⋆˚₀⋆✧˚ Chapter 3 : JAVA syntax ˚✧⋆₀˚⋆

The syntax of a Java program is built upon traditional components found in most programming languages—variables, operators, and control structures. However, Java's defining characteristic is its uncompromising commitment to the **Object-Oriented** paradigm.

In Java, your program is not a collection of loose functions; it is strictly a class or a set of classes working together. For the system to execute your code, one of these classes must contain the specific entry point: `public static void main(String[] args)`. By wrapping every piece of logic within a class structure, Java ensures that your code remains modular and organized from the very first line. This chapter explores the rules and components that govern how we write within this structured environment.

1. The Global Structure of a Java Program

Every Java program follows a strict hierarchical structure. It begins with **imports**, which bring in external tools. This is followed by the **class definition**, which acts as the outer container. Inside, we find the **main method**, the mandatory entry point. Within this method, we declare **variables** and use **objects**—which are initialized via **constructors**—to perform specific tasks.

```
// 1. IMPORT: Loading external utility classes
import java.util.Scanner;
// 2. PRINCIPAL RUNNABLE CLASS: The main container
public class BasicsDemo {
    // 3. MAIN METHOD: The mandatory starting point
    public static void main(String[] args) {
```

```
        // 4. VARIABLES: Storing local data
        int sum = 0;
        int n;
        /* 5. OBJECT & CONSTRUCTOR:
           'Scanner' is the Class.
           'scan' is the Object.
           'new Scanner(System.in)' uses the Constructor with a
Parameter. */
        Scanner scan = new Scanner(System.in);
        // Read input from the user
        n = scan.nextInt();
        // 6. CONTROL STRUCTURE: A 'for' loop for logic
        for (int i = 1; i <= n; i++) {
            sum += i;
        }
        // 7. CLASS METHOD: Built-in behavior to display results
        System.out.println("Sum = " + sum);
    }
}
```

While a Java program can consist of dozens or even hundreds of different classes, they must all be organized around a single **Principal Runnable Class**. This class serves as the conductor of the entire application because it contains the main method. Without this specific "starting point," the program exists only as a collection of blueprints (classes) but has no way to actually begin execution.

2. Variables in JAVA

In Java, the **state** of an object is stored in variables. A variable is essentially a reserved memory location named by an **identifier** used to store a specific piece of data. Every variable must be declared with a **Type** and a **Name** following the syntax: `TYPE NAME;` .

Java provides a variety of primitive types to handle different categories of data, such as integers, real numbers (decimals), characters, and booleans. The following program demonstrates these types and their maximum possible values using Java's built-in constants.

```
class MaxVariablesDemo {

    public static void main(String args[]) {

        // --- Integers ---

        byte largestByte = Byte.MAX_VALUE;       // 8-bit

        short largestShort = Short.MAX_VALUE;    // 16-bit

        int largestInteger = Integer.MAX_VALUE;  // 32-bit

        long largestLong = Long.MAX_VALUE;       // 64-bit

        // --- Real Numbers (Floating Point) ---

        float largestFloat = Float.MAX_VALUE;

        double largestDouble = Double.MAX_VALUE;

        // --- Other Primitive Types ---

        char aChar = 'S';

        boolean aBoolean = true;

        // Displaying the results

        System.out.println("The largest byte value is " +
largestByte);

        System.out.println("The largest short value is " +
largestShort);

        System.out.println("The largest integer value is " +
largestInteger);
```

```
        System.out.println("The largest long value is " +
largestLong);
        System.out.println("The largest float value is " +
largestFloat);
        System.out.println("The largest double value is " +
largestDouble);
        // Character check
        if (Character.isUpperCase(aChar)) {
            System.out.println("The character " + aChar + " is
upper case.");
        } else {
            System.out.println("The character " + aChar + " is
lower case.");
        }
        System.out.println("The value of aBoolean is " +
aBoolean);
    }
}
```

The expected output of the precedent program is :

The largest byte value is 127

The largest short value is 32767

The largest integer value is 2147483647

The largest long value is 9223372036854775807

The largest float value is 3.40282e+38

The largest double value is 1.79769e+308

The character S is upper case.

The value of aBoolean is true

When naming your variables, you must adhere to the following Java naming conventions:

A. **Starting Character:** It must begin with a letter.

B. **Prohibited Words:** It cannot be a Java **keyword** (e.g., class, static, void), a boolean literal (true, false), or the word null.

C. **Special Characters:** Only the underscore _ is permitted; other special characters (like @, #, or space) are invalid.

3. Constants

If you want to create a variable whose value cannot be changed once assigned, use the **final** keyword. This designates the variable as a **constant**.

```
// Syntax: final TYPE NAME = VALUE;

final double PI = 3.14159;
```

4. Operator in JAVA

Java provides a comprehensive set of operators to manipulate variables and perform logic. These are categorized into arithmetic, relational, logical, and bitwise operations.

Category	Operators	Description
Arithmetic	+, -, *, /, %	Standard math (Addition, Subtraction, Multiplication, Division, and Remainder/Modulo).
Relational	==, !=, >, >=, <, <=	Used for comparisons; they return a boolean (true or false).
Logical	&&, `	
Bitwise (Binary)	&, |, ^, ~	AND, OR, XOR, and NOT performed at the bit level.

Another arithmetic operator are the ++ and – operators. The position of ++ and 66 determines when the variable is updated relative to the calculation. This is a common source of logic errors.

- **Prefix (++m):** Increments the value **before** it is used in the expression.
- **Postfix (n++):** Increments the value **after** the current expression is evaluated.

```
int m = 7;

int n = 7;

// Prefix: m becomes 8, then 2 * 8 = 16

int a = 2 * ++m;

System.out.println("a = " + a + ", m = " + m); // Output: a = 16, m = 8

// Postfix: 2 * 7 = 14 is calculated first, then n becomes 8

int b = 2 * n++;

System.out.println("b = " + b + ", n = " + n); // Output: b = 14, n = 8
```

One of specific operator in java is the ternary operator, which is a shorthand way of writing an if-else statement. It evaluates a condition and returns one of two values.

Syntax: condition ? expression1 : expression2;

```
// If x is less than y, a is assigned x. Otherwise, a is assigned y.

int a = (x < y) ? x : y;
```

Another type of operators are the bitwise operators that manipulate the individual bits of an integer. This is often used for low-level flag management or extracting specific data from a binary stream.

- & (AND): 1 if both bits are 1.
- | (OR): 1 if at least one bit is 1.
- ^ (XOR): 1 if bits are different.

- ~ (NOT): Flips all bits.

```
// Extracts the 4th bit and reduces it to a 0 or 1

int fourthBitFromRight = (n & 8) / 8;
```

The next table give outputs for several value of n (in binary 8 = 1000):

Input (n)	Binary of n	Result of (n & 8)	Final Output (/8)
7	0111	0111&1000=0000 (0)	0
9	1001	1001&1000=1000 (8)	1
15	1111	1111&1000=1000 (8)	1
2	0010	0010&1000=0000 (0)	0

5. Data Types and Advanced Syntax in Java

Java is a **strongly typed language**, meaning every variable must be declared with a specific type before it can be used.

Java features eight primitive types, categorized by the nature and size of the data they hold:

Category	**Type**	**Size**	**Description**
Integers	byte	8-bit	Very small integers (-128 to 127)
	short	16-bit	Small integers
	int	32-bit	Standard integers (Most common)
	long	64-bit	Large integers
Real Numbers	float	32-bit	Single-precision floating point
	double	64-bit	Double-precision (Standard for decimals)
Other	char	16-bit	A single Unicode character
	boolean	1-bit/8-bit	Logical true or false

Often, we need to move data from one type to another. This is called type conversion. We could find two type of conversion:

- **Widening Conversion:** Automatic (e.g., int to float). However, large integers converted to float might lose some precision.
- **Typecasting (Narrowing):** Required when there is a risk of losing information.

```
double x = 9.997;

int nx = (int) x; // nx becomes 9 (decimal is truncated)
```

In Java, a **String is not a primitive type; it is a Class.** Therefore, every string we create is an **object**.

```
String e = "ESSAT";

String a = new String("ESSAT");

char ch[] = {'E','S','S','A','T'};

String b = new String(ch);
```

Java dispose of a certain number of String methods used to manipulate Strings, from which we can cite:

- length(): Returns number of characters.
- substring(start, end): Extracts a portion of the string.
- charAt(index): Returns the character at a specific position.
- equals(): Compares content (Use this instead of == for strings!).
- toUpperCase() / toLowerCase(): Changes the case of the text.

Like Strings, Java can handle also **arrays,** which are treated as objects in Java. Once an array is created, its size is fixed.

```
int[] a = new int[100]; // Creates an array of 100 integers

for (int i = 0; i < a.length; i++) {

    a[i] = i; // Accessing by index

}
```

Note: Java provides the **ArrayList** class as a dynamic alternative to the standard array. While a standard array has a **fixed size** determined at the time of creation, an ArrayList

belongs to the Java Collections Framework and can **grow or shrink** automatically as you add or remove elements.

Java provides also massive libraries of pre-written methods organized into **packages**:

A. Numbers and Math (Math class)

The Math class provides static methods for trigonometry and arithmetic:

- Math.sqrt(x): x
- Math.pow(x, y): x^y
- Math.abs(x): Absolute value |x|
- Math.random(): Returns a decimal between 0.0 and 1.0.

B. Characters (Character class)

Useful for validating user input:

- Character.isDigit(c): Checks if it's a number.
- Character.isLetter(c): Checks if it's a letter.

C. Array Utilities (Arrays class)

Requires import java.util.Arrays;:

- Arrays.toString(arr): Converts an array to a readable string format.
- Arrays.asList(arr).contains("Value"): Checks if an element exists in the array.

In order to handle mathematics constant, Java use:

Constant	Type	Value
Math.PI	double	3.141592653589793
Math.E	double	2.718281828459045

6. Input and output in JAVA

To make a program interactive, Java uses specialized objects to handle data flowing into the system (Input) and data displayed to the user (Output).

Reading input is performed by the **Scanner** class. Because this class is not in the default Java package, you must import it from java.util and then create a Scanner object attached to the standard input stream (System.in).

```
import java.util.Scanner; // Must be at the very top of your code

// Create the scanner object

Scanner in = new Scanner(System.in);
```

The Scanner class provides specific methods to capture different data types:

- **Strings:** in.nextLine() (reads a full line) or in.next() (reads a single word).
- **Integers:** in.nextInt(), in.nextShort(), in.nextByte(), in.nextLong().
- **Decimals:** in.nextDouble(), in.nextFloat().
- **Booleans:** in.nextBoolean().

Note: Java does not have a nextChar() method. To read a single character, you must read the input as a String and then extract the first character:

```
char carac = in.next().charAt(0);
```

we can also read array elements using the Scanner.

```
int[] a = new int[5];

for (int i = 0; i < a.length; i++) {

    System.out.print("Enter value " + i + ": ");

    a[i] = in.nextInt(); // Fills the array index with user input

}
```

Output is managed by the System.out object. Unlike the Scanner, System.out is automatically available; we don't need to import a library or create a new object.

We can use println method to display text or variables and then moves the cursor to a new line. Using the + operator allow to combine (concatenate) strings and variables.

```
int a = 10, b = 20;

System.out.println("a: " + a + " b: " + b);
```

Borrowed from the C language, printf allows also for precise formatting, such as limiting the number of decimal places for real numbers.

- **%d**: Placeholder for integers.
- **%.2f**: Placeholder for floating-point numbers (rounded to 2 decimal places).
- **\n**: Command to start a new line.

```
double f = 10.2555;

System.out.printf("Formatted: %.2f\n", f); // Output: 10.26

System.out.println("Actual: " + f);          // Output: 10.2555
```

7. Control structures in JAVA

Control structures determine the "flow" of your program, allowing it to make decisions or repeat actions based on specific conditions. Without these, a program would simply execute line-by-line in a fixed, linear order.

Java uses standard if-else logic and switch cases to branch code execution based on boolean evaluations.

The if-else statement is the fundamental tool for handling logic. If the condition inside the parentheses is true, the first block executes; otherwise, the else block runs.

```
if (yourSales >= target) {
    performance = "Satisfactory";
    bonus = 100 + 0.01 * (yourSales - target);
} else {
    performance = "Unsatisfactory";
    bonus = 0;
}
```

For more complex scenarios, the **else if** ladder is used. The **braces { }** are always used to define the scope of your instructions clearly.

```
if (yourSales >= 2 * target) {
    performance = "Excellent";
    bonus = 1000;
} else if (yourSales >= 1.5 * target) {
    performance = "Fine";
    bonus = 500;
} else if (yourSales >= target) {
    performance = "Satisfactory";
    bonus = 100;
} else {
    System.out.println("You're fired");
}
```

The switch statement is a cleaner alternative to multiple if statements when comparing a single variable against several constant values.

```
Scanner in = new Scanner(System.in);
System.out.print("Select an option (1, 2, 3, 4): ");
int choice = in.nextInt();
switch (choice) {
    case 1:
        // Logic for option 1
        break; // Exits the switch
    case 2:
        // Logic for option 2
```

```
        break;
    default:
        // Handles invalid input
        System.out.println("Bad input");
        break;
}
```

Java uses also Loops to repeat a block of code until a specific condition is no longer met. For this Java uses three kind of loops (while, do/while and for).

Thes while loop checks the condition **before** each iteration. If the condition is false initially, the code inside may never run.

```
while (balance < goal) {
    balance += payment;
    double interest = balance * interestRate / 100;
    balance += interest;
    years++;
}
System.out.println(years + " years.");
```

The do/while loop is used when the code **must execute at least once**. The condition is checked at the end of the block.

```
do {
    balance += payment;
    year++;
    System.out.printf("After year %d, your balance is %,.2f%n",
year, balance);
    System.out.print("Ready to retire? (Y/N)");
    input = in.next();
```

```
} while (input.equals("N"));
```

The for loop is the most concise way to repeat code when you know the number of iterations in advance. It combines initialization, condition, and update in one line.

```
for (int i = 10; i > 0; i--) {
    System.out.println("Counting down ... " + i);
}
```

Java provides also two keywords to manually alter loop behavior:

- **break**: Immediately exits the loop entirely, regardless of the condition.

```
while (years <= 100) {
balance += payment;
double interest = balance * interestRate / 100;
balance += interest;
if (balance >= goal)
   break;
years++;
}
```

- **continue**: Skips the remaining code in the current iteration and jumps back to the **start (header)** of the loop for the next cycle.

```
while (sum < goal) {
System.out.print("Enter a number: "); n = in.nextInt();
if (n < 0)
     continue;
sum += n;
// not executed when n < 0
}
```

Note : Java code must be translated into bytecode before it can be executed by the Java Virtual Machine (JVM).

Compile: Use the javac tool on your source file.

`javac yourprogram.java` (This creates a yourprogram.class file)

Run: Use the java tool on the class containing the main method.

```
java yourmainclass
```