Java Assignment – 1

Saurabh Yadav – 150096724010

Q1) Why Java is considered as platform independent language.

Java is considered a platform-independent language because it uses a unique approach to run programs on any operating system or hardware architecture without modification. This is achieved through the use of the **Java Virtual Machine (JVM)** and the **bytecode** system.

1. Compilation to Bytecode:
   1. Java source code is written in .java files and is compiled into an intermediate form called bytecode, which is stored in .class files.
   2. This bytecode is not specific to any machine; instead, it is designed to be executed on any machine with a JVM.
2. Role of the Java Virtual Machine (JVM):
   1. The JVM is a platform-specific software component that interprets or compiles the bytecode into native machine code for the host system.
   2. Each operating system (Windows, macOS, Linux, etc.) and hardware architecture has its own version of the JVM.
3. Write Once, Run Anywhere (WORA):
   1. Since the bytecode is universal and JVM implementations are available for most platforms, Java programs can run on any system where a compatible JVM is installed.
   2. Developers do not need to rewrite or recompile their code for different platforms, fulfilling the "Write Once, Run Anywhere" promise.

Q2) Explain features of Java:

1. Secure: Java provides a secure execution environment by using a runtime that enforces access control and memory management. It eliminates vulnerabilities like buffer overflows and uses features like bytecode verification and the Security Manager.
2. Portable: Java's platform independence ensures programs run on any system with a JVM. Its use of standardized bytecode and lack of reliance on hardware-specific features make it highly portable.
3. Object-oriented: Java follows an object-oriented programming paradigm, focusing on objects and classes, making code reusable, modular, and easier to maintain.
4. Robust: Java emphasizes reliability with features like automatic garbage collection, exception handling, and strong type-checking, minimizing runtime errors.
5. Multithreaded: Java supports concurrent programming with built-in support for threads, allowing developers to perform multiple tasks simultaneously for better performance.
6. Architecture-neutral: Java's bytecode is designed to run on any machine, regardless of hardware or operating system, making it architecture-agnostic.
7. Interpreted: Java bytecode is interpreted by the JVM, enabling platform independence and dynamic execution without the need for platform-specific compilation.
8. High performance: Java achieves high performance through Just-In-Time (JIT) compilation, which translates bytecode into native machine code at runtime.
9. Distributed: Java supports networked and distributed computing with APIs like RMI (Remote Method Invocation) and support for internet protocols like HTTP and FTP.
10. Dynamic: Java supports dynamic linking of classes and can load classes at runtime, making applications adaptable to evolving environments and requirements.

Q3) Explain the following terms in Java along with example:

#### **i. Keyword**

* Definition: Keywords are reserved words in Java that have predefined meanings and cannot be used for variable names, method names, or identifiers. They form the building blocks of Java syntax.
* Examples of Keywords: class, if, else, while, return, static, etc.

#### **ii. Identifier**

* Definition: Identifiers are names assigned by the programmer to variables, methods, classes, or objects. They must begin with a letter, $, or \_ and cannot use Java keywords. Identifiers are case-sensitive.
* Examples of Identifiers: myVariable, calculateSum, Example (class name).

#### **iii. Literals**

* Definition: Literals represent constant values that are directly assigned to variables. Java supports different types of literals like numeric, character, string, boolean, and null literals.
* Examples of Literals: 10 (integer literal), 3.14 (floating-point literal), "Hello" (string literal), true (boolean literal).

### Q4) Explain Operator Precedence and Associativity in Java:

#### Operator Precedence:

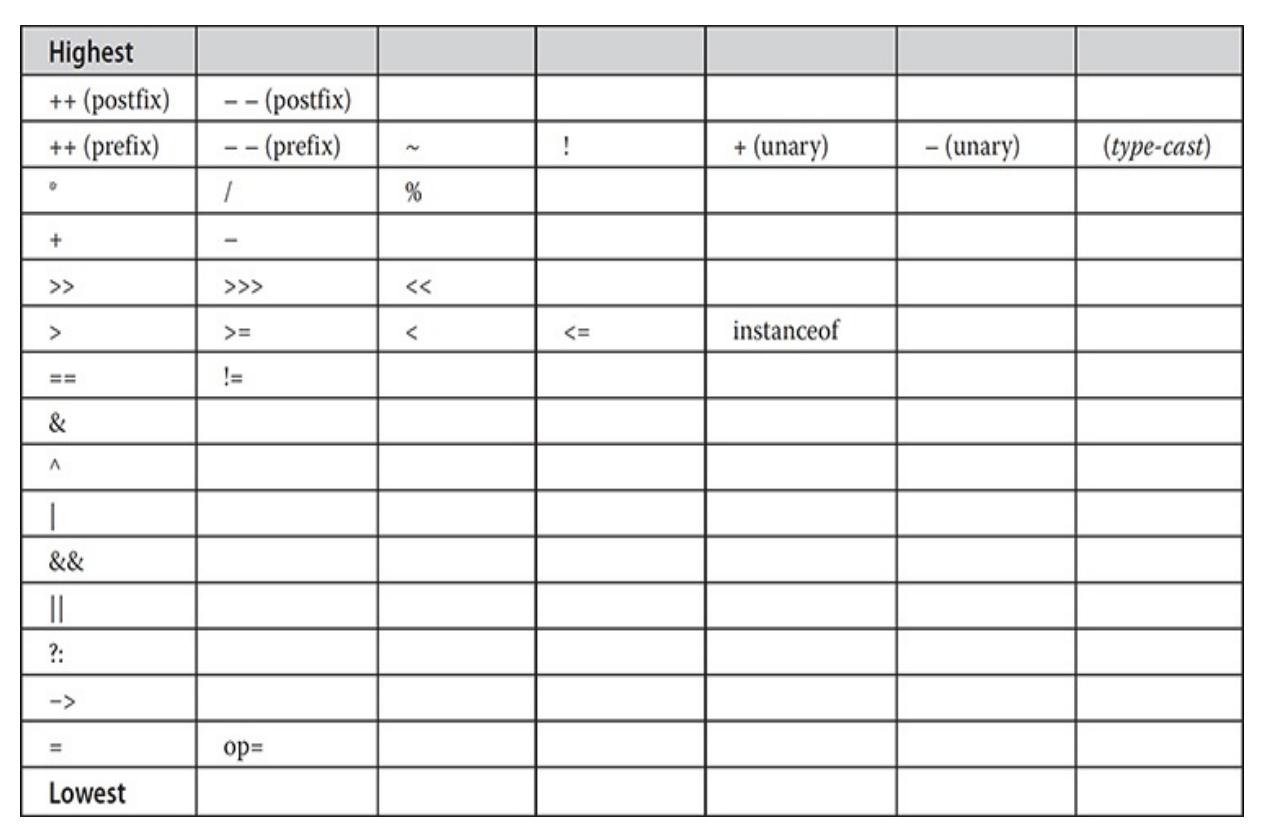
* Definition: Operator precedence determines the order in which operators are evaluated in an expression when multiple operators are present. Operators with higher precedence are evaluated before those with lower precedence.
* Example: int result = 10 + 5 \* 2; // Multiplication (\*) has higher precedence than addition (+)  
  System.out.println(result); // Output: 20 (5 \* 2 is evaluated first, then 10 + 10)

#### Associativity:

* Definition: Associativity determines the order of evaluation for operators with the same precedence when they appear together in an expression. It can be:
  + Left-to-right (most operators): The expression is evaluated from left to right.
  + Right-to-left (assignment and unary operators): The expression is evaluated from right to left.
* Example: // Left-to-right associativity  
  int result = 10 - 5 - 2; //Subtraction (-) is evaluated from left to right  
  System.out.println(result); // Output: 3 ((10 - 5) - 2)  
    
  // Right-to-left associativity  
  int a, b, c;  
  a = b = c = 5; // Assignment (=) is evaluated from right to left  
  System.out.println(a); // Output: 5

Q5) Illustrate the precedence of different operators in Java with the

help of a chart.



Q6) Explain primitive data types in java along with their size, range

and default values.

Java provides 8 **primitive data types** to handle simple data. These types are categorized into four groups: integer, floating-point, character, and boolean.

**Integer types** include byte, short, int, and long. The byte type is 8 bits in size, with a range of -128 to 127, and has a default value of 0. It is useful for memory-efficient storage in arrays. The short type is 16 bits, ranging from -32,768 to 32,767, with a default value of 0, and is also used for saving memory when slightly larger numbers are required. The int type, the most commonly used integer type, is 32 bits with a range of -2,147,483,648 to 2,147,483,647 and a default value of 0. For very large numbers, the long type is used. It is 64 bits in size with a range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 and a default value of 0L.

**Floating-point types** include float and double. The float type is 32 bits and has a range of approximately ±3.40282347E+38F with up to 7 decimal digits of precision, and its default value is 0.0f. It is used for single-precision floating-point computations. The double type, used for double-precision calculations, is 64 bits with a range of approximately ±1.79769313486231570E+308 and up to 15 decimal digits of precision, and it has a default value of 0.0d.

The **character type**, char, is 16 bits in size and can represent a single character or Unicode value, ranging from '\u0000' to '\uffff'. Its default value is '\u0000'. The **boolean type** represents a logical value, either true or false, with a default value of false. It is typically used in control statements and conditional logic.

These data types ensure efficient memory usage and precise control over variables in Java programming.

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

public class Variables {

public static void main(String[] args) {

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

int a = 10;

float b = 10.5f;

char c = 'a';

double d = 10.5;

long e = 10L;

byte f = 10;

boolean g = true;

String message = "Hello, Java!";

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

System.out.println("Float: " + b);

System.out.println("Character: " + c);

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

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

System.out.println("Byte: " + f);

System.out.println("Boolean: " + g);

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

}

}

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

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

* Definition: When the compiler automatically converts a smaller data type into a larger data type during operations, this is known as implicit type conversion or widening conversion.
* Characteristics:
  + No data loss occurs.
  + Happens automatically without requiring explicit instructions from the programmer.
  + Compatible types must be involved.
  + Example of widening conversions: byte → short → int → long → float → double.
* Example:

int num = 100;  
double result = num;

System.out.println("Result: " + result); // Output: 100.0

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

* Definition: When a larger data type is explicitly converted into a smaller data type or a different type by the programmer, it is called explicit type conversion or narrowing conversion.
* Characteristics:
  + Requires casting using the syntax (targetType) value.
  + May result in data loss or truncation.
  + Can lead to unexpected results if the value exceeds the target type's range.
  + Example of narrowing conversions: double → float → long → int → short → byte.
* Example:

double num = 100.99;  
int result = (int) num;

System.out.println("Result: " + result); // Output: 100 (decimal part is truncated)

#### 3. Type Promotion

* Definition: When performing operations, smaller data types are promoted to a larger type to avoid data loss and maintain consistency.
* Characteristics:
  + Operands of smaller data types (byte, short, char) are automatically promoted to int during arithmetic operations.
  + If one operand is a larger type, the result is promoted to the larger type.
  + Type promotion ensures compatibility in mixed-type expressions.
* Example:

byte a = 10;  
byte b = 20;  
int result = a + b; // byte values are promoted to int during addition  
System.out.println("Result: " + result); // Output: 30  
  
double x = 5.5;  
int y = 2;  
double divisionResult = x / y; // int is promoted to double during division  
System.out.println("Result: " + divisionResult); // Output: 2.75

Q9) WAP to demonstrate explicit type conversion.

public class ExplicitConversion {

public static void main(String[] args) {

double d = 100.3;

long l = (long)d;

int i = (int)l;

System.out.println("Long Value " +l);

System.out.println("Int Value "+i);

}

}

Q10) WAP to demonstrate implicit type conversion.

public class ImplicitConversion {

public static void main(String[] args) {

*//demonstrate implicit conversion*

int a = 100;

long b = a;

float c = b;

System.out.println(a);

System.out.println(b);

System.out.println(c);

}

}

Q11) Explain arithmetic operators in java.

### Arithmetic Operators in Java

Arithmetic operators in Java are used to perform basic mathematical operations such as addition, subtraction, multiplication, division, and modulus. These operators work on numeric data types like int, float, double, long, etc.

#### 1. Addition (+)

* Description: Adds two values.
* Example: int a = 10, b = 20;  
  int sum = a + b; // sum = 30  
  System.out.println("Addition: " + sum);

#### 2. Subtraction (-)

* Description: Subtracts the second value from the first.
* Example: int a = 20, b = 10;  
  int difference = a - b; // difference = 10  
  System.out.println("Subtraction: " + difference);

#### 3. Multiplication (\*)

* Description: Multiplies two values.
* Example: int a = 5, b = 4;  
  int product = a \* b; // product = 20  
  System.out.println("Multiplication: " + product);

#### 4. Division (/)

* Description: Divides the first value by the second. For integer division, the fractional part is truncated.
* Example: int a = 20, b = 4;  
  int quotient = a / b; // quotient = 5  
  System.out.println("Division: " + quotient);  
    
  double x = 10, y = 4;  
  double division = x / y; // division = 2.5  
  System.out.println("Division with floating-point: " + division);

#### 5. Modulus (%)

* Description: Returns the remainder of the division of two values.
* Example: int a = 20, b = 6;  
  int remainder = a % b; // remainder = 2  
  System.out.println("Modulus: " + remainder);

Q12) WAP to demonstrate arithmetic operators in java.

public class BasicMath {

public static void main(String[] args) {

int a = 10;

int b = 20;

int c = a + b;

int d = a - b;

int e = a \* b;

int f = a / b;

int g = a % b;

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

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

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

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

System.err.println("a % b = " + g);

}

}

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

(pre and post) operators.

public class PrePostDec {

public static void main(String[] args) {

*//write a program to demonstrate increament and decrement operators*

int a = 10;

int b = 10;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}

}

Q14) WAP to demonstrate compound assignment operator.

public class CompoundAssignmentOperator {

public static void main(String[] args) {

int a = 10;

int b = 2;

System.out.println("Arithmetic Compound Assignment:");

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

a += b; *// a = a + b*

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

a -= b; *// a = a - b*

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

a \*= b; *// a = a \* b*

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

a /= b; *// a = a / b*

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

a %= b; *// a = a % b*

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

int c = 5;

int d = 3;

System.out.println("\nBitwise Compound Assignment:");

System.out.println("c = " + c + ", d = " + d);

c &= d; *// c = c & d*

System.out.println("c &= d: " + c);

c |= d; *// c = c | d*

System.out.println("c |= d: " + c);

c ^= d; *// c = c ^ d*

System.out.println("c ^= d: " + c);

int e = 10;

System.out.println("\nShift Compound Assignment:");

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

e <<= 2; *// e = e << 2*

System.out.println("e <<= 2: " + e);

e >>= 1; *// e = e >> 1*

System.out.println("e >>= 1: " + e);

e >>>= 1; *// e = e >>> 1*

System.out.println("e >>>= 1: " + e);

}

}

### Q15) Explain relational operators in java.

### 1. Equal to (==)

* Description: Checks if two values are equal.
  + Example: int a = 5, b = 5;  
    System.out.println(a == b); // true

### 2. Not equal to (!=)

* Description: Checks if two values are not equal.
  + Example: int a = 5, b = 3;  
    System.out.println(a != b); // true

### 3. Greater than (>)

* Description: Checks if the left value is greater than the right value.
  + Example: int a = 7, b = 5;  
    System.out.println(a > b); // true

### 4. Less than (<)

* Description: Checks if the left value is less than the right value.
  + Example: int a = 5, b = 7;  
    System.out.println(a < b); // true

### 5. Greater than or equal to (>=)

* Description: Checks if the left value is greater than or equal to the right value.
  + Example: int a = 7, b = 7;  
    System.out.println(a >= b); // true

### 6. Less than or equal to (<=)

* Description: Checks if the left value is less than or equal to the right value.
  + Example: int a = 5, b = 7;  
    System.out.println(a <= b); // true

Q16) WAP to demonstrate relational operators in java.

public class RelationalOpp {

public static void main(String[] args) {

*//program to demonstrate relational operators*

int a = 10;

int b = 20;

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

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

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

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

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

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

}

}

Q17) Explain Logical Operators

### Logical Operators in Java

Logical operators in Java are used to combine multiple boolean expressions or values and return a boolean result. These operators help in controlling the flow of logic in decision-making structures like if statements, while loops, etc.

1. Logical AND (&&)

* Description: Returns true only if both operands are true.
* Example: boolean a = true, b = false;  
  System.out.println(a && b); // false (one is false)

### 2. Logical OR (||)

* Description: Returns true if at least one operand is true.
* Example: boolean a = true, b = false;  
  System.out.println(a || b); // true (one is true)

### 3. Logical NOT (!)

* Description: Reverses the boolean value (true becomes false and vice versa).
* Example: boolean a = true;  
  System.out.println(!a); // false (negation of true)

Q18) WAP to demonstrate logical operators in java.

public class LogicalOperator {

public static void main(String[] args) {

boolean a = true;

boolean b = false;

System.out.println("Logical AND:");

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

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

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

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

System.out.println("\nLogical OR:");

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

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

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

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

System.out.println("\nLogical NOT:");

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

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

}

}

Q19) Explain ShortCircuit Operators

### Short-circuit Operators in Java

Short-circuit operators are a type of logical operator in Java that evaluate expressions in a way that stops further evaluation as soon as the result is determined. They are commonly used in conditional statements to optimize performance.

### 1. Short-circuit AND (&&)

* Description: If the first operand is false, the overall expression will always be false, and the second operand will not be evaluated.
* Example:

boolean a = false, b = true;  
System.out.println(a && b); // false (b is not evaluated because a is false)

* Why It’s Efficient: In the case where the first condition is false, there's no need to check the second condition, as the result will be false regardless.

### 2. Short-circuit OR (||)

* Description: If the first operand is true, the overall expression will always be true, and the second operand will not be evaluated.
* Example:

boolean a = true, b = false;  
System.out.println(a || b); // true (b is not evaluated because a is true)

Q20) WAP to demonstrate short circuit operators.

public class ShortCircuitOpp {

public static void main(String[] args) {

*//WAP to demonstrate short circuit operators.*

boolean a = false, b = true;

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

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

}

}

Q21) Explain conditional operator along with syntax and example.

### Conditional (Ternary) Operator in Java

The conditional (ternary) operator is a simple and concise way to perform conditional checks and return values based on the result of the condition. It is a shorthand for the if-else statement and can be used to assign a value to a variable based on a condition.

### Syntax:

condition ? value\_if\_true : value\_if\_false;

* Condition: The condition that is checked. If it evaluates to true, the value after the ? is returned; otherwise, the value after the : is returned.

### Example:

int age = 18;  
String eligibility = (age >= 18) ? "Eligible to vote" : "Not eligible to vote";  
System.out.println(eligibility); // Output: Eligible to vote

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

public class ConditionalOpp {

public static void main(String[] args) {

*//program to demonstrate condtional operators ternary*

int a = 10;

int b = 20;

int c = (a < b) ? a : b;

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

c = (a > b) ? a : b;

System.out.println("largest = " + c);

}

}

Q23) Explain conditional statements (if, if else, nested if else, else

if ladder, switch case)

### Conditional Statements in Java

Conditional statements are used to perform different actions based on different conditions. Java provides several types of conditional statements: if, if-else, nested if-else, else-if ladder, and switch-case.

### 1. if Statement

* Description: The if statement evaluates a condition, and if it is true, the associated block of code is executed.
* Syntax: if (condition) {  
   // code to execute if the condition is true  
  }
* Example: int a = 5;  
  if (a > 3) {  
   System.out.println("a is greater than 3");  
  }

### 2. if-else Statement

* Description: The if-else statement evaluates a condition. If the condition is true, the first block of code runs; if it’s false, the second block runs.
* Syntax: if (condition) {  
   // code to execute if the condition is true  
  } else {  
   // code to execute if the condition is false  
  }
* Example: int a = 2;  
  if (a > 3) {  
   System.out.println("a is greater than 3");  
  } else {  
   System.out.println("a is not greater than 3");  
  }

### 3. Nested if-else Statement

* Description: The nested if-else is when an if or else statement contains another if-else statement inside it.
* Syntax: if (condition1) {  
   if (condition2) {  
   // code to execute if both conditions are true  
   } else {  
   // code to execute if condition1 is true and condition2 is false  
   }  
  } else {  
   // code to execute if condition1 is false  
  }
* Example: int a = 5, b = 10;  
  if (a > 3) {  
   if (b > 5) {  
   System.out.println("a is greater than 3 and b is greater than 5");  
   } else {  
   System.out.println("a is greater than 3 but b is not greater than 5");  
   }  
  }

### 4. else-if ladder Statement

* Description: The else-if ladder is used when you have multiple conditions to check, and only one condition should be executed. Each else-if is checked one by one, and the first true condition’s block is executed.
* 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 above conditions are true  
  }
* Example: int a = 10;  
  if (a > 15) {  
   System.out.println("a is greater than 15");  
  } else if (a == 10) {  
   System.out.println("a is equal to 10");  
  } else {  
   System.out.println("a is less than 10");  
  }

### 5. switch-case Statement

* **Description**: The switch statement evaluates an expression, matches it with multiple case values, and executes the corresponding block of code. It is often used when there are multiple conditions based on the same variable.
* **Syntax**: switch (expression) {  
   case value1:  
   break;  
   case value2:  
   break;  
   default:

//code  
   
}

* **Example**: int day = 3;  
  switch (day) {  
   case 1:  
   System.out.println("Monday");  
   break;  
   case 2:  
   System.out.println("Tuesday");  
   break;  
   case 3:  
   System.out.println("Wednesday");  
   break;  
   default:  
   System.out.println("Invalid day");  
  }

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

import java.util.Scanner;

public class TwoGreater {

public static void main(String[] args) {

int a;

int b;

Scanner in = new Scanner(System.in);

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

a = in.nextInt();

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

b = in.nextInt();

if (a > b) {

System.out.println("The greater number is " + a);

} else {

System.out.println("The greater number is " + b);

}

}

}

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

import java.util.Scanner;

public class EvenOrOdd {

public static void main(String[] args) {

int a;

Scanner in = new Scanner(System.in);

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

a = in.nextInt();

if (a % 2 == 0) {

System.out.println("Even");

} else {

System.out.println("Odd");

}

in.close();

}

}

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

import java.util.Scanner;

public class ThreeGreatestNumber {

public static void main(String[] args) {

int a;

int b;

int c;

Scanner in = new Scanner(System.in);

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

a = in.nextInt();

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

b = in.nextInt();

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

c = in.nextInt();

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

System.out.println("The greatest number is " + a);

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

System.out.println("The greatest number is " + b);

} else {

System.out.println("The greatest number is " + c);

}

in.close();

}

}

Q27) Explain how to use Scanner class for user input. Discuss different methods for taking user input of Scanner class. WAP to demonstrate it.

### Using the Scanner Class for User Input in Java

The Scanner class in Java is a part of the java.util package, which provides methods to take input from various sources like keyboard, file, or string. The most common use of the Scanner class is for reading input from the user via the console.

### Steps to Use the Scanner Class:

1. Import the Scanner class: You need to import the Scanner class from the java.util package.

import java.util.Scanner;

1. Create a Scanner object: Create an instance of the Scanner class, typically associated with System.in to read input from the console.

Scanner scanner = new Scanner(System.in);

1. Use different methods to take input: The Scanner class provides several methods to read different types of data from the user:
   1. nextInt(): Reads an integer.
   2. nextDouble(): Reads a double.
   3. nextLine(): Reads a line of text (string).
   4. next(): Reads a single word (string).
   5. nextBoolean(): Reads a boolean value (true or false).
2. Close the Scanner object: It's a good practice to close the Scanner object after use to prevent resource leakage.

scanner.close();

### Methods for Taking User Input in Scanner Class:

1. **nextInt()**: Reads an integer input from the user.

int number = scanner.nextInt();

1. **nextDouble()**: Reads a double input.

double number = scanner.nextDouble();

1. **nextLine()**: Reads a full line of text (including spaces).

String line = scanner.nextLine();

1. **next()**: Reads a single word (non-space characters).

String word = scanner.next();

1. **nextBoolean()**: Reads a boolean value (true or false).

boolean flag = scanner.nextBoolean();

### Program to Demonstrate User Input Using Scanner Cla**ss**

import java.util.Scanner;

public class ScannerExample {

public static void main(String[] args) {

*// Create a Scanner object*

Scanner scanner = new Scanner(System.in);

*// Taking integer input*

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

int intValue = scanner.nextInt();

System.out.println("You entered the integer: " + intValue);

*// Taking double input*

System.out.print("Enter a double value: ");

double doubleValue = scanner.nextDouble();

System.out.println("You entered the double: " + doubleValue);

*// Taking string input (one word)*

System.out.print("Enter a single word: ");

String word = scanner.next();

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

*// Consume the newline left by next()*

scanner.nextLine();

*// Taking full line input*

System.out.print("Enter a full sentence: ");

String line = scanner.nextLine();

System.out.println("You entered the sentence: " + line);

*// Taking boolean input*

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

boolean boolValue = scanner.nextBoolean();

System.out.println("You entered the boolean: " + boolValue);

scanner.close();

}

}

Q28) Write a Java Program Find out Students Grades using SwitchCase

import java.util.Scanner;

public class StudentGrades {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the student's score: ");

int score = scanner.nextInt();

char grade;

switch ((score > 90) ? 1 : (score >= 80) ? 2 : (score >= 70) ? 3 : (score >= 60) ? 4 : (score >= 50) ? 5 : 6) {

case 1:

grade = 'A';

break;

case 2:

grade = 'B';

break;

case 3:

grade = 'C';

break;

case 4:

grade = 'D';

break;

case 5:

grade = 'E';

break;

default:

grade = 'F';

}

System.out.println("The student's grade is: " + grade);

}

}

Q29) WAP to check whether the inputted character is Vowel or Consonant

import java.util.Scanner;

public class VowelOrConsonent {

public static void main(String[] args) {

char ch;

Scanner in = new Scanner(System.in);

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

ch = in.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("Vowel");

} else {

System.out.println("Consonent");

}

in.close();

}

}