**JAVA**

**Comments in JAVA**

* Single line comments 🡪 Use // to comment out a single line.
* Multi line comments 🡪 Use /\* to start and \*/ end a multi line.

**Variables in JAVA**

* Variables are containers that hold data values.
* They are used to store, manipulate, and display information within program.

variable\_type variable\_name = value;

**Data Types**

* int
* String
* Double
* boolean (true & false)

**String and Char**

|  |  |
| --- | --- |
| **String** | **Char** |
| * The string type is a special type of variable that cannot be changed once it is initialized. | * A char is a single character. |
| * Double quotation (“text”) | * Single quotation (‘A’) |

* int variable can only hold integer values, and a string variables can only hold text.

**Constants**

* A constant is a special type of variable that cannot be changed once it is initialized.
* To declare a constant use the keyword “final” followed by the variable type.

**Naming conventions**

In JAVA it’s important to follow naming conventions to keep your code readable and maintainable.

Here are some KEY RULES;

* Use camelCase (firstName , studentCount)
* Use UPPER\_SNAKE\_CASE (MAX\_VALUE)
* Names can contain letters, digits, underscores, and $(dollar) sign.
* Name must starts with a letter, underscore, $.

**Type Casting Part 1**

In JAVA we can convert integers to doubles, doubles to integers and more.

CASTING

Double to integer

Integer to double

Implicit (automatic)

Explicit (manual)

**Type Casting Part 2**

It is also possible to convert number and booleans to string and vice versa. To convert a value to string we can use the String.valueOf() function;

int number1 = 789;  
double number2 = 789;  
boolean isValid = true;  
String text1 = String.valueOf(number1); // becomes "789"  
String text2 = String.valueOf(number2); // becomes "789.0"  
String text3 = String.valueOf(isValid); // becomes "true"

String to Integer

String numberText = "123";  
int number = Integer.parseInt(numberText);    // becomes 123

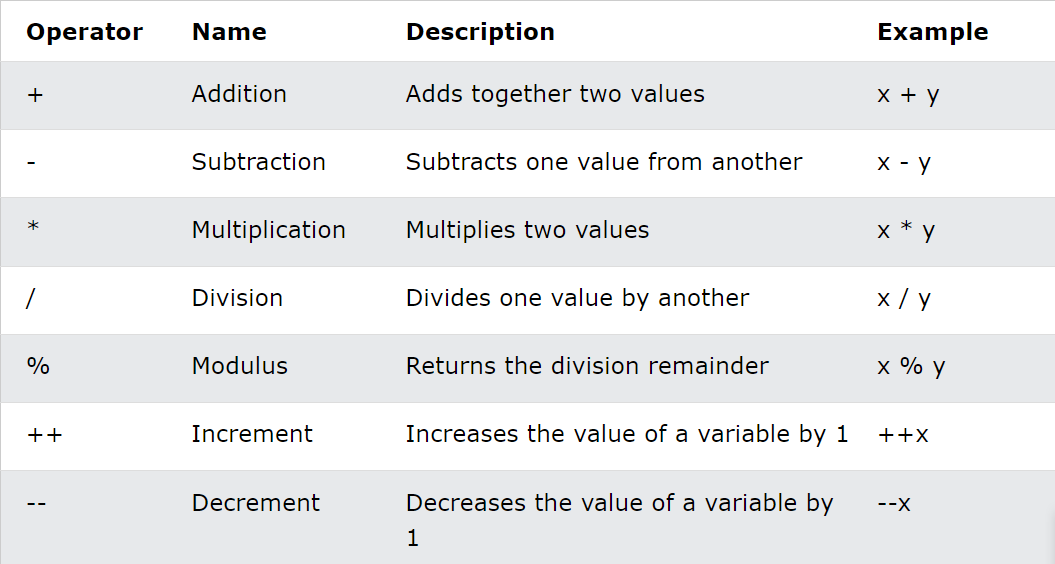
String to Double

String decimalText = "45.67";  
double decimal = Double.parseDouble(decimalText);    // becomes 45.67

String to Boolean

String boolText = "true";  
boolean bool = Boolean.parseBoolean(boolText); // becomes true

**Arithmetic Operators**



**Modulo Operator**

The modulo operator % gives the remainder of a division. In Java, it's used with a simple syntax:

result = dividend % divisor;

* dividend: The number being divided.
* divisor: The number that divides the dividend.
* result: The remainder of the division.

**For example:**

result = 10 % 3;

Here, 10 is divided by 3. 3 goes into 10 three times, with a remainder of 1. So, result will be 1.

Usually modulo is used for checking if a number is even or odd:

* If a number is even, dividing it by 2 will leave a remainder of 0.
* If a number is odd, dividing it by 2 will leave a remainder of 1.

When using modulo with floating-point numbers (doubles), it works similarly to integers but keeps the decimal precision.

**For example:**

double result = 5.2 % 2.0; // result is 1.2

**Increment/Decrement**

Increment and decrement operators are used to increase or decrease the value of a variable by 1. These operators are widely used in programming, especially in loops and counters.

The increment operator is represented by two plus signs **++**, and the decrement operator is represented by two minus signs **--**.

**For example**,

* to increment a variable named count, you can use the increment operator like this:

int count = 5;

count++; // count is now 6

* to decrement a variable named value, you can use the decrement operator like this:

int value = 10;

value--; // value is now 9

Increment (++) and Decrement (--) operators can be used in two ways:

Pre-increment/decrement (++x or --x)

* The operator goes BEFORE the variable
* The value changes IMMEDIATELY
* The new value is used in the expression

int x = 5;

int y = ++x; // x is increased to 6 first, then y becomes 6

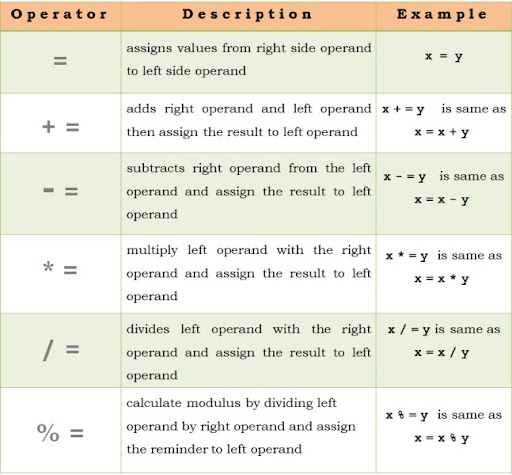
Post-increment/decrement (x++ or x--)

* The operator goes AFTER the variable
* The original value is used first
* The value changes AFTER the expression

int x = 5;

int y = x++; // y becomes 5 first, then x increases to 6

**Arithmetic Shortcuts**



**Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| == | Equal | 1 == 2 returns false |
| != | Not Equal | 1 != 2 returns true |
| > | Greater Than | 1 > 2 returns false |
| < | Less Than | 1 < 2 returns true |
| >= | Greater or Equal | 1 >= 2 returns false |
| <= | Less or Equal | 1 <= 2 returns true |