# JAVA

## Why Java?

1. **Runs Everywhere (Platform Independent) -** Write your program once, and it works on **Windows, Mac, Linux, and even Android phones**.
   *(Thanks to the JVM — Java Virtual Machine.)*

   WORKFLOW – Program code/source code(.java) $\xrightarrow{\text{Compiler}}$ bytecode( .class) $\xrightarrow{\text{Interpreter}}$ Machine code .

2. **Easy to Learn, Hard to Break -** Java has a **clean and readable syntax**, like English sentences. It also **catches errors early** when you compile, so you're less likely to crash programs unexpectedly.

3. **Object-Oriented (Like Building with LEGO) -** Java uses **classes and objects**, which makes programs modular, reusable, and easier to manage — just like building big structures from small LEGO blocks.

4. **Huge Community and Job Opportunities -** Millions of developers use Java. If you get stuck, **someone has already solved it**, and you'll find tutorials, tools, and libraries everywhere.
   *(Also: Java developers are in high demand.)*

5. **Strong, Secure, and Reliable -** Java automatically handles memory cleanup (**garbage collection**) and has strong **built-in security features**. This makes it great for banking, enterprise software, and big business applications. **No manual memory management:** Programmers don't worry about malloc() and free().

   **Example.**

   ```
   class Account {
       String name;
       Account(String name) {
           this.name = name;
       }
   }

   public class Bank {
       public static void main(String[] args) {
           Account a1 = new Account("Alice");
           a1 = new Account("Bob");  // Old "Alice" object is now garbage
       }
   }
   ```

6. **Versatile -** With Java, you can make:
   - Desktop apps
   - Mobile apps (Android)
   - Web applications
   - Games
   - Big enterprise systems

## What is a Java Token?

**A token is the smallest unit of a Java program that the compiler can understand.** When you write Java code, the compiler breaks it into these **tokens** to analyze and execute.

**Types of Java Tokens (with short examples)**

1. **Keywords - Reserved words** with special meaning in Java.
   Example:
   ```
   class Test {
      public static void main(String[] args) { }
   }
   ```
   Here, class, public, static, void are **keywords**.

2. **Identifie**r - **Names** you give to classes, variables, and methods. If variable names also started with digits, the compiler couldn't tell if 123abc is a number or an identifier.
   Example:
   ```
   int age = 25;
   String name = "John";
   ```
   Age, name are **identifiers**.

3. **Literals** - **Constant values** directly written in code.
   Example:
   ```
   int x = 10;        // integer literal
   float y = 3.14f;   // float literal
   char c = 'A';      // char literal
   String s = "Hi";   // string literal

   int b = 0b1010;   // binary literal (prefix 0b)
   int c = 012;      // octal literal (prefix 0)
   int d = 0x1A;     // hexadecimal literal (prefix 0x)
   long e = 123456L; // long literal (suffix L)
   ```

4. **Operators** - **Symbols** that perform operations on variables/values.
   Example:
   ```
   int sum = a + b;   // '+' is an operator
   ```

5. **Separators (or Punctuators) - Symbols that separate code elements.**
   Examples:
   - ;→ ends a statement
   - { } → define a block
   - ( ) → group expressions or parameters
   - , → separates items in a list

## Syntax of Java

```
import java.util.*;

class ASS1eveod
{
   public static void main(String args[])
   {

   }
}
```

**Use case (** *class ASS1eveod { }* **)**

- This class is **package-private** by default.
- It can only be accessed **inside the same package** (i.e., by other classes in the same folder if you don't declare a package explicitly).
- If you don't need the class to be used from outside its package, **this is enough**.
- Used mostly for **helper classes** or internal logic that isn't meant to be called from anywhere else.

```
import java.util.*;

public class ASS1eveod
{
   public static void main(String args[])
   {

   }
}
```

**Use case (** *public class ASS1eveod { }* **)**

- A **public class is visible everywhere**, from any other package or project, as long as it's in the classpath.
- If you make a class public, **the file name MUST match the class name exactly**. Example: if the class is public class ASS1eveod, the file must be saved as ASS1eveod.java.
- Used when you want this class to be **the main entry point** or **accessible from other projects/packages**.

**Use case (** *public static void main(String[] args)* **)**

- **Why public?** JVM can access it from outside the class.
- **Why static?** No object needs to be created to start running code.
- **Why String[] args?** To accept input from the command line.Your code does not need args to run.But the JVM needs to see (String[] args) to know where to begin.
- **Why not void main()?** JVM will not recognize it as the starting point.

**Use case (***Scanner sc1 = new Scanner(System.in);* **)**

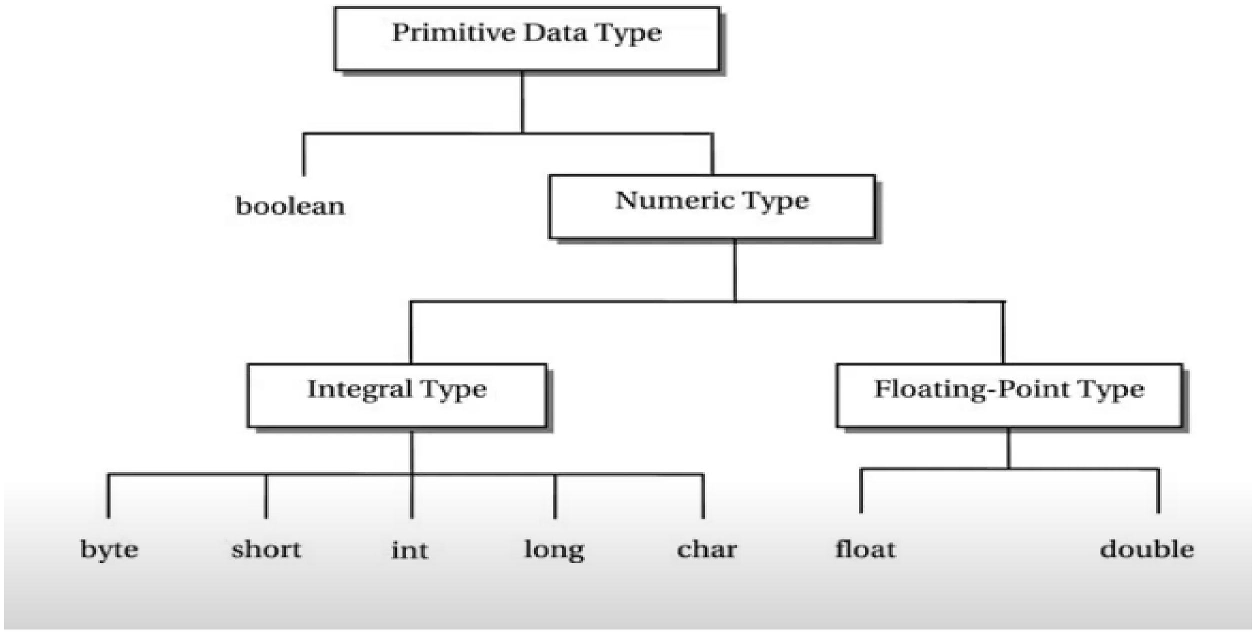
- **Scanner is a class** in the package java.util.
- Scanner → **class name** , sc1 → **variable (reference to the object)**
- System.in **represents standard input stream** — usually the keyboard in console programs. You're telling Scanner: *"Read input from the keyboard stream."*

## Scanner types

- *nextInt(),* *nextDouble(),* *nextFloat(),* *nextBoolean()*
- *next(),* *nextLine() (for Strings)*

## Data Types

- **Primitive** - Primitives store values directly.



1. **byte (-128 to 127 (8-bit signed))**

   *byte b = 100;*

   *System.out.println(b); // Output: 100*

2. **short (-32,768 to 32,767 (16-bit signed))**

   *short s = 30000;*

   *System.out.println(s); // Output: 30000*

3. **int (-2,147,483,648 to 2,147,483,647 (32-bit signed))**

   *int i = 200000;*

   *System.out.println(i); // Output: 200000*

4. **long ( -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (64-bit signed) )**

   *long l = 10000000000L;  // Note the 'L' at the end*

   *System.out.println(l);  // Output: 10000000000*

5. **char ‘ ‘**

   *char c = 'A';*

   *System.out.println(c);     // Output: A*

   *char symbol = 65;          // Unicode value of 'A'*

   *System.out.println(symbol);// Output: A*

6. **float (7 decimal digits precision)**

   *float f = 3.14f;   // 'f' is required to mark it as float*

   *System.out.println(f); // Output: 3.14*

7. **double (15 decimal digits precision)**

   *double d = 3.14159265358979;*

   *System.out.println(d);  // Output: 3.14159265358979*

8. **boolean**

   *boolean isJavaFun = true;*

   *System.out.println(isJavaFun);  // Output: true*

- **Non – Primitive -** Non-primitives store references (addresses) to objects in memory.

1. **Class** – Blueprint for creating objects.
   *Example:* class Student { }
2. **Object** – Instance of a class.
   *Example:* Student s = new Student();
3. **Array** – Fixed-size collection of same type elements.
   *Example:* int[] arr = {1, 2, 3};
4. **Interface** – Contract that defines methods without implementation.
5. **Enum** – Fixed set of constants.
   *Example:* enum Color { RED, GREEN, BLUE }
6. **String** – Immutable object representing text (class type).
   *Example:* String s = "Hello";
7. **Wrapper Classes** – Object versions of primitives (Integer, Double, Boolean, etc.).
   *Example:* Integer x = 10;