Programming Fundamentals (Review)

Teaching Team of
Algorithm and Data Structure

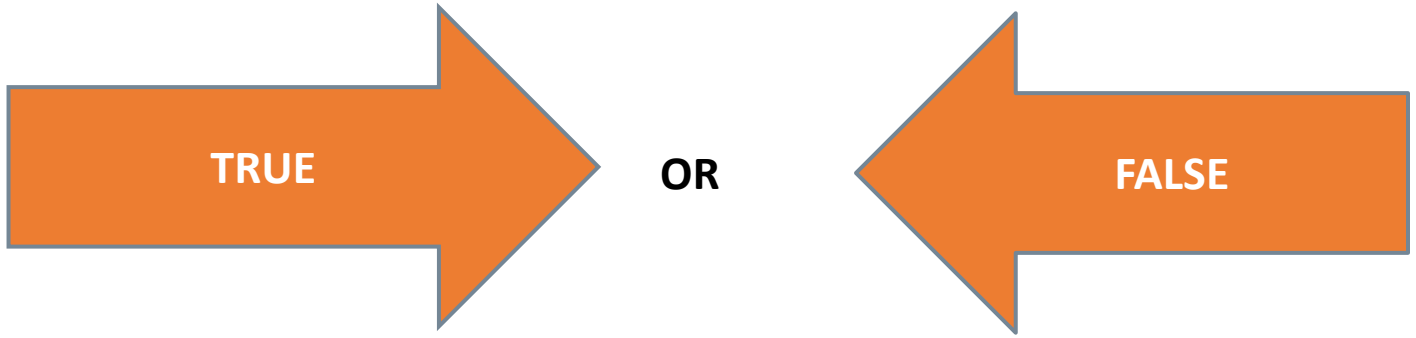
2023/2024

Selection

Definition

- Selection is an instruction used to choose one possibility from several conditions

Condition: a statement or expression (logical statement)



Selection Syntax IF

- General Form:

```
if (condition)
{
Statement
}
```

- ✓ if the condition is true, then the statement will be executed.
- ✓ if the condition is false, then the statement will not be executed.

Selection if...Else

- The IF-ELSE selection structure must have at least 2 statements. If the condition is TRUE or match, then Statement-1 will be executed. However, if the condition is FALSE, the statement 2 will be executed.
- General form:

```
if (condition)
{
Statement 1
}
else
{
Statement 2
}
```

Selection If...else if...else

- General Form:

```
if(condition 1)
{
Statement 1;
}
else if(condition 2)
{
Statement 2;
}
else if(condition 3)
{
Statement 3;
}
….
….
else if(condition X)
{
Statement X;
}
else
{
Statement;
}
```

In the form if ... else if ... else, statement 1 will run if "condition 1" is TRUE. If "condition 1" is FALSE, it will check "condition 2". If "condition 2" is TRUE then statement 2 will run, and so on. And if none of the above conditions are met or TRUE, then **statement** in **else** block will be executed

Example

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int purchase;

    System.out.println("Enter your total purchase ");
    purchase = sc.nextInt();

    if(purchase<=2000000){
        System.out.println("congratulations you got a gas stove gift");
    }
    else if(purchase<=1000000){
        System.out.println("congratulations you get the Teflon gift");
    }
    else if(purchase<=500000){
        System.out.println("congratulations you get the plate gift");
    }
    else {
        System.out.println("csorry you have not been lucky, increase your purchases");
    }
}
```

Selection SWITCH-CASE

- General Form:

```
switch (condition)
{
case constant -1:
        statement -1;
        break ;
case constant -2:
        statement -2;
        break;
...
...
case constant –x:
        statement –x;
        break –x;
default:
        statement;
}
```

This selection syntax will run one of several **"case"** statements in accordance with the condition values that are in the **"switch".** Then the process will continue until the statement **"break"** is found. However, if there is no value in the case that matches the condition value, then the process will proceed to the statement that is in **"default".**

Example

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int number;

    System.out.println("Enter your number ");
    number = sc.nextInt();

    switch(number){
        case 1:
            System.out.println("Monday");
            break;
        case 2:
            System.out.println("Tuesday");
            break;
        case 3:
            System.out.println("Wednesday");
            break;
        case 4:
            System.out.println("Thuraday");
            break;
        case 5:
            System.out.println("Friday");
            break;
        case 6:
            System.out.println("Saturday");
            break;
        case 7:
            System.out.println("Sunday");
            break;

        default :
            System.out.println("Sorry, the number you entered is incorrect");

    }
```

Ternary Operator

- Used in the selection syntax
- General Form:

```
syntax(Condition) ? (true condition) : (false condition)
```

Example

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double number = 5.5;

    String result;

    if(number >= 0)
        result = "Positive number ";
    else
        result = "Negative number ";

    System.out.println(number + " is " + result);
}
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double number = 5.5;

    String result;

    result = (number >= 0)?"Positive number" : "Negative number";

    System.out.println(number + " is " + result);
}
```

Nested Selection

- General Form:

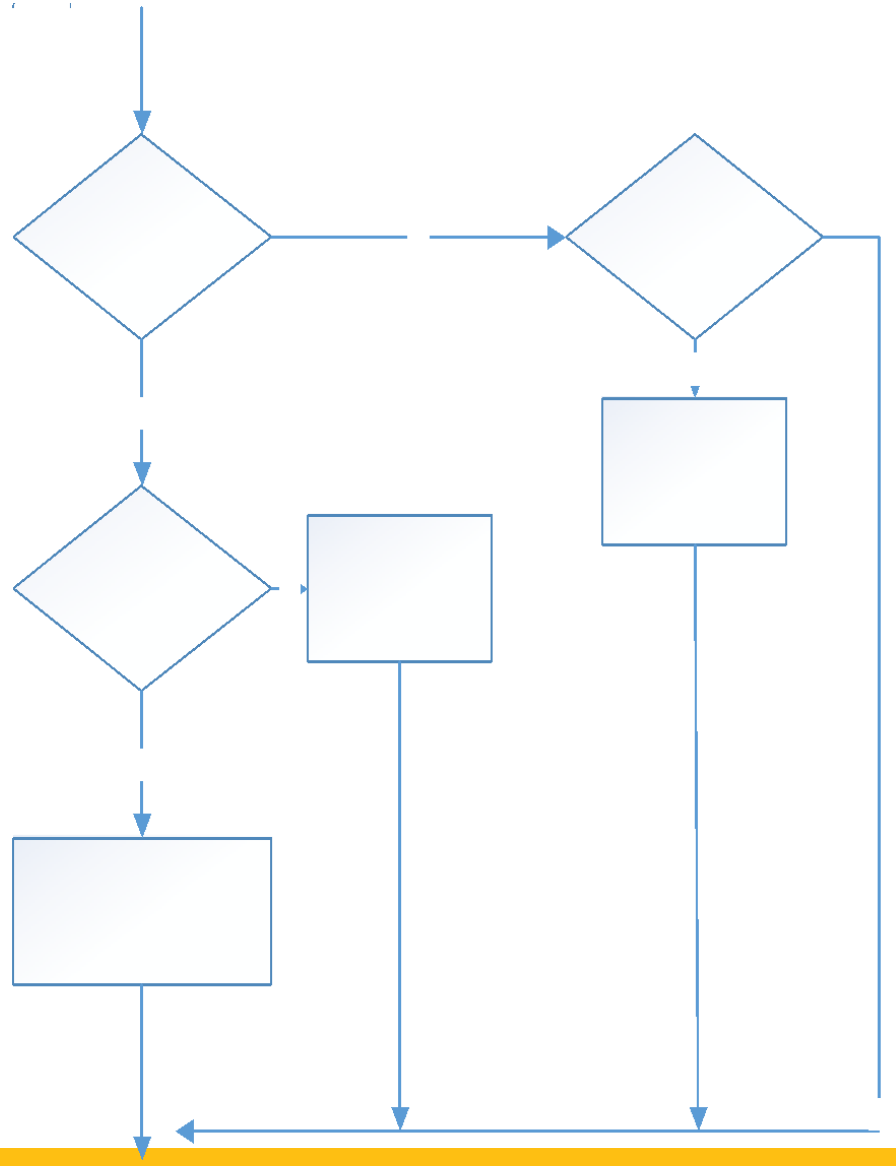
```
if (condition 1){
        if (condition 2){
                statement 1;
                …
                …
                if (condition n){

        statement 2;
                } else {
                        statement 3;
                }
        } else {
                statement n;
        }
} else {
        statement x;
}
```

- The first condition to be selected is the IF condition which is in the outermost position (condition 1).
- If condition 1 is false, then the outermost ELSE statement (pair of the relevant IF) will be processed. However, if the ELSE statement (pair of IF) is not written, then the condition selection will be stopped.
- If it turns out that condition 1 is true, then the next condition that is deeper (condition 2) will be selected. If condition 2 is false, then the ELSE statement (pair of the relevant IF) will be processed. However, if the ELSE statement (pair of IF) is not written, then the condition selection will be stopped.

Example

```java
import java.util.Scanner;
public class cashier {
    public static void main(String[] args) {
        int total, discount, pay;
        String card;
        Scanner sc = new Scanner(System.in);
        System.out.println("The costumer has a card (y or n)? ");
        card = sc.nextLine();
        System.out.println("How much price of groceries? ");
        total = sc.nextInt();
        if(card.equals("y")){
            if(total > 500000){
                discount = 50000;
            }else{
                discount = 25000;
            }
        }
        else{
            if(total>200000){
                discount = 10000;
            }
            else{
                discount = 0;
            }
        }
        pay = total - discount;
        System.out.println("The total price to be paid, is : " + pay);
    }
}
```

Flowchart Example



LOOPING

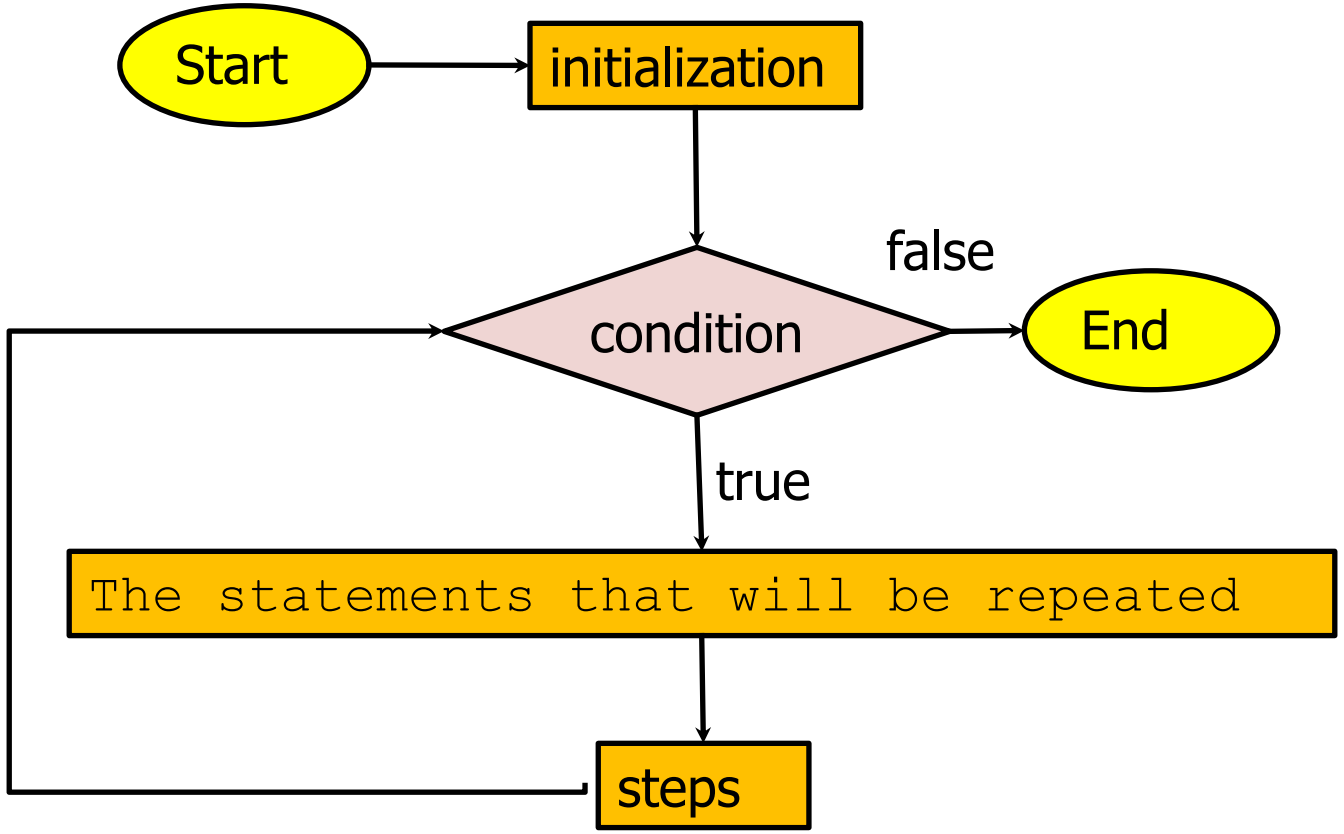
Command : for

- Syntax for command “for()”:

```
for(value inisializasion; Terms of recurrence; Change in value)
{
      … // The commandments that will be repeated
}
```

- **Value initialization** is where we will provide the initial value to the variable counter (variables used to calculate the number of loops).
- The recurrence requirement is a **condition** that must be met in order to keep the loop.
- **Change in value (steps)** is a change that will be made in each round to ensure that the loop will not last continuously.

Workflow for

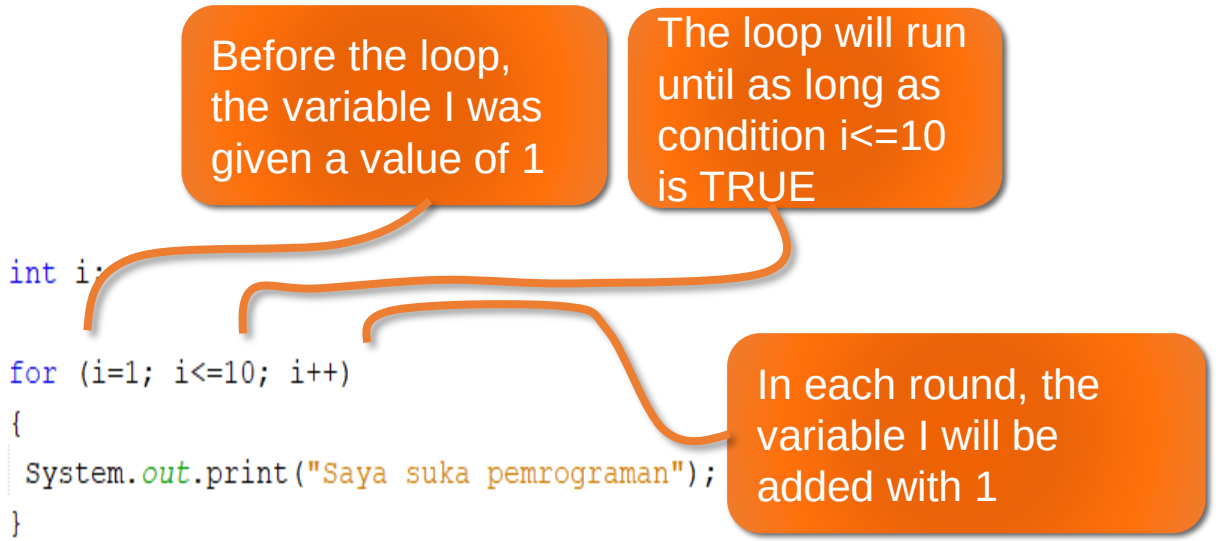


Command : for

- Command for() commonly used to perform task with a certain number of times. We already know how much the iteration will run
- Example: I love programming as much as 10 times

Output

```
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
Saya suka pemrograman
BUILD SUCCESSFUL (total time: 0 seconds)
```



Command while()

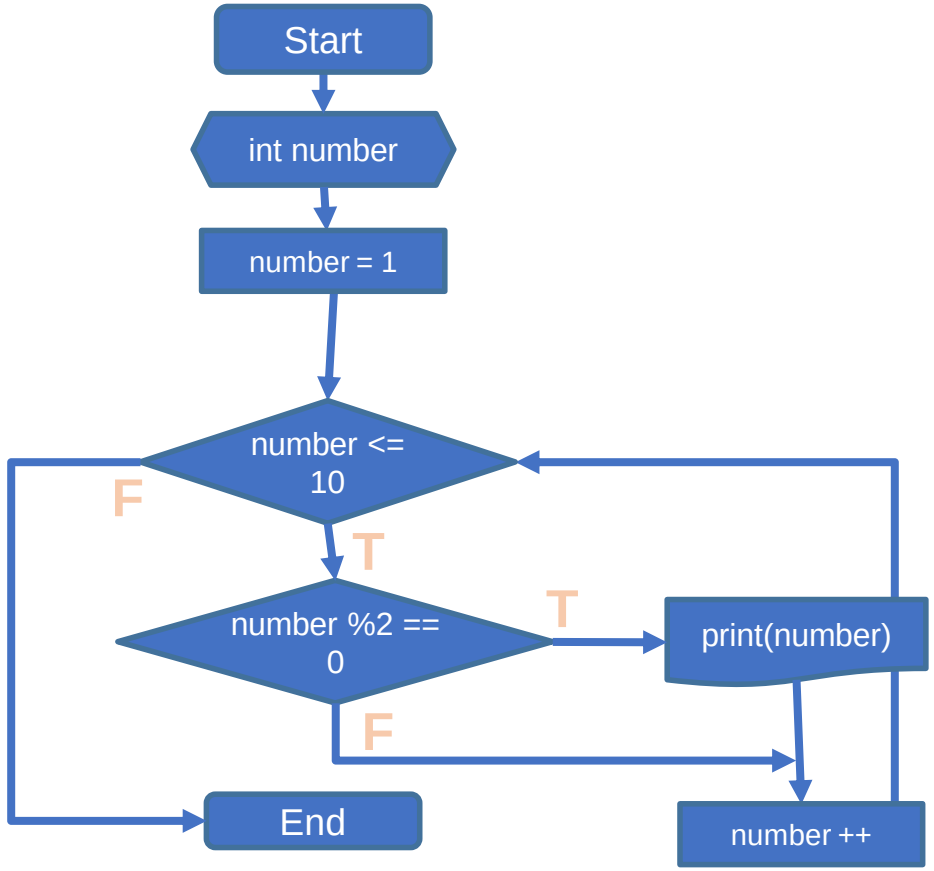
- Syntax of the while :

```
while(condition)
{
      … // The commandments that will be repeated
}
```

- While loop will continue to run as long as the condition is TRUE

Example of while

- Create a program to print **even** numbers from 1 to 10




```
public static void main(String[] args) {
    int angka;
    angka = 1;

    while (angka <= 10)
    {
        if (angka%2==0)
        {
            System.out.println (angka);
        }
    angka ++;
    }
}
```

```
run:
2
4
6
8
10
BUILD SUCCESSFUL
```

do-while

- Syntax do-while():

```
do
{
        … // statements
}
while(condtion);
```

- In principle, the Do-while is the same as the while.
- The Do-while repeats its statement as long as the condition is TRUE
- It is similar to the "while" loop, but with one key difference: in a "do-while" loop, the condition is evaluated after the loop body has been executed at least once
- Therefore, the Do-while at least will execute the statement once even though the condition does not match.

example

```
class ContohDoWhile {
    public static void main (String [] args)
  {
      int i = 6;
      do {
          System.out.println("Java");
          i++;
      } while (i<5);
    }
}
```

Nested Loop

For

```
for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; i++) {
                        // statement
                }
        }
        for (int l = 0; l < n; l++) {
                // statement
        }
}
```

Nested Loop

While

```
int i = 0; // index I declaration
// outer loop condition boundary
while (i < n) {
      int j = 0; // index j declaration
      // innerloop condition boundary
      while (j < n) {
            // statement
            j++
      }
      i++;
}
```

Nested Loop

Do-While

```
int i = 0; // index i declaration
do {
        int j = 0; // index j declaration
        do {
                // statement
                j++;
        } while (j < n);
        i++;
} while (i < n);
```

ARRAY

One-Dimensional Array

- Declaration

```
type namaArrray[];
Atau
type[] namaArray;
      Example: int a[]; int[] a
```

- **Type** is the data type of the array to be created.
- **namaArray** is the name of the array to be created.

One-Dimensional Array

- The Declaration and instantiation of the array object can be combined in an instruction as follows:

```
type[] namaArrray = new type[element_number];
      atau
type namaArrray[] = new type[element_number];
```

- Example :

```
int[] a = new int[10];
or
int a[] = new int[10];
```

Accessing Array Elements

- How to access elements from an array variable is done by referring to index number.

```
nameArray[index]
```

- Example:
  - It is desirable to access an array variable named with index i, then it can be written:

```
a[i]
```

  - Index i can only be 0 or positive with the maximum value is: (element_number-1).

Access Array Elements

- **Example :**

```
String[] cars = {“Avanza", "Brio", “Inova”};
System.out.println(cars[0]); //Displays Avanza
System.out.println(cars[2]); // Displays Inova
```

Example

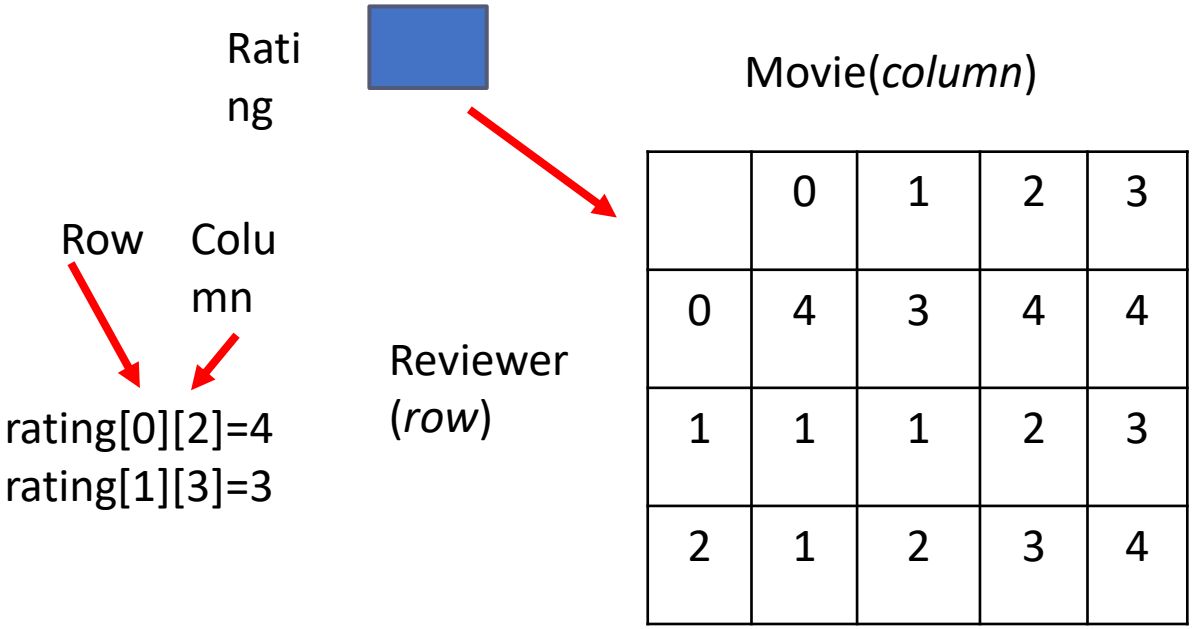
- Summation of ARRAYS

```
public class sampleArray5 {
    public static void main(String[] args) {
        int array[] = {33,4,5,23,1,5,6};
        int total = 0;

        for (int i=0; i<array.length;i++)
        {
            total +=array[i];
        }//menambahkan setiap nilai dari elemen array ke total
        System.out.println(total);
    }
}
```

```
run-single:
77
BUILD SUCCESSFUL
```

2-Dimensional Array (2)

- A 2-dimensional array is an array of index numbers using 2 numbers, one for rows and one for columns
- Example



	0	1	2	3
0	4	3	4	4
1	1	1	2	3
2	1	2	3	4

Declaring 2D Array (1)

- To declare a 2D array variable, the same as a 1D array. Only different from the number of square brackets "[]"
- General form :

```
data_type[][] array_name = new data_type[x][y];
x = number of rows
Y = number of columns

Example
int[][] arr = new int[10][20];
```

Calculates the average of 2D Array

```java
int[][] ratings = new int[3][4];
Scanner scanner = new Scanner(System.in);
int total = 0;
for (int i = 0; i < ratings.length; i++) {
    for (int j = 0; j < ratings[0].length; j++) {
        System.out.print("Masukan array[" + i + "," + j + "]: ");
        ratings[i][j] = scanner.nextInt();
        total += ratings[i][j];
    }
}

for (int rate[] : ratings) {
    for (int i : rate) {
        System.out.print(i + " ");
    }
    System.out.println();
}
float rerata = (float) total / (ratings.length * ratings[0].length);
System.out.println("Nilai total : " + total);
System.out.println("Nilai rerata: " + rerata);
```

```
Masukan array[0,0]: 1
Masukan array[0,1]: 2
Masukan array[0,2]: 3
Masukan array[0,3]: 4
Masukan array[1,0]: 2
Masukan array[1,1]: 3
Masukan array[1,2]: 4
Masukan array[1,3]: 4
Masukan array[2,0]: 3
Masukan array[2,1]: 2
Masukan array[2,2]: 3
Masukan array[2,3]: 4
1 2 3 4
2 3 4 4
3 2 3 4
Nilai total : 35
Nilai rerata: 2.9166667
```

The average value is obtained from the sum of all matrix element values divided by the product of the size of the row and column arrays

FUNCTIONS

DECLARATION OF FUNCTIONS

```
static ReturnDataType functionName() {
// statement
// statement
}
```

- **Static** : Types functions created are static, in order to directly call in the main function which also are static
- **ReturnDataType**: Data type from the returned value (*output*) After the function is executed
- **functionName ()**: function name be made

example:

Function Declaration:

```
static void beriSalam(){
    System.out.println("Halo! Selamat Pagi");
}
```

Calling Function:

```
public static void main(String[] args){
    beriSalam();
}
```

Function that returns the value ... (1)

- A function can return an output value so that it can be processed in the next process.
- Return the value of the function using *keyword* **return**,
- Functions that have **non-void** data type will require a **return**. The **void** function **does not require a return**.
- **The value returned** from a function **must match the function's data type**. For example, if the data type is **int**, then the value that is returned by the function must be an **int** value

SCOPE OF VARIABLE … (1)

- Local Variables: variabel declared in a function, and can only be accessed or identified from within the function itself.

- Global variables: variables declared outside the block function, and can be accessed or identified from any function.
- Global variables in java in the given prefix **static** so that these variables can be called directly.

Function that returns a value... (2)

- **Example**

Making the function with parameters and return value:

```
static int luasPersegi(int sisi){
    int luas = sisi * sisi;
    return luas;
}
```

Dialing functions and give the parameter value:

```
System.out.println("Luas Persegi dengan sisi 5 = " + luasPersegi(5));

int luasan = luasPersegi(6);
```

SCOPE OF VARIABLE ... (2)

```
public class Fungsi1 {

    static int a = 10, b = 5;
    static double c;

    static int Kali() {
        int hasilKali = a * b;
        return hasilKali;

    }

    static void Tambah() {
        int hasilTambah = a + b;
        System.out.println("Hasil Tambah adalah " + hasilTambah);
    }

    public static void main(String[] args) {
        System.out.println("Hasil Kali adalah " + Kali());
        Tambah();

    }

}
```

Global variables

Local variables

Local variables

Recursive Function

- Usually a function will be called (CALL) by another function
- In a recursive function, in a function there is a command to call the function itself (itself). Thus, the process of calling a function will occur repeatedly
- General Form:

```
Return_value_data_type function_name (data_type
parameter_name){
    ...
    Function_name(...)
    ...
}
```

Recursive Function Format

- In general, the recursive function format has the following form

```
if (nilai batas)
        //menyelesaikan masalah
else
        //mendefinisikan kembali masalah
menggunakan rekursi
```

- IF branch is **base case**, while ELSE is **recursion call**
- The recursion call section provides repetition needed to simplify the problem and the base case provides termination
- In order to stop recursion, recursion call should approach the base case in any recursive function calls

Example1

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 - Trace

n * faktorialRekursif(n-1)

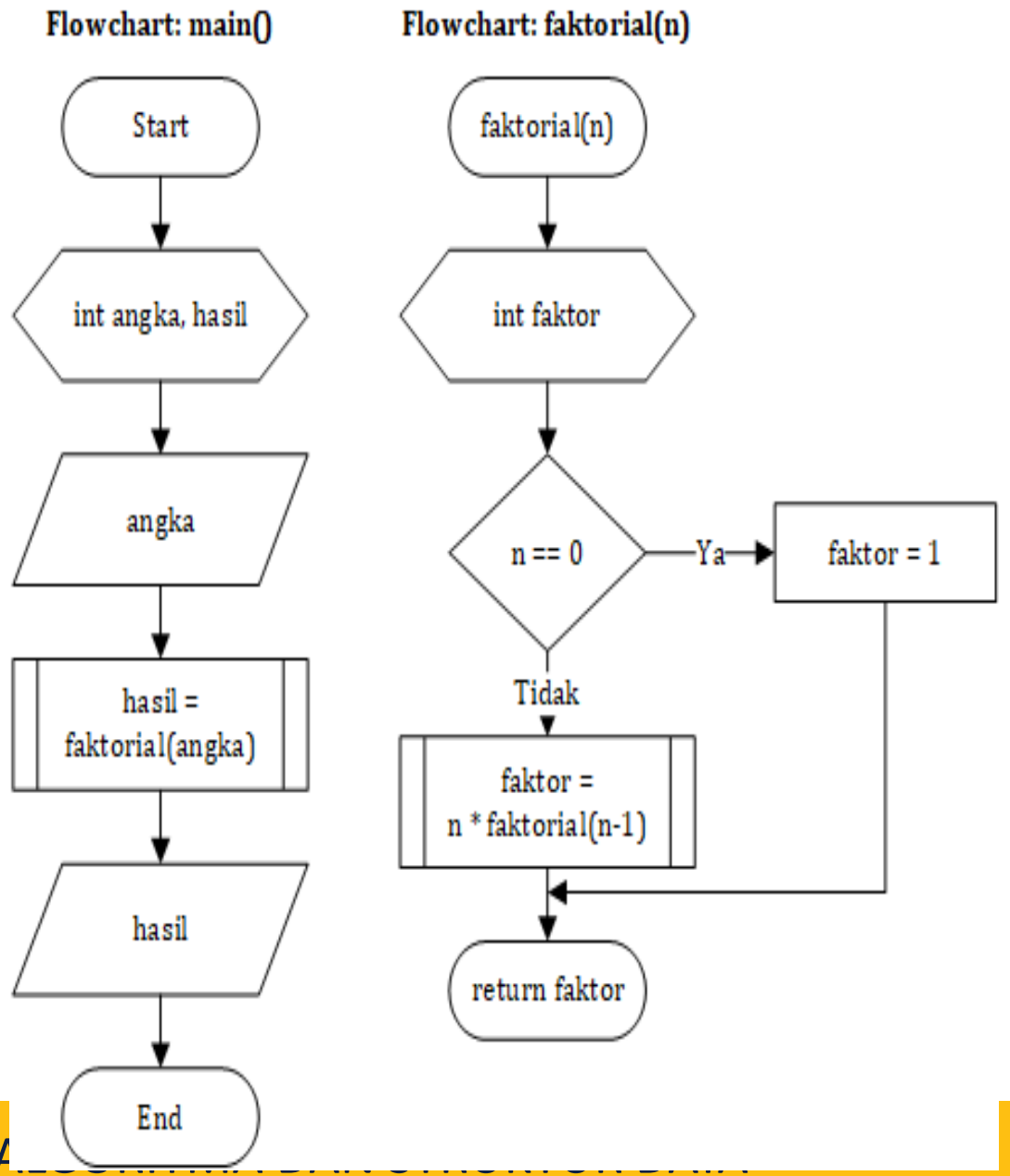
faktorialRekursif(5) = 5 * faktorialRekursif(4)
= 5 * (4 * faktorialRekursif(3))
= 5 * (4 * (3 * faktorialRekursif(2)))
= 5 * (4 * (3 * (2 * faktorialRekursif(1))))
= 5 * (4 * (3 * (2 * (1 * faktorialRekursif(0)))))

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 1 - Factorial



Thanks…
Questions???

PRACTICE

1. Make a flowchart to convert decimal numbers to binary
3. There are three numbers x, y, and z. Make a flowchart to get the largest value among the three numbers.