B.Sc. Computer Science

Unit I

What is Java? (Java क्या है?)

Java एक **High-level, Object-Oriented** और बहुत ही **Popular** programming language है। इसे 1991 में **James Gosling** और उनकी टीम (Sun Microsystems) ने बनाया था।

Java को बनाने का मुख्य मकसद था — **"Write Once, Run Anywhere" (WORA)**। इसका मतलब है कि अगर आपने एक बार Java में code लिख दिया, तो वह किसी भी machine (Windows, Mac, Linux) पर बिना किसी बदलाव के चल सकता है।

Features of Java (जावा की विशेषताएं)

Java के कुछ खास features इसे बाकी languages से अलग बनाते हैं:

1. **Simple:** Java को सीखना आसान है क्योंकि इसका syntax C++ जैसा है, लेकिन इसमें से कठिन चीज़ें (जैसे Pointers) हटा दी गई हैं।
2. **Platform Independent:** Java का code direct machine पर नहीं चलता, बल्कि **Bytecode** में compile होता है। यह Bytecode **JVM (Java Virtual Machine)** की मदद से किसी भी system पर रन हो सकता है।
3. **Secure:** Java में direct pointer access नहीं होता, जिससे viruses और security threats का खतरा कम रहता है।
4. **Robust:** यह एक "मजबूत" language है क्योंकि इसमें memory management (Garbage Collection) और exception handling बहुत अच्छी है।
5. **Object-Oriented (OOPs):** Java पूरी तरह से Objects और Classes पर आधारित है, जिससे बड़े softwares को मैनेज करना आसान हो जाता है।

Uses of Java (Java कहाँ इस्तेमाल होती है?)

- **Mobile Apps:** Android apps Java (या Kotlin) में ही बनती हैं।
- **Web Applications:** बड़ी कंपनियों के servers (जैसे Banking) Java use करते हैं।
- **Big Data:** Hadoop जैसे frameworks Java पर आधारित हैं।
- **Embedded Systems:** Smart cards और electronic devices में।

Java Virtual Machine (JVM)

JVM (Java Virtual Machine) एक engine है जो Java apps को run करने के लिए environment प्रदान करता है। यह Java code (Bytecode) को machine language में convert करता है ताकि processor उसे समझ सके।

सरल शब्दों में, JVM वह जादू है जो Java के **"Write Once, Run Anywhere"** (एक बार लिखो, कहीं भी चलाओ) वादे को पूरा करता है।

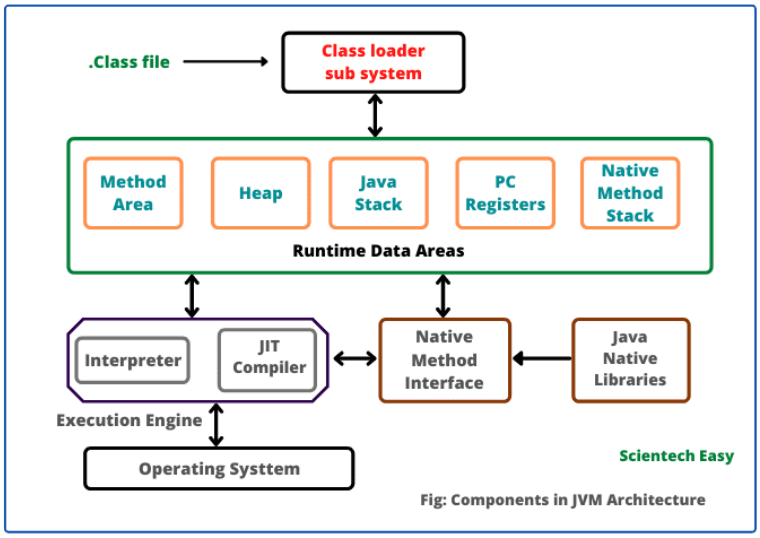

Fig: Components in JVM Architecture

Key Components of JVM (JVM के मुख्य भाग)

JVM के अंदर 3 मुख्य subsystems काम करते हैं:

1. **Class Loader Subsystem:** इसका काम .class files (Bytecode) को memory में load करना है। यह loading, linking और initialization का काम संभालता है।
2. **Runtime Data Areas (Memory Area):** यहाँ Java program का data store होता है। इसके 5 हिस्से होते हैं:
    - **Method Area:** यहाँ class-level data (जैसे static variables) store होता है।
    - **Heap Area:** यहाँ सभी **Objects** store होते हैं। (यह shared memory है)।
    - **Stack Area:** हर thread के लिए अलग stack होता है जहाँ local variables store होते हैं।
    - **PC Registers:** यह याद रखता है कि अभी कौन सी instruction execute हो रही है।
    - **Native Method Stack:** यह Java के अलावा दूसरी languages (जैसे C/C++) के code को handle करता है।
3. **Execution Engine:** यह असली "काम करने वाला" हिस्सा है। इसके अंदर ये चीज़ें होती हैं:
    - **Interpreter:** Code को line-by-line पढ़ता और execute करता है।
    - **JIT (Just-In-Time) Compiler:** जो code बार-बार use होता है, उसे direct machine code में बदल देता है ताकि speed बढ़ सके।

- **Garbage Collector (GC):** यह फालतू Objects को memory से हटा देता है ताकि space खाली हो सके।

JVM Workflow (कैसे काम करता है?)

जब आप java MyProgram command चलाते हैं, तो यह process होता है:

1. **Loading:** Class Loader आपकी class file को ढूँढकर memory में लाता है।
2. **Verification:** JVM चेक करता है कि code सुरक्षित है या नहीं (Bytecode Verifier)।
3. **Execution:** Interpreter और JIT मिलकर code को run करते हैं।

Tokens in Java (जावा में टोकन)

Java में **Tokens** प्रोग्राम की सबसे छोटी इकाई (smallest individual unit) होते हैं। जब हम Java code लिखते हैं, तो compiler पूरे प्रोग्राम को छोटे-छोटे टुकड़ों में तोड़ देता है ताकि उसे समझ सके। इन्हीं टुकड़ों को **Tokens** कहा जाता है।

सरल शब्दों में: जैसे एक वाक्य (sentence) शब्दों से मिलकर बनता है, वैसे ही एक Java program **Tokens** से मिलकर बनता है।

Types of Tokens (टोकन के प्रकार)

Java में मुख्य रूप से 5 तरह के tokens होते हैं:

Token Type	Description	Example
Keywords	Reserved शब्द जिनका मतलब पहले से तय है।	class, public, int, if
Identifiers	Class, method या variable को दिया गया नाम।	Student, rollNo, display()
Literals	Constants या fixed values।	101, "Rahul", 10.5
Operators	Calculations करने के लिए symbols।	+, -, *, =, ==
Separators	Code को अलग करने वाले चिन्ह।	;, ,, { }, ( )

---

1. Keywords (कीवर्ड्स)

ये Java के "Reserved Words" हैं। आप इन्हें variable के नाम के तौर पर इस्तेमाल नहीं कर सकते।

- **Example:** static, void, return, new, while |

2. Identifiers (आइडेंटिफायर्स)

ये वो नाम हैं जो programmer (आप) खुद रखता है।

- **Rules:** ये हमेशा letter, $ या _ से शुरू होने चाहिए। नंबर से शुरू नहीं हो सकते।
- **Example:** int age; (यहाँ age एक identifier है)।

3. Literals (लिटरल्स)

जो values हम variables में store करते हैं, उन्हें literals कहते हैं।

- **Integer Literal:** 50
- **String Literal:** "Java "
- **Character Literal:** 'A'

4. Operators (ऑपरेटर्स)

इनका उपयोग variables और values पर operation करने के लिए किया जाता है।

- **Arithmetic:** +, -, *, /
- **Relational:** >, <, ==, !=

5. Separators/Punctuators (सेपरेटर्स)

ये code के structure को define करते हैं।

- **Semicolon (;):** Statement को खत्म करने के लिए।
- **Braces ({}):** Code block (जैसे class या method) को define करने के लिए।
- **Parentheses (( )):** Methods के लिए।

Example int x = 10 + 20;

यहाँ compiler इसे इन tokens में तोड़ेगा:

1. int ⟶ Keyword
2. x ⟶ Identifier
3. = ⟶ Operator (Assignment)
4. 10 ⟶ Literal
5. + ⟶ Operator (Arithmetic)
6. 20 ⟶ Literal
7. ; ⟶ Separator

Command Line Arguments in Java

Command Line Argument Java में एक ऐसा feature है जिससे हम program को run करते समय ही value/input दे सकते हैं।

जब हम terminal में java FileName लिखते हैं, तो उसके आगे हम जो भी शब्द (words) लिखते हैं, उन्हें ही **Arguments** कहा जाता है। ये arguments main method के String[] args array में store हो जाते हैं।

1. The main Method Connection

Java का program हमेशा यहाँ से शुरू होता है: public static void main(String[] args)

- **String[] args**: यह एक container (array) है जो आपके द्वारा दिए गए inputs को hold करता है।
- **Data Type**: आप command line पर कुछ भी लिखें (10, 20, या "Hello"), Java उसे हमेशा **String** ही मानता है।

---

2. Code Example: Simple Print

This program will take names from the command line and greet the user.

```
public class GreetUser {
    public static void main(String[] args) {
        // Check if the user has provided any argument
        if (args.length > 0) {
            // Accessing the first argument at index 0
            System.out.println("Welcome, " + args[0] + "!");
        } else {
            // Message if no argument is passed
            System.out.println("Please provide a name as an argument.");
        }
    }
}
```

How to Execute (Execution Steps):

1. **Compile**: javac GreetUser.java
2. **Run**: java GreetUser Rahul
3. **Output**: Welcome, Rahul!

---

3. Handling Multiple Arguments

अगर आप एक से ज़्यादा values देते हैं, तो Java उन्हें index (0, 1, 2...) के हिसाब से store करता है।

Example Command: java MyApp Red Green Blue

- args[0] = "Red"
- args[1] = "Green"
- args[2] = "Blue"

---

4. Key Rules (ज़रूरी नियम)

- **Space is the Separator**: Java हर space के बाद वाली चीज़ को एक नया argument मानती है।
- **Converting Data**: अगर आपको calculation करनी है, तो आपको String को number में बदलना होगा (इसे Parsing कहते हैं)।
    - Example: int n = Integer.parseInt(args[0]);
- **Error Prevention**: अगर आप args[0] access करते हैं लेकिन कोई argument नहीं देते, तो program crash हो जाएगा और **ArrayIndexOutOfBoundsException** नाम का error आएगा।

Java and World Wide Web

World Wide Web (WWW) और Java का रिश्ता बहुत पुराना है। Java को मशहूर ही इसलिए किया गया था ताकि static वेब पेज को **Interactive** बनाया जा सके।

- **Applets:** पुराने समय में Java Applets का इस्तेमाल ब्राउज़र के अंदर छोटे प्रोग्राम या गेम्स चलाने के लिए होता था।
- **Platform Independence:** Java का "Write Once, Run Anywhere" (WORA) फीचर इसे इंटरनेट के लिए परफेक्ट बनाता है क्योंकि इंटरनेट पर अलग-अलग तरह के कंप्यूटर्स जुड़े होते हैं।
- **Secure:** वेब पर डेटा भेजने के लिए Java बहुत सुरक्षित मानी जाती है।

Web Browser (वेब ब्राउज़र)

Web Browser एक सॉफ्टवेयर एप्लीकेशन है जिसका उपयोग इंटरनेट से जानकारी (Web Pages) देखने के लिए किया जाता है।

- **Function:** यह यूजर की रिक्वेस्ट को सर्वर तक पहुँचाता है और वहां से HTML/Java कोड लाकर स्क्रीन पर दिखाता है।
- **Examples:** Chrome, Firefox, Safari.
- **Java's Role:** ब्राउज़र में Java Runtime Environment (JRE) का सपोर्ट होता है जो वेब-आधारित जावा प्रोग्राम्स को चलाने में मदद करता है।

Hardware and Software Requirements (ज़रूरतें)

Java को अपने कंप्यूटर पर चलाने के लिए आपको कुछ चीज़ों की ज़रूरत होती है:

1. Hardware Requirements (हार्डवेयर)

Component	Minimum Requirement	Recommended
Processor	Pentium 2 (या उससे ऊपर)	Core i3 / i5 / Ryzen
RAM	2 GB	8 GB or more
Disk Space	500 MB free space	2 GB free space

2. Software Requirements (सॉफ्टवेयर)

- **JDK (Java Development Kit):** यह सबसे ज़रूरी सॉफ्टवेयर है। इसके बिना आप जावा प्रोग्राम **लिख (Code)** नहीं सकते।

- **JRE (Java Runtime Environment):** यह जावा प्रोग्राम को **चलाने (Run)** के लिए ज़रूरी है। (यह JDK के साथ ही आता है)।
- **Operating System:** Windows, macOS, या Linux।
- **Text Editor/IDE:** कोड लिखने के लिए Notepad, VS Code, या IntelliJ IDEA।

Java Environment

Java प्रोग्राम को लिखने और चलाने के लिए जो पूरा सिस्टम या सेटअप चाहिए होता है, उसे ही **Java Environment** कहते हैं। इसमें मुख्य रूप से तीन चीजें शामिल होती हैं: **JDK, JRE, और JVM**।

इन तीनों के बिना जावा पर काम करना नामुमकिन है। आइए इन्हें गहराई से समझते हैं।

---

1. JVM (Java Virtual Machine)

JVM जावा का सबसे महत्वपूर्ण हिस्सा है। इसे "जावा का दिल" भी कहा जा सकता है।

- **Role:** यह आपके लिखे हुए Java code (Bytecode) को मशीन कोड में बदलता है ताकि कंप्यूटर उसे समझ सके।
- **Platform Independence:** JVM की वजह से ही जावा कोड किसी भी OS (Windows, Linux, Mac) पर चल सकता है।
- **Tasks:** कोड को लोड करना, वेरीफाई करना और एक्सीक्यूट (Execute) करना।

---

2. JRE (Java Runtime Environment)

JRE एक ऐसा पैकेज है जो जावा प्रोग्राम को **चलाने (Run)** के लिए ज़रूरी माहौल देता है।

- **Components:** इसमें JVM और कुछ Libraries (जैसे util, lang, math) शामिल होती हैं।
- **Usage:** अगर आप डेवलपर नहीं हैं और सिर्फ अपने कंप्यूटर पर कोई जावा गेम या सॉफ्टवेयर चलाना चाहते हैं, तो सिर्फ JRE काफी है।
- **Note:** JRE में कंपाइलर (javac) नहीं होता, इसलिए इसमें आप नया कोड लिख या कंपाइल नहीं कर सकते।

---

3. JDK (Java Development Kit)

JDK एक पूरा सॉफ्टवेयर डेवलपमेंट किट है जिसका इस्तेमाल जावा एप्लीकेशन **बनाने (Develop)** के लिए होता है।

- **Components:** JDK = JRE + Development Tools (जैसे javac, debugger, javadoc).
- **Usage:** अगर आप एक प्रोग्रामर हैं, तो आपको अपने सिस्टम में JDK इंस्टॉल करना ही होगा।
- **Installation:** जब आप JDK इंस्टॉल करते हैं, तो JRE और JVM अपने आप इंस्टॉल हो जाते हैं।

Comparison Table: Understanding the Difference

Feature	JVM	JRE	JDK
Full Form	Java Virtual Machine	Java Runtime Environment	Java Development Kit
Main Job	Bytecode को रन करना।	प्रोग्राम चलाने के लिए माहौल देना।	कोड को लिखना और कंपाइल करना।
Contains	Interpreter + JIT Compiler	JVM + Libraries	JRE + Tools (javac, etc.)
Target User	Hardware/OS interface	Users (जो सिर्फ ऐप चलाना चाहते हैं)	Developers (जो कोड लिखना चाहते हैं)