Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line Comments

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

Example

// This is a comment

System.out.println("Hello World");

## Java Multi-line Comments

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

### Example

/\* The code below will print the words Hello World

to the screen, and it is amazing \*/

System.out.println("Hello World");

Java Variables

Variables are containers for storing data values.

In Java, there are different **types** of variables, for example:

* String - stores text, such as "Hello". String values are surrounded by double quotes
* int - stores integers (whole numbers), without decimals, such as 123 or -123
* float - stores floating point numbers, with decimals, such as 19.99 or -19.99
* char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* boolean - stores values with two states: true or false

## Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

### Syntax

*type variableName = value;*

### Example

Create a variable called **name** of type String and assign it the value "**John**":

String name = "John";

System.out.println(name);

### Example

Create a variable called **myNum** of type int and assign it the value **15**:

int myNum = 15;

System.out.println(myNum);

**Example**

Change the value of myNum from 15 to 20:

int myNum = 15;

myNum = 20; // myNum is now 20

System.out.println(myNum);

## Final Variables

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

### Example

final int myNum = 15;

myNum = 20; // will generate an error: cannot assign a value to a final variable

## Other Types

A demonstration of how to declare variables of other types:

### Example

int myNum = 5;

float myFloatNum = 5.99f;

char myLetter = 'D';

boolean myBool = true;

String myText = "Hello";

## Declare Many Variables

To declare more than one variable of the **same type**, you can use a comma-separated list:

### Example

Instead of writing:

int x = 5;

int y = 6;

int z = 50;

System.out.println(x + y + z);

You can simply write:

int x = 5, y = 6, z = 50;

System.out.println(x + y + z);

## One Value to Multiple Variables

You can also assign the **same value** to multiple variables in one line:

### Example

int x, y, z;

x = y = z = 50;

System.out.println(x + y + z);

## Identifiers

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

### Example

// Good

int minutesPerHour = 60;

// OK, but not so easy to understand what **m** actually is

int m = 60;

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter, and cannot contain whitespace
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case-sensitive ("myVar" and "myvar" are different variables)
* Reserved words (like Java keywords, such as int or boolean) cannot be used as names

## Java Data Types

As explained in the previous chapter, a [variable](https://www.w3schools.com/java/java_variables.asp) in Java must be a specified data type:

### Example

int myNum = 5; // Integer (whole number)

float myFloatNum = 5.99f; // Floating point number

char myLetter = 'D'; // Character

boolean myBool = true; // Boolean

String myText = "Hello";

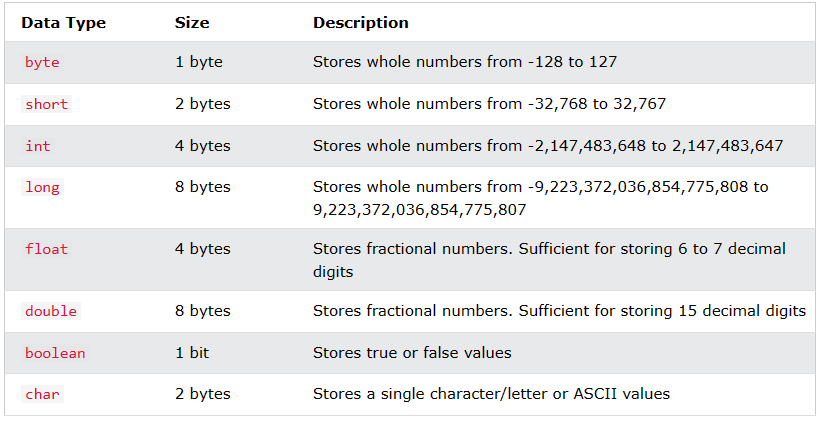
Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp)

## Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:



Non-Primitive Data Types

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

Examples of non-primitive types are [Strings](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp), [Classes,](https://www.w3schools.com/java/java_classes.asp)[Interface](https://www.w3schools.com/java/java_interface.asp), etc.

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* **Widening Casting** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* **Narrowing Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

## Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

### Example

### int x = 100 + 50;

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

### Example

int sum1 = 100 + 50; // 150 (100 + 50)

int sum2 = sum1 + 250; // 400 (150 + 250)

int sum3 = sum2 + sum2; // 800 (400 + 400)

Java divides the operators into the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Bitwise operators

## Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

### 

## Java Assignment Operators

Assignment operators are used to assign values to variables.

### Example

int x = 10;

### The **addition assignment** operator (+=) adds a value to a variable:

### Example

int x = 10;

x += 5;

A list of all assignment operators:

### 

## Java Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

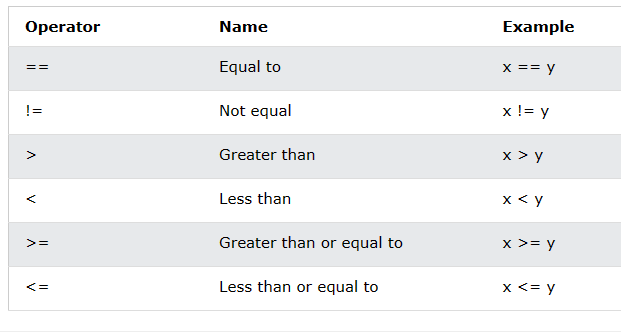
The return value of a comparison is either true or false. These values are known as Boolean values

### Example

int x = 5;

int y = 3;

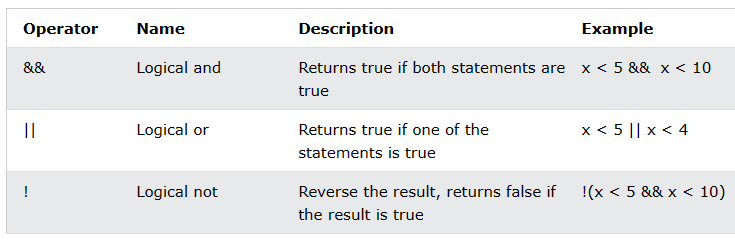
System.out.println(x > y); // returns true, because 5 is higher than 3



## Java Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:



# **Java Strings**