1. Operator Precedence in Java

Definition:

Operator precedence determines the order in which operators are evaluated in expressions.

Order of Precedence (from highest to lowest):

Precedence	Operator Type	Operators
1	Postfix	expr++ expr--
2	Unary	++expr --expr + - ~ !
3	Multiplicative	* / %
4	Additive	+ -
5	Relational	< > <= >=
6	Equality	== !=
7	Logical AND	&&
8	Logical OR	`
9	Conditional (ternary)	? :
10	Assignment	= += -= *= /= %=

Example:

```
public class OperatorPrecedence {
    public static void main(String[] args) {
        int a = 10, b = 5, c = 2;
        int result = a + b * c;  // b * c = 10, a + 10 = 20
        System.out.println("Result = " + result);
    }
}
```

Output:

```
Result = 20
```

2. Control Statements

a. If Statement

```
int num = 5;
if (num > 0) {
   System.out.println("Positive number");
}
```

Output:

```
Positive number
```

---

b. If-Else Statement

```
int num = -3;
if (num > 0) {
   System.out.println("Positive");
} else {
   System.out.println("Negative");
}
```

Output:

```
Negative
```

---

c. Nested If Statement

```
int num = 10;
if (num >= 0) {
   if (num % 2 == 0) {
      System.out.println("Even number");
   }
}
```

Output:

```
Even number
```

d. If-Else Ladder

```
int marks = 75;
if (marks >= 90) {
   System.out.println("Grade A");
} else if (marks >= 75) {
   System.out.println("Grade B");
} else {
   System.out.println("Grade C");
}
```

Output:

```
Grade B
```

---

e. Switch Statement

```
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");
}
```

Output:

```
Wednesday
```

3. Looping Statements

a. While Loop

```
int i = 1;

while (i <= 5) {

    System.out.print(i + " ");

    i++;

}
```

Output:

```
1 2 3 4 5
```

---

b. Do-While Loop

```
int i = 1;

do {

    System.out.print(i + " ");

    i++;

} while (i <= 5);
```

Output:

```
1 2 3 4 5
```

---

c. For Loop

```
for (int i = 1; i <= 5; i++) {

    System.out.print(i + " ");

}
```

Output:

```
1 2 3 4 5
```

---

d. For-each Loop (Enhanced for Loop)

```
int[] numbers = {1, 2, 3, 4, 5};
```

```
for (int num : numbers) {
    System.out.print(num + " ");
}
```

Output:

```
1 2 3 4 5
```

---

4. Jump Statements

a. Break Statement

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) break;
    System.out.print(i + " ");
}
```

Output:

```
1 2
```

---

b. Continue Statement

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    System.out.print(i + " ");
}
```

Output:

```
1 2 4 5
```

Scope and Lifetime of Variables in Java

✅ 1. What is Variable Scope?

Scope refers to the **part of the program** where a variable is **accessible** or **visible**.

Java has the following variable scopes:

- **Local Scope**
- **Instance Scope (Object-level)**
- **Static/Class Scope**
- **Block Scope**

✅ 2. What is Variable Lifetime?

Lifetime of a variable is the **time duration** from when the variable is **created** to when it is **destroyed** (i.e., memory is deallocated).

✅ 3. Types of Variable Scope with Examples

🔹 A. Local Variables

- Declared inside **methods**, **constructors**, or **blocks**.
- Only accessible **within that method or block**.
- Memory is allocated when the method is **called** and destroyed after **execution ends**.

```
public class LocalScopeExample {
  public void show() {
    int x = 10; // Local variable
    System.out.println("x = " + x);
  }
}
```

Output:

```
x = 10
```

📌 Cannot access x outside the show() method.

🔹 **B. Instance Variables**

- Declared **inside a class** but **outside any method**.
- Each **object** of the class has its **own copy**.
- Lifetime: **Till object exists in memory.**

```
public class InstanceScopeExample {
   int count = 5; // Instance variable
   public void display() {
      System.out.println("Count = " + count);
   }
}
```

Output:

```
Count = 5
```

📌 count can be accessed by any **non-static method** of the class.

🔹 **C. Static (Class) Variables**

- Declared with the static keyword inside a class.
- Belongs to the **class**, not to objects.
- Only **one copy exists**, shared among all objects.
- Lifetime: From **class loading** till **program termination**.

```
public class StaticScopeExample {
   static int shared = 100; // Static variable

   public static void printShared() {
      System.out.println("Shared = " + shared);
   }
}
```

Output:

```
Shared = 100
```

📌 Accessed using class name or any static method.

🔹 **D. Block Scope**

- Declared inside **if**, **for**, or **while** blocks.
- Exists **only within** that block.

```
public class BlockScopeExample {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            int square = i * i; // block scope
            System.out.println("Square of " + i + " = " + square);
        }
        // System.out.println(square); // ❌ Error: 'square' cannot be accessed here
    }
}
```

Output:

```
Square of 0 = 0
Square of 1 = 1
Square of 2 = 4
```

Summary Table

Scope Type	Where Declared	Access	Lifetime
Local Variable	Inside methods or blocks	Only in that method/block	Till method/block finishes
Instance Variable	Inside class, outside method	Via object	Till object is referenced
Static Variable	Inside class with static	Via class or static methods	Till class is unloaded
Block Variable	Inside {} like loop/if	Only inside the block	Till block ends

---

✅ 5. Best Practices

- Use **local variables** where possible to avoid memory leaks.
- Use **instance variables** for maintaining object state.
- Use **static variables** for constants or shared resources.
- Avoid using same variable names in nested scopes.

S.No	Program Title	Definition / Description
1	**Number Pyramid**	Print numbers in a pyramid pattern row-wise: 1 → 2 3 → 4 5 6, etc.
2	**Count and Capitalize Vowels**	Count total vowels in a string and convert lowercase vowels to uppercase.
3	**Prime Without % Operator**	Check if a number is prime without using the modulus (%) operator.
4	**Reverse a Number**	Input a number and print its reverse (e.g., 123 → 321).
5	**Sum of Digits**	Calculate the sum of digits of a number using a loop.
6	**Palindrome Number**	Check if a number reads the same backward and forward.
7	**Armstrong Number**	Verify whether a 3-digit number is an Armstrong number (e.g., 153).
8	**Fibonacci Series**	Generate Fibonacci series up to n terms.
9	**Pattern of Alphabets**	Print a character pattern using alphabets like: A → B C → D E F
10	**Factorial Without Recursion**	Calculate factorial using loops (not recursion).
11	**Sum of Even/Odd Digits**	Separate and add even and odd digits of an input number.
12	**Count Frequency of Each Character**	Find how many times each character appears in a string.
13	**Swap Two Numbers Without Third Variable**	Swap two numbers using only arithmetic operations.
14	**Print Only Prime Digits**	Extract and print only prime digits from a number (2, 3, 5, 7).
15	**Find GCD Without % Operator**	Calculate the Greatest Common Divisor (GCD) using subtraction or loops.

S.No	Program Title	Definition / Description
16	**Replace Spaces with Underscore**	Replace all spaces in a sentence with underscores (_).
17	**Print Triangle of Stars**	Output a right-angle triangle using asterisks (*).
18	**Check Armstrong in Range**	List all Armstrong numbers between two given numbers.
19	**Sum of Squares of Digits**	Calculate the sum of squares of each digit (e.g., 123 → $1^2+2^2+3^2$).
20	**Reverse Words in a String**	Reverse each word in a sentence individually (e.g., "I am here" → "I ma nahsI").