

Java Branching Statements

Description

Java provides three branching statements break, continue and return. The break and continue in Java are two essential keyword beginners needs to familiar while using loops (for loop, while loop and do while loop). break statement in java is used to break the loop and transfers control to the line immediate outside of loop while continue is used to escape current execution (iteration) and transfers control back to the start of the loop. Both break and continue allow the programmer to create sophisticated algorithm and looping constructs.

In this java tutorial, we will see the example of break and continue statement in Java and some important points related to breaking the loop using label and break statement. break keyword can also be used inside switch statement to break current choice and if not used it can cause fall-through on switch. Both the break statement and the continue statement can be unlabeled or labeled. Although it's far more common to use a break and continue unlabeled.

Let's understand unlabeled continue and break statement using java program. Below program will do the addition of all even numbers of array till it encounters 0 or negative number from an array.

```
public class BreakContinueDemo {  
    public static void main(String[] args) {  
        int [] numbers = {10,23,19,34,54,23,76,39,65,24,8,0,12,55};  
        int sum =0;  
        for(int i=0; i< numbers.length; i++){  
            if(numbers[i]<=0){  
                System.out.println("Break statement coming  
because number is = "+ numbers[i]);  
                break;  
            }else if (numbers[i]%2 != 0){  
                System.out.println("Odd number found in array,  
ignoring number " + numbers[i]);  
                continue;  
            }  
            sum += numbers[i];  
        }  
        System.out.println("Sum of even numbers = " + sum);  
    }  
}
```

```

        continue;

    }else

        sum = sum + numbers[i];

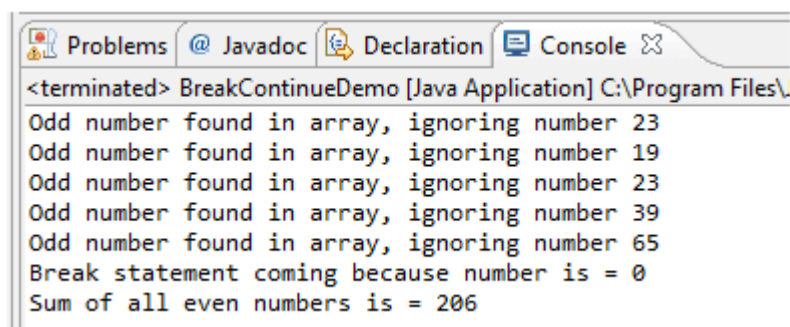
    }

    System.out.println("Sum of all even numbers is = " + sum);

}
}

```

Output:



```

<terminated> BreakContinueDemo [Java Application] C:\Program Files\
Odd number found in array, ignoring number 23
Odd number found in array, ignoring number 19
Odd number found in array, ignoring number 23
Odd number found in array, ignoring number 39
Odd number found in array, ignoring number 65
Break statement coming because number is = 0
Sum of all even numbers is = 206

```

Labeled Statements

Although many statements in a Java program can be labeled, it's most common to use labels with loop statements like for or while, in conjunction with break and continue statements. A label statement must be placed just before the statement being labeled, and it consists of a valid identifier that ends with a colon (:).

You need to understand the difference between labeled and unlabeled break and continue. The labeled varieties are needed only in situations where you have nested loop and need to indicate which of the nested loops you want to break from, or from which of the nested loops you want to continue with the next iteration. A break statement will exit out of the labeled loop, as opposed to the innermost loop, if the break keyword is combined with a label. Here is an example program that uses continue to print a triangular multiplication table for 0 through 9.

```

public class LabeledBreakContinueDemo {
    public static void main(String[] args) {
        int breaklimit = 9;
        outer: for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {

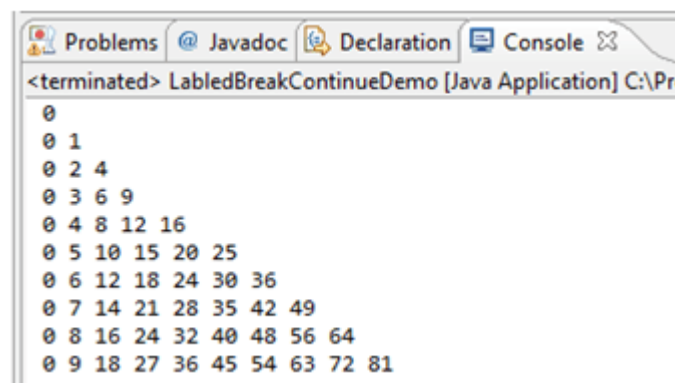
```

```

        if (j > i) {
            System.out.println();
            continue outer;
        }
        System.out.print(" " + (i * j));
    }
    if(i==breaklimit){
        break outer;
    }
}
System.out.println();
}

```

Output:



```

<terminated> LabeledBreakContinueDemo [Java Application] C:\Pr
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81

```

The return statement

The last control statement is return. The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method. As such, it is categorized as a jump statement. At any time in a method, the return statement can be used to cause execution to branch back to the caller of the method. Thus, the return statement immediately terminates the method in which it is executed. The following example illustrates this point. In below program main () is calling method and checkEven() is called the method. Execution of checkEven() method ends when return statement encountered.

```

public class ReturnDemo {

    public static void main(String[] args) {

        for (int k =25; k< 31; k++){

            new ReturnDemo().checkEven(k);

        }
    }
}

```

```

    }

    public boolean checkEven(int a){

        if (a%2 == 0){

            System.out.println(a + " is even number");

            return true;

        }

        System.out.println(a + " is odd number");

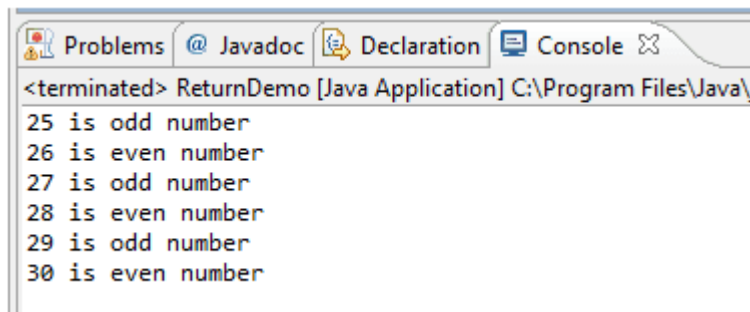
        return false;

    }

}

```

Output:



```

<terminated> ReturnDemo [Java Application] C:\Program Files\Java\
25 is odd number
26 is even number
27 is odd number
28 is even number
29 is odd number
30 is even number

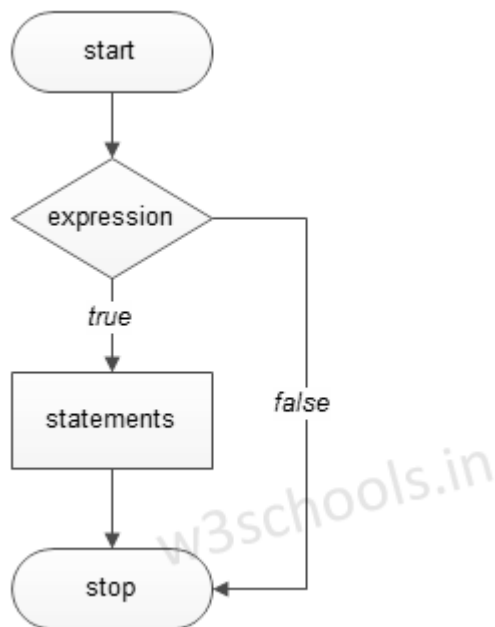
```

Java decision-making

Java decision-making statements allow you to make a decision, based upon the result of a condition.

All the programs in Java have set of statements, which are executed sequentially in the order in which they appear. It happens when jumping of statements or repetition of certain calculations is not necessary. However, there may arise some situations where programmers have to change the order of execution of statements based on certain conditions which involve kind of decision-making statements. In this chapter, you will learn about how the control flow statements work.

The flowchart of Decision-making technique in Java can be expressed as:



Java has such decision-making capabilities within its program by the use of following the decision making statements:

Decision Making Statements in Java

- if Statement
 - if statement
 - if-else statement
 - else-if statement
- Conditional Operator
- switch statement

- if statement

If statements in Java is used to control the program flow based on some condition, it's used to execute some statement code block if the expression evaluated to true; otherwise, it will get skipped. This statement is the simplest way to modify the control flow of the program.

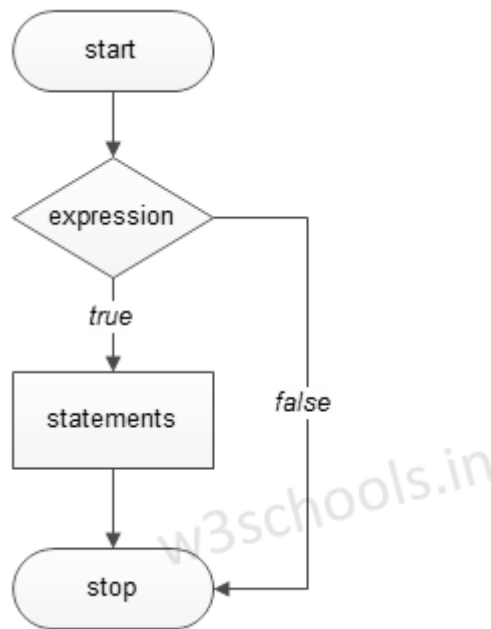
The basic format of if statement is:

Syntax:

```
if(test_expression)
{
    statement 1;
    statement 2;
    ...
}
```

'Statement n' can be a statement or a set of statements, and if the test expression evaluated to *true*, the statement block will get executed, or it will get skipped.

Figure - Flowchart of if Statement:



Example of a Java Program to Demonstrate If Statements:

Example:

```
public class Sample{  
  
    public static void main(String args[]){  
        int a=20, b=30;  
  
        if(b>a)  
            System.out.println("b is greater");  
        }  
    }  
}
```

Program Output:

```
run:  
b is greater  
BUILD SUCCESSFUL (total time: 0 seconds)
```

If else statements

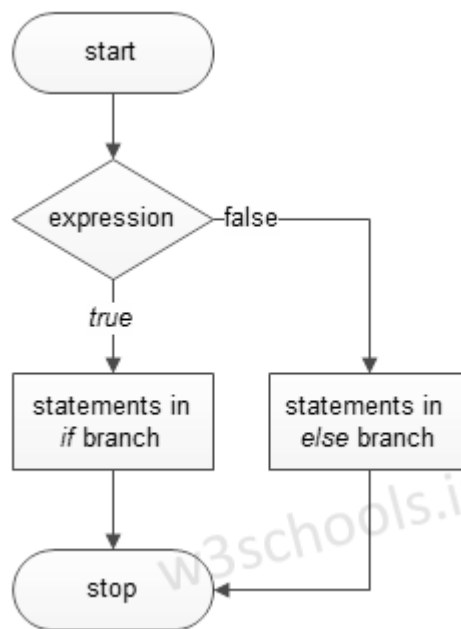
If else statements in Java is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement code block.

The basic format of if else statement is:

Syntax:

```
if(test_expression)
{
    //execute your code
}
else
{
    //execute your code
}
```

Figure - Flowchart of if-else Statement:



Example of a Java Program to Demonstrate If else Statements:

Example:

```
public class Sample {

    public static void main(String args[]) {
        int a = 80, b = 30;

        if (b > a) {
            System.out.println("b is greater");
        }
    }
}
```

```

    } else {
        System.out.println("a is greater");
    }
}

}

```

Program Output:

```

run:
a is greater
BUILD SUCCESSFUL (total time: 0 seconds)

```

else if statements:

else if statements in Java is like another if condition, it's used in the program when if statement having multiple decisions.

The basic format of else if statement is:

Syntax:

```

if(test_expression)
{
    //execute your code
}
else if(test_expression n)
{
    //execute your code
}
else
{
    //execute your code
}
}

```

Example of a Java Program to Demonstrate else If Statements:

Example:

```

public class Sample {

    public static void main(String args[]) {
        int a = 30, b = 30;

        if (b > a) {
            System.out.println("b is greater");
        }
    }
}

```



```

        else if(a > b){
            System.out.println("a is greater");
        }
        else {
            System.out.println("Both are equal");
        }
    }
}

```

Program Output:

```

run:
Both are equal
BUILD SUCCESSFUL (total time: 0 seconds)

```

Java switch statement:

Java switch statement is used when you have multiple possibilities for the if statement.

The basic format of the switch statement is:

Syntax:

```

switch(variable)
{
    case 1:
        //execute your code
        break;

    case n:
        //execute your code
        break;

    default:
        //execute your code
        break;

}

```

After the end of each block it is necessary to insert a *break statement* because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each case onwards after matching the case block.

Example of a Java Program to Demonstrate Switch Statements:

Example:

```
public class Sample {

    public static void main(String args[]) {
        int a = 5;

        switch (a) {
            case 1:
                System.out.println("You chose One");
                break;

            case 2:
                System.out.println("You chose Two");
                break;

            case 3:
                System.out.println("You chose Three");
                break;

            case 4:
                System.out.println("You chose Four");
                break;

            case 5:
                System.out.println("You chose Five");
                break;

            default:
                System.out.println("Invalid Choice. Enter a no between 1
and 5");
                break;
        }
    }

}
```

Program Output:

```
run:
You chose Five
BUILD SUCCESSFUL (total time: 0 seconds)
```

When none of the cases is evaluated to *true*, the default case will be executed, and *break statement* is not required for default statement.