0. Computing Basics

▼ Main method

```
public class MyFirstJavaClass {

    public static void main(String[] args) {
        System.out.println("Hello World");
    }

}
```

▼ Variables

▼ Primitive data types (just for knowledge)

Type	Size (bits)	Minimum	Maximum
`byte`	8	-2^7 -128 10000000	2^7-1 128 01111111
`short`	16	-2^{15} -32768	$2^{15}-1$ 32767
`int`	32	-2^{31} -2147483648	$2^{31}-1$ 2147483647
`long`	64	-2^{63}	$2^{63}-1$

Whole numbers in Java use the 2's complement number system.

MSB has value of -2^{n-1} (e.g. for a 4 bit 2s compliment, the bits represent $-8, 4, 2, 1$ respectively

Number Systems and Arithmetic Calculations

Type	Size (bits)
`float`	32
`double`	64

Java uses a subset of IEEE 754 to represent floating point numbers.

Type	Size (bits)	Minimum	Maximum	
`char`	16	0	$2^{16}-1$ 65535	(Java uses UTF-16) Note that `char` in some other programming languages is 8 bits instead of 16. (e.g. Python)
`boolean`	1	-	-	`true` or `false`

▼ String class

Can be seen as an array of `char`

> ❗ Use single quotes for characters and double quotes for strings.

```
System.out.println("H" + "I"); // Prints HI
System.out.println('H' + 'I'); // Prints 145
```

> ❗ String comparison must be done with the `.equals` method

▼ ~~Literals~~

In Java, there are different types of literals:

1. Integer literals: These represent whole numbers and can be specified in decimal, binary, octal, or hexadecimal format. For example:

```
int a = 10; // decimal notation
int b = 0b1010; // binary notation
int c = 012; // octal notation
int d = 0xA; // hexadecimal notation
```

2. Floating-point literals: These represent decimal numbers with a fractional part. They can be specified using a decimal point or using scientific notation. For example:

```
double x = 3.14; // decimal notation
double y = 3.0e8; // scientific notation
```

3. Boolean literals: These represent a boolean value, which can be either true or false. For example:

```
boolean b1 = true;
boolean b2 = false;
```

4. Character literals: These represent a single character enclosed in single quotes. For example:

```
char c1 = 'A';
char c2 = '\u0041'; // Unicode representation of 'A'
```

```

```

5. String literals: These represent a sequence of characters enclosed in double quotes. For example:

```
String s = "Hello, world!";
```

It's important to note that literals are immutable, meaning their values cannot be changed once they are defined. Also, literals have a specific type and can only be used to initialize variables of compatible types.

▼ Variable declaration and initialization

```
int i; // This is a variable declaration
i = 3; // This is variable initialization (first time)
i = 3; // This is variable re-assignment (subsequent times)

int j = 3; // This is variable declaration and initialization
```

▼ Comments

```
// Single line comments

/*
Multi
line
comments
*/

/*
 * Multi line / block comments
 */

/**
 * Javadocs
 */
public int add(int a, int b) {}
```

▼ Operators

Arithmetic operators

Operator	Meaning	Example
+	add / concatenate strings	
-	subtract	
*	multiply	
/	divide	

Operator	Meaning	Example
%	modulus / remainder	
++	increment (by 1)	`++i` or `i++`
--	decrement (by 1)	

Assignment operator

Operator	Meaning	Example
=	assignment	
+=	add and assign	`x += 3`

▼ ~~Bitwise operators~~

Operator	Meaning
|	bitwise or
&	bitwise and
~	bitwise not
^	bitwise xor
>>	right shift
<<	left shift

Comparison operators

Operator	Meaning
==	Equal to
!=	Not equal
>	Greater than
<	Smaller than
>=	Greater than or equal to
<=	Smaller than or equal to

Logical operators

Operator	Meaning
&&	Logical and
||	Logical or
!	Logical not

- Short circuit property of logical operators

▼ Operator precedence and associativity

Type	Operators Precedence	Note	Associativity
Postfix increment and decrement	`i++` `i--`	`i++` will increment value of `i` **after** using its value e.g. `int i = 3;` `int j = i++;` Then `j = 3`, `i = 4`	Left to Right
Prefix increment and decrement, and unary	`++i` `--i` `+i` `-i` `~i` `!i`	`++i` will increment value of `i` **before** using its value e.g. `int i = 3;` `int j = ++i;` Then `j = 4`, `i = 4`	Right to Left
Multiplicative	`*` `/` `%`		Left to Right
Additive	`+` `-`		Left to Right
Shift	`<<` `>>` `>>>`	`>>>` is the unsigned right bit-shift operator (when shifting negative numbers, pad MSB with 0 instead of 1)	Left to Right
Relational	`<` `>` `<=` `>=` `instanceof`		Left to Right
Equality	`==` `!=`		Left to Right
Bitwise AND	`&`		Left to Right
Bitwise XOR	`^`		Left to Right
Bitwise OR	`|`		Left to Right
Logical AND	`&&`		Left to Right
Logical OR	`||`		Left to Right
Ternary	`? :`		Right to Left
Assignment	`=`	Includes all other assignments like `+=` `/=` etc	Right to Left

▼ Control Structure

▼ Conditional statements

```
if (true) {
    // do this
}



if (true) {
    // do this
} else if (true) {
    // do that
} else {
    // do another thing
}
```

```
switch (expression) {
    case x:
        // code
        break;
    case y:
        // code
        break;
    default:
        // code
}
```

▼ Loops

- Initialization
- Loop condition
- Increment / decrement

```
// For loops
for (int i = 0; i < 3; i++) {
    System.out.println("Hi");
}

// While loops
int i = 0;
while (i < 3) {
    System.out.println("Bye");
    i++;
}

// Do-while loops
int i = 0;
do {
    System.out.println("Yay");
    i++;
} while (i < 3);
```

- do-while loops are not used that often but they do come handy in specific situations
- For while loops, always remember to increment / decrement the loop parameter

▼ Break, Continue

- Continue: Goes back to the start of the loop
- Break: Gets out of the loop

▼ Nested for-loops (advanced)

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        System.out.println(String.format("i = %d, j = %d", i, j));
    }
}
```

▼ Arrays

```
String[] names = new String[] {"Alice", "Bob", "Charlie"};
    for (String name : names) {
      System.out.println(name);
    }

    int[] integers = new int[10];
    for (int i = 0; i < 3; i++) {
      integers[i] = 100;
    }
    for (int i = 0; i < 5; i++) { // Numbers default to 0
      System.out.println(integers[i]);
    }

    String[] blanks = new String[3];
    for (int i = 0; i < 3; i++) { // Objects default to null
      System.out.println(blanks[i]);
    }

    boolean[] bools = new boolean[3]; // Booleans default to false
    System.out.println(bools[0]);
```

▼ Matrices (advanced)

```
int[][] matrix1 = {{1, 2, 3}, {4, 5, 6}};
int[][] matrix2 = new int[2][3];
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        System.out.printf("%d\n", matrix1[i][j]);
    }
}
```