Answer the following

1. Explain the Features of Java.

Java is a high-level, object-oriented programming language developed by **Sun Microsystems** (now owned by Oracle) and released in 1995. It is widely used for building applications across a variety of platforms, including web, mobile, desktop, and enterprise systems.

Java has several powerful features that make it a popular programming language. Here are some of the key features:

1. Object-Oriented

- **Everything is an Object**: Java is based on the object-oriented programming (OOP) paradigm. It uses concepts like classes, objects, inheritance, polymorphism, abstraction, and encapsulation, which allow developers to build modular, reusable, and maintainable code.

2. Platform-Independent

- **Write Once, Run Anywhere (WORA)**: Java programs are compiled into bytecode, which is platform-independent. The bytecode can be run on any system that has a Java Virtual Machine (JVM) installed, making it cross-platform.

3. Simple and Easy to Learn

- Java has a straightforward syntax similar to C++, but it eliminates complex features such as pointers and multiple inheritance, making it easier to learn and use.

4. Secure

- Java provides several security features such as access control and robust memory management. The JVM's security manager restricts untrusted code from performing unauthorized operations, and Java's built-in libraries provide encryption, secure communication, and authentication.

5. Robust

- Java is designed to eliminate certain types of programming errors like memory leaks and pointer-related issues. It provides strong memory management via automatic garbage collection, exception handling, and type-checking mechanisms.

6. Multithreaded

- Java supports multithreading, allowing multiple threads of execution to run simultaneously. This makes Java ideal for developing applications that require parallel processing, like games, animations, or server-based applications.

7. High Performance

- Although Java is an interpreted language (since it runs on the JVM), the use of Just-In-Time (JIT) compilers, bytecode optimization, and performance tuning techniques make Java highly performant.

8. Dynamic and Extensible

- Java is dynamic in nature and supports dynamic memory allocation, which reduces memory wastage and enhances performance. Java's libraries can be extended through third-party APIs and frameworks.

9. Distributed

- Java supports distributed computing. It has libraries (such as RMI and CORBA) for building distributed applications, making it ideal for building network-based applications.

10. Portable

- Java's portability stems from its platform independence. The same Java program can be executed on different platforms without needing modifications.

11. Automatic Memory Management (Garbage Collection)

- Java automatically manages memory using a built-in garbage collector. Developers don’t need to

manually deallocate memory, which reduces the chances of memory leaks.

2. Explain why Java is Platform Independent.

Java is considered platform-independent primarily due to its unique architecture, which allows Java programs to run on any device that has the Java Virtual Machine (JVM) installed. Here’s a breakdown of the reasons behind Java's platform independence:

1. Java Compilation to Bytecode:

- When Java source code is written (with a .java extension), it is compiled by the Java compiler (javac) into an intermediate form called **bytecode** (with a .class extension). This bytecode is not specific to any particular hardware or operating system.

2. Java Virtual Machine (JVM):

- The JVM is a crucial component that enables Java's platform independence. It is an abstract computing machine that interprets the bytecode and executes it on the host machine. Since the JVM is available for various platforms (Windows, macOS, Linux, etc.), the same bytecode can run on any platform with a compatible JVM.

3. Write Once, Run Anywhere (WORA):

- This famous slogan of Java captures its essence of platform independence. Developers can write their code once, compile it into bytecode, and run that bytecode on any platform that has a JVM, without needing to modify the code. This eliminates the issues of compatibility and allows for greater flexibility in deploying applications.

4. Standardized Java API:

- Java provides a rich set of standard libraries (Java APIs) that offer consistent functionality across different platforms. This standardization means that developers can rely on the same set of libraries to perform tasks regardless of the underlying operating system.

5. Abstracting Hardware Differences:

- The JVM abstracts the underlying hardware and operating system details. This means that developers do not need to worry about the specifics of the hardware they are running on, as the JVM handles these differences, making Java applications more portable.

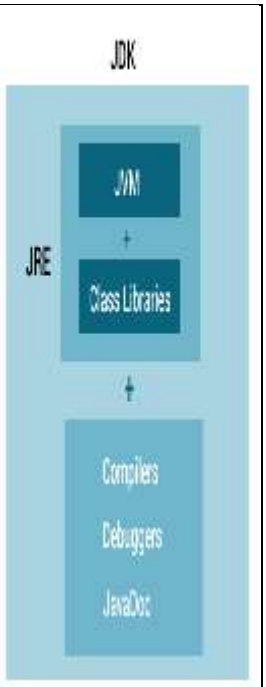
6. Dynamic Class Loading:

- Java supports dynamic loading of classes at runtime, which adds to its platform independence. This allows Java applications to adapt to different

environments by loading different classes based on the platform they are running on.

3. Differentiate between JDK,JRE and JVM.

Feature	JDK (Java Development Kit)	JRE (Java Runtime Environment)	JVM (Java Virtual Machine)
Definition	A software development kit used to develop Java applications. It includes tools for developing, debugging, and compiling Java programs.	A software environment that provides the libraries and resources needed to run Java applications.	A virtual machine that runs Java bytecode and converts it to machine-specific code.
Purpose	To develop and compile Java applications.	To provide the environment needed to run Java applications.	To execute the bytecode and convert it into machine-specific instructions.
Components	Includes JRE, compilers (`javac`), debuggers, and development tools.	Includes JVM and core libraries like java.util, java.io, etc.	Part of JRE, responsible for running Java applications.
Contains	JRE + development tools (javac, javadoc, etc.).	JVM + core Java libraries.	Classloader, bytecode verifier, interpreter, and JIT compiler.
Who Uses It?	Developers for writing, compiling, and debugging Java code.	End-users to run Java applications.	End-users indirectly; JVM runs behind the scenes when a Java program is executed.
Installation Size	Larger than JRE (contains extra development tools).	Smaller, as it contains only runtime resources.	Not a standalone entity, but part of the JRE.
Execution of Java Code	Compiles source code into bytecode (via `javac` compiler).	Executes the bytecode in a Java application.	Loads and runs the bytecode by converting it into machine-specific instructions.
Example Usage	Used by developers to write and compile Java programs.	Used by users to run compiled Java programs like desktop applications.	Internally used by both JDK and JRE to execute the Java bytecode.

Image	 JRE JVM + Class Libraries	 JDK JRE + Compilers + Debuggers ...	 JDK JRE JVM + Class Libraries + Compilers Debuggers JavaDoc