# Function part 2

Teaching Team of Programming Fundamentals 2023

# Learning Outcome

After completing this topic, students should be proficient in the following:

- ➢Mastering the basic concept of recursive function
- ➢Implementing recursive function as the one of problem-solving options

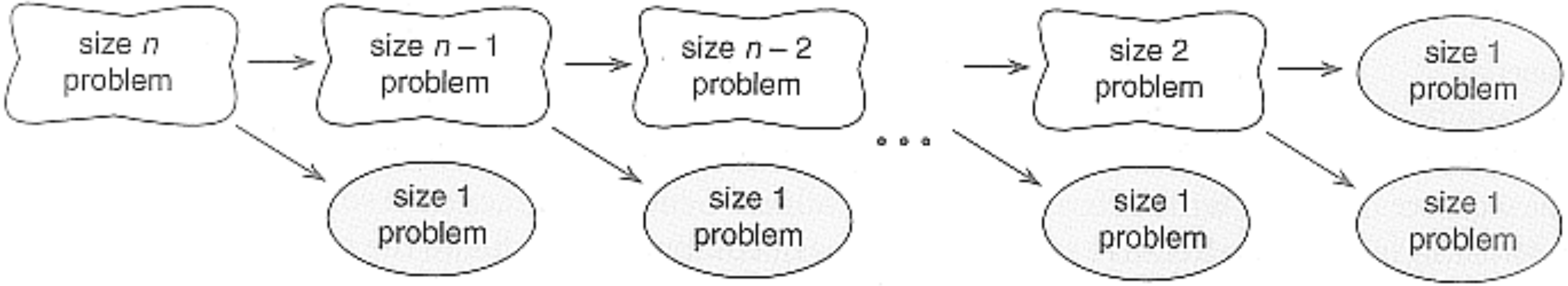
# Recursive Function

- Usually, a function is **called** by **other functions**
- A recursive function is a function that **calls itself** during its execution. In other words, the function performs a task in part and delegates the remaining task to a new invocation of itself.
- This process continues until a **base case** is reached, at which point the function returns results without making further recursive calls.

```
Return_data_type function_name (parameter){
    ...
    function_name(parameter_value);
    ...
}
```

# Recursive Function

- Problem solving in a recursive function is usually called as *decrease and conquer*
- The basic ide is, splitting the problem into the smaller problem that has a clear and simple solution.



- Assume that the problem of size 1 can be solved easily (i.e., the simple case).
- We can recursively split the problem into a problem of size 1 and another problem of size n-1.

# Component of Recursive Function

- **Base Case**
  - Base case is a condition that, when met, causes the function to stop calling itself and instead return a result without further recursion.
  - Base case is crucial to prevent the recursive calls from continuing indefinitely, leading to what is known as infinite recursion.
  - Base case represents the smallest or simplest version of the problem that can be directly solved without further recursion
- **Recursion call / Reduction step / Recursive case**
  - The recursive case defines how the function calls itself with a modified version of the problem.
  - In the recursive case, the function is applied to a smaller or simpler instance of the problem.
  - The result of the recursive call is often combined with other calculations to produce the final result.
  - Recursive function usually uses return keyword to return the result
  - The recursion call will approach convergently to the base case

# Basic Format of Recursive Function

- Basically the recusrive function will follow this format

```
if (limit_condition)
   //solve the most simple problem or base case
else
   //define the simpler problem using recursive call
```

- IF is for **base case**, while ELSE will perform **recursion call**
- Recursion call will provide ”looping” to define the simpler problem that will convergently move to the base case condition.
- To ensure **recursion termination, each recursive call must progressively approach the base case**

# Trace a Recursive Function

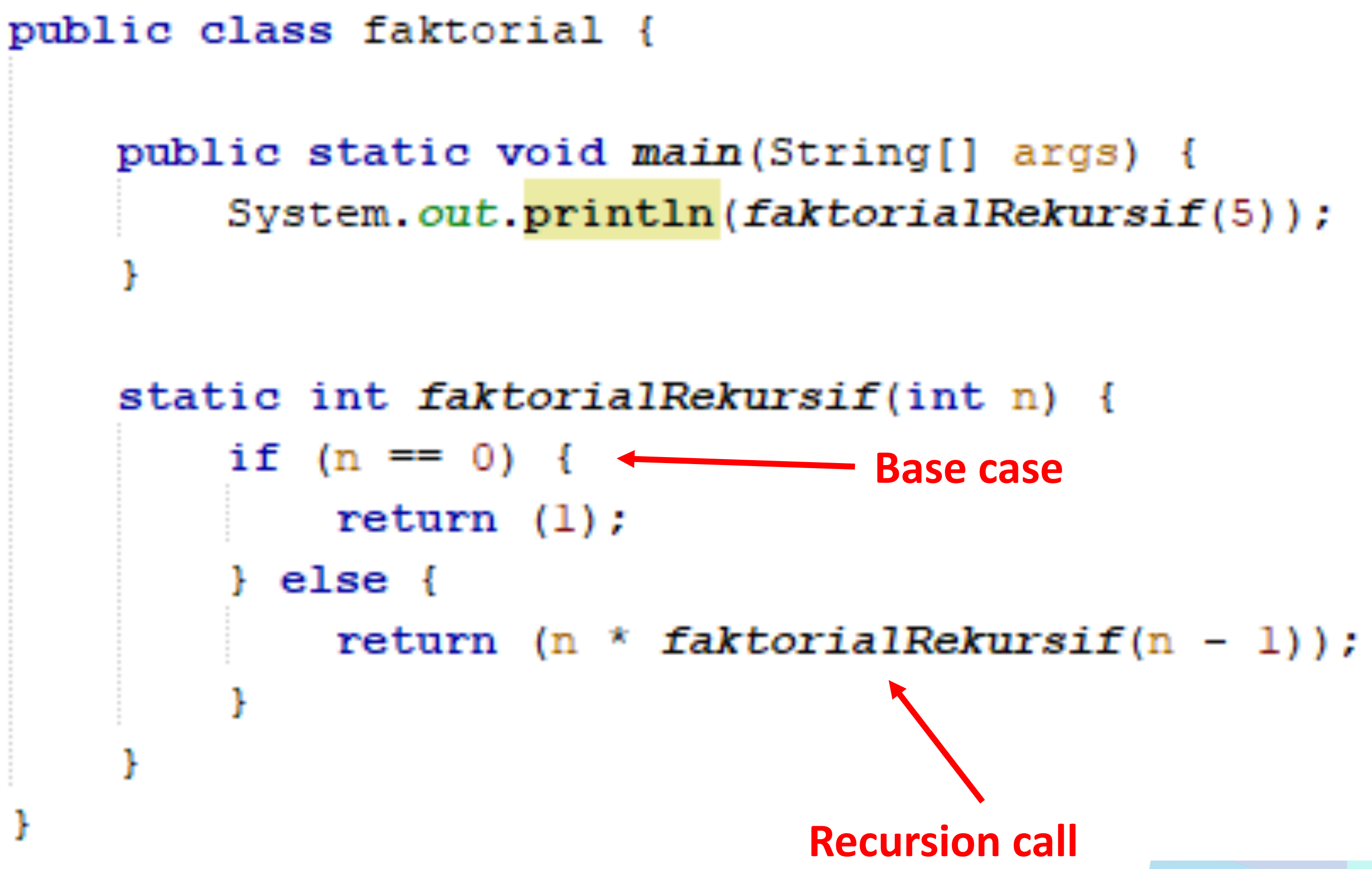
The execution of a recursive function occurs **in two stages**:

- **Expansion phase**: recursive function calls which progressively approach the base case.
- **Substitution phase**: Solutions computed in reverse, starting from the base case

# Example #1

Factorial Function

- Base case: n = 0
- Recursion call: f(n) = n * f(n-1)



```
public class faktorial {

    public static void main(String[] args) {
        System.out.println(faktorialRekursif(5));
    }

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

Base case

Recursion call

# Example #1 – Tracing the Recursive Function

**n * faktorialRekursif(n-1)**

```
faktorialRekursif(5)   = 5 * faktorialRekursif(4)
                       = 5 * (4 * faktorialRekursif(3))
                       = 5 * (4 * (3 * faktorialRekursif(2)))
                       = 5 * (4 * (3 * (2 * faktorialRekursif(1))))
                       = 5 * (4 * (3 * (2 * (1 * faktorialRekursif(0)))))
```

**Expansion phase**

```
                       = 5 * (4 * (3 * (2 * (1 * 1))))
                       = 5 * (4 * (3 * (2 * 1)))
                       = 5 * (4 * (3 * 2))
                       = 5 * (4 * 6)
                       = 5 * 24
                       = 120
```

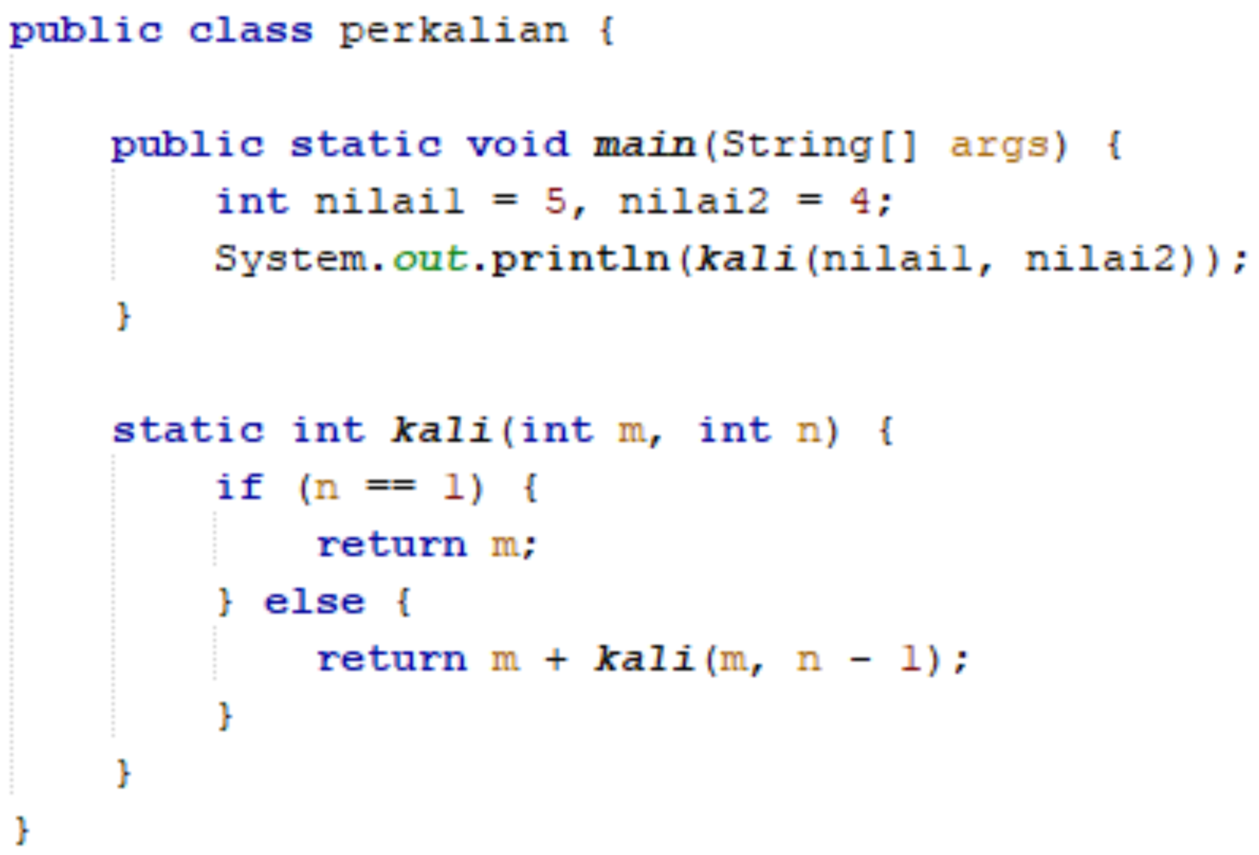
**Substitution phase**

# Example #2

- Example: suppose we want to create a recursive function to multiply integer m and integer n using addition
- We need to identify the base case and recursion call
  - **Base case**: if **n** equals to **1**, the result will be **m**
  - **Recursion call**: m * n = m + m(n-1)

$$m * n \begin{cases} m, & n = 1 \\ m + m\,(n-1), & n>1 \end{cases}$$

# Example #2 - Trace



```
public class perkalian {

    public static void main(String[] args) {
        int nilai1 = 5, nilai2 = 4;
        System.out.println(kali(nilai1, nilai2));
    }

    static int kali(int m, int n) {
        if (n == 1) {
            return m;
        } else {
            return m + kali(m, n - 1);
        }
    }
}
```

**Expansion phase**

kali(5, 4) = 5 + kali(5, 3)
= 5 + (5 + kali(5, 2))
= 5 + (5 + (5+ kali(5, 1)))

---

= 5 + (5 + (5 + 5))
= 5 + (5 + 10)
= 5 + 15
= 20

**Substitution phase**

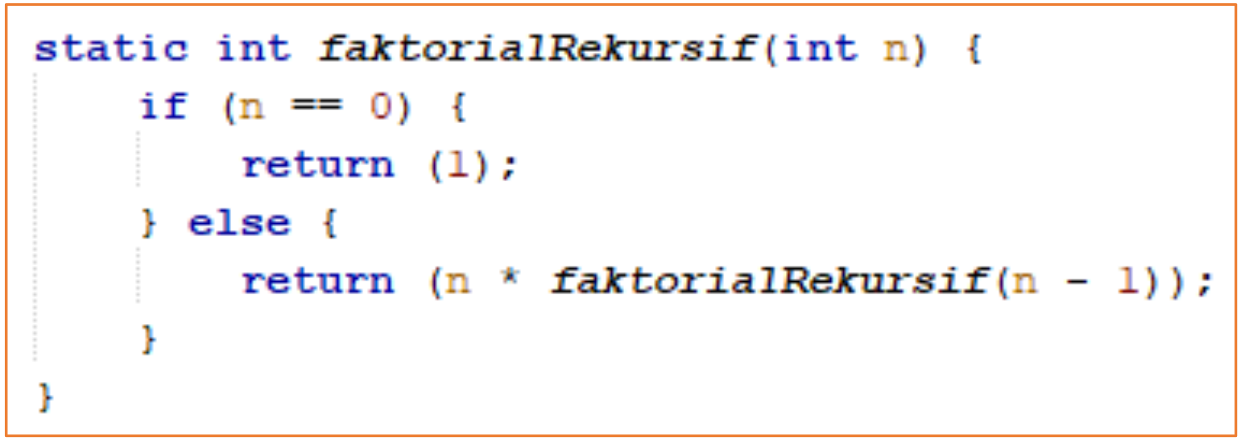
# Recursive vs Iterative Function

# Recursive vs Iterative Function

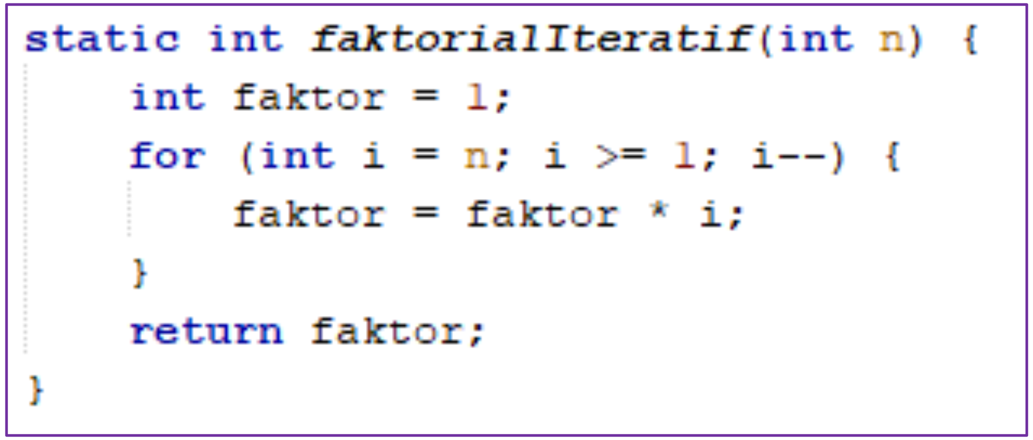
- The iteration involving selection structure (IF-ELSE) and recursive function calls
- Iteration will stop when the base case is fulfilled
- Iteration will continue endlessly if the base case is never met
- Requires more memory and higher processor workload due to multiple function calls
- Reads more clearly, models closer to the problem, for example: factorial

- Looping with repetition structures (FOR/WHILE).
- Looping will stop when the loop condition evaluates to FALSE.
- Looping will continue endlessly if the loop condition is always true.
- Requires less memory and lower processor workload as the looping process is contained within one function.
- Reads less clearly, the model is less aligned with the problem.

# Recursive vs Iterative Function

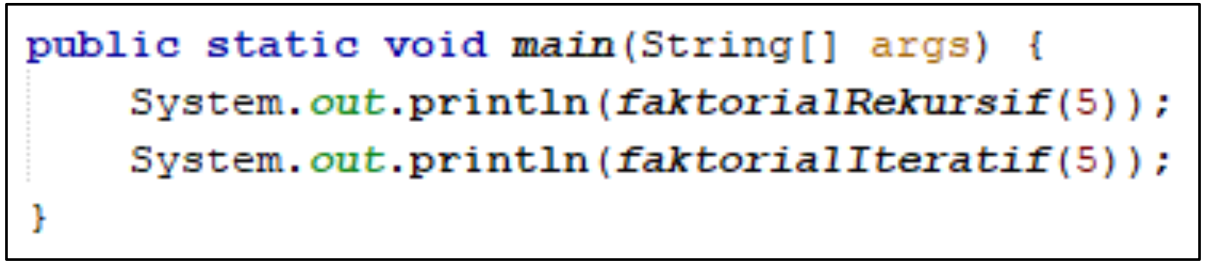


```
static int faktorialRekursif(int n) {
    if (n == 0) {
        return (1);
    } else {
        return (n * faktorialRekursif(n - 1));
    }
}
```



```
static int faktorialIteratif(int n) {
    int faktor = 1;
    for (int i = n; i >= 1; i--) {
        faktor = faktor * i;
    }
    return faktor;
}
```

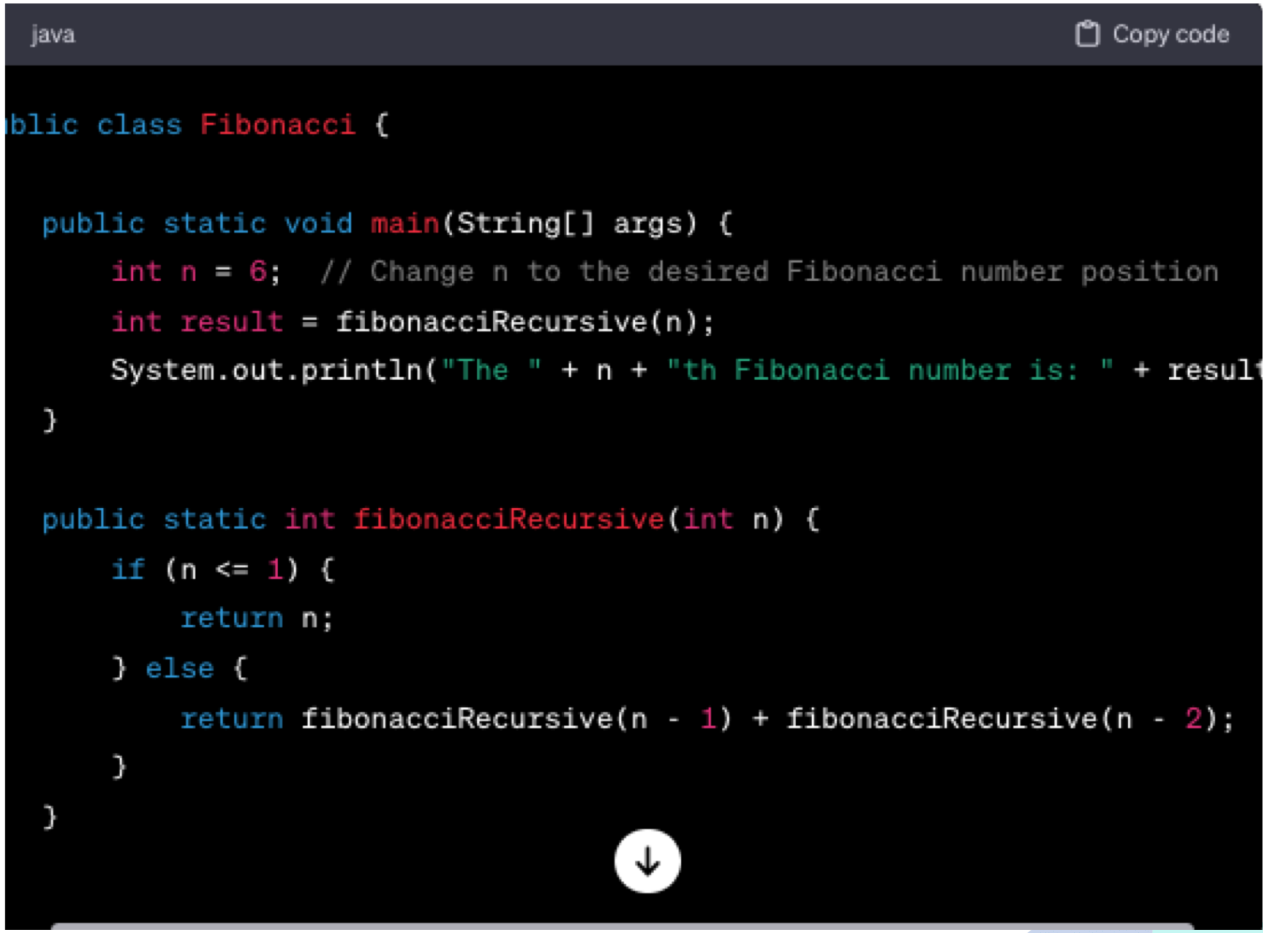
Main function



```
public static void main(String[] args) {
    System.out.println(faktorialRekursif(5));
    System.out.println(faktorialIteratif(5));
}
```

# When do we need to implement recursive function?

- ➢Solving difficult problems is done iteratively.
- ➢It does not consider memory-saving and program execution speed factors.
- ➢Let's consider a classic example → the Fibonacci sequence. The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones, usually starting with 0 and 1.



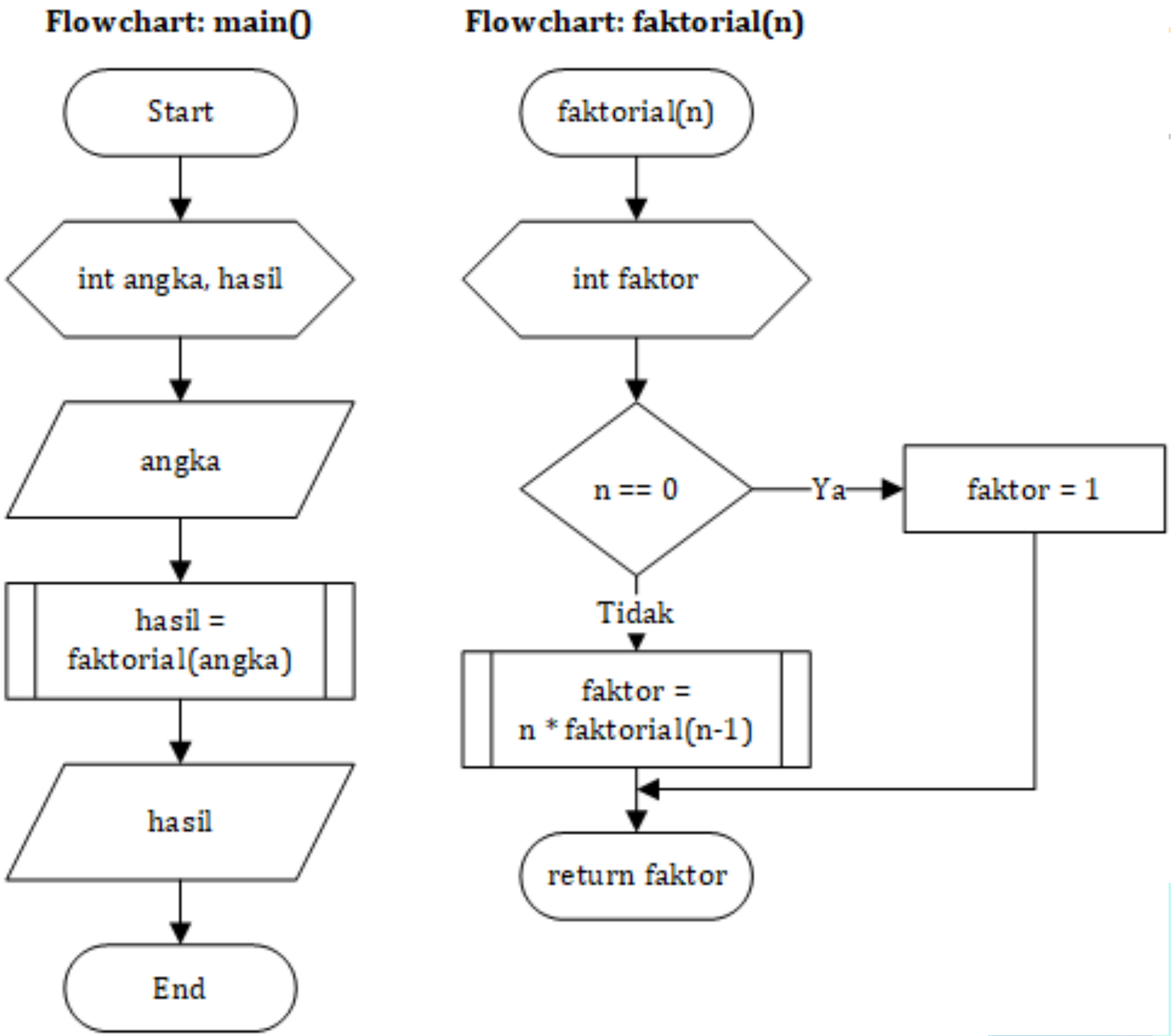
```java
blic class Fibonacci {

    public static void main(String[] args) {
        int n = 6;  // Change n to the desired Fibonacci number position
        int result = fibonacciRecursive(n);
        System.out.println("The " + n + "th Fibonacci number is: " + resul
    }

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

# When do we need to implement recursive function?

- The **fibonacciRecursive** function calculates the n$^{th}$ Fibonacci number using recursion.
- The base case is when n is 0 or 1, in which case the method returns n.
- The recursive case involves calling the **fibonacciRecursive** function for the two preceding Fibonacci numbers and summing them.

# Example #1 - Solution

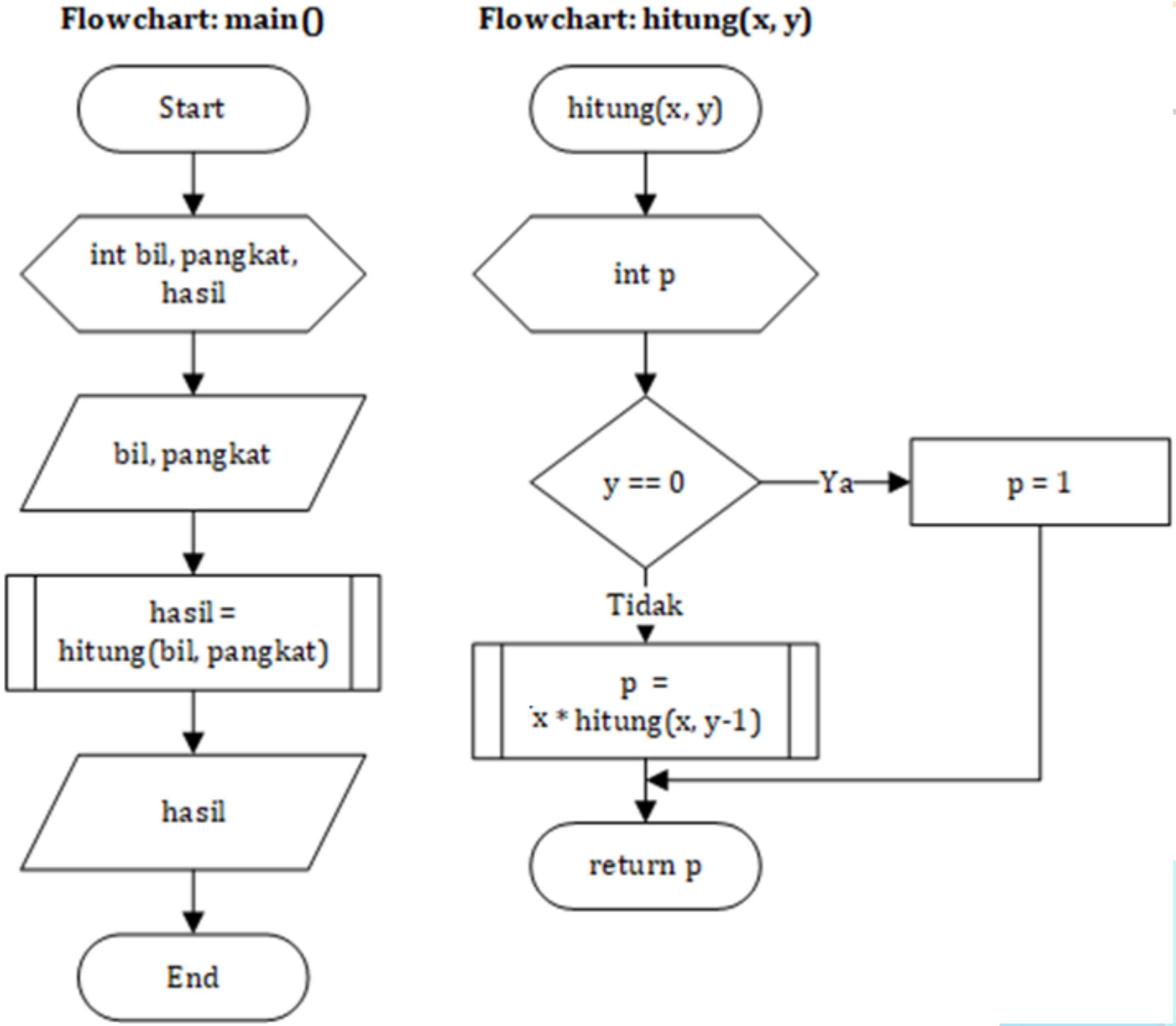
- CREATE A FLOWCHART TO CALCULATE FACTORIAL VALUE USING RECURSIVE FUNCTION



# Example #2

- There is a program to calculate the value of X power Y. As we know, the value of X power Y is calculated by multiplying X by itself (Y-1) times. However, if Y is 0 (X power 0), then the value is 1.
- Therefore, to calculate the value of X power Y, the program must impose a condition that if Y = 0, then the value of X becomes 1.
- Create the flowchart!

# Example #2 - Solution


Flowchart: main()

Start

int bil, pangkat, hasil

bil, pangkat

hasil = hitung(bil, pangkat)

hasil

End

Flowchart: hitung(x, y)

hitung(x, y)

int p

y == 0

Ya

p = 1

Tidak

p = x * hitung(x, y-1)

return p

# Individual assignment

1. Create a flowchart to calculate and print the total with input N:

   1 + 2 + 3 + 4 + 5 + … + … + N

   With the following approach:

   a) Iterative function

   b) Recursive function

2. Create flowchart to create Fibonanci series

   Pattern of fibonanci : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ….

   **a number is calculated by adding 2 preceedings number*

2. Calculate the investment return of an individual on purchasing gold bars. The investment profit from gold is 11.7% each year. Create a flowchart to determine the amount of money after a certain number (N) of years, for example, 10 years!

# Team-based assignment

1. Identify, according to each group's project, which features require the use of recursive functions.
2. Create an algorithm in the form of a flowchart according to the identified needs based on task number 1