

Object-Oriented Programming with Java

Module 1: Fundamentals & Core Concepts





What is Java?

High-Level & Class-Based

Object-oriented programming
language designed for simplicity and
power.

Widely Used

Mobile apps, web applications, and enterprise systems worldwide.

"Write Once, Run Anywhere"

Platform independent — bytecode
runs on any JVM regardless of OS.

Key Features of Java



Platform Independence

Bytecode runs on any JVM regardless of the underlying operating system.

Object-Oriented

Built on encapsulation, inheritance, and polymorphism principles.

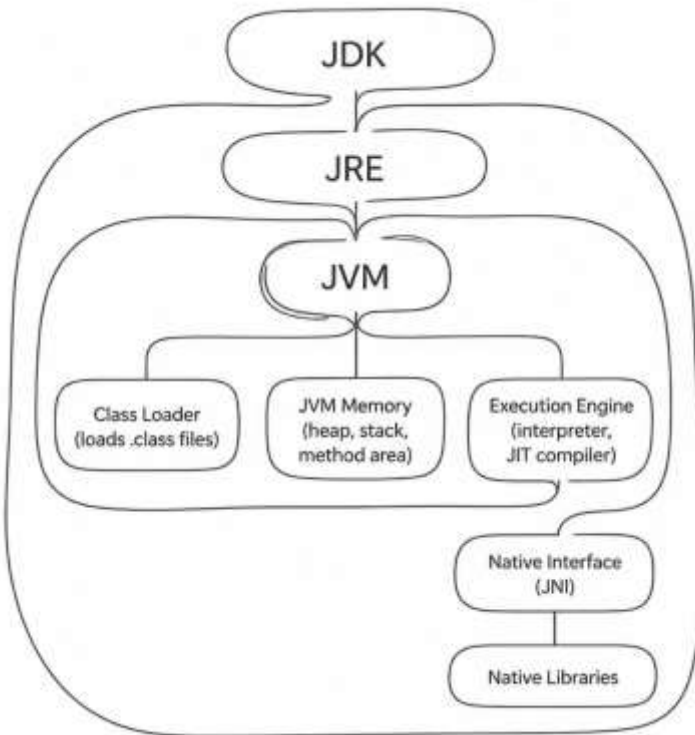
Secure & Robust

Built-in security, exception handling, and automatic garbage collection.

Multithreading

Execute multiple tasks simultaneously for high-performance applications.

Java Architecture



JVM — Java Virtual Machine

Converts bytecode to machine-specific instructions at runtime.

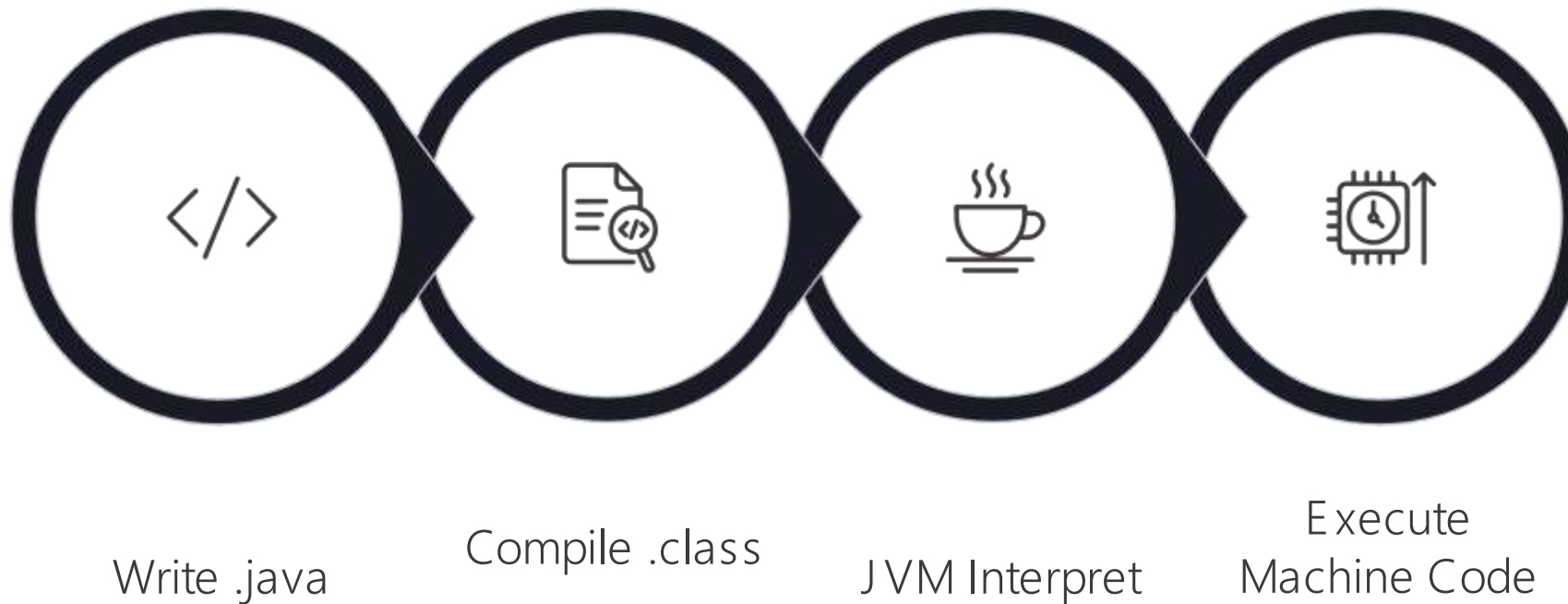
JDK — Java Development Kit

Complete development environment including compiler and tools.

JRE — Java Runtime Environment

Runtime package bundling libraries and the JVM for execution.

Java Compilation & Execution Flow



Java's two-stage compilation process is the secret behind its platform independence — the same bytecode file runs on any machine with a JVM installed.

Two Programming Paradigms

1 Procedural Programming

Step-by-step approach organized around **functions**. The program is a sequence of instructions executed in order.

- Languages: C, Pascal, Fortran
- Focus: How to do it
- Data and functions are separate

2 Object-Oriented Programming

Based on **objects and classes** that bundle data together with the methods that operate on it.

- Languages: Java, C++, Python
- Focus: What acts on what
- Data and behavior are unified

Three Core OOP Principles



Encapsulation

Bundle data and methods together. Restrict direct access using **getters and setters** via public methods.



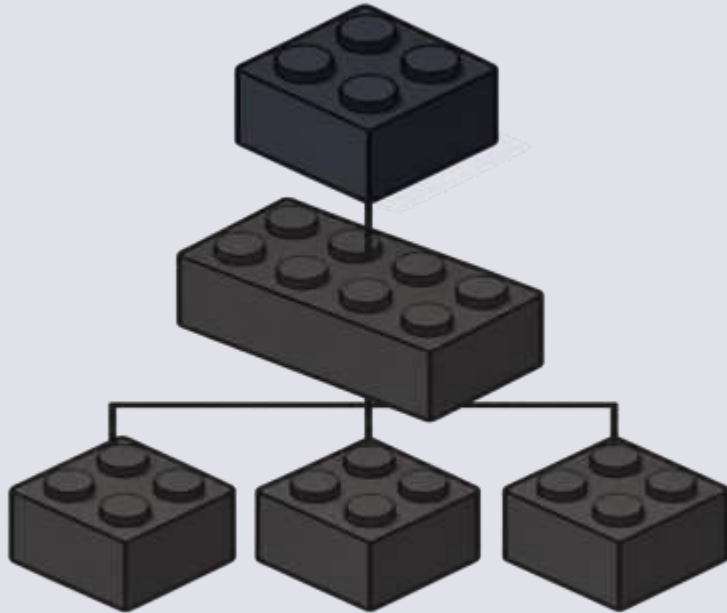
Inheritance

A subclass inherits properties from a superclass, enabling **code reuse** and hierarchical relationships.



Polymorphism

Same method call behaves differently based on object type — via **overloading** or **overriding**.



Blocks of Code in Java

01

Method Block

Code executed when a method is called by the program.

02

Conditional Block

`if, else if, else` — branching logic based on conditions.

03

Loop Block

`for, while, do-while` — repeat execution of statements.

04

Try-Catch Block

Handles exceptions to keep the program running gracefully.



Definition: Code inside a method executes only when the method is called.

Example:

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

Output: Hello

Conditional Block

- Definition: Executes different blocks based on conditions.
- Example:
 - `int marks = 75;`
 - `if (marks >= 90)`
 - `System.out.println("Grade A");`
 - `else if (marks >= 60)`
 - `System.out.println("Grade B");`
 - `else`
 - `System.out.println("Grade C");`
- Output: Grade B

Loop Block

- Definition: Repeats a block of code multiple times.
- Example:
- ```
for (int i = 1; i <= 5; i++) {
```
- ```
    System.out.println(i);
```
- ```
}
```
- Output:
- 1 2 3 4 5

# Try-Catch Block

- Definition: Handles exceptions and prevents abnormal program termination.
- Example:
- ```
try {
```
- ```
 int a = 10 / 0;
```
- ```
} catch (ArithmeticException e) {
```
- ```
 System.out.println("Cannot divide by zero");
```
- ```
}
```
- Output: Cannot divide by zero

Lexical Issues: Identifiers & Keywords

Identifiers

Names for variables, methods, and classes in Java.

- Must start with a letter, `_`, or `$`
- No spaces or reserved keywords allowed
- Case-sensitive: `myVar` $\neq$ `MyVar`

⚠ Common errors: starting with digits, using keywords as names, or including invalid characters.

Reserved Keywords

Words with predefined meaning — cannot be used as identifiers.

`class`

`int`

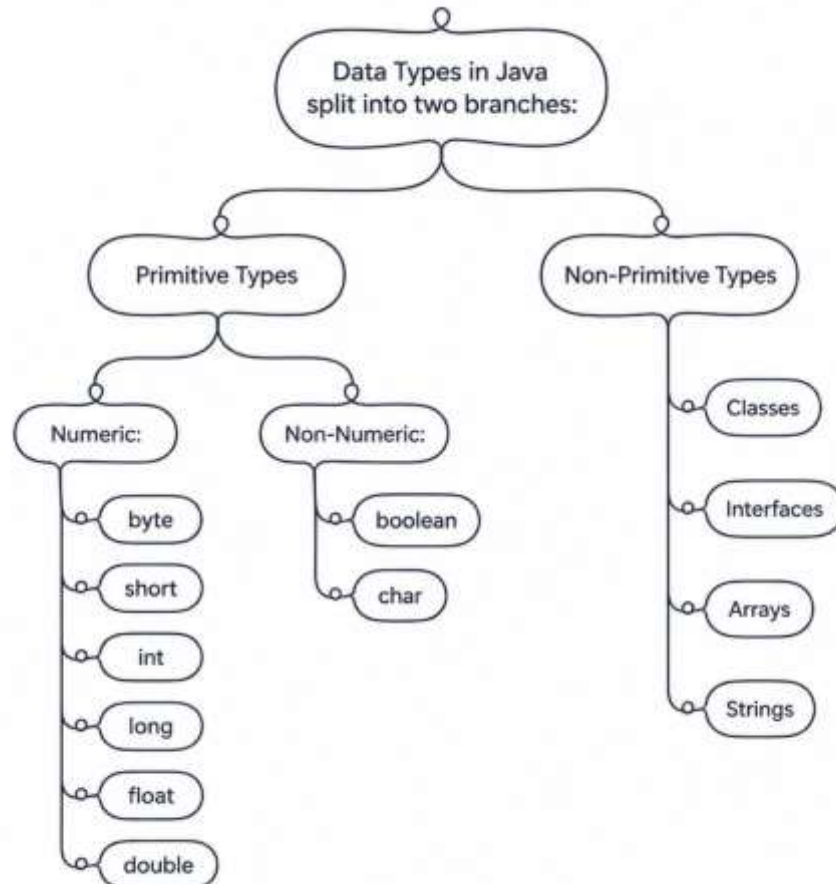
`if`

`for`

`while`

`return`

Data Types in Java



Primitive Types

8 built-in types that store simple values directly in memory:

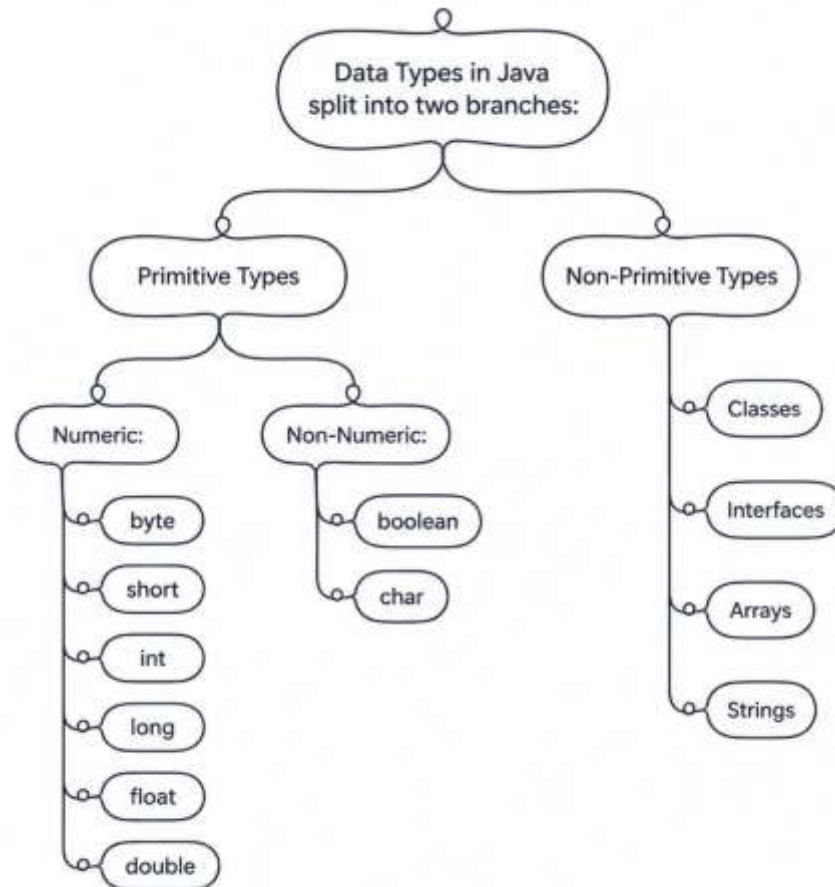
- Integer: byte, short, int, long
- Floating-point: float, double
- Character: char
- Boolean: boolean

Non-Primitive Types

Reference types that point to objects in memory:

- Classes, Interfaces, Arrays, Strings

Data Types in Java



Primitive Types

8 built-in types that store simple values directly in memory:

- Integer: byte, short, int, long
- Floating-point: float, double
- Character: char
- Boolean: boolean

Non-Primitive Types

Reference types that point to objects in memory:

- Classes, Interfaces, Arrays, Strings

Floating-Point Type Examples

- `float temperature = 36.5f;`
- `double pi = 3.1415926535;`
- `System.out.println(temperature);`
- `System.out.println(pi);`
- Output: 36.5
- 3.1415926535

Character and Boolean Examples

- `char grade = 'A';`
- `boolean isPassed = true;`
- `System.out.println(grade);`
- `System.out.println(isPassed);`
- Output: A
- true

Non-Primitive Data Types

- Reference types that store the address of objects.
- Examples: String, Array, Class, Interface

String Example

- `String name = "Divya";`
- `System.out.println(name);`
- Output: Divya

Array Example

- `int[] numbers = {10, 20, 30};`
- `System.out.println(numbers[1]);`
- Output: 20

Variables

A **variable** is a named memory location used to store data.

Syntax

dataType variableName = value;

Example Program

```
public class VariablesDemo { public static void main(String[] args) {  
String name = "Ravi"; int marks = 90;  
    System.out.println("Name: " + name);  
    System.out.println("Marks: " + marks); } }
```

Type Conversion and Casting

Widening Conversion (Automatic)

A smaller data type is converted into a larger data type automatically

```
public class WideningDemo { public static void main(String[] args) {  
    int x = 10; double y = x;  
    System.out.println(y); } }
```

Output

10.0

Narrowing Conversion (Explicit Casting)

A larger data type is converted into a smaller data type manually.

```
public class NarrowingDemo {  
    public static void main(String[] args) {  
        double x = 9.78; int y = (int) x;  
        System.out.println(y); } }
```

Output

9

Automatic Type Promotion in Expressions

During arithmetic operations, **byte**, **short**, and **char** are automatically promoted to **int**.

Example Program

```
public class PromotionDemo {  
    public static void main(String[] args) {  
        byte a = 10;  
        byte b = 20;  
        int c = a + b;  
        System.out.println(c); } }
```

Promotion order:

byte → short → int → long → float → double

Output

30

Arrays

An **array** is a collection of elements of the same type.

Declaration

```
int[] arr;
```

Initialization

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

```
public class ArrayDemo {  
    public static void main(String[] args) {  
        int[] arr = {10, 20, 30, 40, 50};  
        for (int i = 0; i < arr.length; i++) {  
            System.out.println(arr[i]);  
        }  
    }  
}
```

Output

10 20 30 40 50

Operators in Java

Arithmetic Operators

Operator

+

Description

Addition

-

Subtraction

*

Multiplication

/

Division

%

Modulus

Relational Operators

```
public class ArithmeticDemo {  
    public static void main(String[] args) {  
        int a = 10, b = 3;  
        System.out.println(a + b);  
        System.out.println(a - b);  
        System.out.println(a * b);  
        System.out.println(a / b);  
        System.out.println(a % b); } }
```

Operator

==

!=

> 13 7 30 3 1

<

>=

<=

Description

Equal to

Not equal to

Greater than

Less than

Greater than or equal

Less than or equal

OUTPUT

13 7 30 3 1

Relational Operators

Operator

==

!=

>

<

>=

<=

Description

Equal to

Not equal to

Greater than

Less than

Greater than or equal

Less than or equal

```
public class RelationalDemo {  
    public static void main(String[] args) {  
        int x = 10, y = 20;  
        System.out.println(x < y);  
        System.out.println(x == y); } }
```

Output

true

false

Boolean Logical Operators

```
public class LogicalDemo {  
    public static void main(String[] args) {  
        boolean a = true;  
        boolean b = false;  
        System.out.println(a && b);  
        System.out.println(a || b);  
        System.out.println(!a); } }
```

Output

False
true
false

Assignment Operator

Operator

=

+=

-=

*=

/=

Example

x = 10

x += 5

x -= 2

x *= 3

x /= 2

```
public class AssignmentDemo {  
    public static void main(String[] args) {  
int x = 10;  
x += 5;  
x *= 2;  
System.out.println(x); } }
```

Output

30

Ternary Operator (? :)

A shorthand form of the **if-else** statement.

Syntax

condition ? value1 : value2;

Example Program

```
public class TernaryDemo {  
    public static void main(String[] args) {  
        int a = 10, b = 20;  
        int max = (a > b) ? a : b;  
        System.out.println(max); } }
```

Output

20

Control Statements

if Statement

Executes a block if the condition is true.

Example Program

```
public class IfDemo {  
    public static void main(String[] args) {  
        int age = 18;  
        if (age >= 18) {  
            System.out.println("Eligible to vote"); } } }
```

if-else Statement

Example Program

```
public class IfElseDemo {  
    public static void main(String[] args) {  
        int num = 5;  
        if (num % 2 == 0)  
            System.out.println("Even");  
        else System.out.println("Odd"); } }
```

Traditional switch Statement

Example Program

```
public class SwitchDemo {  
    public static void main(String[] args) {  
        int day = 2;  
        switch (day) {  
            case 1: System.out.println("Monday");  
            break;  
            case 2: System.out.println("Tuesday");  
            break;  
            default: System.out.println("Invalid day"); } } }
```

Iteration Statements

while Loop

Example Program

```
public class WhileDemo {  
    public static void main(String[] args) {  
        int i = 1;  
        while (i <= 5) {  
            System.out.println(i); i++; } } }
```

do-while Loop

Example Program

```
public class DoWhileDemo {  
    public static void main(String[] args) {  
        int i = 1;  
        do {  
            System.out.println(i);  
            i++;  
        } while (i <= 5);  
    } }
```

for Loop

Example Program

```
public class ForDemo {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

For-Each Loop

Used to traverse arrays.

Example Program

```
public class ForEachDemo {  
    public static void main(String[] args) {  
        int[] arr = {10, 20, 30};  
        for (int x : arr) {  
            System.out.println(x);  
        }  
    }  
}
```

Local Variable Type Inference in a for Loop

Java allows **var** for local variable inference.

Example Program

```
public class VarForDemo {  
    public static void main(String[] args) {  
        for (var i = 1; i <= 5; i++) {  
            System.out.println(i); } } }
```

Nested Loops

A loop inside another loop.

Example Program

```
public class NestedLoopDemo {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 3; i++) {  
            for (int j = 1; j <= 3; j++) {  
                System.out.print(j + " ");  
                System.out.println(); } } }
```

Jump Statements

break Statement

Terminates the loop immediately.

Example Program

```
public class BreakDemo {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 5; i++) {  
            if (i == 3)  
                break;  
            System.out.println(i); } } }
```

continue Statement

Skips the current iteration.

Example Program

```
public class ContinueDemo {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 5; i++) {  
            if (i == 3)  
                continue;  
            System.out.println(i); } } }
```

return Statement

Exits from a method.

Example Program

```
public class ReturnDemo {  
    static int square(int n) { return n * n; }  
    public static void main(String[] args) {  
        System.out.println(square(5)); } }
```



A T M E
College of Engineering

GOLD

QS I-GAUGE
INDEPENDENT RATING



THANK YOU