**JAVA ASSIGNMENT 1**

**Name : Ragini Singh**

**Cohort : Jensen Huang**

**Roll No : 150096724023**

**Teacher Name : Dr Ritu Jain**

**Assignment Name : Java Assignment 1**

**Date : 08/01/2025**

**Que 1. Why is Java considered as a platform independent language.?**

**ANSWER :** Java is considered platform-independent because of the concept of "Write Once, Run Anywhere". The key to this is the Java Virtual Machine (JVM). Java code is compiled into bytecode, which is not platform-dependent. The bytecode can be executed on any machine that has a JVM, making Java platform-independent. The JVM interprets and runs the bytecode on different operating systems without needing to modify the source code.

**Que 2. Features of Java**

* **Simple: Java is designed to be easy to learn and use. Its syntax is similar to C++, but it eliminates certain complexities (like pointers).**

* **Object-Oriented: Java is based on object-oriented principles, which makes it easier to design modular and maintainable software.**

* **Platform-Independent: As mentioned, Java can run on any platform that has a JVM.**

* **Secure: Java provides a secure environment for executing code by using features like bytecode verification, runtime security checks, and the Security Manager.**

* **Robust: Java has strong memory management (via garbage collection) and error-handling features like exceptions.**
* **Multithreaded: Java supports multithreading, which allows multiple tasks to be executed concurrently.**

* **Distributed Computing: Java has libraries for distributed computing, making it easier to work in networked environments.**

* **High Performance: Java includes features like Just-In-Time (JIT) compilers to boost performance.**

* **Dynamic: Java can dynamically load classes at runtime, allowing flexibility in program execution.**

**Que 3. Terms in Java**

* **Keyword: Reserved words in Java that have predefined meanings and cannot be used as identifiers. Examples: int, class, public.**

**>>   Identifier: Names used to identify variables, functions, classes, etc. They must begin with a letter, underscore, or dollar sign and can be followed by letters, numbers, underscores, or dollar signs.**

**>> Literals: Constants in Java that represent fixed values like numbers, characters, or strings.**

**Que 4. Operator Precedence and Associativity**

* **Operator Precedence determines the order in which operators are evaluated in an expression.**
* **Associativity refers to the direction in which operators of the same precedence are evaluated (left to right or right to left).**

**For example:**

* **\* and / have higher precedence than + and -.**
* **++ (postfix) has higher precedence than + or -.**

**Que 5 . Operator Precedence in Java**

* Here’s a chart of operator precedence in Java (from highest to lowest):

| **Operator Type** | **Operators** |
| --- | --- |
| Postfix | expr++, expr-- |
| Unary | ++expr, --expr, +, -, ~, ! |
| Multiplicative | \*, /, % |
| Additive | +, - |
| Relational | <, >, <=, >=, instance of |
| Equality | ==, != |
| Logical AND | && |
| Logical OR | ` |
| Conditional | ?: (Ternary) |
| Assignment | =, +=, -=, \*=, /=, %= |

**Que 6. Primitive Data Types in Java**

Java has 8 primitive data types:

| **Type** | **Size** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| byte | 1 byte | -128 to 127 | 0 |
| short | 2 bytes | -32,768 to 32,767 | 0 |
| int | 4 bytes | -2^31 to 2^31-1 | 0 |
| long | 8 bytes | -2^63 to 2^63-1 | 0L |
| float | 4 bytes | ±3.40282347E+38 to ±1.40239846E-45 | 0.0f |
| double | 8 bytes | ±1.7976931348623157E+308 to ±4.9E-324 | 0.0d |
| char | 2 bytes | 0 to 65,535 (Unicode) | '\u0000' |
| Boolean | 1 bit | true or false | false |

**Que 7. WAP to demonstrate how to create variables of different types**

public class Variable Demo {

public static void main(String[] ergs) {

byte b = 100;

short s = 30000;

int i = 123456;

long l = 123456789L;

float f = 10.5f;

double d = 20.99;

char c = 'A';

Boolean bool = true;

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);

}

}

**Que 8.** **Implicit Type Conversion, Explicit Type Conversion, and Type Promotion in Java**

**>> Implicit Type Conversion**: Also called type coercion, this is automatically done by Java when a smaller type is converted to a larger type (e.g., int to long).

>> **Explicit Type Conversion**: Also known as type casting, this is done manually by the programmer using casting syntax, usually when a larger type is converted to a smaller type.

>>**Type Promotion**: Happens when smaller data types like byte, short, or char are automatically promoted to int in arithmetic operations.

**Que9. WAP to demonstrate explicit type conversion**

public class Explicit Conversion {

public static void main(String[] ergs) {

double d = 9.99;

int i = (int) d; // Explicit conversion

System.out.println("Double: " + d);

System.out.println("Integer: " + i);

}

}

**Que10.** **WAP to demonstrate implicit type conversion**

public class ImplicitConversion {

public static void main(String[] args) {

int x = 100;

long y = x; // Implicit conversion from int to long

System.out.println("Int: " + x);

System.out.println("Long: " + y);

}

}

**Que11. Arithmetic Operators in Java**

Java supports the following arithmetic operators:

* + (Addition)
* - (Subtraction)
* \* (Multiplication)
* / (Division)
* % (Modulus)

**Que12.** **WAP to demonstrate arithmetic operators in Java**

public class ArithmeticOperators {

public static void main(String[] args) {

int a = 10, b = 5;

System.out.println("Addition: " + (a + b));

System.out.println("Subtraction: " + (a - b));

System.out.println("Multiplication: " + (a \* b));

System.out.println("Division: " + (a / b));

System.out.println("Modulus: " + (a % b));

}

}

**Que13. WAP to demonstrate increment and decrement operators**

public class IncrementDecrement {

public static void main(String[] args) {

int x = 5;

System.out.println("Post-increment: " + x++);

System.out.println("Pre-increment: " + ++x);

System.out.println("Post-decrement: " + x--);

System.out.println("Pre-decrement: " + --x);

}

}

**Que14.** **WAP to demonstrate compound assignment operator**

public class CompoundAssignment {

public static void main(String[] args) {

int x = 5;

x += 3; // x = x + 3

System.out.println("After +=: " + x);

x \*= 2; // x = x \* 2

System.out.println("After \*=: " + x);

}

}

**Que15. Relational Operators in Java**

Relational operators are used to compare two values. They return a boolean value:

* == (Equal to)
* != (Not equal to)
* > (Greater than)
* < (Less than)
* >= (Greater than or equal to)
* <= (Less than or equal to)

**Que16. WAP to demonstrate relational operators**

public class RelationalOperators {

public static void main(String[] args) {

int x = 10, y = 5;

System.out.println("x == y: " + (x == y));

System.out.println("x != y: " + (x != y));

System.out.println("x > y: " + (x > y));

System.out.println("x < y: " + (x < y));

System.out.println("x >= y: " + (x >= y));

System.out.println("x <= y: " + (x <= y));

}

}

**Que17. Logical Operators in Java**

Java supports three logical operators:

* && (Logical AND)
* || (Logical OR)
* ! (Logical NOT)

**Que18.** **WAP to demonstrate logical operators in Java\**

public class LogicalOperators {

public static void main(String[] args) {

boolean a = true, b = false;

System.out.println("a && b: " + (a && b));

System.out.println("a || b: " + (a || b));

System.out.println("!a: " + (!a));

}

}

**Que19. Short-circuit operators in Java**

Java also supports short-circuit versions of logical operators:

* && (AND) - returns false as soon as one operand is false.
* || (OR) - returns true as soon as one operand is true.

**Que20.** **WAP to demonstrate short-circuit operators**

public class ShortCircuitOperators {

public static void main(String[] args) {

int a = 5, b = 0;

if (a != 0 && (b / a > 0)) { // Short-circuit, b/a won't be evaluated

System.out.println("True");

} else {

System.out.println("False");

}

}

}

**Que21**. **Conditional Operator (Ternary Operator)**

The **conditional operator** is a shorthand for if-else. Syntax:

If the condition is true, expression1 is evaluated; otherwise, expression2 is evaluated.

**Que22. WAP to find greater of two numbers using conditional operator**

public class GreaterNumber {

public static void main(String[] args) {

int a = 10, b = 5;

int greater = (a > b) ? a : b;

System.out.println("Greater number: " + greater);

}

}

**Que23**. **Conditional Statements in Java**

* **if**: Executes a block of code if the condition is true.
* **if-else**: Executes one block of code if true, another if false.
* **Nested if-else**: An if statement inside another if.
* **else-if ladder**: A series of else-if conditions to check multiple possibilities.
* **switch-case**: Matches an expression against multiple cases.

**Que24.** **WAP to find greater of two numbers using if-else statement**

public class GreaterNumberIfElse {

public static void main(String[] args) {

int a = 10, b = 20;

if (a > b) {

System.out.println("Greater number is: " + a);

} else {

System.out.println("Greater number is: " + b);

}

}

}

**Que25. WAP to find whether the inputted number is even or odd**

import java.util.Scanner;

public class EvenOdd {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a number: ");

int num = sc.nextInt();

if (num % 2 == 0) {

System.out.println(num + " is Even");

} else {

System.out.println(num + " is Odd");

}

}

}

**Que26**. **WAP to find the greatest among three numbers using if-else**

import java.util.Scanner;

public class GreatestNumber {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter three numbers: ");

int a = sc.nextInt(), b = sc.nextInt(), c = sc.nextInt();

if (a > b && a > c) {

System.out.println("Greatest number is: " + a);

} else if (b > a && b > c) {

System.out.println("Greatest number is: " + b);

} else {

System.out.println("Greatest number is: " + c);

}

}

}

**Que27**. **Using Scanner Class for User Input**

import java.util.Scanner;

public class UserInput {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a number: ");

int num = sc.nextInt();

System.out.println("You entered: " + num);

}

}

Scanner supports multiple methods for different input types:

* nextInt(): For integer input.
* nextDouble(): For double input.
* nextLine(): For string input.

**Que28.** **WAP to find out students' grades using Switch Case**

import java.util.Scanner;

public class Grade {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter marks: ");

int marks = sc.nextInt();

char grade;

switch (marks / 10) {

case 10:

case 9:

grade = 'A';

break;

case 8:

grade = 'B';

break;

case 7:

grade = 'C';

break;

case 6:

grade = 'D';

break;

default:

grade = 'F';

}

System.out.println("Grade: " + grade);

}

}

**Que29**. **WAP to check whether the inputted character is a Vowel or Consonant**

import java.util.Scanner;

public class VowelConsonant {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a character: ");

char ch = sc.next().charAt(0);

if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ||

ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

System.out.println(ch + " is a Vowel");

} else {

System.out.println(ch + " is a Consonant");

}

}

}