**What is Java?**

Java is a **high-level**, **object-oriented**, **platform-independent**, **secure**, and **robust** programming language developed by **James Gosling** at Sun Microsystems (now Oracle) in **1995**.

**Key Features of Java**

| **Feature** | **Description** |
| --- | --- |
| **Object-Oriented** | Everything is an object (except primitives). |
| **Platform Independent** | Write Once, Run Anywhere (WORA) using the JVM. |
| **Simple & Secure** | Syntax is easy, and memory is managed by Garbage Collector. |
| **Robust** | Strong memory management and exception handling. |
| **Multithreaded** | Built-in support for multithreaded applications. |
| **Distributed** | Can work with networks and web-based applications easily. |

### 🔷 Java Architecture (JDK, JRE, JVM)

| **Component** | **Description** |
| --- | --- |
| **JDK (Java Development Kit)** | Includes tools to compile and run Java programs. |
| **JRE (Java Runtime Environment)** | Contains libraries and JVM to run Java code. |
| **JVM (Java Virtual Machine)** | Executes the .class bytecode file on any OS. |

**Flow:**

1. You write .java code ➡️
2. Java Compiler compiles it into .class (bytecode) ➡️
3. JVM interprets and runs it on your machine.

### Your First Java Program

**📄 HelloWorld.java**

java

Copy code

public class HelloWorld {

public static void main(String[] args) {

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

}

}

#### ✅ Explanation:

| **Line** | **What it does** |
| --- | --- |
| public class HelloWorld | Defines a class named HelloWorld. |
| public static void main(String[] args) | Entry point of any Java application. |
| System.out.println(...) | Prints a message to the console. |

**How to Run Java Program (Command Line Method)**

1. Open terminal / command prompt
2. Save your file as: HelloWorld.java
3. Run these commands:

bash

Copy code

javac HelloWorld.java // Compiles the code

java HelloWorld // Runs the bytecode

### 🔐 Java Keywords Used

* public: Accessible from anywhere
* class: Defines a class
* static: Method belongs to the class, not objects
* void: Method doesn’t return anything
* main: Entry point method

**Lesson 2: Variables and Data Types**

### 🔷 What is a Variable?

A **variable** is a name given to a memory location that stores a value. Think of it as a labeled box where you can store data and retrieve it later.

You need to **store** numbers temporarily to perform operations — that’s what variables are for.

#### 🔧 Why?

* You can **store values** in memory with a name (label).
* You can **reuse or update** these values anytime.

### 🔷 Types of Variables in Java

| **Type** | **Example Use** |
| --- | --- |
| **Local** | Inside a method/block |
| **Instance** | Belongs to an object (non-static) |
| **Static** | Belongs to the class (shared among objects) |
| ****Data Types:**** ****To Tell Java What Kind of Data You’re Using**** Java is **statically typed**, meaning every variable **must** have a type defined **before** it can be used. 🔧 Why?  * Helps Java **allocate memory** properly. * Allows the compiler to **catch errors early**. * Optimizes the performance of the application.   **Example:**  int age = 25; // Integer type  String name = "Tom"; // String type  Imagine writing a function to add two numbers. Java needs to **know their types** (like int, float, etc.) to perform the right operation. |  |

### 🔷 Java Data Types

Java has two types of data types:

#### 1. **Primitive Data Types** (8 types)

| **Type** | **Size** | **Example** | **Description** |
| --- | --- | --- | --- |
| byte | 1 byte | byte b = 100; | Small integers (-128 to 127) |
| short | 2 bytes | short s = 1000; | Larger integers |
| int | 4 bytes | int x = 50000; | Default for integers |
| long | 8 bytes | long l = 900000000L; | Huge numbers (add L) |
| float | 4 bytes | float f = 5.75f; | Decimal values (add f) |
| double | 8 bytes | double d = 19.99; | More precise decimal |
| char | 2 bytes | char c = 'A'; | Single character |
| boolean | 1 bit | boolean isTrue = true; | Only true or false |

#### 2. **Non-Primitive Data Types**

* **String**
* **Arrays**
* **Classes, Interfaces, Objects**

**ublic class DataTypesExample {**

**public static void main(String[] args) {**

**int age = 25;**

**double salary = 85643.75;**

**char grade = 'A';**

**boolean isJavaFun = true;**

**String name = "Alice";**

**System.out.println("Name: " + name);**

**System.out.println("Age: " + age);**

**System.out.println("Salary: $" + salary);**

**System.out.println("Grade: " + grade);**

**System.out.println("Loves Java? " + isJavaFun);**

**}**

**}**

**🧾 Output:**

Name: Alice

Age: 25

Salary: $85643.75

Grade: A

Loves Java? True

### ****Casting: When You Need to Switch Between Types****

Sometimes you need to **convert** one type to another — this is why **casting** is used.

**Example:**

int score = 95;

double percentage = score; // Automatically cast to double

This gives you **flexibility** when working with different types.

### 🔁 Type Casting

#### ✅ Implicit Casting (Widening) — safe

int num = 10;

double d = num; // int to double

#### ❌ Explicit Casting (Narrowing) — may lose data

double pi = 3.14;

int intPi = (int) pi; // double to int → 3

### 🔐 Final Keyword (****To Create Constants)****

If you have a value that should **never change**, like π (3.14), Java lets you make it **constant** using final.

**Why?**

* Prevent accidental changes
* Improve readability and maintainability

final int MAX\_USERS = 100;

// MAX\_USERS = 200; // ❌ Error! Final value can't be changed

🧑‍🏫 **Lesson 3: Operators in Java**

### 🔷 What is an Operator?

An **operator** is a special symbol that performs operations on variables and values. Think of them like the verbs of programming — they do actions like adding, comparing, or combining data.

**🔷 Types of Operators in Java**

| **Category** | **Description** | **Examples** |
| --- | --- | --- |
| **Arithmetic** | Perform mathematical operations | +, -, \*, /, % |
| **Relational** | Compare two values | ==, !=, >, <, >=, <= |
| **Logical** | Combine multiple conditions | &&, ` |
| **Assignment** | Assign values to variables | =, +=, -=, \*=, /= |
| **Unary** | Single operand operation | ++, --, +, -, ! |
| **Bitwise** | Operate on bits | &, ` |

## 1️⃣ **Arithmetic Operators**

These operators perform **basic mathematical operations** on numeric data types (int, float, double, etc.).

| **Operator** | **Meaning** | **Description** |
| --- | --- | --- |
| + | Addition | Adds two operands |
| - | Subtraction | Subtracts second operand from first |
| \* | Multiplication | Multiplies two operands |
| / | Division | Divides first operand by second |
| % | Modulus | Returns remainder after division |

### Examples:

java

Copy code

int a = 15;

int b = 4;

System.out.println("a + b = " + (a + b)); // 19

System.out.println("a - b = " + (a - b)); // 11

System.out.println("a \* b = " + (a \* b)); // 60

System.out.println("a / b = " + (a / b)); // 3 (integer division)

System.out.println("a % b = " + (a % b)); // 3 (remainder)

**Note:**

* Integer division truncates decimals. For precise division use double or float:

java

Copy code

double c = 15.0, d = 4.0;

System.out.println(c / d); // 3.75

## 2️⃣ **Relational Operators**

Used to **compare two values** and return a boolean result (true or false).

| **Operator** | **Meaning** | **Description** |
| --- | --- | --- |
| == | Equal to | Checks if two values are equal |
| != | Not equal to | Checks if two values are not equal |
| > | Greater than | Checks if left operand is greater |
| < | Less than | Checks if left operand is smaller |
| >= | Greater than or equal | Checks if left operand is greater or equal |
| <= | Less than or equal | Checks if left operand is less or equal |

### Examples:

int x = 10, y = 20;

System.out.println(x == y); // false

System.out.println(x != y); // true

System.out.println(x > y); // false

System.out.println(x < y); // true

System.out.println(x >= 10); // true

System.out.println(y <= 20); // true

## 3️⃣ **Logical Operators**

Combine multiple **boolean expressions** to form complex logical conditions.

| **Operator** | **Meaning** | **Description** |
| --- | --- | --- |
| && | Logical AND | True if **both** operands true |
| ` |  | ` |
| ! | Logical NOT | Negates the boolean value |

### Examples:

java

Copy code

boolean a = true;

boolean b = false;

System.out.println(a && b); // false (both must be true)

System.out.println(a || b); // true (either true)

System.out.println(!a); // false (negation)

## 4️⃣ **Assignment Operators**

Used to **assign** values to variables. Includes shorthand operators that perform an operation and assignment simultaneously.

| **Operator** | **Meaning** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| = | Assign | a = 5; | Assign 5 to a |
| += | Add and assign | a += 3; | Equivalent to a = a + 3 |
| -= | Subtract and assign | a -= 2; | Equivalent to a = a - 2 |
| \*= | Multiply and assign | a \*= 4; | Equivalent to a = a \* 4 |
| /= | Divide and assign | a /= 2; | Equivalent to a = a / 2 |
| %= | Modulus and assign | a %= 3; | Equivalent to a = a % 3 |

### Examples:

int a = 10;

a += 5; // a = 10 + 5 = 15

System.out.println("a += 5: " + a);

a -= 3; // a = 15 - 3 = 12

System.out.println("a -= 3: " + a);

a \*= 2; // a = 12 \* 2 = 24

System.out.println("a \*= 2: " + a);

a /= 4; // a = 24 / 4 = 6

System.out.println("a /= 4: " + a);

a %= 4; // a = 6 % 4 = 2

System.out.println("a %= 4: " + a);

## 5️⃣ **Unary Operators**

Operate on a **single operand**. Mostly used to increment/decrement values or negate booleans.

| **Operator** | **Meaning** | **Explanation** |
| --- | --- | --- |
| ++ | Increment by 1 | a++ (post-increment), ++a (pre-increment) |
| -- | Decrement by 1 | a-- (post-decrement), --a (pre-decrement) |
| + | Unary plus | Indicates positive value (rarely used explicitly) |
| - | Unary minus | Negates a number |
| ! | Logical NOT | Negates a boolean |

### Examples:

java

int a = 5;

System.out.println(a++); prints 5, then a = 6 (post-increment)

System.out.println(++a); increments to 7, then prints 7 (pre-increment)

System.out.println(a--); // prints 7, then a = 6 (post-decrement)

System.out.println(--a); // decrements to 5, then prints 5 (pre-decrement)

int b = -a; // Unary minus

System.out.println(b); // -5

boolean flag = true;

System.out.println(!flag); // false (logical NOT)

## 6️⃣ **Bitwise Operators** (Less common but important for performance/low-level programming)

Operate on the **bit-level representation** of integers.

| **Operator** | **Meaning** | **Description** |
| --- | --- | --- |
| & | Bitwise AND | Bits set to 1 if both bits are 1 |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | Bits set to 1 if bits are different |
| ~ | Bitwise NOT | Inverts all bits |
| << | Left shift | Shifts bits to the left |
| >> | Right shift | Shifts bits to the right |

### Examples:

int a = 5; // binary: 0101

int b = 3; // binary: 0011

System.out.println(a & b); // 1 (0001)

System.out.println(a | b); // 7 (0111)

System.out.println(a ^ b); // 6 (0110)

System.out.println(~a); // -6 (inverts bits and adds 1 for two’s complement)

System.out.println(a << 1); // 10 (0101 << 1 = 1010)

System.out.println(a >> 1); // 2 (0101 >> 1 = 0010)

**Lesson 4: Control Flow Statements in Java**

## Why Use Control Flow Statements?

In programming, **not all code runs sequentially** — sometimes you want to:

* **Make decisions** (do something only if a condition is true)
* **Repeat tasks** multiple times (loops)
* Choose between multiple alternatives

Control flow statements help you **control the path your program takes**.

## 1️⃣ **If Statement**

The if statement executes a block of code only if a given condition is true.

**Syntax:**

java

Copy code

if (condition) {

// code to execute if condition is true

}

**Example:**

java

Copy code

int age = 18;

if (age >= 18) {

System.out.println("You are an adult.");

}

## 2️⃣ **If-Else Statement**

If the condition is true, execute the first block; otherwise, execute the else block.

**Syntax:**

java

Copy code

if (condition) {

// code if true

} else {

// code if false

}

**Example:**

java

Copy code

int age = 16;

if (age >= 18) {

System.out.println("You are an adult.");

} else {

System.out.println("You are a minor.");

}

## 3️⃣ **Else-If Ladder**

Used to check multiple conditions in sequence.

**Syntax:**

java

Copy code

if (condition1) {

// code block 1

} else if (condition2) {

// code block 2

} else {

// default code

}

**Example:**

java

Copy code

int score = 85;

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 F");

}

## 4️⃣ **Switch Statement**

Used when you want to compare a variable against many constant values.

**Syntax:**

java

Copy code

switch (variable) {

case value1:

// code block

break;

case value2:

// code block

break;

default:

// default code block

}

**Example:**

java

Copy code

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("Invalid day");

}

## 5️⃣ **While Loop**

Repeats a block of code **while** a condition is true.

**Syntax:**

java

Copy code

while (condition) {

// code to repeat

}

**Example:**

java

Copy code

int count = 1;

while (count <= 5) {

System.out.println("Count: " + count);

count++;

}

## 6️⃣ **Do-While Loop**

Similar to while but guarantees the loop runs **at least once** because condition is checked after the loop body.

**Syntax:**

java

Copy code

do {

// code to repeat

} while (condition);

**Example:**

java

Copy code

int count = 1;

do {

System.out.println("Count: " + count);

count++;

} while (count <= 5);

## 7️⃣ **For Loop**

Repeats a block of code a **fixed number of times**. It has initialization, condition, and update parts.

**Syntax:**

java

Copy code

for (initialization; condition; update) {

// code to repeat

}

**Example:**

java

Copy code

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

System.out.println("i = " + i);

}

## 8️⃣ **Break and Continue**

* break **exits** the loop or switch immediately.
* continue **skips the current iteration** and goes to the next.

**Example with break:**

java

Copy code

for (int i = 1; i <= 10; i++) {

if (i == 5) {

break; // exit loop when i is 5

}

System.out.println(i);

}

**Example with continue:**

java

Copy code

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

if (i == 3) {

continue; // skip printing when i is 3

}

System.out.println(i);

}

## Summary Table

| **Control Flow Statement** | **Description** |
| --- | --- |
| if | Executes code if condition is true |
| if-else | Executes code for true or false condition |
| else-if ladder | Checks multiple conditions sequentially |
| switch | Selects code block based on value |
| while | Loop while condition is true |
| do-while | Loop runs at least once, then checks condition |
| for | Loop with initialization, condition, update |
| break | Exits current loop or switch |
| continue | Skips current loop iteration |

# Practice Exercises: Control Flow Statements in Java

### 1. Even or Odd

Write a program that takes an integer and prints "Even" if the number is even, and "Odd" if it’s odd.

### 2. Largest of Three Numbers

Write a program that takes three integers and prints the largest number using if-else statements.

### 3. Grade Calculator

Write a program that takes a score (0-100) and prints the grade based on:

* 90 and above: Grade A
* 80 to 89: Grade B
* 70 to 79: Grade C
* Below 70: Grade F

Use else-if ladder.

### 4. Day of the Week

Write a program that takes an integer (1-7) and prints the day of the week using a switch statement. If the number is outside 1-7, print "Invalid day".

### 5. Print Numbers 1 to 10 Using a While Loop

Use a while loop to print numbers from 1 to 10.

### 6. Sum of First N Natural Numbers Using For Loop

Write a program to calculate the sum of first N natural numbers where N is a positive integer input.

### 7. Print Multiples of 3 Between 1 to 20

Use a for loop and continue statement to print multiples of 3 from 1 to 20.

### 8. Exit Loop on Condition

Write a loop that prints numbers from 1 to 10 but exits the loop when the number reaches 6 using a break statement.

**import java.util.Scanner;**

**public class ControlFlowPractice {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**// 1. IF-ELSE IF-ELSE Example**

**System.out.print("Enter your score (0-100): ");**

**int score = scanner.nextInt();**

**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: F");**

**}**

**// 2. SWITCH Example**

**System.out.print("Enter day number (1-7): ");**

**int day = scanner.nextInt();**

**switch (day) {**

**case 1: System.out.println("Monday"); break;**

**case 2: System.out.println("Tuesday"); break;**

**case 3: System.out.println("Wednesday"); break;**

**case 4: System.out.println("Thursday"); break;**

**case 5: System.out.println("Friday"); break;**

**case 6: System.out.println("Saturday"); break;**

**case 7: System.out.println("Sunday"); break;**

**default: System.out.println("Invalid day");**

**}**

**// 3. FOR LOOP – Print even numbers from 1 to 20**

**System.out.println("\nEven numbers from 1 to 20:");**

**for (int i = 1; i <= 20; i++) {**

**if (i % 2 == 0) {**

**System.out.print(i + " ");**

**}**

**}**

**// 4. WHILE LOOP – Print numbers until user enters 0**

**System.out.println("\n\nEnter numbers (0 to stop):");**

**int num;**

**while (true) {**

**num = scanner.nextInt();**

**if (num == 0) {**

**break; // Exit the loop**

**}**

**System.out.println("You entered: " + num);**

**}**

**// 5. DO-WHILE LOOP – At least run once**

**int secret = 5;**

**int guess;**

**do {**

**System.out.print("Guess the secret number (1-10): ");**

**guess = scanner.nextInt();**

**} while (guess != secret);**

**System.out.println("Correct! 🎉");**

**// 6. CONTINUE Statement – Skip multiples of 3 from 1 to 10**

**System.out.println("\nNumbers from 1 to 10 (excluding multiples of 3):");**

**for (int i = 1; i <= 10; i++) {**

**if (i % 3 == 0) {**

**continue;**

**}**

**System.out.print(i + " ");**

**}**

**scanner.close();**

**}**

**}**

**Lesson 5: Arrays in Java**

## ✅ What is an Array?

An **array** is a **fixed-size**, **indexed** collection of **homogeneous elements** (same data type), stored in **contiguous memory locations**.

### Key Features:

* Zero-based indexing (arr[0] is the first element)
* Fixed length (cannot resize after declaration)
* Efficient access using index
* Stored in heap memory

## 🧠 Why Use Arrays?

Arrays are useful when:

* You want to store **multiple values** of the same type.
* You want **fast access** using index.
* You want to perform operations like searching, sorting, or iteration efficiently.

## 📦 Declaring and Creating Arrays

### Syntax:

java

Copy code

datatype[] arrayName = new datatype[size];

### Example:

java

Copy code

int[] numbers = new int[5]; // Array of 5 integers

You can also declare and initialize at the same time:

java

Copy code

int[] numbers = {10, 20, 30, 40, 50};

## 📄 Accessing Elements

java

Copy code

System.out.println(numbers[0]); // Prints 10

numbers[2] = 99; // Updates index 2

## 🔁 Traversing Arrays

### Using for loop:

java

Copy code

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

System.out.println(numbers[i]);

}

### Using for-each loop:

java

Copy code

for (int num : numbers) {

System.out.println(num);

}

## ⚠️ Common Mistakes

* **ArrayIndexOutOfBoundsException**:  
  Accessing invalid index:

java

Copy code

int[] arr = new int[3];

arr[3] = 10; // ERROR: valid indices are 0, 1, 2

## ✨ Example Program

java

Copy code

public class ArrayExample {

public static void main(String[] args) {

int[] scores = {90, 80, 75, 60, 85};

int total = 0;

for (int score : scores) {

total += score;

}

double average = total / (double) scores.length;

System.out.println("Average score: " + average);

}

}

## Key Methods and Operations (with Examples)

### 1️⃣ ****Accessing and Modifying Elements****

java

Copy code

int[] arr = {10, 20, 30};

System.out.println(arr[0]); // 10

arr[1] = 50;

System.out.println(arr[1]); // 50

### 2️⃣ Arrays.toString() ****– Convert array to readable String****

java

Copy code

import java.util.Arrays;

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

System.out.println(Arrays.toString(arr)); // [1, 2, 3, 4, 5]

### 3️⃣ Arrays.sort() ****– Sort the array in ascending order****

java

Copy code

int[] arr = {5, 2, 8, 1};

Arrays.sort(arr);

System.out.println(Arrays.toString(arr)); // [1, 2, 5, 8]

### 4️⃣ Arrays.copyOf() ****– Copy array to new array of specified length****

java

Copy code

int[] arr = {1, 2, 3};

int[] newArr = Arrays.copyOf(arr, 5); // adds 0s

System.out.println(Arrays.toString(newArr)); // [1, 2, 3, 0, 0]

### 5️⃣ Arrays.copyOfRange() ****– Copy specific range****

java

Copy code

int[] arr = {10, 20, 30, 40, 50};

int[] subArr = Arrays.copyOfRange(arr, 1, 4); // from index 1 to 3

System.out.println(Arrays.toString(subArr)); // [20, 30, 40]

### 6️⃣ Arrays.equals() ****– Compare two arrays (element-wise)****

java

Copy code

int[] a = {1, 2, 3};

int[] b = {1, 2, 3};

System.out.println(Arrays.equals(a, b)); // true

### 7️⃣ Arrays.fill() ****– Fill entire array with a value****

java

Copy code

int[] arr = new int[5];

Arrays.fill(arr, 7);

System.out.println(Arrays.toString(arr)); // [7, 7, 7, 7, 7]

### 8️⃣ Arrays.binarySearch() ****– Search for an element (sorted array only)****

java

Copy code

int[] arr = {1, 3, 5, 7, 9};

int index = Arrays.binarySearch(arr, 5);

System.out.println("Index of 5: " + index); // 2

🔺 **Note**: Array must be sorted before using binarySearch().

### 9️⃣ System.arraycopy() ****– Low-level efficient array copy****

java

Copy code

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

int[] dest = new int[4];

System.arraycopy(src, 0, dest, 0, src.length);

System.out.println(Arrays.toString(dest)); // [1, 2, 3, 4]

### 🔟 ****Multi-dimensional Arrays****

java

Copy code

int[][] matrix = {

{1, 2},

{3, 4}

};

System.out.println(matrix[1][0]); // 3

To print a 2D array:

java

Copy code

for (int[] row : matrix) {

System.out.println(Arrays.toString(row));

}

### 🔁 ****Looping with for-each****

java

Copy code

int[] arr = {10, 20, 30};

for (int value : arr) {

System.out.println(value);

}