Recursion in Java – Notes

1️⃣ Definition

- **Recursion** = A method calling **itself** directly or indirectly.
- **Purpose:**
- Solve repetitive tasks efficiently
- Simplify complex problems like factorial, Fibonacci, tree traversal

2️⃣ Types of Recursion

1. **Direct Recursion:**
2. Method calls itself

```
void fun() {
    fun();
}
```

3. **Indirect Recursion:**
4. Method A calls B, Method B calls A

```
void funA() { funB(); }
void funB() { funA(); }
```

3️⃣ Structure of a Recursive Method

1. **Base Case:** Condition to stop recursion
2. **Recursive Case:** Method calls itself with a smaller problem

Example – Factorial:

```
int factorial(int n) {
    if (n == 0) return 1; // base case
    return n * factorial(n - 1); // recursive call
}
```

4️⃣ Example – Fibonacci Series

```
int fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
```

```
    return fib(n-1) + fib(n-2);
}
```

5️⃣ Example – Sum of Array Elements

```
int sum(int[] arr, int n) {
    if (n <= 0) return 0; // base case
    return sum(arr, n-1) + arr[n-1]; // recursive call
}
```

6️⃣ Key Points

- Every recursive method **must have base case**
- Too deep recursion → **StackOverflowError**
- Recursion can often be converted to **iteration**
- Recursive solution is often **shorter & elegant**, sometimes less efficient

💡 **Shortcut:**

> Recursion = Method calls itself → Base case stops → Solve smaller problems → Combine results