Java

🧠 1. Introduction to Java

Java is a high-level, object-oriented, platform-independent, and secure programming language developed by James Gosling at Sun Microsystems in 1995. It is widely used for building desktops, web, mobile, and enterprise applications.

💡 Key Features of Java

Feature	Description
Simple	Easy to learn, especially for those with C/C++ background
Object-Oriented	Everything in Java is an object (except primitives)
Platform-Independent	Write once, run anywhere (WORA) due to Java Virtual Machine (JVM)
Secure	Provides security through its runtime environment and no explicit pointers
Robust	Strong memory management and exception handling
Multithreaded	Supports multi-tasking using threads
Portable	Java bytecode can run on any device with JVM

🛠️ How Java Works

Write code in a .java file.

Compile it using the Java compiler (javac), which produces **bytecode** (.class file).

Run the bytecode on the **Java Virtual Machine (JVM)**.

🔹 JDK (Java Development Kit)

It helps you **write and run** Java programs. It includes tools like the **compiler** and **JVM**.

🔹 JVM (Java Virtual Machine)

It **runs your Java program**. It understands the code and makes it work on any computer.

🔹 IDE (Integrated Development Environment)

A **special app** to write Java code easily. It gives help like auto-complete and one-click run.

Examples: IntelliJ IDEA, Eclipse, NetBeans

🔹 Code Editor

A **simple text editor** for writing code. Not as smart as an IDE.

Examples: VS Code, Notepad++

Feature	**Java**	**C++**
Platform	Runs on any system using JVM	Runs only on the system it's compiled for
Memory	Handles memory automatically	You manage memory yourself (new, delete)
Pointers	Not allowed	Allowed
Speed	A bit slower	Very fast
Use	Mobile apps, websites	Games, system software
Syntax	Easier to learn	More complex

🖨️ How to Print Something in Java

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

psvm
sout
```

🧾 Taking Input from User in Java

✅ Step 1: Import Scanner Class

We need a Scanner to take input from the keyboard.

```
import java.util.Scanner;
```

✅ Step 2: Create Scanner Object

This connects the scanner to the keyboard.

```
Scanner input = new Scanner(System.in);
```

🧪 Full Example:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = input.nextLine();

        System.out.print("Enter your age: ");
        int age = input.nextInt();

        System.out.println("Hello " + name + ", you are " + 
age + " years old.");
    }
}
```

🧠 Variables in Java

✅ What is a Variable?

A **variable** is a name that stores **data** in the computer's **memory (RAM)**.

When you create a variable, Java **reserves memory space** to store its value. The **data type** decides **how much memory** is needed and **what kind of data** can be stored.

🧱 Types of Variables in Java

🔹 1. Primitive Variables (basic built-in types)

Java has **8** primitive data types:

Data Type	Example	Size in Memory	Description
byte	byte a = 10;	1 byte	Small whole number (-128 to 127)
short	short b = 100;	2 bytes	Medium whole number
int	int age = 25;	4 bytes	Common whole number
long	long big = 100000L;	8 bytes	Large whole number
float	float pi = 3.14f;	4 bytes	Decimal with less precision
double	double price = 99.99;	8 bytes	Decimal with high precision
char	char letter = 'A';	2 bytes	Stores one character
boolean	boolean isOn = true;	1 bit	True or false

🔹 2. Non-Primitive Variables (Objects / Reference types)

These store **addresses (references)** in memory instead of actual values.

Type	Example	Description
String	String name = "Alwerad";	Stores text (sequence of chars)
Arrays	int[] numbers = {1, 2, 3};	Stores multiple values
Objects	Student s = new Student();	Stores custom object types

🧠 How Variables Work in Memory

➤ Example:

```
int x = 5;

String name = "Ali";
```

🧠 Conditional Statements in Java

Conditional statements are used to make decisions in your code. They allow the program to execute certain blocks of code based on whether a condition is **true** or **false**.

✅ 1. if Statement

The **if** statement is used to test a condition. If the condition is **true**, the block of code inside the **if** is executed.

```
if (condition) {
    // code to execute if the condition is true
}
Example:
int age = 18;
if (age >= 18) {
    System.out.println("You are an adult.");
}
```

✅ 2. if-else Statement

The **if-else** statement allows you to define an alternative block of code to execute if the condition is **false**.

```
if (condition) {
    // code to execute if true
} else {
    // code to execute if false
}
Example:
int age = 16;
if (age >= 18) {
    System.out.println("You are an adult.");
} else {
    System.out.println("You are a minor.");
}
```

✅ 3. if-else if-else Statement

If you have multiple conditions to test, you can use **else if** to check for more than two conditions.

```
if (condition1) {
    // code for condition1
} else if (condition2) {
    // code for condition2
} else {
    // code if all conditions are false
}
```

Example:

```
int age = 20;
if (age >= 18) {
    System.out.println("You are an adult.");
} else if (age >= 13) {
    System.out.println("You are a teenager.");
} else {
    System.out.println("You are a child.");
}
```

✅ 4. switch Statement

The **switch** statement is used to simplify multiple if-else statements when you have several possible conditions based on a single variable.

```
switch (variable) {
    case value1:
        // code if variable == value1
        break;
    case value2:
        // code if variable == value2
        break;
    default:
        // code if no case matches
}
```

Example:

```
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");
}
```

🧠 Important Notes:

- **if** checks a condition, and if **true**, it executes the block of code.
- **else if** checks additional conditions if the previous ones are **false**.
- **else** provides a fallback block of code when all previous conditions are **false**.
- **switch** is useful when testing a variable against multiple constant values.

🔁 Loops in Java (For Beginners)

Loops are used when we want to **repeat** a block of code **multiple times**.

✅ Types of Loops in Java:

Loop Type	Use Case
for	When you know how many times to loop
while	When you loop while a condition is true
do-while	Like `while`, but runs at least once

🔹 1. for Loop

```
for (initialization; condition; update) {
    // code to repeat
}
```

Example:

```
for (int i = 1; i <= 5; i++) {
    System.out.println("Hello " + i);
}
```

🧠 This prints "Hello" 5 times.

🔹 2. while Loop

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

Example:

```
int i = 1;
while (i <= 5) {
    System.out.println("Hi " + i);
    i++;
}
```

🧠 Runs **as long as** `i <= 5`.

🔷 3. do-while Loop

```
do {
    // code to run
} while (condition);
```

Example:

```
int i = 1;
do {
    System.out.println("Hey " + i);
    i++;
} while (i <= 5);
```

🧠 This loop **runs at least once**, even if the condition is false.

🧠 **Loop Flow Chart (Simple):**

Initialization → Condition → Code → Update → back to Condition...