**Java Conditions**

**Java If… else statement**

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.

**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

}

**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. We can add multiple else if condition here

} else {

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

}

**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:

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

Instead of writing:

int time = 20;

if (time < 18) {

System.out.println("Good day.");

} else {

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

}

You can simply write:

int time = 20;

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

System.out.println(result);

Here, if the value time is less than 18 then print Good evening.

Otherwise print Good day

**Java switch statement**

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

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.

**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.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

**The default keyword**

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

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"

Note that if the default statement is used as the last statement in a switch block, it does not need a break.

**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.

**While loop**

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

while (*condition*) {

*// code block to be executed*

}

**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.

do {

*// code block to be executed*

}

while (*condition*);

**Java for loop**

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

for (*statement 1*; *statement 2*; *statement 3*) {

*// code block to be executed*

}

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

**For Each loop**

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

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

*// code block to be executed*

}

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

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

for (String i : cars) {

System.out.println(i);

}

**Break**

the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

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

if (i == 4) {

break;

}

System.out.println(i);

}

**Continue**

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

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

if (i == 4) {

continue;

}

System.out.print(i);

}

Here, when the i==4 it will skip the iteration and jump to next

Hence, apart from 4 it will start printing 5

Output:

012356789