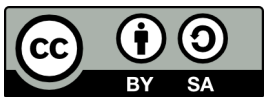
Control Flow Statements

Java Fundamentals

Libre Education



Control Flow Statements

Control Flow Statements check a condition and then execute code based on the result of the condition.

if Statement

The if Statement, commonly called the if-then statement, checks a condition and if the condition returns true, a block of code is executed.

Syntax

```
if(condition){
  code that executes if the condition is true
}
```

Example

```
int i = 3;
if(i == 3){
   System.out.println("i is equal to 3");
}
```

Note

If the code to execute is only one statement the '{ }' can be omitted

if-else Statement

The if-else statement also allows code to be executed if the condition returns false. If the statement returns false the else clause is executed.

Syntax

```
if(condition){
  code that executes if the condition is true
}else{
  code that executes if the condition is false
}
```

Example

```
int i = 5;
if(i <= 3){
   System.out.println("i is smaller than or equal to 3");
}else{
   System.out.println("i is bigger than 3");
}
```

Combining if-else Statements

Multiple if-else statements can be combined to check multiple conditions.

Syntax

```
if(condition){
  code that executes if this condition is true
}else if(condition)
  code that executes if this condition is true
}else if(condition)
  code that executes if this condition is true
}else{
  code that executes if all the conditions are false
}
```

Example

```
int i = 3;
if(i == 1){
   System.out.println("i is equal to 1");
}else if(i == 2){
   System.out.println("i is equal to 2");
}else if(i == 3){
   System.out.println("i is equal to 3");
}else if(i == 4){
   System.out.println("i is equal to 4");
}else {
   System.out.println("i is not equal to any of these numbers");
}
```

switch Statement

A switch statement allows you to easily check multiple return values for a variable or expression, much like combining multiple if-else statements. The switch checks for equality between the variable or expression and the possible return value or 'case'. It also has a 'default' clause for if none of the values are equal to the variable/expression.

Syntax

```
switch (variable/expression){
   case value:
       code that executes if this value is equal to the variable/expression
       break;
   case value:
       code that executes if this value is equal to the variable/expression
       break;
   case value:
       code that executes if this value is equal to the variable/expression
       break;
   default:
       code that executes if none of the previous value are equal to the
     variable/expression
       break;
}
```

Example

```
int i = 2;
switch (i){
   case 1:
       System.out.println("i is equal to 1");
       break;
   case 2:
       System.out.println("i is equal to 2");
       break;
   case 3:
       System.out.println("i is equal to 3");
       break;
   case 4:
       System.out.println("i is equal to 4");
       break;
   default:
       System.out.println("i is not equal to any of these numbers");
       break;
}
```

Note

The expression must evaluate to byte, short, char, int or String.

Multiple Cases

If the same code needs to be executed for multiple cases, cases can be combined into the same statement.

Syntax

```
switch (condition/expression){
   case value:case value:
       code that executes if one or more of the values are equal to the
     variable/expression
       break;
   case value:case value:case value:
       code that executes if one or more of the values are equal to the
     variable/expression
       break;
}
```

Example

```
int i = 5;
switch (i){
   case 1:case 2:
       System.out.println("i is equal to 1 or 2");
       break;
   case 3:case 4:case 5:
       System.out.println("i is equal to 3, 4 or 5");
       break;
   default:
       System.out.println("i is not equal to any of these numbers");
       break;
}
```

while Statement

The while statement will continue to execute a block of code while its condition returns true, so it can be considered as a loop. It will stop executing the code block when its condition returns false.

Syntax

```
while (condition){
   code that executes while the condition is true
}
```

Example

```
int i = 1;
while (i <= 4){
    System.out.println("Currently in loop " + i);
    i++;
}
/* Output:
Currently in loop 1
Currently in loop 2
Currently in loop 3
Currently in loop 4
*/
```

do-while Statement

The do-while statement is very similar to the while statement, the difference is that the condition is checked after the code is executed, meaning the code gets executed at least once.

Syntax

```
do{
    code that executes while the condition is true
}while (condition);
```

Example

```
int i = 1;
do{
    System.out.println("Currently in loop " + i);
    i++;
}while (i <= 4);
/* Output:
Currently in loop 1
Currently in loop 2
Currently in loop 3
Currently in loop 4
*/
```

for Statement

The for statement provides a more efficient and structured way of looping. It allows you to initialize the variable that will be used to control looping, the condition and the operation that will be applied to the variable after each loop.

Syntax

```
for(initialization; condition; operation){
   code that executes while the condition is true
}
```

Example

```
for(int i = 1; i <= 4; i++){
   System.out.println("Currently in loop " + i);
}
/* Output:
Currently in loop 1
Currently in loop 2
Currently in loop 3
Currently in loop 4
*/
```

for each Statement

The for each statement, also called the optimized for statement, is a for statement that is enhanced to loop through arrays and Collections. It allows you to specify the element found in the array/collection and the array/collection that will be looped through.

Syntax

```
for(data_type variable : array/collection){
   code that executes while the condition is true
}
```

Example

```
int arr[] = {1, 2, 3, 4};
for(int i : arr){
   System.out.println("Currently at number " + i);
}
/* Output:
Currently at number 1
Currently at number 2
Currently at number 3
Currently at number 4
*/
```

Loop Control Statements

break Statement

When the break statement is called inside a loop, the loop is terminated and the program continues running directly after the loop. It is also used to terminate a case in the switch statement.

Example

```
for(int i = 1; i <= 100; i++){
    System.out.println("Currently in loop " + i);
    if(i == 4){
        break;  // Causes the loop to end after 4 iterations, instead of 100
    }
}
/* Output:
Currently in loop 1
Currently in loop 2
Currently in loop 3
Currently in loop 4
*/
```

continue Statement

The continue statement causes the loop to jump to the next step or iteration in the loop.

Example

```
for(int i = 1; i <= 4; i++){
    if(i == 3){
        continue;
    }
    System.out.println("Currently in loop " + i);
}
/* Output:
Currently in loop 1
Currently in loop 2
Currently in loop 4
*/
```

Other Resources

1. The Java Tutorials - (docs.oracle.com/javase/tutorial/java/nutsandbolts/ if.html)
2. TutorialsPoint - Decision Making (www.tutorialspoint.com/java/java_ decision_making.htm)
3. TutorialsPoint - Loop Control (www.tutorialspoint.com/java/java_loop_ control.htm)
4. ZetCode (zetcode.com/lang/java/flow)
5. Udemy Blog (blog.udemy.com/for-each-loop-java)
6. Java For Dummies - if Statements (www.dummies.com/how-to/content/how-to- use-if-statements-in-java.html)
7. Java For Dummies - switch Statements (www.dummies.com/how-to/content/ switch-statements-in-java.html)