**Java Introduction**

**Java** is a popular, high-level programming language known for its simplicity, versatility, and cross-platform capabilities.

Created by Sun Microsystems in 1995, it has since become a fundamental language in software development.

In 2010, **Oracle Corporation** acquired Sun Microsystems now its called **“Oracle Java”**.

* Java follows the **"write once, run anywhere"** meaning programs written in J**ava can run on any device** that supports the **Java Virtual Machine (JVM).**

Why java is more popular:

**Java Features:**

* **Platform Independent:** Java programs can run on any device that has the Java Virtual Machine (JVM).
* **Object-Oriented:** Java uses objects and classes to organize and structure code.
* **Simple and Easy:** Java is designed to be easy to learn and use.
* **Secure:** Java has built-in security features to protect applications.
* **Robust:** Java handles errors well, making it reliable for big applications.
* **Multithreading:** Java allows multiple tasks to run at the same time, improving performance.
* **High Performance:** Java's performance is good thanks to the Just-In-Time (JIT) compiler.
* **Huge Library:** Java has a large collection of pre-built libraries and tools for various tasks.
* **Huge community:problem solving by communities .**

1. **Quick Help:** If you face a problem, chances are someone else has faced it too. You can find answers quickly on platforms like Stack Overflow, GitHub, and Reddit.
2. **Lots of Tutorials:** There are thousands of tutorials, articles, and guides created by the Java community to help you learn and solve problems.
3. **Open-Source Libraries:** The community has created and shared many open-source libraries and frameworks, which can save you time when developing your applications.
4. **Regular Updates:** Java's large community ensures that the language is frequently updated with new features and bug fixes.

### **Applications of Java:**

### **Web Development:** Using frameworks like Spring and Hibernate.

* **Mobile Development**: Primarily for Android apps using Java or Kotlin.
* **Enterprise Applications**: Java is widely used for large-scale, distributed enterprise systems.
* **Game Development**: For example, Minecraft is built in Java.
* **IoT and Cloud-Based Applications**: Java is commonly used in connected devices and cloud platforms.

**JAVA PLATFORMS**

### **Java SE (Standard Edition)**

### **Java EE (Enterprise Edition)**

### **Java ME (Micro Edition)**

### **Java SE (Standard Edition)**

**Example**: **Desktop Applications**

A **text editor** (like Notepad) that allows users to create and edit text files.

**Real-world Usage**: Many desktop applications like **Eclipse IDE** (for Java development) and **Apache NetBeans** are built using Java SE. They offer features like code editing, debugging, and project management.

### **Java EE (Enterprise Edition)**

**Example**: **Web Applications**

**Example Application**: **E-commerce platforms** (like Amazon).

**Real-world Usage**: Java EE is commonly used in enterprise environments. For instance, many **banking applications** use Java EE to manage transactions and customer data, allowing users to perform secure online banking operations.

### **Java ME (Micro Edition)**

**Example**: **Mobile Applications**

**Example Application**: **Mobile games** or applications like **Java-based mobile browsers**.

**Real-world Usage**: Early smartphones often used Java ME to run applications. For example, **mobile games** like **Angry Birds** (in its earlier versions) were developed using Java ME for compatibility with various devices.

### **Real-Time Examples:-**

**Java SE**: Desktop applications like Eclipse IDE, Apache NetBeans, and text editors.

**Java EE**: Enterprise-level applications like e-commerce websites and banking systems.

**Java ME**: Mobile applications and games designed for feature phones and early smartphones.

**Java Extension:-**

**.java File**: Contains Java source code. Written and edited by developers.

* A file that contains Java source code.
* **Who Uses It**: Developers write and edit their code in these files.
* **Naming**: The filename should match the main class inside it. For example, if your class is called sample, the file should be named sample.java.
* **How It Works**:
* You write your Java code in the .java file.
* When you compile it (using a command like javac), it turns into a .class file, which the computer can run.

### **Example:**

**public class Sample {**

**public static void main(String[] args) {**

**System.out.println("Hello, World!"); // This prints "Hello, World!" to the screen**

**}}**

### **.class File**

* A .class file is the result of compiling your .java file.
* It contains the bytecode and that can be executed by Java Virtual Machine (JVM)
* **Example**: After you compile sample.java using the command javac sample.java, it creates a file called sample.class. This file contains the bytecode.

**IDE(Integrated Development Environments IDEs)**

Are software applications that provide developers with tools to write, compile, debug, and manage Java code more easily. Here are some popular Java IDEs:

### Eclipse

### IntelliJ IDEA

### NetBeans

### JDeveloper

### Apache NetBeans

1. VisualStudio Code.

**Customization**: Most IDEs allow you to customize the interface and add plugins for additional features.

**Debugging**: IDEs typically have built-in debugging tools to help find and fix errors in our code.

**Version Control**: Many IDEs integrate with version control systems (like Git) for collaborative development.

### **Java Architecture Components**

* **Java Virtual Machine (JVM)**:
  + **Role**: Executes Java bytecode.
  + **Key Feature**: Provides platform independence, allowing the same bytecode to run on any device with a JVM.
* **Java Runtime Environment (JRE)**:
  + **Role**: Provides the environment to run Java applications.
  + **Includes**: JVM and core libraries but no development tools.
* **Java Development Kit (JDK)**:
  + **Role**: Complete toolkit for developing Java applications.
  + **Includes**: JRE, Java compiler (javac), and other development tools.
* **JVM**: Executes bytecode.
* **JRE**: Runs Java applications (includes JVM).
* **JDK**: Develops Java applications (includes JRE and tools(libraries)).

**JIT (Just-In-Time) Compilation** is a crucial feature of the Java Virtual Machine (JVM) that enhances the performance of Java applications.

### **What is JIT Compilation?**

Converts Java bytecode into native machine code at runtime.

**How It Works**:

* + When a Java program is executed, the JVM initially interprets the bytecode (which is slower).
  + The JIT compiler identifies frequently executed code (hot spots) and improving execution speed.
  + compiles that bytecode into native machine code.

**Key FeaturesJIT:**

* **Performance Boost**: By compiling code that is executed often, JIT can reduce the time taken for subsequent executions.
* **Adaptive Optimization**: The JIT compiler can optimize code based on actual runtime behavior, making it more efficient.
* **Combines Interpretation and Compilation**: JIT provides the flexibility of both interpreting and compiling, allowing for quick startup times and improved performance for long-running applications.

### 

### **Benefits of JIT:**

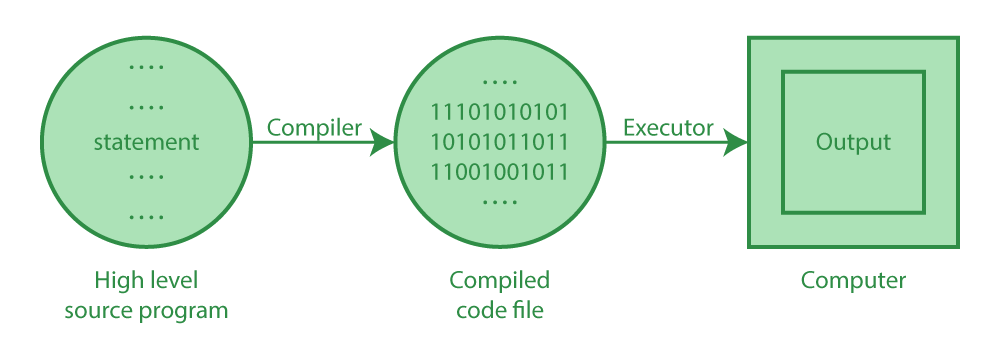
* **Faster Execution**: Programs run faster after the JIT compiler has optimized the bytecode.
* **Reduced Overhead**: Once compiled, the machine code doesn’t need to be interpreted again, which saves **processing time.**

JIT compilation helps Java applications run faster by converting bytecode into native machine code at runtime, optimizing frequently used code to enhance performance.

### **Setting Up a Java Workstation Environment:**

1. **Install the JDK**: Download and install the latest JDK
2. **Choose an IDE**: Download and install your preferred IDE (Eclipse, IntelliJ IDEA, or NetBeans).
3. **Set Up Version Control**: If using Git, install Git and create a repository on GitHub or a similar platform.
4. **Configure Environment Variables**: Set the JAVA\_HOME variable and update the PATH variable to include the JDK's bin directory.
5. **Start Coding**: Create a new project in your IDE, write your Java code, and run your applications.

**Compiler and Interpreter**

****

****

****

**Java Sample Program:-**

public class Sample {

// sample is a instance of a class

//class is a keyword, it is used for defined a class

//public is AM(access modifiers)

public static void main(String[] args) {

//main() method

//{ } block

//[] array indication

System.out.println("Hello, World!"); //standard output method in java

}

}

**public**: This keyword makes the class accessible from other parts of the program.

**class**: Defines a class in Java. A class is like a blueprint for objects, but here it's used to organize the program.

**Sample**: The name of the class. The file containing this code should be named Sample.java.

**public**: Makes the main method accessible to the Java runtime.

**static**: This means the method belongs to the class itself, not to instances of the class.

**void**: Indicates that the main method doesn’t return any value.

**main**: This is the entry point of any Java program. The JVM (Java Virtual Machine) starts running the code from this method.

**String[] args**: This allows the program to accept command-line arguments (data passed when the program is run). It's an array of String objects.

**System.out**: Refers to the standard output stream, which is usually the console.

**println()**: A method that prints the text inside the parentheses to the console, followed by a new line.

**Class is divided into 2 types:**

1. **Driver class.**

It contains main method:

Syntax

public class mainCls{

Public static void main(String[ ] args){

objCls o =new objCls();

o.add();

}

}

1. **Object class.**

It contains the logic of object

public class objCls{

Public void add(){

int a=5;

int b=10;

int c=a+b;

System.out.println(“value of c is:” +c);

} }

**Java Variables**

Variable in Java is a data container that stores the data values during Java program execution.

Every variable is assigned a data type which designates the type and quantity of value it can hold. A Variable is a Data Holder which stores the data .

Variable is a memory location name of the data.

The Java variables have mainly 2 types :

**1.Local variables**

**2. Global variables**

* **Instance variables**
* **Static variables.**

**Global Variable:**

A Global variable is a variable which is declared directly within class , outside method or constructor.

If Global variable is static , then it is called **class variable.**

If Global variable is non static , then it is called **Instance variable.**

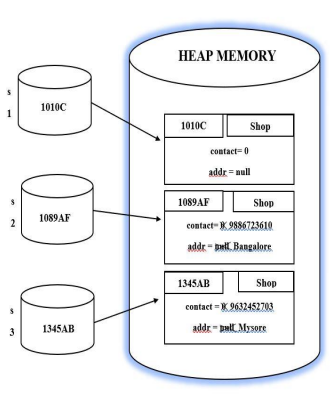
**Instance Variable:**

• **A non static Global variable is called Instance Variable.**

• Instance Variable is created in memory only when an Object gets created.

• Number of copies of each instance variable depends on number of objects.

• Instance Variable is stored inside an Object as a part of Heap memory.



Class Greeting{

long contact;

String addr;

public static void main(String[] args){

Shop s1 = new Shop();

Shop s2 = new Shop();

s2.contact=9886723160L;

s2.addr="Bangalore";

Shop s3 = new Shop();

s3.contact=7887623160L;

s3.addr="Mysore";}}

**Static Variable:**

A static variable is common to all the instances (or objects) of the class because it is a class level variable.

In other words you can say that only a single copy of static variable is created and shared among all the instances of the class.

Memory allocation for such variables only happens once when the class is loaded in the memory.

Static variable Syntax static keyword followed by data type, followed by variable name.

**Syntax:-**

**static data\_type variable\_name ;**

public class Student {

static int studentCount = 0;

public Student() {

studentCount++;

}

public static void main(String[] args) {

Student s1 = new Student();

Student s2 = new Student();

Student s3 = new Student();

Student s4 = new Student();

System.out.println("Total students created: " + Student.studentCount);

}}

**Local Variable:**

• Local variable is a variable which is created within a method or constructor or block.

• Local variable must be initialized before use . i.e.., Default initialization is not applicable for local variable.

• The scope of local Variable is Limited .

• Local variable cannot be accessed using Object reference or by using this keyword.

public class A

{

static int m=100; **//static variable**

void method()

{

int n=90; **//local variable**

}

public static void main(String args[])

{

int data=50; **//instance variable**

}

}//end of class

public class A

{

static int m = 100;  **// Static variable**

int data = 50;  **// Instance variable**

void method()

{

int n = 90;  **// Local variable**

System.out.println("Local variable n: " + n);

System.out.println("Instance variable data: " + data);

System.out.println("Static variable m: " + m);

}

public static void main(String args[])

{

A obj1 = new A(); **// Creating instance**

obj1.method();

}

}

public class A

{

static int m = 100; // Static variable

int data = 50; // Instance variable

void method()

{

int data = 50;

int n = 90; // Local variable

System.out.println("Local variable n: " + n);

System.out.println("Instance variable data: " + data);

System.out.println("Static variable m: " + m);

}

public static void main(String args[])

{

A a = new A(); // Creating instance

a.method();

}

}

**Class and Object:**

A **class** is a blueprint or template that defines the structure and behavior (attributes and methods) of objects. It describes what an object will look like and what it can do, but it doesn’t represent any actual data or specific instance. You can think of a class as a recipe, and objects are the dishes made from that recipe.

#### **Key Elements of a Class:**

* **Attributes (Fields)**: Represent the properties or data members of the class. For example, in a Car class, attributes might include color, model, and year.
* **Methods**: Define the behaviors or functions that objects of the class can perform. For example, a Car class may have methods like start(), stop(), and accelerate().
* **Constructor**: A special method used to create and initialize objects. Constructors usually set up initial values for the attributes.

An **object** is an instance of a class. When a class is instantiated (using the new keyword in Java), it creates an object in memory, with specific values for the attributes defined by the class. Each object has its own identity, attributes, and behaviors as defined by the class. If the class is a blueprint, then an object is the actual product built from that blueprint.

**Objects as Instances**: An object is an instance of a class and represents a specific entity.

Car c=new Car();

Car c1=new Car()

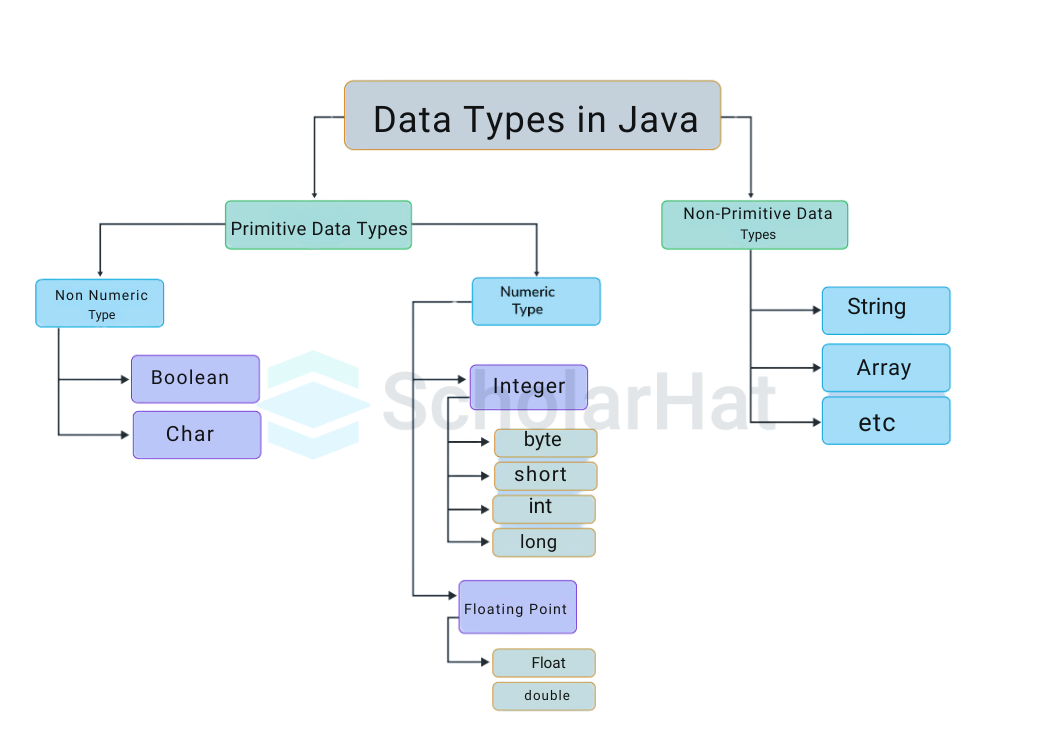
Int string char boolean int a=11;

**Data types**

* Data can be given certain names in the program, these names are called variables.
* These variables point to a memory location where the value is stored. This value can change during the program run.
* The type of data stored by a variable is represented using data types.
* Data types in Java are all about handling and storing data.
* Java data types are divided into two types such as primitive data types and non-primitive data types.
* Primitive data types in Java, like int, char, and boolean, store simple values directly.
* Non-primitive data types in Java, such as arrays, classes, and interfaces, are derived from objects.
* Understanding both primitive and non-primitive data types in Java is important for effective programming and memory management.



1. **Primitive data type:** This particular data type includes float, short, boolean, byte, char, long, int, and double.
2. **Non-primitive data type**: This particular data type includes arrays, interfaces, strings, and classes.



| **byte** | **1 byte** | **Stores whole numbers from -128 to 127** |
| --- | --- | --- |
| **short** | **2 bytes** | **Stores whole numbers from -32,768 to 32,767** |
| **int** | **4 bytes** | **Stores whole numbers from -2,147,483,648 to 2,147,483,647** |
| **long** | **8 bytes** | **Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807** |
| **float** | **4 bytes** | **Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits** |
| **double** | **8 bytes** | **Stores fractional numbers. Sufficient for storing 15 decimal digits** |
| **boolean** | **1 bit** | **Stores true or false values** |
| **char** | **2 bytes** | **Stores a single character/letter or ASCII values** |

**American Standard Code for Information Interchange.**

**Byte**

public class Main {

public static void main(String[] args) {

byte myNum = 126;

System.out.println(myNum);

}

}

**Short**

short myNum = 30000;

**Long**

long myNum = 15000000000L;

**Float**

float myNum = 8.79f;

**Double**

double myNum = 78.36d;

**Char**

char scholarHat = ‘S’

**//Checking Ascii values for characters**

public class Main {

public static void main(String[] args) {

char myVar1 = 65, myVar2 =90, myVar3 = 97;

System.out.println(myVar1);

System.out.println(myVar2);

System.out.println(myVar3);

}

}

**Non Primitives:**

**Arrays:-**

data\_type array\_name[array\_size];

example:

int[] a = {25, 50, 75, 100};

System.out.println(a[0]);

}

}

**String:-**

String greeting = "Welcome to ScholarHat";

**Class:-**

public class ScholarHat {

int x = 5;

}

**Object:-**

public class R{

int x = 50;

public static void main(String[] args) {

R r= new R();

System.out.println(r.x);

}

}

**Interface:**

interface ScholarHat {

public void contentwriters(); // interface method (does not have a body)

public void seo(); // interface method (does not have a body)

}

**Methods in Java:**

A method is a collection of statements that perform some specific task and return the result to the caller.

A method is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as functions.

**Why use methods?**

To reuse code: define the code once, and use it many times.

Java provides some pre-defined methods, such as

**System.out.println();**

but you can also create your own methods to perform certain actions:

**Syntax:-**

return\_type MethodName ( )

{

// Body of the Method / logic / implementation

return statement;

}

Mymethod(){

………

}

A method can perform some specific task without returning anything.

Methods allow us to reuse the code without retyping the code.

Methods are time savers and help us to reuse the code without retyping the code.

**Rules for Methods :**

• Methods must have either return type or void but not both.

int method( ) { int meth(int c ) {

return 40; return c;

return d;

} }

boolean m1 ( ) { int m2 ( ) {

return true; return 87;

return false; } System.out.println(“Hello”); }

• Void method cannot return any data.

• A method can have only one return type or only one return statement.

• Return type of method and the returning value must match.

• Inside a method , return statement must be the last executable statement.

• A Method gets executed only when we invoke it.

• A Method can be invoked multiple times .



**Example:-**

public class Mainclass {

void session() {

System.out.println(“welcome to java session");

}

public static void main(String[] args) {

session();

session();

}

}

**Parameters and Arguments Example program**

public class mainCls {

static void methodname(String firstName, String lastName) { // Parameters

System.out.println(firstName + " programming" + lastName );

}

public static void main(String[] args) {

methodname("c","language"); // Arguments

methodname("c++","language");

methodname("java","language");

}

}

**public** **class** paCls {

**static** **void** GREET(**int** id, String name) { // Parameters

System.***out***.println(id + " RCS" + name );

}

**public** **static** **void** main(String[] args) {

// Arguments

*GREET*(1,"M");

*GREET*(2,"POOJA");

}}

**public** **class** paCls {

String greet(**int** id,String name) {

**return** id + name;

}

**public** **static** **void** main(String[] args) {

// Arguments

paCls p=**new** paCls();

System.out.println(p.greet(6, "pooja"));

System.out.println(p.greet(2, "vinay"));

}

}

**Return Values**

The void keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int,char , etc.) instead of void , and use the return keyword inside the method:

public class returnCls {

static int myMethod(int x, int y) {

return x + y;

}

public static void main(String[] args) {

System.out.println(myMethod(5, 3));

}

}

Types of Methods in Java

**1. Methods Without Parameters and Without Return Type :** Perform actions without inputs or outputs.

These methods perform actions but do not take any inputs or return any value.

class Example {

// Method without parameters and without return type

void greeting() {

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

}

}

**2. Methods With Parameters and Without Return Type :** Take inputs to perform actions but do not return a value.

These methods take inputs but do not return any value.

class Example {

// Method with parameters and without return type

void printSum(int a, int b) {

System.out.println("Sum: " + (a + b));

}

}

**3. Methods Without Parameters and With Return Type :** Provide output without requiring inputs.

These methods do not take any inputs but return a value.

class Example {

// Method without parameters and with return type

int getNumber() {

return 42;

}

}

**4. Methods With Parameters and With Return Type:** Both take inputs and provide outputs.

These methods take inputs and return a value.

class Example {

// Method with parameters and with return type

int mul(int a, int b) {

return a \* b;

}

}

public class mythodsExample {

void greeting() {

System.out.println("Hello!"); // Method without parameters and without return type

}

void printSum(int a, int b) {

System.out.println("Sum: " + (a + b)); // Method with parameters and without return type

}

int getNumber() {

return 42; // Method without parameters and with return type

}

int mul(int a, int b) {

return a \* b; // Method with parameters and with return type

}

public static void main(String[] args) {

mythodsExample exp = new mythodsExample();

exp.greeting(); // Calling method without parameters and without return type

exp.printSum(5, 10); // Calling method with parameters and without return type

int number = exp.getNumber(); // Calling method without parameters and with return type

System.out.println("The number is: " + number);

int product = exp.mul(4, 5); // Calling method with parameters and with return type

System.out.println("Product: " + product);

}

}