BASICS OF JAVA PROGRAMMING

What is Java?

Definition:

Java is a high-level, Object-Oriented Programming Language designed to have as few implementation dependencies as possible.

- Java is a programming language, where a language is a medium that is needed for communication
- Programming language is needed to communicate with computers/machines.
- Where a program refers to a set of instructions it is necessary to write programs to develop applications.
- Where application refers to collections of multiple different programs.
- The goal of any application is to perform some specific task

History of Java

- Java was invented by James Gosling and his team at Sun Microsystems (later acquired by Oracle) in 1995.
- The original project started in 1991, and Java was initially called Oak (named after an oak tree outside Gosling's office).
- The language was later renamed Java, inspired by the popular coffee consumed by the team.

Features

- Object-Oriented: Object-Oriented Programming Paradigm centered around the concept of “Object”, Which are instances of classes.

- Platform Independence: Java is platform independent that is java doesn't depend upon one particular platform and it follows WORA Architecture(Write Once Run Anywhere).

- Robust and Secure: Java has strong memory management, exception handling, and features like garbage collection, which helps in managing memory leaks and improving program reliability.

- Concurrency: Java has built-in support for multithreading, enabling developers to build applications that can perform multiple operations simultaneously.

- Wide Use: Java is used for developing various types of applications, including web applications, desktop applications, mobile applications (especially Android apps), and enterprise-level solutions.

Use cases:-

* Web Development
* Moblie Application(Android)
* Enterprise Applications, etc

Writing a Java Program?

- Java is a programming language that lets us communicate with computers. Like any language, Java has grammar and rules that we need to follow to write programs effectively.

Declaring a Class in Java: • In Java, declaring a class is the first step in writing a Java program.

- A class is a blueprint from which objects are created, and it encapsulates data (fields) and behavior (methods). Here's how you can declare a class in Java.
- A class is declared using the class keyword followed by the class name.

```
class ClassName {
    // Fields (attributes or properties)
    // Methods (functions or behavior) }
```

Java Main( ) Method:

- In Java, the main() method is the entry point of any Java application.
- When you run a Java program, the execution starts from the main() method.
- It is essential because it serves as the starting point where the JVM (Java Virtual Machine) begins executing the program.

Syntax of Main() Method:

```
    public static void main(String[] args) {
// code to be executed
    }
```

Printing a Message:

- In Java, you can print a message to the console using the System.out.println() method. This method is commonly used to

display output to the user. It prints the specified message and then moves the cursor to a new line.

Syntax of System.out.println() :

System. out. println("Message to be printed");

Alternative Syntax of Main()

- In Java, the main() method must follow certain rules to serve as the entry point of the program.
- While the syntax must adhere to the general structure, certain variations in the syntax are allowed, such as the order of keywords or how the parameters are written. Below is an explanation of five different valid forms of the main() method.

Overview of Main() Method Syntaxes in Java

Syntax 1:

```
public static void main(String[] args) {
    // Code to execute
}
```

- This is the most common and standard form of the main() method in Java. It follows the conventional order of the public, static, and void keywords. The parameter String[] args is used to accept command-line arguments in the form of an array of String.

Syntax 2:

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

```
        // Code to execute
}
```

- This variation is still valid. In Java, the order of the public and static modifiers doesn't matter. You can interchange them without affecting the behavior of the program. Both forms are equivalent in terms of functionality.

Syntax 3:

```
public static void main(String args[]) {
    // Code to execute
}
```

- In Java, when declaring an array, the brackets [] can be placed either after the type (String[]) or after the variable name (args[]).

So, this form is also valid and works the same as the standard form.

Syntax 4:

```
public static void main(String... args) {
    // Code to execute
}
```

- This is a valid variable (variable arguments) form of the main() method. The ellipsis (...) allows the method to accept zero or more arguments.
- Functionally, this form is equivalent to String[] args because String... represents an array of String. It can be useful when you want the flexibility of passing varying numbers of arguments.

Syntax 5:

```
public static void main(String[] Car) {
    // Code to execute
}
```

- This is a valid variation where the parameter name has been changed from args to Car.
- The name of the parameter is irrelevant to the Java Virtual Machine (JVM).
- It can be any valid identifier, as long as the type is String[]. The JVM looks only at the method signature (return type, method name, and parameter types), not the parameter names.

What is programming?

Programming is the process of creating instructions that a computer can execute. These instructions, called "code," are written in a programming language, which is a formal language designed to communicate with computers.

The flow of a Java program follows a structured process from writing the code to its execution. Here's a breakdown of the typical flow:

Writing Code in Java

Source Code: The programmer writes the source code in `.java` files using classes, methods, and variables. Java follows the Object-Oriented Programming (OOP) paradigm, meaning the code revolves around objects and their interactions.

Class Declaration: Every Java program has at least one class declaration,and within this class lies the `main()` method, which serves as the entry point for execution.

Compilation

Javac (Java Compiler): The Java source code (`.java` files) is passed to the Java compiler, `javac`, which converts the human-readable code into bytecode. Bytecode is an intermediate representation stored in `.class` files, which can be executed by the Java Virtual Machine (JVM).

Syntax Check: During this step, the compiler ensures the code follows the correct syntax rules. Any errors will prevent successful compilation.

Execution via JVM

Java Virtual Machine (JVM): Once the code is compiled into bytecode, the JVM comes into play. It reads the `.class` files and interprets the bytecode line by line or uses the Just-In-Time (JIT) compiler to convert the bytecode into machine code for faster execution.

Platform Independence: The bytecode allows Java to be platform-independent because JVMs are available for different operating systems (Windows, Linux, macOS), making Java applications portable across platforms.

Execution Flow in the Program

Main Method Execution: The JVM looks for the `main()` method in the entry class. This method serves as the starting point for the program's execution.

Control Flow: The flow of control then proceeds based on the logic defined in the code—sequentially, through method calls, conditionals (if-else, switch), loops (for, while), and object interactions.

Garbage Collection: Java's memory management system uses automatic garbage collection to reclaim memory used by objects that are no longer referenced.

Termination

Program End: The program ends when the `main()` method completes its execution, or when the `System. exit()` method is called. Exception handling can also cause termination in cases of unhandled exceptions.

The flow of execution in Java is linear and predictable but allows for complex behaviors with features like multithreading and exception handling.