

Recursion



By: Mamoun Nawahdah (Ph.D.)
2022



Recursion

- To use recursion is to program using recursive methods—that is, to use methods that invoke themselves.
- A recursive method is one that invokes itself directly or indirectly.
- Case Study: Computing Factorials

$$0! = 1;$$

$$n! = n \times (n - 1)!; n > 0$$



Computing Factorial

- ❖ Let **factorial(n)** be the method for computing **n!**.
- ❖ If you call the method with **n = 0**, it immediately returns the result.
- ❖ The method knows how to solve the simplest case, which is referred to as the **base case** or the **stopping condition**.
- ❖ If you call the method with **n > 0**, it reduces the problem into a **subproblem** for computing the factorial of **n - 1**.
- ❖ The **subproblem** is essentially the same as the original problem, but it is simpler or smaller.



Computing Factorial

factorial(4) = 4 * factorial(3)

= 4 * (3 * factorial(2))

= 4 * (3 * (2 * factorial(1)))

= 4 * (3 * (2 * (1 * factorial(0))))

= 4 * (3 * (2 * (1 * 1)))

= 4 * (3 * (2 * 1))

= 4 * (3 * 2)

= 4 * (6)

= 24

factorial(0) = 1;

factorial(n) = n*factorial(n-1);



factorial(n)

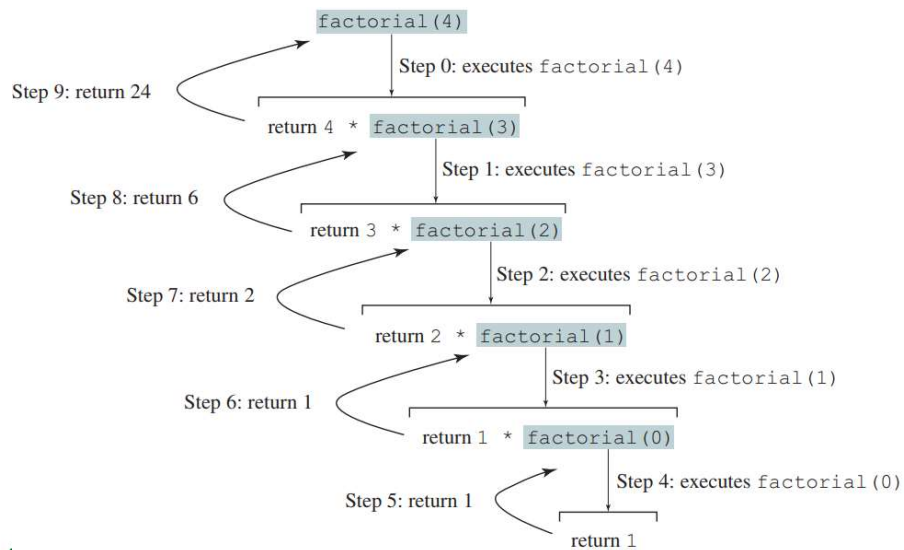
```
int factorial(int n) {
    if (n == 0)                // base case
        return 1;
    else                        // recursion
        return n * factorial(n - 1);
}
```

Notes:

- For a recursive method to terminate, the problem must eventually be reduced to a **stopping case**, at which point the method returns a result to its caller.
- If recursion does not reduce the problem in a manner that allows it to eventually converge into the base case or a base case is not specified, **infinite recursion** can occur. The method runs infinitely and causes a **StackOverflowError**.



Invoking factorial(4)



Fibonacci Numbers

Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55 89...

indices: 0 1 2 3 4 5 6 7 8 9 10 11

$\text{fib}(0) = 0;$

$\text{fib}(1) = 1;$

$\text{fib}(\text{index}) = \text{fib}(\text{index} - 1) + \text{fib}(\text{index} - 2); \text{index} \geq 2$

$$\begin{aligned}
 \text{fib}(3) &= \text{fib}(2) + \text{fib}(1) \\
 &= (\text{fib}(1) + \text{fib}(0)) + \text{fib}(1) \\
 &= (1 + 0) + \text{fib}(1) \\
 &= 1 + \text{fib}(1) \\
 &= 1 + 1 = 2
 \end{aligned}$$



7

Fibonacci Numbers, cont.

```

public static long fib(long index) {
    if (index == 0) // Base case
        return 0;
    else if (index == 1) // Base case
        return 1;
    else // Reduction and recursive calls
        return fib(index - 1) + fib(index - 2);
}

```



8

Characteristics of Recursion

All recursive methods have the following characteristics:

- The method is implemented using an **if-else** or a **switch** statement that leads to different cases.
- One or more base cases (the simplest case) are used to stop recursion.
- Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.



9

Characteristics of Recursion

- ❖ In general, to solve a problem using recursion, you break it into subproblems.
- ❖ If a subproblem resembles the original problem, you can apply the same approach to solve the subproblem recursively.
- ❖ This subproblem is almost the same as the original problem in nature with a smaller size.



10

Problem Solving Using Recursion

- ❖ Let us consider a simple problem of printing a message for n times.
- ❖ You can break the problem into two subproblems:
 - one is to print the message one time and the other is to print the message for n-1 times.
 - The second problem is the same as the original problem with a smaller size.
 - The base case for the problem is n==0. You can solve this problem using recursion as follows:

```
public static void nPrintln(String message, int times) {
    if (times >= 1) {
        System.out.println(message);
        nPrintln(message, times - 1);
    } // The base case is times == 0
}
```



Think Recursively

- ❖ Many of the problems can be solved using recursion if you *think recursively*.
- ❖ For example, the palindrome problem can be solved recursively as follows:

```
public static boolean isPalindrome(String s) {
    if (s.length() <= 1) // Base case
        return true;
    else if (s.charAt(0) != s.charAt(s.length() - 1)) // Base case
        return false;
    else
        return isPalindrome(s.substring(1, s.length() - 1));
}
```



12