Java Basics

Introduction to Java

- **Java** is a **class-based**, **object-oriented** programming language.
- Designed to be **secure**, **portable**, and **reliable**.
- Famous for its principle:
  "Write Once, Run Anywhere" (WORA) → Java code runs on any device with a **Java Virtual Machine (JVM)**.

Why Learn Java?

- Used in **Android apps**, **web applications**, **enterprise systems**, **games**, and **cloud applications**.
- Easy to learn for beginners but powerful enough for professionals.

Java Development Environment

To run Java programs, you need to set up the environment.

Key Components:

Component	Description
JVM (Java Virtual Machine)	Runs Java programs by converting bytecode into machine code.
JRE (Java Runtime Environment)	JVM + libraries needed to run Java apps.
JDK (Java Development Kit)	JRE + development tools (compiler, debugger). Required for writing and compiling programs.

Environment Variables:

- **PATH** → Lets you run Java commands (javac, java) from anywhere.
- **JAVA_HOME** → Points to the folder where Java is installed.

Java Basic Syntax

Java Basics

Every Java program is built using **classes**, **methods**, and **statements**.

Example: Hello World Program

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

Explanation:

- public class HelloWorld → Defines a class named HelloWorld.
- public static void main(String[] args) → Entry point of the program.
- System.out.println("Hello, World"); → Prints text to the console.

Output:

```
Hello, World
```

Comments in Java

Comments are notes in the code. They are ignored by the compiler.

Types of Comments:

- **Single-line** → // This is a comment
- **Multi-line** →

  ```
  /* This is a
     multi-line comment */
  ```
- **Documentation (Javadoc)** →

  ```
  /** This is a documentation comment */
  ```

Java Basics

Data Types in Java

Data types define the kind of values a variable can hold.

Categories:

1. **Primitive Data Types** (basic, built-in)
   - byte, short, int, long, float, double, char, boolean
2. **Non-Primitive Data Types** (objects)
   - String, Arrays, Classes, Interfaces

Example:

```
byte b = 100;
int i = 100000;
long l = 10000000000L;
float f = 3.14f;
double d = 3.14159;
char c = 'A';
boolean flag = true;

String str = "Hello, Java";
int[] arr = {1, 2, 3, 4, 5};
Integer wrapperInt = Integer.valueOf(50);
```

Variables in Java

Variables are containers for storing data.

Types of Variables:

Java Basics

Type	Scope	Example
Local	Inside methods/blocks	int localVar = 5;
Instance	Inside class, outside methods	int instanceVar = 10;
Static	Shared across all objects	static String staticVar = "I am static";
Final	Value cannot change	final int MAX = 100;

Keywords in Java

- **Keywords** are reserved words with special meaning.
- Examples: class, public, static, void, int, return, new, if, else, switch, final, package, import.

You cannot use keywords as variable names.

Operators in Java

Operators perform actions on variables and values.

Types of Operators:

Type	Symbols	Example
Arithmetic	+ - * / %	a + b
Relational	== != > < >= <=	a > b
Logical	&& || !	x && y
Assignment	= += -= *= /= %=	a += 5
Unary	+ - ++ -- !	++a

Java Basics

Type	Symbols	Example
Ternary	?:	(a > b) ? a : b
Bitwise	& | ^ ~ << >> >>>	a & b

Decision Making (Control Statements)

Control statements let programs choose execution paths.

Examples:

- **if** → Executes block if condition is true.
- **if-else** → Chooses between two blocks.
- **if-else-if** → Tests multiple conditions.
- **switch** → Selects one block from many options.

Example:

```
int number = 10;


if (number > 0) {
   System.out.println("Positive");
} else if (number == 0) {
   System.out.println("Zero");
} else {
   System.out.println("Negative");
}


int day = 3;
```

```
switch (day) {
   case 1: System.out.println("Monday"); break;
   case 2: System.out.println("Tuesday"); break;
   case 3: System.out.println("Wednesday"); break;
   default: System.out.println("Other day");
}
```

Loops in Java

Loops allow code to run repeatedly while a condition is true.

Types of Loops:

- **for loop** → Repeats a block a fixed number of times.
- **while loop** → Repeats while condition is true.
- **do-while loop** → Executes at least once, then checks condition.
- **for-each loop** → Iterates through arrays/collections.

Example:

```
for (int i = 1; i <= 5; i++) {
   System.out.println("Count: " + i);
}
```

Output:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

Java Basics

Loops in Java

Loops are **control statements** that allow a block of code to be executed repeatedly as long as a condition is true. They help reduce code repetition and make programs more efficient.

The 4 Types of Loops in Java

1. for loop

- Used when the number of iterations is **known in advance**.
- Syntax includes initialization, condition, and increment/decrement.

```
for (int i = 1; i <= 5; i++) {
   System.out.println("i = " + i);
}
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

---

2. while loop

- Used when the number of iterations is **not known in advance**.
- The condition is checked **before** each iteration.

```
int j = 1;
while (j <= 5) {
    System.out.println("j = " + j);
    j++;
}
```

Output:

```
j = 1
j = 2
j = 3
j = 4
j = 5
```

---

3. do-while loop

- Similar to while, but the condition is checked **after** executing the block.
- Ensures the loop body runs **at least once**, even if the condition is false.

```
int k = 1;
do {
    System.out.println("k = " + k);
    k++;
} while (k <= 5);
```

Output:

```
k = 1
k = 2
k = 3
k = 4
k = 5
```

---

4. for-each loop (Enhanced for loop)

- Used to iterate over **arrays** or **collections**.
- Simplifies iteration without needing an index.

```
int[] numbers = {10, 20, 30, 40, 50};
for (int num : numbers) {
   System.out.println("num = " + num);
}
```

Output:

```
num = 10
num = 20
num = 30
num = 40
num = 50
```

Complete Example: LoopsDemo

```
public class LoopsDemo {
   public static void main(String[] args) {

      // 1. For loop
      System.out.println("For Loop:");
      for (int i = 1; i <= 5; i++) {
         System.out.println("i = " + i);
      }
```

```
    // 2. While loop
    System.out.println("\nWhile Loop:");
    int j = 1;
    while (j <= 5) {
        System.out.println("j = " + j);
        j++;
    }

    // 3. Do-While loop
    System.out.println("\nDo-While Loop:");
    int k = 1;
    do {
        System.out.println("k = " + k);
        k++;
    } while (k <= 5);

    // 4. Enhanced For Loop (for-each loop)
    System.out.println("\nEnhanced For Loop:");
    int[] numbers = {10, 20, 30, 40, 50};
    for (int num : numbers) {
        System.out.println("num = " + num);
    }
}
```

}

Summary

- **Java** → Class-based, Object-Oriented, Portable.
- **JVM, JRE, JDK** → Core components of the Java environment.
- **Basic Syntax** → Programs are built with classes, methods, and statements.
- **Comments** → Single-line (//), multi-line (/* ... */), and Javadoc (/** ... */).
- **Data Types** → Primitive (byte, int, double, etc.) + Non-Primitive (String, Arrays, Classes).
- **Variables** → Local, Instance, Static, Final.
- **Keywords** → Reserved words (cannot be reused, e.g., class, public, static).
- **Operators** → Arithmetic, Relational, Logical, Assignment, Unary, Ternary, Bitwise.
- **Decision Making** → if, if-else, if-else-if, switch.
- **Loops** → for, while, do-while, for-each.

  - **for loop** → Best when you know the exact number of iterations.
  - **while loop** → Best when the number of iterations is unknown in advance.
  - **do-while loop** → Always executes at least once before checking the condition.
  - **for-each loop** → Best for arrays and collections; avoids index errors.