**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

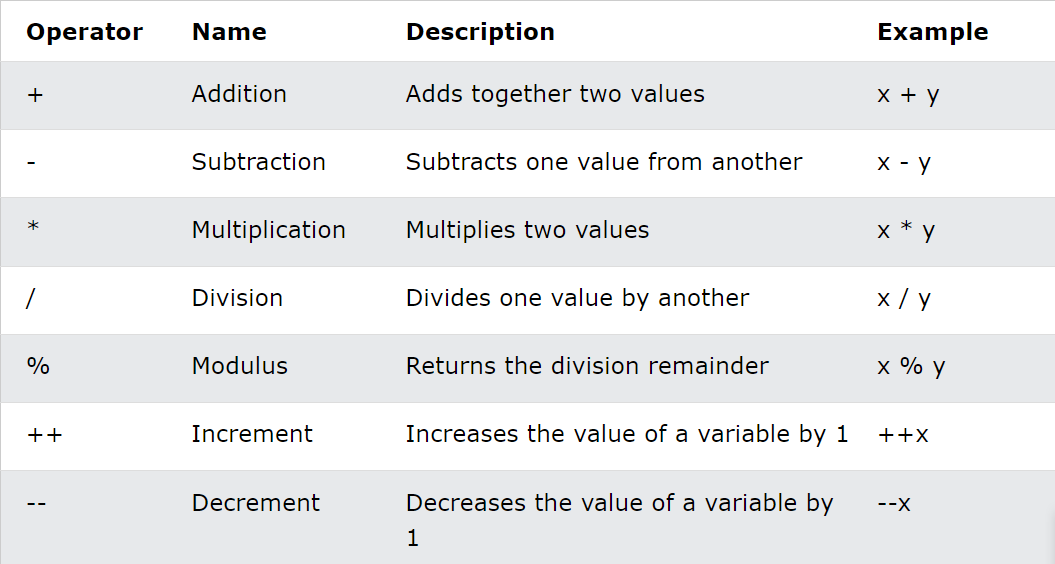
String to Double

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

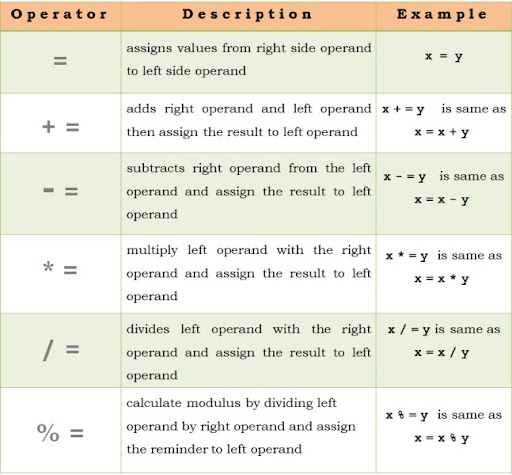
Post-increment/decrement (x++ or x--)

* 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 |

**Logical Operators**

Logical operators are used to check combinations of comparisons that return **true** or **false**.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| && | And - true if all operands are true | a && b |
| || | Or - true if any operand is true | a || b |
| ! | Not - true if the operand is false | !a |

**If Statement**

If statements allow us to execute code with conditions.

**For example,**

int age = 20;

String status = "Child";

if (age > 18) {

status = "Adult";

}

age += 1;

The above code checks whether the age variable is bigger than 18. If it is, it will set status to hold "Adult" string.

In the end, the code will increment age by 1 whether the age is bigger than 18 or not.

**If - Else**

**if** allows us to execute particular code if a condition is met, but what if we want to execute something else if the condition is not met?

For that we have the **else** statement:

int age = 15;

String status = "None";

if (age >= 18) {

status = "Adult";

} else {

status = "Young";

}

int age = 15;

String status = "None";

if (age >= 18) {

status = "Adult";

} else {

status = "Young";

}

In the above example, age is smaller than 18 which means it enters the else code, and status will hold "Young".

We can even make it more profound using the **else if** statement:

int age = 68;

String status = "None";

if (age < 18) {

status = "Young";

} else if (age >= 18 && age <= 65) {

status = "Adult";

} else {

status = "Old";

}

Here it checks whether age is smaller than 18, if not, it will continue to the next condition and check whether age is between 18 and 65. If that condition is also not met, it will set status to "Old".

We can add as many **else if** statements as we want:

if (condition1) {

code;

} else if (condition2) {

code;

} else if (condition3) {

code;

}

...

**Switch Statement**

The switch statement is like a multi-way if statement. Instead of evaluating a single condition, it checks the value of a variable against multiple cases and executes the code associated with the matching case.

switch (variable) {

case value1:

// Code to execute if variable equals value1

break;

case value2:

// Code to execute if variable equals value2

break;

// ... more cases

default:

// Code to execute if no case matches

}

* The **switch** keyword is followed by the variable you want to test in parentheses.
* Each **case** represents a possible value of the variable.
* The code inside each **case** is executed if the variable matches that case's value.
* The **break** statement is crucial; it exits the **switch** after a case is executed. Without it, execution would "fall through" to the next case.
* The default **case** is optional and is executed if no other case matches.

int day = 3;

String dayName;

switch (day) {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

// ... cases for other days

default:

dayName = "Invalid day";

}

You can also combine multiple cases into one:

int day = 3;

String dayName;

switch (day) {

case 1:

case 2:

case 3:

dayName = "Start of week";

break;

// ... cases for other days

default:

dayName = "Invalid day";

}