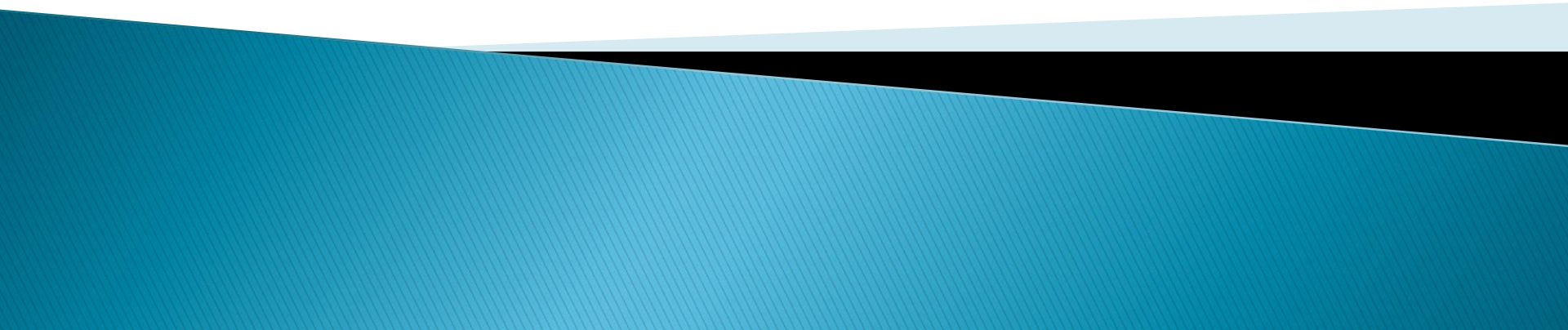


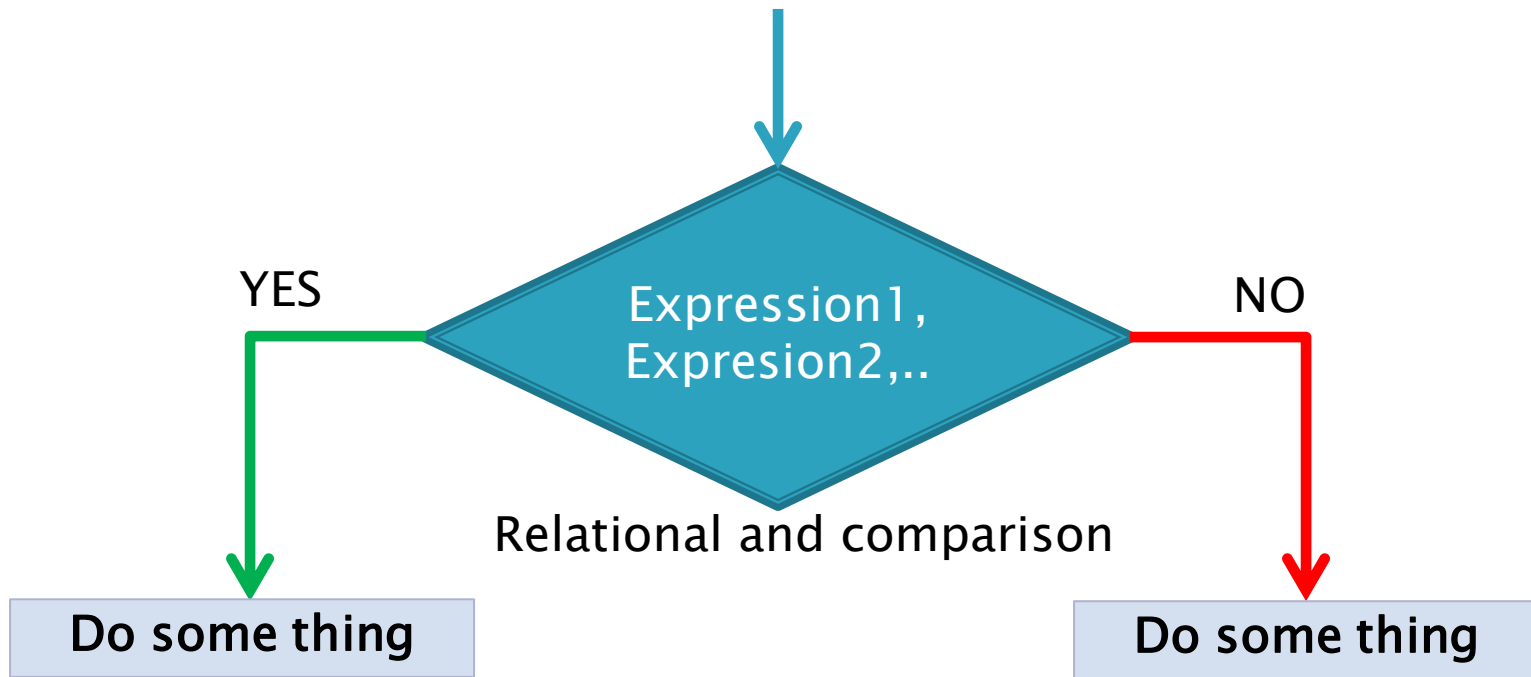
Introduction to Java week#3

10/05/2023



week	Topic	Calendar
1	JAVA IDE (NetBean) Installation ,Configuration and Compile	3 - 7 April 2023
2	Basic structure of Java ,Data & Variable type, operator & basic logic	17 - 21 April 2023
3	Function(Method) create & calling, Input & output	20 - 24 April 2023
4	Loop statement ,Array variable	27 - 31 April 2023
5	Object-oriented programming (OOP),Class & Object, Encapsulation	1 - 5 May 2023
6	Inheritance, Polymorphism, Interfaces	8 - 12 May 2023
7	Packages, Access Modifiers(Public ,Protected ,Private class)	15 - 19 May 2023
8	Collections (Array list, HashMap, Stack)	22 - 26 May 2023
9	Exception	29 May - 2 June 2023
10	Working with files(Read, Write)	5 - 9 June 2023
11	Thread Programming	12 - 16 June 2023

Basic Logic



Basic Logic

Relational and comparison operators

- ▶ ใช้สำหรับเปรียบเทียบข้อมูล โดยการสร้าง **expression** ขึ้นมา ค่าที่ได้จากการเปรียบเทียบจากตัวดำเนินการเหล่านี้จะเป็น **Boolean Value (true or false)**

ตารางตัวดำเนินการเปรียบเทียบในภาษา **Java**

Operater	Example	Result
==	a == b	true if `a` equal to `b`, otherwise false
!=	a != b	true if `a` not equal to `b`, otherwise false
<	a < b	true if `a` less than `b`, otherwise false
>	a > b	true if `a` greater than `b`, otherwise false
<=	a <= b	true if `a` less than or equal to `b`, otherwise false
>=	a >= b	true if `a` greater than or equal to `b`, otherwise false

Basic Logic

Logical operators

- ▶ เป็นตัวดำเนินการที่ใช้สำหรับ **expression** ตั้งแต่หนึ่ง **expression** ขึ้นไป

ตารางตัวดำเนินการตรรกศาสตร์ในภาษา **Java**

Operator	Name	Example
!	Not	! true
&&	And	true && true
	Or	true false

Basic Logic

Logical operators

ตารางค่าความจริงของตัวดำเนินการตรรกศาสตร์ Not

Value 1	Expression	Result
true	! true	false
false	!false	true

ตัวดำเนินการทางตรรกศาสตร์ Not จะกลับค่าจาก true เป็น false และในทางกลับกัน

Basic Logic

Logical operators

ตารางค่าความจริงของตัวดำเนินการตรรกศาสตร์ And

Value 1	Value 2	Expression	Result
true	true	true && true	true
true	false	true && false	false
false	false	false && false	false

ตัวดำเนินการทางตรรกศาสตร์ And ถ้า expression ย่อยทั้งสองเป็น true จะได้ผลลัพธ์เป็น true นอกเหนือจากนั้นจะได้ผลลัพธ์เป็น false

Basic Logic

Logical operators

ตารางค่าความจริงของตัวดำเนินการตรรกศาสตร์ Or

Value 1	Value 2	Expression	Result
true	true	true true	true
true	false	true false	true
false	false	false false	false

ตัวดำเนินการทางตรรกศาสตร์ Or ถ้า expression ย่อยอย่างน้อยหนึ่งตัวเป็น true จะได้ผลลัพธ์เป็น true นอกเหนือจากนั้นจะได้ผลลัพธ์เป็น false

Basic Logic

Logical operators

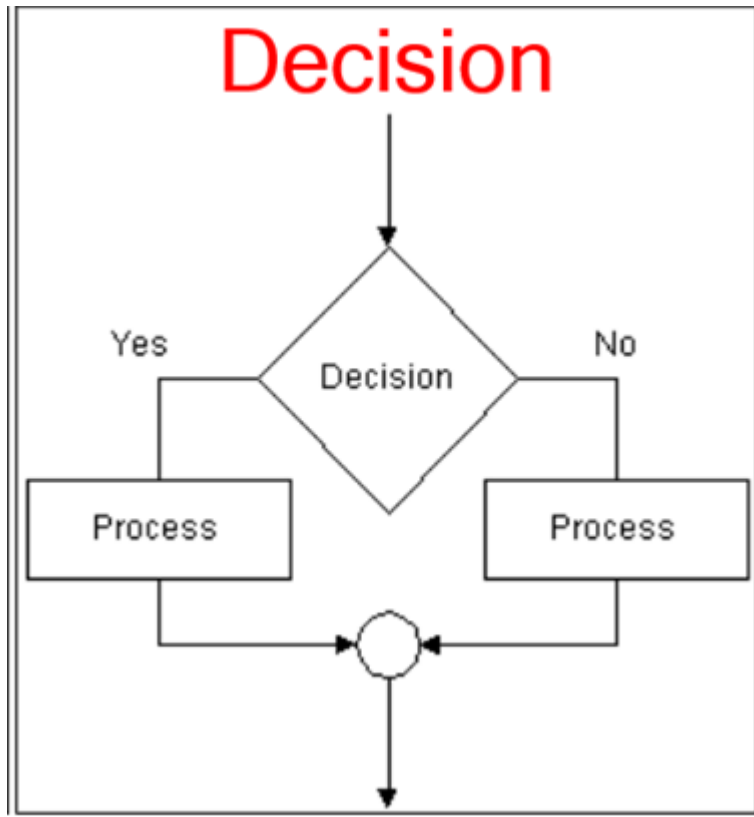
```
int score = 8;  
int level = 4;
```

```
boolean1 = (score >= 10 && level >= 5)  
boolean2 = (score >= 10 || level >= 3)
```

```
boolean1 = false  
boolean2 = true
```

Basic Logic

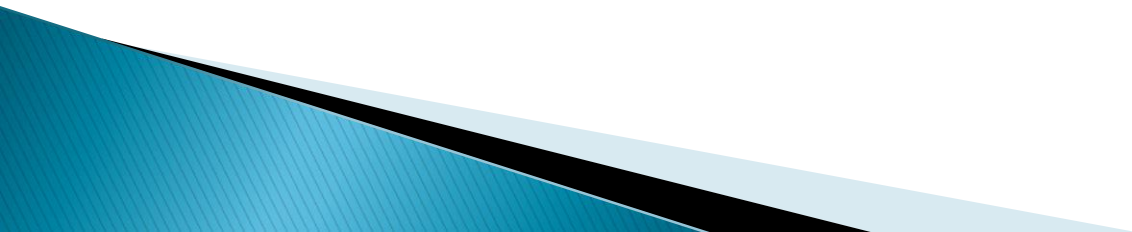
Decision Making



คำสั่งตัดสินใจ เป็นประโยค
คำสั่งที่ใช้ควบคุมให้
โปรแกรมเลือกดำเนินไปใน
เส้นทางใดเส้นทางหนึ่ง โดย
ขึ้นอยู่กับผลการตรวจสอบ
เงื่อนไข

Decision Making

Decision Making

- ▶ คำสั่ง **if**
 - ▶ คำสั่ง **if else**
 - ▶ คำสั่ง **else-if**
 - ▶ คำสั่ง **switch**
 - ▶ **Ternary Operator**
- 

Decision Making

คำสั่ง **If**

```
if (expression) {  
    statements  
}
```

Decision Making

คำสั่ง If

```
public class IfStatement {  
    public static void main(String[] args) {  
        String username = "mateo";  
        String password = "1234";  
  
        if (username == "mateo") {  
            System.out.println("Your username has a permission.");  
        }  
  
        if (username == "mateo" && password == "1234") {  
            System.out.println("You're now logged in.");  
        }  
    }  
}
```

Decision Making

คำสั่ง **If else**

```
if (expression) {  
    statements  
}else{  
    statements  
}
```

Decision Making

คำสั่ง If else

```
public class IfElseStatement {  
    public static void main(String[] args) {  
        String username = "mateo";  
        String password = "abcd";  
  
        if (username == "mateo" && password == "1234") {  
            System.out.println("You're now logged in.");  
        } else {  
            System.out.println("Sorry, your username or password is incorrect.");  
        }  
    }  
}
```

Decision Making

คำสั่ง **else-if**

```
if(expression-1) {  
    statements  
}  
else if(expression-2) {  
    statements  
}  
else {  
    statements  
}
```


Decision Making

```
if (score < 0 || score > 100) {  
    System.out.println("You must enter a correct score, try again later");  
} else {  
  
    if (score >= 80) {  
        System.out.println("Your score is excellent.");  
        System.out.println("You grant grade S.");  
    } else if (score >= 60) {  
        System.out.println("Your score is good.");  
        System.out.println("You grant grade A.");  
    } else if (score >= 40) {  
        System.out.println("Your score is fair.");  
        System.out.println("You grant grade B.");  
    } else {  
        System.out.println("Your score is poor.");  
        System.out.println("You grant grade C.");  
    }  
}  
}
```

Decision Making

คำสั่ง Switch

```
switch(variable) {  
    case val1 :  
        statement  
        break;  
    case val2 :  
        statement  
        break;  
  
    ..... •  
  
    default :  
        statement  
}
```

Decision Making

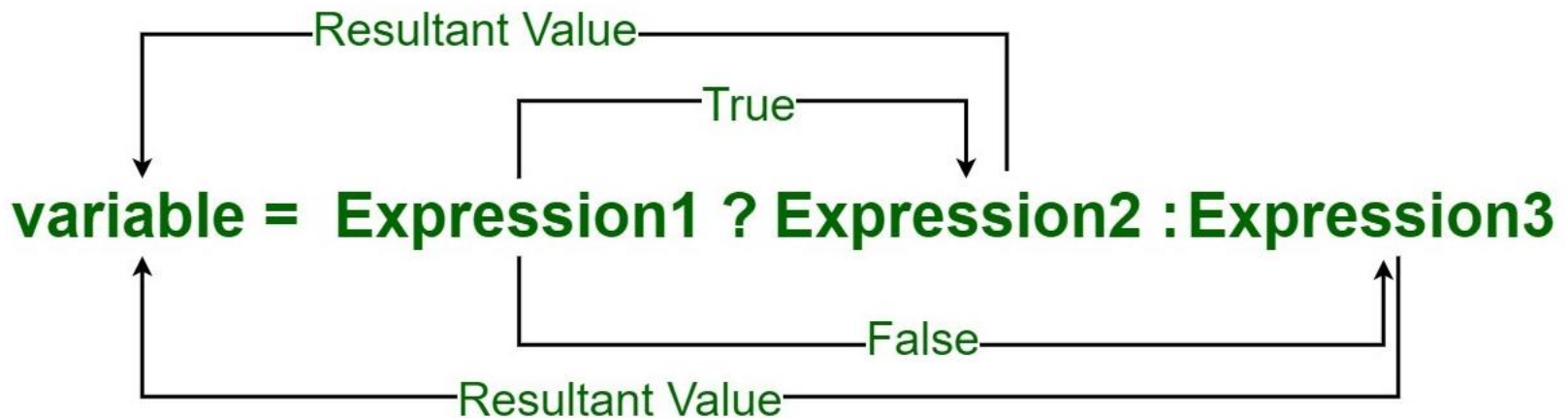
คำสั่ง Switch

```
char floor = 'G' ;
switch (floor) {
    case 'G' :
        System.out.println("Elevator is going to ground floor.");
        break;
    case '1' :
        System.out.println("Elevator is going to first floor.");
        break;
    case '2' :
        System.out.println("Elevator is going to second floor.");
        break;
    case '3' :
        System.out.println("Elevator is going to third floor.");
        break;
    default:
        System.out.println("Elevator don't know where to go.");
}
```

Decision Making

Ternary Operator

Conditional or Ternary Operator (?:) in Java

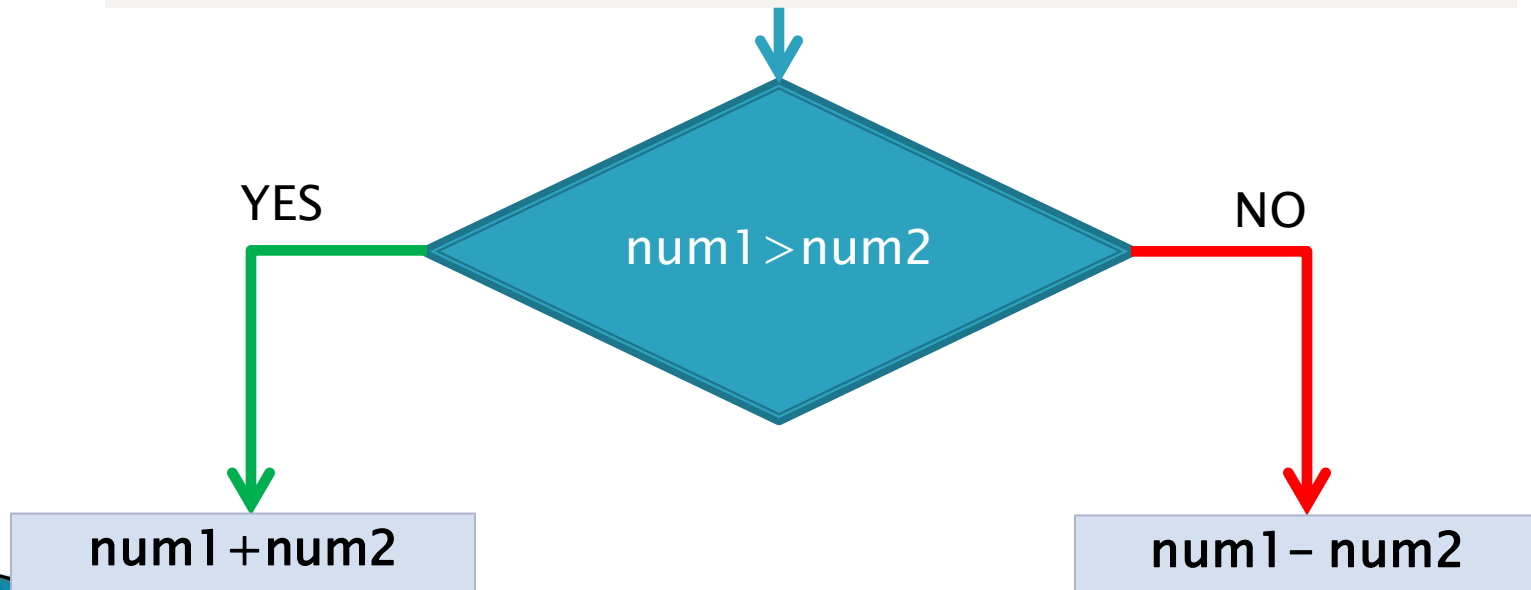


Decision Making

Ternary Operator

```
num1 = 10;  
num2 = 20;
```

```
res = (num1 > num2) ? (num1 + num2) : (num1 - num2)
```



Function (Method)

เป็นกลุ่มของคำสั่งที่สร้างขึ้นมาเพื่อทำหน้าที่บางอย่าง
การสร้างเมธอดจะสามารถทำให้เราใช้ได้นั่นซ้ำๆ
โดยที่ไม่ต้องเขียนโปรแกรมใหม่

Function (Method)

รูปแบบในการสร้างเมธอดในภาษา Java เป็นดังนี้

```
access_modifier type name ( parameter1, parameter2, ... )  
{  
.....do something .....  
}
```

- ▶ **type** เป็นประเภทของเมธอดที่จะสร้างขึ้น โดยสามารถเป็นได้ทั้ง **primitive data type** หรือ **reference types** ได้ และถ้าหากเมธอดไม่มีการส่งค่ากลับจะใช้คำสั่ง **void**
- ▶ **name** เป็นชื่อของเมธอดหรือ **identifier** ซึ่งมีกฎในการตั้งชื่อเช่นเดียวกันกับตัวแปร
- ▶ **parameters** เป็นลิสต์ของตัวแปรที่จะส่งเข้าไปใช้ในเมธอด โดยสามารถมีมากกว่า 1 หรือไม่มีก็ได้
- ▶ **access_modifier** เป็นการกำหนดระดับการเข้าถึงของเมธอด ซึ่งจะอยู่ในเรื่องของออบเจ็ค โดยมี 4 แบบ คือ **private protected public** และ **default** (ไม่ต้องกำหนด)

Function (Method)

```
private static void sayWelcome () {  
    System.out.println("Welcome to Calculator Program");  
}  
  
private static int sum (int a, int b) {  
    return a + b;  
}  
}
```


Function (Method)

```
public class CreateMethod {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Calculator Program");  
  
        int x = 2;  
        int y = 3;  
        System.out.println("x + y = " + (x + y));  
        System.out.println("10 + 20 = " + (10 + 20));  
    }  
}
```

Function (Method)

```
public class CreateMethod {  
    public static void main(String[] args) {  
        sayWelcome();  
  
        int x = 2;  
        int y = 3;  
        System.out.println("x + y = " + sum(x, y));  
        System.out.println("10 + 20 = " + sum(10, 20));  
  
    }  
  
    private static void sayWelcome () {  
        System.out.println("Welcome to Calculator Program");  
    }  
  
    private static int sum (int a, int b) {  
        return a + b;  
    }  
}
```

Function (Method)

```
public class CreateMethod {  
    public static void main(String[] args) {  
        sayWelcome();  
  
        int x = 2;  
        int y = 3;  
        System.out.println("x + y = " + sum(x, y));  
        System.out.println("10 + 20 = " + sum(10, 20));  
    }  
  
    private static void sayWelcome () {  
        System.out.println("Welcome to Calculator Program");  
    }  
  
    private static int sum (int a, int b) {  
        return a + b;  
    }  
}
```

The diagram illustrates the execution flow of the code. Red arrows originate from the following lines and point to the corresponding method definitions:

- An arrow from `sayWelcome();` in the `main` method points to the `sayWelcome ()` method definition.
- An arrow from `sum(x, y)` in the first `println` statement points to the `sum (int a, int b)` method definition.
- An arrow from `sum(10, 20)` in the second `println` statement points to the `sum (int a, int b)` method definition.

Function (Method)

```
public class CreateMethod {  
    public static void main(String[] args) {  
        sayWelcome();  
  
        int x = 2;  
        int y = 3;  
        System.out.println("x + y = " + sum(x, y));  
        System.out.println("10 + 20 = " + sum(10, 20));  
    }  
}
```

Welcome to Calculator Program

x + y = 5

10 + 20 = 30

```
        return a + b;
```

```
    }
```

```
}
```

Function (Method)

Return keyword

เมธอดยังสามารถส่งค่ากลับไปยังที่ที่มันถูกเรียกได้ โดยการใช้คำสั่ง **return** คำสั่งสุดท้ายในเมธอด การ **return** หมายถึงการสิ้นสุดการทำงานของเมธอดและโปรแกรมจะไปทำงานต่อที่ที่มันถูกเรียก

```
public static float getPI ()  
{  
    return 3.14f;  
}
```

Function (Method)

Return keyword

เมธอดยังสามารถส่งค่ากลับไปยังที่ที่มันถูกเรียกได้ โดยการใช้คำสั่ง **return** คำสั่งสุดท้ายในเมธอด การ **return** หมายถึงการสิ้นสุดการทำงานของเมธอดและโปรแกรมจะไปทำงานต่อที่ที่มันถูกเรียก

```
public static float getPI ()  
{  
    return 3.14f;  
}
```

Function (Method)

Return keyword

```
public static float getPI ()  
{  
    return 3.14f;  
}
```

```
public class MethodReturn {  
    public static void main(String[] args) {  
  
        float pi = getPI();  
        System.out.println("Value of PI is " + pi);  
  
    }  
}
```

Value of PI is 3.14

Function (Method)

Overloading Method

เป็นความสามารถหนึ่งในภาษา **Java** ที่สามารถสร้างเมธอดโดยใช้ชื่อเดียวกันได้ แต่สิ่งที่แตกต่างกันคือจำนวนพารามิเตอร์ของมัน ในการทำงานโปรแกรมจะทำถ้าหากประเภทของพารามิเตอร์และลำดับของมันตรงกัน

```
public static void hello (int n)
public static void hello (String name)
public static void hello (String name, String name2
```


Function (Method)

Overloading Method

```
public static void main(String[] args) {  
  
    hello(3);  
    hello("Marcus");  
    hello("Alice", "Emma");  
  
}
```

```
public static void hello (int n)  
public static void hello (String name)  
public static void hello (String name, String name2
```

Function (Method)

Recursive Method

เป็นการที่เมธอดมีการเรียกใช้งานเมธอดตัวเอง จึงจำเป็นต้องสร้างจุด **break point** ไม่เช่นนั้นจะเกิด การทำงานที่ไม่รู้จบ

```
public static void callMyself(long i) {  
    if (i < 0)  
    {  
        return;  
    }  
    System.out.print(i);  
  
    i = i - 1;  
  
    callMyself(i);  
}
```

► Assignments

ให้นำ **assignment** คราวที่แล้ว มาสร้างเป็น **function** ให้เหมาะสม

Thank you

