PART – A

1. **What is Java? Explain its significance in modern software development.**

Java is a high-level, object-oriented programming language that was developed by Sun Microsystems in the mid-1990s
It was developed by James Gosling.
Java is designed to be platform-independent.
Code written in Java can run on any device that has a Java Virtual Machine (JVM) installed.
Java run on principle of WORA (Write Once and Run Anywhere).
Java is Virtual Machine based Language.
Java is robust, Highly Secure Language, easy to learn Language.

Significance in Modern Software Development:

Enterprise Applications: Java is widely used in building large-scale enterprise applications, especially in industries like finance, healthcare, and e-commerce.

Mobile Development: Java is the primary language for Android app development. The Android operating system uses a modified version of the JVM, allowing developers to build and deploy mobile applications efficiently.

Web Development: Java is used to create dynamic and interactive web applications through frameworks like Spring, Hibernate, and Struts.

Big Data: Java is instrumental in the big data ecosystem.

Cloud Computing: Java plays a crucial role in cloud-based applications and services. Many cloud platforms, such as Amazon Web Services (AWS) and Google Cloud Platform (GCP).

2. List and explain the key features of Java.

1. Object-Oriented

Java encourages the use of objects and classes, which makes the code modular, reusable, and easier to manage.

2. Platform-Independent

Java code can run on any device with a Java Virtual Machine (JVM). This means you can write code once and run it anywhere.

3. Robust

Java has strong memory management, automatic garbage collection, and exception handling, which makes it reliable and less prone to crashes.

4. Secure

Java provides built-in security features like bytecode verification and sandboxing to protect against malicious code.

5. Multithreaded

Java supports concurrent execution of multiple threads, which allows for better performance in applications that require multitasking.

6. Dynamic

Java is capable of adapting to an evolving environment. It supports dynamic linking of new class libraries, methods, and objects.

3. **What is the difference between compiled and interpreted languages? Where does Java fit in?**

Compiled Languages

In compiled languages, the source code is translated directly into machine code by a compiler.

This machine code is then executed by the computer's CPU.

Examples: C, C++, and Rust.

Interpreted Languages

In interpreted languages, the source code is executed line-by-line by an interpreter.

The code is translated into machine code at runtime.

Examples: Python, Ruby, and JavaScript.

4.Explain the concept of platform independence in Java

Platform independence means that Java code can run on any operating system.

This is achieved through the use of the Java Virtual Machine (JVM).

When we write Java code, it is first compiled by the Java compiler into an intermediate form called **bytecode**.

This bytecode is platform-independent and does not depend on any specific hardware or operating system.

The JVM is a piece of software that acts as an interpreter between the bytecode and the underlying hardware

Each operating system has its own implementation of the JVM.

5. What are the various applications of Java in the real world?

1. Enterprise Applications

Java is extensively used in large-scale enterprise applications due to its robustness, scalability, and maintainability.

2. Mobile Applications

Java is the primary language for developing Android apps. The Android operating system uses a version of the Java language and API, making it a key player in the mobile app development industry.

3. Cloud Computing

Java plays a significant role in cloud-based applications and services. Cloud platforms such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure offer strong support for Java applications.

10. Internet of Things (IoT)

Java is also used in IoT applications, where devices communicate and interact with each other over the internet. Java's platform independence and robustness make it a good fit for IoT systems.

PART – 2

1. **Who developed Java and when was it introduced?**

Java was developed by James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems.
The project began in 1991, initially named "Oak" after an oak tree outside Gosling's office.
It was later renamed "Java" in 1994.
Java was officially introduced to the public on May 23, 1995.

2. **What was Java initially called? Why was its name changed?**

Java was initially called "Oak." The name was inspired by an oak tree that stood outside James Gosling's office. However, the name "Oak" had to be changed because it was already a registered trademark for another technology.

The developers chose "Java" as the new name, inspired by a type of coffee from Indonesia. They wanted a name that conveyed a sense of energy, dynamism, and vitality, and "Java" fit the bill perfectly.

3. **Describe the evolution of Java versions from its inception to the present.**

Java 8 (Java SE 8) (2014):

Major update with lambda expressions, the Stream API, the new date and time API, and Nashorn JavaScript engine.

Java 9 (Java SE 9) (2017):

Introduced the module system (Project Jigsaw), JShell (the interactive Java REPL), and enhanced performance.

Java 11 (Java SE 11) (2018):

LTS (Long-Term Support) release with new features like the HttpClient API, and multiple garbage collector enhancements.

Java 12-16 (2019-2021):

Introduced various enhancements, including switch expressions, text blocks, records, and pattern matching for instanceof.

Java 17 (Java SE 17) (2021):

Another LTS release with features like sealed classes, pattern matching for switch, and enhanced pseudo-random number generators.

Java 21 (Java SE 21) (Expected in 2024):

Expected to be another LTS release with further enhancements and new features to keep up with modern software development needs.

4. **What are some of the major improvements introduced in recent Java versions?**

Java 21 –
Further Enhancements to Pattern Matching: Expanding the capabilities of pattern matching in more contexts.

Memory API Improvements: Continued enhancements to the Foreign Function & Memory API.

New Language Features: Introducing modern constructs to keep Java competitive with newer programming languages.

5. **How does Java compare with other programming languages like C++ and Python in terms of evolution and usability?**

Java:

Birth: Developed by James Gosling at Sun Microsystems and released in 1995.

Major Updates: Regular updates, with significant versions including Java 5 (generics, enhanced for loop), Java 8 (lambda expressions, Stream API), and the latest versions (virtual threads, pattern matching).

Philosophy: "Write Once, Run Anywhere" ensures platform independence and portability

C++:

- **Birth:** Created by Bjarne Stroustrup in 1985 as an extension of the C language.
- **Major Updates:** Key versions include C++98, C++11 (auto keyword, lambda expressions), C++14, C++17 (optional, variant, and any types), and C++20 (concepts, ranges).
- **Philosophy:** Focuses on performance and system-level programming, offering low-level manipulation of data and resources.

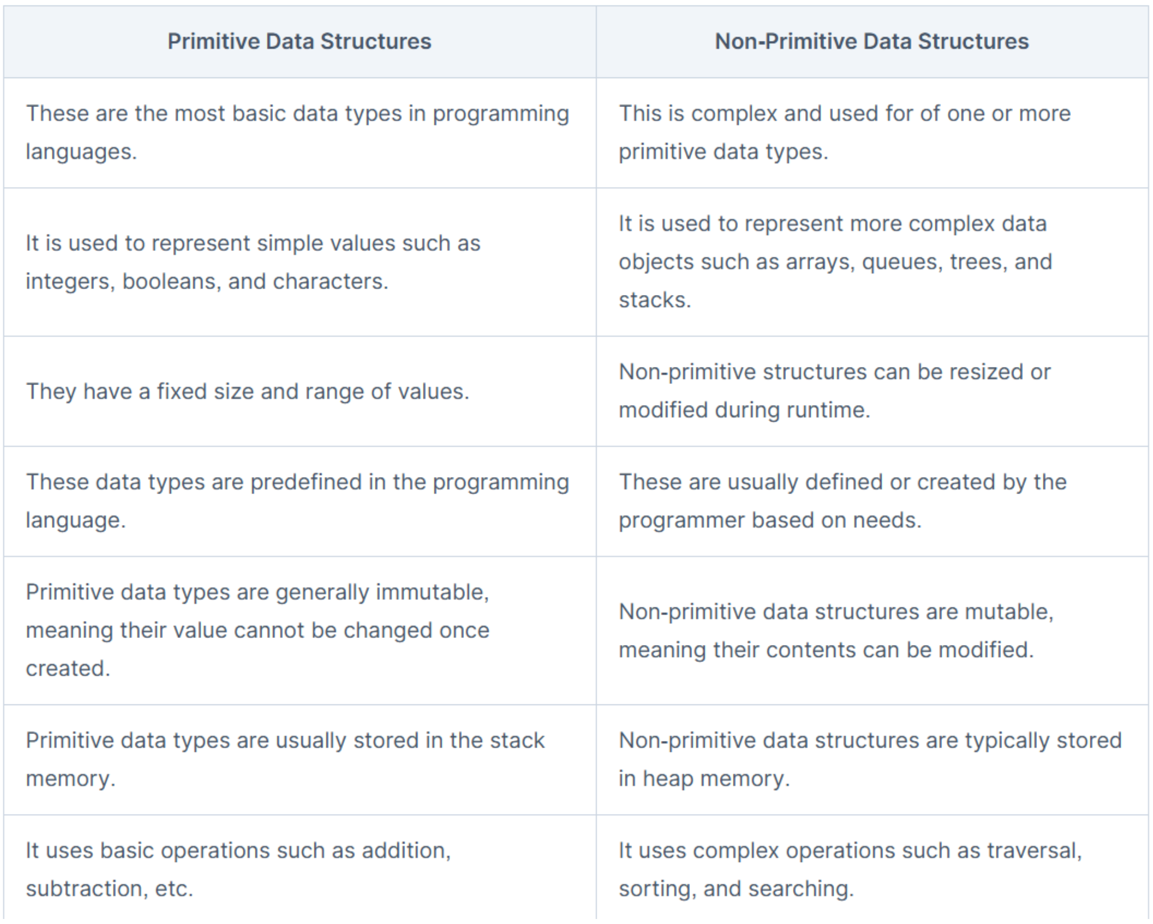
Python:

- **Birth:** Created by Guido van Rossum and released in 1991.
- **Major Updates:** Significant versions include Python 2 (2000) and Python 3 (2008) with improved syntax and features.
- **Philosophy:** Emphasizes readability and simplicity, making it easy to learn and use.

1. Explain the importance of data types in Java.

It define the nature of data that can be stored in variables, allowing the compiler to perform type-checking and ensure that only compatible operations are performed, preventing errors and enhancing code reliability , essentially, by specifying the type of data, programmers can control how the data is manipulated and allocated memory efficiently, making their code more robust and understandable.

2. Differentiate between primitive and non-primitive data types.



Primitive Data Structures	Non-Primitive Data Structures
These are the most basic data types in programming languages.	This is complex and used for of one or more primitive data types.
It is used to represent simple values such as integers, booleans, and characters.	It is used to represent more complex data objects such as arrays, queues, trees, and stacks.
They have a fixed size and range of values.	Non-primitive structures can be resized or modified during runtime.
These data types are predefined in the programming language.	These are usually defined or created by the programmer based on needs.
Primitive data types are generally immutable, meaning their value cannot be changed once created.	Non-primitive data structures are mutable, meaning their contents can be modified.
Primitive data types are usually stored in the stack memory.	Non-primitive data structures are typically stored in heap memory.
It uses basic operations such as addition, subtraction, etc.	It uses complex operations such as traversal, sorting, and searching.

3. List and briefly describe the eight primitive data types in Java.

- Boolean – its size is not fixed. Its has only two possible values that is true and false.
- Character – size is of 2 bytes. It uses only letters a-z or A-Z. it comes under integral type. Unicode character
- Byte – size is 1 byte, range is -128 to 127. it comes under integral type.
- Short – it has size of 2 bytes that is 16 bits with a range of -32768 -to 32767. it comes under integral type.

- Int – its has a size of 4 bytes which means 32 bits. It also comes under integral type.
- Long – its has a size of 8 bytes which means 64 bits. It is commonly used for scientific calculation. . it comes under integral type.
- Float – its has a size of 4 bytes that is 32 bits. it is used for decimal values with single precision.it comes under floating type
- Double – its has a size of 8 bytes that is 64 bits. offering greater precision than float.

3. Provide examples of how to declare and initialize different data types.

int a = 90;

boolean a= true;

byte a= 127;

char a= 'a';

short a= 1234;

long a= 12300*60*40;

float a= 4.05f;

double a= 4.05678d;

4. What is type casting in Java? Explain with an example.

When a datatype converted into another data type is called type casting. It is of 2 type

- Implicit – happens automatically when a smaller data type is converted into larger datatype.

  byte a = 120;

  int b = a;

  System.out.println(b);

- Explicit – wider type into narrow type. Requires manual type casting because there is risk of data loss.

A larger data type is converted into a smaller data type

```
double d = 999.99;

int n = (int)d;

System.out.println(n);
```

6. Discuss the concept of wrapper classes and their usage in Java.

7. What is the difference between static and dynamic typing? Where does Java stand?

PART – 4

1. What is JDK? How does it differ from JRE and JVM?

The JDK is a software development kit used to develop Java applications. It includes everything needed to compile, debug, and run Java applications.

JRE ((Java Runtime Environment)

The JRE is an installation package that provides an environment to **only run (not develop)** the Java program (or application) onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

Java Virtual Machine (JVM):

- **Definition:** The JVM is a part of the JRE. It is an abstract machine that enables your computer to run a Java program.

2. **Explain the main components of JDK.**

The Java Development Kit (JDK) is a crucial tool for Java developers, and it consists of several components that work together to develop and run Java programs.

1. Java Compiler (javac)
The Java compiler translates Java source code into bytecode, which is the language understood by the Java Virtual Machine (JVM). This bytecode is platform-independent, allowing Java applications to run on any device with a JVM.

2. Java Runtime Environment (JRE)
The JRE provides the necessary libraries and JVM to run Java applications. It includes the JVM, core libraries, and other components to execute Java programs.

3. Java Virtual Machine (JVM)
The JVM is the engine that runs Java bytecode. It provides a runtime environment with memory management, garbage collection, and other services to ensure Java programs run efficiently.

4. Java API Libraries
The JDK comes with a vast set of Java Application Programming Interface (API) libraries, which provide pre-written code for common programming tasks. These libraries cover areas like data structures, networking, file I/O, and more.

5. Java Debugger (jdb)
The Java Debugger is a tool for finding and fixing bugs in Java programs. It allows developers to inspect and control the execution of their programs, set breakpoints, and examine variables.

6. Java Documentation (javadoc)
The JDK includes a tool called javadoc that generates HTML documentation from Java source code comments. This documentation is essential for understanding and maintaining code, especially in large projects.

3.What is the significance of the PATH and CLASSPATH environment variables in Java?

1. PATH Environment Variable
The PATH variable is used by the operating system to locate executable files. When you run a command in the terminal or command prompt, the system searches through the directories listed in the PATH variable to find the executable.

In Java, the PATH **variable is important because:**

- It tells the system where to find the Java Development Kit (JDK) executables, such as javac (the Java compiler) and java (the Java interpreter).

- Without the correct PATH configuration, you would have to provide the full path to these executables every time you compile or run a Java program.

3. **CLASSPATH Environment Variable**

The CLASSPATH variable is used by the Java Virtual Machine (JVM) and Java tools to locate Java classes and libraries. It specifies the directories and JAR files that the JVM should search to find class definitions.
The CLASSPATH **variable is significant because:**

- It allows the JVM to locate the classes you are trying to use in your Java programs, whether they are part of the standard Java API, third-party libraries, or your own custom classes.
-
- If the CLASSPATH is not set correctly, the JVM will not be able to find and load the required classes, leading to ClassNotFoundException or NoClassDefFoundError errors.

4. **What are the differences between OpenJDK and Oracle JDK?**

OpenJDK
- **License**: OpenJDK is open-source and distributed under the GNU General Public License (GPL) with a linking exception.
- **Development**: It's developed and maintained by the Java community, including contributors from Oracle and other organizations.

- **Cost**: Free to use for both personal and commercial purposes.
- **Support**: Community-based support; no official commercial support from Oracle.
- **Features**: Generally, it has the same features as the Oracle JDK, as it is the reference implementation of the Java SE specification.

Oracle JDK

- **License**: Oracle JDK is distributed under Oracle's own commercial license.
- **Development**: Developed and maintained by Oracle.
- **Cost**: Free for personal and development use; requires a subscription for commercial use.
- **Support**: Offers commercial support and long-term support (LTS) options from Oracle.
- **Features**: May include additional commercial features, performance enhancements, and tools not present in OpenJDK.

Key Differences:

- **License and Cost**: OpenJDK is free and open-source, while Oracle JDK requires a commercial license for production use.
- **Support**: Oracle JDK provides official support and LTS options, which are not available for OpenJDK.
- **Additional Features**: Oracle JDK might have extra commercial features and tools not included in OpenJDK.

8. **What is Just-In-Time (JIT) compilation, and how does it improve Java performance?**

JIT compilation translates Java bytecode into native machine code just before it is executed. This process allows the JVM to optimize the code for the specific hardware and operating system it is running on.

How JIT Improves Performance:

1. **Hotspot Optimization**: JIT identifies frequently executed code sections (hotspots) and compiles them into native code for faster execution.
2. **Adaptive Optimization**: It can make runtime optimizations based on the actual usage patterns, leading to more efficient code.
3. **Reduced Interpretation Overhead**: By compiling bytecode to native code, JIT reduces the need for interpretation, speeding up execution.
4. **Inline Caching**: It optimizes method calls by inlining frequently called methods, reducing method call overhead.

5. **Dynamic Deoptimization**: If an optimization is invalidated, JIT can revert the code to a less optimized form, ensuring correctness without sacrificing performance.

9. Discuss the role of the Java Virtual Machine (JVM) in program execution.

The Java Virtual Machine (JVM) is a critical component of the Java platform that plays a pivotal role in the execution of Java programs. Here's an overview of its key functions and responsibilities:

1. Bytecode Execution

The JVM is responsible for executing Java bytecode, which is generated by the Java compiler (javac). This bytecode is platform-independent, allowing Java programs to be "write once, run anywhere."

2. Platform Independence

The JVM abstracts the underlying hardware and operating system, enabling Java programs to run on any device with a compatible JVM. This platform independence is one of Java's core strengths.

3. Memory Management

The JVM manages memory allocation and deallocation for Java programs. It includes:

- **Heap Memory**: Where objects are stored.
- **Stack Memory**: Where method calls and local variables are stored.
- **Garbage Collection**: The JVM automatically reclaims memory by removing objects that are no longer in use, preventing memory leaks.

4. Just-In-Time (JIT) Compilation

The JVM uses JIT compilation to improve performance. It translates frequently executed bytecode into native machine code at runtime, optimizing the code for the specific hardware.

6. Multithreading Support

The JVM provides built-in support for multithreading, allowing Java programs to execute multiple threads concurrently. This is essential for developing responsive and scalable applications.

7. Exception Handling

The JVM manages exception handling, allowing Java programs to gracefully handle runtime errors. It provides a robust mechanism for detecting and responding to exceptions.

9. Interpreting and Executing Bytecode
The JVM starts by interpreting bytecode, which means it reads and executes bytecode instructions one by one. As the program runs, the JVM identifies "hot" code sections and uses JIT compilation to optimize them.