



METHOD IN Java

METHOD

คือการรวมชุดคำสั่งเพื่อสามารถเรียกใช้งานซ้ำได้
เพื่อลดการเขียนโค้ดซ้ำในโปรแกรม
โดยในภาษา Java จะเขียน method ไว้ใน Class
อีกทั้ง method ยังสามารถบอกรถึงพฤติกรรมของ
Class นั้นได้ด้วย

METHOD

เช่น โปรแกรมเครื่องคิดเลข



- Method การบวก
- Method การลบ
- Method การคูณ
- Method การหาร



องค์ประกอบของ Method

```
static returnType methodName(parameter) {  
    methodBody  
    return ...  
}
```

returnType คือชนิดข้อมูลที่ต้องการส่งออกจาก method นี้ไปยังผู้ที่เรียกใช้
methodName คือชื่อของ method ที่เราต้องการจะตั้ง ส่วนกฎการตั้งชื่อเหมือนกันตัวแปร.
parameter คือ ส่วนที่บอกว่า method นี้ผู้ที่เรียกใช้สามารถใส่ค่าอะไรเข้ามาได้บ้าง
methodBody คือ ส่วนที่เราสามารถเขียนโค้ดต่างๆลงไปได้ตามที่เรต้องการ
return คือ เมื่อ method ทำงานเสร็จแล้ว สามารถส่งค่ากลับไปให้ผู้เรียกใช้ได้

ประโยชน์ของ Method

- ลดการเขียนโค้ดซ้ำซ้อน - เขียนครั้งเดียวใช้ได้หลายครั้ง
- ทำให้โค้ดเป็นระเบียบ - แบ่งเป็นส่วนๆ อ่านและเข้าใจง่าย
- แก้ไขง่าย - เมื่อต้องการแก้ไข ก็แก้ไขแค่ที่ method
- ทดสอบง่าย - สามารถทดสอบแยกทีละ method ได้
- ปลอดภัย - สามารถควบคุมการเข้าถึงข้อมูลได้

ตัวอย่าง Method

```
class Main {  
    public static void main(String[] args) {  
        int x = 2;  
        if (x % 2 == 0) {  
            System.out.println("Even");  
        } else {  
            System.out.println("Odd");  
        }  
  
        int y = 3;  
        if (y % 2 == 0) {  
            System.out.println("Even");  
        } else {  
            System.out.println("Odd");  
        }  
  
        int z = 5;  
        if (z % 2 == 0) {  
            System.out.println("Even");  
        } else {  
            System.out.println("Odd");  
        }  
    }  
}
```



```
class Main {  
    public static void main(String[] args) {  
        int x = 2;  
        evenOrOdd(x);  
        int y = 3;  
        evenOrOdd(y);  
        int z = 5;  
        evenOrOdd(z);  
    }  
  
    static void evenOrOdd(int num) {  
        if (num % 2 == 0) {  
            System.out.println("Even");  
        } else {  
            System.out.println("Odd");  
        }  
    }  
}
```


คำศัพท์น่ารู้

```
class Main {  
    public static void main(String[] args) {  
        int x = 2;  
        evenOrOdd(x);  
        int y = 3;  
        evenOrOdd(y);  
        int z = 5;  
        evenOrOdd(z);  
    }  
}
```

Caller

```
static void evenOrOdd(int num) {  
    if (x % 2 == 0) {  
        System.out.println("Even");  
    } else {  
        System.out.println("Odd");  
    }  
}
```

Callee

Caller หมายถึง method ที่เรียกใช้ method อื่น

Callee หมายถึง method ที่ถูกเรียกใช้จาก method อื่น หรือ Caller
อธิบาย :

ใน method main มีการเรียกใช้ method ที่มีชื่อว่า evenOrOdd ดังนั้น method main จะถูกเรียกว่า Caller และ method ที่ชื่อ evenOrOdd จะถูกเรียกว่า Callee เพราะ ถูก method main หรือ Caller เรียกใช้

การสร้าง Method แบบที่ 1

```
static void myFunction() {  
    System.out.println("Hello World!");  
}
```

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```

method ที่มี return type เป็น void
และ ไม่รับ parameter
อธิบาย:

การที่ไม่มี return type หรือ return type เป็น void นั้นหมายความว่า เมื่อ caller เรียกใช้ method นี้แล้ว callee จะไม่มีค่าอะไรส่งออกไปให้กับ caller

การที่ไม่รับ parameter นั้นหมายความว่า method นี้ไม่ได้รับค่าอะไรเข้ามาให้ตัวเอง

ลองพิมพ์ Method แบบที่ 1

```
public class Main {  
    public static void main(String[] args) {  
        myFunction();  
        doSomething();  
    }  
  
    static void myFunction() {  
        System.out.println("Hello World");  
    }  
  
    static void doSomething() {  
        int x = 2;  
        char y = 'A';  
        System.out.println(x+y);  
    }  
}
```

Visualize Method แบบที่ 1

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

```
static void myFunction() {  
    System.out.println("Hello World");  
}
```

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```

Visualize Method แบบที่ 1

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

```
static void myFunction() {  
    System.out.println("Hello World");  
}
```

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```


Visualize Method แบบที่ 1

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

Caller

Callee

```
static void myFunction() {  
    System.out.println("Hello World");  
}
```

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```

Visualize Method แบบที่ 1

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

Caller

```
static void myFunction() {  
    System.out.println("Hello World");  
}
```

Callee

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```

Visualize Method แบบที่ 1

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

```
static void myFunction() {  
    System.out.println("Hello World");  
}
```

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```


Visualize Method แบบที่ 1

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

Caller

```
static void myFunction() {  
    System.out.println("Hello World");  
}
```

Callee

```
static void doSomething() {  
    int x = 2;  
    char y = 'A';  
    System.out.println(x+y);  
}
```

การสร้าง Method แบบที่ 2

```
static void add(int a, int b) {  
    System.out.println("sum: " + a + b);  
}
```

```
static void evenOrOdd(int num) {  
    if (num % 2 == 0) {  
        System.out.println("Even");  
    } else {  
        System.out.println("Odd");  
    }  
}
```

method ที่มี return type เป็น void
แต่ รับ parameter

อธิบาย :

method ที่รับ parameter สามารถรับ
ค่าอะไรก็ได้เข้ามาใน method

โดยที่ caller สามารถใส่ argument
เข้ามาตามที่ callee สร้าง parameter

ข้อดี :

ทำให้ การสร้าง method ของเรามี
ความยืดหยุ่นมากขึ้น

การใช้งาน Method ทำ2

Source:

```
public class Main {  
    public static void main(String[] args) {  
        add(3, 9);  
    }  
  
    static void add(int a, int b) {  
        System.out.println("sum: " + a + b);  
    }  
}
```

Output:

sum: 39

อธิบาย :

argument ที่ caller ใส่เข้ามาใน method add() ต้องตรงกับ **parameter** ที่ method add() กำหนดไว้

ทำไมผลลัพธ์ถึงเป็น "sum: 39":

argument ที่เราใส่เข้ามาคือ 3 และ 9 ตามลำดับ เมื่อใส่เข้ามาแล้ว **argument** ก็จะวิ่งไปที่ **callee** ตามลำดับ แล้วพิมพ์ตัวแปรนั้นออกมา ตามคำสั่ง **println**

Visualize Method แบบที่ 2

```
public static void main(String[] args) {  
    add(3, 9);  
}
```

```
static void add(int a, int b) {  
    System.out.println("sum: " + a + b);  
}
```

Visualize Method แบบที่ 2

```
public static void main(String[] args) {  
    add(3, 9);  
}
```

```
static void add(int a, int b) {  
    System.out.println("sum: " + a + b);  
}
```

Visualize Method แบบที่ 2

```
public static void main(String[] args) {  
    add(3, 9);  
}
```

caller

```
static void add(int a, int b) {  
    System.out.println("sum: " + a + b);  
}
```

callee

Visualize Method แบบที่ 2

```
public static void main(String[] args) {  
    add(3, 9);  
}
```

caller

```
static void add(int a, int b) {  
    System.out.println("sum: " + a + b);  
}
```

callee

Visualize Method แบบที่ 2

```
public static void main(String[] args) {  
    add(3, 9);  
}
```

caller

```
static void add(int a, int b) {  
    System.out.println("sum: " + a + b);  
}
```

callee

การสร้าง Method แบบที่ 3

```
static void notThingReturn() {  
    return;  
}
```

```
static int returnWithInt() {  
    return 17;  
}
```

```
static String returnWithString() {  
    return "Remy";  
}
```

```
static boolean returnWithBoolean() {  
    return true;  
}
```

method ที่มี return ทุกชนิด

อธิบาย :

method สามารถมี **return type** ได้
ทุกชนิด ทุกชนิดในที่นี้ครอบคลุมทั้ง
Primitive Data Types และ Reference
Data Types

เมื่อมีการ return type ออกไป return
type จะถูกส่งไปให้กับ caller

การใช้งาน Method ทำ3

Source:

```
public class Main {  
    public static void main(String[] args) {  
        String name = returnWithString();  
        System.out.println(name);  
    }  
  
    static String returnWithString() {  
        return "Remy";  
    }  
}
```

Output:

Remy

Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = returnWithString();  
    System.out.println(name);  
}
```

```
static String returnWithString() {  
    return "Remy";  
}
```

Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = returnWithString();  
    System.out.println(name);  
}
```

```
static String returnWithString() {  
    return "Remy";  
}
```

Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = returnWithString();  
    System.out.println(name);  
}
```

Caller

```
static String returnWithString() {  
    return "Remy";  
}
```

Callee

Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = returnWithString();  
    System.out.println(name);  
}
```

Caller

Callee

```
static String returnWithString() {  
    return "Remy";  
}
```


Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = returnWithString();  
    System.out.println(name);  
}
```

Caller

```
static String returnWithString() {  
    return "Remy";  
}
```

Callee



Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = returnWithString();  
    System.out.println(name);  
}
```

Caller

“Remy”

```
static String returnWithString() {  
    return “Remy”;  
}
```

Callee

Visualize Method แบบที่ 3

```
public static void main(String[] args) {  
    String name = "Remy";  
    System.out.println(name);  
}
```

Caller

```
static String returnWithString() {  
    return "Remy";  
}
```

Callee



การสร้าง Method แบบที่ 2+3

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


ลองพิมพ์ Method แบบที่ 2+3

Source:

```
public class Main {  
    public static void main(String[] args) {  
        int x = add(5,10);  
        System.out.println(x);  
    }  
  
    static int add(int a, int b) {  
        int sum = a + b;  
        return sum;  
    }  
}
```

Output:

15

Visualize Method แบบที่ 2+3

```
public static void main(String[] args) {  
    int x = add(5,10);  
    System.out.println(x);  
}
```

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

Visualize Method แบบที่ 2+3

```
public static void main(String[] args) {  
    int x = add(5,10);  
    System.out.println(x);  
}
```

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

Visualize Method แบบที่ 2+3

caller

```
public static void main(String[] args) {  
    int x = add(5,10);  
    System.out.println(x);  
}
```

callee

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



Visualize Method แบบที่ 2+3

caller

```
public static void main(String[] args) {  
    int x = add(5, 10);  
    System.out.println(x);  
}
```

callee

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



Visualize Method แบบที่ 2+3

caller

```
public static void main(String[] args) {  
    int x = add(5, 10);  
    System.out.println(x);  
}
```

callee

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

Visualize Method แบบที่ 2+3

caller

```
public static void main(String[] args) {  
    int x = add(5, 10);  
    System.out.println(x);  
}
```

callee

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

```
graph TD; subgraph caller; C1[main] -- "5" --> C2[add]; C1 -- "10" --> C2; C2 -- "sum" --> C1; end; subgraph callee; C2[add]; end;
```

Visualize Method แบบที่ 2+3

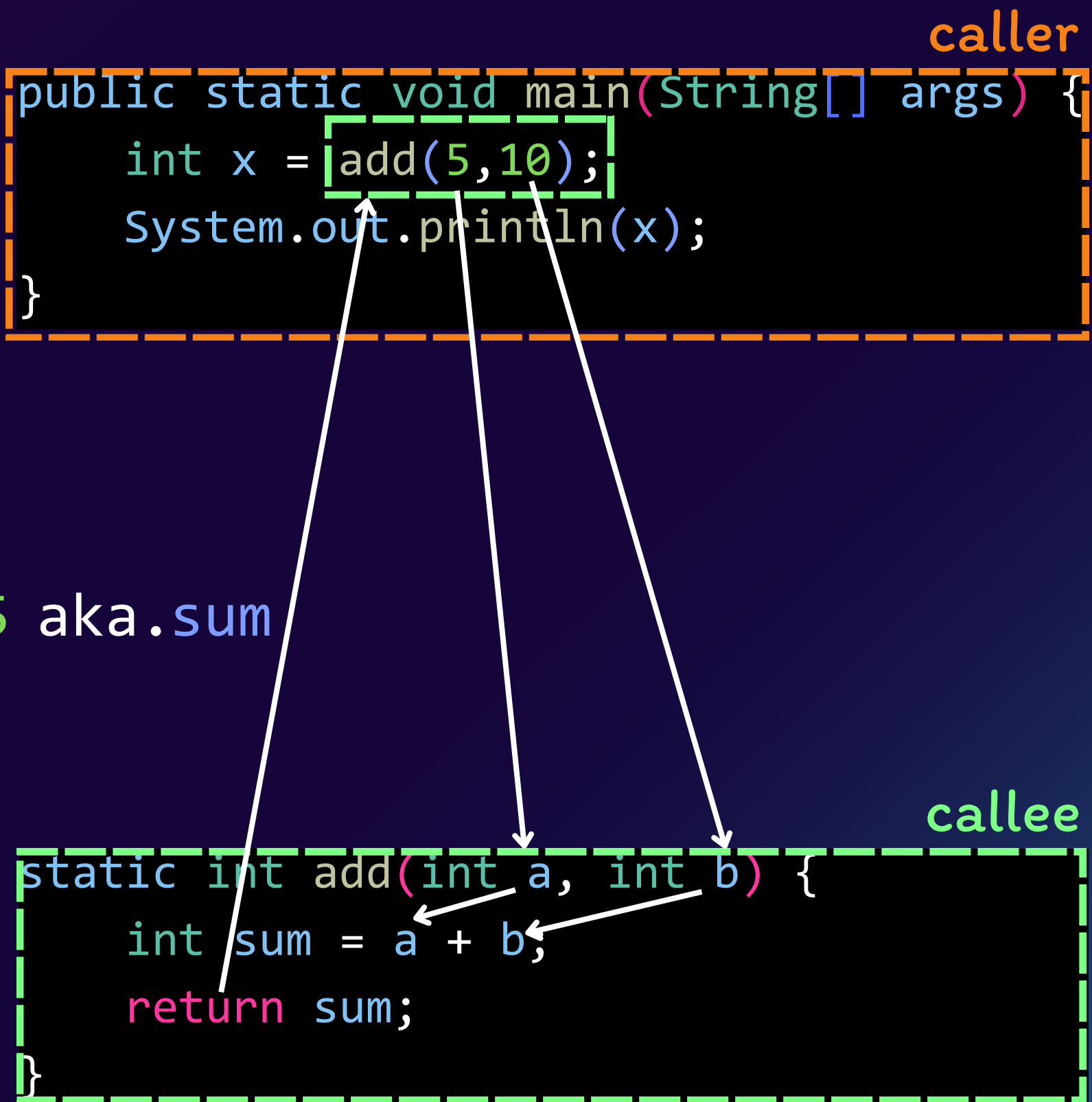
caller

```
public static void main(String[] args) {  
    int x = add(5, 10);  
    System.out.println(x);  
}
```

15 aka.sum

callee

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



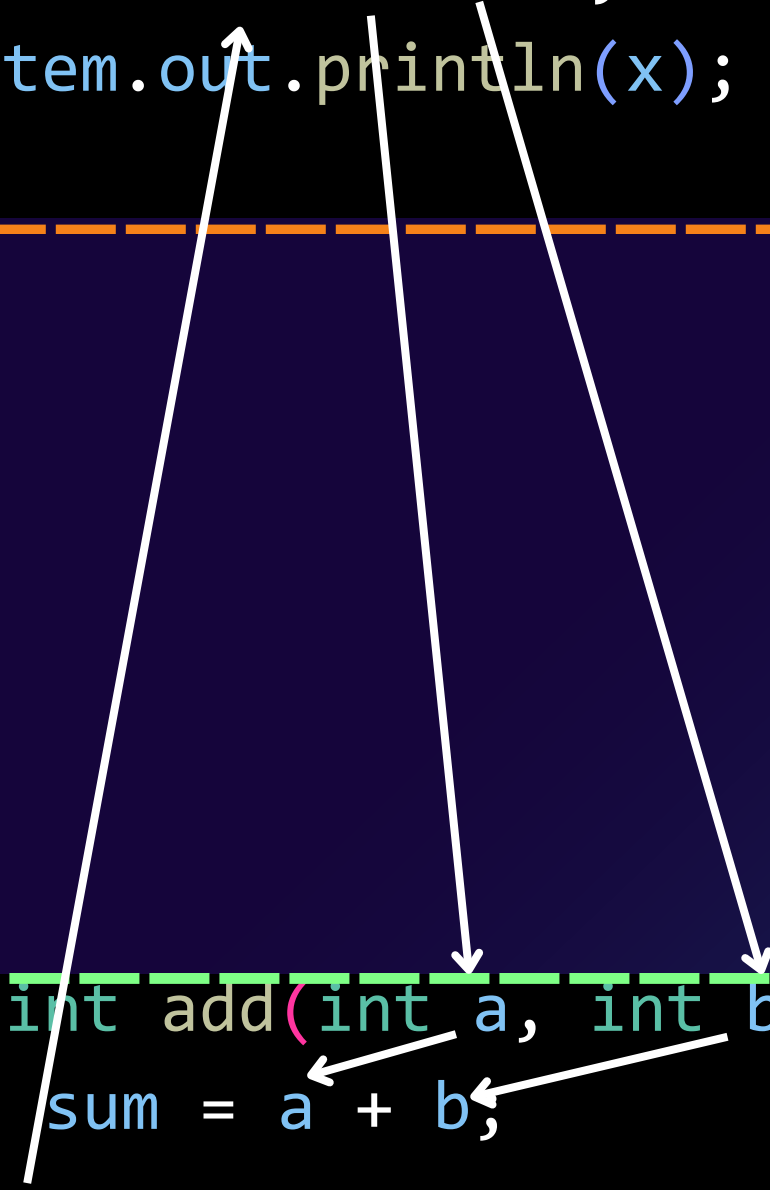
Visualize Method แบบที่ 2+3

caller

```
public static void main(String[] args) {  
    int x = 15 aka.sum ;  
    System.out.println(x);  
}
```

callee

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





รู้จักกับ

Pass By Value และ Pass By Reference



Pass by value and Pass by reference

pass by reference



fillCup()

pass by value



fillCup()

Pass by Value

Source:

```
public class Main {  
    public static void main(String[] args) {  
        int x = 5;  
        increment(x);  
        System.out.println("Caller: " + x);  
    }  
  
    static void increment(int x) {  
        x = x + 1;  
        System.out.println("Callee: " + x);  
    }  
}
```

Output:

```
Callee : 6  
Caller  : 5
```

อธิบาย :

การส่งค่าไปใน method จะเกิดสิ่งที่เรียกว่า **Pass by Value**

ตัว Caller จะทำการ **copy** ค่าที่เราส่งเข้าไป ในที่นี้ก็คือ x และ Callee ก็จะได้รับค่า copy x จาก Caller

เมื่อ ใน Callee มีการเปลี่ยนแปลงค่าใน **Caller** จะไม่ได้รับผลกระทบใดๆ

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

Caller

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Callee

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

Caller

Copy x

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Callee

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

Caller

Copy x

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Callee

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

Caller

Copy x

Copy x

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Callee

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

Caller

Copy x

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Callee

Copy x

Copy x

Visualize Pass by Value

```
public static void main(String[] args) {  
    int x = 5;  
    increment(x);  
    System.out.println("Caller: " + x);  
}
```

Caller

Copy x

```
static void increment(int x) {  
    x = x + 1;  
    System.out.println("Caller: " + x);  
}
```

Callee

Copy x

Copy x



Pass by Reference

นำเสียดายที่ ภาษา Java ไม่มีสิ่งนี้





แต่เราสามารถทำสิ่งที่คล้ายกับ
Pass by reference
ได้



Pass by Reference

Source:

```
public class Main {  
    public static void main(String[] args) {  
        int[] Caller = {1, 2, 3};  
        change(Caller);  
        System.out.println("Caller: " + Arrays.toString(Caller));  
    }  
    static void change(int[] nums) {  
        nums[0] = 4;  
        nums[1] = 5;  
        nums[2] = 6;  
        System.out.println("Callee: " + Arrays.toString(nums));  
    }  
}
```

Output:

```
Callee: [4, 5, 6]  
Caller: [4, 5, 6]
```

อธิบาย :

การส่ง **array** เข้าไปใน method จะเกิดสิ่งที่เรียกว่า **Pass by Value** เช่นกัน

ตัว Caller จะทำการ **ส่งค่าที่อยู่ของตัว array** เข้าไปใน Callee และ Callee ก็จะได้รับค่าที่อยู่ของ array นั้นเข้ามาจาก Caller โดยที่ nums ตัวนี้คือค่าที่อยู่ของ array ที่ caller ส่งเข้ามา ไม่ใช่ค่าจริงๆของ array

เมื่อ ใน Callee มีการเปลี่ยนแปลงค่าใน **Caller** จะได้รับผลกระทบตามไปด้วย