



# បណ្ណបណ្ណាលក្នុង សម្បត្តិ អេច អ ឌី Korea Software HRD Center

## ស្វែងយល់ពី Method ក្នុង Java Java Method

ណែនាំដោយ : Dr. Kim Tae Kyung



<http://www.kshrd.com.kh>

୭. Structure of method

୮. Return type of method

୯. Access modifier

୧୦. Parameter

## ១. Structure of Method

- អ្វីទៅជា **Method** ?
  - **Method** ជាបណ្តុំនៃ **statements** ដែលប្រមូលផ្តុំគ្នា ហើយបង្កើតឡើងសម្រាប់ដោះស្រាយបញ្ហាអ្វីមួយ ដដែលៗ។ មានន័យថា Method បង្កើតតែម្តង អាចហៅថាទៅប្រើបានច្រើនដង និង ច្រើនទិសដៅ។
  - ហើយក្នុងភាសា C / C++ គេហៅថា **function** និង ភាសាខ្លះទៀតហៅថា **procedure**។
- អត្ថប្រយោជន៍នៃការប្រើ Method :
  - ងាយស្រួលកែប្រែកូដ និងស្វែងរកកំហុស
  - ចំណេញពេលវេលា
  - ងាយស្រួលមើលកូដ មិនរញ្ជើរច្រើន
  - ការបន្ថយការសរសេរកូដដែរ។

## 9. Structure of Method

- ការប្រៀបធៀបរវាងការប្រើ method និងមិនប្រើ method

### Don't use method

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 10; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 40; i <= 50; i++)
    sum += i;
System.out.println("Sum from 40 to 50 is " + sum);
```

### Use method

```
public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 30 is " + sum(20, 30));
    System.out.println("Sum from 40 to 50 is " + sum(40, 50));
}

public static int sum(int start, int end) {
    int sum = 0;
    for (int i = start; i <= end; i++)
        sum += i;

    return sum;
}
```

## ១. Structure of Method

- Method មាន **Header** និង **Body** ។
  - **Method Header** ជាផ្នែកសំរាប់ប្រកាស សិទ្ធិសំរាប់អោយប្រើ, ប្រភេទតំលៃ, ឈ្មោះ និងតំលៃសំរាប់ផ្តល់អោយ Method ។
  - **Method Body** ជាផ្នែកសំរាប់សរសេរកូដ ។
- ទំរង់ក្នុងការបង្កើត method

```
[modifier] returnType/void nameOfMethod ([Parameter List]) {  
    // method body  
}
```

## ១. Structure of Method

```
[modifier] returnType/void nameOfMethod ([Parameter List]) {  
    // method body  
}
```

- **modifier** : គឺជាការកំណត់នូវប្រភេទនៃ access type ដូចជា **public**, **private**, **protected**, **default** របស់ method។
- **returnType**: method អាច return តំលៃចេញវិញ។
- **nameOfMethod**: គឺជាឈ្មោះរបស់ method។ ហើយចាប់ផ្តើមដោយអក្សរតូច និងជា **camelCase**។



## 9. Structure of Method

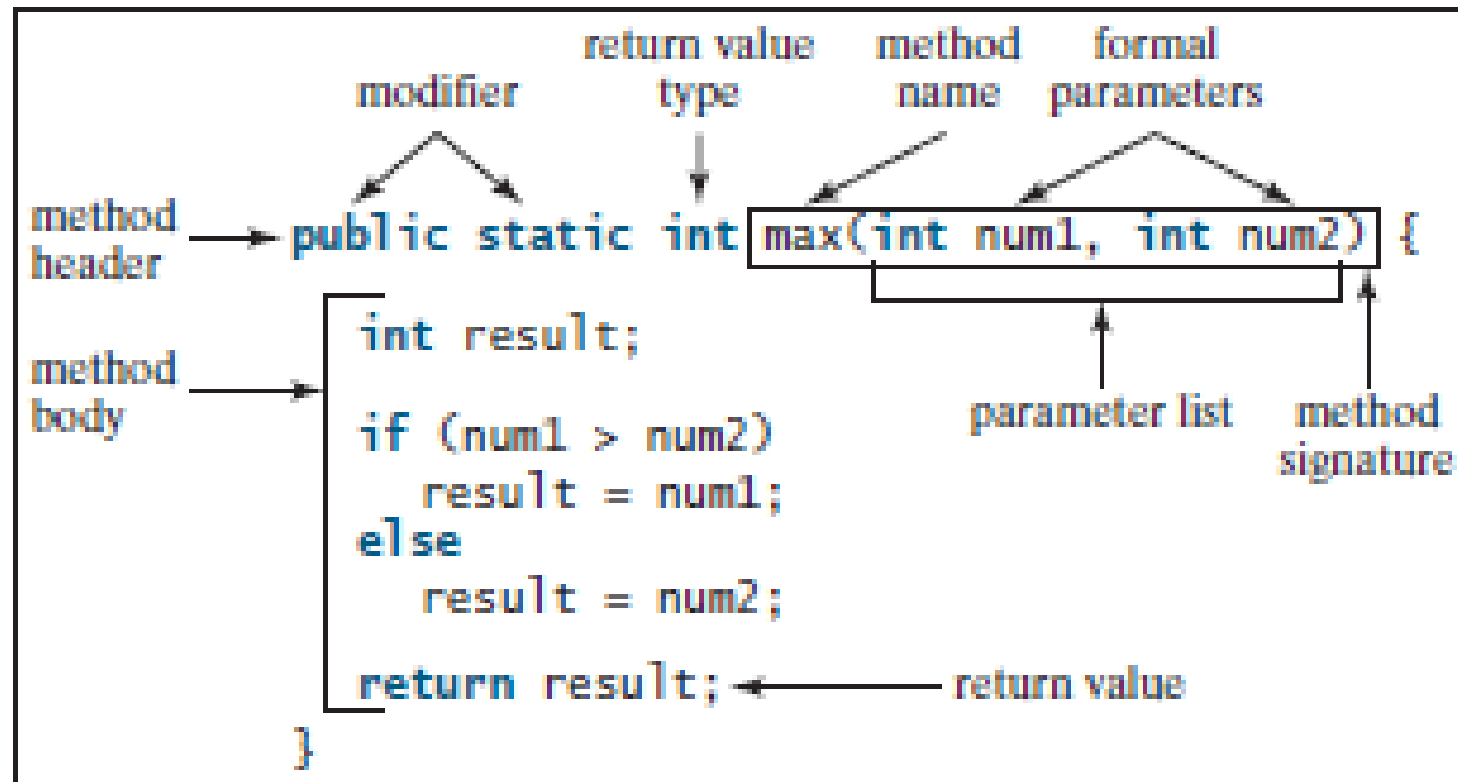
```
[modifier] returnType/void nameOfMethod ([Parameter List]) {  
    // method body  
}
```

- **Parameter List:** គឺជាប៉ារ៉ាម៉ែត្រ(តម្លៃ) ដែលបោះអោយ method នៅពេលដែល method ត្រូវបានហៅ មកប្រើ ។ Method អាចមាន parameters និងអត់មាន parameters។
- **method body:** គឺជាកន្លែងដែលសរសេរនូវ statement ដែលអាចអោយ method ធ្វើការងារអ្វីមួយ។

## 9. Structure of Method

ឧទាហរណ៍៖

Define a method



Invoke a method

```
int z = max(x, y);
```

↑ ↑  
actual parameters  
(arguments)



## ១. Structure of Method

- ការដាក់ឈ្មោះអោយ method នៅក្នុង Java៖
  - ឈ្មោះរបស់ method គួរតែជា **កិរិយាសព្ទ** ហើយចាប់ផ្តើមដោយអក្សរតូច។
  - ចំពោះ method ដែលមានច្រើនពាក្យគឺប្រើអក្សរតូចនៅខាងដើមហើយអក្សរធំនៅខាងដើមពាក្យនីមួយៗ (**camelCase**)។

Ex. run

runFast

getName

setName

## ២. Type of Method

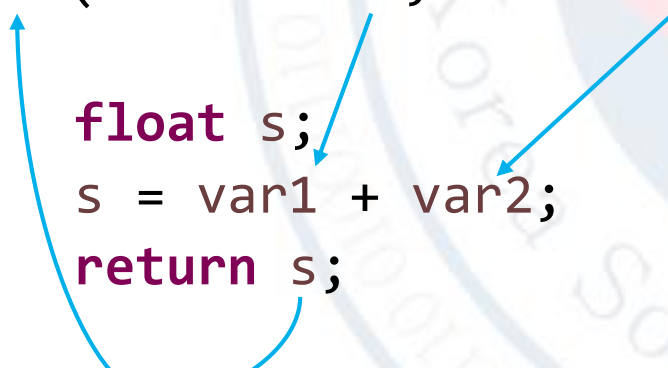
- នៅក្នុងភាសា Java បានបែងចែក Method ជាពីរប្រភេទគឺ return type method និង non-return type method។
  - Return type method គឺប្រើសំរាប់បោះតម្លៃទៅអោយ Method វិញនៅពេលដំនើរការចប់តាមរយៈ **return keyword** ។

```
[modifier] returnType nameOfMethod ([Parameter List]) {  
    statement (s);  
    return value;  
}
```

## ២. Type of Method


- return** ជា keyword មួយប្រើសម្រាប់បញ្ចប់ method និងបញ្ជូនតម្លៃដែលនៅបន្ទាប់ពីវាទៅឱ្យ method ។ តម្លៃដែលផ្តល់ឱ្យ method ត្រូវមានប្រភេទ ទិន្នន័យដូចទៅនឹង DataType របស់ method នោះ។

```
float sum(float var1, float var2)
{
    float s;
    s = var1 + var2;
    return s;
}
```



The diagram illustrates the execution of the first method. A blue arrow points from the expression 'var1 + var2' in the assignment statement 's = var1 + var2;' to the variable 's'. Another blue arrow points from 's' in the 'return s;' statement back to the 'float' return type in the method signature 'float sum(...)'. A third blue arrow points from the '+' operator in the assignment statement to the 'float' return type, indicating the data type of the result.

```
float sum(float var1, float var2)
{
    return var1 + var2;
}
```



The diagram illustrates the execution of the second method. Two blue arrows point from the variables 'var1' and 'var2' in the expression 'var1 + var2' directly to the 'float' return type in the method signature 'float sum(...)'. A third blue arrow points from the '+' operator in the expression to the 'float' return type, indicating the data type of the result.

## ២. Type of Method

- ឧទាហរណ៍ :

```
public class MyMethod{  
    public static void main(String[] args){  
        int number = numberOne();  
        System.out.println(number);  
    }  
    public static int numberOne(){  
        return 1;  
    }  
}
```

របៀបហៅ method

របៀបបង្កើត method

Output: 1

## ២. Type of Method

- Non-return type method គឺជា Method ដែលមិនបោះតំលៃអ្វីទៅអោយ method វិញទេនៅពេលដែលដំនើរការចប់ ។ Method នេះមិនមាន return keyword ឡើយ ។

```
[modifier] void nameOfMethod ([Parameter List]) {  
    statement (s);  
}
```

**void** ជា keyword មួយប្រើសម្រាប់ជំនួសឱ្យ **dataType** នៅពេលដែលប្រើ non-return type method។

## ២. Type of Method

➤ ឧទាហរណ៍ :

```
public class MyMethod{  
    public static void main(String[] args){  
        printStatement();  
    }  
    public static void printStatement(){  
        System.out.println("This is my statement!");  
    }  
}
```

រៀបចំ method

រៀបចំ method

**Output:** This is my statement!



## ២. Type of Method

### ចំណាំ៖

- **Return Type Method:** Return-type និង Value ដែលត្រូវ Return ត្រូវតែជាប្រភេទតែមួយ។
- **Non-Return Type Method :** គឺជា void method នោះមិនចាំបាច់ return អ្វីទាំងអស់។

## ២. Type of Method

ឧទាហរណ៍ Method ដែល Return Object៖

```
public Student getNewStudent(){  
    String name="Sok";  
    int age=20;  
    return new Student(name, age);  
}
```

## ២. Type of Method

ឧទាហរណ៍ Method ដែល Return Object៖

```
class Test {  
    public static void main(String args[]) {  
        Rectangle ob1 = new Rectangle(40, 50);  
        Rectangle ob2;  
  
        ob2 = ob1.getRectangleObject();  
        System.out.println("ob1.length : " + ob1.length);  
        System.out.println("ob1.breadth: " + ob1.breadth);  
  
        System.out.println("ob2.length : " + ob2.length);  
        System.out.println("ob2.breadth: " + ob2.breadth);  
    }  
}
```

```
class Rectangle {  
    int length;  
    int breadth;  
  
    Rectangle(int l, int b) {  
        length = l;  
        breadth = b;  
    }  
  
    Rectangle getRectangleObject() {  
        Rectangle rect = new Rectangle(10, 20);  
        return rect;  
    }  
}
```

**Output:** ob1.length : 40  
ob1.breadth: 50  
ob2.length : 10  
ob2.breadth: 20

## ២. Type of Method

### Recursive Methods:

- គឺជា Method ដែលហៅខ្លួនវា។
- ពេលវាហៅខ្លួនវា គឺវាដោះស្រាយបញ្ហាឲ្យបានតែតូចៗ។
- ពេលដែលបញ្ហាចូលដល់ចំណុចតូចបំផុតវានឹងឈប់ហៅខ្លួនវា។

```
public class Test {  
    public static void main(String[] args) {  
        int theAnswer = triangle(12);  
        System.out.println("Triangle=" + theAnswer);  
    }  
  
    public static int triangle(int n) {  
        if (n == 1)  
            return 1;  
        return (n + triangle(n - 1));  
    }  
}
```

**Output:** Triangle=78

## ៣. Modifier

- **Modifiers** គឺជា Keywords ដែលត្រូវបានប្រើសំរាប់កំណត់ទៅអោយ **class**, **method** ឬ **variable**។
- គេចែក modifiers ជា ២ ប្រភេទគឺ៖
  - **Access Modifiers**
  - **Non Access Modifiers**

## ៣. Access Modifier

- **Access Modifiers:** គឺជាប្រភេទ keyword ដែលប្រើសំរាប់កំណត់កំរិតប្រភេទនៃការ access របស់ **classes, methods, variables,** និង **constructor**។
- Access Modifier នៅក្នុង Java មាន ៤ ប្រភេទ៖
  - **Public** : អាច Access បានទៅគ្រប់ class, field, constructor, និង method។
  - **Protected** : អាច Access បាននៅក្នុង class, package, និង subclass។
  - **Default** : អាច Access បាននៅក្នុង class, package របស់ខ្លួន។
  - **Private** : អាច Access បានតែក្នុង class របស់ខ្លួនប៉ុណ្ណោះ។



## ៣. Access Modifier

- សូមពិនិត្យមើលតារាងដែលបង្ហាញអំពី **Access Modifier** ដូចខាងក្រោម៖

Access Modifiers	Class	package	subclass	world
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
default	Yes	Yes	No	No
private	Yes	No	No	No

**ចំណាំ៖** ចំពោះ class, method, constructor ឬ variable ណាមិនប្រើប្រាស់ Access Modifier ទេ វានឹងចាប់យក **Default** ដោយស្វ័យប្រវត្តិ។

## ៣. Access Modifier

### ឧទាហរណ៍នៃការប្រកាស Access Modifiers

```
public class AccessModifier{
```

```
    public void publicMethod(){// public accessModifier }
```

public modifier

```
    protected void protectedMethod(){// protectedaccessModifier }
```

protected modifier

```
    void defaultMethod(){// default accessModifier }
```

default modifier

```
    private void privateMethod(){// private accessModifier }
```

private modifier

```
}
```

## ៣. Access Modifier

### ចំណាំ

- Method ដែលប្រកាសជា **Public** នៅក្នុង Supper Class ត្រូវតែជា **Public** នៅក្នុង Subclass ដែរ។
- Method ដែលប្រកាសជា **Protected** នៅក្នុង Supper Class អាចជា **Public** ឬ **Protected** នៅក្នុង Subclass ។
- Method ដែលជា **Default** នៅក្នុង Supper Class អាចជា **Public** ឬ **Protected** ឬ **Default** នៅក្នុង Subclass ។
- Method ដែលប្រកាសជា **Private** គ្មានការ **inherit** ទេ។

## ៣. Non Access Modifier

- **Non Access Modifiers:** គឺជាប្រភេទ keyword ដែលប្រើសំរាប់បន្ថែមមុខងារមួយចំនួនទៅអោយ classes, methods ឬ variables ។
- **Non Access Modifiers** នៅក្នុង Java programming មានដូចជា៖
  - Static modifier
  - Synchronized
  - Final modifier
  - Volatile
  - Abstract modifier
  - Transient

## ៤. Parameter

- Parameter ជា **Local variable** ដែលបង្កើតនៅក្នុង expression របស់ Method ។ វាមានតួនាទី រង់ចាំទទួលតំលៃពីខាងក្រៅបោះមកអោយ ហើយតំលៃនោះត្រូវបានយកទៅអនុវត្តនៅក្នុង Method តាមរយៈ argument នៅពេល method ត្រូវបានហៅទៅប្រើប្រាស់។

```
int Sum( int a, int b ){  
    return a + b;  
}
```

## ៤. Parameter

- **ឈ្មោះរបស់ parameter** មិនអាចប្រកាសជាន់គ្នាពីរដង ក្នុង **scope** តែមួយទេ។ ប៉ុន្តែវាអាចមានឈ្មោះដូចគ្នាទៅនឹងឈ្មោះរបស់ **field** ក្នុង class។

```
private int width, height;
```

```
public int rectangle(int width, int height)
{
    this.width = width;
    this.height = height;
    return this.width * this.height;
}
```



## ៤. Parameter

### ➤ ឧទាហរណ៍:

```
public class MyMethod{  
    public static void main(String[] args){  
        printSomething("Hi");  
    }  
    public static void printSomething (String sth) {  
        System.out.println(sth);  
    }  
}
```

**Output:** Hi

រៀបរៀង method

រៀបបង្កើត method

## ៤. Parameter

➤ ឧទាហរណ៍:

```
public class MyMethod{  
    public static void main(String[] args){  
        int sum2Number = sum(1,1);  
        System.out.println(sum2Number);  
    }  
    public static int sum(int a, int b) {  
        return a + b;  
    }  
}
```

**Output:** 2

រៀបរៀង method

រៀបបង្កើត method

## ៤. Parameter

- Java មិន Support Default Value ទេ ខុសប្លែកពី Programming ផ្សេងៗ ។

```
int sum(int a, int b=0)
{
    return a+b;
}
```

**Error**

## ៤. Parameter

- **Over Loading Method** ជា method ដែលមានឈ្មោះដូចគ្នាហើយ វាចាំបាច់ត្រូវមានលក្ខណៈខុសគ្នាដូចជា **return type** របស់ method, **DataType** របស់ parameter, ចំនួន **parameter** ជាដើម។

ឧទាហរណ៍:

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
int sum(int a, int b, int c) {  
    return a + b + c;  
}
```

```
double sum(int a, double b) {  
    return a + b;  
}
```

```
double sum(double a, int b) {  
    return a + b;  
}
```

## ៤. Parameter

- **Variable Argument (var-args)** គឺជា Parameter ពិសេសដែលអាចអោយរង់ចាំតំលៃដែលមាន **DataType** ដូចគ្នា ផ្តល់អោយវា។ តំលៃដែលបោះអោយនោះអាចមួយ ឬ ច្រើនតំលៃ។ វាស្រដៀងទៅនឹង **Array** ដែរ។

```
public static void main(String[] args) {  
    printMax(34, 3, 3, 2, 56.5);  
    printMax(new double[] { 1, 2, 3 });  
}
```

```
static void printMax(double... numbers) {  
    statement(s);  
}
```