

Day-1: Introduction + Setup

What is Programming?

Programming means **giving instructions to a computer** to do a task.

We write these instructions using a **programming language**.

What is Java?

Java is a **high-level, object-oriented programming language**.

It is widely used at industry levels.

Why Java?

- Works on all platforms (Write Once, Run Anywhere).
- Used in Android apps, web apps, banking systems, etc.
- Secure and fast.

JVM, JRE, JDK

Term	Meaning	Purpose
JVM	Java Virtual Machine	Runs Java programs
JRE	Java Runtime Environment	Contains JVM + libraries needed to run Java programs
JDK	Java Development Kit	Contains JRE + tools to write and compile Java programs

To run Java → JRE

To write Java → JDK

Install Java + IDE (Eclipse): Live Explanation

First Program: Printing Hello World,

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

Day-2: Basics of Java Syntax

Structure of a Java Program

Every Java program has a **class**.

Inside the class, we write the `main()` method.

All code runs inside the `main()` method.

Class

- Every Java program must have a class.
- The class name and file name should be the same. Example: `class Hello { }`

Public

- Access modifier.
- Means this method can be used by JVM from anywhere.
- JVM must access `main()`, so it should be public.

Static

- Belongs to the class, not to an object.
- JVM can call `main()` **without creating an object**.

Void

- Return type.
- `main()` does not return any value, so return type is void.

`main()` Method

- This is the starting point of the program.
- Java begins execution from here.
- Syntax: `public static void main(String[] args)`

`String[] args`

- Used to take command-line inputs.
- `args` is an array of Strings.
- Even if we don't use it, it must be written.

Statements

- Actual code written inside the `main()` method.
- Each statement ends with a semicolon ;

System.out.println()

- Used to print output on the screen.
- Placed inside the main() method.

public → JVM can access it

static → JVM can call it without object

void → no return value

main → starting method

String[] args → command-line input

Comments

Used to explain the code.

Not executed by Java.

Types:

- Single-line → //
- Multi-line → /* */

Tokens (Overview):

Smallest parts of a Java program.

Types:

- **Keywords** → Reserved words (class, int, if, etc.)
- **Identifiers** → Names given to variables, classes, methods
- **Literals** → Fixed values (10, 3.5, 'A', true)
- **Operators** → Symbols used for operations (+, -, *, /, etc.)

Day-3: Data Types + Variables

Variables

A variable is a name used to store data.

Value stored in a variable can change.

Must be declared before use.

Example: `int age = 20;`

Primitive Data Types

Java has 8 primitive data types

1. byte

Size: 1 byte

Range: -128 to 127

Used for small numbers.

2. short

Size: 2 bytes

Range: -32,768 to 32,767

3. int

Size: 4 bytes

Most commonly used for whole numbers.

4. long

Size: 8 bytes

Used for very large numbers.

Ends with L (example: 100000L)

5. float

Size: 4 bytes

Decimal values (less precision).

Ends with f (example: 3.14f)

6. double

Size: 8 bytes

Decimal values (more precision).

Default type for decimals.

7. char

Size: 2 bytes

Stores a single character.

Written in single quotes → 'A'

8. Boolean

Size: 1 bit

true or false.

Range + Memory Idea

Each data type has a fixed size in memory.

Smaller types store smaller values (byte < short < int < long).

float and double store decimal values.

char stores Unicode characters.

boolean stores only true/false.

Type Conversion (Widening)

Converting smaller type → larger type.

Happens automatically.

Examples: byte → short → int → long → float → double

```
int a = 10;
```

```
double b = a; // automatic conversion
```

Type Casting (Narrowing)

Converting larger type → smaller type.

Done manually because data may be lost.

Syntax: (smallerType) value

```
double x = 10.5;
```

```
int y = (int) x; // manual casting
```

Day-4: Input in Java + Operators

Scanner Class

Used to take input from the user

Present in java.util package.

Must be imported before use.

Common methods:

- `nextInt()` → reads integer
- `nextFloat()` → reads float
- `next()` → reads one word
- `nextLine()` → reads full line

Operators

Symbols used to perform operations on values or variables.

Arithmetic Operators

Used for basic mathematical operations.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (modulus)

Assignment Operators

Used to assign values to variables.

Operator	Meaning
=	Assign value
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulus and assign

Ternary Operator

Short form of if-else.

Syntax:

condition ? value1 : value2

Increment / Decrement Operators

Operator	Meaning
++	Increase value by 1
--	Decrease value by 1

Types: – Pre: ++a – Post: a++

Comparison Operators

Used to compare two values. Result → true/false.

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

Logical Operators

Used to combine conditions.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

Bitwise Operators

Work on bits (0s and 1s).

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<,>>	Left shift, Right Shift

Day-5: Conditional Statements

Conditional statements are used to **make decisions** in a program. They allow the program to **choose different actions** based on conditions (true/false).

if Statement

Used to run a block of code **only if** a condition is true.

If the condition is false, the block is skipped.

```
if (age > 18) {  
    System.out.println("Adult");  
}
```

if-else

Used when you want two possible outcomes.

If condition is true → run if-block.

If condition is false → run else-block.

```
if (marks >= 35) {  
    System.out.println("Pass");  
} else {  
    System.out.println("Fail");  
}
```

else-if

Used when there are multiple conditions.

Checks conditions one by one from top to bottom.

First true condition gets executed.

```
if (score >= 90) {  
    System.out.println("Grade A");  
} else if (score >= 75) {  
    System.out.println("Grade B");  
} else if (score >= 60) {  
    System.out.println("Grade C");  
} else {  
    System.out.println("Grade D");  
}
```

nested if-else

An if-else **inside another if-else**.

Used for checking conditions inside another condition.

Helps in multi-level decision making.

```
if (age >= 18) {  
    if (citizen == true) {  
        System.out.println("Eligible to vote");  
    } else {  
        System.out.println("Not a citizen");  
    }  
} else {  
    System.out.println("Underage");  
}
```

Switch

Used when you want to compare **one value** with **multiple fixed cases**.

Cleaner than writing many else-if statements.

Uses **case**, **break**, and **default**.

```
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("Invalid day");  
}
```

switch vs if-else

Switch

Best for fixed values (1, 2, 3, 'A', 'B', etc.)

Cleaner and easier to read.

Cannot check ranges or complex conditions.

If-else:

Best for ranges (age > 18), comparisons, and complex conditions.

More flexible than switch.

Can handle any type of logical condition.

Day-6: Loops

While

Repeats a block as long as the condition is true.

Condition is checked before the loop runs.

If condition is false at the start → loop will not run.

```
while (condition) {  
    // repeat  
}
```

for Loop

Used when the number of repetitions is **known**.

Has 3 parts: initialization, condition, update.

Most commonly used loop in Java.

```
for (start; condition; update) {  
    // repeat  
}
```

do-while Loop

Runs the block **at least once**.

Condition is checked **after** the loop runs.

Good for menus or user-input based loops.

```
do {  
    // repeat  
} while (condition);
```

Difference

while

- Checks condition first
- May run 0 times

for

- Best when number of iterations is known
- Compact structure (start; condition; update)

do-while

- Runs at least once
- Condition checked after the loop

When to Use What

Use while:

When you don't know how many times the loop should run.

Condition-based loops.

Use for:

When you know the exact number of iterations.

Counting loops (1 to 100, A to Z, etc.)

Use do-while:

When the loop must run at least once.

Menu-driven programs, user-input loops.