# **History of Java Programming Language**

Java is an Object-Oriented programming language developed by **James Gosling** in the early 1990s. The team initiated this project to develop a language for digital devices such as set-top boxes, television, etc. Originally C++ was considered to be used in the project but the idea was rejected for several reasons(For instance C++ required more memory). Gosling endeavoured to alter and expand C++ however before long surrendered that for making another stage called **Green**. James Gosling and his team called their project “**Greentalk**” and its file extension was **.gt** and later became known as “**OAK**”.

**Why “Oak”?**

The name **Oak** was used by **Gosling** after an **oak tree** that remained outside his office. Also, Oak is an image of solidarity and picked as a national tree of numerous nations like the U.S.A., France, Germany, Romania, etc. But they had to later rename it as “**JAVA**” as it was already a trademark by **Oak Technologies**. **“JAVA”** Gosling and his team did a brainstorming session and after the session, they came up with several names such as **JAVA, DNA, SILK, RUBY, etc.** **The Java** name was decided after much discussion since it was so unique. The name Java originates from a sort of **espresso bean**, Java. Gosling came up with this name while having a coffee near his office. Java was created on the principles like **Robust, Portable, Platform Independent, High Performance, Multi Thread, etc.** and was called one of the **Ten Best Products of 1995** by the **TIME MAGAZINE**. Currently, Java is used in **internet programming, mobile devices, games, e-business solutions, etc.**

# **Object Oriented Programming (OOPs):**

# As the name suggests, Object-Oriented Programming or OOPs refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks assigned by you. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function

**Basic structure of Java Program:**

* **Documentation Section(Optional - Good to have)**
* **Package Declaration**
* **Import Statements(Optional)**
* **Interface Section(Optional)**
* **Class Definition**
* **Class Variables and Constants**
* **Main Method Class**
* **Methods and Behaviors**

// Documentation section

/

\* This is a Java program.

\*/

***// Package declaration***

package com.example;

***// Import statements***

import java.util.Scanner;

***// Class definition***

public class HelloWorld {

***// Main method***

public static void main(String[] args) {

**// Create a Scanner object**

Scanner scanner = new Scanner(System.in);

**// Prompt the user for input**

System.out.println("What is your name?");

**// Get the user's input**

String name = scanner.nextLine();

**// Print a greeting to the user**

System.out.println("Hello, " + name + "!");

}

}

**Compilation and execution of a program in java**

Java is a platform-independent programming language. This means that Java code can be compiled and run on any platform that has a Java Virtual Machine (JVM) installed.

The compilation process in Java converts Java source code into bytecode. Bytecode is a platform-independent intermediate representation of the Java program. The JVM is responsible for executing the bytecode.

The execution process in Java involves the following steps:

* The JVM loads the bytecode into memory.
* The JVM verifies the bytecode to make sure that it is valid.
* The JVM executes the bytecode.

The following is an example of a simple Java program:

**public class HelloWorld {**

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

**System.out.println("Hello, World!");**

**}**

**}**

To compile this program, you would use the following command:

**javac HelloWorld.java**

This would generate a bytecode file called HelloWorld.class. To execute the program, you would use the following command:

**java HelloWorld**

This would print the message "Hello, World!" to the console.

Here are some of the key points to remember about compilation and execution in Java:

* Java source code is compiled into bytecode.
* Bytecode is a platform-independent intermediate representation of the Java program.
* The JVM is responsible for executing the bytecode.
* The compilation process verifies the bytecode to make sure that it is valid.
* The execution process loads the bytecode into memory and executes it.

| **Java JDK(Java Development Kit)** | JDK = Development Tools + JRE (Java Runtime Environment)  The JDK (Java Development Kit) is a software development kit that develops applications in Java. Along with JRE, the JDK also consists of various development tools (Java Debugger, JavaDoc, compilers, etc.) |
| --- | --- |
| **JRE Java Runtime Environment** | JRE = Libraries for running the application + JVM (Java Virtual Machine)  The Java Runtime Environment (JRE) is an implementation of JVM. It is a type of software package that provides class libraries of Java, JVM, and various other components for running the applications written in Java programming. |
| **JVM Java Virtual Machine** | JVM = Only the runtime environment that helps in executing the Java bytecode.  The Java Virtual Machine (JVM) is a platform-independent abstract machine and it also comes with a Just-in-Time (JIT) compiler for converting the Java source code into a low-level machine language. |

The Java Virtual Machine (JVM) is the runtime environment that executes Java bytecode. It is responsible for loading classes, managing memory, and executing bytecode instructions. The JVM is also responsible for providing a platform-independent environment for Java applications to run on.

The JVM architecture is layered, with each layer providing a specific set of functionality. The following diagram shows the different layers of the JVM architecture:

The **Classloader** is responsible for loading classes into the JVM. It searches for class files in the classpath, which is a set of directories and JAR files. The Classloader loads the class files into the method area, which is a shared memory area where class data is stored.

The **Runtime Data Area** is a memory area that is used by the JVM to store data during runtime. It is divided into several different areas, including the heap, stack, and method area.

The **Heap** is a memory area where objects are allocated. Objects are created by the JVM when a new instance of a class is created. The heap is garbage collected, which means that the JVM automatically reclaims memory that is no longer being used by objects.

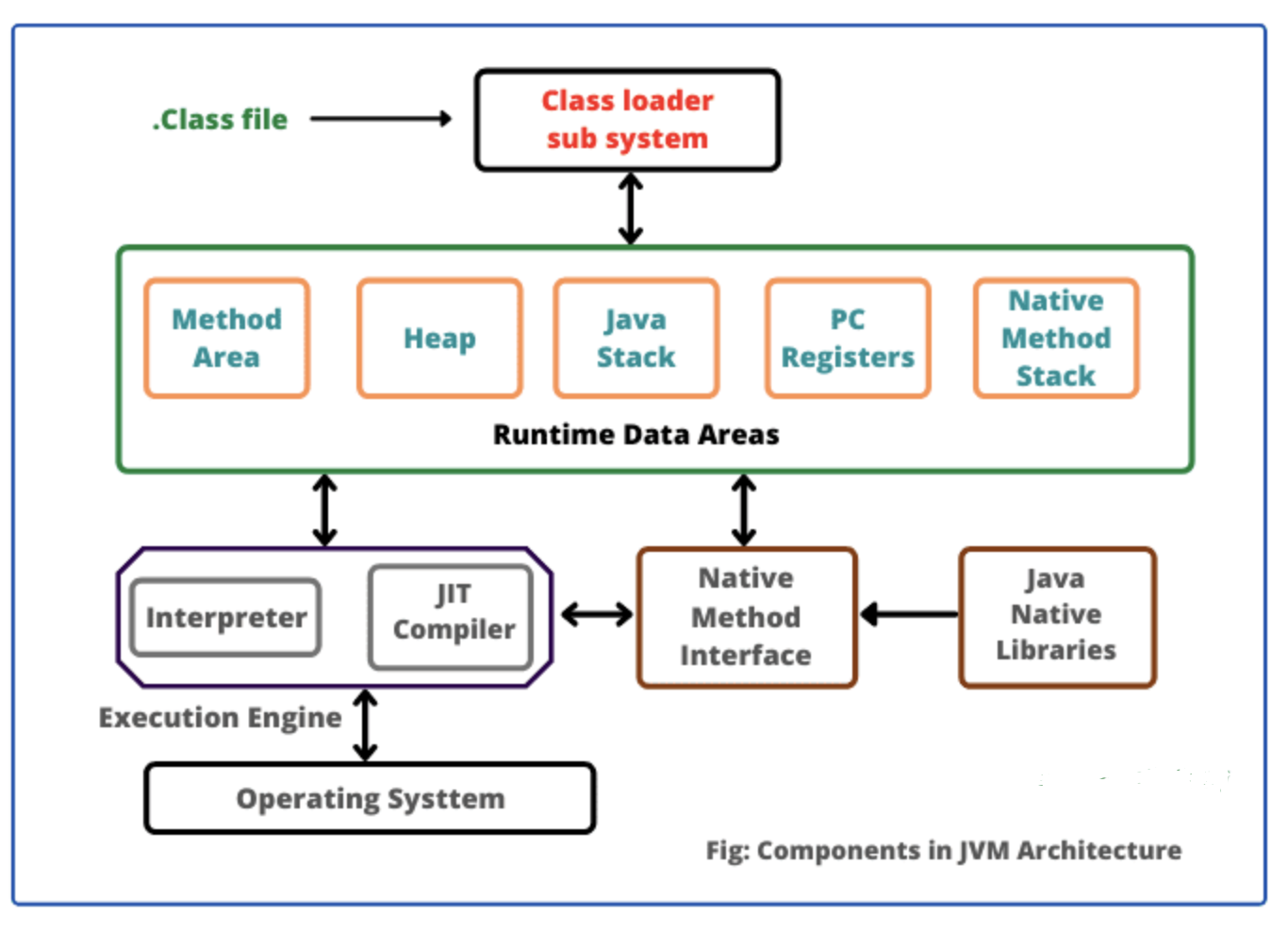
The **Stack** is a memory area where method calls are stored. When a method is called, the JVM creates a new stack frame for the method. The stack frame stores the local variables and partial results of the method.

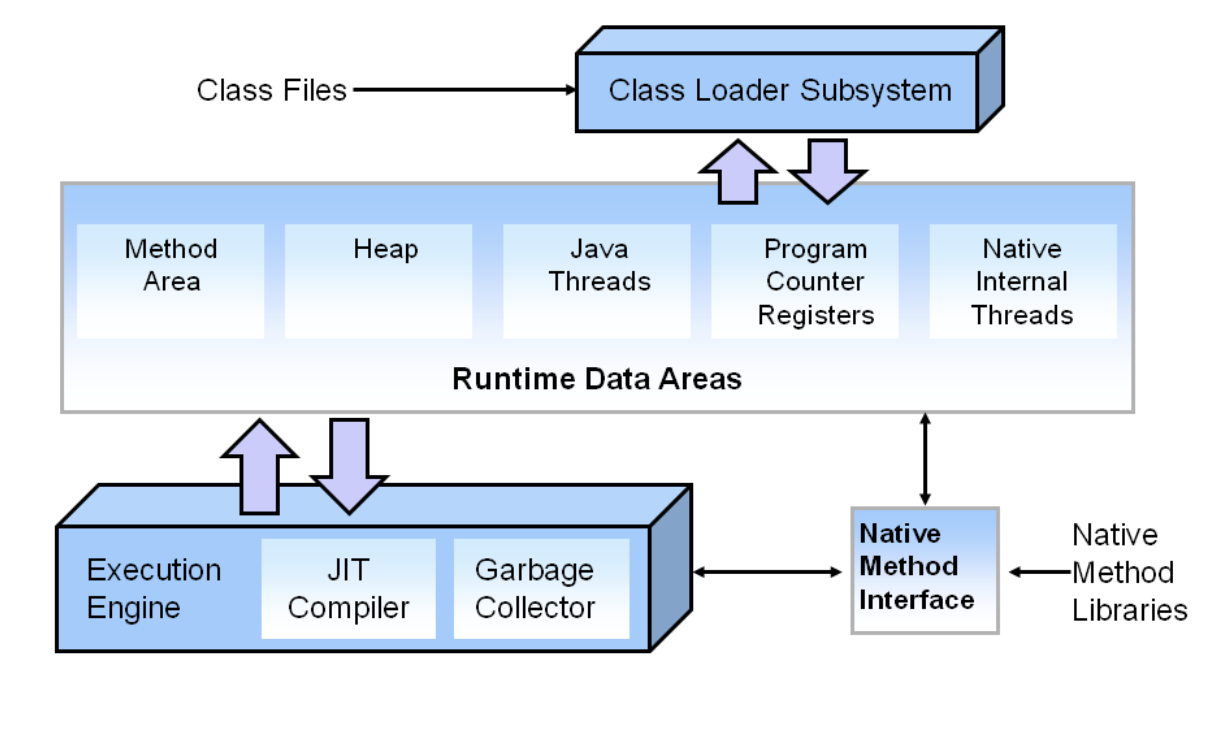
The **Method Area** is a memory area where class data is stored. This includes the runtime constant pool, field and method data, and the code for methods and constructors.

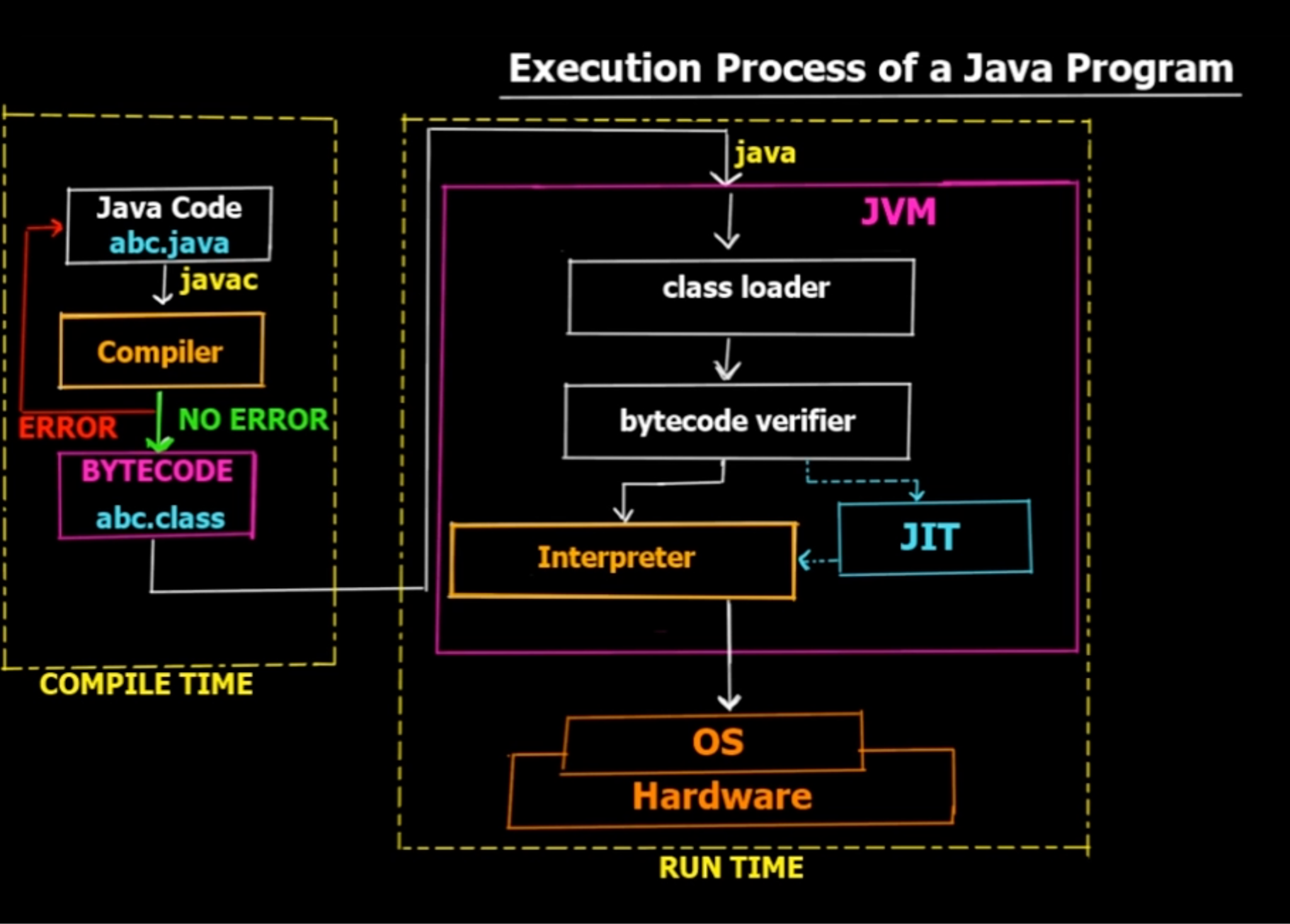
The **Execution Engine** is responsible for executing bytecode instructions. It interprets the bytecode instructions and performs the corresponding operations.

The **Native Method Interface** (JNI) is a layer that allows Java code to call native methods. Native methods are methods that are written in a language other than Java, such as C or C++.

The JVM architecture is designed to be platform-independent. This means that Java applications can run on any platform that has a JVM installed. The JVM is also designed to be efficient and scalable. It can run on a wide range of hardware platforms, from small embedded devices to large mainframe computers.







**Key benefits of the JVM architecture:**

* **Platform independence:**
* Java applications can run on any platform that has a JVM installed. This makes it easy to port Java applications to different platforms.
* **Efficiency:**
* The JVM is designed to be efficient and scalable. It can run on a wide range of hardware platforms, from small embedded devices to large mainframe computers.
* **Security:**
* The JVM provides a secure environment for Java applications to run in. It includes features such as bytecode verification and sandboxing.
* **Robustness:**
* The JVM is designed to be robust and reliable. It includes features such as garbage collection and exception handling.

Different types of programming languages with examples:

1. **Low-Level Languages:**

- **Machine Language:** Binary code directly understandable by the computer's hardware.

- Example: 01010110 11001010 00110111 (binary instructions)

- **Assembly Language:** Mnemonics representing machine instructions.

- Example: MOV AX, 5 (assembly instruction to move value 5 into register AX)

2. **High-Level Languages:**

- **Procedural Languages:**

- C: A general-purpose procedural language.

- Example:

#include <stdio.h>

int main() {

printf("Hello, World!\n");

return 0;

}

**- Object-Oriented Languages:**

- Java: A widely-used object-oriented language.

- Example:

public class HelloWorld {

public static void main(String[] args) {

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

}

}

**- Functional Languages:**

- Haskell: A purely functional programming language.

- Example:

main = putStrLn "Hello, World!”

**- Scripting Languages:**

- Python: A powerful scripting language known for its simplicity and readability.

- Example:

print("Hello, World!")

3. **Markup Languages:**

- HTML (Hypertext Markup Language): Used for creating web pages.

- Example:

```html

<!DOCTYPE html>

<html>

<head>

<title>Hello, World!</title>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

4. **Query Languages:**

- SQL (Structured Query Language): Used for managing and manipulating relational databases.

- Example:

```sql

SELECT \* FROM Customers WHERE Country='Germany';

5. **Concurrency Control Languages:**

- CUDA (Compute Unified Device Architecture): Used for parallel computing on NVIDIA GPUs.

- Example: CUDA C/C++ code for parallel processing.

6. **Statistical Languages:**

- R: Widely used for statistical analysis and data visualization.

- Example:

# Create a vector

x <- c(1, 2, 3, 4, 5)

# Calculate the mean

mean(x)

```

7. Logical Languages:

- Prolog: Used for symbolic and logic programming.

- Example:

```prolog

parent(george, charlotte).

parent(william, george).

8. **Query Languages:**

- GraphQL: A query language for APIs.

- Example:

query {

user(id: 123) {

name

email

}

}