# Introduction to Java

Java is a high-level, object-oriented programming language developed by James Gosling at Sun Microsystems (now owned by Oracle) and released in 1995. It follows the Write Once, Run Anywhere (WORA) principle, meaning Java programs can run on any system that has a Java Virtual Machine (JVM).

**Features of Java**

* **Platform-Independent**: Java programs run on any OS with a compatible JVM.
* **Object-Oriented**: Supports principles like encapsulation, inheritance, and polymorphism.
* **Robust and Secure**: Includes strong memory management, exception handling, and security features.
* **Multithreading**: Supports concurrent execution of multiple threads.
* **High Performance**: Uses Just-In-Time (JIT) compilation to improve speed.

## JDK, JRE, and JVM

Java consists of three main components:

### 1. Java Virtual Machine (JVM)

JVM is an abstract machine that provides a runtime environment for Java applications. It converts **bytecode** into machine-specific code. The key tasks of JVM include:

* **Loading** the bytecode
* **Verifying** and **executing** the code
* **Memory management (Garbage Collection)**

### 2. Java Runtime Environment (JRE)

JRE is a part of the **JDK** and includes:

* **JVM**
* **Core libraries**
* **Other supporting files** to run Java applications.

JRE **does not** include development tools like the compiler.

### 3. Java Development Kit (JDK)

JDK is a complete software development package that includes:

* **JRE (JVM + Libraries)**
* **Java Compiler (javac)**
* **Debugger and other development tools**

**JDK = JRE + Development Tools**

### JDK Versions

Java has different versions, such as:

* **JDK SE (Standard Edition)** – For general-purpose programming.
* **JDK EE (Enterprise Edition)** – For large-scale enterprise applications.
* **JDK ME (Micro Edition)** – For mobile and embedded devices.

### Summary Table

| **Component** | **Purpose** |
| --- | --- |
| **JVM** | Converts bytecode to machine code and executes it. |
| **JRE** | Contains JVM and libraries required to run Java programs. |
| **JDK** | Includes JRE + development tools (compiler, debugger, etc.). |

Here’s a concise note on **Basic Syntax: Variables, Data Types, and Operators in Java** 🚀

## Java Basic Syntax

Every Java program starts with a **class**.

The main method is the entry point of the program:

public class HelloWorld {

public static void main(String[] args) {

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

}

}

Java is **case-sensitive** (Hello and hello are different).

Statements end with a **semicolon (;)**.

Blocks of code are enclosed in **curly braces {}**.

## Variables in Java

A **variable** is a container for storing data. It has a **type** and a **name**.

**Declaring Variables:**

dataType variableName = value;

**Example:**

int age = 25;

String name = "Bibek";

double price = 99.99;

**Types of Variables:**

1. **Local Variable** – Declared inside a method, accessible only within it.
2. **Instance Variable (Non-static Variable)** – Belongs to an object, declared inside a class but outside methods.
3. **Static Variable (Class Variable)** – Shared among all objects of a class using static.

## Data Types in Java

Java has **two categories** of data types:

### A. Primitive Data Types (8 types)

| **Data Type** | **Size** | **Example** |
| --- | --- | --- |
| byte | 1 byte | byte b = 100; |
| short | 2 bytes | short s = 32000; |
| int | 4 bytes | int num = 10; |
| long | 8 bytes | long bigNum = 100000L; |
| float | 4 bytes | float pi = 3.14f; |
| double | 8 bytes | double price = 99.99; |
| char | 2 bytes | char letter = 'A'; |
| boolean | 1 bit | boolean isJavaFun = true; |

### B. Non-Primitive Data Types

* **String**: String name = "Java";
* **Arrays**: int[] numbers = {1, 2, 3};
* **Classes & Objects**: Custom-defined data types.

## Operators in Java

Operators perform operations on variables and values.

### A. Arithmetic Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| % | Modulus (Remainder) | a % b |

### B. Relational (Comparison) Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

### C. Logical Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| && | Logical AND | (a > 5 && b < 10) |
| ` |  | ` |
| ! | Logical NOT | !(a > b) |

### D. Assignment Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| = | Assign value | a = 10 |
| += | Add & assign | a += 5 (same as a = a + 5) |
| -= | Subtract & assign | a -= 5 |
| \*= | Multiply & assign | a \*= 5 |
| /= | Divide & assign | a /= 5 |

### E. Unary Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| ++ | Increment | a++ (Post-increment) or ++a (Pre-increment) |
| -- | Decrement | a-- (Post-decrement) or --a (Pre-decrement) |

**Example Program Using Variables, Data Types, and Operators**

public class JavaBasics {

public static void main(String[] args) {

int a = 10, b = 5;

boolean isJavaFun = true;

double sum = a + b;

System.out.println("Sum: " + sum);

System.out.println("Is Java fun? " + isJavaFun);

System.out.println("10 > 5: " + (a > b));

}

}

## Array

An array in Java is a data structure that stores multiple values of the same data type in contiguous memory locations. Arrays have a fixed size and can store both primitive and object types.  
  
### Declaration and Initialization:  
1. \*\*Single Dimensional Array:\*\*  
 ```java  
 int[] numbers = new int[5]; // Declaration and memory allocation  
 int[] values = {10, 20, 30, 40, 50}; // Direct initialization  
 ```  
  
2. \*\*Accessing Array Elements:\*\*  
 ```java  
 numbers[0] = 5; // Assigning value  
 System.out.println(numbers[0]); // Accessing value  
 ```  
  
3. \*\*Iterating Through an Array:\*\*  
 ```java  
 for (int i = 0; i < values.length; i++) {  
 System.out.println(values[i]);  
 }  
 // Using Enhanced For Loop  
 for (int num : values) {  
 System.out.println(num);  
 }  
 ```

## 2D Array

A two-dimensional array in Java is an array of arrays, used for storing tabular data.  
  
### Declaration and Initialization:  
```java  
int[][] matrix = new int[3][3]; // Declaration with size  
int[][] predefinedMatrix = {  
 {1, 2, 3},  
 {4, 5, 6},  
 {7, 8, 9}  
}; // Direct initialization  
```  
  
### Accessing Elements:  
```java  
matrix[0][0] = 5; // Assigning value  
System.out.println(matrix[0][0]); // Accessing value  
```  
  
### Iterating Through a 2D Array:  
```java  
for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[i].length; j++) {  
 System.out.print(matrix[i][j] + " ");  
 }  
 System.out.println();  
}  
```

## ArrayList

ArrayList is a resizable array in Java provided by the `java.util` package. Unlike arrays, it can dynamically grow and shrink as needed.  
  
### Declaration and Initialization:  
```java  
import java.util.ArrayList;  
ArrayList<String> names = new ArrayList<>();  
```  
  
### Adding Elements:  
```java  
names.add("Alice");  
names.add("Bob");  
```  
  
### Accessing Elements:  
```java  
System.out.println(names.get(0));  
```  
  
### Removing Elements:  
```java  
names.remove("Alice");  
```  
  
### Iterating Through an ArrayList:  
```java  
for (String name : names) {  
 System.out.println(name);  
}  
```

## String

A `String` in Java is an immutable sequence of characters stored as an object of the `String` class.  
  
### Declaration and Initialization:  
```java  
String str1 = "Hello";  
String str2 = new String("World");  
```  
  
### String Methods:  
```java  
System.out.println(str1.length()); // Get length  
System.out.println(str1.toUpperCase()); // Convert to uppercase  
System.out.println(str1.concat(str2)); // Concatenation  
System.out.println(str1.equals(str2)); // Compare two strings  
```

## Wrapper Class

Wrapper classes in Java provide an object representation of primitive data types. They allow us to use primitives as objects, which is useful for collection frameworks.  
  
### Common Wrapper Classes:  
1. \*\*Integer\*\* → `Integer num = Integer.valueOf(10);`  
2. \*\*Double\*\* → `Double d = Double.valueOf(5.5);`  
3. \*\*Character\*\* → `Character c = Character.valueOf('A');`  
  
### Autoboxing and Unboxing:  
- \*\*Autoboxing:\*\* Automatic conversion of a primitive to its wrapper object.  
 ```java  
 Integer obj = 100; // Autoboxing  
 ```  
  
- \*\*Unboxing:\*\* Automatic conversion of a wrapper object back to a primitive.  
 ```java  
 int num = obj; // Unboxing  
 ```

## Function (Method)

A method in Java is a block of code that performs a specific task. Methods are used for code reusability and better modularity.  
  
### Defining a Method:  
```java  
public static int add(int a, int b) {  
 return a + b;  
}  
```  
  
### Calling a Method:  
```java  
public static void main(String[] args) {  
 int result = add(5, 3);  
 System.out.println(result);  
}  
```  
  
### Method Overloading:  
Method overloading allows multiple methods with the same name but different parameters.  
```java  
public static int multiply(int a, int b) {  
 return a \* b;  
}  
public static double multiply(double a, double b) {  
 return a \* b;  
}  
```

**Loops in Java**

Loops in Java are used to execute a block of code multiple times until a specific condition is met. Java provides three primary types of loops:

**1. For Loop**

The for loop is used when the number of iterations is known beforehand.

**Syntax:**

for(initialization; condition; update) {

// Code to be executed

}

**Example:**

for(int i = 1; i <= 5; i++) {

System.out.println("Iteration: " + i);

}

**2. While Loop**

The while loop executes a block of code as long as the given condition is true.

**Syntax:**

while(condition) {

// Code to be executed

}

**Example:**

int i = 1;

while(i <= 5) {

System.out.println("Iteration: " + i);

i++;

}

**3. Do-While Loop**

The do-while loop executes the block of code at least once, and then continues executing it as long as the condition is true.

**Syntax:**

do {

// Code to be executed

} while(condition);

**Example:**

int i = 1;

do {

System.out.println("Iteration: " + i);

i++;

} while(i <= 5);

**4. Enhanced For Loop (For-Each Loop)**

The enhanced for loop (also known as the for-each loop) is used to iterate over arrays and collections.

**Syntax:**

for(dataType variable : array/collection) {

// Code to be executed

}

**Example:**

int[] numbers = {1, 2, 3, 4, 5};

for(int num : numbers) {

System.out.println("Number: " + num);

}

**5. Loop Control Statements**

* **break statement:** Exits the loop immediately.
* **continue statement:** Skips the current iteration and moves to the next one.

**Example using break and continue:**

for(int i = 1; i <= 5; i++) {

if(i == 3) {

break; // Exits the loop when i is 3

}

System.out.println("Iteration: " + i);

}

for(int i = 1; i <= 5; i++) {

if(i == 3) {

continue; // Skips iteration when i is 3

}

System.out.println("Iteration: " + i);

}

**Summary**

* **For Loop:** Used when the number of iterations is known.
* **While Loop:** Used when the condition is evaluated before execution.
* **Do-While Loop:** Used when the condition is evaluated after execution.
* **For-Each Loop:** Used to iterate over arrays and collections.
* **Break & Continue:** Used for controlling loop execution.

\*\*Formatted Printing in Java (printf)

**1. Introduction to printf**

printf in Java is used for formatted output. It belongs to the System.out class and allows precise control over text alignment, number formatting, and spacing. It is similar to printf in C and C++.

**Syntax:**

System.out.printf(formatString, arguments);

* formatString: A string containing format specifiers (e.g., %d, %s, %f).
* arguments: The values that replace the format specifiers.

**2. Common Format Specifiers**

| **Specifier** | **Description** | **Example Usage** | **Output** |
| --- | --- | --- | --- |
| %d | Integer (decimal) | System.out.printf("%d", 123); | 123 |
| %03d | Integer (zero-padded, 3 digits) | System.out.printf("%03d", 7); | 007 |
| %-5d | Left-aligned integer (5 spaces) | System.out.printf("%-5d", 42); | 42 |
| %s | String | System.out.printf("%s", "Java"); | Java |
| %15s | Right-aligned string (15 spaces) | System.out.printf("%15s", "Hello"); | Hello |
| %-15s | Left-aligned string (15 spaces) | System.out.printf("%-15s", "Hello"); | Hello |
| %f | Floating-point (default 6 decimal places) | System.out.printf("%f", 3.14); | 3.140000 |
| %.2f | Floating-point (2 decimal places) | System.out.printf("%.2f", 3.14159); | 3.14 |
| %n | Newline (platform-independent) | System.out.printf("Hello%nWorld"); | Hello (new line) World |

**3. Examples of Using printf**

**Example 1: Formatting Strings and Numbers**

public class Main {

public static void main(String[] args) {

String name = "Alice";

int age = 25;

System.out.printf("Name: %-10s Age: %02d%n", name, age);

}

}

**Output:**

Name: Alice Age: 25

Explanation:

* %-10s → Left-aligns the string in a 10-character space.
* %02d → Prints the integer with at least 2 digits, adding leading zeros if necessary.

**Example 2: Formatting Floating-Point Numbers**

public class Main {

public static void main(String[] args) {

double price = 5.6789;

System.out.printf("Price: $%.2f%n", price);

}

}

**Output:**

Price: $5.68

Explanation:

* %.2f → Rounds to 2 decimal places.

**Example 3: Formatting Multiple Values**

public class Main {

public static void main(String[] args) {

String product = "Laptop";

int quantity = 2;

double price = 799.99;

System.out.printf("Product: %-10s Quantity: %02d Price: $%.2f%n", product, quantity, price);

}

}

**Output:**

Product: Laptop Quantity: 02 Price: $799.99

**4. Key Advantages of printf**

* Provides **better control** over formatting.
* Useful for aligning **columns of data** in a table.
* Makes numbers **more readable** (e.g., currency, padding).
* **Reduces concatenation** compared to System.out.println.

**5. Summary**

1. **printf** is used for formatted output in Java.
2. It uses **format specifiers** like %d, %s, and %f to format integers, strings, and floating-point numbers.
3. Allows **alignment, padding, and decimal precision control**.
4. Helps create **readable and structured output**.

This concludes the note on **Formatted Printing in Java (printf)**.