Java Strings

Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

Example

Create a variable of type String and assign it a value:

String greeting = "Hello";

String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the length() method:

Example

String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of the txt string is: " + txt.length());

More String Methods

There are many string methods available, for example toUpperCase() and toLowerCase():

Example

String txt = "Hello World";

System.out.println(txt.toUpperCase()); // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase()); // Outputs "hello world"

Finding a Character in a String

The indexOf() method returns the **index** (the position) of the first occurrence of a specified text in a string (including whitespace):

Example

String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7

Java counts positions from zero.  
0 is the first position in a string, 1 is the second, 2 is the third ...

String Concatenation

The + operator can be used between strings to combine them. This is called **concatenation**:

Example

String firstName = "John";

String lastName = "Doe";

System.out.println(firstName + " " + lastName);

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.

You can also use the concat() method to concatenate two strings:

Example

String firstName = "John ";

String lastName = "Doe";

System.out.println(firstName.concat(lastName));

Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

String txt = "We are the so-called "Vikings" from the north.";

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

|  |  |  |
| --- | --- | --- |
| Escape character | Result | Description |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

The sequence \"  inserts a double quote in a string:

Example

String txt = "We are the so-called \"Vikings\" from the north.";

The sequence \'  inserts a single quote in a string:

Example

String txt = "It\'s alright.";

The sequence \\  inserts a single backslash in a string:

Example

String txt = "The character \\ is called backslash.";

Six other escape sequences are valid in Java:

|  |  |
| --- | --- |
| Code | Result |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |
| \f | Form Feed |

Adding Numbers and Strings

WARNING!

Java uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

int x = 10;

int y = 20;

int z = x + y; // z will be 30 (an integer/number)

If you add two strings, the result will be a string concatenation:

Example

String x = "10";

String y = "20";

String z = x + y; // z will be 1020 (a String)

If you add a number and a string, the result will be a string concatenation:

Example

String x = "10";

int y = 20;

String z = x + y; // z will be 1020 (a String)

# **Java Math**

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

## Math.max(x,y)

The Math.max(x,y) method can be used to find the highest value of x and y:

### Example

Math.max(5, 10);

## Math.min(x,y)

The Math.min(x,y) method can be used to find the lowest value of of x and y:

### Example

Math.min(5, 10);

## Math.sqrt(x)

The Math.sqrt(x) method returns the square root of x:

### Example

Math.sqrt(64);

## Math.abs(x)

The Math.abs(x) method returns the absolute (positive) value of x:

### Example

Math.abs(-4.7);

## Math.random()

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

### Example

Math.random();

## 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.");

}

### Example

int time = 20;

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

System.out.println(result);