:**

**JavaSwitchEnumExample.java**

1. //Java Program to demonstrate the use of Enum
2. //in switch statement
3. **public** **class** JavaSwitchEnumExample {
4. **public** **enum** Day {  Sun, Mon, Tue, Wed, Thu, Fri, Sat  }
5. **public** **static** **void** main(String args[])
6. {
7. Day[] DayNow = Day.values();
8. **for** (Day Now : DayNow)
9. {
10. **switch** (Now)
11. {
12. **case** Sun:
13. System.out.println("Sunday");
14. **break**;
15. **case** Mon:
16. System.out.println("Monday");
17. **break**;
18. **case** Tue:
19. System.out.println("Tuesday");
20. **break**;
21. **case** Wed:
22. System.out.println("Wednesday");
23. **break**;
24. **case** Thu:
25. System.out.println("Thursday");
26. **break**;
27. **case** Fri:
28. System.out.println("Friday");
29. **break**;
30. **case** Sat:
31. System.out.println("Saturday");
32. **break**;
33. }
34. }
35. }
36. }

**Output:**

Sunday

Monday

Twesday

Wednesday

Thursday

Friday

Saturday

Java Wrapper in Switch Statement

Java allows us to use four wrapper classes: Byte, Short, Integer and Long in switch statement.

**Example:**

**WrapperInSwitchCaseExample.java**

1. //Java Program to demonstrate the use of Wrapper class
2. //in switch statement
3. **public** **class** WrapperInSwitchCaseExample {
4. **public** **static** **void** main(String args[])
5. {
6. Integer age = 18;
7. **switch** (age)
8. {
9. **case** (16):
10. System.out.println("You are under 18.");
11. **break**;
12. **case** (18):
13. System.out.println("You are eligible for vote.");
14. **break**;
15. **case** (65):
16. System.out.println("You are senior citizen.");
17. **break**;
18. **default**:
19. System.out.println("Please give the valid age.");
20. **break**;
21. }
22. }
23. }

**Output:**

You are eligible for vote.

# Loops in Java

The Java for loop is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

There are three types of for loops in Java.



* Simple for Loop
* For-each or Enhanced for Loop
* Labeled for Loop

## Java Simple for Loop

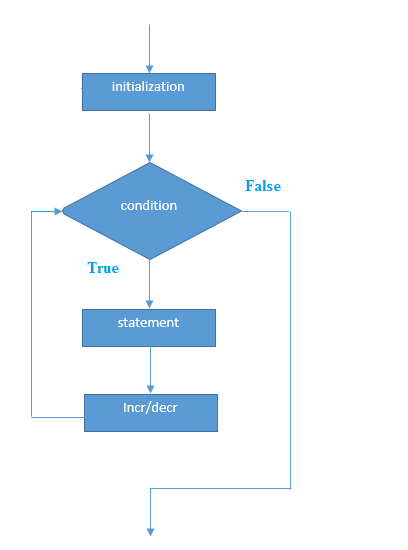
A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
4. **Statement**: The statement of the loop is executed each time until the second condition is false.

**Syntax:**

1. **for**(initialization; condition; increment/decrement){
2. //statement or code to be executed
3. }

**Flowchart:**



**Example:**

**ForExample.java**

1. //Java Program to demonstrate the example of for loop
2. //which prints table of 1
3. **public** **class** ForExample {
4. **public** **static** **void** main(String[] args) {
5. //Code of Java for loop
6. **for**(**int** i=1;i<=10;i++){
7. System.out.println(i);
8. }
9. }
10. }

**Output:**

1

2

3

4

5

6

7

8

9

10

## Java Nested for Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

**Example:**

**NestedForExample.java**

1. **public** **class** NestedForExample {
2. **public** **static** **void** main(String[] args) {
3. //loop of i
4. **for**(**int** i=1;i<=3;i++){
5. //loop of j
6. **for**(**int** j=1;j<=3;j++){
7. System.out.println(i+" "+j);
8. }//end of i
9. }//end of j
10. }
11. }

**Output:**

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

**Pyramid Example 1:**

**PyramidExample.java**

1. **public** **class** PyramidExample {
2. **public** **static** **void** main(String[] args) {
3. **for**(**int** i=1;i<=5;i++){
4. **for**(**int** j=1;j<=i;j++){
5. System.out.print("\* ");
6. }
7. System.out.println();//new line
8. }
9. }
10. }

**Output:**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Pyramid Example 2:**

**PyramidExample2.java**

1. **public** **class** PyramidExample2 {
2. **public** **static** **void** main(String[] args) {
3. **int** term=6;
4. **for**(**int** i=1;i<=term;i++){
5. **for**(**int** j=term;j>=i;j--){
6. System.out.print("\* ");
7. }
8. System.out.println();//new line
9. }
10. }
11. }

**Output:**

\* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

## Java for-each Loop

The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on the basis of elements and not the index. It returns element one by one in the defined variable.

**Syntax:**

1. **for**(data\_type variable : array\_name){
2. //code to be executed
3. }

**Example:**

**ForEachExample.java**

1. //Java For-each loop example which prints the
2. //elements of the array
3. **public** **class** ForEachExample {
4. **public** **static** **void** main(String[] args) {
5. //Declaring an array
6. **int** arr[]={12,23,44,56,78};
7. //Printing array using for-each loop
8. **for**(**int** i:arr){
9. System.out.println(i);
10. }
11. }
12. }

**Output:**

12

23

44

56

78

## Java Labeled For Loop

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break/continue specific for loop.

#### **Note: The break and continue keywords breaks or continues the innermost for loop respectively.**

**Syntax:**

1. labelname:
2. **for**(initialization; condition; increment/decrement){
3. //code to be executed
4. }

**Example:**

**LabeledForExample.java**

1. //A Java program to demonstrate the use of labeled for loop
2. **public** **class** LabeledForExample {
3. **public** **static** **void** main(String[] args) {
4. //Using Label for outer and for loop
5. aa:
6. **for**(**int** i=1;i<=3;i++){
7. bb:
8. **for**(**int** j=1;j<=3;j++){
9. **if**(i==2&&j==2){
10. **break** aa;
11. }
12. System.out.println(i+" "+j);
13. }
14. }
15. }
16. }

**Output:**

1 1

1 2

1 3

2 1

If you use **break bb;**, it will break inner loop only which is the default behaviour of any loop.

**LabeledForExample2.java**

1. **public** **class** LabeledForExample2 {
2. **public** **static** **void** main(String[] args) {
3. aa:
4. **for**(**int** i=1;i<=3;i++){
5. bb:
6. **for**(**int** j=1;j<=3;j++){
7. **if**(i==2&&j==2){
8. **break** bb;
9. }
10. System.out.println(i+" "+j);
11. }
12. }
13. }
14. }

**Output:**

1 1

1 2

1 3

2 1

3 1

3 2

3 3

## Java Infinitive for Loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

**Syntax:**

1. **for**(;;){
2. //code to be executed
3. }

**Example:**

**ForExample.java**

1. //Java program to demonstrate the use of infinite for loop
2. //which prints an statement
3. **public** **class** ForExample {
4. **public** **static** **void** main(String[] args) {
5. //Using no condition in for loop
6. **for**(;;){
7. System.out.println("infinitive loop");
8. }
9. }
10. }

**Output:**

infinitive loop

infinitive loop

infinitive loop

infinitive loop

infinitive loop

ctrl+c

Now, you need to press ctrl+c to exit from the program.

## Java for Loop vs while Loop vs do-while Loop

|  |  |  |  |
| --- | --- | --- | --- |
| **Comparison** | **for loop** | **while loop** | **do-while loop** |
| Introduction | The Java for loop is a control flow statement that iterates a part of the [programs](https://www.javatpoint.com/java-programs) multiple times. | The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition. | The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition. |
| When to use | If the number of iteration is fixed, it is recommended to use for loop. | If the number of iteration is not fixed, it is recommended to use while loop. | If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop. |
| Syntax | for(init;condition;incr/decr){ // code to be executed } | while(condition){ //code to be executed } | do{ //code to be executed }while(condition); |
| Example | //for loop for(int i=1;i<=10;i++){ System.out.println(i); } | //while loop int i=1; while(i<=10){ System.out.println(i); i++; } | //do-while loop int i=1; do{ System.out.println(i); i++; }while(i<=10); |
| Syntax for infinitive loop | for(;;){ //code to be executed } | while(true){ //code to be executed } | do{ //code to be executed }while(true); |

Java While Loop

The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.

**Syntax:**

1. **while** (condition){
2. //code to be executed
3. Increment / decrement statement
4. }

**The different parts of do-while loop:**

1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. When the condition becomes false, we exit the while loop.

**Example**:

i <=100

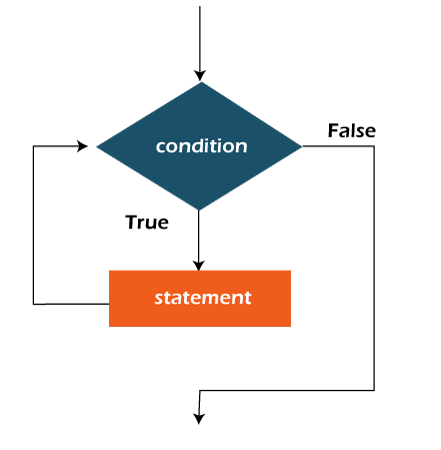
2. Update expression: Every time the loop body is executed, this expression increments or decrements loop variable.

**Example:**

**i++;**

**Flowchart of Java While Loop**

Here, the important thing about while loop is that, sometimes it may not even execute. If the condition to be tested results into false, the loop body is skipped and first statement after the while loop will be executed.



**Example:**

In the below example, we print integer values from 1 to 10. Unlike the for loop, we separately need to initialize and increment the variable used in the condition (here, i). Otherwise, the loop will execute infinitely.

**WhileExample.java**

1. **public** **class** WhileExample {
2. **public** **static** **void** main(String[] args) {
3. **int** i=1;
4. **while**(i<=10){
5. System.out.println(i);
6. i++;
7. }
8. }
9. }

**Output:**

1

2

3

4

5

6

7

8

9

10

Java Infinitive While Loop

If you pass **true** in the while loop, it will be infinitive while loop.

**Syntax:**

1. **while**(**true**){
2. //code to be executed
3. }

**Example:**

**WhileExample2.java**

1. **public** **class** WhileExample2 {
2. **public** **static** **void** main(String[] args) {
3. // setting the infinite while loop by passing true to the condition
4. **while**(**true**){
5. System.out.println("infinitive while loop");
6. }
7. }
8. }

**Output:**

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

ctrl+c

In the above code, we need to enter Ctrl + C command to terminate the infinite loop.

# Java do-while Loop

The Java do-while loop is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

Java do-while loop is called an **exit control loop**. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java do-while loop is executed at least once because condition is checked after loop body.

**Syntax:**

1. **do**{
2. //code to be executed / loop body
3. //update statement
4. }**while** (condition);

**The different parts of do-while loop:**

1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. As soon as the condition becomes false, loop breaks automatically.

**Example:**

**i <=100**

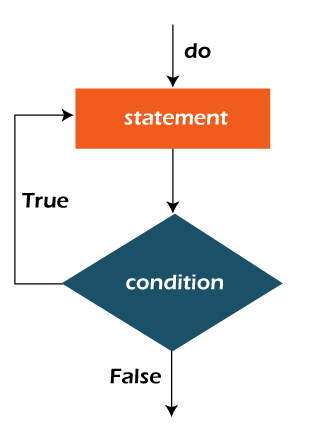
2. Update expression: Every time the loop body is executed, the this expression increments or decrements loop variable.

**Example:**

**i++;**

#### **Note: The do block is executed at least once, even if the condition is false.**

**Flowchart of do-while loop:**



**Example:**

In the below example, we print integer values from 1 to 10. Unlike the for loop, we separately need to initialize and increment the variable used in the condition (here, i). Otherwise, the loop will execute infinitely.

**DoWhileExample.java**

1. **public** **class** DoWhileExample {
2. **public** **static** **void** main(String[] args) {
3. **int** i=1;
4. **do**{
5. System.out.println(i);
6. i++;
7. }**while**(i<=10);
8. }
9. }

**Output:**

1

2

3

4

5

6

7

8

9

10

## Java Infinitive do-while Loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

**Syntax:**

1. **do**{
2. //code to be executed
3. }**while**(**true**);

**Example:**

**DoWhileExample2.java**

1. **public** **class** DoWhileExample2 {
2. **public** **static** **void** main(String[] args) {
3. **do**{
4. System.out.println("infinitive do while loop");
5. }**while**(**true**);
6. }
7. }

**Output:**

infinitive do while loop

infinitive do while loop

infinitive do while loop

ctrl+c

In the above code, we need to enter Ctrl + C command to terminate the infinite loop.

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

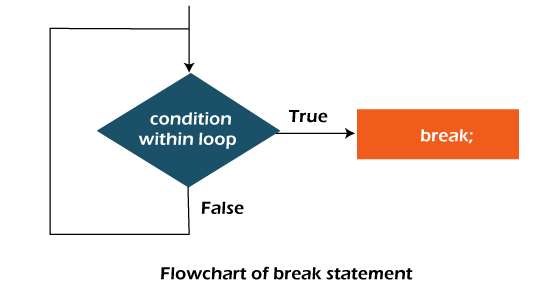
The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

**Syntax:**

1. jump-statement;
2. **break**;

**Flowchart of Break Statement**



Java Break Statement with Loop

**Example:**

**BreakExample.java**

1. //Java Program to demonstrate the use of break statement
2. //inside the for loop.
3. **public** **class** BreakExample {
4. **public** **static** **void** main(String[] args) {
5. //using for loop
6. **for**(**int** i=1;i<=10;i++){
7. **if**(i==5){
8. //breaking the loop
9. **break**;
10. }
11. System.out.println(i);
12. }
13. }
14. }

**Output:**

1

2

3

4

Java Break Statement with Inner Loop

It breaks inner loop only if you use break statement inside the inner loop.

**Example:**

**BreakExample2.java**

1. //Java Program to illustrate the use of break statement
2. //inside an inner loop
3. **public** **class** BreakExample2 {
4. **public** **static** **void** main(String[] args) {
5. //outer loop
6. **for**(**int** i=1;i<=3;i++){
7. //inner loop
8. **for**(**int** j=1;j<=3;j++){
9. **if**(i==2&&j==2){
10. //using break statement inside the inner loop
11. **break**;
12. }
13. System.out.println(i+" "+j);
14. }
15. }
16. }
17. }

**Output:**

1 1

1 2

1 3

2 1

3 1

3 2

3 3

Java Break Statement with Labeled For Loop

We can use break statement with a label. The feature is introduced since JDK 1.5. So, we can break any loop in Java now whether it is outer or inner loop.

**Example:**

**BreakExample3.java**

1. //Java Program to illustrate the use of continue statement
2. //with label inside an inner loop to break outer loop
3. **public** **class** BreakExample3 {
4. **public** **static** **void** main(String[] args) {
5. aa:
6. **for**(**int** i=1;i<=3;i++){
7. bb:
8. **for**(**int** j=1;j<=3;j++){
9. **if**(i==2&&j==2){
10. //using break statement with label
11. **break** aa;
12. }
13. System.out.println(i+" "+j);
14. }
15. }
16. }
17. }

**Output:**

1 1

1 2

1 3

2 1

Java Break Statement in while loop

**Example:**

**BreakWhileExample.java**

1. //Java Program to demonstrate the use of break statement
2. //inside the while loop.
3. **public** **class** BreakWhileExample {
4. **public** **static** **void** main(String[] args) {
5. //while loop
6. **int** i=1;
7. **while**(i<=10){
8. **if**(i==5){
9. //using break statement
10. i++;
11. **break**;//it will break the loop
12. }
13. System.out.println(i);
14. i++;
15. }
16. }
17. }

**Output:**

1

2

3

4

Java Break Statement in do-while loop

**Example:**

**BreakDoWhileExample.java**

1. //Java Program to demonstrate the use of break statement
2. //inside the Java do-while loop.
3. **public** **class** BreakDoWhileExample {
4. **public** **static** **void** main(String[] args) {
5. //declaring variable
6. **int** i=1;
7. //do-while loop
8. **do**{
9. **if**(i==5){
10. //using break statement
11. i++;
12. **break**;//it will break the loop
13. }
14. System.out.println(i);
15. i++;
16. }**while**(i<=10);
17. }
18. }

**Output:**

1

2

3

4

Java Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

**Syntax:**

1. jump-statement;
2. **continue**;

Java Continue Statement Example

**ContinueExample.java**

1. //Java Program to demonstrate the use of continue statement
2. //inside the for loop.
3. **public** **class** ContinueExample {
4. **public** **static** **void** main(String[] args) {
5. //for loop
6. **for**(**int** i=1;i<=10;i++){
7. **if**(i==5){
8. //using continue statement
9. **continue**;//it will skip the rest statement
10. }
11. System.out.println(i);
12. }
13. }
14. }

**Output:**

1

2

3

4

6

7

8

9

10

As you can see in the above output, 5 is not printed on the console. It is because the loop is continued when it reaches to 5.

Java Continue Statement with Inner Loop

It continues inner loop only if you use the continue statement inside the inner loop.

**ContinueExample2.java**

1. //Java Program to illustrate the use of continue statement
2. //inside an inner loop
3. **public** **class** ContinueExample2 {
4. **public** **static** **void** main(String[] args) {
5. //outer loop
6. **for**(**int** i=1;i<=3;i++){
7. //inner loop
8. **for**(**int** j=1;j<=3;j++){
9. **if**(i==2&&j==2){
10. //using continue statement inside inner loop
11. **continue**;
12. }
13. System.out.println(i+" "+j);
14. }
15. }
16. }
17. }

**Output:**

1 1

1 2

1 3

2 1

2 3

3 1

3 2

3 3

Java Continue Statement with Labelled For Loop

We can use continue statement with a label. This feature is introduced since JDK 1.5. So, we can continue any loop in Java now whether it is outer loop or inner.

**Example:**

**ContinueExample3.java**

1. //Java Program to illustrate the use of continue statement
2. //with label inside an inner loop to continue outer loop
3. **public** **class** ContinueExample3 {
4. **public** **static** **void** main(String[] args) {
5. aa:
6. **for**(**int** i=1;i<=3;i++){
7. bb:
8. **for**(**int** j=1;j<=3;j++){
9. **if**(i==2&&j==2){
10. //using continue statement with label
11. **continue** aa;
12. }
13. System.out.println(i+" "+j);
14. }
15. }
16. }
17. }

**Output:**

1 1

1 2

1 3

2 1

3 1

3 2

3 3

Java Continue Statement in while loop

**ContinueWhileExample.java**

1. //Java Program to demonstrate the use of continue statement
2. //inside the while loop.
3. **public** **class** ContinueWhileExample {
4. **public** **static** **void** main(String[] args) {
5. //while loop
6. **int** i=1;
7. **while**(i<=10){
8. **if**(i==5){
9. //using continue statement
10. i++;
11. **continue**;//it will skip the rest statement
12. }
13. System.out.println(i);
14. i++;
15. }
16. }
17. }

**Output:**

1

2

3

4

6

7

8

9

10

Java Continue Statement in do-while Loop

**ContinueDoWhileExample.java**

1. //Java Program to demonstrate the use of continue statement
2. //inside the Java do-while loop.
3. **public** **class** ContinueDoWhileExample {
4. **public** **static** **void** main(String[] args) {
5. //declaring variable
6. **int** i=1;
7. //do-while loop
8. **do**{
9. **if**(i==5){
10. //using continue statement
11. i++;
12. **continue**;//it will skip the rest statement
13. }
14. System.out.println(i);
15. i++;
16. }**while**(i<=10);
17. }
18. }

**Output:**

1

2

3

4

6

7

8

9

10