Recursion in Java - From Basics to Advanced

What is Recursion?

Recursion is a programming technique where a method calls itself to solve a smaller instance of the problem until a base condition is met.

Basic Syntax in Java

Example:

```
public class RecursionExample {
    static void printNumbers(int n) {
        if (n == 0) return;
        System.out.println(n);
        printNumbers(n - 1);
    }
    public static void main(String[] args) {
        printNumbers(5);
    }
}
```

How Recursion Works (Call Stack)

Each recursive call is pushed onto the call stack. When the base case is reached, the stack unwinds and the results are returned back.

Types of Recursion

- Direct Recursion
- Indirect Recursion
- Tail Recursion
- Head Recursion

Examples of Recursion

- Factorial
- Fibonacci
- Sum of Digits
- Reverse a String

When to Use Recursion

Use when the problem can be broken into similar subproblems or when iterative logic is too complex.

Drawbacks of Recursion

- StackOverflowError for deep calls
- Higher memory usage
- Slower if not optimized

Optimizing Recursion

- Use Memoization to store subproblem results
- Apply tail recursion (not optimized in Java)

Advanced Recursive Problems

- Backtracking: N-Queens, Sudoku
- Divide & Conquer: Merge Sort, Quick Sort
- Generating Subsets, Permutations

Common Mistakes

- Missing base case
- Infinite recursion
- Not reducing the problem size
- Not returning recursive results