INTRODUCTION TO JAVA

Java programming is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle Corporation) in the mid-1990s. It was designed to be platform-independent, meaning that Java programs can run on any device that has a Java Virtual Machine (JVM) installed, regardless of the underlying hardware and operating system.

Java programs are typically written in text files with a .java extension. These source files are then compiled by the Java compiler into bytecode, which is a platform-independent representation of the program. The bytecode can be executed on any device that has a compatible JVM.

Java is widely used for developing a variety of applications, including desktop, web, and mobile applications, as well as enterprise software and large-scale systems. It is also used extensively in server-side development, especially for building web services and backend systems.

Key features of Java programming include:

1. **Object-oriented**: Java is based on the object-oriented programming (OOP) paradigm, which emphasizes the organization of code into reusable components called objects.
2. **Platform independence**: Java programs are compiled into bytecode, which can be executed on any device with a JVM. This "write once, run anywhere" capability makes Java suitable for developing applications ranging from desktop to web and mobile.
3. **Automatic memory management**: Java features automatic garbage collection, which manages memory allocation and deallocation, relieving programmers from the burden of manual memory management.
4. **Rich standard library**: Java comes with a comprehensive standard library (Java API) that provides a wide range of functions and utilities for common tasks, such as networking, I/O operations, data structures, and more.
5. **Security**: Java includes built-in security features to protect against various vulnerabilities, such as sandboxing for running untrusted code and a robust security architecture.
6. **Multi-threading support**: Java provides built-in support for concurrent programming with multi-threading, allowing developers to create applications that can perform multiple tasks simultaneously.

Java is widely used in enterprise software development, web development (via technologies like JavaServer Pages and servlets), mobile app development

What is Java programming language used for?

Because Java is a free-to-use and a versatile language, it builds localized and distributed software. Some common uses of Java include:

1. Game Development

Many popular mobile, computer, and video games are built in Java. Even modern games that integrate advanced technology like machine learning or virtual reality are built with Java technology.

2. Cloud computing

Java is often referred to as WORA – Write Once and Run Anywhere, making it perfect for decentralized cloud-based applications. Cloud providers choose Java language to run programs on a wide range of underlying platforms.

3. Big Data

Java is used for data processing engines that can work with complex data sets and massive amounts of real-time data.

4. Artificial Intelligence

Java is a powerhouse of machine learning libraries. Its stability and speed make it perfect for artificial intelligence application development like natural language processing and deep learning.

5. Internet of Things

Java has been used to program sensors and hardware in edge devices that can connect independently to the internet.

Why is Java such a popular choice among modern-day software developers?

Java is popular because it has been designed for ease of use. Some reasons developers continue to choose Java over other programming languages include:

High quality learning resources

Java has been around for a long time, so many learning resources are available for new programmers. Detailed documentation, comprehensive books, and courses support developers through the learning curve. In addition, beginners can start writing code in Core Java before moving to Advanced Java.

Inbuilt functions and libraries

When using Java, developers don't need to write every new function from scratch. Instead, Java provides a rich ecosystem of in-built functions and libraries to develop a range of applications.

Active community support

Java has many active users and a community that can support developers when they face coding challenges. The Java platform software is also maintained and updated regularly.

High-quality development tools

Java offers various tools to support automated editing, debugging, testing, deployment, and change management. These tools make Java programming time and cost-efficient.

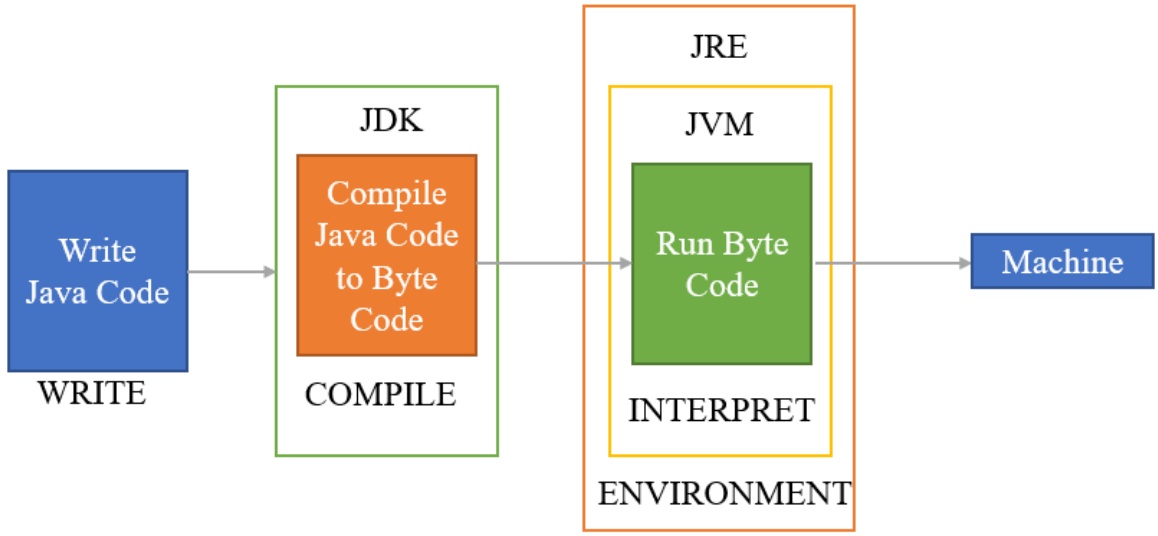
Platform Independent

Java code can run on any underlying platform like Windows, Linux, iOS, or Android without rewriting. This makes it especially powerful in today's environment, where we want to run applications on multiple devices.

Security

Users can download untrusted Java code over a network and run it in a secure environment in which it cannot do any harm. Untrusted code cannot infect the host system with a virus nor can it read or write files from the hard drive. The security levels and restrictions in Java are also highly configurable.

Architecture of JAVA:



Java provides several data types that programmers can use to define variables and manipulate data within their programs. These data types can be broadly categorized into two groups: primitive data types and reference data types.

Primitive Data Types:

1. **byte**: 8-bit signed integer data type. Range: -128 to 127.
2. **short**: 16-bit signed integer data type. Range: -32,768 to 32,767.
3. **int**: 32-bit signed integer data type. Range: -2^31 to 2^31 - 1.
4. **long**: 64-bit signed integer data type. Range: -2^63 to 2^63 - 1.
5. **float**: 32-bit floating-point data type for single-precision floating-point numbers.
6. **double**: 64-bit floating-point data type for double-precision floating-point numbers.
7. **char**: 16-bit Unicode character data type. Range: '\u0000' (0) to '\uffff' (65,535).
8. **boolean**: Represents true or false values.

Reference Data Types:

1. **Class types**: These are user-defined types created using classes. Examples include objects of classes you define in your Java code.
2. **Array types**: Arrays are objects that store multiple values of the same type in a contiguous memory location.

3. **Interface types**: Interfaces define a contract for classes to implement. You can create variables of an interface type that reference objects of classes that implement the interface.
4. **Enumeration types (enums)**: Enums are a special data type that enables a variable to be a set of predefined constants.

CONDITIONAL STATEMENT:

In Java, conditional statements are used to control the flow of the program based on certain conditions. The most common conditional statements are:

1. **if statement**: It executes a block of code if the specified condition is true.

   javaCopy code
   if

2. **if-else statement**: It executes one block of code if the specified condition is true and another block if the condition is false.

   javaCopy code
   if else

3. **if-else-if ladder**: It checks multiple conditions and executes the block of code corresponding to the first true condition.

   javaCopy code
   if else if
   else if

   else

JAVA CONSTRUCTOR

In Java, a constructor is a special type of method that is automatically called when an instance (object) of a class is created. It is used to initialize the newly created object. Constructors have the same name as the class and do not have a return type, not even **void**.

Here's a basic example of a constructor in Java:

```
public class MyClass {
    int myNumber; // instance variable
```

```
    // Constructor
    public MyClass() {
        myNumber = 10; // initializing instance variable
    }

    public static void main(String[] args) {
        // Creating an object of MyClass
        MyClass obj = new MyClass();

        // Accessing the instance variable initialized in
the constructor
        System.out.println("My number is: " + obj.myNumber);
// Output: My number is: 10
    }
}
```

In this example:

- We have a class named **MyClass**.
- Inside the class, we have an instance variable **myNumber**.
- We have a constructor **MyClass()** which initializes **myNumber** to 10.
- In the **main** method, we create an object **obj** of type **MyClass** using the **new** keyword, which automatically invokes the constructor.
- We then access the instance variable **myNumber** using the object **obj** and print its value.