Java Assignment 01

Name:Samarth Devadiga

Roll no. 119

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

**Ans.**

Java is considered a platform-independent language because of its ability to run on different platforms without requiring modifications to the source code. This is primarily due to its "write once, run anywhere" (WORA) capability.

This means when you write a Java code and using javac compiler it produces “.class” file which contains the Bytecode.

This is achieved through the following:

### **1. Bytecode**

When Java source code is compiled, it is not converted into platform-specific machine code. Instead, it is compiled into an intermediate code called bytecode.

Bytecode is a platform-neutral, low-level representation of the program.

### **2. Java Virtual Machine (JVM)**

The bytecode is executed by the Java Virtual Machine (JVM), which is available for various operating systems (Windows, macOS, Linux, etc.).

Each platform has its own JVM implementation tailored to that specific environment. The JVM translates the bytecode into machine code that the underlying system can understand.

### **3. Consistency Across Platforms**

Since bytecode remains the same regardless of the operating system or hardware, the only requirement to run a Java program is a compatible JVM.

This decoupling of the program from the hardware and operating system ensures platform independence.

**Q2. Explain features of Java.**

**Ans.**

The Features of Java are as follows:

### **1. Platform Independent**

Java achieves platform independence through its "write once, run anywhere" (WORA) principle.

Code is compiled into bytecode, which can run on any system with a Java Virtual Machine (JVM).

### **2. Object-Oriented**

Java supports object-oriented programming (OOP) concepts like inheritance, polymorphism, abstraction, and encapsulation.

Almost Everything in Java is treated as an object.

### **3. Simple**

Java has a clean and easy-to-understand syntax, eliminating complex features like explicit pointers and operator overloading.

It has extensive libraries and predefined methods that simplify development.

### **4. Secure**

Java provides robust security features, such as:

No explicit use of pointers to avoid unauthorized memory access.

Bytecode verification and runtime security checks.

A built-in security manager for defining access permissions.

**5. Robust**

Java emphasizes reliability with:

Automatic memory management using garbage collection.

Exception handling to manage runtime errors.

Strong type-checking during compilation.

### **6. Multithreaded**

Java supports multithreading, allowing simultaneous execution of multiple threads for efficient utilization of CPU resources.

It Provides built-in classes like Thread and methods for thread synchronization.

### **7. Portable**

Java's bytecode format ensures portability across platforms without requiring changes in the source code.

Java libraries and APIs are designed to be consistent across platforms.

### **8. High Performance**

While Java is interpreted, modern JVMs use Just-In-Time (JIT) compilation to convert bytecode into native machine code at runtime, improving performance.

### **9. Dynamic**

Java is highly dynamic because it supports the loading of classes at runtime.

It also supports reflection, enabling inspection and modification of code during execution.

### **10. Distributed**

Java facilitates the development of distributed applications through APIs like RMI (Remote Method Invocation) and CORBA.

It supports features like sockets for easy communication over networks.

### **11. Extensive Libraries**

Java provides a vast standard library (Java API) for tasks like I/O operations, networking, data structures, and more, speeding up development.

### **12. Interpreted and Compiled**

Java programs are first compiled into bytecode and then interpreted by the JVM, combining the benefits of both compilation and interpretation.

**Q3. Explain the following terms in Java along with example:**

**i. Keyword**

**ii. Identifier**

**iii. Literals**

**Ans.**

**i. Keyword**

Keywords are reserved words in Java that have a predefined meaning and purpose in the language. They cannot be used as identifiers (e.g., variable names, method names).

Characteristics:

Keywords are always written in lowercase.

Examples include int, class, if, for, void, etc.

Example:

public class Main // 'public' and 'class' are keywords

{

public static void main(String[] args)

{

int number = 10; // 'int' is a keyword

if (number > 0) // 'if' is a keyword

{

System.out.println("Positive number");

}

}

}

### **ii. Identifier**

Identifiers are the names given to variables, methods, classes, or objects in Java.

Rules for Naming Identifiers:

* 1. Must start with a letter (A-Z or a-z), a dollar sign ($), or an underscore (\_).
  2. Can be followed by letters, digits (0-9), or underscores.
  3. Cannot be a Java keyword.
  4. Case-sensitive.

Example:

class identifierexample

{

public static void main(String[] args) // 'Student' is a class identifier

{

int age = 20;

displayAge(age); // 'age' is a variable identifier

}

public static void displayAge(int age) // 'displayAge' is a method identifier

{

System.out.println(age);

}

}

### **iii. Literals**

Literals represent fixed values directly assigned to variables in the program. They define the type of data (integer, character, string, etc.).

**Types of Literals in Java**:

* 1. **Integer Literals**: Numbers like 10, 100.
  2. **Floating-point Literals**: Decimal numbers like 3.14, 2.0.
  3. **Character Literals**: Single characters enclosed in single quotes, e.g., 'A'.
  4. **String Literals**: Text enclosed in double quotes, e.g., "Hello".
  5. **Boolean Literals**: true or false.
  6. **Null Literal**: Represents the absence of a value, null.

Example:

class literalexample

{

public static void main(String[ ]args)

{

int number = 100; // Integer literal

double pi = 3.14; // Floating-point literal

char grade = 'A'; // Character literal

String name = "Alice"; // String literal

boolean isJavaFun = true; // Boolean literal

System.out.println("number = "+number);

System.out.println("pi = "+pi);

System.out.println("grade = "+grade);

System.out.println("name = "+name);

System.out.println("isJavaFun = "+isJavaFun);

}

}

**Q4. Explain operator precedence and associativity.**

**Ans.**

Operator precedence and associativity determine the order in which operators are evaluated in a Java expression when multiple operators are present.

### **1. Operator Precedence**

Operator precedence is defined as the priority of operators in an expression. Operators with higher precedence are evaluated before operators with lower precedence.

Eg:

Int 5+2\*6;

In this 2\*6 is performed first as precedence of “ \* ” is greater than “ + “so 12+5=17

17 Will be the result.

### **2. Associativity**

When operators of the same precedence appear in an expression, associativity determines the order in which they are evaluated.

**Types of Associativity**:

* 1. Left-to-Right Associativity: Most operators (e.g., +, -, \*, /, %) are evaluated from left to right.

Eg.

int result = 100 / 10 \* 5; // Left-to-right associativity

In this 100/10=10 is done first and then 10\*5=50

2. Right-to-Left Associativity: Assignment operators (=, +=, -=) and some unary

operators are evaluated from right to left.

Eg.

int a, b, c;

a = b = c = 10; // Right-to-left associativity

System.out.println(a); // Output: 10

In this 10 is assigned to c first then b then a (right to left).

**Q5. Illustrate the precedence of different operators in Java with the help of a chart.**

**Ans.**

| **Precedence** | **Operator** | **Description** | **Associativity** |
| --- | --- | --- | --- |
| **1 (Highest)** | **()** | **Parentheses (for grouping)** | **Left-to-right** |
|  | **[]** | **Array access** | **Left-to-right** |
|  | **.** | **Member access (object.field, object.method())** | **Left-to-right** |
| **2** | **++ -- (postfix)** | **Post-increment, Post-decrement** | **Left-to-right** |
| **3** | **++ -- (prefix), + -** | **Pre-increment, Pre-decrement, Unary plus/minus** | **Right-to-left** |
|  | **~** | **Bitwise complement** | **Right-to-left** |
|  | **!** | **Logical NOT** | **Right-to-left** |
| **4** | **\* / %** | **Multiplication, Division, Modulus** | **Left-to-right** |
| **5** | **+ -** | **Addition, Subtraction** | **Left-to-right** |
| **6** | **<< >> >>>** | **Bitwise shift operators** | **Left-to-right** |
| **7** | **< <= > >=** | **Relational operators** | **Left-to-right** |
| **8** | **== !=** | **Equality operators** | **Left-to-right** |
| **9** | **&** | **Bitwise AND** | **Left-to-right** |
| **10** | **^** | **Bitwise XOR** | **Left-to-right** |
| **11** | **`** | **`** | **Bitwise OR** |
| **12** | **&&** | **Logical AND** | **Left-to-right** |
| **13** | **`** |  | **`** |
| **14** | **?:** | **Ternary (conditional) operator** | **Right-to-left** |
| **15** | **= += -= \*= /= %=** | **Assignment operators** | **Right-to-left** |

**Q6. Explain primitive data types in java along with their size, range and default values.**

**Ans.**

In Java, primitive data types are the basic building blocks of data manipulation. These types represent simple values and are predefined by the language. Java has 8 primitive data types, classified into four categories: integer, floating-point, character, and boolean.

Primitive types are faster than objects and consume less memory.

### **1. Integer Data Types**

Integer data types store whole numbers.

| **Data Type** | **Size (in bytes)** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| byte | 1 | -128 to 127 | 0 |
| short | 2 | -32,768 to 32,767 | 0 |
| int | 4 | -2,147,483,648 to 2,147,483,647 | 0 |
| long | 8 | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0L |

### **2. Floating-Point Data Types**

Floating-point data types store numbers with decimal points.

| **Data Type** | **Size (in bytes)** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| float | 4 | ±3.4e−038 to ±3.4e+038 (7 decimal digits) | 0.0f |
| double | 8 | ±1.7e−308 to ±1.7e+308 (15 decimal digits) | 0.0d |

### **3. Character Data Type**

The char type is used to store a single Unicode character.

| **Data Type** | **Size (in bytes)** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| char | 2 | 0 to 65,535 (Unicode values) | \u0000 (null) |

### **4. Boolean Data Type**

The boolean type is used to store a logical value: **true** or **false**.

| **Data Type** | **Size (in bytes)** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| boolean | 1 (or bit; JVM-dependent) | true or false | false |

**Q7. WAP to demonstrate how to create variables of different types.**

**Ans.**

class diffvariablesQ7

{

public static void main(String[] args)

{

byte smallNumber = 100;

short mediumNumber = 20000;

int largeNumber = 100000;

long veryLargeNumber = 10000000000L;

float pi = 3.1415f;

double bigDecimal = 12345.6789;

char letter = 'A';

boolean isJavaFun = true;

System.out.println("Integer Data Types:");

System.out.println("byte: " + smallNumber);

System.out.println("short: " + mediumNumber);

System.out.println("int: " + largeNumber);

System.out.println("long: " + veryLargeNumber);

System.out.println("\nFloating-point Data Types:");

System.out.println("float: " + pi);

System.out.println("double: " + bigDecimal);

System.out.println("\nCharacter Data Type:");

System.out.println("char: " + letter);

System.out.println("\nBoolean Data Type:");

System.out.println("boolean: " + isJavaFun);

}

}

**Output:**

**Integer Data Types:**

**byte: 100**

**short: 20000**

**int: 100000**

**long: 10000000000**

**Floating-point Data Types:**

**float: 3.1415**

**double: 12345.6789**

**Character Data Type:**

**char: A**

**Boolean Data Type:**

**boolean: true**

**Q8. Explain implicit type conversion, explicit type conversion and type promotion in Java.**

**Ans.**

In Java, type conversion refers to changing the data type of a value from one type to another. This can be done implicitly (automatically by the compiler) or explicitly (manually by the programmer).

Additionally, type promotion occurs when smaller data types are promoted to larger types during calculations.

### **1. Implicit Type Conversion (Type Casting)**

It is Also known as widening conversion.

The compiler automatically converts a smaller data type to a larger data type because it is safe and does not result in data loss.

It happens between compatible types.

#### **Rules:**

Smaller types (byte, short, char) are promoted to int in arithmetic operations.

Order of widening: byte → short → int → long → float → double.

### **2. Explicit Type Conversion (Type Casting)**

Also known as narrowing conversion.

The programmer manually converts a larger data type into a smaller data type.

Data loss or precision loss might occur.

Syntax: (targetType) value.

#### **Rules:**

Explicit casting is required when converting from a larger type to a smaller type, or between incompatible types.

**3. Type Promotion in Expressions**

In expressions, smaller data types (byte, short, char) are automatically promoted to int.

If one operand is of a larger type (e.g., float or double), the entire expression is promoted to the larger type.

**Q9. WAP to demonstrate explicit type conversion.**

**Ans.**

**class explicitypexpQ9**

**{**

**public static void main(String[] args)**

**{**

**double largeNumber = 123.456;**

**int convertedInt = (int) largeNumber;**

**float convertedFloat = (float) largeNumber;**

**int largeInt = 130;**

**byte convertedByte = (byte) largeInt;**

**System.out.println("Original double: " + largeNumber);**

**System.out.println("Converted to int: " + convertedInt);**

**System.out.println("Converted to float: " + convertedFloat);**

**System.out.println("Original int: " + largeInt);**

**System.out.println("Converted to byte: " + convertedByte);**

**}**

**}**

**Output:**

**Original double: 123.456**

**Converted to int: 123**

**Converted to float: 123.456**

**Original int: 130**

**Converted to byte: -126**

**Q10. WAP to demonstrate implicit type conversion.**

**Ans.**

**class implicittypconv**

**{**

**public static void main(String[] args)**

**{**

**byte smallNumber = 42;**

**int intNumber = smallNumber;**

**float decimalNumber = intNumber;**

**long largeInteger = intNumber;**

**double largeDecimal = decimalNumber;**

**System.out.println("Original byte value: " + smallNumber);**

**System.out.println("Implicitly converted to int: " + intNumber);**

**System.out.println("Implicitly converted to float: " + decimalNumber);**

**System.out.println("Implicitly converted to long: " + largeInteger);**

**System.out.println("Implicitly converted to double: " + largeDecimal);**

**}**

**}**

**Output:**

**Original byte value: 42**

**Implicitly converted to int: 42**

**Implicitly converted to float: 42.0**

**Implicitly converted to long: 42**

**Implicitly converted to double: 42.0**

**Q11. Explain arithmetic operators in java.**

**Ans.**

Arithmetic operators in Java are used to perform basic mathematical operations on variables and values. These operators can be applied to primitive numeric types (int, float, double, etc.).

| **Operator** | **Symbol** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- | --- |
| Addition | + | Adds two values. | a + b | Sum of a and b. |
| Subtraction | - | Subtracts one value from another. | a - b | Difference of a and b. |
| Multiplication | \* | Multiplies two values. | a \* b | Product of a and b. |
| Division | / | Divides one value by another. | a / b | Quotient of a divided by b. |
| Modulus | % | Returns the remainder of division. | a % b | Remainder when a is divided by b. |

### **Behavior with Data Types**

**Integer Division**: If both operands are integers, the result of division will also be an integer, with the fractional part truncated.

**Floating-point Division**: If either operand is a floating-point number, the result will include the decimal point.

**Q12. WAP to demonstrate arithmetic operators in java.**

**Ans.**

**class ArithmeticoperatorQ12**

**{**

**public static void main(String[] args) {**

**int a = 20, b = 6;**

**double x = 15.5, y = 4.2;**

**System.out.println("Integer Arithmetic Operations:");**

**System.out.println("a + b = " + (a + b));**

**System.out.println("a - b = " + (a - b));**

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

**System.out.println("a / b = " + (a / b));**

**System.out.println("a % b = " + (a % b));**

**System.out.println("\nFloating-point Arithmetic Operations:");**

**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));**

**}**

**}**

**Output:**

**Integer Arithmetic Operations:**

**a + b = 26**

**a - b = 14**

**a \* b = 120**

**a / b = 3**

**a % b = 2**

**Floating-point Arithmetic Operations:**

**x + y = 19.7**

**x - y = 11.3**

**x \* y = 65.10000000000001**

**x / y = 3.6904761904761902**

**x % y = 2.8999999999999995**

**Q13. WAP to demonstrate increment (pre and post) and decrement**

**(pre and post) operators.**

**class IncrementdecrementQ13**

**{**

**public static void main(String[] args)**

**{**

**int a = 5, b = 10;**

**System.out.println("Initial values:");**

**System.out.println("a = " + a);**

**System.out.println("b = " + b);**

**System.out.println("\nPost-increment:");**

**System.out.println("a++ = " + (a++));**

**System.out.println("Value of a after post-increment: " + a);**

**System.out.println("\nPre-increment:");**

**System.out.println("++a = " + (++a));**

**System.out.println("Value of a after pre-increment: " + a);**

**System.out.println("\nPost-decrement:");**

**System.out.println("b-- = " + (b--));**

**System.out.println("Value of b after post-decrement: " + b);**

**System.out.println("\nPre-decrement:");**

**System.out.println("--b = " + (--b));**

**System.out.println("Value of b after pre-decrement: " + b);**

**}**

**}**

**Output:**

**Initial values:**

**a = 5**

**b = 10**

**Post-increment:**

**a++ = 5**

**Value of a after post-increment: 6**

**Pre-increment:**

**++a = 7**

**Value of a after pre-increment: 7**

**Post-decrement:**

**b-- = 10**

**Value of b after post-decrement: 9**

**Pre-decrement:**

**--b = 8**

**Value of b after pre-decrement: 8**

**Q14. WAP to demonstrate compound assignment operator.**

**Ans.**

**class CompoundassignmentoprQ14**

**{**

**public static void main(String[] args)**

**{**

**int a = 10, b = 5;**

**System.out.println("Initial values:");**

**System.out.println("a = " + a);**

**System.out.println("b = " + b);**

**System.out.println("\nUsing compound assignment operators:");**

**a += b;**

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

**a -= b;**

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

**a \*= b;**

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

**a /= b;**

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

**a %= b;**

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

**}**

**}**

**Output:**

**Initial values:**

**a = 10**

**b = 5**

**Using compound assignment operators:**

**a += b: 15**

**a -= b: 10**

**a \*= b: 50**

**a /= b: 10**

**a %= b: 0**

**Q15. Explain relational operators in java.**

**Ans.**

Relational operators in Java are used to compare two values or expressions. These operators return a boolean result (true or false) based on the specified relation.

| **Operator** | **Symbol** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- | --- |
| **Equal to** | **==** | **Checks if two values are equal.** | **a == b** | **true if a equals b.** |
| **Not equal to** | **!=** | **Checks if two values are not equal.** | **a != b** | **true if a does not equal b.** |
| **Greater than** | **>** | **Checks if the left value is greater.** | **a > b** | **true if a is greater than b.** |
| **Less than** | **<** | **Checks if the left value is smaller.** | **a < b** | **true if a is less than b.** |
| **Greater than or equal to** | **>=** | **Checks if the left value is greater than or equal to the right value.** | **a >= b** | **true if a is greater than or equal to b.** |
| **Less than or equal to** | **<=** | **Checks if the left value is less than or equal to the right value.** | **a <= b** | **true if a is less than or equal to b.** |

### **Key Features**

1. Relational operators return a boolean value (true or false).
2. They work with primitive data types like int, float, char, double, etc.
3. They are commonly used in conditional statements like if, while, and for.

**Q16. WAP to demonstrate relational operators in java.**

**Ans.**

// 16. WAP to demonstrate relational operators in java.

class ReleationoperatorQ16

{

public static void main(String[] args)

{

int a = 15, b = 20;

System.out.println("Values:");

System.out.println("a = " + a);

System.out.println("b = " + b);

System.out.println("\nRelational Operators:");

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

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

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

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

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

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

double x = 15.0;

System.out.println("\nComparison with double:");

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

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

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

}

}

**Output:**

**Values:**

**a = 15**

**b = 20**

**Relational Operators:**

**a == b: false**

**a != b: true**

**a > b: false**

**a < b: true**

**a >= b: false**

**a <= b: true**

**Comparison with double:**

**a == x: true**

**a > x: false**

**a <= x: true**

**Q17. Explain logical operators in Java.**

**Ans.**

Logical operators in Java are used to perform logical operations on boolean expressions.

They are mainly used to combine multiple conditions and return a boolean result (true or false).

| **Operator** | **Symbol** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- | --- |
| AND | && | Returns true if **both** operands are true. | (a > 10) && (b < 20) | true if both conditions are true. |
| OR | || |  | || | Returns true if **at least one** operand is true. |
| NOT | ! | Returns true if the operand is false and vice versa. | !(a > 10) | true if the condition is false. |

1. **AND (&&)**:

Combines two conditions and evaluates to true only if **both** conditions are true.

Example: (a > 10) && (b < 20) is true if both a > 10 and b < 20.

1. **OR (||)**:

Combines two conditions and evaluates to true if **at least one** condition is true.

Example: (a > 10) || (b > 20) is true if either a > 10 or b > 20.

1. **NOT (!)**:

Reverses the boolean value of an expression.

Example: !(a > 10) is true if a > 10 is false.

1. **Short-circuit Evaluation**:

For &&, if the first condition is false, the second condition is not evaluated.

For ||, if the first condition is true, the second condition is not evaluated.

### **Key Notes**

Logical operators are essential for controlling the flow of a program, especially in conditional statements (if, while, for).

Logical operators work only with boolean values.

Use parentheses () to group conditions for better readability and precedence control.

**Q18. WAP to demonstrate logical operators in java.**

**Ans.**

**class LogicalopratorQ18**

**{**

**public static void main(String[] args)**

**{**

**int a = 15, b = 10;**

**System.out.println("AND (&&) Operator:");**

**System.out.println("(a > 10) && (b < 20): " + ((a > 10) && (b < 20)));**

**System.out.println("(a > 20) && (b < 20): " + ((a > 20) && (b < 20)));**

**System.out.println("\nOR (||) Operator:");**

**System.out.println("(a > 10) || (b > 20): " + ((a > 10) || (b > 20)));**

**System.out.println("(a > 20) || (b > 20): " + ((a > 20) || (b > 20)));**

**System.out.println("\nNOT (!) Operator:");**

**System.out.println("!(a > 10): " + (!(a > 10)));**

**System.out.println("!(b > 20): " + (!(b > 20)));**

**boolean isStudent = true, hasJob = false;**

**System.out.println("\nCombining Logical Operators:");**

**System.out.println("isStudent && hasJob: " + (isStudent && hasJob));**

**System.out.println("isStudent || hasJob: " + (isStudent || hasJob));**

**System.out.println("!(isStudent && hasJob): " + (!(isStudent && hasJob)));**

**}**

**}**

**Output:**

**AND (&&) Operator:**

**(a > 10) && (b < 20): true**

**(a > 20) && (b < 20): false**

**OR (||) Operator:**

**(a > 10) || (b > 20): true**

**(a > 20) || (b > 20): false**

**NOT (!) Operator:**

**!(a > 10): false**

**!(b > 20): true**

**Combining Logical Operators:**

**isStudent && hasJob: false**

**isStudent || hasJob: true**

**!(isStudent && hasJob): true**

**Q19. Explain short circuit operators in Java.**

**Ans.**

Short-circuit operators are a subset of logical operators in Java.

They are used to evaluate boolean expressions, but they stop evaluating as soon as the result is determined, thus improving performance.

Java provides two short-circuit logical operators:

1. Logical AND (&&)
2. Logical OR (||)
3. **Logical AND (&&)**:
   1. The expression A && B will evaluate to true only if both A and B are true.
   2. Short-circuit behavior: If A is false, the expression A && B will always be false regardless of the value of B. So, B is not evaluated if A is false.
4. **Logical OR (||)**:
   1. The expression A || B will evaluate to true if at least one of A or B is true.
   2. Short-circuit behavior: If A is true, the expression A || B will always be true regardless of the value of B. Therefore, B is not evaluated if A is true.

**Q20. WAP to demonstrate short circuit operators.**

**Ans.**

**class ShortcircuitoperatorQ20**

**{**

**public static void main(String[] args)**

**{**

**int a = 10, b = 20;**

**System.out.println("Short-circuit AND (&&) Operator:");**

**if (a > 20 && ++b > 20) {**

**System.out.println("Both conditions are true.");**

**} else {**

**System.out.println("Short-circuit AND: Second condition not evaluated, b = " + b);**

**}**

**System.out.println("\nShort-circuit OR (||) Operator:");**

**if (a < 20 || ++b > 20) {**

**System.out.println("At least one condition is true.");**

**} else {**

**System.out.println("This will not be executed.");**

**}**

**System.out.println("Short-circuit OR: Second condition not evaluated, b = " + b);**

**if (a < 20 || b > 20) {**

**System.out.println("Both conditions can be true in OR (||).");**

**}**

**}**

**}**

**Output:**

**Short-circuit AND (&&) Operator:**

**Short-circuit AND: Second condition not evaluated, b = 20**

**Short-circuit OR (||) Operator:**

**At least one condition is true.**

**Short-circuit OR: Second condition not evaluated, b = 20**

**Both conditions can be true in OR (||).**

**Q21. Explain conditional operator along with syntax and example.**

The conditional operator (also known as the ternary operator) is a shorthand for an if-else statement. It is used to assign a value based on a condition.

It is called "ternary" because it involves three operands.

**Syntax:**

**condition ? expression1 : expression2;**

**condition**: This is the condition that is evaluated. If it is true, expression1 is executed. If it is false, expression2 is executed.

**expression1**: The value returned if the condition is true.

**expression2**: The value returned if the condition is false.

If the condition is true, the value of expression1 is assigned or returned.

If the condition is false, the value of expression2 is assigned or returned.

**Q22.WAP to find greater of two numbers using conditional operator.**

**Ans.**

**public class GreateroftwoQ22 {**

**public static void main(String[] args) {**

**int a = 10, b = 20;**

**int max = (a > b) ? a : b;**

**System.out.println("The larger number is: " + max);**

**String result = (a == b) ? "Equal" : "Not Equal";**

**System.out.println("Comparison result: " + result);**

**}**

**}**

**Output:**

**The larger number is: 20**

**Comparison result: Not Equal**

**Q23. Explain conditional statements (if, if else, nested if else, else if ladder, switch case)**

**Ans.**

Conditional statements are used to execute different code blocks based on certain conditions.

They are fundamental in controlling the flow of a program.

Below are the different types of conditional statements in Java:

### **1. if Statement**

The simplest conditional statement. It executes a block of code if the specified condition is true.

**Syntax:**

**if (condition)**

**{**

**// Code to execute if the condition is true**

**}**

### **2. if-else Statement**

An if-else statement is used when you need to execute one block of code if the condition is true and another block of code if the condition is false.

**Syntax:**

**if (condition)**

**{**

**// Code to execute if the condition is true**

**}**

**else**

**{**

**// Code to execute if the condition is false**

**}**

### **3. else if Ladder**

An else if ladder allows you to test multiple conditions.

It is useful when you need to check multiple conditions and execute different code based on the condition that is true.

**Syntax:**

**if (condition1)**

**{**

**// Code to execute if condition1 is true**

**}**

**else if (condition2)**

**{**

**// Code to execute if condition2 is true**

**}**

**else if (condition3)**

**{**

**// Code to execute if condition3 is true**

**}**

**else**

**{**

**// Code to execute if none of the conditions are true**

**}**

### **4. Nested if-else**

A **nested if-else** is an if-else statement inside another if or else block. It allows you to check multiple conditions in a hierarchical manner.

**Syntax:**

if (condition1) {

if (condition2) {

// Code to execute if condition1 and condition2 are true

} else {

// Code to execute if condition1 is true but condition2 is false

}

} else {

// Code to execute if condition1 is false

}

### 5. switch Statement

A switch statement is an alternative to the if-else ladder when you have multiple values of the same variable and want to execute different code based on the value of that variable.

**Syntax:**

**switch (variable) {**

**case value1:**

**// Code to execute if variable == value1**

**break;**

**case value2:**

**// Code to execute if variable == value2**

**break;**

**default:**

**// Code to execute if variable does not match any case**

**}**

### **Key Differences**

1. **if**: Used for simple condition checking, executes a block of code if the condition is true.
2. **if-else**: Executes one block of code if the condition is true, and another block if it is false.
3. **else if ladder**: A series of conditions that are checked sequentially; executes different code based on which condition is true.
4. **Nested if-else**: if-else statements inside other if-else blocks to handle complex conditions.
5. **switch**: Used to select one of many code blocks to be executed based on the value of a variable.

**Q24. WAP to find greater of two numbers using if else statement.**

**Ans.**

**import java.util.Scanner;**

**public class GreaterNumberUsingIfElseQ24 {**

**public static void main(String[] args) {**

**// Create a Scanner object to read user input**

**Scanner scanner = new Scanner(System.in);**

**// Input two numbers from the user**

**System.out.print("Enter the first number: ");**

**int num1 = scanner.nextInt();**

**System.out.print("Enter the second number: ");**

**int num2 = scanner.nextInt();**

**// Use if-else statement to find the greater number**

**if (num1 > num2) {**

**System.out.println("The greater number is: " + num1);**

**} else {**

**System.out.println("The greater number is: " + num2);**

**}**

**// Close the scanner**

**scanner.close();**

**}**

**}**

**Output:**

**Enter the first number: 44**

**Enter the second number: 55**

**The greater number is: 55**

**Q25. WAP to find whether the inputted number is even or odd.**

**Ans.**

**import java.util.Scanner;**

**public class EvenOrOddQ25 {**

**public static void main(String[] args) {**

**// Create a Scanner object to read user input**

**Scanner scanner = new Scanner(System.in);**

**// Input a number from the user**

**System.out.print("Enter a number: ");**

**int num = scanner.nextInt();**

**// Check if the number is even or odd**

**if (num % 2 == 0) {**

**System.out.println(num + " is an even number.");**

**} else {**

**System.out.println(num + " is an odd number.");**

**}**

**// Close the scanner**

**scanner.close();**

**}**

**}**

**Output:**

**Enter a number: 55**

**55 is an odd number.**

**Q26. WAP to find greatest among three numbers using if else.**

**Ans.**

**import java.util.Scanner;**

**public class GreatestOfThreeNumbersQ26**

**{**

**public static void main(String[] args)**

**{**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the first number: ");**

**int num1 = scanner.nextInt();**

**System.out.print("Enter the second number: ");**

**int num2 = scanner.nextInt();**

**System.out.print("Enter the third number: ");**

**int num3 = scanner.nextInt();**

**if (num1 >= num2 && num1 >= num3) {**

**System.out.println("The greatest number is: " + num1);**

**} else if (num2 >= num1 && num2 >= num3) {**

**System.out.println("The greatest number is: " + num2);**

**} else {**

**System.out.println("The greatest number is: " + num3);**

**}**

**scanner.close();**

**}**

**Output:**

**Enter the first number: 66**

**Enter the second number: 77**

**Enter the third number: 88**

**The greatest number is: 88**

**Q27. Explain how to use Scanner class for user input. Discuss**

**different methods for taking user input of Scanner class. WAP to**

**demonstrate it.**

**Ans.**

The **Scanner class** is a part of the java.util package and is used to obtain input from the user. It can read various types of input, such as strings, integers, floating-point numbers, and more.

Steps to Use the Scanner Class:

Step1: import java.util.Scanner;

Step2:create an object of scanner class:

Scanner scanner = new Scanner(System.in);

Step3: Use the appropriate method to read the type of input you need.

The Scanner class provides different methods for different data types.

### **Common Methods of Scanner Class:**

1. **nextInt()**: Reads an integer value.  
    int num = scanner.nextInt();

**2.nextDouble()**: Reads a double value.  
  
 double num = scanner.nextDouble();

**3.nextLine()**: Reads a whole line of text (including

spaces).  
  
 String str = scanner.nextLine();

**4.next()**: Reads a single word (i.e., up to the first space).  
 String str = scanner.next();

**5.nextBoolean()**: Reads a boolean value (true or false).  
  
 boolean flag = scanner.nextBoolean();

**6.nextFloat()**: Reads a floating-point value.  
  
 float num = scanner.nextFloat();

**Program:**

**import java.util.Scanner;**

**public class ScannerExampleQ27 {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter an integer: ");**

**int num1 = scanner.nextInt();**

**System.out.print("Enter a floating-point number: ");**

**float num2 = scanner.nextFloat();**

**scanner.nextLine();**

**System.out.print("Enter a string: ");**

**String str = scanner.nextLine();**

**System.out.print("Enter a boolean value (true/false): ");**

**boolean flag = scanner.nextBoolean();**

**System.out.println("\nYou entered:");**

**System.out.println("Integer: " + num1);**

**System.out.println("Floating-point number: " + num2);**

**System.out.println("String: " + str);**

**System.out.println("Boolean: " + flag);**

**scanner.close();**

**}**

**}**

**Output:**

**Enter an integer: 5**

**Enter a floating-point number: 3.14**

**Enter a string: Hello World**

**Enter a boolean value (true/false): true**

**Integer: 5**

**Floating-point number: 3.14**

**String: Hello World**

**Boolean: true**

**28.Write a Java program to find out student grades Using switch case**

**Ans.**

**import java.util.Scanner;**

**public class StudentGrade {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the marks of the student: ");**

**int marks = scanner.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;**

**case 5:**

**grade = 'E';**

**break;**

**default:**

**grade = 'F';**

**}**

**System.out.println("The grade of the student is: " + grade);**

**scanner.close();**

**}**

**}**

**Output:**

**Enter the marks of the student: 77**

**The grade of the student is: C**

**29.WAP to check weather inputted character is vowel or Constant.**

**Ans.**

**import java.util.Scanner;**

**public class VowelOrConsonant {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter a character: ");**

**char ch = scanner.next().toLowerCase().charAt(0);**

**if (ch >= 'a' && ch <= 'z') {**

**if (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.");**

**}**

**} else {**

**System.out.println("Please enter a valid alphabetic character.");**

**}**

**scanner.close();**

**}**

**}**

**Output:**

**Enter a character: F**

**f is a consonant.**