

# Recursion

---

CSGE601021 Dasar-Dasar Pemrograman 2  
Fakultas Ilmu Komputer Universitas Indonesia

# Why?

Suppose you want to find all the files under a directory that contains a particular word. How do you solve this problem? There are several ways to solve this problem. An intuitive solution is to use recursion by searching the files in the subdirectories recursively.

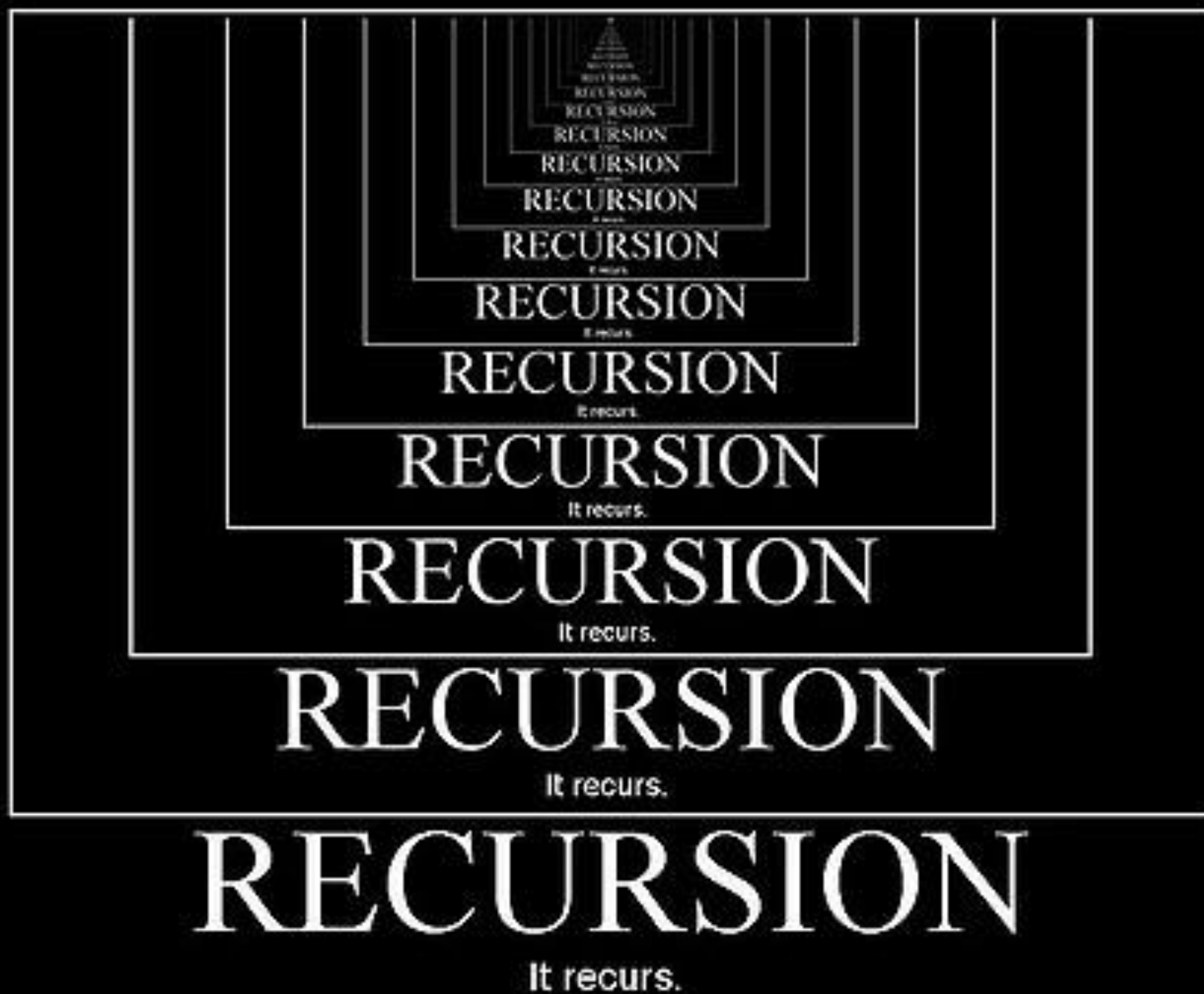
# Why?

$$0! = 1$$

$$n! = n \cdot (n - 1)!$$

Try this one:

Create a method to compute the factorial of a non-negative integer  $n$ , with **no loops**.



# Solution = Recursion

$$0! = 1$$

$$n! = n \cdot (n - 1)!$$

Create a method to compute the factorial of a non-negative integer  $n$ , with **no loops**.

```
public static int factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

# Solution = Recursion

$$0! = 1$$

$$n! = n \cdot (n - 1)!$$

Create a method to compute the factorial of a non-negative integer  $n$ , with **no loops**.

```
public static int factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

Recursion = loops with no loops!

## Bonus: One-liner solution

$$0! = 1$$

$$n! = n \cdot (n - 1)!$$

Create a method to compute the factorial of a non-negative integer  $n$ , with **no loops**.

```
public static int factorialRec(int n) {  
  
    return n == 0 ? 1 : n * factorialRec(n-1);  
  
}
```


Short if-else: `condition ? trueCase : falseCase`

# Recursion: Base case + recursive case



```
public static int factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```





# Recursion: Base case + recursive case

```
public static int factorialRec(int n) {  
  
    if (n == 0)  Base case  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

# Recursion: Base case + recursive case

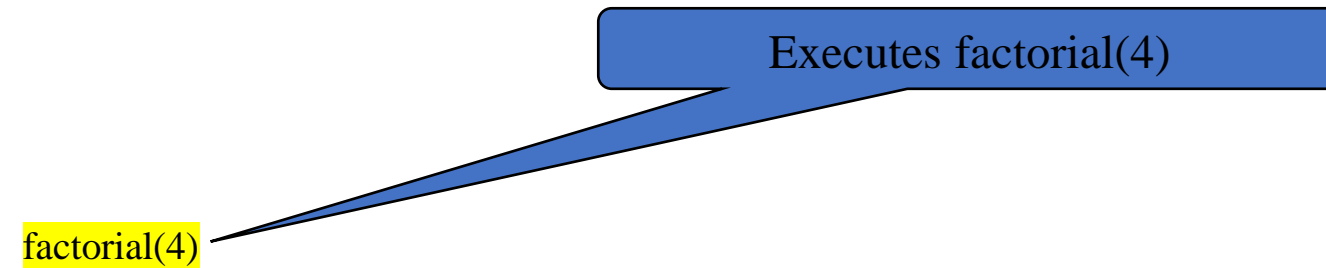
```
public static int factorialRec(int n) {  
  
    if (n == 0)  Base case  
        return 1;  
    else  
        return n * factorialRec(n-1);  Recursive case  
}
```

# Recursion: Base case + recursive case

```
public static int factorialRec(int n) {  
  
    if (n == 0)  Base case  
        return 1;  
    else  
        return n * factorialRec(n-1);  Recursive case  
}
```

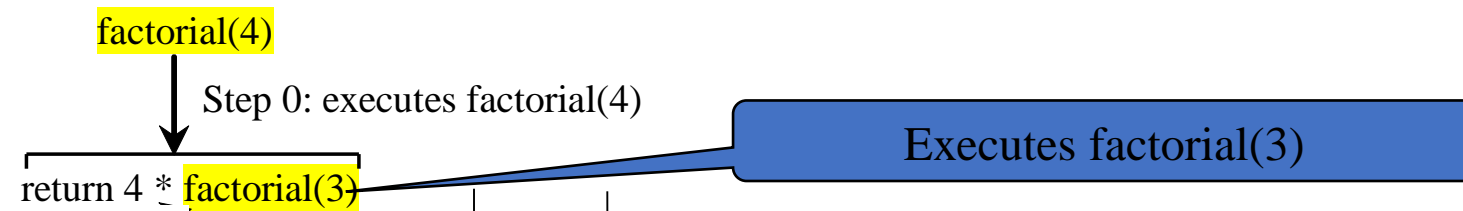
**( ! ) Make sure argument in recursive case gets simpler!**

# Trace Recursive factorial

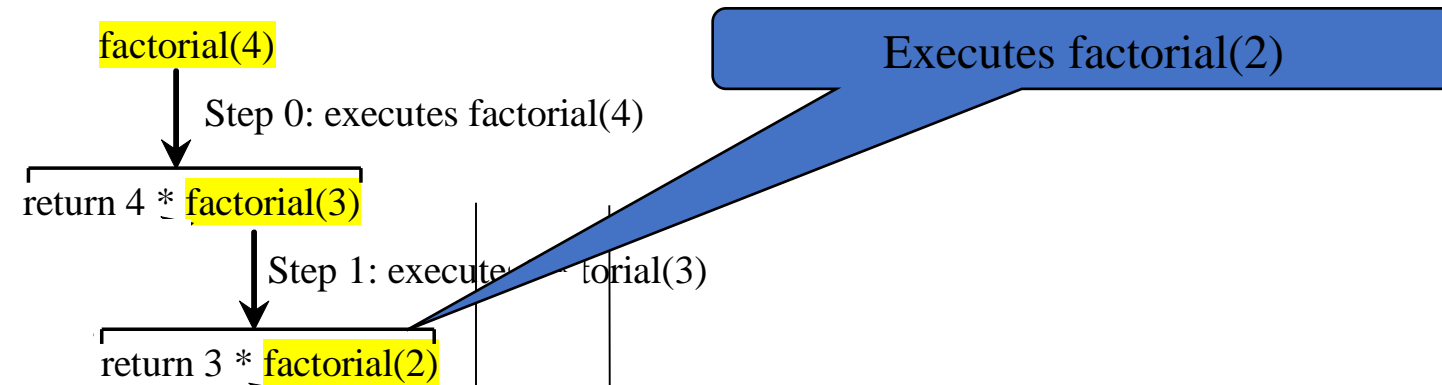


Stack
Space Required for factorial(4)
Main method

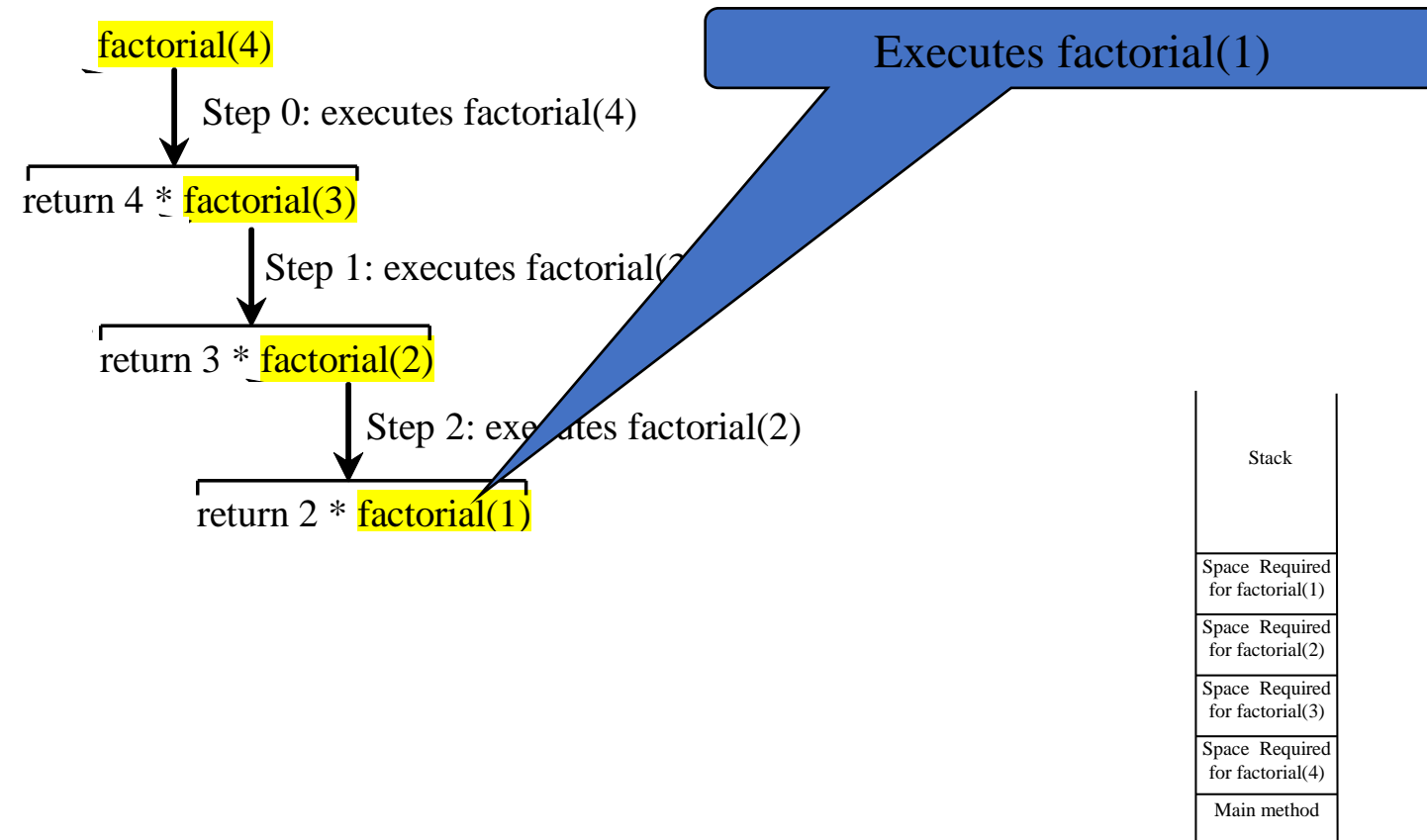
# Trace Recursive factorial



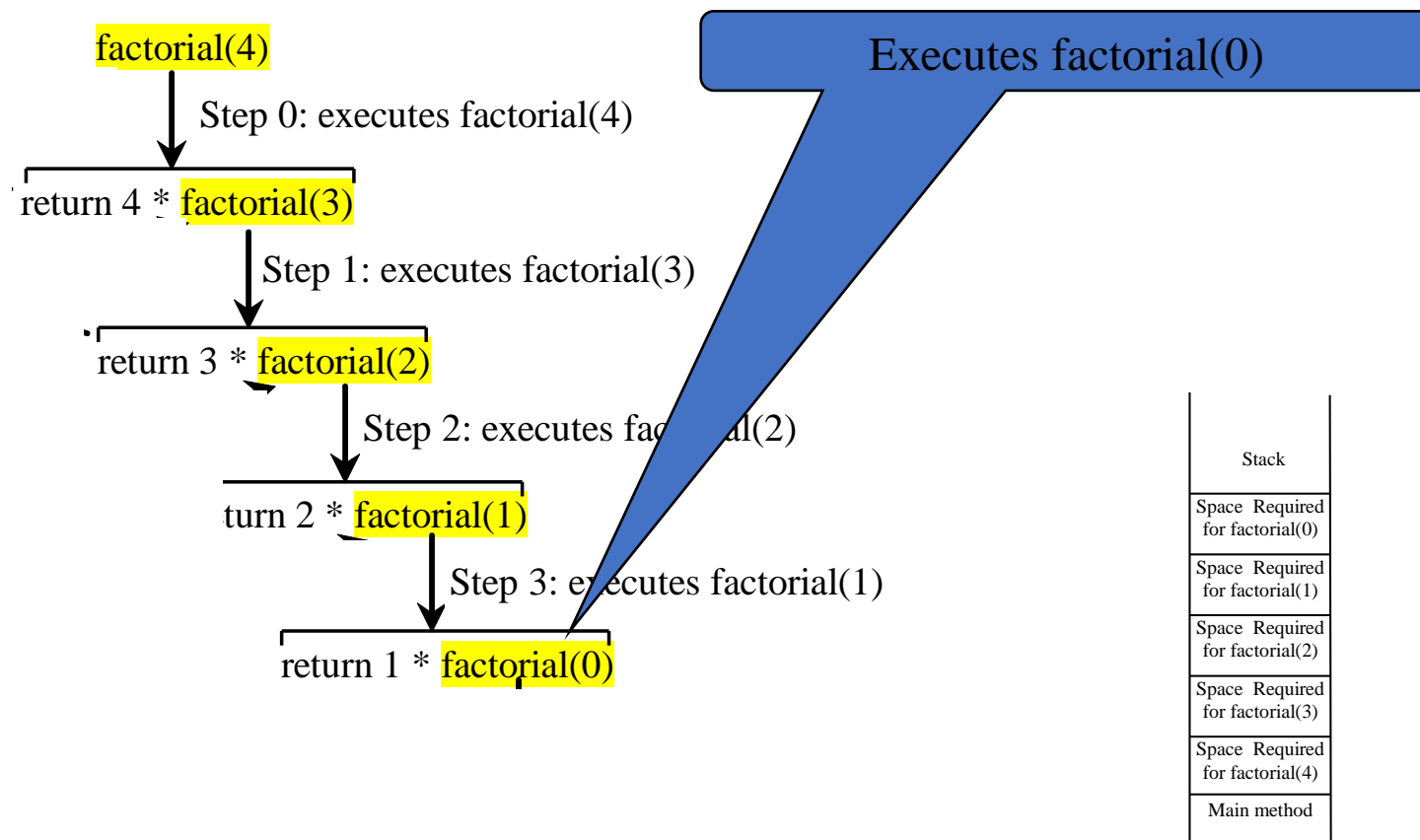
# Trace Recursive factorial



# Trace Recursive factorial

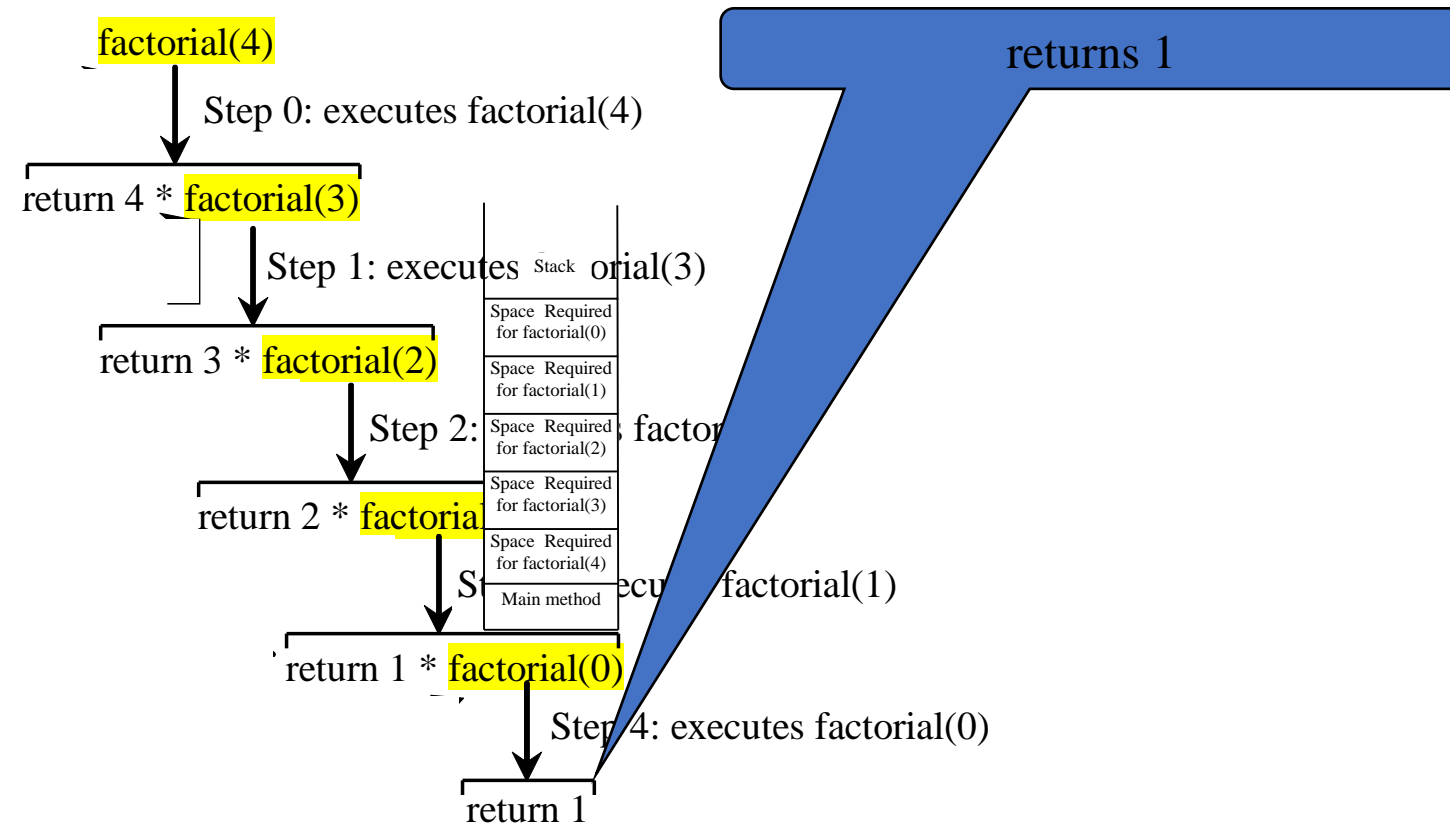


# Trace Recursive factorial

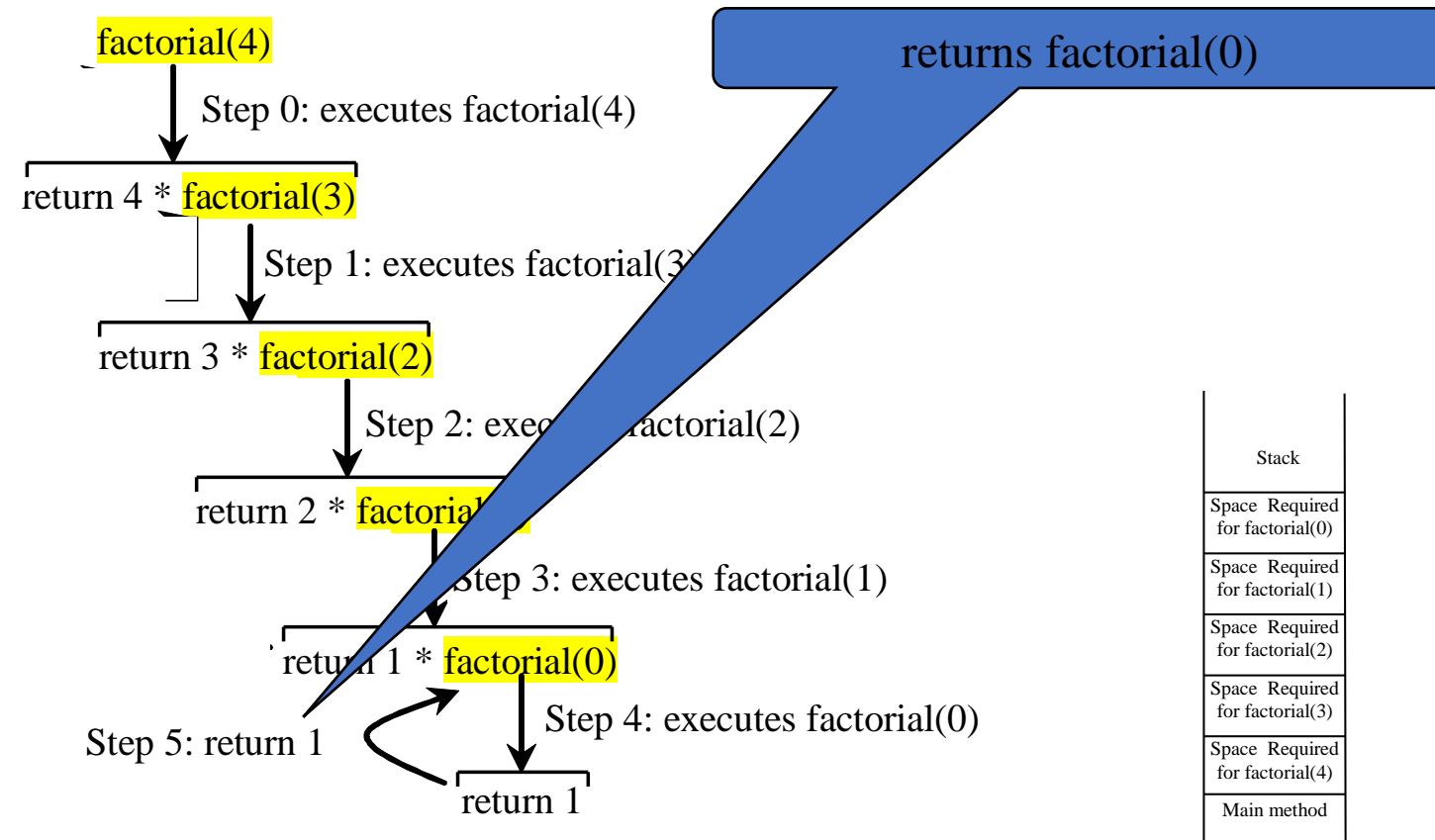




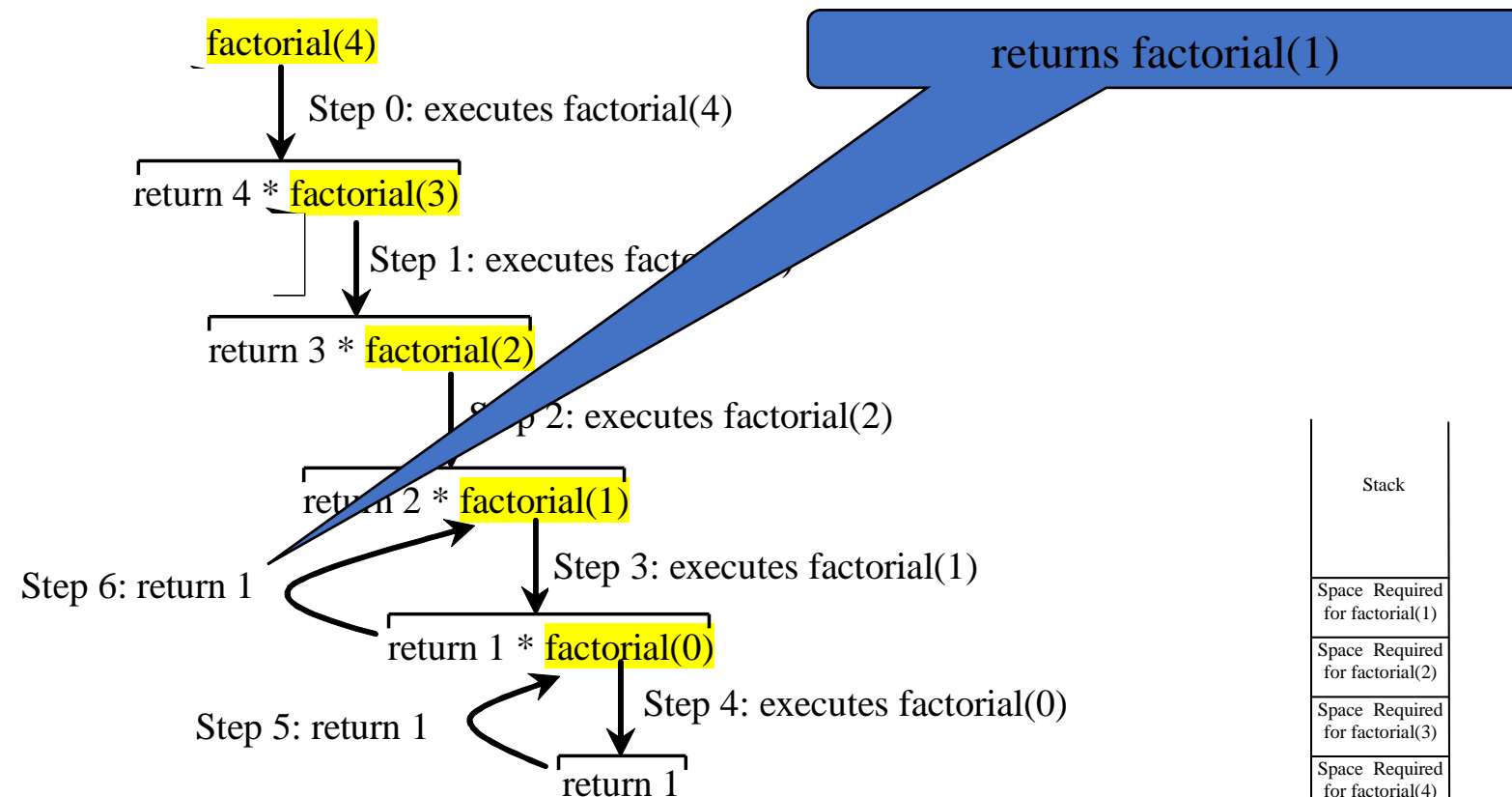
# Trace Recursive factorial



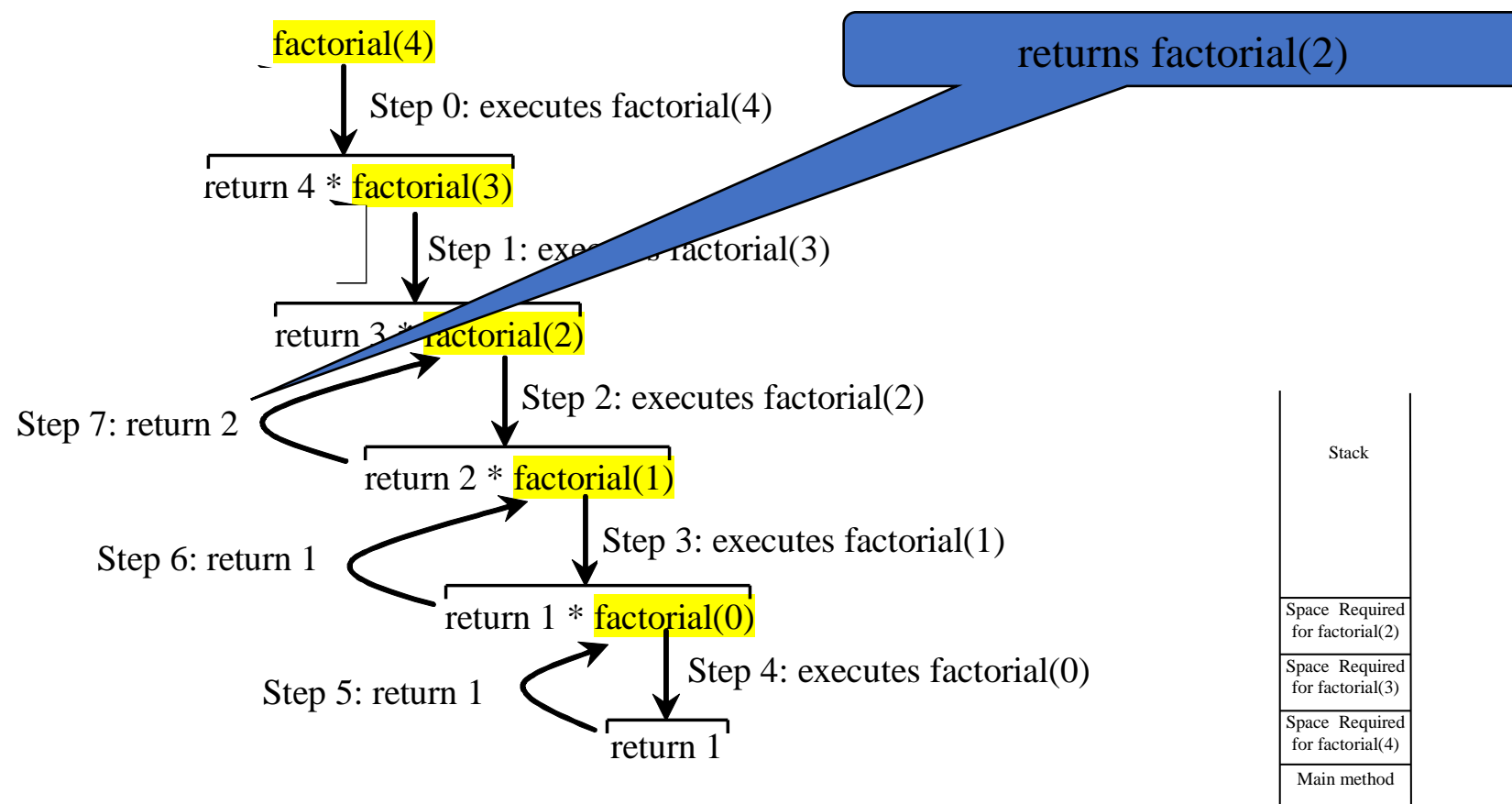
# Trace Recursive factorial



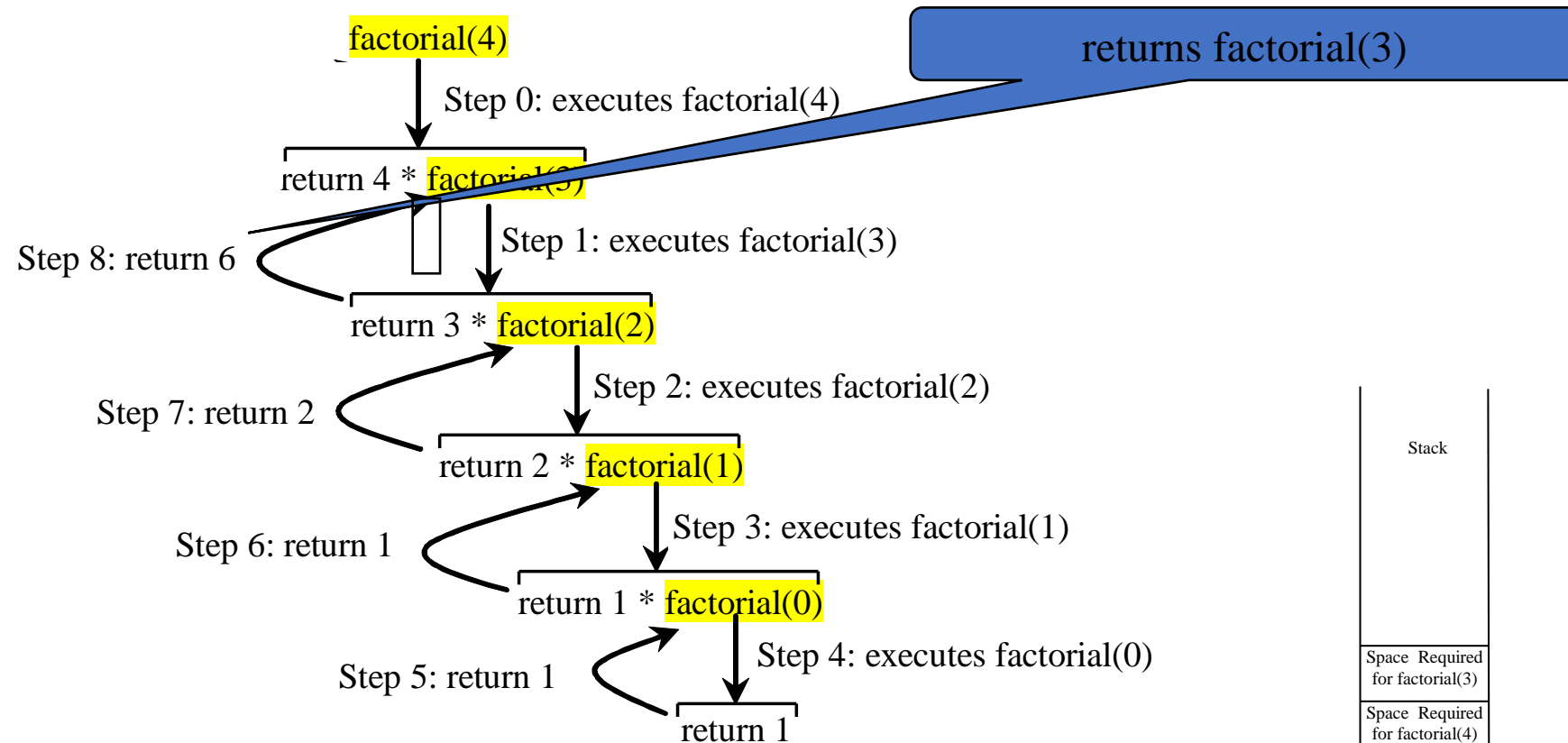
# Trace Recursive factorial



# Trace Recursive factorial

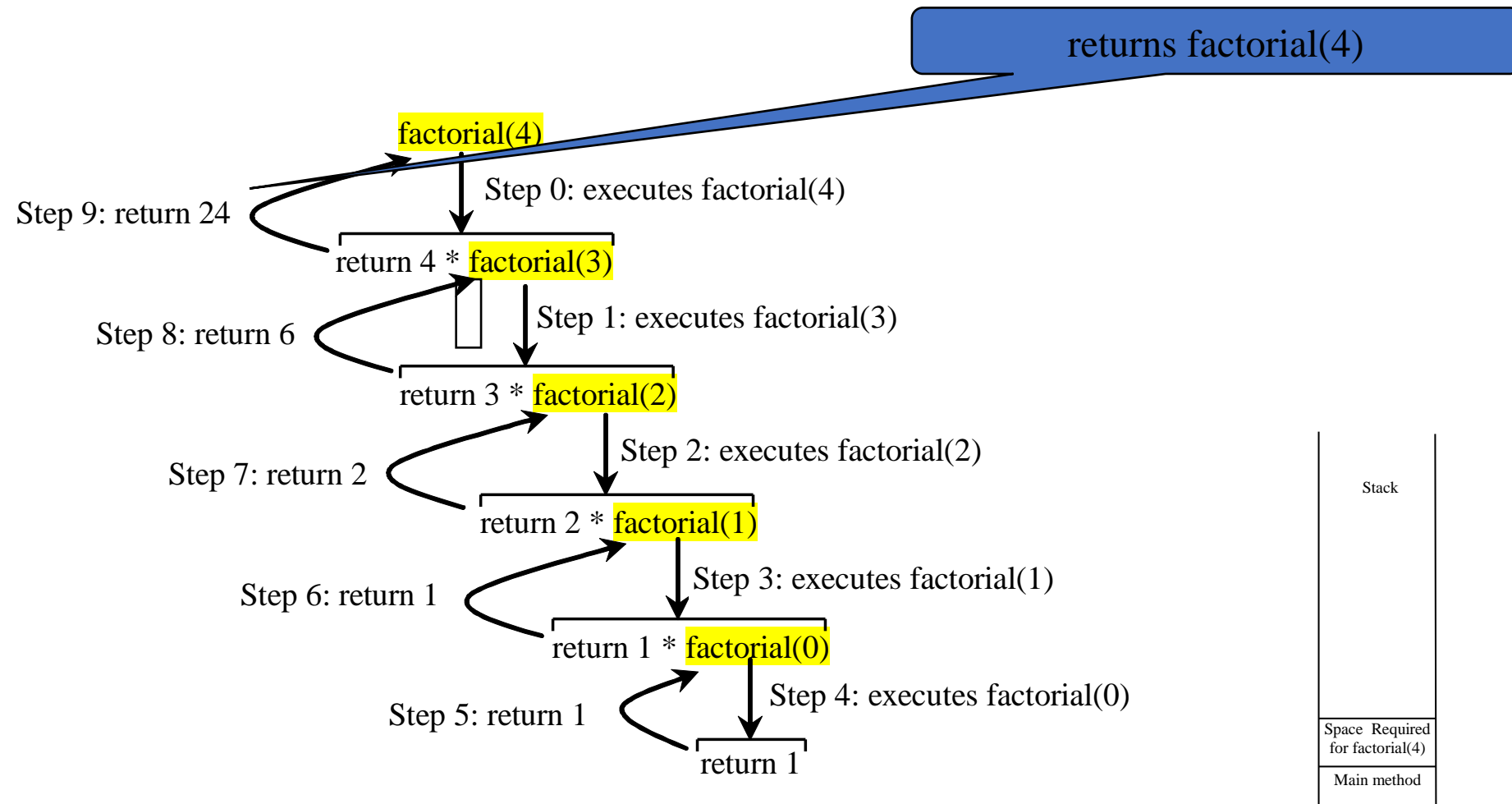


# Trace Recursive factorial



Stack
Space Required for factorial(3)
Space Required for factorial(4)
Main method

# Trace Recursive factorial



# Quiz time: Recursive palindrom checking

Create a recursive method to check if a string is a palindrom.

**Example:**

```
System.out.println(isPalin("ada ada"));
System.out.println(isPalin("malam"));
System.out.println(isPalin("o"));
System.out.println(isPalin("lalala"));
```

# Quiz time: Recursive palindrom checking

Create a recursive method to check if a string is a palindrom.

## **Example:**

```
System.out.println(isPalin("ada ada"));
System.out.println(isPalin("malam"));
System.out.println(isPalin("o"));
System.out.println(isPalin("lalala"));
```

## **Output:**

```
true
true
true
false
```



# Quiz time: Recursive palindrom checking (1)

```
public static boolean isPalin(String s) {  
    if(s.length() == 0 || s.length() == 1)  
        return true;  
    else  
        return s.charAt(0) == s.charAt(s.length()-1) &&  
            isPalin(s.substring(1, s.length()-1));  
}
```

## Quiz time: Recursive palindrom checking (2)

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

# Don't do this anywhere!

```
public static void forever() {  
    forever() ;  
}
```

# Don't do this anywhere!

```
public static void forever() {  
    forever() ;  
}
```

***( X ) When run, it throws `java.lang.StackOverflowError`!***

# Quiz time: Guess what the code does!

```
public static void cd(int n, String m) {  
  
    if (n == 0) {  
        System.out.println(m);  
    } else {  
        System.out.println(n);  
        cd(n-1, m);  
    }  
  
}
```

# Quiz time: Guess what the code does!

```
public static void cd(int n, String m) {  
  
    if (n == 0) {  
        System.out.println(m);  
    } else {  
        System.out.println(n);  
        cd(n-1, m);  
    }  
  
}
```

Answer:

Count down from n to 1, then print message m

# Bonus: Countdown with timer!

```
public static void cd(int n, String m) {  
  
    if (n == 0) {  
        System.out.println(m);  
    } else {  
        System.out.println(n);  
        try {  
            Thread.sleep(1000); // sleep for 1000 ms  
        } catch (InterruptedException e) {}  
        cd(n-1, m);  
    }  
  
}
```

Quiz time: What's the output of `cd2(5, "HBD!")` !

```
public static void cd2(int n, String m) {  
  
    if (n == 0) {  
        System.out.println(m);  
    } else {  
        cd2(n-1, m);  
        System.out.println(n);  
    }  
  
}
```



Quiz time: What's the output of `cd2(5, "HBD!")` !

```
public static void cd2(int n, String m) {  
  
    if (n == 0) {  
        System.out.println(m);  
    } else {  
        cd2(n-1, m);  
        System.out.println(n);  
    }  
  
}
```

**Output**

HBD!

1

2

3

4

5

# Decimal-to-binary conversion

**For example, to convert 23 to its binary representation, you first repeatedly divide 23 by 2:**

23 / 2	is 11	remainder 1
11 / 2	is 5	remainder 1
5 / 2	is 2	remainder 1
2 / 2	is 1	remainder 0
1 / 2	is 0	remainder 1

**Then, you read these remainders from bottom to top.  
So, 23 in binary is 10111.**

## Quiz time: Recursive decimal-to-binary conversion

Given a positive, base-10 integer, print its binary form.

PS: Base case is when the integer is 0, print nothing.

```
public static void printDecToBin(int dec) {  
  
    // complete the code..  
  
}
```

## Quiz time: Recursive decimal-to-binary conversion

Given a positive, base-10 integer, print its binary form.

PS: Base case is when the integer is 0, print nothing.

```
public static void printDecToBin(int dec) {  
  
    if (dec > 0) {  
        printDecToBin(dec / 2);  
        System.out.print(dec % 2);  
    }  
  
}
```

# Quiz time: Fibonacci sequence

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

$$\text{fib}(2) = 1$$

$$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$$

## Quiz time: Fibonacci sequence

 $\text{fib}(1) = 1$  $\text{fib}(2) = 1$  $\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$ 

```
public static int fib(int n) {  
  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib(n-1) + fib(n-2);  
  
}
```

# Quiz time: Print a number in reverse order

Example:

input: 12345      output: 54321

```
public static void printReverse(int number) {  
  
    // complete the code..  
  
}
```

# Quiz time: Print a number in reverse order

Example:

input: 12345      output: 54321

```
public static void printReverse(int x) {  
  
    if (x == 0) return;  
    else {  
        System.out.print(x % 10);  
        printReverse(x / 10);  
    }  
}
```



# Revisiting this factorial recursion

Try it out with  $n = 13$ , what happens?

```
public static int factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

# Revisiting this factorial recursion

Try it out with  $n = 13$ , what happens?

It returns 1,932,053,504

```
public static int factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

# Revisiting this factorial recursion

Try it out with  $n = 13$ , what happens?

It returns 1,932,053,504 (should've been 6,227,020,800). **But why?**

```
public static int factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

# Revisiting this factorial recursion

Try it out with  $n = 13$ , what happens?

It returns 1,932,053,504 (should've been 6,227,020,800). **But why?**

```
public static int factorial
```

Max value of int is 2,147,483,647

```
    if (n == 0)
```

```
        return 1;
```

```
    else
```

```
        return n * factorialRec(n-1);
```

```
}
```

# Revisiting this factorial recursion

Solution:

```
public static long factorialRec(int n) {  
  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialRec(n-1);  
  
}
```

# Tips: Avoiding infinite recursion

Most of the time, an infinite recursion will cause the program to throw a `StackOverflowError`. But if the program is slow, it may take a long time to fill the stack.

If you know which method is causing an infinite recursion, check that there is a base case. There should be some condition that makes the method return without making a recursive invocation. If not, you need to rethink the algorithm and identify a base case.

If there is a base case, but the program doesn't seem to be reaching it, add a print statement at the beginning of the method that displays the parameters. Now when you run the program you see a few lines of output every time the method is invoked, and you can see the values of the parameters. If the parameters are not moving toward the base case, you might see why not.



# THANK YOU

Credits: Chapter 5 & 6 of Think Java book by Allen Downey and Chris Mayfield  
DDP2 - Recursion