Part 1: Introduction to Java

1. What is Java? Explain its significance in modern software development.
Java is a high-level, object-oriented programming language developed by Sun Microsystems). It is designed to be platform-independent, meaning that once written, Java code can run on any device or operating system that has a Java Virtual Machine (JVM).

2. List and explain the key features of Java.
Platform Independence: Java code can be written once and run anywhere (thanks to the JVM).
Object-Oriented: Everything in Java is an object, which promotes modularity and reusability of code.
Automatic Memory Management: Java has garbage collection, which automatically manages memory by cleaning up unused objects.
Multi-threading: Java provides built-in support for multithreading, allowing developers to write programs that can execute multiple tasks concurrently.
Security: Java provides a robust security model that prevents malicious code from performing unsafe operations.
Robust Exception Handling: Java has a strong exception handling mechanism, ensuring that errors can be caught and handled efficiently.
Distributed Computing: Java supports networking capabilities for building distributed systems.

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

Compiled Languages:

- In a compiled language, the source code is directly translated into machine code(byte code) by a compiler.
- Java source code is first compiled into bytecode by the Java compiler (javac).

Interpreted Languages:

- In an interpreted language, the source code is translated into machine code line-by-line at runtime by an interpreter.
- **Java is a** hybrid language that combines both compiled and interpreted features:
- The bytecode is then executed by the JVM, which interprets or just-in-time compiles the bytecode into native machine code for the specific platform. This makes Java platform-independent.

4. Explain the concept of platform independence in Java.
Platform independence in Java refers to the ability of Java programs to run on any platform (Windows, Linux, macOS, etc.) without needing to be rewritten. This is achieved through the use of the Java Virtual Machine (JVM), which translates the compiled Java bytecode into platform-specific machine code at runtime. As long as the system has a JVM installed, the Java code can run on that system

5. What are the various applications of Java in the real world?
Java is used in a variety of applications in the real world, including:
Web Applications: Java is extensively used in web development, especially for large-scale enterprise applications (e.g., Spring framework).
Mobile Applications: Java is the primary language for developing Android applications.
Enterprise Applications: Java is used in backend systems for large organizations (e.g., banking, e-commerce).
Scientific Applications: Java is used in scientific computing due to its speed, security, and reliability.
Big Data: Java plays a significant role in big data technologies like Hadoop and Apache Spark.

Games and Interactive Media: Java is used in the development of cross-platform games and simulations.

Part 2: History of Java

1. Who developed Java and when was it introduced?
Java was developed by **James Gosling** and his team at Sun Microsystems in **1991**, and it was officially released in **1995**.
2. What was Java initially called? Why was its name changed?
Java was initially called **Oak**, named after an oak tree outside James Gosling's office. However, the name was changed to **Java** in 1995 due to trademark issues and also because Java was a name that could be associated with a global and dynamic technology (the word "Java" being associated with coffee, which reflects energy and productivity).
3. Describe the evolution of Java versions from its inception to the present.
Java 1.0 (1996): The first official version, with key features like platform independence, object-oriented programming, and a rich API.
Java 2 (J2SE) (1998): Introduced significant updates like the Swing GUI toolkit and the Collections Framework.
Java 5 (2004): Introduced generics, metadata annotations, enhanced for-loop, and more.
Java 6 (2006): Improved performance, introduced scripting support with Java Compiler API.
Java 7 (2011): Language enhancements, including the try-with-resources statement.
Java 8 (2014): Introduced lambdas, streams, and the new date and time API.
Java 9 (2017): Introduced the module system (Project Jigsaw).
Java 10 - 14 (2017–2020): Focused on performance improvements, garbage collection, and new language features.
Java 15 (2020) and beyond: Continuous improvement with features like record types, sealed classes, and pattern matching.
4. What are some of the major improvements introduced in recent Java versions?
Java 8: Lambdas, Streams API, new Date/Time API.
Java 9: Modular system (Project Jigsaw).
Java 10: Local variable type inference (`var` keyword).
Java 14: Introduced features like `records` and `pattern matching`.
Java 16: Enhanced performance with new garbage collectors.

5. How does Java compare with other programming languages like C++ and Python in terms of evolution and usability?
C++: Java is easier to learn and use compared to C++ because it does not require manual memory management, and it has a simpler syntax. However, C++ may be preferred for performance-critical systems.
Python: Java has a more complex syntax and requires explicit declaration of types, making it more verbose than Python. However, Java offers better performance and is better suited for large-scale systems and applications.

Part 3: Data Types in Java

1. Explain the importance of data types in Java.
Data types are essential in Java because they define the type and size of data that can be stored in memory. They help Java handle memory efficiently and provide safety through type checking, ensuring that variables are used correctly and consistently.

2. Differentiate between primitive and non-primitive data types.
Primitive Data Types: These are predefined by Java and include types like `int`, `char`, `float`, etc. They hold simple values and are stored directly in memory.
Non-Primitive Data Types: These include classes, interfaces, and arrays. They are more complex and can hold references to objects or groups of objects.
3. List and briefly describe the eight primitive data types in Java.
byte: 1 byte, range: -128 to 127.
short: 2 bytes, range: -32,768 to 32,767.
int: 4 bytes, range: -2^31 to 2^31 - 1.
long: 8 bytes, range: -2^63 to 2^63 - 1.
float: 4 bytes, single precision floating-point.
double: 8 bytes, double precision floating-point.
char: 2 bytes, used for storing characters (Unicode).
boolean: 1 bit, stores `true` or `false`.

4. Provide examples of how to declare and initialize different data types.
```
int a = 10;
double price = 19.99;
char letter = 'A';
boolean isJavaFun = true;
```

5. What is type casting in Java? Explain with an example.
Type casting in Java is the process of converting one data type to another. There are two types of casting:
Implicit Casting (Widening): Automatically converts a smaller type to a larger type.
Explicit Casting (Narrowing): Requires manual conversion.
Example:
```
// Implicit casting (widening)
int num = 100;
double d = num; // int to double

// Explicit casting (narrowing)
double price = 9.99;
int priceInt = (int) price; // double to int
```

Coding Questions on Data Types:

1. Write a Java program to declare and initialize all eight primitive data types and print their values.

Code:-

```
public class PrimitiveDataTypes {
        static int i;
   public static void main(String[] args) {


      // Declaring and initializing primitive data types
      byte b = 10;
      short s = 20;
      int i = 30;
      long l = 40L;
      float f = 50.5f;
      double d = 60.6;
      char c = 'A';
      boolean bool = true;
                System.out.println(i);
      System.out.println("Byte: " + b);
      System.out.println("Short: " + s);
      System.out.println("Int: " + i);
      System.out.println("Long: " + l);
      System.out.println("Float: " + f);
      System.out.println("Double: " + d);
      System.out.println("Char: " + c);
      System.out.println("Boolean: " + bool);
   }
}
```

2. Write a Java program that takes two integers as input and performs all arithmetic operations on them.

Code :-

```
import java.util.Scanner;

public class ArithmeticOperations {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);


      System.out.print("Enter first number: ");
      int num1 = scanner.nextInt();
      System.out.print("Enter second number: ");
      int num2 = scanner.nextInt();


      int sum = num1 + num2;
      int diff = num1 - num2;
      int product = num1 * num2;
      double division = (double) num1 / num2;
```

```
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + diff);
        System.out.println("Product: " + product);
        System.out.println("Division: " + division);
    }
}
```

3. Implement a Java program to demonstrate implicit and explicit type casting.

Code:-

```
public class TypeCasting {
    public static void main(String[] args) {
        // Implicit Type Casting (Widening)
        int intVar = 10;
        double doubleVar = intVar;  // No explicit cast needed
        System.out.println("Implicit casting (int to double): " + doubleVar);

        // Explicit Type Casting (Narrowing)
        double doubleValue = 9.99;
        int intValue = (int) doubleValue;  // Explicit cast needed
        System.out.println("Explicit casting (double to int): " + intValue);
    }
}
```

4. Create a Java program that converts a given integer to a double and vice versa using wrapper classes.

Code:-

```
public class WrapperClassConversion {
    public static void main(String[] args) {
        // Convert integer to double
        Integer intObj = 100;
        Double doubleObj = intObj.doubleValue();
        System.out.println("Integer to Double: " + doubleObj);

        // Convert double to integer
        Double doubleValue = 45.67;
        Integer intValue = doubleValue.intValue();
        System.out.println("Double to Integer: " + intValue);
    }
}
```

5. Write a Java program to swap two numbers using a temporary variable and without using a temporary variable.

Code:-

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

        int a = 5, b = 10;
        System.out.println("Before swapping (Using temp variable): a = " + a + ", b = " + b);
        int temp = a;
        a = b;
```

```
        b = temp;
        System.out.println("After swapping: a = " + a + ", b = " + b);
        a = 5;
        b = 10;
        System.out.println("\nBefore swapping (Without temp variable): a = " + a + ", b = " + b);
        a = a + b;
        b = a - b;
        a = a - b;
        System.out.println("After swapping: a = " + a + ", b = " + b);
    }
}
```

6. Create a Java program to check whether a given number is even or odd using command-line arguments.

Code:-

```
public class EvenOdd {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Please provide a number as a command-line argument.");
            return;
        }

        int number = Integer.parseInt(args[0]);

        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }
    }
}
```

Part 4: Java Development Kit (JDK)

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

- JDK (Java Development Kit):
  - The JDK is a software development environment used for developing Java applications and applets. It includes the JRE, a compiler (javac), and a variety of other tools (like the Java debugger and JavaDoc).
  - Essentially, it's what you need if you want to *create* Java programs.
- JRE (Java Runtime Environment):
  - The JRE provides the environment to *run* Java applications. It contains the JVM, core Java classes, and supporting files.
  - If you only want to run existing Java programs, you only need the JRE.
- JVM (Java Virtual Machine):
  - The JVM is the heart of the Java platform. It's responsible for executing Java bytecode. It's an abstract computing machine that enables Java's "write once, run anywhere" capability.
  - The JVM is within the JRE.

2. Explain the main components of JDK.

- javac (Java Compiler):
  - Translates Java source code (.java files) into bytecode (.class files).
- java (Java Interpreter):
  - Launches the JVM and executes the bytecode.
- JRE (Java Runtime Environment):
  - As described above, it provides the runtime environment.
- Javadoc:
  - Generates API documentation from Java source code comments.
- jdb (Java Debugger):
  - Helps to debug Java programs.

3. Write a simple Java program to print "Hello, World!" and explain its structure.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

 public class HelloWorld { ... }:

- Defines a public class named HelloWorld. Java programs are organized into classes.

 public static void main(String[] args) { ... }:

- The main method is the entry point of the program. When you run the program, the JVM starts executing code from this method.
- public: Makes the method accessible from anywhere.
- static: Allows the method to be called without creating an instance of the class.
- void: Indicates that the method doesn't return any value.

- String[] args: An array of strings that can be used to pass command-line arguments to the program.

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

- Prints the string "Hello, World!" to the console.
- System.out: Standard output stream.
- println(): A method that prints a line of text.

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

- OpenJDK:
  - An open-source implementation of the Java SE platform.
  - Developed and maintained by the OpenJDK community.
  - Often used as the basis for other JDK distributions.
  - Released under the GPL license.
- Oracle JDK:
  - A commercial distribution of the JDK by Oracle.
  - Based on OpenJDK, but may include additional features and optimizations.
  - Historically, Oracle JDK has had a less permissive license than OpenJDK. However, recent Oracle JDK releases are also under the GPL license.
  - Oracle provides commercial support.
- Key Differences:
  - Licensing (has become very similar recently)
  - Support (Oracle provides commercial support)
  - Release cadence, and certain enterprise features.

7. Explain how Java programs are compiled and executed.

1. Writing the Source Code:
   - First write the Java code in a text editor and save it with a .java extension (e.g., HelloWorld.java).
2. Compilation:
   - Then use the javac compiler to compile the source code: javac HelloWorld.java.
   - If there are no errors, the compiler generates a bytecode file named HelloWorld.class.
3. Execution:
   - After that use the java command to run the program: java HelloWorld.
   - The JVM loads the HelloWorld.class file.
   - The JVM then executes the bytecode in the main method.

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

- JIT Compilation:
  - The JVM uses a JIT compiler to improve the performance of Java programs.
  - Instead of interpreting bytecode line by line, the JIT compiler analyzes the bytecode and compiles frequently used parts of the code into native machine code during runtime.
  - This native code can be executed directly by the CPU, resulting in faster execution.
- Performance Improvement:

    - JIT compilation significantly improves the performance of Java programs, especially for long-running applications that execute the same code repeatedly.
    - It bridges the gap between the platform independence of bytecode and the performance of native code.

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

- Platform Independence:
    - The JVM is the key to Java's "write once, run anywhere" capability. It provides an abstraction layer between the Java program and the underlying operating system and hardware.
- Bytecode Execution:
    - The JVM is responsible for loading and executing Java bytecode.
- Memory Management:
    - The JVM manages memory allocation and garbage collection, freeing developers from manual memory management.
- Security:
    - The JVM provides a secure environment for running Java programs. It enforces security policies and prevents malicious code from accessing system resources.
- Runtime Environment:
    - The JVM provides the runtime environment that Java programs need to execute. This includes handling exceptions, managing threads, and providing access to system resources.