

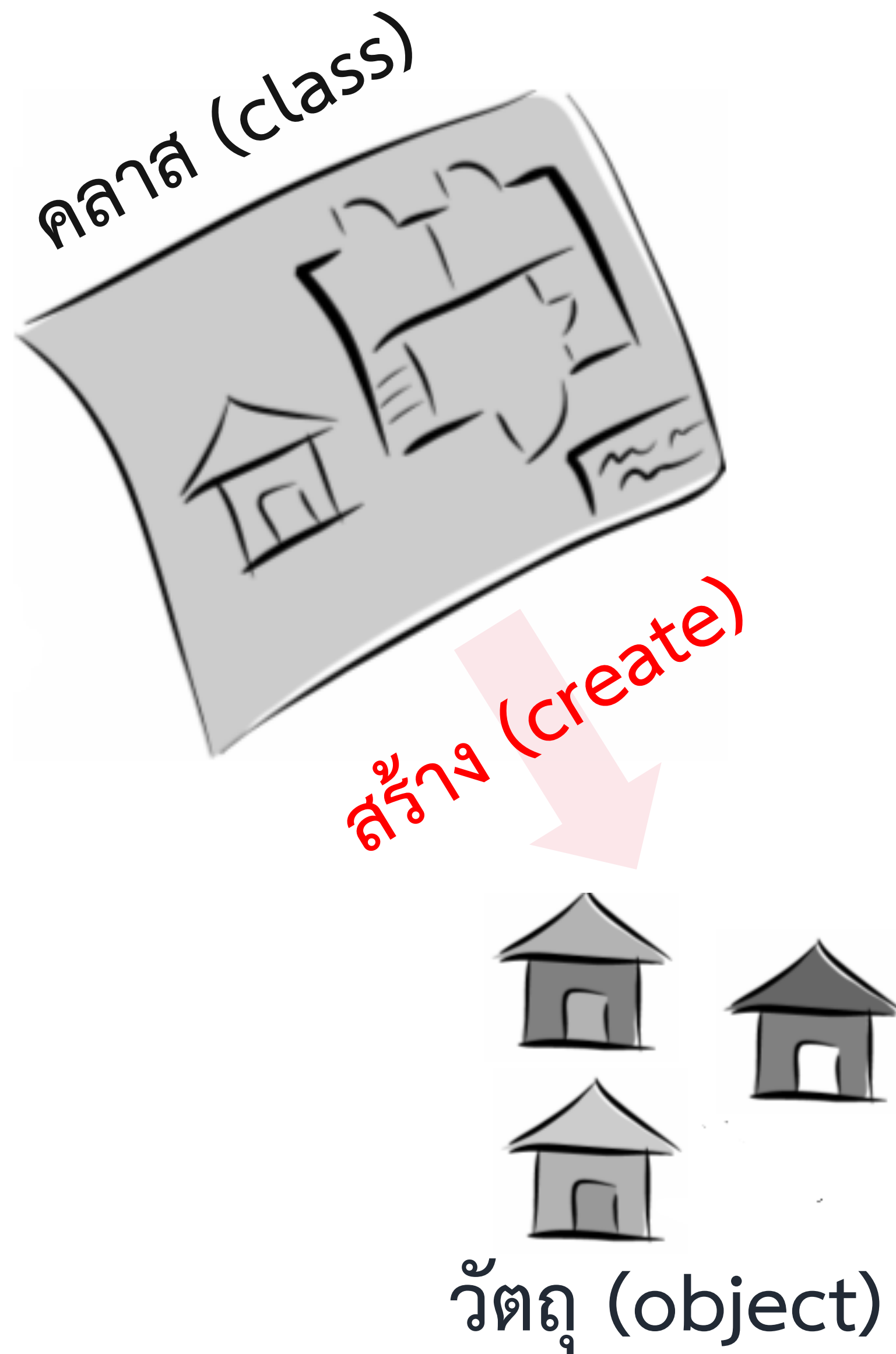
บทที่ 4 การเขียนโปรแกรมเชิงวัตถุเบื้องต้น

บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

แบบแปลนและวัตถุที่สร้างจากแบบ



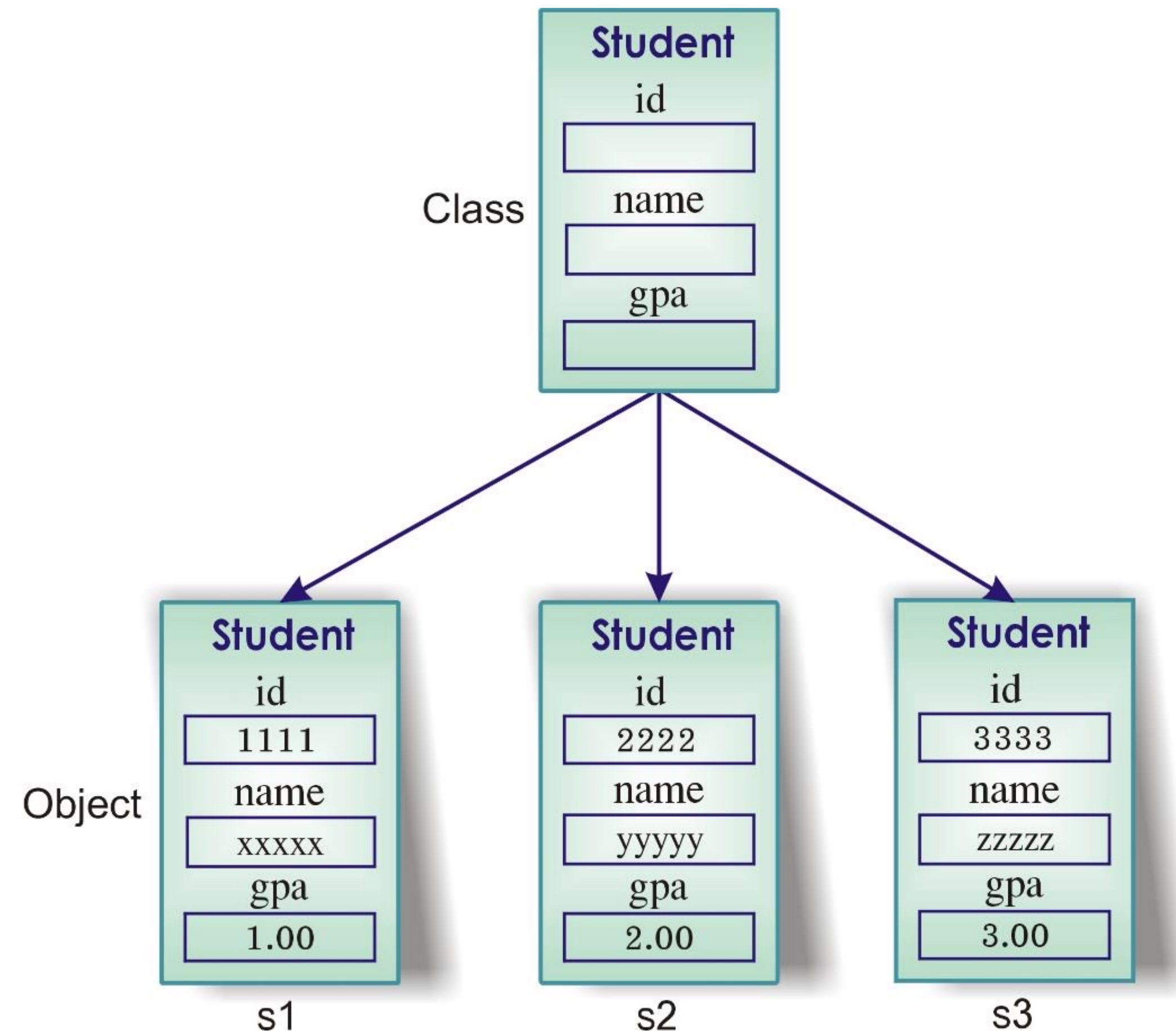
คลาส (Class) คือ แบบแผ่น หรือ แม่แบบ (template/blueprint) ที่ใช้สำหรับการสร้างวัตถุ (Object) เพื่อกำหนดแอตทริบิวต์ (Attribute) และเมธอด (Method) ของวัตถุ ซึ่งเป็นตัวอธิบายรายละเอียด และหน้าที่ต่าง ๆ

วัตถุ (Object) คือ สิ่งที่มี (1) สถานะ (states) หรือแอตทริบิวต์ (Attribute) และ (2) แสดงพฤติกรรม (behaviors) หรือเมธอด (Method)

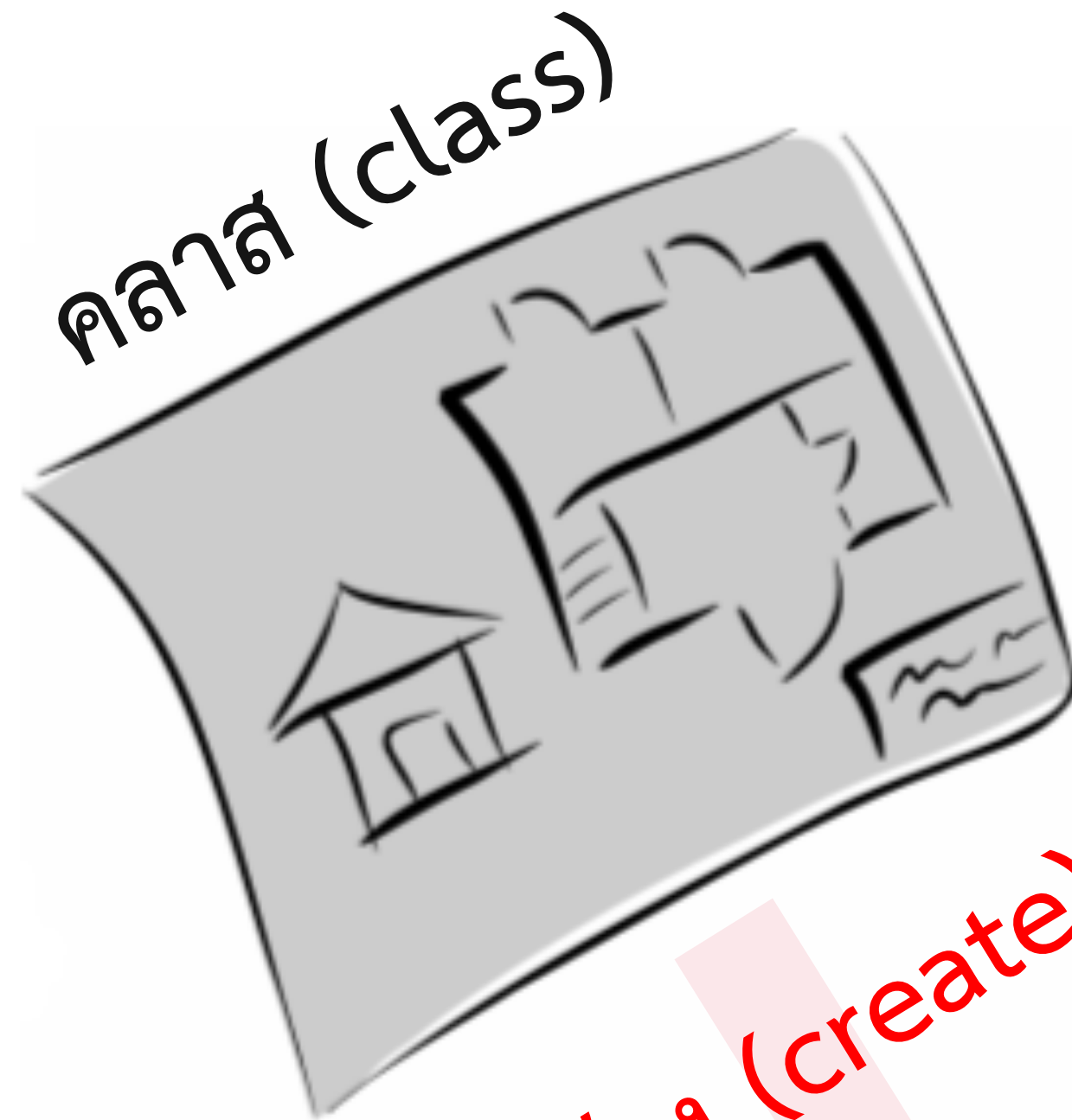
นอกจากนี้ วัตถุที่สร้างมาจากคลาส บางครั้งเรียกว่าเป็น instance ของคลาส ซึ่งคลาสหนึ่งคลาสสามารถสร้างอ็อบเจกต์ได้หลายอ็อบเจกต์ อาทิเช่น คลาสชื่อ Student อาจสร้างอ็อบเจกต์ชื่อ s1, s2 หรือ s3 ซึ่งเป็นอ็อบเจกต์ชนิด Student

ถ้าคลาสเปรียบเสมือน **ชนิดสินค้า** แล้ววัตถุก็เปรียบเสมือน **สินค้า** ในแต่ละชนิด

แอททริบิวต์



แบบแปลนและวัตถุที่สร้างจากแบบ



ประโยชน์ของการสร้างคลาส

- (1) จัดการหรืออ้างอิงการใช้งานในภายหลัง
- (2) ติดป้ายบ่งบอกว่าสิ่งที่เก็บคืออะไร
- (3) เป็นการให้ความหมายของสิ่งที่จัดเก็บ

วัตถุ (object)

การสร้างคลาสในภาษาจาวา

header class

①

②

③

```
[modifier] class <Classname> {
```

```
[class member]
```

```
}
```

class body

- [modifier] คือ

private
→ public

คีย์เวิร์ด (keyword) ของภาษาจาวาที่ใช้ในการอธิบายระดับการเข้าถึง (access modifier)

- class คือ

คีย์เวิร์ดของภาษาจาวาเพื่อระบุว่าเป็นการประกาศคลาส

- <Classname> คือ

ชื่อคลาส

- [class member] คือ

เมธอดหรือคุณลักษณะ

การสร้างคลาสในภาษาจาวา



```
public class ComplexNumber{  
  
}
```

ComplexNumber.java

คลาสในภาษาจาวาจะถูกเขียนอยู่ในไฟล์สกุล java

การประกาศและสร้างวัตถุในภาษาจาวา

คลาส (Class)



Home myHome1 = new Home();
Home myHome2 = new Home();
Home myHome3 = new Home();



วัตถุ (Object)

รูปแบบ

```
class ref;           // ประกาศ
ref = new class ();  // สร้าง
```

ตัวอย่าง

obj. name / var. name / ref.

```
① String str;
② str = new String("Alex");
```

```
ComplexNumber c;
c = new ComplexNumber();
```

การประกาศและสร้างวัตถุในภาษาจาวา

คลาส (Class)



Home myHome1 = new Home();
Home myHome2 = new Home();
Home myHome3 = new Home();



วัตถุ (object)

ตัวแปรวัตถุ (instance variable)

รูปแบบ

```
class ref = new class ();
```

ตัวอย่าง

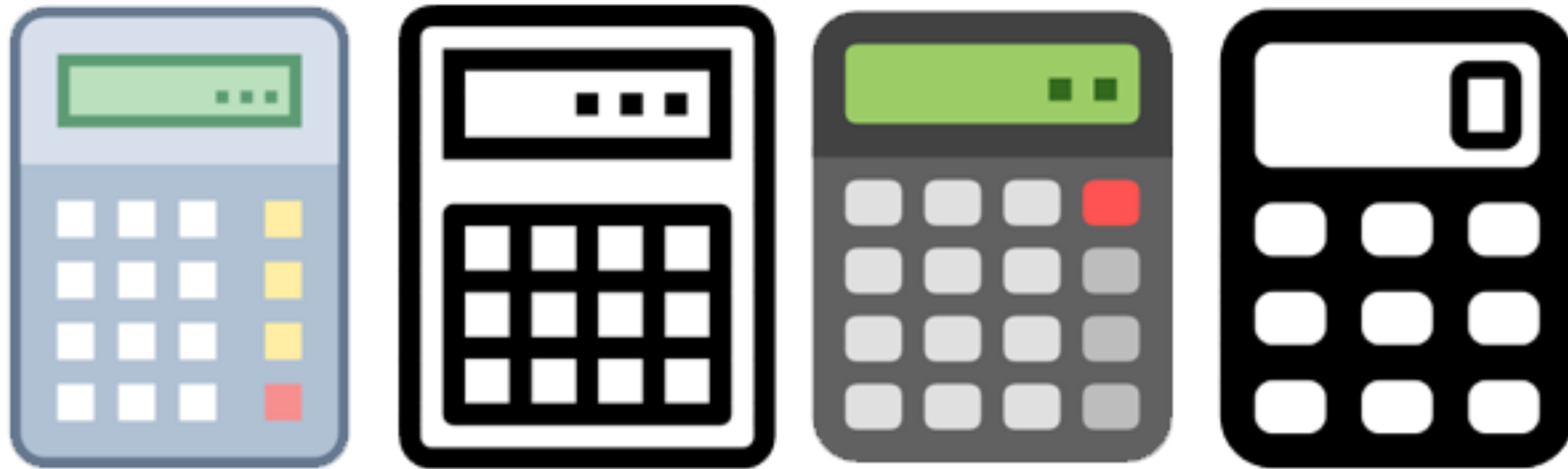
```
String str = new String("Alex");  
ComplexNumber c = new ComplexNumber ();
```

①

③

②

ส่วนประกอบของคลาส



- **แอททริบิวต์** คือ สิ่งที่บ่งบอกถึงลักษณะต่าง ๆ ของวัตถุในคลาส เช่น สี ขนาด รูปแบบฟอนต์ รุ่น เป็นต้น
- **เมธอด** คือ สิ่งที่อธิบายการทำงานของวัตถุในคลาส เช่น บวก() ลบ() คูณ() หาร() เป็นต้น

ส่วนประกอบของคลาส



```
public class ComplexNumber {
```

แอททริบิวต์ (Attribute)

```
private double realNum ;  
private double imagNum ;
```

* Attribute → จะไม่มี () หรือ {} ต่อท้าย

```
public void myAdd (ComplexNumber c1, ComplexNumber c2) {  
    realNum = c1.realNum + c2.realNum;  
    imagNum = c1.imagNum + c2.imagNum;  
    System.out.print(realNum + " + " + imagNum + "i");  
}  
public void myPrint(ComplexNumber c) {  
    System.out.print(c.realNum + " + " + c.imagNum + "i");  
}
```

เมธอด (Method)

```
}
```

ตัวอย่างคลาส

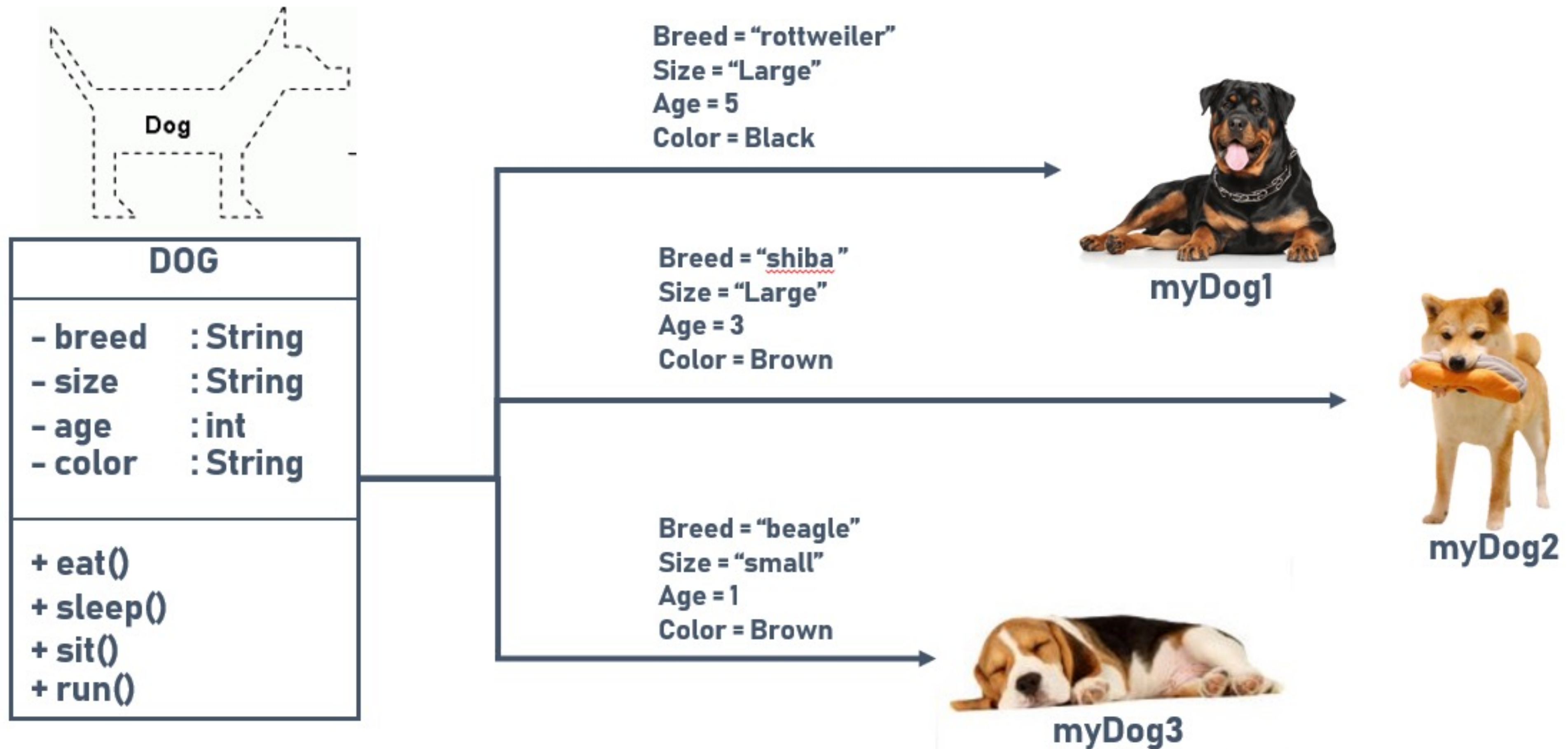


```
public class Math{  
  
    public static double E = xxx;  
    public static double PI = 3.14xx;  
  
    public static double sin(double a) {...}  
    public static double cos(double a) {...}  
    public static double exp(double a) {...}  
    public static double log(double a) {...}  
    public static double sqrt(double a) {...}  
    public static double ceil(double a) {...}  
    public static double pow(double a, double b) {...}  
    public static double abs(double a) {...}  
  
}
```

แอททริบิวต์



แอททริบิวต์ คือ สิ่งที่บ่งบอกถึงลักษณะต่าง ๆ ของวัตถุในคลาส เช่น สี ขนาด รูปแบบฟอนต์ รุ่น เป็นต้น สำหรับการพัฒนาโปรแกรมเชิงวัตถุ แอททริบิวต์ คือ ข้อมูลที่เก็บอยู่ในอ็อบเจกต์ หรือข้อมูลที่ต้องการเก็บ ซึ่งสามารถแบ่งเป็นตัวแปร และ ค่าคงที่



แอททริบิวต์



การประกาศแอททริบิวต์

num, firstName, myLastName
↑
optional

① **Modifier** | ② **DataType** | ③ **AttributeName** [= value] ;

- **Modifier** คือ

คีย์เวิร์ดของภาษาจาวาที่อธิบายคุณสมบัติต่าง ๆ ของตัวแปร หรือ ค่าคงที่

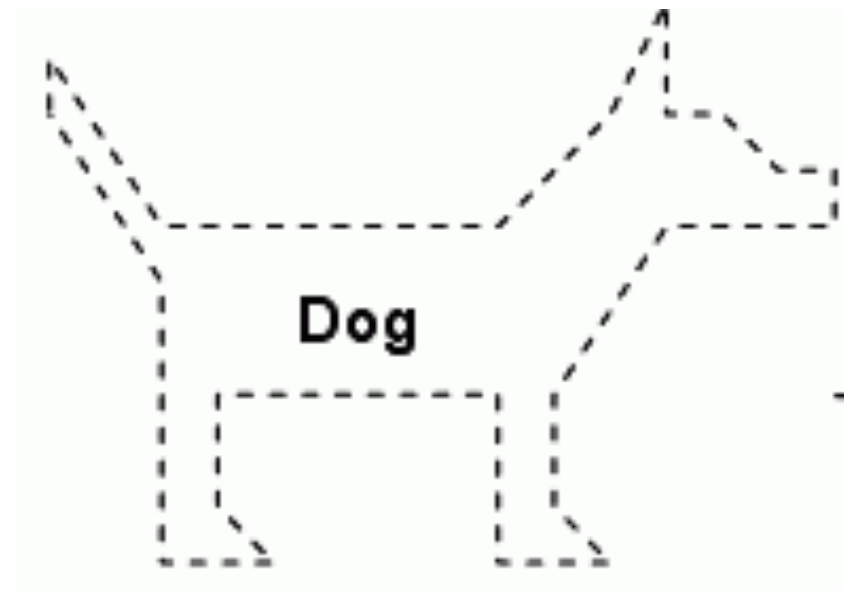
- **DataType** คือ

ชนิดข้อมูลซึ่งอาจเป็นชนิดข้อมูลพื้นฐาน หรือ ชนิดคลาส

- **AttributeName** คือ

ชื่อของคุณลักษณะ

แอททริบิวต์



DOG	
- breed	: String
- size	: String
- age	: int
- color	: String
+ eat() + sleep() + sit() + run()	

```
public class Dog{  
    private String breed;  
    private String size;  
    private int age;  
    private String color;  
  
}
```

การใช้งานแอตทริบิวต์

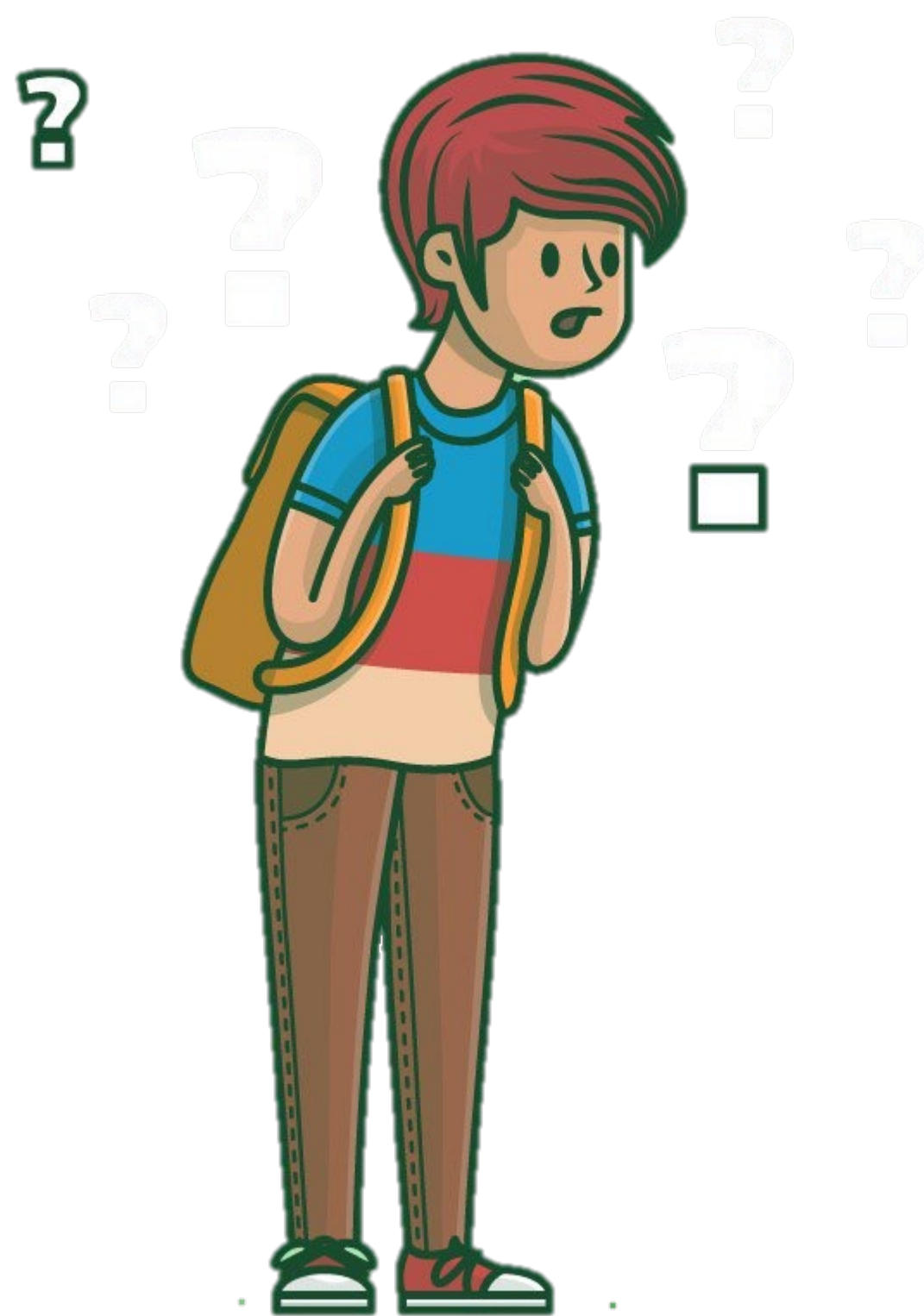
รูปแบบ

obj. ตัวโน้ต attr. ชื่อแอตทริบิวต์

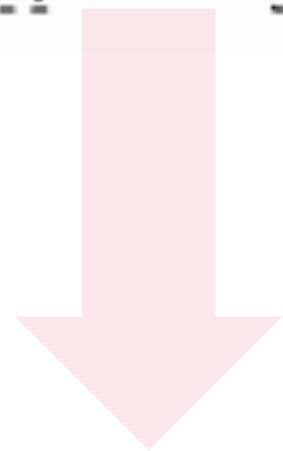
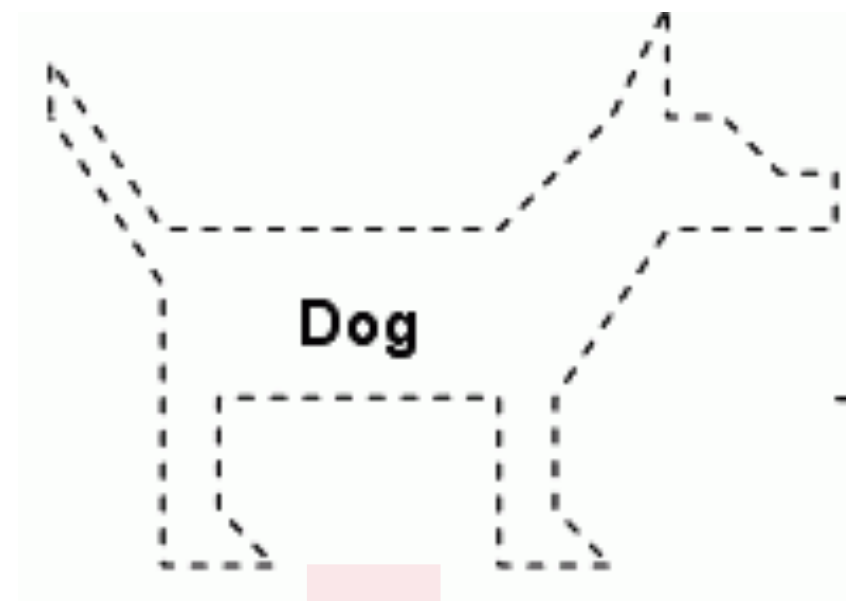
`ref.attributeName`

ตัวอย่าง

```
Dog myDog1 = new Dog();  
myDog1.age = 5;  
myDog1.color = "Black";
```



การกำหนดค่าแอตทริบิวต์



myDog1

```
public class Main{  
    public static void main (String [] args){  
        ① Dog myDog1 = new Dog();  
        ② {  
            myDog1.breed = "rottweiler";  
            myDog1.size = "Large";  
            myDog1.age = 5;  
            myDog1.color = "Black";  
        }  
    }  
}
```

ค่าเริ่มต้นของตัวแปร VS ค่าของแอททริบิวต์

```
public class Main {  
    public static void main(String[] args) { } นี่คือ method  
        int num; * ตัวแปร  
        System.out.println(" num : " + num );  
    } ()  
}
```

output

```
javac -classpath ./run_dir/junit-4.12.jar -d . Main.java  
Main.java:4: error: variable num might not have been initialized  
    System.out.println(" num : " + num );  
                                ^
```

ใน java ตัวแปรปกติจะไม่มีการกำหนดค่าเริ่มต้น

1 error

compiler exit status 1

ค่าเริ่มต้นของตัวแปร VS ค่าของแอททริบิวต์

```
public class Main {  
    public int num; *attribute  
    public static void main(String[] args) {  
        Main m = new Main();  
        System.out.println(" num : " + m.num );  
    }  
}
```

{}

ใน java ถ้าตัวแปรเป็น attribute จะมีการกำหนดค่าเริ่มต้นให้

output

```
javac -classpath ./run_dir/junit-4.12.jar -d . Main.java  
java -classpath ./run_dir/junit-4.12.jar Main  
num : 0
```


ค่าเริ่มต้นของตัวแปร VS ค่าของแอตทริบิวต์

```
public class Main {  
    public double num;  
    public String name;  
    public char blood;  
    public boolean hasPhone;
```

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

Ⓐ {

```
        Main m = new Main();  
        System.out.println(" num : " + m.num );  
        System.out.println(" name : " + m.name );  
        System.out.println(" blood : " + m.blood );  
        System.out.println(" hasPhone : " + m.hasPhone );
```

Ⓑ {

```
        //System.out.println(" num : " + new Main().num );  
        //System.out.println(" name : " + new Main().name );  
        //System.out.println(" blood : " + new Main().blood );  
        //System.out.println(" hasPhone : " + new Main().hasPhone );  
    }  
}
```

สร้างในเมธอดแล้วไม่มีการใช้ซ้ำ

Output

```
❑ javac -classpath ./run_dir/junit-4.12.jar -d . Main.java  
❑ java -classpath ./run_dir/junit-4.12.jar Main
```

```
num : 0.0  
name : null  
blood :  
hasPhone : false
```

ค่าเริ่มต้นของ attribute :

float, double = 0.0

byte, int, short, long = 0

boolean = false

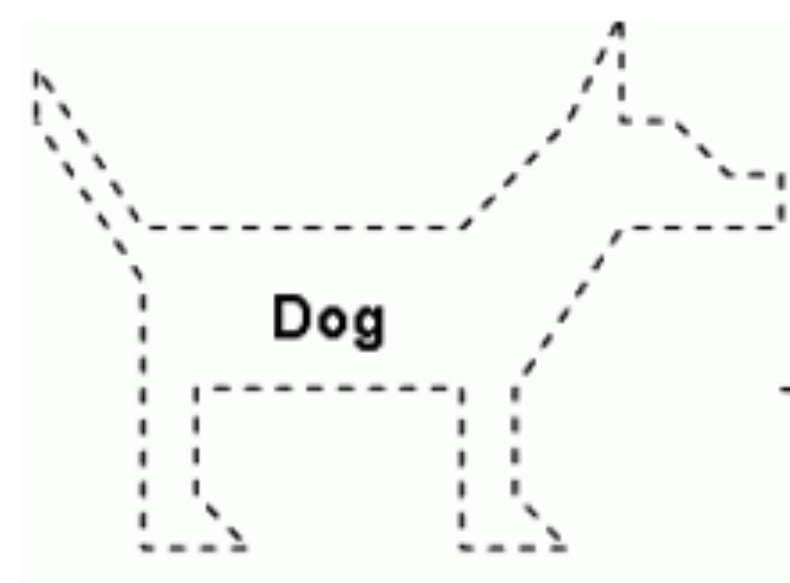
char = ''

all ตัวแปรแบบอ้างอิง = null

เมธอด

เมธอด คือ (1) สิ่งที่ใช้บรรยายการทำงานของวัตถุในคลาส เช่น eat() sleep() sit() run() เป็นต้น หรือ (2) วิธีการหรือการกระทำที่นิยามอยู่ในคลาสหรืออ็อบเจกต์เพื่อใช้ในการจัดการกับคุณลักษณะของอ็อบเจกต์ ซึ่งเปรียบเทียบกับ function, procedure หรือ subroutine ของโปรแกรมเชิงกระบวนการ

- เมธอดจะทำงานก็ต่อเมื่อได้รับข้อความ
- ชื่อของเมธอดควรเป็นคำกริยา



DOG	
- breed	: String
- size	: String
- age	: int
- color	: String
+ eat() + sleep() + sit() + run()	

sit()



myDog1

eat()



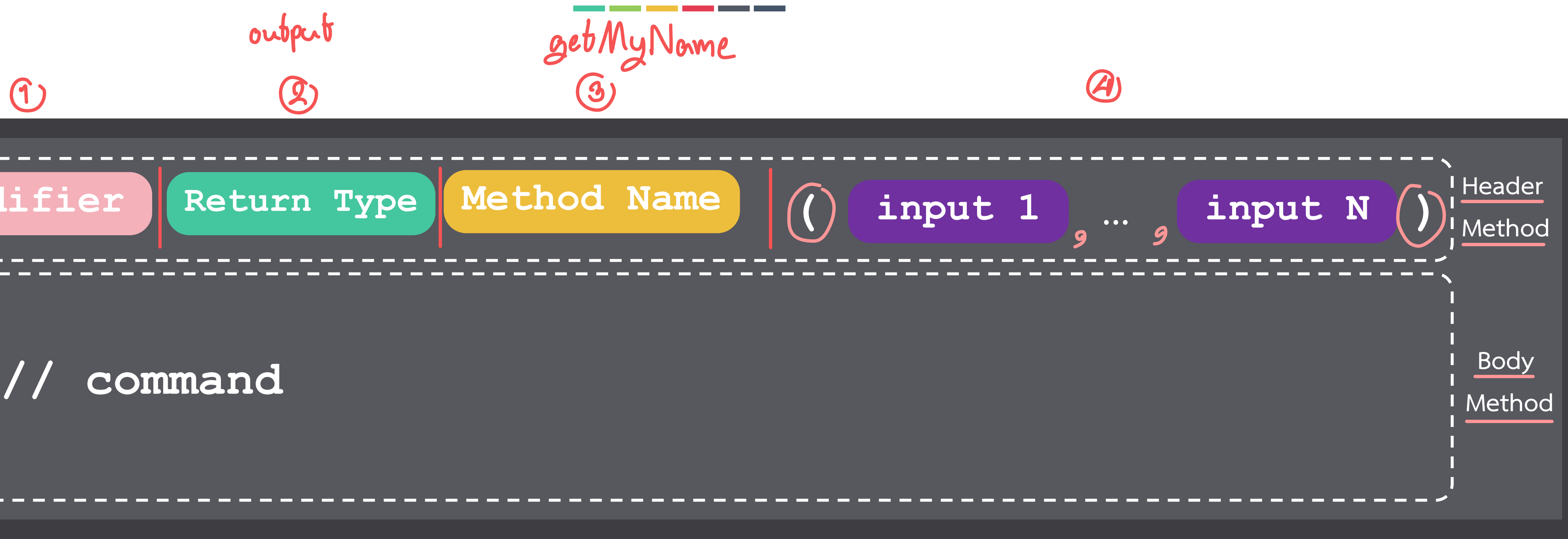
myDog2

sleep()



myDog3

การประกาศเมธอด



เมธอด ประกอบไปด้วย 2 ส่วนหลัก ได้แก่ Header และ Body ซึ่งในส่วน Header จะใช้เพื่อระบุ (1) Access Modifier, (2) Return Type, (3) Method Name และ (4) Input ของเมธอด ขณะที่ส่วน Body ใช้ระบุขั้นตอนการทำงานซึ่งให้ได้ผลลัพธ์ตามที่ต้องการ

การประกาศเมธอด

`x = printMyName(); // error`

`x = getMyName(); // x = "Bank"`

นักศึกษาสามารถสร้างเมธอดได้เป็น 4 แบบ โดยจัดกลุ่มตามกับรับค่าและการคืนค่า

① `public void printMyName() {
 System.out.print(" Bank ");
}`
→ keyword นี้จะไม่มีการ return ค่า

② `public String getMyName() {
 return "Bank";
}`

4

③ `public void prtName(String name) {
 System.out.print(name);
}`
เก็บข้อมูลลง database

④ `public int addTwo(int a) {
 return (a+2);
}`
เก็บค่าและคืนค่าเพิ่มในด้านลบ

อย่างไรก็ตาม การคืนค่า (return) คือการคืนค่าให้ผู้เรียกใช้งานนำค่าไปใช้งานต่อ ซึ่งไม่เท่ากับการใช้คำสั่ง print ()

การเรียกใช้เมธอด



รูปแบบของคำสั่งที่มีการเรียกใช้เมธอดเป็นดังนี้

```
obj.methodName( [arguments] ) ;
```

โดยที่ arguments อาจจะเป็นข้อมูล **ค่าคงที่หรือตัวแปร** นอกจากนี้ ชนิดข้อมูลของ arguments ที่ใช้ในการเรียกเมธอด จะต้องสอดคล้องกันกับชนิดข้อมูลของ arguments ของเมธอด

การเรียกใช้เมธอด



`myDog1.sit();`



`myDog1`

```
public class Main{  
    public static void main (String [] args) {  
        ① Dog myDog1 = new Dog();  
        ② {  
            myDog1.breed = "rottweiler";  
            myDog1.size = "Large";  
            myDog1.age = 5;  
            myDog1.color = "Black";  
        }  
        ③ myDog1.sit();  
    }  
}
```

ตัวอย่างการเรียกใช้เมธอด

```

public class Player{
    public int HP, ATK; attribute
    public void cut (Player p) { method
        p.HP -= ATK;
    }
}

public class Main {
    public static void main (String [] args) {
        Player p1 = new Player();
        Player p2 = new Player();

        // P2 is attacked by P1
        P1.cut(P2);
    }
}
  
```

p₁ {
p₂

$$p.HP = p.HP - ATK$$
(P2) (P1)

1 ไม่ล้น 1 Class

* ตย. ที่ไม่ดี

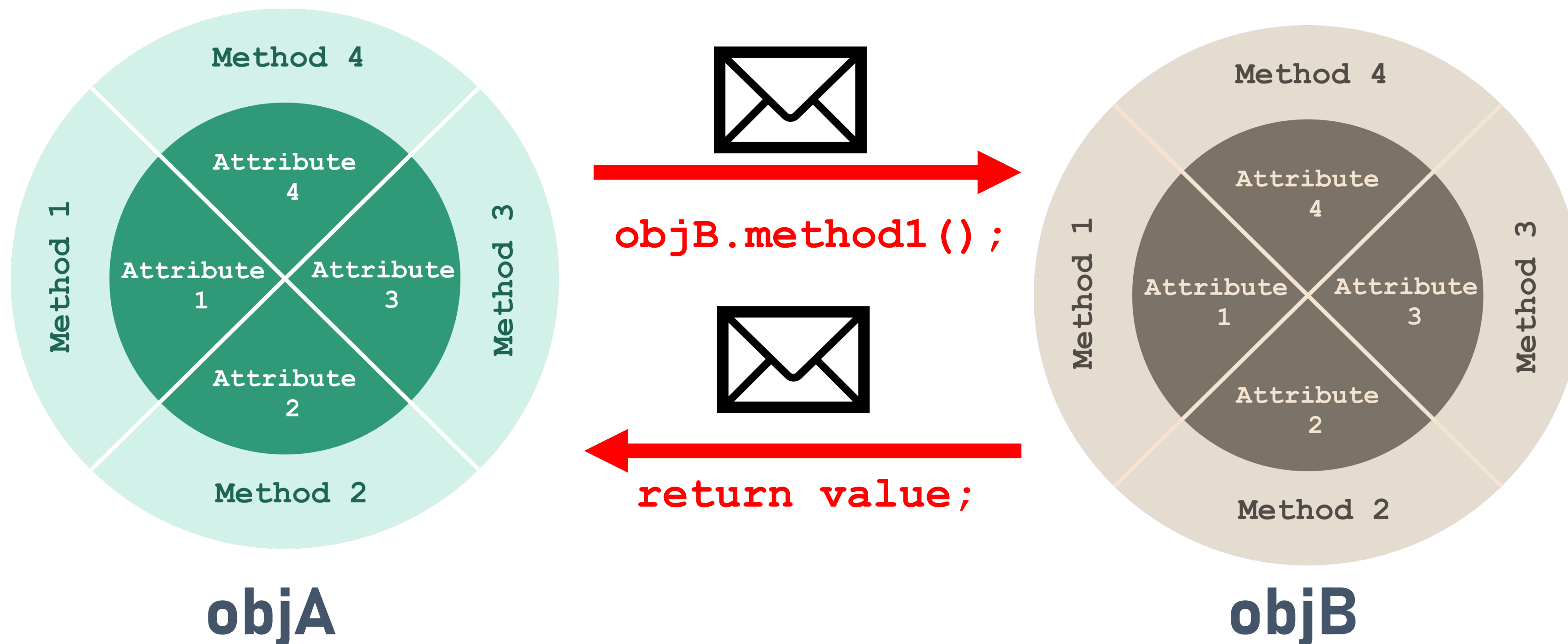
P₁ เรียกใช้ cut

P₂ โดน P₁ ใช้ cut ใจ

การสื่อสารระหว่างอ็อบเจกต์



คือ การรับส่งข้อมูลระหว่างอ็อบเจกต์ หรือ ภายในอ็อบเจกต์ ตัวอย่างเช่น ข่าวสารจะส่งผ่าน method 3 จากอ็อบเจกต์ objA ที่เป็นผู้ส่ง (sender) เพื่อเรียกการทำงานของเมธอดที่ชื่อ method1 จากอ็อบเจกต์ objB ที่เป็นผู้รับ (receiver) ซึ่ง objB อาจส่งค่า (return value) บางค่ากลับมายัง objA



การเรียกใช้เมธอด



- คลาสเดียวกัน

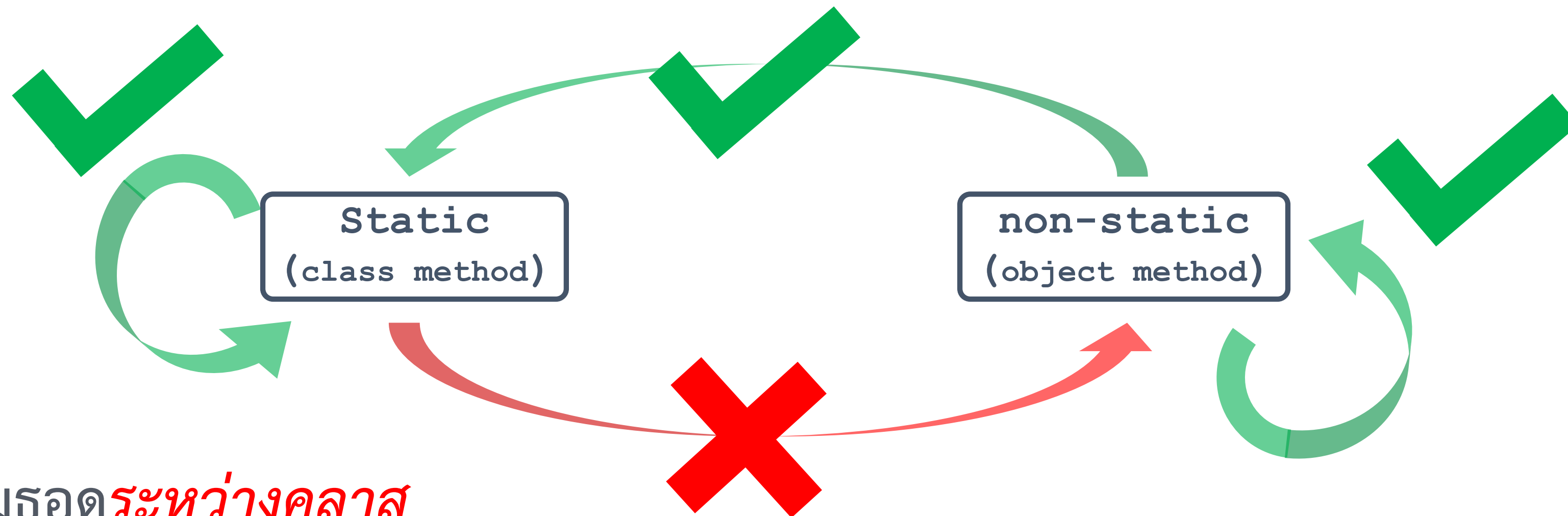
การเรียกใช้เมธอดภายในคลาสเดียวกัน ในบางกรณีจะสามารถทำได้โดยไม่ต้อง
จำเป็นต้องสร้างอ็อบเจกต์ของคลาสขึ้นมาก่อน และสามารถเรียกเมธอดได้ทุก
เมธอด

- คลาสที่ต่างกัน

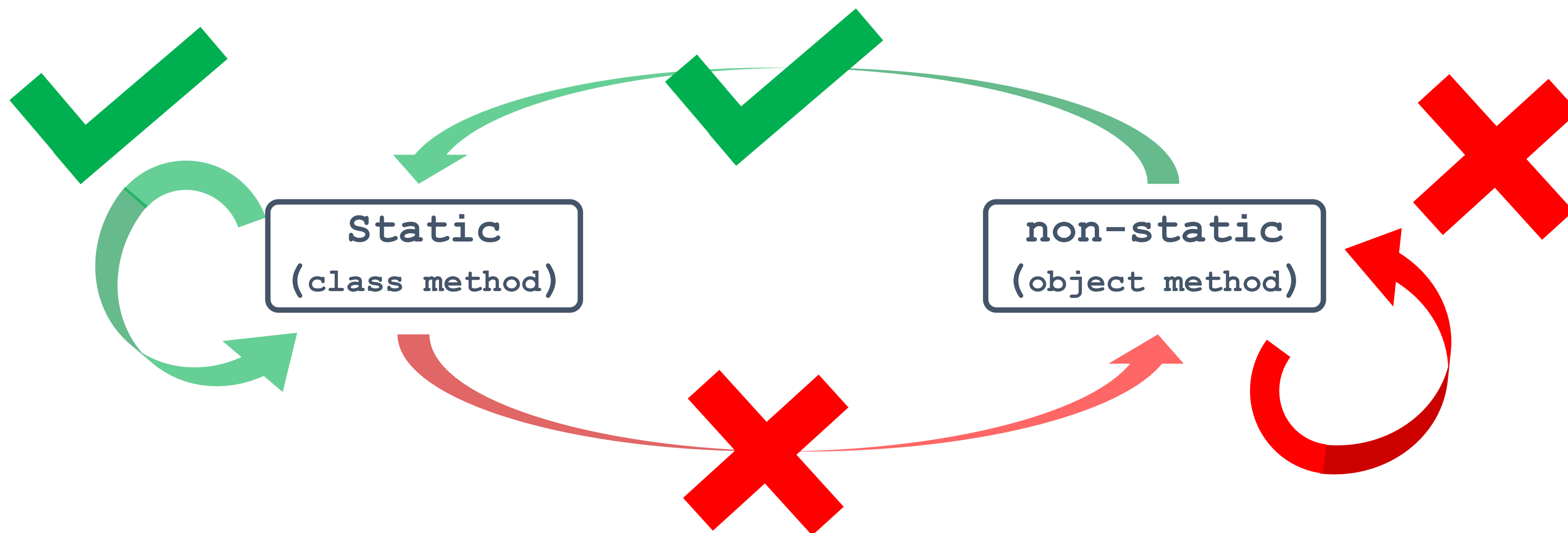
การเรียกใช้เมธอดจากคลาสที่ต่างกัน จะต้องมีการสร้างอ็อบเจกต์ของคลาสที่
มีเมธอดที่จะถูกเรียกใช้งานก่อน จึงจะสามารถเรียกใช้เมธอดได้

ความสัมพันธ์ระหว่างเมธอดแบบ *static* และ ไม่ *static*

- การเรียกใช้เมธอด*ภายในคลาสเดียวกัน*



- การเรียกใช้เมธอด*ระหว่างคลาส*



ซึ่งความสัมพันธ์ดังกล่าว
 สอดคล้องกันทั้งในกรณี
 attribute และ method

ตัวอย่างความสัมพันธ์ระหว่างเมธอดแบบ static และ ไม่ static



```

public class Main{
    public static void printMe(){
        System.out.println(" Me ");
    }
    public void printIT(){
        System.out.println(" IT ");
    }
    public static void main(String[] args) {
        House h = new House();
        (1) h.printSize(); (2) printSize(); // (1)
        h.printHouseID(); // (2)
        h.printDetail1(); // (3)
        h.printDetail2(); // (4)
        h.printHi(); // (5)
        printSize(); // (6)
        printHi(); // (7)

        printMe(); // (8)
        Main m = new Main();
        m.printIT(); // (9)
    }
}
  
```

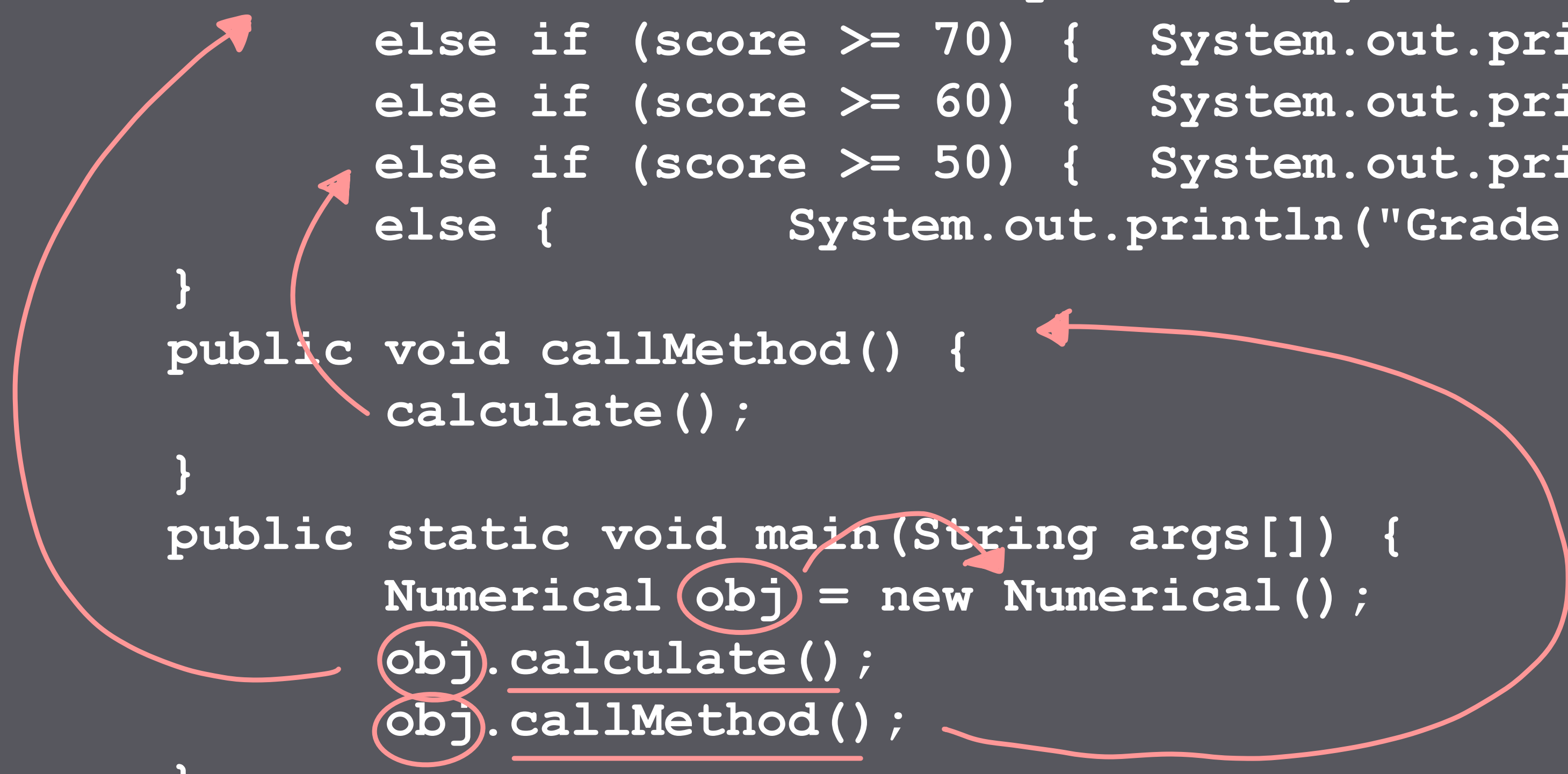
```

public class House{
    public static int size ;
    public int house_id ;
    public static void printHi(){
        System.out.println(" Hi ");
        printSize();
    } public static void printSize(){
        System.out.println("Size "+ size);
    } public void printHouseID(){
        System.out.println("House "+ house_id);
    } public void printDetail1(){
        printSize();
    } public void printDetail2(){
        printHouseID();
    }
}
  
```

การเรียกใช้เมธอดในคลาสเดียวกัน



```
public class Numerical {  
    public void calculate() {  
        double score = Math.random()*100;  
        if (score >= 80) { System.out.println("Grade is A"); }  
        else if (score >= 70) { System.out.println("Grade is B"); }  
        else if (score >= 60) { System.out.println("Grade is C"); }  
        else if (score >= 50) { System.out.println("Grade is D"); }  
        else { System.out.println("Grade is F"); }  
    }  
    public void callMethod() {  
        calculate();  
    }  
    public static void main(String args[]) {  
        Numerical obj = new Numerical();  
        obj.calculate();  
        obj.callMethod();  
    }  
}
```



การเรียกใช้เมธอดต่างคลาสกัน



```
public class Numerical {
    public void calculate() {
        double score = Math.random()*100;
        if (score >= 80) { System.out.println("Grade is A"); }
        else if (score >= 70) { System.out.println("Grade is B"); }
        else if (score >= 60) { System.out.println("Grade is C"); }
        else if (score >= 50) { System.out.println("Grade is D"); }
        else { System.out.println("Grade is F"); }
    }
}

public class Main {
    public static void main(String args[]) {
        Numerical obj = new Numerical();
        obj.calculate();
    }
}
```

การสร้างออปเจกมาใช้เพียงครั้งเดียว



```
public class Member {  
    public static void main(String[] args) {  
        int s = 10;  
        new Member().printNum(s);  
  
        Member m = new Member();  
        m.printNum(s);  
    }  
    public void printNum(int s) {  
        System.out.println("s is "+s);  
    }  
}
```

Output:

s is 10

s is 10

การส่งค่าเข้าไปในเมธอด (Argument)



พารามิเตอร์ (parameter) และ **อาร์กิวเมนต์ (argument)** คือ ตัวแปรที่ใช้สำหรับการรับและส่งค่าตัวแปรของระหว่างวัตถุ ผ่านทางเมธอดต่าง ๆ ซึ่ง parameter และ argument คือ สิ่งเดียวกัน

- แต่เราจะใช้คำว่า **parameter** ก็ต่อเมื่อค่าตัวแปรเข้ามาในเมธอด (**อยู่ในเมธอด**)
- ขณะที่ **argument** จะเรียกใช้ตอนส่งค่าเข้าเมธอด (**ตอนเรียกใช้งาน**)

```
public class Main {  
    public void call(String name) {  
        System.out.println("Hi ,"+name) ;  
    }  
    public static void main(String args[]) {  
        Main m = new Main() ;  
        m.call("Bank") ;  
    }  
}
```

กรณีที่เมธอดมี argument ที่จะรับค่าเพื่อนำไปใช้ในเมธอด อาทิเช่น คำสั่งที่เรียกใช้เมธอด **call(...)** จะต้องส่ง argument ที่มีชนิดข้อมูลเป็น String ไปด้วย

การส่งค่าเข้าไปในเมธอด (Argument)



อาร์กิวเมนต์สำหรับชนิดข้อมูลพื้นฐาน เราสามารถที่จะส่งค่าคงที่ข้อมูล ตัวแปร และ นิพจน์

```
Animal a1 = new Animal ();  
a1.setAge(5);    // ค่า  
  
int a = 5;  
a1.setAge(a);    // ตัวแปร  
  
a1.setAge(4+2);  // นิพจน์  
a1.setAge(4+a);  // นิพจน์
```

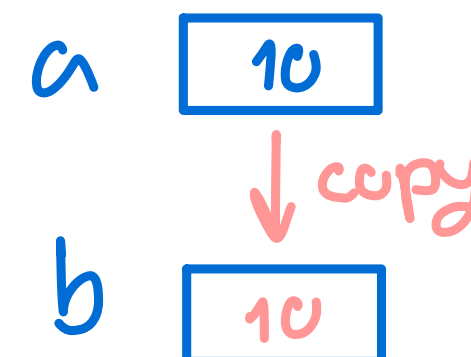
นอกจากนี้ อาร์กิวเมนต์สำหรับชนิดข้อมูลวัตถุ หรือ แบบอ้างอิง เราจะต้องส่งอ็อบเจกต์ที่มีชนิดข้อมูลที่สอดคล้องไปเท่านั้น ยกเว้นกรณีที่ argument นั้นมีชนิดข้อมูลเป็น String ซึ่งในกรณีนี้จะสามารถส่งข้อมูลค่าคงที่ได้

การเปลี่ยนแปลงค่าของ Argument



- การส่ง argument ที่มีชนิดข้อมูลเป็น **แบบพื้นฐาน** หากมีการเปลี่ยนแปลงค่าของ argument ภายในเมธอดที่ถูกเรียกใช้งาน **จะไม่มีผลทำให้ค่าของ argument ที่ส่งไปเปลี่ยนค่าไปด้วย**
- การส่ง argument ที่มีชนิดข้อมูลเป็น **แบบอ้างอิง** จะเป็นการส่งตำแหน่งอ้างอิงของอ็อบเจกต์ไปให้กับเมธอดที่ถูกเรียกใช้งาน ดังนั้น การเปลี่ยนแปลงค่าของคุณลักษณะของอ็อบเจกต์จะมีผล **ทำให้ค่าของคุณลักษณะของอ็อบเจกต์ที่ส่งไปเปลี่ยนไปด้วย**

```
int a = 10;  
int b = a;
```



String

การส่งค่าเข้าไปในเมธอดชนิดข้อมูลพื้นฐาน

```
public class Student {
    double gpa;
    public void setGPA(double GPA) {
        gpa = GPA;
    }
    public double getGPA() {
        return gpa;
    }
}

public class Main {
    public static void main(String args[]) {
        Student s1 = new Student();
        s1.setGPA(3.0);
        System.out.println(s1.gpa);
    }
}
```

การส่งค่าเข้าไปในเมธอดชนิดข้อมูลพื้นฐาน

```
public class Main {  
    public static void main(String[] args) {  
        int s = 10;  
        System.out.println(s);  
        DemoArg d1 = new DemoArg();  
        d1.changeNumber(s);  
        System.out.println(s);  
    }  
}  
  
public class DemoArg {  
    public void changeNumber(int s) {  
        s = 1000;  
        // System.out.println(s);  
    }  
}
```

s = 10

s = ~~10~~
1000

Output:
10
// 1000
10

การส่งค่าเข้าไปในเมธอดชนิดข้อมูลวัตถุ

```
public class Main {  
    public void change(Dog b) {  
        b.name = "me";  
    }  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.name = "John";  
        System.out.println(d.name); "John"  
        new Main().change(d);  
        System.out.println(d.name); "me"  
    }  
}  
  
public class Dog {  
    public String name;  
    public void printName() {  
        System.out.println(name);  
    }  
}
```



Output:
John
me

การส่งค่าเข้าไปในเมธอดชนิดข้อมูลวัตถุ



```
public class MyDate {  
    private int day, month, year;  
    public void showInfo() {  
        System.out.println(day + "/" + month + "/" + year);  
    }  
}  
  
public class Student {  
    private String name;  
    private MyDate dob;  
    public void setDOB(MyDate d) {  
        dob = d;  
    }  
}
```

การส่งค่าเข้าไปในเมธอดชนิดข้อมูลวัตถุ

```
public class Main {  
    public static void main(String args[]) {  
        Student s1 = new Student();  
        MyDate d1 = new MyDate();  
        d1.day = 16;  
        d1.month = 12;  
        d1.year = 2023;  
        s1.setDOB(d1);  
    }  
}
```

ตัวอย่าง

```

public class Main {
    public int num;
    public void addOne(int i){
        i += 1;
    }
    public void addTwo(Main m){
        m.num += 2;
    }
    → public static void main(String[] args) {
        * int i = 100;
        Main m = new Main();    i = 100

        // Step 1          i before : 100
        ① System.out.println(" i before : " + i);
        m.addOne(i); i = 101
        ② System.out.println(" i after : " + i);
           i after : 101
    }
  
```

```

// Step 2
m.num = 10;
System.out.println(" obj m before : " + m.num);
m.addTwo(m);
System.out.println(" obj m after : " + m.num);

// Step 3
Main n = m;
System.out.println(" obj m before : " + m.num);
System.out.println(" obj n before : " + n.num);
m.addTwo(n);
System.out.println(" obj m after : " + m.num);
System.out.println(" obj n after : " + n.num);

}
}
  
```

การส่งค่าเข้าไปในเมธอด (Argument)

เมธอดใด ๆ อาจมี argument สำหรับรับค่ามากกว่าหนึ่งตัว แต่การเรียกใช้เมธอดเหล่านี้จะต้องส่ง argument ที่มีชนิดข้อมูลที่สอดคล้องกันและมีจำนวนเท่ากัน

```
public class NumericalSample {  
    public void calMax(int i, double d) {  
        if (i > d) {  
            System.out.println("Max = "+i);  
        } else {  
            System.out.println("Max = "+d);  
        }  
    }  
    public static void main(String args[]) {  
        NumericalSample obj = new NumericalSample();  
        obj.calMax(3,4.0);  
    }  
}
```

การส่งค่าเข้าไปในเมธอด (Argument)



ชนิดข้อมูลของ argument ที่จะส่งผ่านไปยังเมธอด ไม่จำเป็นที่จะต้องเป็นชนิดข้อมูลเดียวกัน แต่ต้องเป็นชนิดข้อมูลที่สามารถแปลงข้อมูลให้กว้างขึ้นได้โดยอัตโนมัติ

```
public class NumericalSample {  
    public void calMax(int i, double d) {  
        if (i > d) {  
            System.out.println("Max = "+i);  
        } else {  
            System.out.println("Max = "+d);  
        }  
    }  
    public static void main(String args[]) {  
        NumericalSample obj = new NumericalSample();  
        obj.calMax(3, 4);  
    }  
}  
Max = 4.0
```


การส่งค่าเข้าไปในเมธอด (Argument)

ชนิดข้อมูลของ argument ที่จะส่งผ่านไปยังเมธอด ไม่จำเป็นที่จะต้องเป็นชนิดข้อมูลเดียวกัน แต่ต้องเป็นชนิดข้อมูลที่สามารถแปลงข้อมูลให้กว้างขึ้นได้โดยอัตโนมัติ

```
public class NumericalSample {  
    public void calMax(int i, double d) {  
        if (i > d) {  
            System.out.println("Max = "+i);  
        } else {  
            System.out.println("Max = "+d);  
        }  
    }  
    public static void main(String args[]) {  
        NumericalSample obj = new NumericalSample();  
        obj.calMax(3.0, 4.0);  
    }  
}
```

```
Main.java:11: error: incompatible types: possible lossy conversion from double to int  
        obj.calMax(3.0, 4.0);
```

การรับค่าที่ส่งกลับมาจากเมธอด

- เมธอดใดๆของคลาสสามารถที่จะมีค่าที่ส่งกลับมาได้ ซึ่งชนิดข้อมูลของค่าที่จะส่งกลับอาจเป็นชนิดข้อมูลแบบพื้นฐาน หรือเป็นชนิดข้อมูลแบบอ้างอิง
- เมธอดที่มีค่าที่จะส่งกลับมาจะต้องมีคำสั่ง **return** ซึ่งจะระบุค่าที่ส่งกลับโดยมีรูปแบบดังนี้

```
return value;
```

- คำสั่งที่เรียกใช้เมธอด อาจจะได้รับค่าที่ส่งกลับมาเก็บไว้ในตัวแปรหรือเป็นตัวถูกดำเนินการในนิพจน์ ตัวอย่างเช่น

```
Student s1 = new Student();  
s1.setGPA(3.2);  
double d = s1.getGPA();  
System.out.println("GPA:" + d);
```

1. ใน 1 method จะใช้ return ก็ได้
- 2.
3. เมื่อ return จนจบ method นั้นทันที

```
public class Student {  
    private String id, name;  
    private double gpa;  
    public void setGPA(double GPA) {  
        gpa = GPA;  
    }  
    public double getGPA() {  
        return gpa;  
    }  
}
```

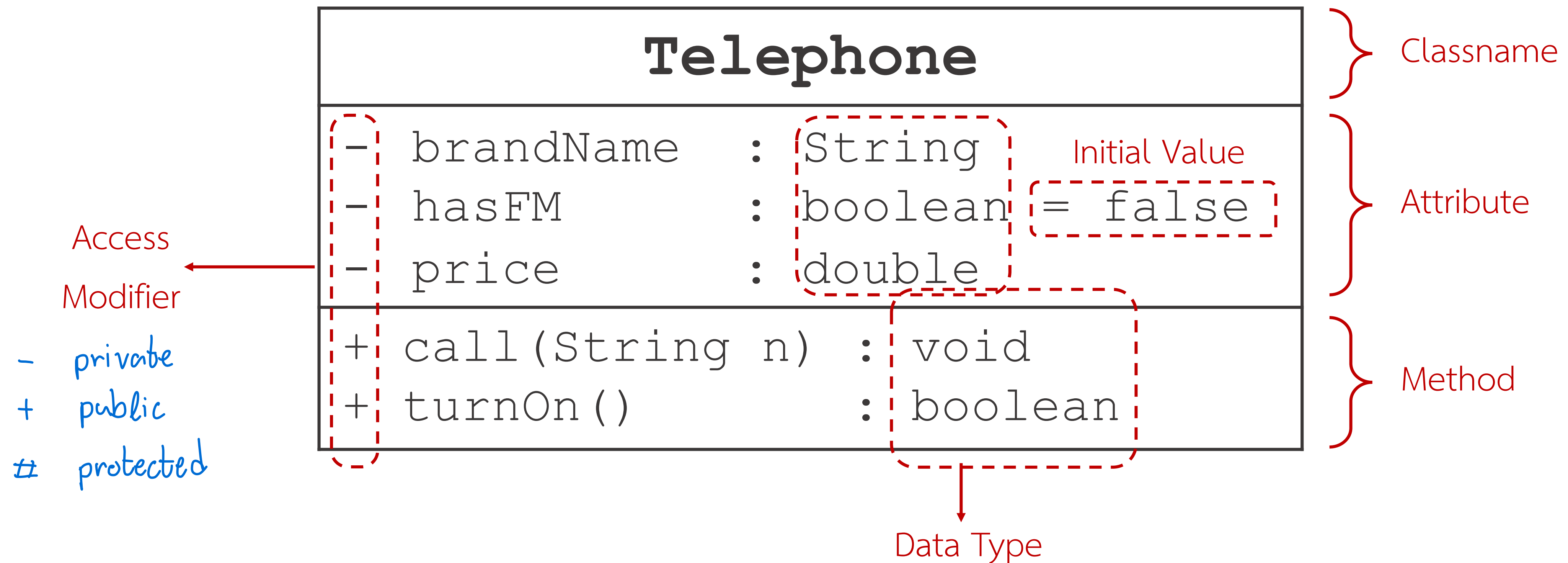
Unified Modeling Language (UML)



Unified Modeling Language (UML) เป็นภาษาที่สามารถนำรูปภาพฟลักมาจำลองโปรแกรมเชิงอ็อบเจกต์ได้ ซึ่งประกอบด้วย 2 ส่วนคือ

- ไดอะแกรมของคลาส (Class Diagram)
- ไดอะแกรมของอ็อบเจกต์ (Object Diagram)

ไดอะแกรมของคลาส



ไดอะแกรมของคลาส คือ เป็นสัญลักษณ์ที่ใช้แสดงรายละเอียดของคลาส ซึ่งประกอบด้วย 3 ส่วนหลัก ได้แก่ (1) ชื่อของคลาส (2) คุณลักษณะภายในคลาส และ (3) เมธอดภายในคลาส

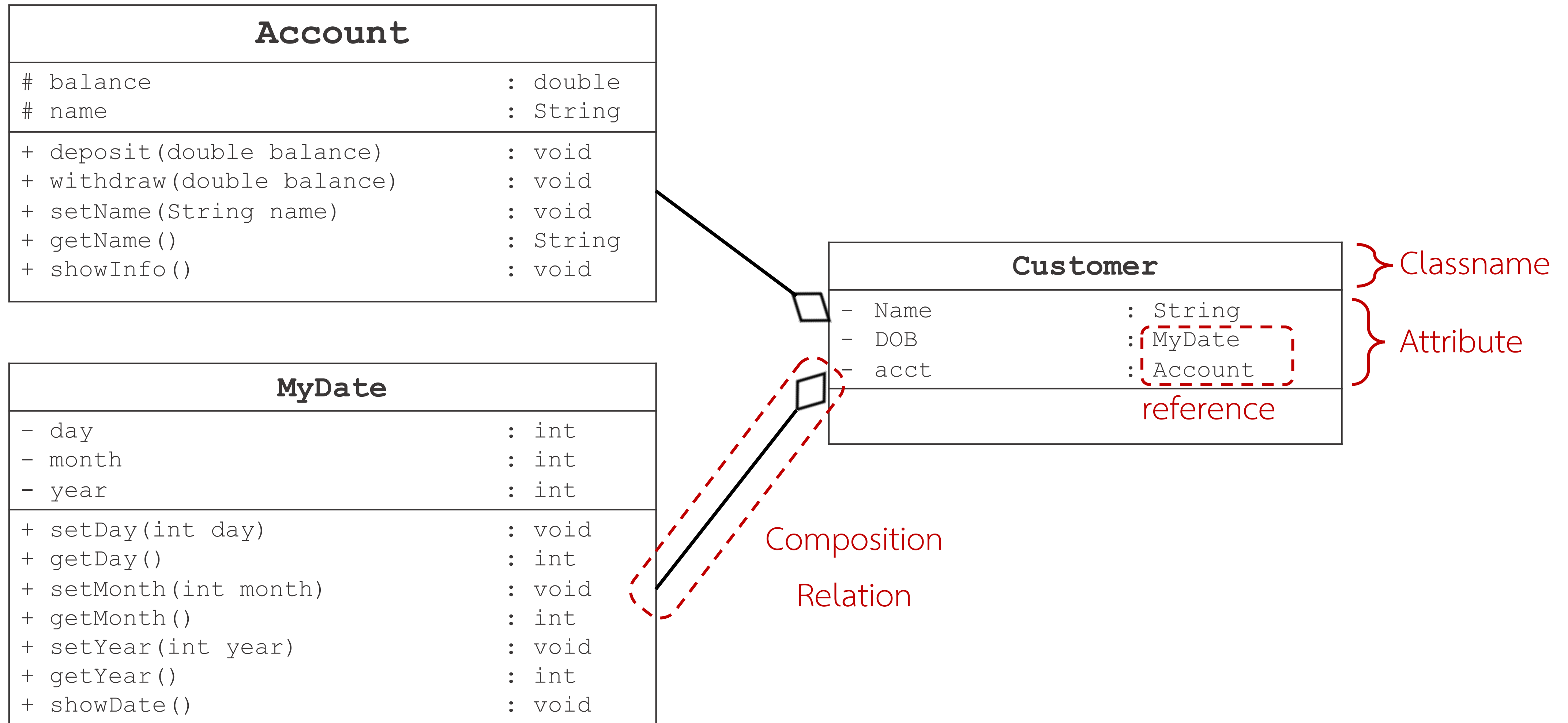
คลาสไดอะแกรม

Telephone	
-	brandName : String
-	hasFM : boolean
-	price : double
+	call(String n) : void
+	turnOn() : boolean



```
public class Telephone {  
    private String brandName;  
    private boolean hasFM;  
    private double price;  
  
    public void call(String n) {  
        ...  
    }  
    public boolean turnOn() {  
        ...  
    }  
}
```


คลาสไดอะแกรม



ชนิดข้อมูลนามธรรม

ชนิดข้อมูลแบบนามธรรม (Abstract Data Type) หมายถึง
ชนิด หรือ คลาสของวัตถุที่ถูกนิยามด้วยเซตของ
ข้อมูลและตัวดำเนินการ ซึ่งเป็นการนิยามหลักการทำงานและ
พฤติกรรมของข้อมูลเท่านั้น แต่ไม่ระบุถึงวิธีการสร้างหรือ
โปรแกรมเพื่อใช้งานจริง ดังนั้นชนิดข้อมูลดังกล่าวจึงถูก
เรียกว่า นามธรรม

Type

5 3 1
7 9 0 2
4

Operations

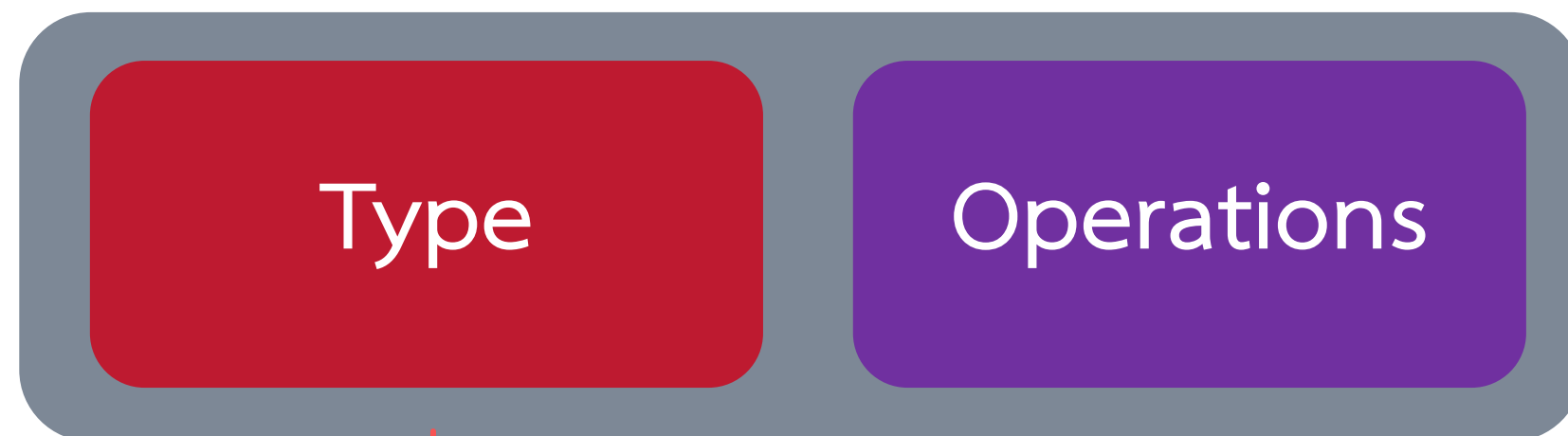
× ÷ + −

Object Oriented Programming

vs Data Structure

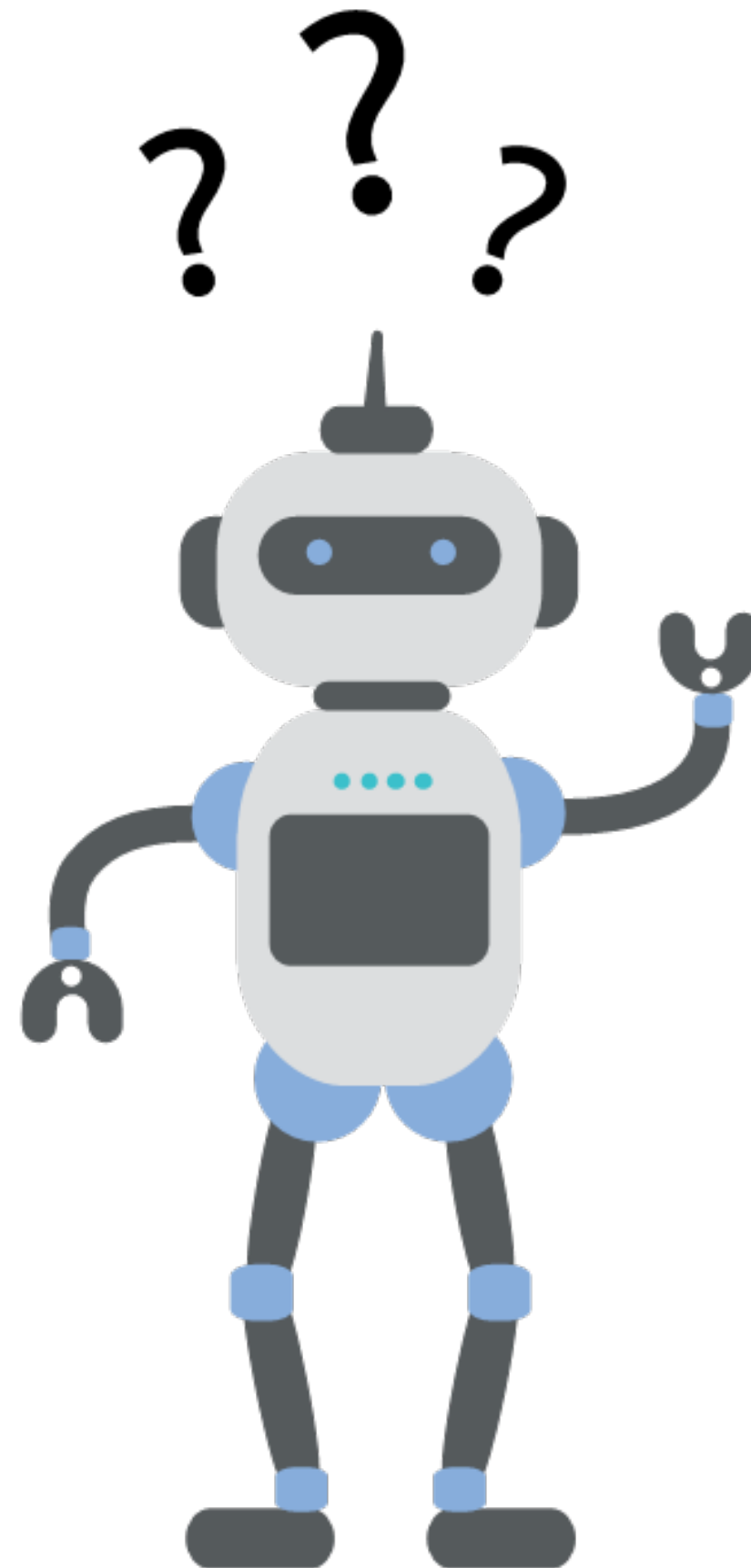


Abstract Data Type



value

Data Structure



Class



Object Oriented
Programming

ชนิดข้อมูล

Abstract Data Type

Type

Operations

ชนิดข้อมูล

Abstract Data Type

Type

Operations

Integer

$+$, $-$, \times , \div

Floating
point

$+$, $-$, \times , \div

“..., -3, -2, -1, 0, 1, 2, 3, ...”

“..., -4.3, -2.3, -1.0, 0.1, 1.0, 1.2, 2.3, ...”

ชนิดข้อมูล

Abstract Data Type

Type

Operations

Integer

$+$, $-$, \times , \div

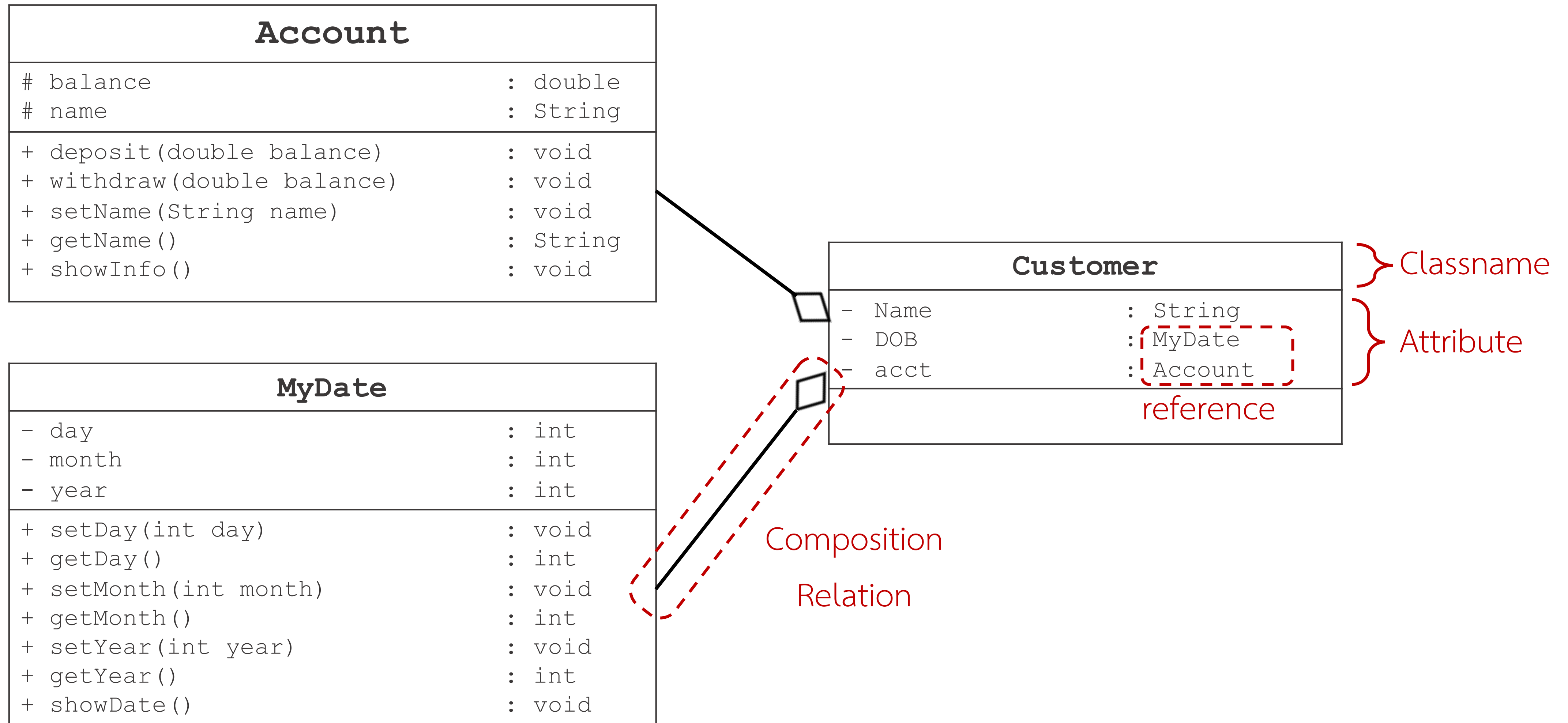
Floating
point

$+$, $-$, \times , \div

Character

...

คลาสไดอะแกรม



ตัวอย่างการใช้งาน Composition และ Reference

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
    ...  
}  
  
public class Account {  
    protected String name;  
    protected double balance;  
    ...  
}  
  
public class Customer {  
    private String name;  
    private MyDate dob;  
    private Account acct;  
}
```

Account	
# balance	: double
# name	: String
+ deposit(double balance)	: void
+ withdraw(double balance)	: void
+ setName(String name)	: void
+ getName()	: String
+ showInfo()	: void

MyDate	
- day	: int
- month	: int
- year	: int
+ setDay(int day)	: void
+ getDay()	: int
+ setMonth(int month)	: void
+ getMonth()	: int
+ setYear(int year)	: void
+ getYear()	: int
+ showDate()	: void

Customer	
- Name	: String
- DOB	: MyDate
- acct	: Account



โครงสร้างของอ็อบเจกต์

ประกอบไปด้วย 2 ส่วน คือ (1) ส่วนที่ระบุชื่อของอ็อบเจกต์ และ (2) ส่วนที่ระบุค่าของคุณลักษณะภายในอ็อบเจกต์

