Recursion is a powerful programming technique where a method calls itself to solve a problem. It's commonly used for tasks that can be divided into smaller, similar subproblems. To master recursion, it's important to understand two critical components:

1. **Base Case**: A condition that stops the recursion from continuing indefinitely. Without it, the function would call itself forever.
2. **Recursive Case**: The part where the function calls itself, breaking the problem into smaller parts.

In this tutorial, we'll explore recursion in Java with two simple problems.

Problem 1: Factorial Calculation

Objective: Calculate the factorial of a number using recursion.

What is a Factorial?

The factorial of a number `n` is the product of all positive integers less than or equal to `n`. Mathematically, it's defined as:

$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$n! = n \times (n - 1) \times (n - 2) \times \dots \times 1n!=n×(n−1)×(n−2)×⋯×1

The factorial of `0` is defined as `1`.

Recursive Breakdown

- **Base Case**: If `n == 0`, return `1` because `0! = 1`.
- **Recursive Case**: If `n > 0`, return `n * factorial(n-1)`.

Problem 2: Sum of Natural Numbers

Objective: Find the sum of natural numbers up to `n` using recursion.

What is the Sum of Natural Numbers?

The sum of natural numbers from `1` to `n` is:

$S(n) = 1 + 2 + 3 + \dots + n$

Recursive Breakdown

- **Base Case**: If `n == 0`, return `0` because the sum of zero numbers is `0`.
- **Recursive Case**: If `n > 0`, return `n + sum(n-1)`.