JAVA

Assignment 1

By Gaurav Rathod

Introduction

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

Ans. Java is a high-level, object-oriented programming language developed by **Sun Microsystems** (now owned by **Oracle Corporation**) and released in **1995**. It is known for its **platform independence**, as it follows the "Write Once, Run Anywhere" (**WORA**) principle, meaning Java applications can run on any system that has a Java Virtual Machine (**JVM**).

Significance of Java in Modern Software Development

Java remains a cornerstone of modern software development due to its versatility, security, and scalability. Here's why Java is important today:

1. Platform Independence

Java's bytecode runs on the JVM, making it possible to execute Java applications on various platforms (Windows, macOS, Linux) without modification.

2. Object-Oriented Programming (OOP)

Java follows OOP principles (Encapsulation, Inheritance, Polymorphism, Abstraction), which help in creating modular, reusable, and maintainable code.

3. Strong Community and Ecosystem

With millions of developers and vast online resources, Java has a large ecosystem, including frameworks like Spring, Hibernate, and Jakarta EE that simplify enterprise software development.

4. Enterprise and Web Applications

Java is extensively used in enterprise-level applications, banking, finance, healthcare, and e-commerce, thanks to frameworks like Spring Boot and Java EE.

5. Mobile Development (Android)

Java is the primary language for Android app development using Android Studio, making it crucial for mobile software development.

6. Security and Reliability

Java includes built-in security features like automatic memory management, garbage collection, and exception handling, making it one of the most secure languages.

7. Scalability and Performance

Java supports multi-threading and can handle high-performance applications, making it ideal for cloud computing, microservices, and large-scale applications.

8. Integration with Modern Technologies

Java integrates well with modern technologies like Big Data (Apache Hadoop), Artificial Intelligence, Blockchain, and Cloud Computing

Q2. List and explain the key features of Java. ?

Ans. Key Features of Java

Java is a powerful and widely-used programming language that offers several key features, making it ideal for modern software development. Below are the most significant features of Java:

1. Platform Independence ("Write Once, Run Anywhere" - WORA)

- Java programs are **compiled into bytecode**, which can run on any system with a **Java Virtual Machine (JVM)**.
- Unlike languages like C/C++, Java does **not require recompilation** for different platforms.
- This makes Java highly **portable** and useful for cross-platform applications.

2. Object-Oriented Programming (OOP)

Java follows the **OOP paradigm**, which helps in writing **modular, reusable, and maintainable** code. The four main principles of OOP in Java are:

- **Encapsulation** – Data hiding and access control.
- **Abstraction** – Hiding implementation details from the user.
- **Inheritance** – Code reuse through hierarchical relationships.
- **Polymorphism** – Allowing a single function to have multiple implementations.

3. Simple and Easy to Learn

- Java's syntax is inspired by C and C++, but with **simplified memory management** (no need for explicit pointers).
- The removal of complex features like **explicit memory allocation (malloc, free), multiple inheritance (replaced with interfaces)** makes Java more beginner-friendly.

4. Robust and Secure

- Java has strong **memory management** with **automatic garbage collection**, reducing memory leaks.
- It features **exception handling** to manage runtime errors effectively.
- Java provides **security features** like **bytecode verification, class loading restrictions, and a security manager** to prevent unauthorized access.

5. Multithreading and Concurrency

- Java supports **multithreading**, allowing multiple tasks to run in parallel, improving performance.
- The **Thread class and Runnable interface** make it easy to implement multithreading.
- Used in applications like **gaming, real-time systems, and large-scale data processing**.

Q3. What is the difference between compiled and interpreted languages? Where does Java fit in?

Ans. Difference Between Compiled and Interpreted Languages

Programming languages are broadly classified into **compiled** and **interpreted** languages based on how their code is executed.

Feature	Compiled Language	Interpreted Language
Execution Process	Translates entire code into machine code **before execution**	Translates and executes code **line-by-line** at runtime
Speed	Faster execution (since code is pre-compiled)	Slower execution (due to line-by-line interpretation)
Error Detection	Errors detected at compile-time (before execution)	Errors detected at runtime (during execution)
Portability	Requires recompilation for different platforms	More portable as it runs directly on the interpreter
Examples	C, C++, Rust, Go	Python, JavaScript, PHP

Q 4. Explain the concept of platform independence in Java.?

Ans **Platform Independence in Java**

Platform Independence is one of the most important features of Java, which makes it highly popular in software development.

Platform independence means that a Java program can run on any operating system or device without needing to be rewritten or recompiled. In simple terms, the same Java code can run on **Windows, Linux, macOS**, or any other system that supports **Java Virtual Machine (JVM)**.

Java achieves platform independence through its unique **Compilation and Execution Process:**

1. **Source Code Compilation:**

    - The Java compiler (**javac**) converts the human-readable **Java source code** (`.java`) into **bytecode** (`.class`).
    - Bytecode is an intermediate code that is **not dependent on any operating system or hardware**.
2. **Execution on JVM:**

    The generated bytecode is executed by the **Java Virtual Machine (JVM)**.

    - Each operating system has its own version of JVM, which converts the bytecode into **machine-specific code** at runtime.

Diagram Representation:

```
scss
CopyEdit
Java Code (.java)
        ↓
    Compiler (javac)
        ↓
    Bytecode (.class)
        ↓
    JVM (Platform-Specific)
        ↓
Machine Code (Output)
```

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

Ans. Real-World Applications of Java

Java is a versatile programming language used in a wide range of applications, from web development to artificial intelligence. Below are some of the major real-world applications of Java:

1. Web Applications

Java is widely used for developing dynamic web applications using frameworks like:

- **Spring Boot** (for enterprise-level applications)
- **JavaServer Faces (JSF)**
- **Hibernate** (for database management)
- **Struts** (for MVC-based development)

Examples:

- LinkedIn
- Amazon
- eBay

---

2. Mobile Applications (Android Development)

Java is the **primary language** for Android app development. The **Android SDK** uses Java to create mobile applications.

Examples:

- Instagram
- WhatsApp
- Spotify (Android version)

---

3. Enterprise Applications

Java is widely used in **banking, finance, and corporate applications** due to its security and scalability. The **Spring** and **Jakarta EE (formerly Java EE)** frameworks are commonly used for enterprise solutions.

Examples:

- Banking Systems (JPMorgan Chase, Citibank)
- ERP (Enterprise Resource Planning) software
- CRM (Customer Relationship Management) applications

---

4. Cloud-Based Applications ☁

Java is widely used in cloud computing due to its cross-platform compatibility and security. Many cloud providers offer Java-based solutions.

Examples:

- AWS (Amazon Web Services)
- Microsoft Azure
- Google Cloud Platform (GCP)

Part 2: History of Java

1. Who developed Java and when was it introduced?

Ans. Java was developed by **James Gosling** and his team at **Sun Microsystems** in **1991**. It was officially released in **1995**.

Initially, Java was called **"Oak"**, but the name was later changed to **Java** due to trademark issues. The goal was to create a platform-independent, object-oriented programming language for embedded systems, but it later became widely used in **web, mobile, enterprise, and cloud applications**.

After **Sun Microsystems** was acquired by **Oracle Corporation** in **2010**, Oracle took over Java's development and maintenance.

2. What was Java initially called? Why was its name changed?
3. • The name **Oak** was inspired by an **oak tree** outside **James Gosling's** office at **Sun Microsystems**.
4. • However, **"Oak" was already trademarked** by another company, so the developers had to choose a new name.
5. • They eventually settled on **"Java"**, inspired by **Java coffee**, as the team often drank coffee while working on the project.

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

Java has undergone significant evolution since its inception, introducing numerous features and enhancements across its versions. Here's an overview of Java's version history:

Version	Release Date	Notable Features
JDK 1.0	January 23, 1996	Initial release with basic language features and core libraries.
JDK 1.1	February 19, 1997	Introduced inner classes, JavaBeans, JDBC, RMI, and reflection.
Java SE 7	July 28, 2011	Introduced the try-with-resources statement, diamond operator, and support for dynamic languages.
Java SE 8	March 18, 2014	Added lambda expressions, the Stream API, a new date and time API, and Nashorn JavaScript engine.
Java SE 9	September 21, 2017	Introduced the module system (Project Jigsaw), JShell (REPL), and improvements to the Stream API.
Java SE 10	March 20, 2018	Introduced local-variable type inference (var keyword) and enhancements to garbage collection.
Java SE 11	September 25, 2018	Long-Term Support (LTS) release; added new HTTP client, local-variable syntax for lambda parameters, and removed JavaFX from the JDK.
Java SE 12	March 19, 2019	Introduced switch expressions (preview) and microbenchmark suite.
Java SE 17	September 14, 2021	LTS release; included sealed classes, pattern matching for switch (preview), and enhanced pseudorandom number generators.
Java SE 18	March 22, 2022	Introduced a simple web server and code snippets in Java API documentation.
Java SE 19	September 20, 2022	Added virtual threads (preview) and structured concurrency (preview).
Java SE 20	March 21, 2023	Continued incubation and preview features, including pattern matching for switch and record patterns.
Java SE 21	September 19, 2023	LTS release; introduced sequenced collections, string templates (preview), and virtual threads as a standard feature.

The evolution of Java reflects its adaptability and responsiveness to the changing needs of developers and the software industry, continually introducing features that enhance performance, security, and developer productivity.

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

Java has introduced several significant improvements in its recent versions, enhancing developer productivity, performance, and language capabilities. Here's an overview of some major enhancements:

Java SE 20 (March 21, 2023)

- **Scoped Values (Incubator):** Introduced the `ScopedValue` class for safe and efficient data sharing across threads, improving thread-local data handling.
- **Unicode 15.0 Support:** Updated to support Unicode 15.0, ensuring compatibility with the latest character sets and symbols.
- **SSL Parameters Enhancements:** Added methods `getNamedGroups` and `setNamedGroups` in the `SSLParameters` class to prioritize key exchange algorithms in TLS/DTLS connections.
- **Deprecated Thread Methods:** The `suspend`, `resume`, and `stop` methods in the `Thread` class now throw `UnsupportedOperationException`, discouraging their use due to inherent risks.

Java SE 21 (September 19, 2023)

- **Pattern Matching for `switch`:** Officially integrated pattern matching in `switch` statements, allowing more expressive and safer type checks.
- **Record Patterns:** Enabled deconstruction of record values for more concise and readable data retrieval.
- **Virtual Threads:** Introduced virtual threads to simplify concurrent programming by reducing the complexity of thread management.
- **Sequenced Collections:** Added `SequencedCollection`, `SequencedSet`, and `SequencedMap` interfaces to define collections with a well-defined element order.
- **Key Encapsulation Mechanism API:** Introduced an API to enhance secure operations with symmetric cryptosystems.
- **Unnamed Classes and Instance `main` Methods (Preview):** Allowed defining `main` methods without explicitly naming a class, reducing boilerplate code.

Java SE 22 (March 19, 2024)

- **Unnamed Patterns and Variables:** Officially incorporated unnamed patterns and variables, initially previewed in Java 21, to simplify code by using the underscore (_) as a placeholder.
- **Foreign Function & Memory API:** Enhanced the API to allow Java applications to interact more efficiently with non-Java code and memory, improving performance and interoperability.

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

Comparison of Java with C++ and Python

Java, C++, and Python are three of the most popular programming languages, each with unique strengths. Below is a comparison based on **evolution, usability, and key features**.

1. Evolution of Java, C++, and Python

Language	Initial Release	Major Evolutionary Changes
Java	1995 (by Sun Microsystems)	Introduced JVM (platform independence), automatic memory management (garbage collection), multi-threading, and security improvements. Recent updates include virtual threads, pattern matching, and records.
C++	1985 (by Bjarne Stroustrup)	Extended from C with object-oriented programming. Major updates include C++11 (smart pointers, lambda expressions), C++17 (structured bindings), and C++20 (concepts, coroutines, and modules).
Python	1991 (by Guido van Rossum)	Focused on readability and simplicity. Major updates include Python 3 (Unicode support, type hints), async/await, and improved standard libraries.

---

2. Usability Comparison

Feature	Java	C++	Python
Syntax Simplicity	Medium complexity	Complex (manual memory management)	Very simple (easy-to-read syntax)
Performance	Good (JIT-compiled)	High (closer to hardware)	Slower (interpreted, but optimized with JIT in PyPy)
Memory Management	Automatic (Garbage Collection)	Manual (pointers, memory allocation)	Automatic (Garbage Collection)
Platform Independence	High (JVM allows cross-platform execution)	Low (compiled for specific OS)	High (interpreted, but may need platform-specific optimizations)
Concurrency Support	Strong (multi-threading with JVM support)	Strong (threading and parallelism)	Limited due to GIL (Global Interpreter Lock)
Use Cases	Enterprise apps, web apps, Android development	System programming, game engines, high-performance applications	Data science, AI, web development, automation

---

3. Key Strengths of Each Language

- **Java**: Best for large-scale **enterprise applications**, Android development, and cloud computing due to strong memory management and cross-platform support.
- **C++**: Preferred for **high-performance systems**, game engines, and embedded systems where control over memory is crucial.
- **Python**: Ideal for **data science, AI, and automation**, as well as beginner-friendly programming.

Part 3: Data Types in Java

1. Explain the importance of data types in Java.

Importance of Data Types in Java

Data types in Java are essential for **memory efficiency, type safety, performance optimization, and preventing errors**. They define the type of data a variable can store and how much memory it will use.

- **Memory Efficiency** – Prevents unnecessary memory usage by allocating the correct amount of space.
- **Type Safety** – Ensures that variables hold only valid values, reducing errors.
- **Performance Optimization** – Choosing the right data type improves execution speed.
- **Code Reliability** – Makes code more readable, maintainable, and bug-free.
- **Prevention of Errors** – Avoids issues like integer overflow and precision loss.

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

Difference Between Primitive and Non-Primitive Data Types in Java

Java data types are categorized into **primitive** and **non-primitive** types, each serving different purposes in memory management and data handling.

Feature	Primitive Data Types □	Non-Primitive Data Types □
Definition	Basic built-in data types that store simple values.	Complex data types that store objects and collections.
Examples	`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`	`String`, `Array`, `Class`, `Interface`, `List`, `HashMap`
Memory Storage	Stored in **stack memory** for fast access.	Stored in **heap memory** (references stored in stack).
Stores	Actual values directly.	Memory reference to an object.

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

Eight Primitive Data Types in Java
Java has **eight primitive data types**, which store simple values and are categorized as **integer, floating-point, character, and boolean** types.

Data Type	Size	Default Value	Description
byte	1 byte (8 bits)	`0`	Stores small integers (-128 to 127), useful for memory optimization.
short	2 bytes (16 bits)	`0`	Stores medium-sized integers (-32,768 to 32,767).
int	4 bytes (32 bits)	`0`	Most commonly used for whole numbers (-2^{31} to $2^{31}-1$).
long	8 bytes (64 bits)	`0L`	Stores large integers (-2^{63} to $2^{63}-1$), used for big numbers.
float	4 bytes (32 bits)	`0.0f`	Stores decimal numbers (single-precision, ~6-7 decimal digits).
double	8 bytes (64 bits)	`0.0d`	Stores decimal numbers (double-precision, ~15-16 decimal digits).
char	2 bytes (16 bits)	`\u0000`	Stores a single character (Unicode-supported). Example: `'A'`.
boolean	1 bit (size depends on JVM)	`false`	Stores `true` or `false`, used for logical operations.

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

```
public class DataTypesExample {

    public static void main(String[] args) {

        byte b = 100;

        short s = 20000;

        int i = 500000;

        long l = 10000000000L;

        float f = 5.75f;

        double d = 19.99;

        char c = 'J';

        boolean bool = true;

        String text = "Java is awesome!";


        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);

        System.out.println("String: " + text);

    }

}
```

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

Type Casting in Java

Type casting in Java is the process of converting one data type into another. It is classified into two types:

1. Implicit (Widening) Type Casting

- **Automatically** converts a smaller data type to a larger data type.
- **No data loss** occurs.
- **Example:** `byte → short → int → long → float → double`

2. Explicit (Narrowing) Type Casting

- **Manually** converts a larger data type to a smaller data type.
- **Data loss** may occur if the value exceeds the smaller type's range.
- **Example:** `double → float → long → int → short → byte`

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

Wrapper Classes in Java

In Java, **wrapper classes** are used to convert **primitive data types** into **objects**. They provide a way to use primitive values as objects, which is useful in **collections, generics, and object-oriented programming**.

---

1. List of Wrapper Classes

Each primitive data type has a corresponding wrapper class in **`java.lang`** package:

Primitive Type	**Wrapper Class**
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

---

2. Why Use Wrapper Classes?

Collections & Generics – Collections (e.g., `ArrayList<Integer>`) require objects, not primitives.
Null Values Support – Unlike primitives, wrapper objects can be `null`.
Utility Methods – Provide useful methods (`parseInt()`, `valueOf()`, etc.).
Autoboxing & Unboxing – Java automatically converts primitives to objects and vice versa.

6. What is the difference between static and dynamic typing? Where does Java stand?
7. **Static vs. Dynamic Typing: Key Differences**

Feature	Static Typing	Dynamic Typing
Definition	Type checking is done **at compile-time**.	Type checking is done **at runtime**.
Declaration	Variables must have a **fixed type** at declaration.	Variables **can change type** dynamically.
Error Detection	Errors are caught **before execution (compile-time)**.	Errors may appear **during execution (runtime)**.
Performance	**Faster** due to early type checking.	**Slower** because types are checked at runtime.
Flexibility	**Less flexible**, but safer.	**More flexible**, but riskier.
Example Languages	Java, C, C++, Swift	Python, JavaScript, Ruby

Coding Questions on Data Types:

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

```
public class PrimitiveDataTypes {

  public static void main(String[] args) {

    byte byteValue = 127;

    short shortValue = 32767;

    int intValue = 2147483647;

    long longValue = 9223372036854775807L;

    float floatValue = 3.14f;

    double doubleValue = 3.141592653589793;

    char charValue = 'A';

    boolean booleanValue = true;



    System.out.println("byte value: " + byteValue);

    System.out.println("short value: " + shortValue);

    System.out.println("int value: " + intValue);

    System.out.println("long value: " + longValue);
```

```
        System.out.println("float value: " + floatValue);

        System.out.println("double value: " + doubleValue);

        System.out.println("char value: " + charValue);

        System.out.println("boolean value: " + booleanValue);

    }

}
```

Output

```
byte value: 127

short value: 32767

int value: 2147483647

long value: 9223372036854775807

float value: 3.14

double value: 3.141592653589793

char value: A

boolean value: true
```

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

```
import java.util.Scanner;

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

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


      int sum = num1 + num2;
      int difference = num1 - num2;
      int product = num1 * num2;
```

```
        int quotient = num1 / num2;

        System.out.println("\nResults:");
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.println("Quotient: " + quotient);
        scanner.close();
    }
}
```

```
Output  Enter first integer: 12
 Enter second integer: 5
  Results:
  Sum: 17
  Difference: 7
  Product: 60
  Quotient: 2
```

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

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

        int intValue = 100;
        double doubleValue = intValue;

        System.out.println("Implicit Type Casting:");
        System.out.println("Integer value: " + intValue);
        System.out.println("Converted to double: " + doubleValue);


        double doubleValue2 = 99.99;
        int intValue2 = (int) doubleValue2;

        System.out.println("\nExplicit Type Casting:");
        System.out.println("Double value: " + doubleValue2);
        System.out.println("Converted to integer: " + intValue2);

        int largeValue = 130;
        byte byteValue = (byte) largeValue;

    }
}
```

Output

```
Implicit Type Casting:
Integer value: 100
Converted to double: 100.0

Explicit Type Casting:
Double value: 99.99
Converted to integer: 99
```

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

```
public class WrapperClassConversion {
    public static void main(String[] args) {
        Integer intValue = 42;
        Double doubleValue = intValue.doubleValue();

        System.out.println("Integer to Double Conversion:");
        System.out.println("Integer value: " + intValue);
        System.out.println("Converted to double: " + doubleValue);


        Double doubleValue2 = 99.99;
        Integer intValue2 = doubleValue2.intValue();

        System.out.println("\nDouble to Integer Conversion:");
        System.out.println("Double value: " + doubleValue2);
        System.out.println("Converted to integer: " + intValue2);
    }
}
```

Output

```
Integer to Double Conversion:
Integer value: 42
Converted to double: 42.0

Double to Integer Conversion:
Double value: 99.99
Converted to integer: 99
```

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

```
import java.util.Scanner;
```

```
public class SwapNumbers {
  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();
    System.out.println("\nBefore Swapping:");
    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);
    int temp = num1;
    num1 = num2;
    num2 = temp;

    System.out.println("\nAfter Swapping (using a temporary variable):");
    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);

    num1 = num1 + num2; // Step 1
    num2 = num1 - num2; // Step 2
    num1 = num1 - num2; // Step 3

    System.out.println("\nAfter Swapping (without using a temporary variable):");
    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);

    scanner.close();
  }
}
```

Output

```
Enter first number: 10
Enter second number: 20

Before Swapping:
num1 = 10
num2 = 20

After Swapping (using a temporary variable):
num1 = 20
num2 = 10

After Swapping (without using a temporary variable):
num1 = 10
num2 = 20
```

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

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

    if (args.length != 1) {
      System.out.println("Please provide exactly one integer as a command-line argument.");
      return;
    }

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

      if (number % 2 == 0) {
        System.out.println(number + " is an even number.");
      } else {
        System.out.println(number + " is an odd number.");
      }
    } catch (NumberFormatException e) {
      System.out.println("Invalid input! Please enter a valid integer.");
    }
  }
}
```

Output

```
java EvenOrOdd 10
10 is an even number.
```