# Java Notes in Easy Language

**✅ Unit 1: Basics of Java**

**🔹 1. What is Java?**

* Java is a **high-level, object-oriented** programming language.
* Developed by **Sun Microsystems** in 1995, now owned by **Oracle**.
* Java is **platform-independent** due to the **JVM (Java Virtual Machine)**.

**🔹 2. Java Terminologies**

| **Term** | **Description** |
| --- | --- |
| **JDK** (Java Development Kit) | Contains tools to write and run Java programs (compiler + JRE). |
| **JRE** (Java Runtime Environment) | Contains JVM + libraries to run Java programs. |
| **JVM** (Java Virtual Machine) | Runs compiled .class bytecode on any OS. |

**🔹 3. Features of Java**

* **Simple** – Easy to learn, similar to C++
* **Object-Oriented** – Based on classes and objects
* **Platform Independent** – Runs on any OS with JVM
* **Secure** – No pointer, has built-in security
* **Robust** – Handles errors via exceptions
* **Multithreaded** – Can run multiple tasks simultaneously

**🔹 4. Java Program Structure**

public class HelloWorld {

public static void main(String[] args) {

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

}

}

**✏️ Explanation:**

* public class HelloWorld: Declares a class named HelloWorld.
* public static void main(String[] args): Entry point of the program.
* System.out.println: Used to print output.

**🔹 5. Compilation & Execution**

Steps to run a Java program:

javac HelloWorld.java // Compiles to HelloWorld.class

java HelloWorld // Runs the program

**🔹 6. Comments in Java**

* **Single-line**: // This is a comment
* **Multi-line**:
* /\* This is a
* multi-line comment \*/

**Unit 2: Data Types and Control Flow** in Java

**✅ 1. Data Types**

* **Primitive Types**:
  + int – integer values (e.g., 5)
  + float – single-precision floating-point (e.g., 3.14f)
  + double – double-precision floating-point (e.g., 3.14159)
  + char – single character (e.g., 'A')
  + boolean – true/false
* **Non-Primitive**:
  + String – a sequence of characters (e.g., "Hello")

**✅ 2. Type Casting**

* **Implicit (Widening)**: smaller type → larger type (e.g., int to double)
* **Explicit (Narrowing)**: larger type → smaller type (e.g., double to int using (int))

**✅ 3. Variables and Constants**

* **Variables**: Containers for storing data (e.g., int x = 10;)
* **Constants**: Immutable values, declared with final (e.g., final double PI = 3.14;)

**✅ 4. Operators**

* **Arithmetic**: +, -, \*, /, %
* **Relational**: ==, !=, <, >, <=, >=
* **Logical**: &&, ||, !
* **Assignment**: =, +=, -=, etc.

**✅ 5. Input and Output**

* Use the Scanner class:
* Scanner sc = new Scanner(System.in);
* int num = sc.nextInt();
* String text = sc.nextLine();

**✅ 6. Decision Making**

**if Statement**

if (condition) {

// code

}

**if-else Statement**

if (condition) {

// if block

} else {

// else block

}

**else-if Ladder**

if (cond1) {

} else if (cond2) {

} else {

}

**switch-case**

switch (variable) {

case value1:

// code

break;

default:

// default code

}

**✅ 7. Loops**

**for loop**

for (int i = 0; i < 5; i++) {

// code

}

**while loop**

while (condition) {

// code

}

**do-while loop**

do {

// code

} while (condition);

**✅ 8. break and continue**

* **break**: exits the loop
* **continue**: skips current iteration

**Unit 3: Object-Oriented Programming (OOP) Concepts in Java**

**✅ Unit 3: Object-Oriented Programming (OOP)**

**1. Classes and Objects**

* **Class**: A blueprint for creating objects. Contains fields (variables) and methods.
* class Car {
* String color;
* void drive() {
* System.out.println("Car is driving");
* }
* }
* **Object**: An instance of a class.
* Car myCar = new Car();
* myCar.drive();

**2. Constructors**

* Used to initialize objects.
* **Default Constructor** (no parameters)
* **Parameterized Constructor** (takes arguments)
* class Person {
* String name;
* Person(String n) {
* name = n;
* }
* }

**3. this Keyword**

* Refers to the current object.
* class Student {
* String name;
* Student(String name) {
* this.name = name;
* }
* }

**4. Inheritance**

* One class (child) inherits from another (parent).
* class Animal {
* void sound() {
* System.out.println("Animal sound");
* }
* }
* class Dog extends Animal {
* void bark() {
* System.out.println("Bark");
* }
* }

**5. Method Overloading and Overriding**

* **Overloading**: Same method name, different parameters (within same class).
* void add(int a, int b) {}
* void add(double a, double b) {}
* **Overriding**: Subclass provides a specific implementation of a superclass method.
* @Override
* void sound() {
* System.out.println("Dog barks");
* }

**6. Encapsulation**

* Wrapping data (variables) and methods into a single unit.
* Use private for data, and provide get/set methods.
* class Account {
* private int balance;
* public void setBalance(int b) { balance = b; }
* public int getBalance() { return balance; }
* }

**7. Abstraction**

* Hiding internal details and showing only functionality.
* Use **abstract class** or **interface**.

**Abstract Class**

abstract class Shape {

abstract void draw();

}

class Circle extends Shape {

void draw() {

System.out.println("Drawing circle");

}

}

**Interface**

interface Printable {

void print();

}

class Document implements Printable {

public void print() {

System.out.println("Printing document");

}

}

**8. Polymorphism**

* Same action performed differently depending on the object.

**Compile-time (Method Overloading)**

**Run-time (Method Overriding)**

**✅ Unit 5: Exception Handling and File Handling**

**1. Exception Handling**

* **Exception**: An unwanted event that disrupts program flow.
* Java uses a **try-catch-finally** mechanism.

**2. try-catch Block**

try {

int result = 10 / 0; // May throw exception

} catch (ArithmeticException e) {

System.out.println("Cannot divide by zero!");

}

**3. finally Block**

* Always executes (whether exception occurs or not).

finally {

System.out.println("Always executes");

}

**4. Multiple catch Blocks**

try {

int arr[] = new int[5];

arr[10] = 20;

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index error");

} catch (Exception e) {

System.out.println("Other error");

}

**5. throw Keyword**

* Used to explicitly throw an exception.

throw new ArithmeticException("Custom error");

**6. throws Keyword**

* Declares exceptions a method might throw.

void check(int age) throws ArithmeticException {

if (age < 18)

throw new ArithmeticException("Not eligible");

}

**7. Custom Exception**

* User-defined exception by extending Exception class.

class MyException extends Exception {

MyException(String msg) {

super(msg);

}

}

**✅ File Handling in Java**

**1. File Class (java.io.File)**

* Used to create, delete, and check file information.

import java.io.File;

File myFile = new File("data.txt");

if (myFile.exists()) {

System.out.println("File exists");

}

**2. FileWriter (Writing to a File)**

import java.io.FileWriter;

FileWriter writer = new FileWriter("output.txt");

writer.write("Hello File");

writer.close();

**3. FileReader / Scanner (Reading from a File)**

**Using Scanner**

import java.io.File;

import java.util.Scanner;

File file = new File("data.txt");

Scanner sc = new Scanner(file);

while (sc.hasNextLine()) {

System.out.println(sc.nextLine());

}

sc.close();

**Using BufferedReader**

import java.io.BufferedReader;

import java.io.FileReader;

BufferedReader br = new BufferedReader(new FileReader("data.txt"));

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

**4. Handling File Exceptions**

try {

FileWriter writer = new FileWriter("file.txt");

writer.write("Test");

writer.close();

} catch (IOException e) {

System.out.println("An error occurred.");

}

**Common Exceptions in File Handling**

| **Exception** | **Cause** |
| --- | --- |
| IOException | General input/output error |
| FileNotFoundException | File not found |
| SecurityException | No write/read permission |

**✅ Unit 4: Arrays and Strings in Java**

**1. Arrays**

* A collection of elements of the same data type.
* Fixed size, indexed starting from 0.

**1D Array Declaration and Initialization**

int[] numbers = new int[5];

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

**Accessing Elements**

System.out.println(nums[2]); // Output: 3

**Traversing Array**

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

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

}

**2. 2D Arrays (Matrices)**

* Arrays of arrays, like a table with rows and columns.

**Declaration and Initialization**

int[][] matrix = {

{1, 2, 3},

{4, 5, 6}

};

**Accessing Elements**

System.out.println(matrix[1][2]); // Output: 6

**Nested Loop Traversal**

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();

}

**3. String Basics**

* Strings are objects in Java (immutable).
* Can be declared using:
* String s1 = "Hello";
* String s2 = new String("World");

**4. String Methods**

| **Method** | **Description** |
| --- | --- |
| length() | Returns length of string |
| charAt(index) | Returns character at index |
| substring(start, end) | Returns substring |
| equals(str) | Compares values (case-sensitive) |
| equalsIgnoreCase() | Compares values (case-insensitive) |
| compareTo() | Lexicographically compares strings |
| toLowerCase() | Converts to lowercase |
| toUpperCase() | Converts to uppercase |
| concat(str) | Appends one string to another |
| contains(str) | Checks if string contains another |
| replace(old, new) | Replaces character/substring |
| split(regex) | Splits string using a pattern |
| trim() | Removes leading/trailing whitespace |

**Example**

String name = " Java Programming ";

System.out.println(name.trim().toUpperCase()); // Output: JAVA PROGRAMMING

**5. String Comparison**

String a = "Hello";

String b = "hello";

// Value comparison

System.out.println(a.equals(b)); // false

System.out.println(a.equalsIgnoreCase(b)); // true

// Reference comparison (not recommended for content)

System.out.println(a == b); // false

**6. StringBuilder and StringBuffer**

* Used for **mutable** strings.

**StringBuilder Example**

StringBuilder sb = new StringBuilder("Hello");

sb.append(" World");

System.out.println(sb); // Hello World