# Java If ... Else

## Java Conditions and If Statements

Java supports the usual logical conditions from mathematics:

* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b
* Equal to a == b
* Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

## The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

### Syntax

if (*condition*) {

*// block of code to be executed if the condition is true*}

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

### Example

if (20 > 18) {

System.out.println("20 is greater than 18");}

We can also test variables:

### Example

int x = 20;int y = 18;if (x > y) {

System.out.println("x is greater than y");}

#### Example explained

In the example above we use two variables, ****x**** and ****y****, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

## The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

### Syntax

if (*condition*) {

*// block of code to be executed if the condition is true*} else {

*// block of code to be executed if the condition is false*}

### Example

int time = 20;if (time < 18) {

System.out.println("Good day.");} else {

System.out.println("Good evening.");}// Outputs "Good evening."

#### Example explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

## The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

### Syntax

if (*condition1*) {

*// block of code to be executed if condition1 is true*} else if (*condition2*) {

*// block of code to be executed if the condition1 is false and condition2 is true*} else {

*// block of code to be executed if the condition1 is false and condition2 is false*}

### Example

int time = 22;if (time < 10) {

System.out.println("Good morning.");} else if (time < 20) {

System.out.println("Good day.");} else {

System.out.println("Good evening.");}// Outputs "Good evening."

#### Example explained

In the example above, time (22) is greater than 10, so the ****first condition**** is false. The next condition, in the else if statement, is also false, so we move on to the else condition since ****condition1**** and ****condition2**** is both false - and print to the screen "Good evening".

However, if the time was 14, our program would print "Good day."

## Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the ****ternary operator**** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

### Syntax

*variable* = (*condition*) ? *expressionTrue* :  *expressionFalse*;

Instead of writing:

### Example

int time = 20;if (time < 18) {

System.out.println("Good day.");} else {

System.out.println("Good evening.");}

You can simply write:

### Example

int time = 20;

String result = (time < 18) ? "Good day." : "Good evening.";

System.out.println(result);

# Java Switch

## Java Switch Statements

Use the switch statement to select one of many code blocks to be executed.

### Syntax

switch(*expression*) {

case x:

*// code block*

break;

case y:

*// code block*

break;

default:

*// code block*}

## Java Switch Statements

Use the switch statement to select one of many code blocks to be executed.

### Syntax

switch(*expression*) {

case x:

*// code block*

break;

case y:

*// code block*

break;

default:

*// code block*

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.
* The break and default keywords are optional, and will be described later in this chapter

The example below uses the weekday number to calculate the weekday name:

int day = 4;switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

case 4:

System.out.println("Thursday");

break;

case 5:

System.out.println("Friday");

break;

case 6:

System.out.println("Saturday");

break;

case 7:

System.out.println("Sunday");

break;}// Outputs "Thursday" (day 4)

## The break Keyword

When Java reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

## The default Keyword

The default keyword specifies some code to run if there is no case match:

### Example

int day = 4;switch (day) {

case 6:

System.out.println("Today is Saturday");

break;

case 7:

System.out.println("Today is Sunday");

break;

default:

System.out.println("Looking forward to the Weekend");}// Outputs "Looking forward to the Weekend"

# Java While Loop

## Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

## Java While Loop

The while loop loops through a block of code as long as a specified condition is true:

### Syntax

while (*condition*) { *// code block to be executed*}

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

### Example

int i = 0;while (i < 5) {

System.out.println(i);

i++;}

****Note:**** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

## The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### Syntax

do { *// code block to be executed*}while (*condition*);

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

### Example

int i = 0;  
do {

System.out.println(i);

i++;}

#### Example explained

Statement 1 sets a variable before the loop starts (int i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

## Another Example

This example will only print even values between 0 and 10:

### Example

for (int i = 0; i <= 10; i = i + 2) {

System.out.println(i);}

## For-Each Loop

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an ****array****:

### Syntax

for (*type* *variableName* : *arrayName*) {

*// code block to be executed*}

The following example outputs all elements in the ****cars**** array, using a "**for-each**" loop:

### Example

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);}