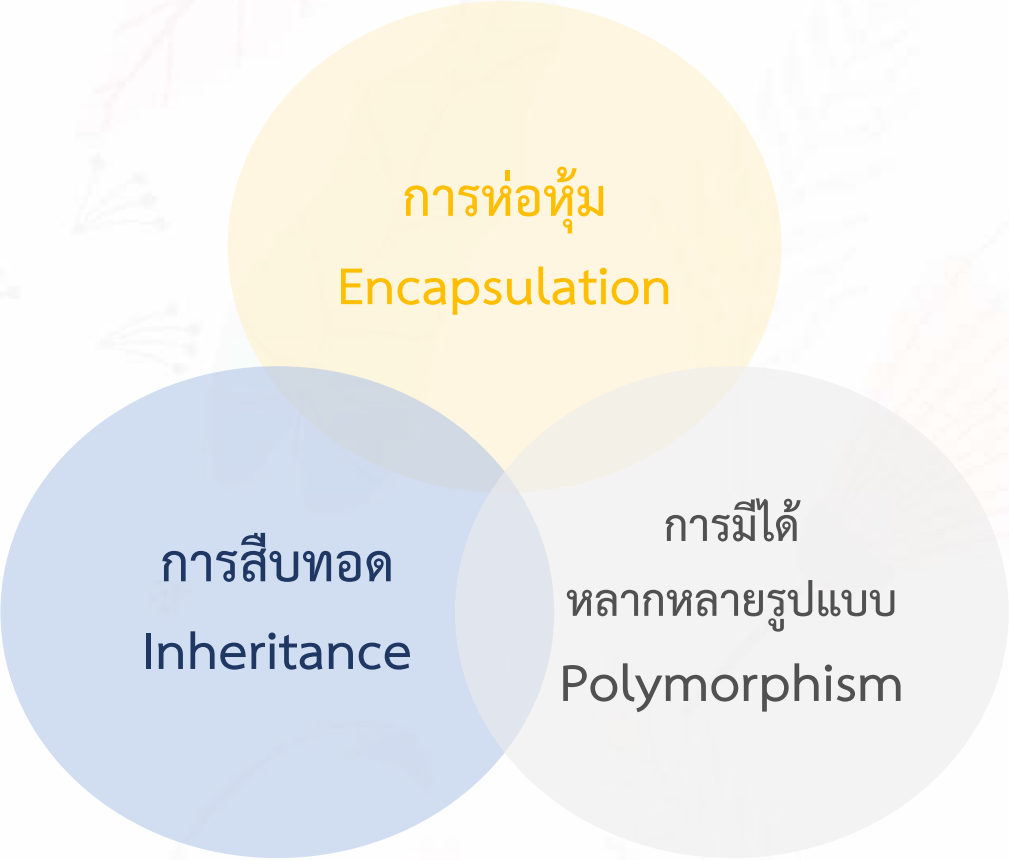
บทที่ 5

หลักการห่อหุ้มและสืบทอด

บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์

การเขียนโปรแกรมเชิงวัตถุ



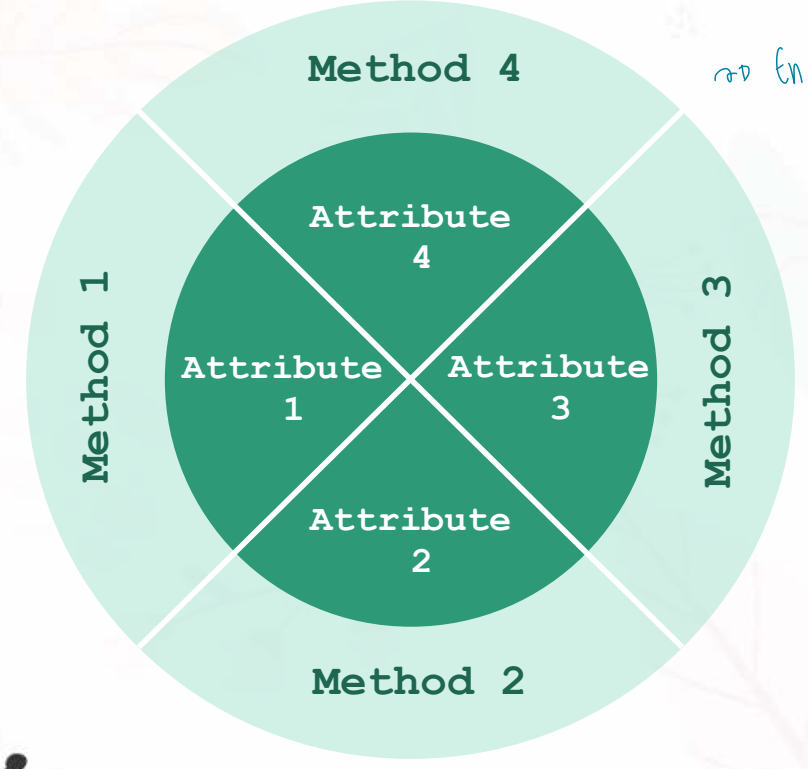
การเขียนโปรแกรมเชิงวัตถุ

การห่อหุ้ม
Encapsulation

การสืบทอด
Inheritance

การมีได้
หลากหลายรูปแบบ
Polymorphism

การห่อหุ้ม (Encapsulation)



→ Encapsulation Attribute ด้วย Method (มี Method เป็นตัวกลางระหว่าง User , Attribute

การห่อหุ้ม คือ กระบวนการปกปิดโค้ดและข้อมูลสำหรับการจัดการ หรือสามารถกล่าวได้ว่า กระบวนการห่อหุ้ม เป็นวิธีการปกป้องข้อมูลจากการเข้าถึงจากภายนอกหรือโปรแกรมอื่น ๆ (Data-Hiding)

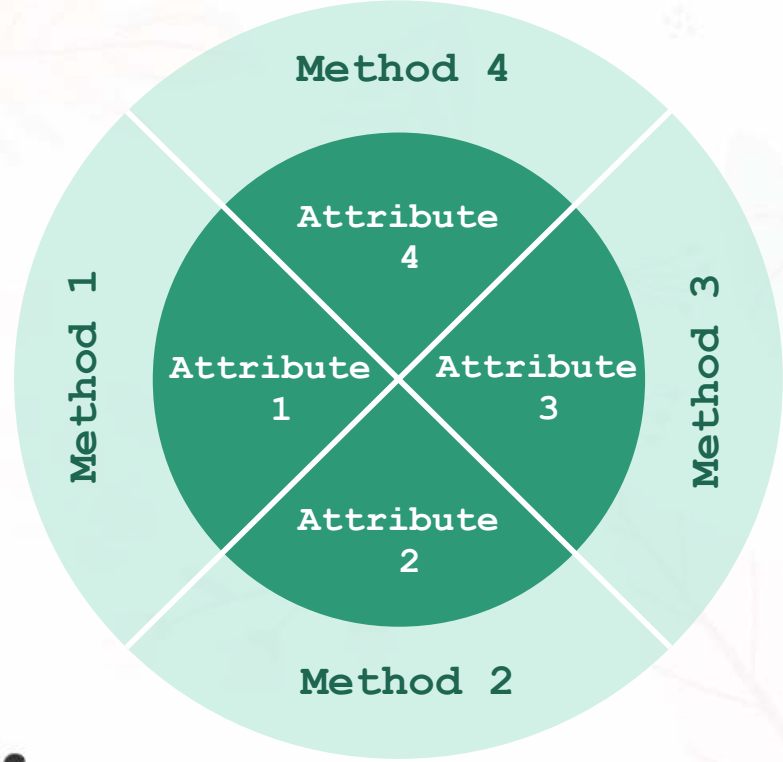
↳ ไม่สามารถเข้าถึงข้อมูลได้โดยตรง ต้องเข้าผ่าน Method

การห่อหุ้ม (Encapsulation)

ทำไมนักศึกษาถึงควรใช้งานหลักการห่อหุ้มแอททริบิวท์ในการเขียนโปรแกรมเชิงวัตถุ

- ความเป็นส่วนตัวและสามารถป้องกันวัตถุอื่นเข้ามาทำลายคุณสมบัติบางอย่าง
  - เลขบัญชีธนาคารติดลบแทนที่จะเป็นเลขศูนย์หรือเลขบวก
  - ความกว้างและความสูงของสี่เหลี่ยมน้อยกว่าหรือเท่ากับศูนย์
- ทำให้สามารถเปลี่ยนแปลงชนิดของตัวแปรได้ โดยไม่กระทบกระเทือนต่อคลาสอื่นที่เรียกใช้งาน
- สามารถเปลี่ยนแปลงวิธีการทำงานของวัตถุได้โดยไม่กระทบต่อผู้ใช้วัตถุ
- เกิดความยืดหยุ่น , ปลอดภัยมากขึ้น
  - เช่น ถ้าเปรียบวัตถุเป็นรถยนต์ และผู้ใช้วัตถุคือคนขับ บริษัทผู้ผลิตรถยนต์สามารถเปลี่ยนเครื่องยนต์รุ่นใหม่ เปลี่ยนระบบเบรค ฯลฯ โดยที่คนขับยังสามารถขับรถได้เหมือนเดิม

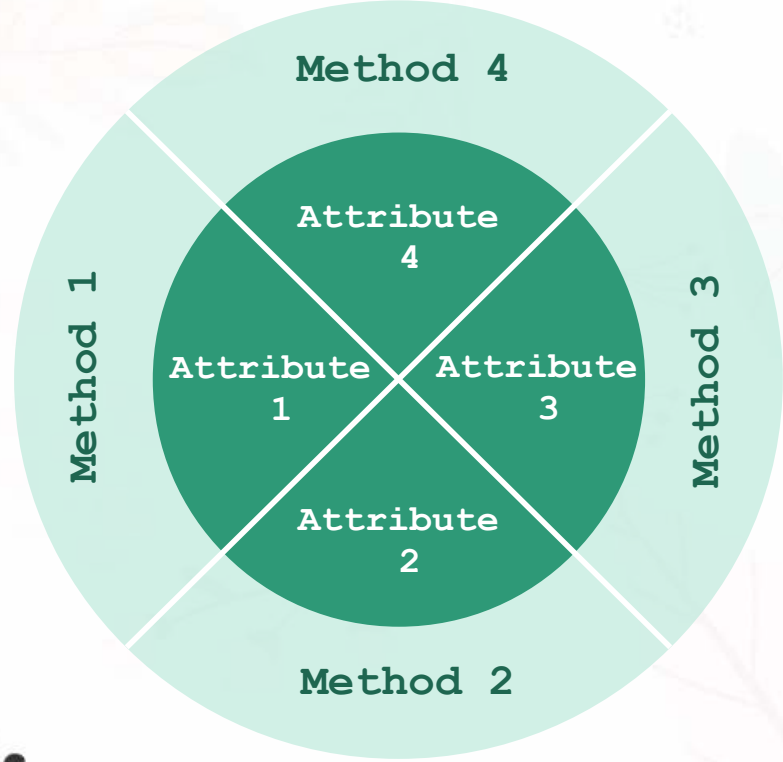
การห่อหุ้ม (Encapsulation)



การใช้งานหลักการห่อหุ้มแอททริบิวท์ในการเขียนโปรแกรมเชิงวัตถุสามารถทำได้ด้วย 2 ขั้นตอนดังนี้

- กำหนดสิทธิการเข้าถึง (Access Modifier) แอททริบิวท์เป็น private
- สร้างเมธอดเพื่อใช้เป็นตัวกลางระหว่างผู้เรียกใช้งานกับแอททริบิวท์และกำหนดสิทธิการเข้าถึงเมธอดเป็น public ซึ่งเมธอดเหล่านี้จะเรียกว่าเมธอด setter และ เมธอด getter

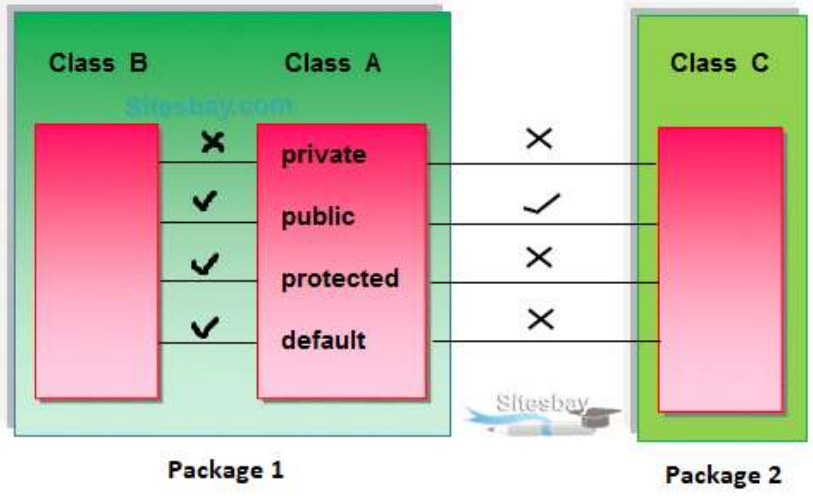
การห่อหุ้ม (Encapsulation)



การใช้งานหลักการห่อหุ้มแอททริบิวท์ในการเขียนโปรแกรมเชิงวัตถุสามารถทำได้ด้วย 2 ขั้นตอนดังนี้

- กำหนดสิทธิการเข้าถึง (Access Modifier) แอททริบิวท์เป็น private
- สร้างเมธอดเพื่อใช้เป็นตัวกลางระหว่างผู้เรียกใช้งานกับแอททริบิวท์และกำหนดสิทธิการเข้าถึงเมธอดเป็น public ซึ่งเมธอดเหล่านี้จะเรียกว่าเมธอด setter และ เมธอด getter

การกำหนดสิทธิการเข้าถึง (Access Modifier)



การกำหนดสิทธิการเข้าถึง (Access Modifier) เป็นคำสำคัญที่ใช้กำหนดระดับการเข้าถึง (accessibility) ของคลาส (class), เมธอด (method), และแอททริบิวท์ (attribute) ในการเขียนโปรแกรมและการออกแบบโปรแกรมในรูปแบบ Object-Oriented Programming (OOP) โดยมี 4 ระดับหลัก ได้แก่ public, protected, default และ private

การใช้ access modifier เป็นส่วนสำคัญใน

- ช่วยจัดการความเป็นส่วนตัวของเมธอดและแอททริบิวท์
- ช่วยสร้างการเชื่อมโยงระหว่างคลาส
- ช่วยเพิ่มความปลอดภัยของโปรแกรม

การกำหนดสิทธิการเข้าถึง (Access Modifier)

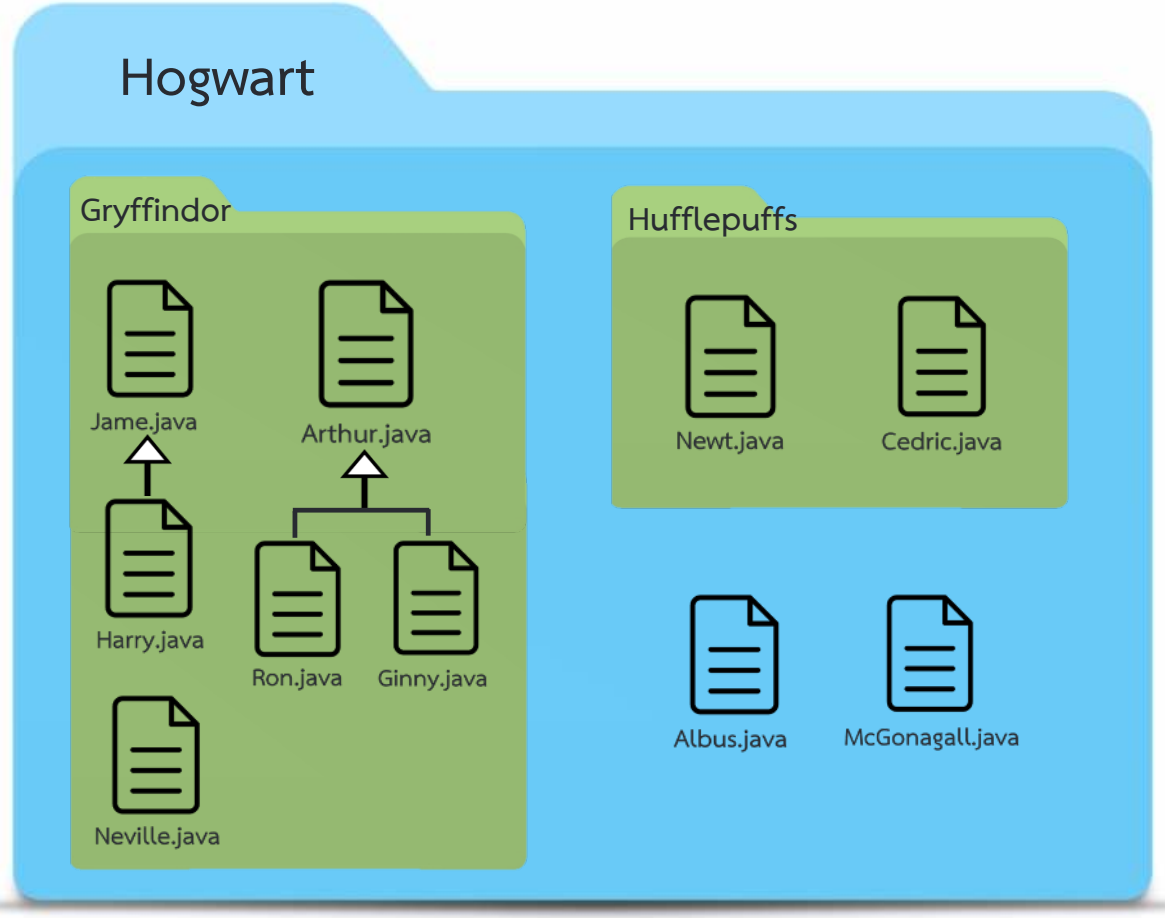
1. **public** เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น public จะหมายความว่า สามารถเข้าถึงได้จากทุกที่ในโปรเจค ไม่ว่าจะอยู่ในแพ็กเกจเดียวกันหรือแพ็กเกจต่างกันที่อยู่ในโปรเจคเดียวกัน public ถือว่าเป็นระดับการเข้าถึงที่กว้างที่สุดและสามารถใช้ในกรณีที่คุณต้องการให้สามารถเรียกใช้ได้จากที่ใดก็ได้

2. **protected** เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น protected จะหมายความว่า สามารถเข้าถึงได้จากคลาสที่อยู่ในแพ็กเกจเดียวกันหรือสามารถคลาสที่ได้รับการสืบทอด (inheritance) เท่านั้น protected ถือว่าเป็นระดับการเข้าถึงที่มีความจำกัดมากกว่า public แต่ยังคงเปิดให้คลาสที่สืบทอดสามารถเข้าถึงได้

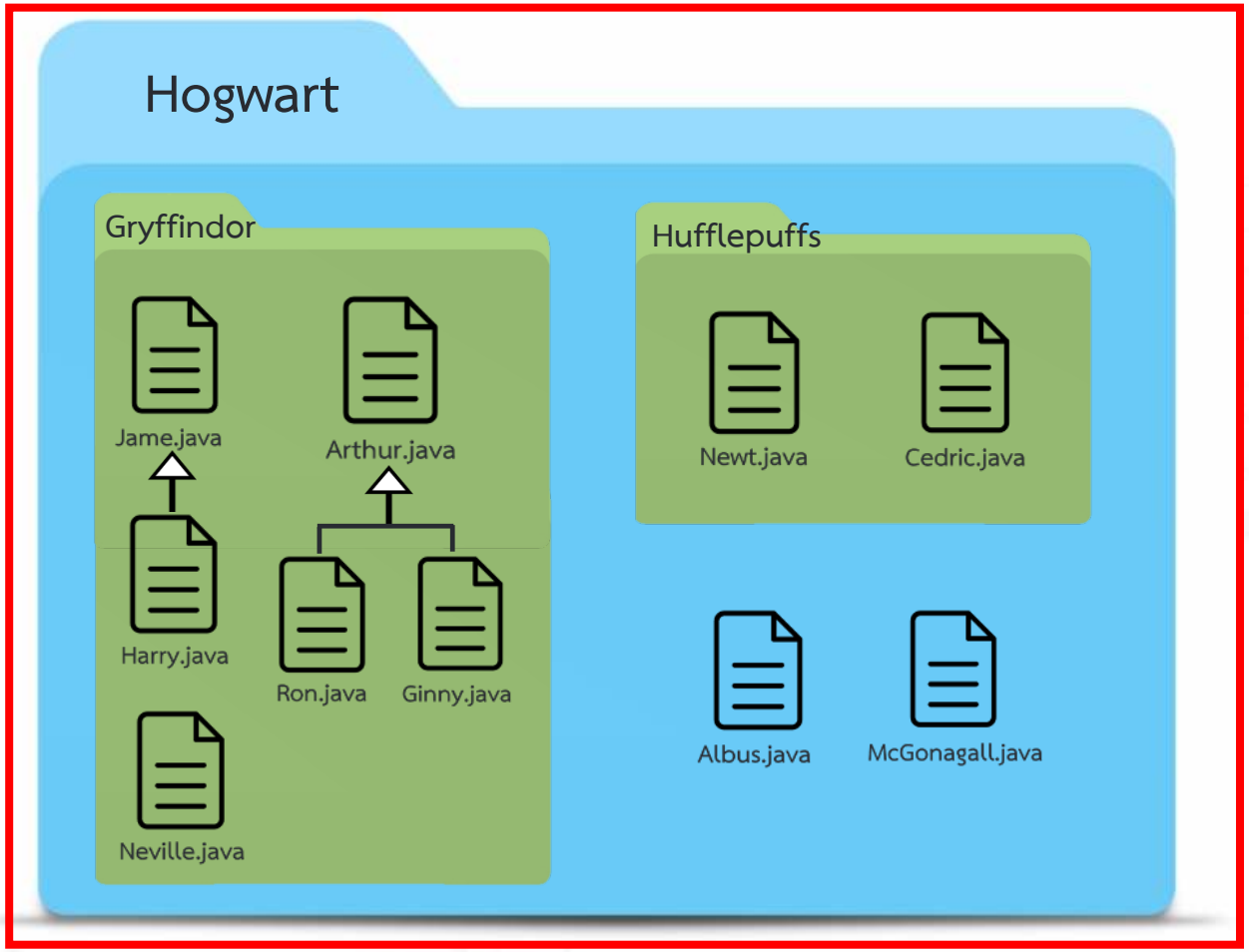
3. **default** (ไม่มี access modifier) เมื่อคลาส เมธอด หรือแอททริบิวท์ไม่มีการระบุ access modifier จะถือว่ามี access modifier เป็น default หรือหรือเรียกว่า “package private” ซึ่งหมายความว่า สามารถเข้าถึงได้เฉพาะในแพ็กเกจเดียวกันเท่านั้น และไม่สามารถเรียกใช้จากแพ็กเกจอื่นได้

4. **private** เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น private จะหมายความว่า สามารถเข้าถึงได้เฉพาะจากภายในคลาสเท่านั้น ไม่สามารถเข้าถึงจากภายนอกคลาสได้

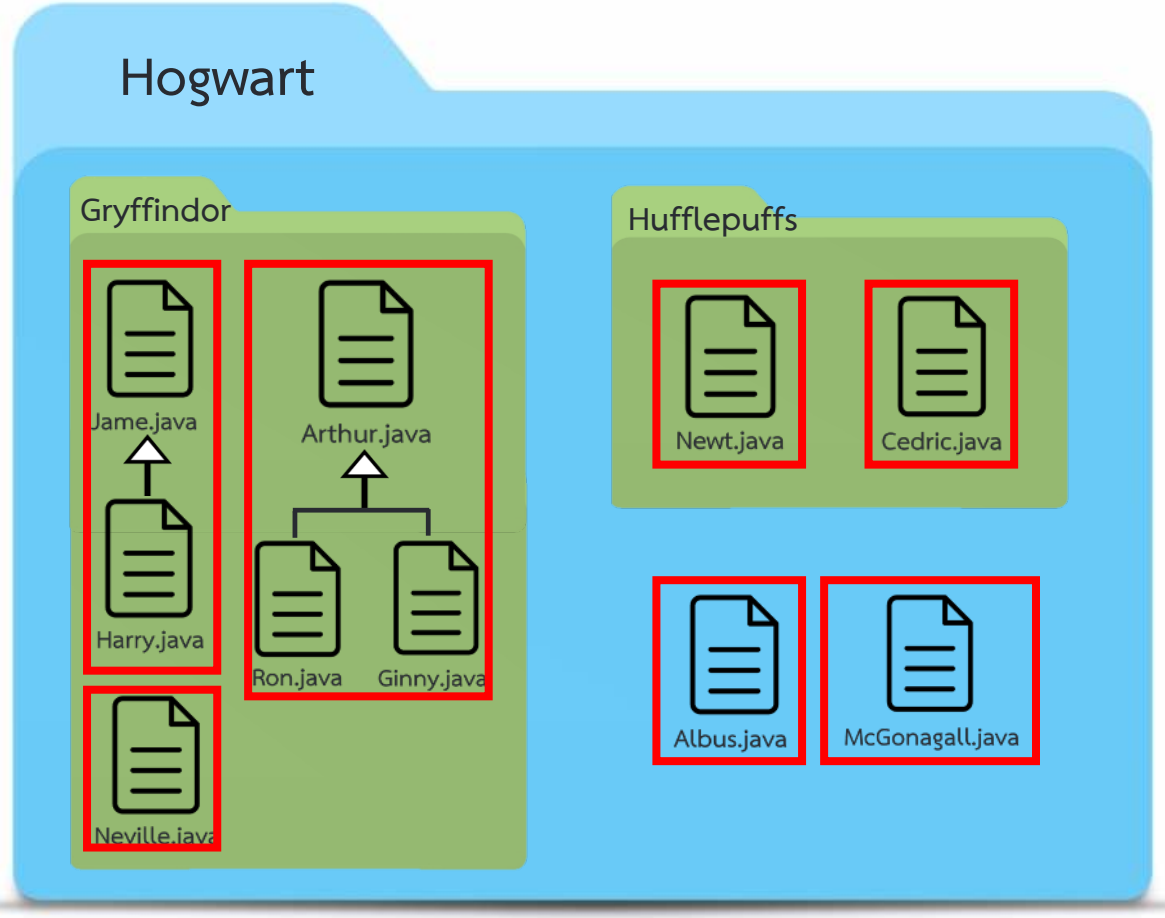
การกำหนดสิทธิการเข้าถึง (Access Modifier)



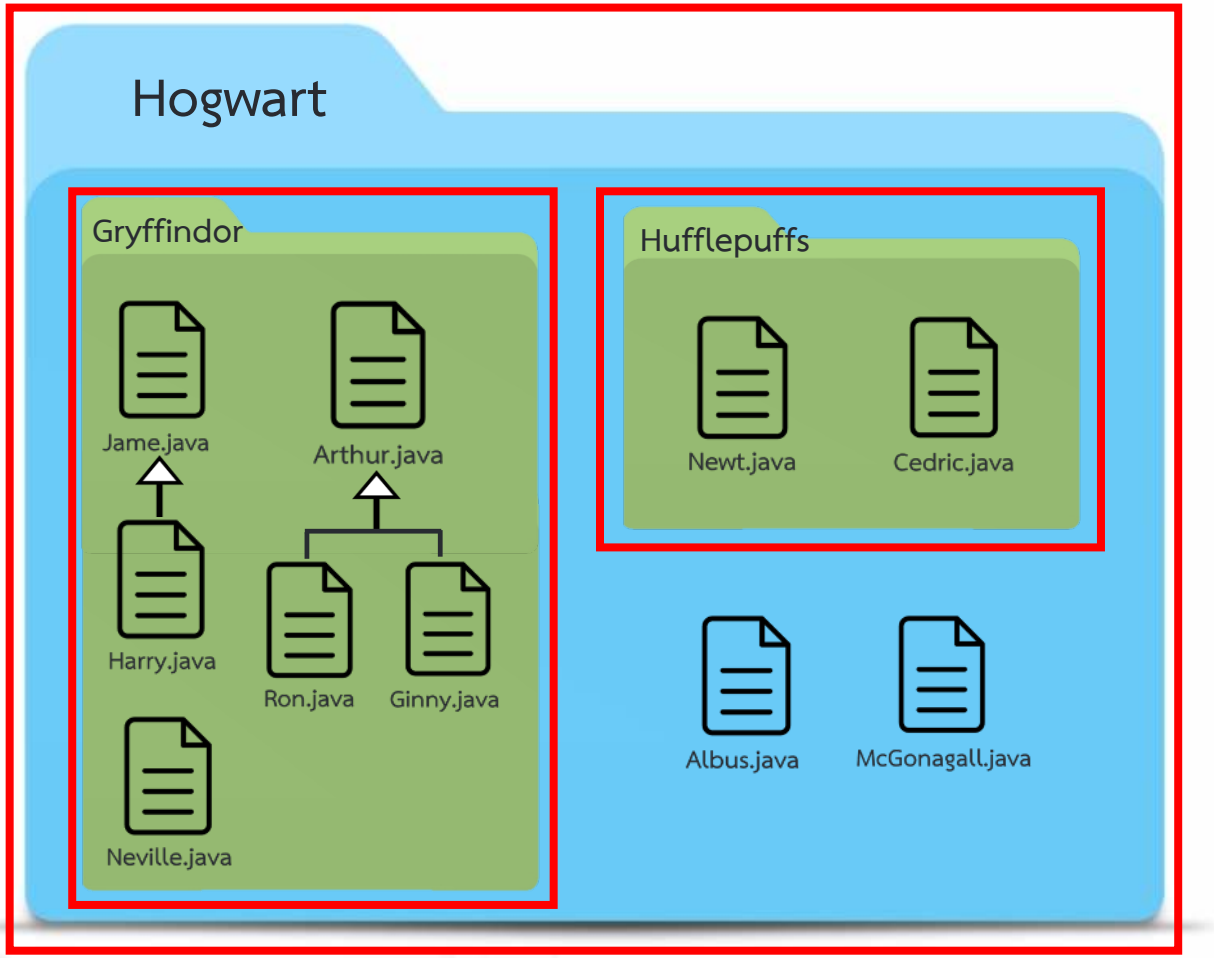
การกำหนดสิทธิการเข้าถึงแบบ public



การกำหนดสิทธิการเข้าถึงแบบ protected

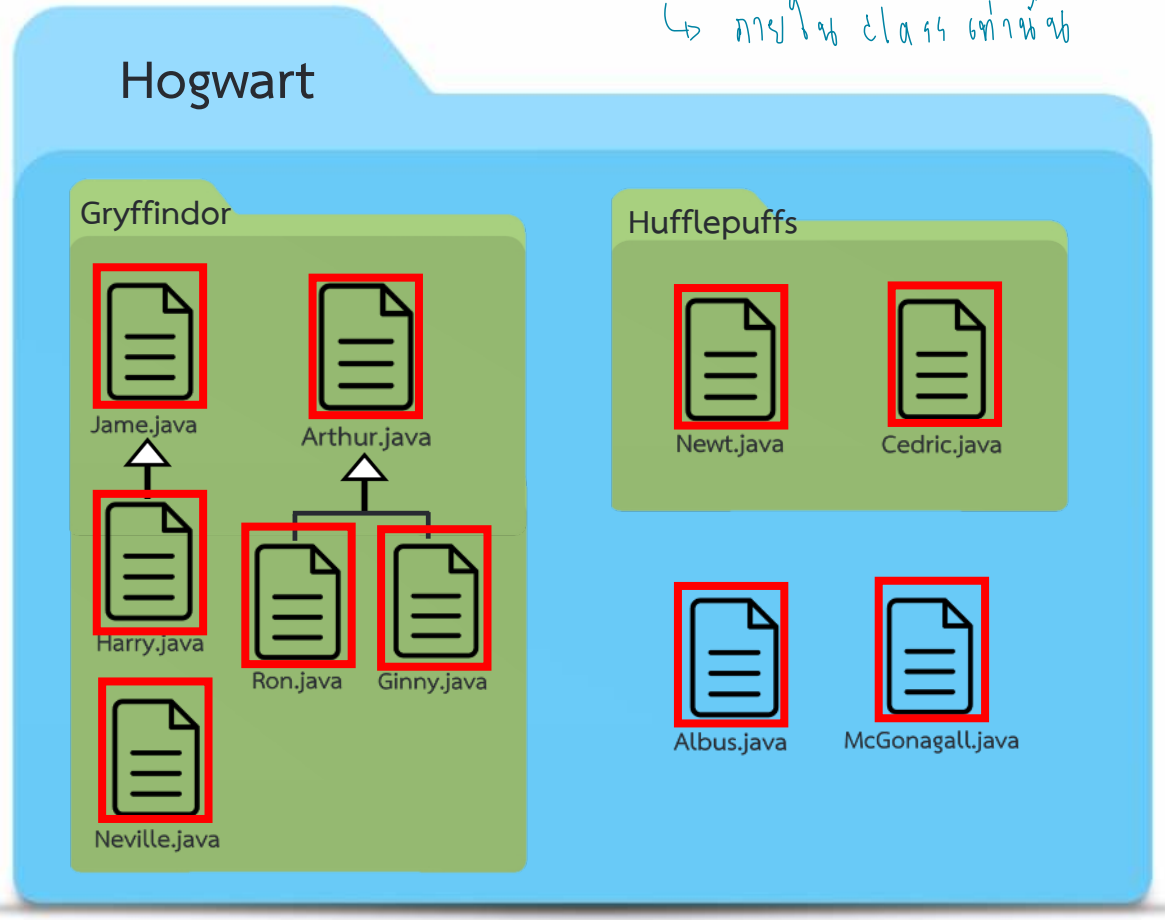


การกำหนดสิทธิการเข้าถึงแบบ default



การกำหนดสิทธิการเข้าถึงแบบ private

↳ ภายใน class เท่านั้น



มีผลต่อ class เท่านั้น

การกำหนดสิทธิการเข้าถึง (Access Modifier)

1 **public** เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น public จะหมายความว่า สามารถเข้าถึงได้จากทุกที่ในโปรเจค ไม่ว่าจะอยู่ในแพ็กเกจเดียวกันหรือแพ็กเกจต่างกันที่อยู่ในโปรเจคเดียวกัน public ถือว่าเป็นระดับการเข้าถึงที่กว้างที่สุดและสามารถใช้ในกรณีที่คุณต้องการให้สามารถเรียกใช้ได้จากที่ใดก็ได้

2 **protected** เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น protected จะหมายความว่า สามารถเข้าถึงได้จากคลาสที่อยู่ในแพ็กเกจเดียวกันหรือสามารถคลาสที่ได้รับการสืบทอด (inheritance) เท่านั้น protected ถือว่าเป็นระดับการเข้าถึงที่มีความจำกัดมากกว่า public แต่ยังคงเปิดให้คลาสที่สืบทอดสามารถเข้าถึงได้

3 **default** (ไม่มี access modifier) เมื่อคลาส เมธอด หรือแอททริบิวท์ไม่มีการระบุ access modifier จะถือว่ามี access modifier เป็น default หรือหรือเรียกว่า “package private” ซึ่งหมายความว่า สามารถเข้าถึงได้เฉพาะในแพ็กเกจเดียวกันเท่านั้น และไม่สามารถเรียกใช้จากแพ็กเกจอื่นได้

4 **private** เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น private จะหมายความว่า สามารถเข้าถึงได้เฉพาะจากภายในคลาสเท่านั้น ไม่สามารถเข้าถึงจากภายนอกคลาสได้

ตัวอย่าง Access Modifier ที่ 1

```
public class Rectangle {
    private double width;
    private double height;
    public double getArea() {
        return width * height;
    }
}

public class Main {
    public static void main( String[] args) {
        Rectangle rect = new Rectangle();
        rect.width = 10;
    }
}
```

① Attribute ทุกตัว ต้องกำหนด Access Modifier เป็น private

ตามหลัก encapsulation !

ตัวอย่าง Access Modifier ที่ 1

```
public class Rectangle {
    private double width;
    private double height;
    public double getArea() {
        return width * height;
    }
}

public class Main {
    public static void main( String[] args) {
        Rectangle rect = new Rectangle();
        rect.width = 10;
    }
}
```

เรียกใช้งานไม่ได้
เพราะเป็น private
อยู่คนละ class

```
C:\WINDOWS\system32\cmd.exe

C:\Java\ch08\02>javac Rectangle.java

C:\Java\ch08\02>javac TestPrivate.java
TestPrivate.java:5: width has private access in Rectangle
        rect.width = 10;
            ^
1 error

C:\Java\ch08\02>_
```

ตัวอย่าง Access Modifier ที่ 2

```
public class Ant{
  private String name;
  private int age;
  public void greeting(){
    System.out.println(name);
  }
}

public class Main{
  public static void main(String[] args) {
    Ant a1 = new Ant();
    a1.greeting();  # เรียกใช้ method ได้ เพราะ method เป็น public
  }
}
```

null

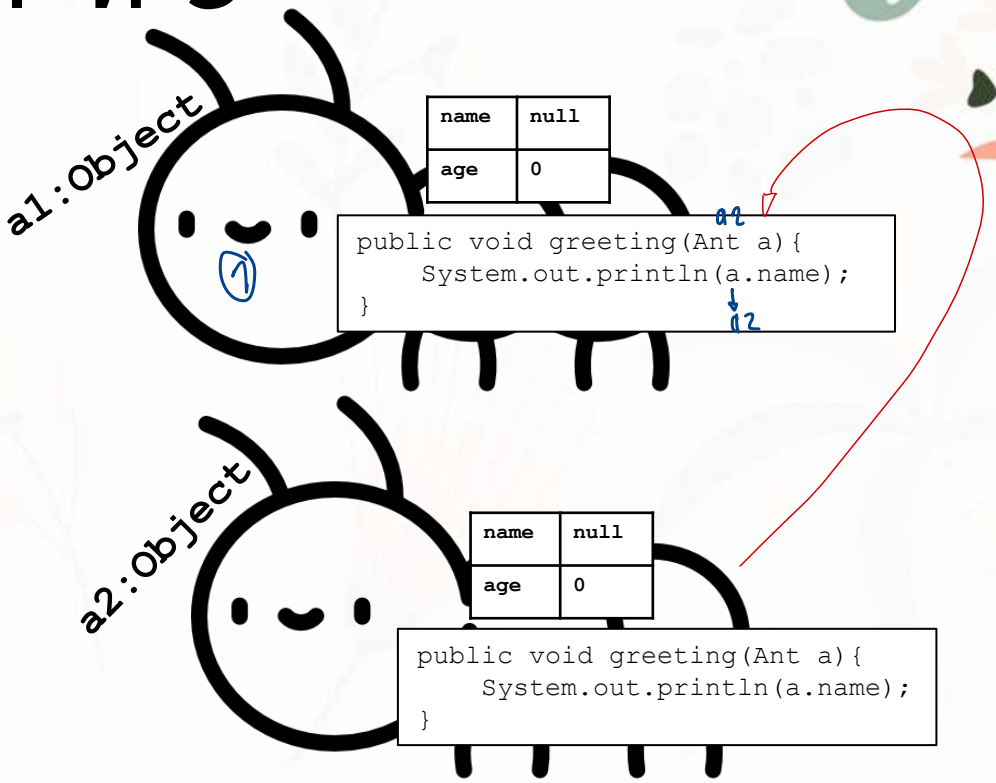
ตัวอย่าง Access Modifier ที่ 3

```
public class Ant{
  private String name;
  private int age;
  public void greeting(Ant a){
    System.out.println(a.name);
  }
}

public class Main{
  public static void main(String[] args) {
    Ant a1 = new Ant();
    Ant a2 = new Ant();

    a1.greeting(a2);
  }
}
```

input

a1 เรียกใช้ method greeting โดยใส่ a2 เข้า parameter

null

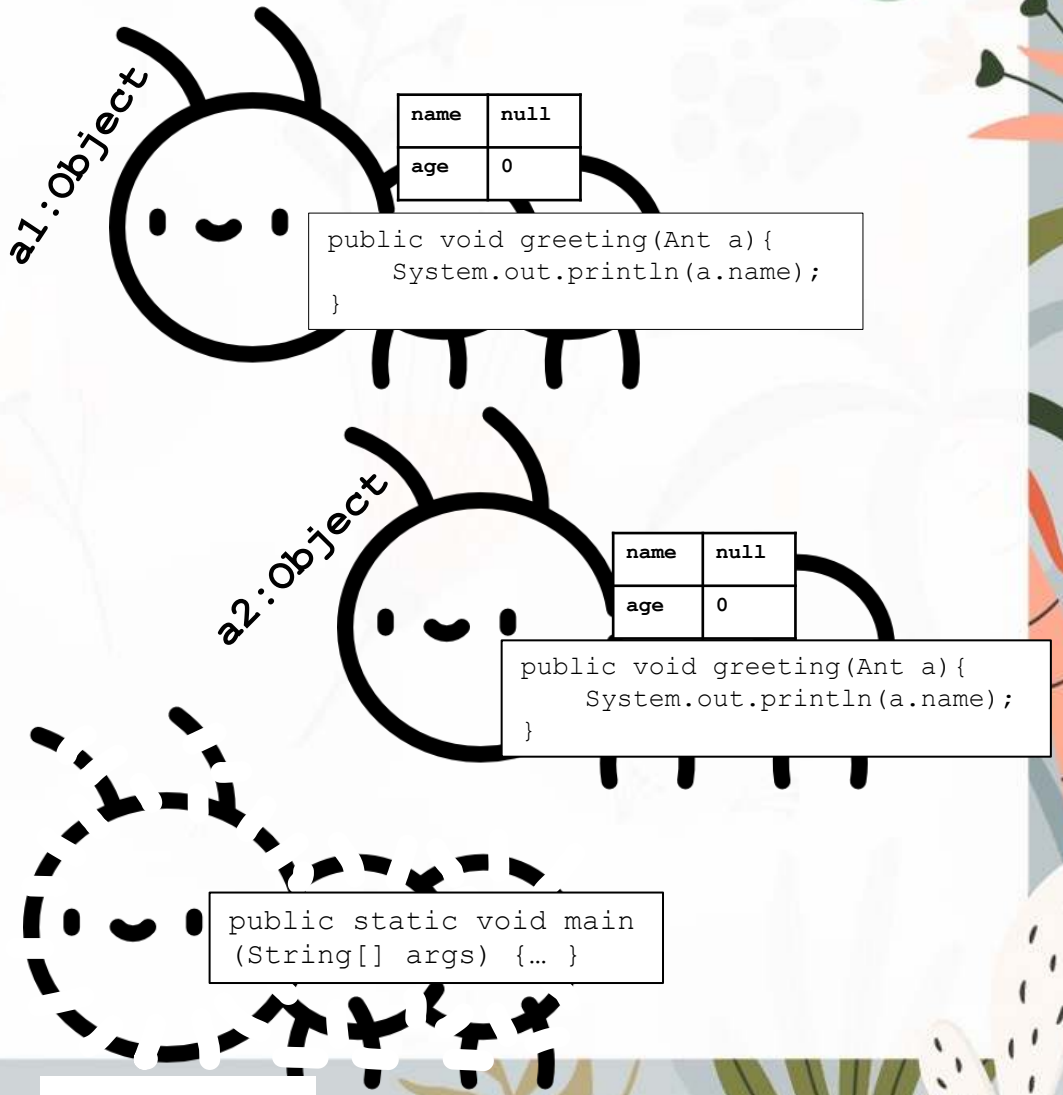


ตัวอย่าง Access Modifier ที่ 4

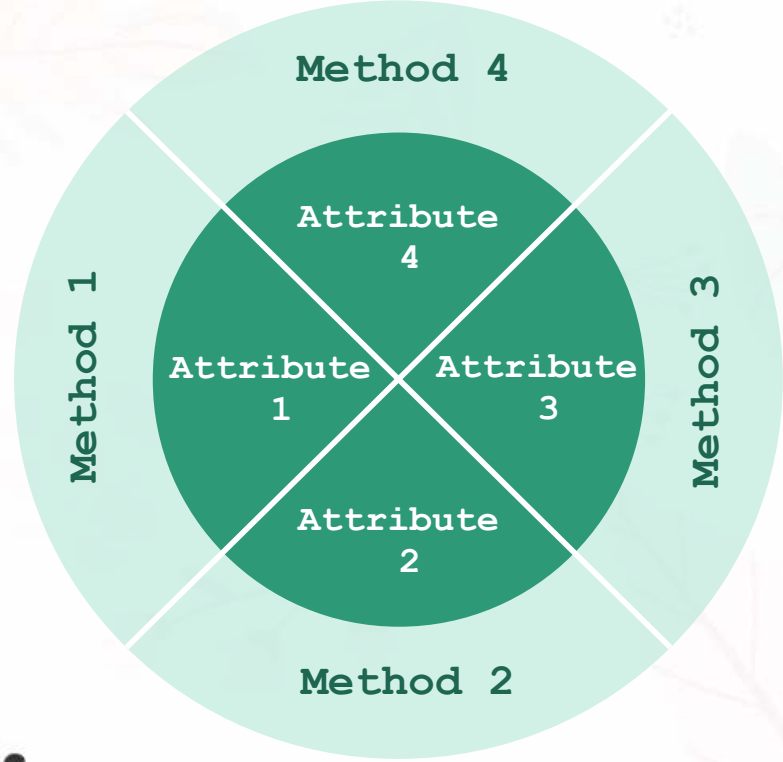
```
public class Ant{
  private String name;
  private int age;
  public void greeting(Ant a){
    System.out.println(a.name);
  }
  public static void main(String[] args) {
    Ant a1 = new Ant();
    a1.name = "John";
    Ant a2 = new Ant();
    a2.name = "Alex";

    a1.greeting(a2);
    System.out.println(a1.name);
  }
}
```

ต้องใช้ name ได้ เพราะอยู่ใน class เดียวกัน

Alex
John



การห่อหุ้ม (Encapsulation)

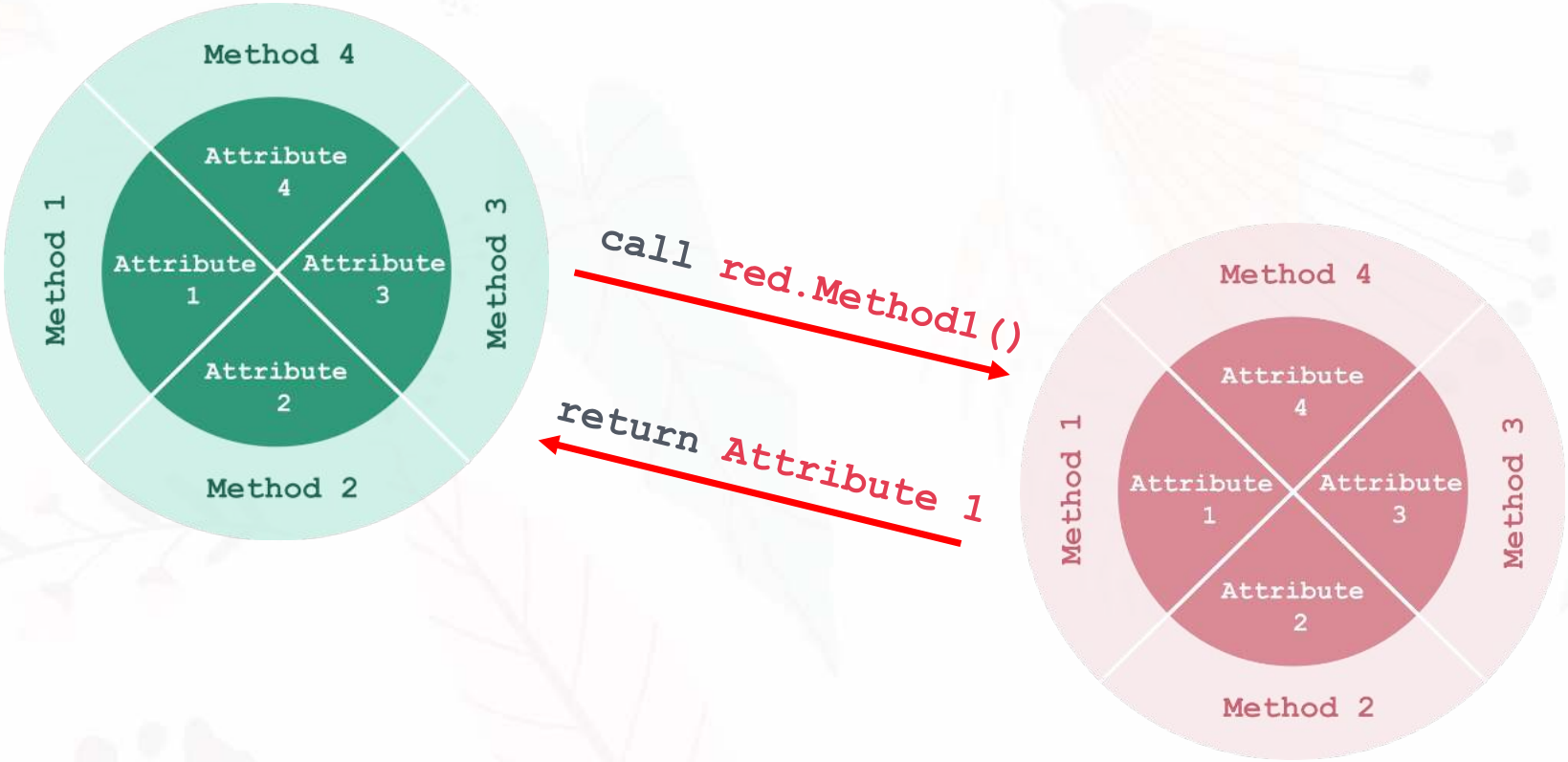


การใช้งานหลักการห่อหุ้มแอททริบิวท์ในการเขียนโปรแกรมเชิงวัตถุสามารถทำได้ด้วย 2 ขั้นตอนดังนี้

- กำหนดสิทธิการเข้าถึง (Access Modifier) แอททริบิวท์เป็น private
- สร้างเมธอดเพื่อใช้เป็นตัวกลางระหว่างผู้เรียกใช้งานกับแอททริบิวท์และกำหนดสิทธิการเข้าถึงเมธอดเป็น public ซึ่งเมธอดเหล่านี้จะเรียกว่าเมธอด setter และ เมธอด getter

↳ ต้องสร้างขึ้นมาให้เรียกใช้

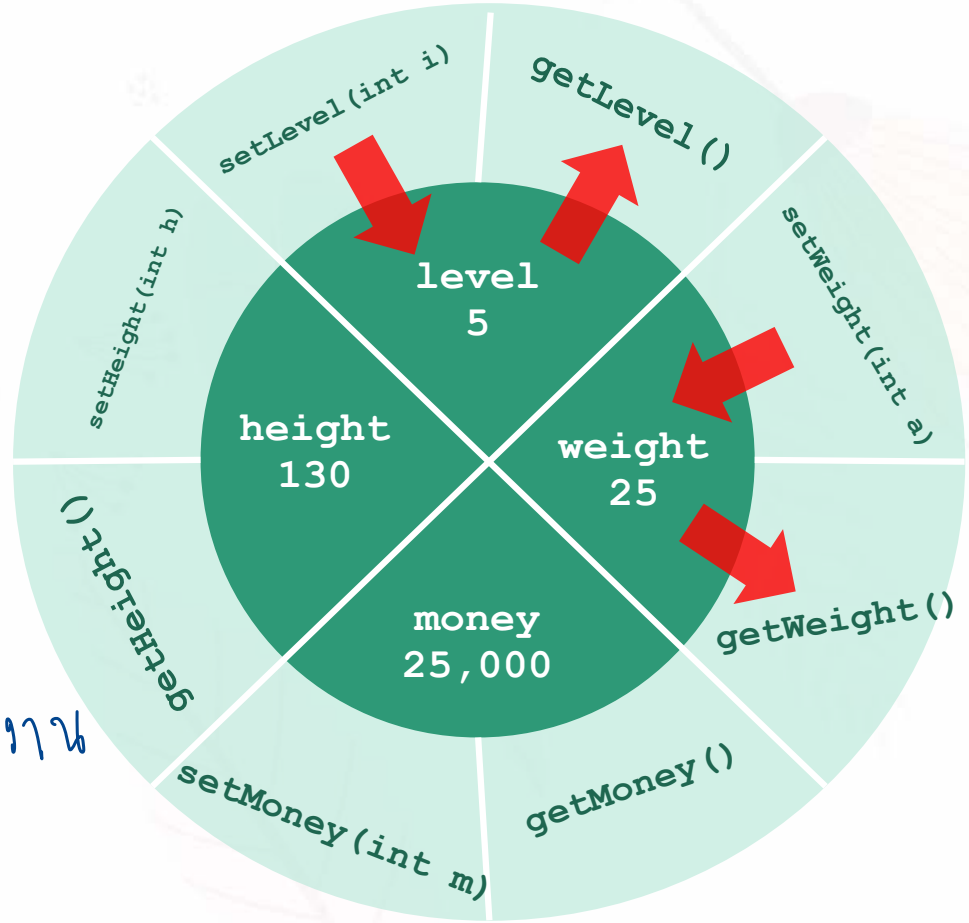
การห่อหุ้ม (Encapsulation)



การห่อหุ้ม (Encapsulation)

① setter
→ เอาค่าภายนอก
มาเก็บใน
Attribute ภายใน

② getter
→ เอาค่า Attribute
ตัวพื้นของส่งไป
ให้ผู้เรียกใช้งาน



เมธอดแบบ setter

กำหนดค่า Attribute

เมธอดแบบ setter ถือว่าเป็นเมธอดแบบ accessor **ที่ใช้ในการกำหนดค่าของคุณลักษณะ** โดยทั่วไปชื่อของเมธอดแบบ setter จะขึ้นต้นด้วยคำว่า set แล้วตามด้วยชื่อของคุณลักษณะ ซึ่งมีรูปแบบดังนี้

```
① ②
public void setAttributeName (dataType arg) {
        attributeName = arg;
}
```

datatype ต้องตรงกับ Attribute

โดยมีหลักการคร่าว ๆ ดังนี้

- Return Type จะกำหนดเป็น void
- Parameter นิยมกำหนดเป็นชนิดข้อมูลเดียวกับแอททริบิวท์
- ไม่มี Return Statement

เมธอดแบบ getter

เมธอดแบบ getter ถือว่าเป็นเมธอดแบบ accessor **ที่ใช้ในการเรียกค่าของคุณลักษณะ** โดยทั่วไปชื่อของเมธอดแบบ getter จะขึ้นต้นด้วยคำว่า get แล้วตามด้วยชื่อของคุณลักษณะ ซึ่งมีรูปแบบดังนี้

ส่วนมากจะเหมือน Attribute

```
public dataType getAttributeName() {
        return attributeName;
 }
```

โดยมีหลักการคร่าว ๆ ดังนี้

- Return Type นิยมกำหนดเป็นชนิดข้อมูลเดียวกับแอททริบิวท์
- ไม่มีการรับ Parameter
- Return Statement จะคืนค่าเป็นแอททริบิวท์ที่สอดคล้องกับชื่อของเมธอด

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

Capital Letter

```
public class Student {
   private double gpa;
    public void setGpa(double g) { // หรือ public void setGPA(...)
        gpa = g;
    }
    public double getGpa() {        // หรือ public double getGPA()
        return gpa;
    }
}
public class Main {
    public static void main(String args[]) {
        Student s1 = new Student();
        double temp = new Scanner(System.in).nextDouble();
# เก็บค่า s1.setGpa(temp);
        System.out.println("GPA: "+s1.getGpa());
    }
}
```

#เรียกใช้

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student {
    private double gpa;
    public void setGpa(double g) {
        if ((g < 0) || (g > 4.00)) { # check ข้อมูลที่รับมาว่าอยู่ใน rangeไหม
            System.out.println("Incorrect Format!"); # ไม่
        } else {
            gpa = g;  # อยู่ = เก็บค่า
        }
    }
    public double getGpa() {
        return gpa;
    }
}
```

ใช้เพื่อเลี่ยงการกดผิดที่หน้าเครื่อง
ex. กดเงิน กดลบ -1000

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student{
        private double gpa;
        private String gender;

        public void setGPA(double g){
                gpa = g;
        }
        public double getGPA(){
                return gpa;
        }
        public void setGender(String s){
                gender = s;
        }
        public String getGender(){
                return gender;
        }
}
```

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student{
        private double gpa;
        private char gender;
        public void setGPA(double g){ gpa = g; }
        public double getGPA(){ return gpa; }
        public void setGender(String s){
            if (s.equals("female"))
                gender = 'f';
            else if (s.equals("male"))
                gender = 'm';
        }
        public String getGender(){
            if (gender == 'f')
                return "female";
            else if (gender == 'm')
                return "male";
        }
}
```

#ก.เก็บอย่างใช้อย่าง

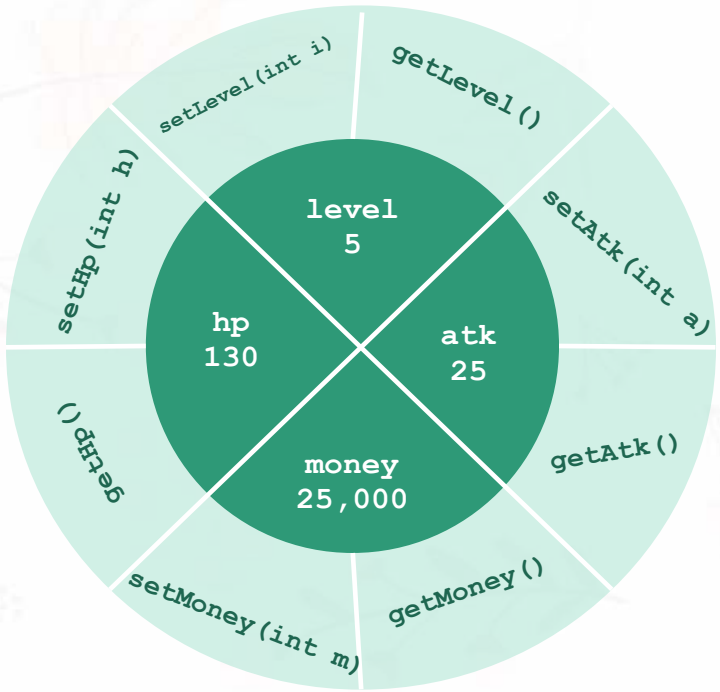
ตัวอย่างโปรแกรมที่ไม่ใช้หลักการของการ

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

public class Main {
   public static void main(String args[]) {
       Student s1 = new Student();
       double temp = Double.parseDouble(args[0]);
       if ((temp<0) || (temp>4.00)) {
              System.out.println("Incorrect Format!");
       } else {
              s1.gpa = temp;
              System.out.println("GPA: "+s1.gpa);
       }
   }
}
```

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student {
        private String name;
        private double gpa;
        public void setName(String n) {
                name = n;
        } public void setGPA(double g) {
            if ((g < 0) || (g > 4.00)) {
                System.out.println("Incorrect Format!");
            } else {
                gpa = g;
            }
        } public String getName() {
                return name;
        } public double getGPA() {
                return gpa;
        }
}
```

การห่อหุ้ม (Encapsulation)

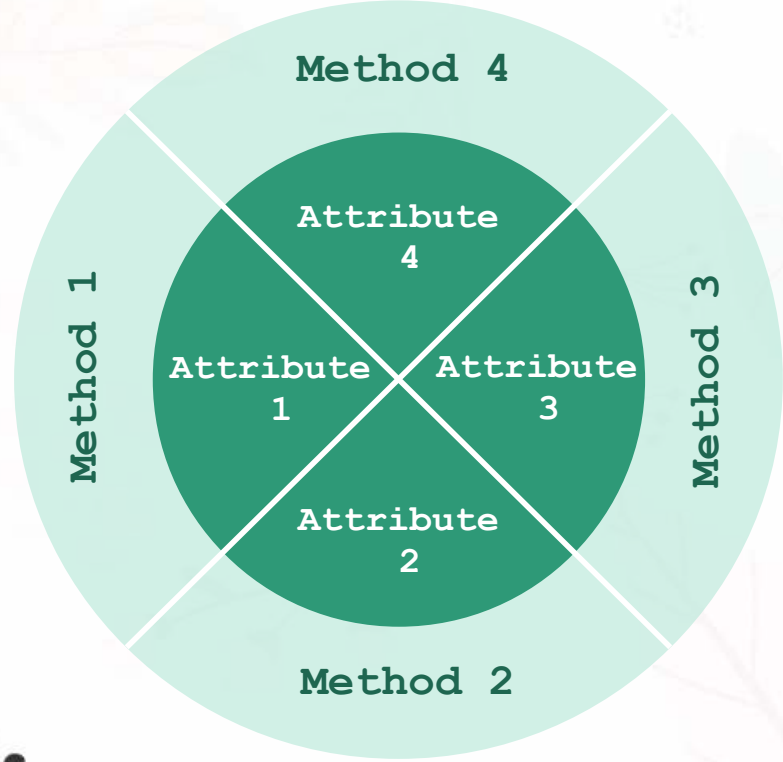


Player

Player	
- hp	: int
- atk	: int
- level	: int
- money	: int
+ setHp (int h)	: void
+ getHp ()	: int
+ setLevel (int l)	: void
+ getLevel ()	: int
+ setMoney (int m)	: void
+ getMoney ()	: int
+ setAtk (int a)	: void
+ getAtk ()	: int

```
public class Player{
    private int hp;
    private int atk;
    private int level;
    private int money;

    public void setHp (int h {
        if (h >= 0) {
          hp = h;
        } else {
          hp = 0;
        }
    } public int getHp () {
        return hp;
    }
          .
          .
          .
}
```

การห่อหุ้ม (Encapsulation)



การใช้งานหลักการห่อหุ้มแอททริบิวท์ในการเขียนโปรแกรมเชิงวัตถุสามารถทำได้ด้วย 2 ขั้นตอนดังนี้

- กำหนดสิทธิการเข้าถึง (Access Modifier) แอททริบิวท์เป็น private
- สร้างเมธอดเพื่อใช้เป็นตัวกลางระหว่างผู้เรียกใช้งานกับแอททริบิวท์และกำหนดสิทธิการเข้าถึงเมธอดเป็น public ซึ่งเมธอดเหล่านี้จะเรียกว่าเมธอด setter และ เมธอด getter

ตัวอย่างการห่อหุ้มที่ 1

```
public class Account {
    private double balance;
    public double getBalance(){   # getter
       return balance;
    }
    public void deposit (double amount){   # setter
        balance += amount;
    }
    public void withdraw (double amount){   # setter
       if (balance - amount >= 0)
          balance -= amount;
       else
          System.out.print("Not enough");
    }
}
```

คีย์เวิร์ด this

มีความหมายถึง object ไม่ใช่ class

คียเวิร์ด this หมายถึง **อ็อบเจกต์ของตัวเอง** ซึ่งนักศึกษาสามารถเรียกใช้เมธอดหรือคุณลักษณะภายในคลาสได้ โดยใช้คีย์เวิร์ด this ที่มีรูปแบบดังนี้

```
this.methodName();
this.attributeName;
```

อ้างอิงถึง object ของ Attribute / method ที่สัมพันธ์

โดยทั่วไป ไม่นิยมใช้คีย์เวิร์ด this ในคำสั่ง ยกเว้นในกรณีที่จำเป็น

```
public class Main{
    private String name;
    public void setName(String name) {
        this.name = name;            // name = name;
    }public void showDetails() {
        System.out.println("Name: "+ this.name);
    }
}
```

หมายความว่าอะไร ?

ตัวอย่างการห่อหุ้มที่ 2

```
public class Gamer {
    private String team;
    public void setTeam(String t) { this.team = t;      }
    public String getTeam() {       return this.team;   }

    public boolean isSameTeam(Player p) {
      if(this.getTeam().equals(p.getTeam())){ return true; }
      else { return false; }
    }

    public static void main(String[] args) {
        Gamer g = new Gamer();
        g.setTeam("A");
        System.out.println(g.getTeam());
        // กรณีคลาสเดียวกัน และเรียกใช้ผ่าน object ตัว g
        System.out.println(g.team);
    }
}
```

```
return (this.getTeam().equals(p.getTeam()));
```

สามารถลดรูปได้

ผลลัพธ์:

A

A

ตัวอย่างการห่อหุ้มที่ 3

```
public class Gamer {
    private String team;
    public void setTeam(String t) { this.team = t;      }
    public String getTeam() {       return this.team;   }
    public boolean isSameTeam(Player p) {
       return (this.getTeam().equals(p.getTeam()));
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Gamer g = new Gamer();
        g.setTeam("A");
        System.out.println(g.getTeam());

        // กรณีไม่ใช้คลาสเดียวกัน และเรียกใช้ผ่าน object ตัว g
        System.out.println(g.team);
    }
}
```

ผลลัพธ์:

A

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - team has private access in Gamer at Main.main(Main.java:10)

การเขียนโปรแกรมเชิงวัตถุ

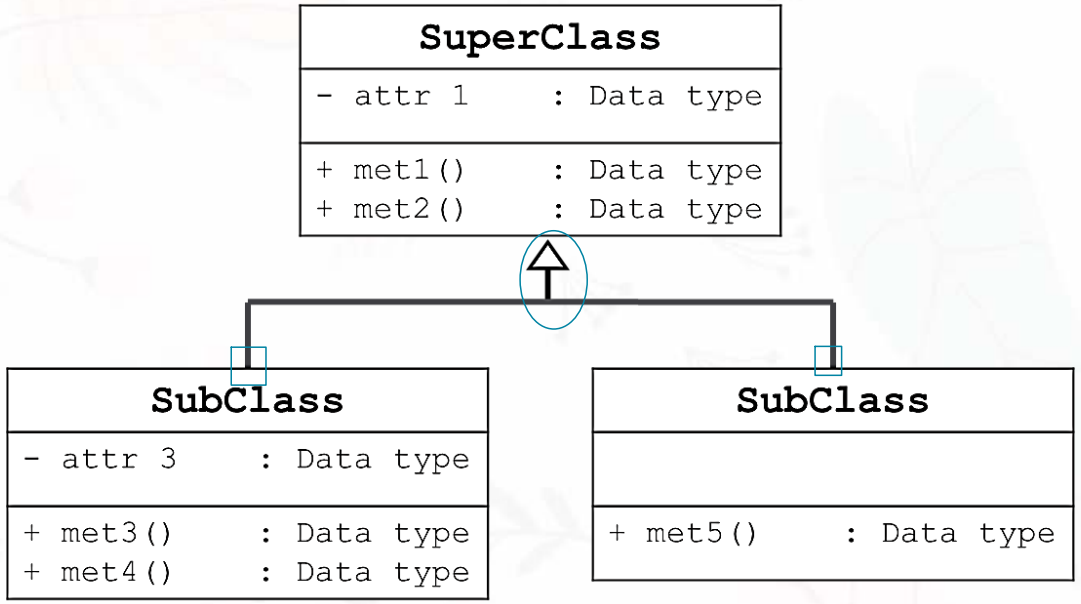
การห่อหุ้ม
Encapsulation

การสืบทอด
Inheritance

การมีได้
หลากหลายรูปแบบ
Polymorphism

การสืบทอด (Inheritance)

อะไรที่แม่มี ลูกก็จะมี
แต่ไม่สามารถใช้ได้



การสืบทอด (Inheritance) ในภาษาจาวาเป็นหนึ่งในแนวคิดหลักของ Object-Oriented Programming การสืบทอดช่วยให้คลาสหนึ่ง (ที่เรียกว่า subclass) สามารถรับคุณสมบัติจากคลาสอื่น (ที่เรียกว่า superclass) ได้ การสืบทอด

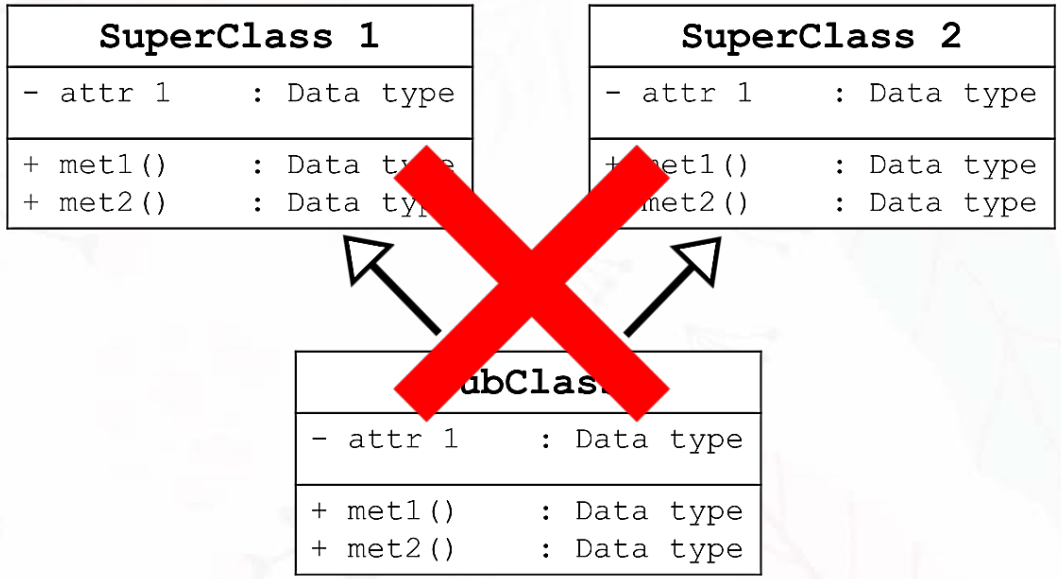
ลูก

แม่

- ช่วยลดความซ้ำซ้อนของโค้ดทำให้โครงสร้างของโปรแกรมมีความเป็นระเบียบมากขึ้น
- ช่วยให้สามารถสร้างคลาสใหม่ที่ขยายความสามารถของคลาสที่มีอยู่แล้ว โดยไม่ต้องแก้ไขโค้ดของคลาสเดิม

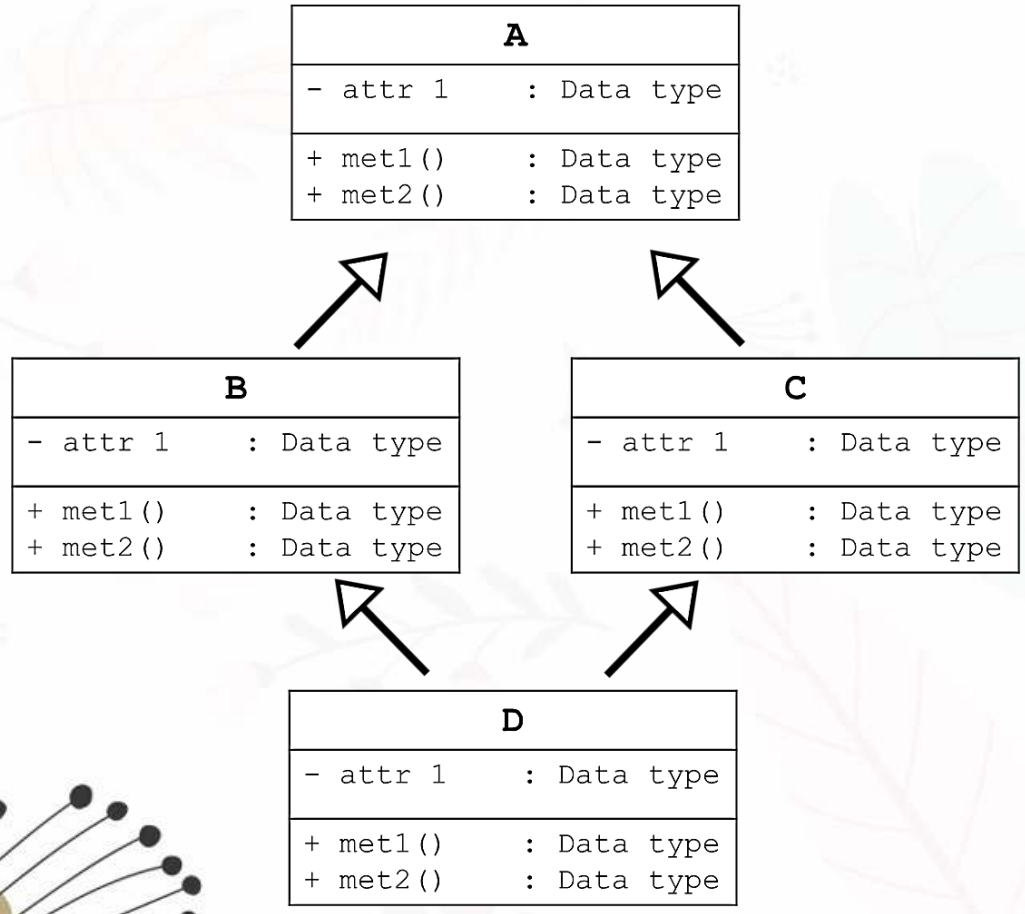
การสืบทอด (Inheritance)

แม่ 1 class มีลูกได้หลาย class
ลูก 1 ตัว มีแม่ได้ 1 class



นอกจากนี้ ในภาษาจาวายังรองรับการสืบทอดแบบหลายระดับ ทำให้ subclass สามารถสืบทอดจาก subclass อื่นๆ ที่สืบทอดมาจาก superclass อย่างไรก็ตาม ในภาษาจาวาไม่รองรับ **"multiple inheritance" (การสืบทอดจากหลายคลาส)** แต่สามารถใช้ interfaces เพื่อจัดการข้อจำกัด “multiple inheritance” ได้

การสืบทอด (Inheritance)

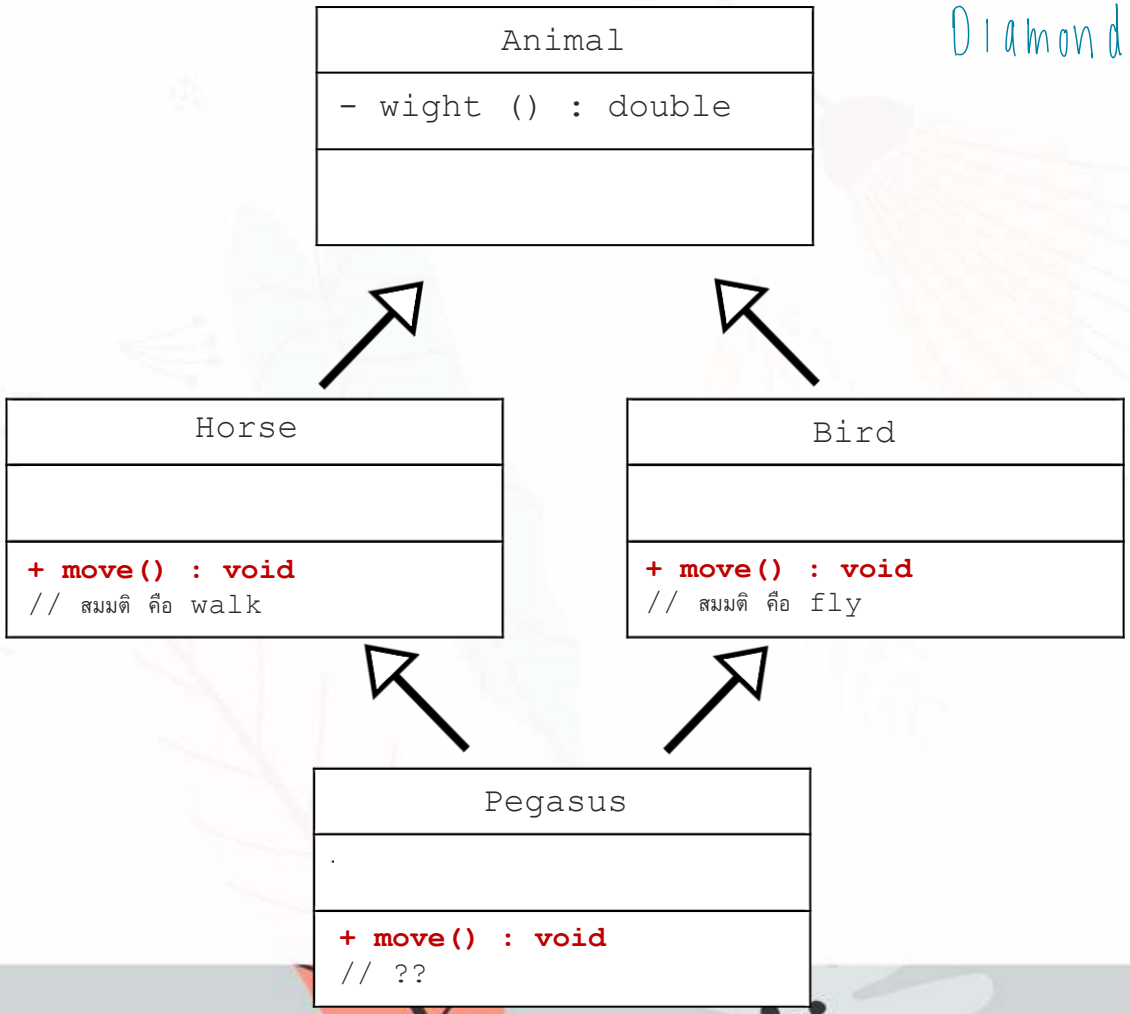


ที่ภาษาจาวาไม่อนุญาตให้มีการสืบทอดมากกว่า 1 คลาส เพื่อหลีกเลี่ยงการเกิดปัญหา “Diamond Problem”

“Diamond Problem” คือ การกำกวมที่เกิดขึ้นเมื่อคลาสใดคลาสหนึ่ง (Class D) มีการสืบทอดมาจากมากกว่าหนึ่งคลาส (Class B และ Class C) และคลาสทั้งสองมีการสืบทอดมาจากคลาสเดียวกัน (Class A) และเกิดการ Overridden ของ met1() ในทุกคลาส (Class A, B และ C) คลาส D จะไม่ทราบว่า met1() จะควรทำงานเหมือนของคลาสใดระหว่าง (Class B และ Class C)

การสืบทอด (Inheritance)

Diamond Problem !



การสืบทอด (Inheritance)

การสืบทอด (Inheritance) สามารถ**ลด**ปริมาณโค้ดที่มี**ความซ้ำซ้อน**กันลง โดยการแชร์โค้ดหรือทรัพยากรในส่วนที่ใช้ร่วมกันกับคลาสลูกทั้งหลาย นอกจากนี้ เทคนิคการสืบทอดยังช่วยให้โปรแกรมของเรา**มีความยืดหยุ่นต่อการเปลี่ยนแปลง** ซึ่งสามารถสรุปได้ดังนี้

ข้อดี

การนำกลับมาใช้ (Reusability)

อำนวยความสะดวกต่อการนำกลับใช้งานอีกครั้ง

ข้อดี

การเพิ่มเติม/ขยายขีดความสามารถ (Extensibility)

สามารถเพิ่มเติมกระบวนการทำงานต่าง ๆ ของคลาสลูกต่อจากคลาสแม่ได้ เพื่อให้เกิดความเหมาะสมมากขึ้น

ข้อดี

การปกปิดข้อมูล (Data hiding)

คลาสแม่สามารถกำหนดได้ว่าจะปกปิดข้อมูลส่วนใดเป็น private หรือไม่อนุญาติให้ปรับเปลี่ยนโค้ดส่วนไหน

ข้อดี

การเขียนใหม่ (Overriding)

เมธอดที่สืบทอดมาจากคลาสแม่สามารถเขียนกระบวนการทำงานใหม่ได้

การสืบทอด (Inheritance)

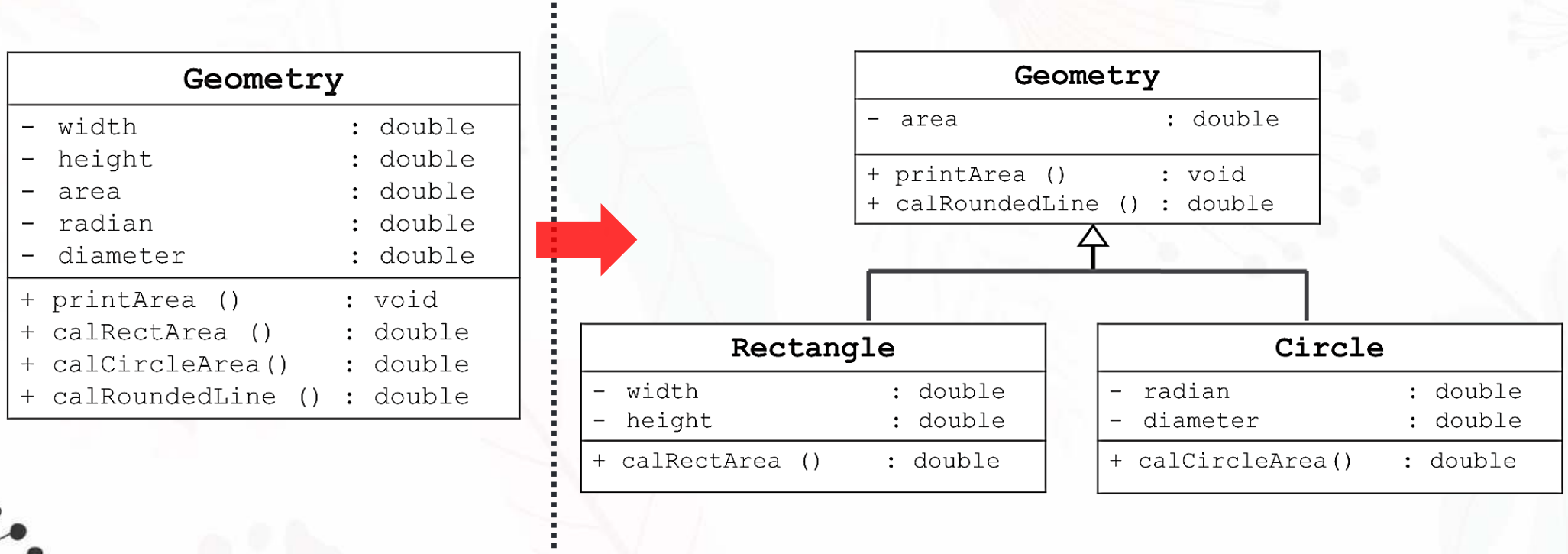
วิธีการออกแบบการสืบทอดทำได้ 2 แบบหลัก ได้แก่

(1) Top-Down

(2) Bottom-Up

การสืบทอด (Inheritance)

วิธีการออกแบบการสืบทอด ได้แก่ (1) Top-Down และ (2) Bottom-Up

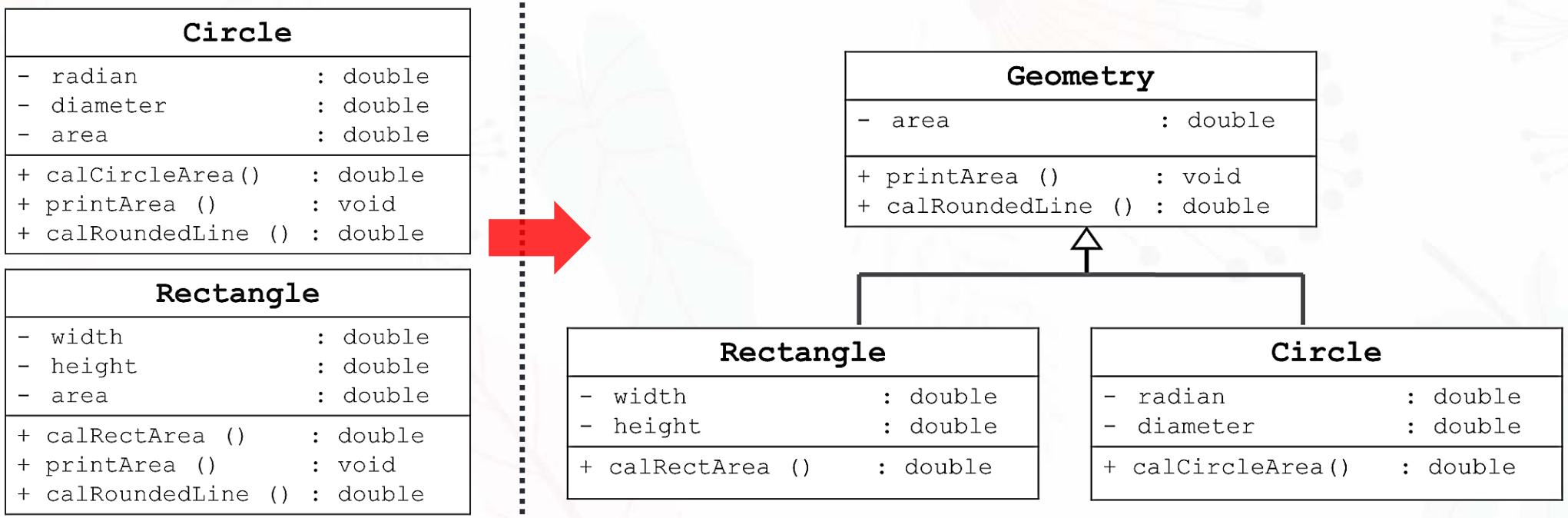


การสืบทอด (Inheritance)

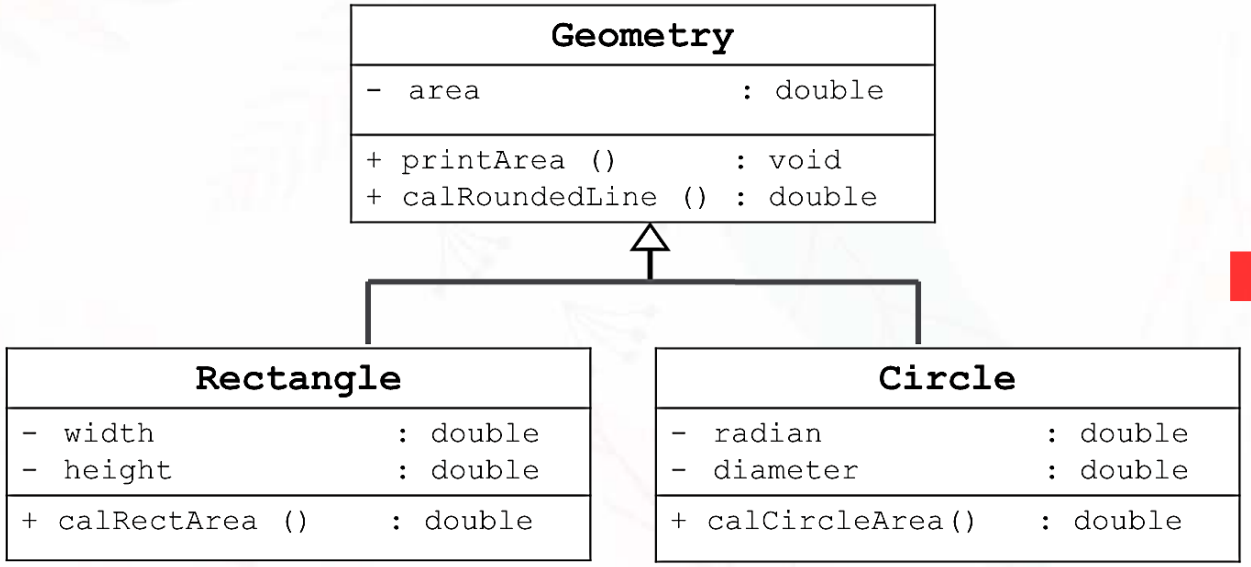
วิธีการออกแบบการสืบทอด ได้แก่ (1) Top-Down และ (2) Bottom-Up

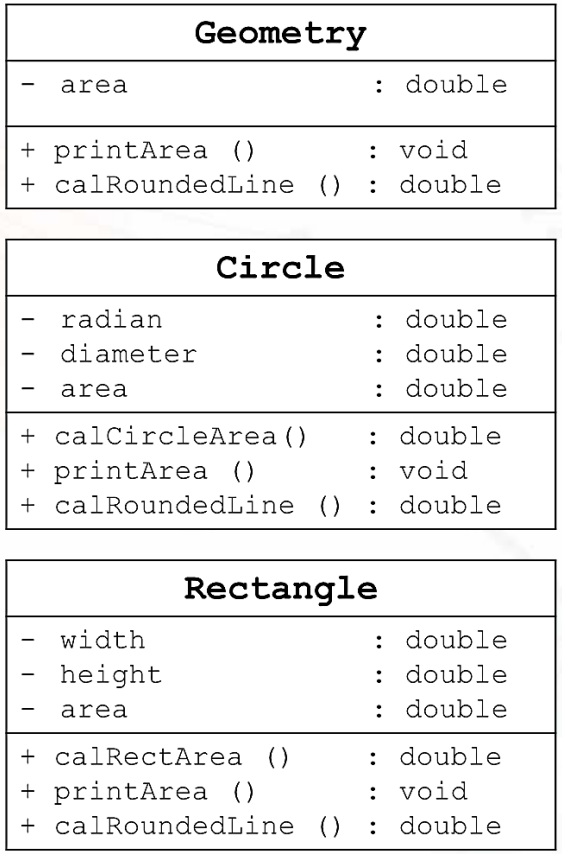
Attribute

Method

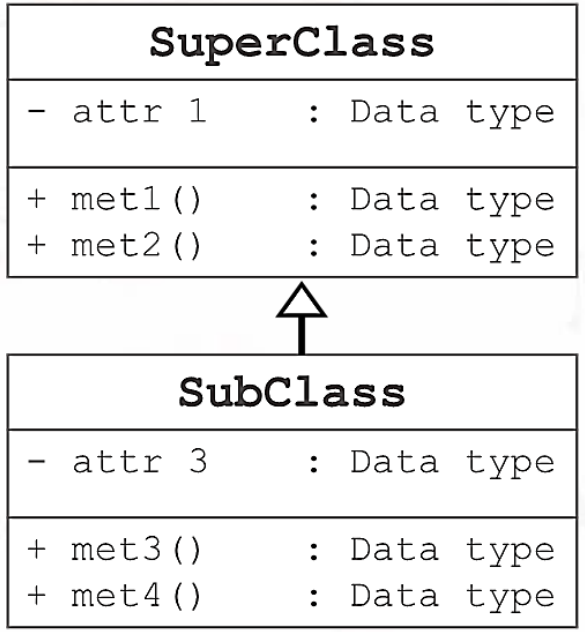


การสืบทอด (Inheritance)





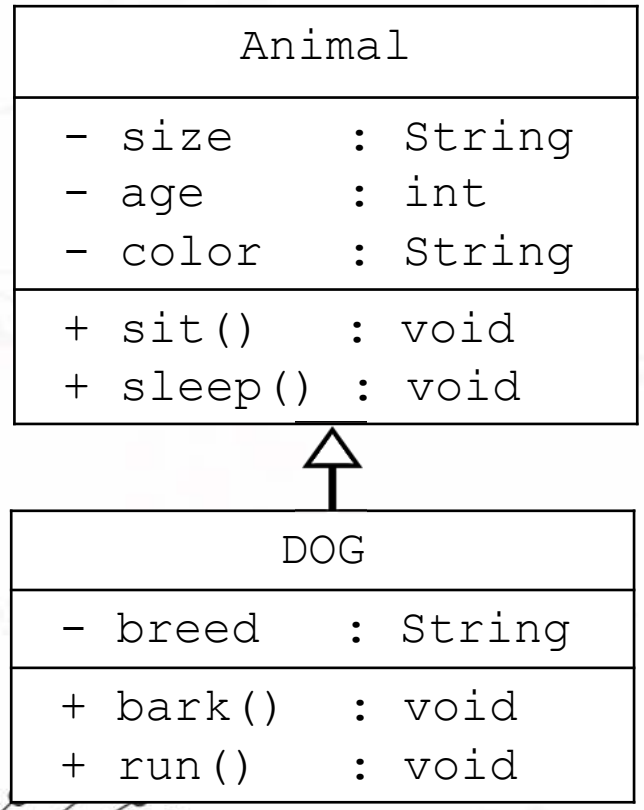
การสืบทอด (Inheritance)




```
public class SuperClass{
    private datatype attr1;

    public void met1(){…}
    public int met2(){…}

}

public class SubClass extends SuperClass  {
    private datatype attr3;

    public double met3(){…}
    public String met4(){…}

}
```

extends ใส่ที่ลูก , ใส่ชื่อ class แม่

การสืบทอด (Inheritance)




```
public class Animal{
    private String size;
    private int age;
    private String color;

    public void sit(){…}
    public void sleep(){…}
}

public class Dog extends Animal{
    private String breed;
    public void bark(){…}
    public void run(){…}
}
```

*แม้ว่าคลาส Dog จะสืบทอดมาจากคลาส Animal แต่นักศึกษาควรแยกคนละไฟล์กัน ”1 ไฟล์ ต่อ 1 คลาส”

ไม่ใช้ inheritance

ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent

การนำคลาสที่มีอยู่แล้ว (Student) มา reuse โดยการเพิ่มเติมคุณลักษณะหรือเมธอดใหม่ลงในคลาสใหม่ ได้แก่ GradStudent นักศึกษาสามารถที่จะเลือกวิธีการได้ 2 แบบ คือ

- สร้างคลาสขึ้นมาใหม่โดยไม่อ้างอิงกับคลาสเดิมที่ชื่อ Student

```
Student
-------------------------------------
- id                    : String
- name                  : String
- gpa                   : double
-------------------------------------
+ setID(String i)       : void
+ setName(String n)     : void
+ setGPA(double g)      : void
+ showDetails()         : void
```

```
GradStudent
-------------------------------------
- id                    : String
- name                  : String
- gpa                   : double
- thesisTitle           : String
- supervisor            : String
-------------------------------------
+ setID(String i)       : void
+ setName(String n)     : void
+ setGPA(double g)      : void
+ showDetails()         : void
+ setThesisTitle(String t) : void
+ setSupervisor (String s) : void
+ showThesis()          : void
```

ไม่ใช้ inheritance = ซ้ำซ้อน

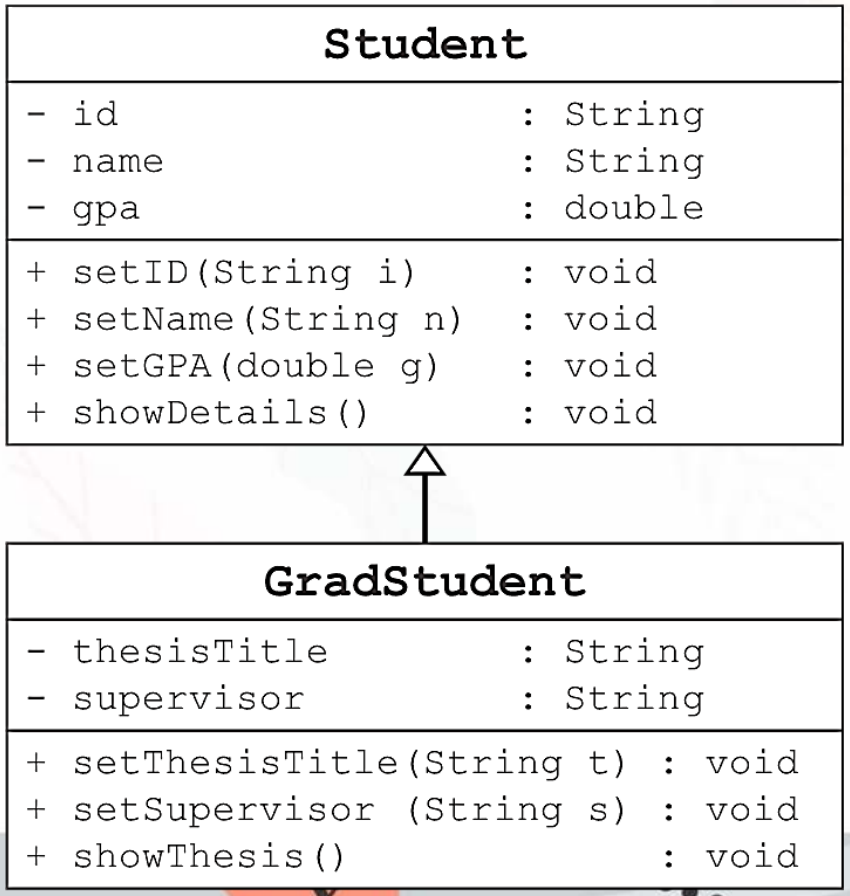
ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent

```
public class Student {
    private String id;
    private String name;
    private double gpa;

    public void setID(String i) {
        id = i;
    }
    public void setName(String n) {
        name = n;
    }
    public void setGpa(double g) {
        gpa = g;
    }
    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}
```

```
public class GradStudent {
    private String id, name;
    private double gpa;
    private String thesisTitle;
    private String supervisor;
    public void setID(String i) {
        id = i;
    }public void setName(String n) {
        name = n;
    }public void setGpa(double g) {
        gpa = g;
    }public void setThesisTitle(String t) {
        thesisTitle = t;
    }public void setSupervisor(String s) {
        supervisor = s;
    }public void showThesis() {
        System.out.println("Title: "+thesisTitle);
        System.out.println("Supervisor: "+supervisor);
    }
}
```

ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent

- สร้างคลาสที่สืบทอดมาจากคลาสเดิมที่ชื่อ Student



ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent

```java
public class Student {
    private String id;
    private String name;
    private double gpa;

    public void setID(String i) {
        id = i;
    }
    public void setName(String n) {
        name = n;
    }
    public void setGpa(double g) {
        gpa = g;
    }
    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}
```

```java
public class GradStudent extends Student {
    private String thesisTitle;
    private String supervisor;
    public void setThesisTitle(String t) {
        thesisTitle = t;
    }
    public void setSupervisor(String s) {
        supervisor = s;
    }

    public void showThesis() {
        System.out.println("Title: "+thesisTitle);
        System.out.println("Supervisor: "+supervisor);
    }
}
```

การพิจารณาการสืบทอดโดยอ้างอิงจากความสัมพันธ์

สำหรับการเขียนโปรแกรมเชิงวัตถุจะสามารถแบ่งแยกความสัมพันธ์ออกได้เป็น 2 แบบ ได้แก่

- **Is-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**เป็น**อีกสิ่งหนึ่งด้วย เช่น
  - แมวเป็นสัตว์
  - ต้นส้มเป็นต้นไม้
  - รถยนต์เป็นพาหนะ

- **Has-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**มี**อีกสิ่งหนึ่งด้วย เช่น
  - คนมีความสามารถว่ายน้ำ
  - ปลามีความสามารถว่ายน้ำ

การพิจารณาการสืบทอดโดยอ้างอิงจากความสัมพันธ์

สำหรับการเขียนโปรแกรมเชิงวัตถุจะสามารถแบ่งแยกความสัมพันธ์ออกได้เป็น 2 แบบ ได้แก่

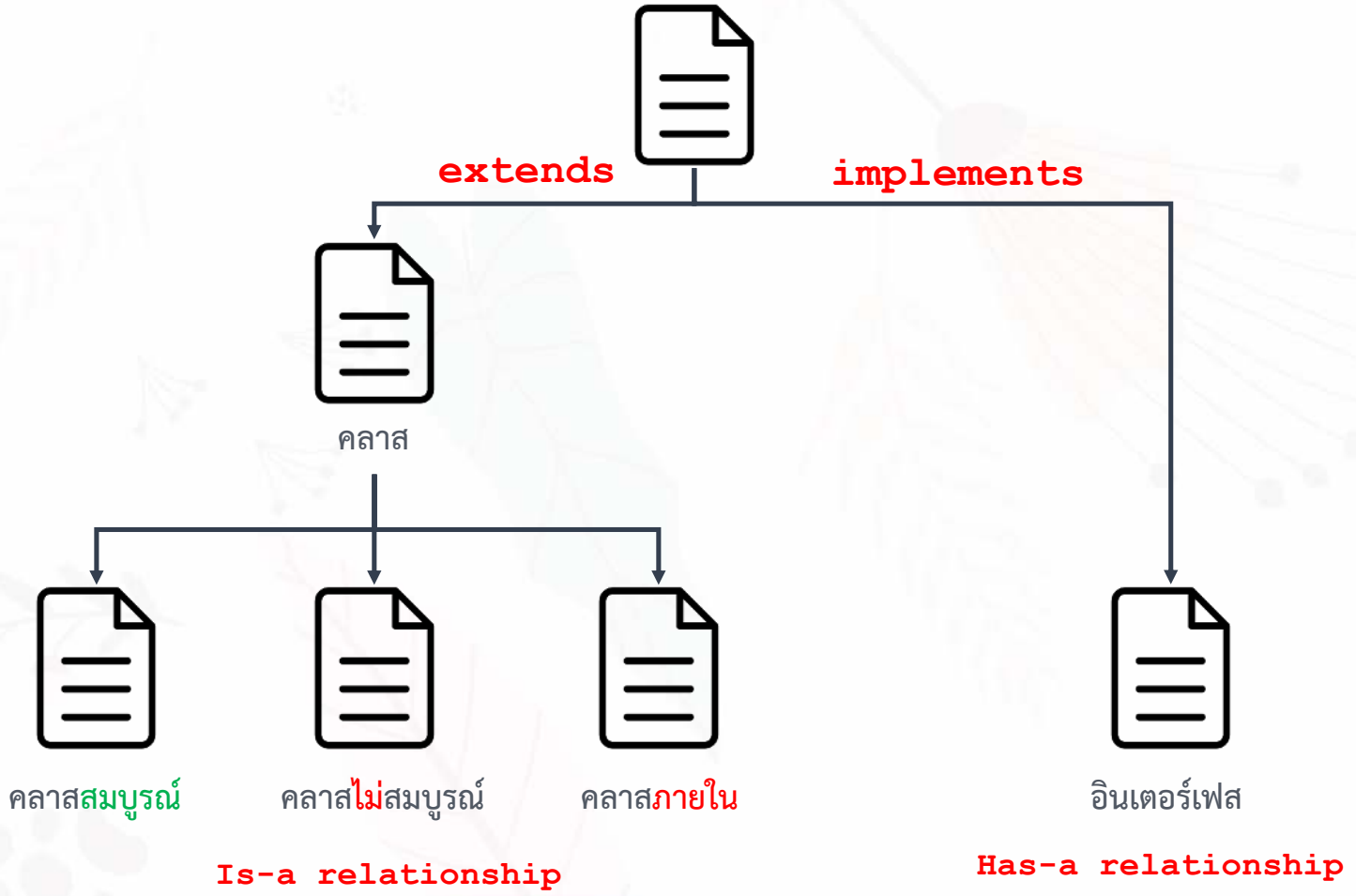
- **Is-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**เป็น**อีกสิ่งหนึ่งด้วย เช่น
  - แมวเป็นสัตว์
  - ต้นส้มเป็นต้นไม้
  - รถยนต์เป็นพาหนะ

การสืบทอดแบบ extends

- **Has-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**มี**อีกสิ่งหนึ่งด้วย เช่น
  - คนมีความสามารถว่ายน้ำ
  - ปลามีความสามารถว่ายน้ำ

การสืบทอดแบบ implements

การพิจารณาการสืบทอดโดยอ้างอิงจากความสัมพันธ์



ตัวอย่างการสืบทอดที่ไม่ถูกต้อง

```
public class Shirt {
      char size;
      float price;
}

public class Skirt extends Shirt {
      double height;
}
```

ตัวอย่างการสืบทอดที่ถูกต้อง

```
public class Clothing {
      char size;
      float price;
}
public class Shirt extends Clothing {
}
public class Skirt extends Clothing {
      double height;
}
```

ทำไมถึงเกิดข้อผิดพลาด

```java
public class Student {
    private String id;
    private String name;
    private double gpa;

    public void setID(String i) {
        id = i;
    }

    public void setName(String n) {
        name = n;
    }

    public void setGPA(double g) {
        gpa = g;
    }

    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}
```

ลูกมี หน้าตาเหมือนแม่ แต่ไม่ได้ใช้สมบัติได้

```java
public class GradStudent extends Student {
    private String thesisTitle;
    private String supervisor;

    public void setThesisTitle(String t) {
        thesisTitle = t;
    }
    public void setSupervisor(String s) {
        supervisor = s;
    }
    public void showThesis() {
        System.out.println("Title: "+thesisTitle);
        System.out.println("Supervisor: "+supervisor);
    }
    public void printMyName() {
        System.out.println("My name is: "+ name);
    }  แววว

    public static void main(String[] args) {
        new GradStudent().printMyName();
    }
}
```

ผลลัพธ์

```
GradStudent.java:16: error: name has private access in Student
System.out.println("My name is: "+ name);
```

วิธีการแก้ไขแบบที่ 1

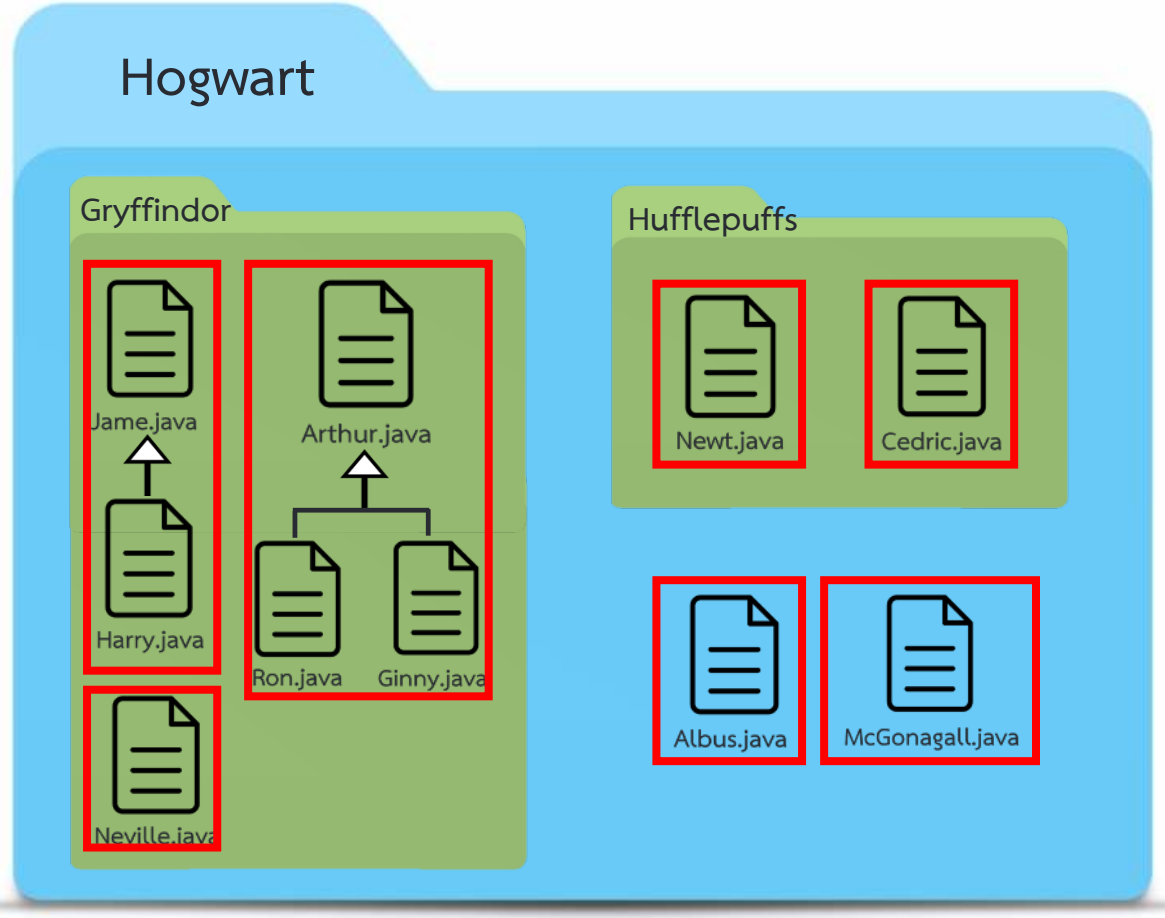
```java
public class Student {
   private String id;
   private String name;
   private double gpa;

   public void setID(String i) {
         id = i;
   }
   public void setName(String n) {
         name = n;
   }
   public void setGPA(double g) {
         gpa = g;
   }
   public void showDetails() {
         System.out.println("ID: "+id);
         System.out.println("Name: "+name);
         System.out.println("GPA: "+gpa);
   }
   public String getName() {
          return name;
   }
}
```

```java
public class GradStudent extends Student {
   private String thesisTitle;
   private String supervisor;

   public void setThesisTitle(String t) {
         thesisTitle = t;
   }
   public void setSupervisor(String s) {
         supervisor = s;
   }
   public void showThesis() {
         System.out.println("Title: "+thesisTitle);
         System.out.println("Supervisor: "+supervisor);
   }
   public void printMyName() {
         System.out.println("My name is: "+ getName());
   }
   public static void main(String[] args) {
           new GradStudent().printMyName();
   }
}
```

ผลลัพธ์

My name is: null

การกำหนดสิทธิการเข้าถึงแบบ protected



① อนุญาตตัวมัน, subclass

② อยู่ใน package เดียวกัน

การกำหนดสิทธิการเข้าถึง (Access Modifier)

1. public เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น public จะหมายความว่า สามารถเข้าถึงได้จากทุกที่ในโปรเจค ไม่ว่าจะอยู่ในแพ็กเกจเดียวกันหรือแพ็กเกจต่างกันที่อยู่ในโปรเจคเดียวกัน public ถือว่าเป็นระดับการเข้าถึงที่กว้างที่สุดและสามารถใช้ในกรณีที่คุณต้องการให้สามารถเรียกใช้ได้จากที่ใดก็ได้

2. protected เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น protected จะหมายความว่า สามารถเข้าถึงได้จากคลาสที่อยู่ในแพ็กเกจเดียวกันหรือสามารถคลาสที่ได้รับการสืบทอด (inheritance) เท่านั้น protected ถือว่าเป็นระดับการเข้าถึงที่มีความจำกัดมากกว่า public แต่ยังคงเปิดให้คลาสที่สืบทอดสามารถเข้าถึงได้

3. default (ไม่มี access modifier) เมื่อคลาส เมธอด หรือแอททริบิวท์ไม่มีการระบุ access modifier จะถือว่ามี access modifier เป็น default หรือหรือเรียกว่า “package private” ซึ่งหมายความว่า สามารถเข้าถึงได้เฉพาะในแพ็กเกจเดียวกันเท่านั้น และไม่สามารถเรียกใช้จากแพ็กเกจอื่นได้

4. private เมื่อคลาส เมธอด หรือแอททริบิวท์ถูกกำหนดเป็น private จะหมายความว่า สามารถเข้าถึงได้เฉพาะจากภายในคลาสเท่านั้น ไม่สามารถเข้าถึงจากภายนอกคลาสได้

การกำหนดสิทธิการเข้าถึง (Access Modifier)

modifier	คลาส เดียวกัน	คลาสที่อยู่ในแพจเก็จเดียวกัน	คลาส ที่เป็นsubclass	คลาสใด ๆ
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

วิธีการแก้ไขแบบที่ 2

```java
public class Student {

    protected String id;
    protected String name;
    protected double gpa;

    public void setID(String i) {
          id = i;
    }

    public void setName(String n) {
          name = n;
    }

    public void setGPA(double g) {
          gpa = g;
    }

    public void showDetails() {
          System.out.println("ID: "+id);
          System.out.println("Name: "+name);
          System.out.println("GPA: "+gpa);
    }
}
```

```java
public class GradStudent extends Student {

    private String thesisTitle;
    private String supervisor;

    public void setThesisTitle(String t) {
          thesisTitle = t;
    }
    public void setSupervisor(String s) {
          supervisor = s;
    }
    public void showThesis() {
          System.out.println("Title: "+thesisTitle);
          System.out.println("Supervisor: "+supervisor);
    }
    public void printMyName() {
          System.out.println("My name is: "+ name);
    }
    public static void main(String[] args) {
            new GradStudent().printMyName();
    }
}
```

ผลลัพธ์

My name is: null

คลาสที่ชื่อ Object

#ทุก class ต้องมี superclass (แม่)

ภาษาจาวาได้กำหนดให้คลาสใดๆ สามารถจะสืบทอดคลาสอื่นได้เพียงคลาสเดียวเท่านั้น ดังนั้น คลาสทุกคลาสในภาษาจาวา ถ้าไม่ได้สืบทอดจากคลาสใดเลยจะถือว่าสืบทอดจากคลาสที่ชื่อ Object

ตัวอย่างเช่น

```
public class Student {
    . . .
}

public class Student extends Object {
    . . .
}
```

#ถ้าไม่ระบุ จะ extends Object ให้เอง

คลาสที่ชื่อ Object จะมีเมธอดที่สำคัญคือ

public String toString() และ public boolean equals(Object o)

↳ print ชื่อ class @ Address

คีย์เวิร์ด super

super เป็นคำหลักที่ใช้เพื่ออ้างอิงโดยตรงไปยัง Superclass (หรือคลาสแม่) ของคลาสปัจจุบัน ซึ่งสามารถเรียกใช้งานเมธอดของ Superclass โดยอาศัย super เพื่อเรียกเมธอดที่ถูกโอเวอร์ไรด์ (overridden) ในซูเปอร์คลาส

ตัวอย่าง

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

super.makeSound() ในคลาส Dog ถูกใช้เพื่อเรียกเมธอด makeSound() จากคลาส Animal. นี่เป็นวิธีที่มีประโยชน์ในการเรียกใช้เมธอดที่ถูกโอเวอร์ไรด์จากซูเปอร์คลาส

คีย์เวิร์ด super

```java
public class Student {

    private String id;
    private String name;
    private double gpa;

    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}
```

ผลลัพธ์

ID: null

Name: null

GPA: 0.0

Title: null

Supervisor: null

```java
public class GradStudent extends Student {

    private String thesisTitle;
    private String supervisor;
    @Override
    public void showDetails() {
        System.out.println("Title: "+ thesisTitle);
        System.out.println("Supervisor: "+supervisor);
    }

    public void show() {
        super.showDetails();
        this.showDetails();
    }
    public static void main(String[] args) {
        GradStudent g = new GradStudent();
        g.show ();
    }
}
```

* super. ใช้ในกรณีที่มีค.สับสนเท่านั้น!!

คีย์เวิร์ด super

```java
public class Student {

   private String id;
   private String name;
   private double gpa;

   public void showDetails() {
      System.out.println("ID: "+id);
      System.out.println("Name: "+name);
      System.out.println("GPA: "+gpa);
   }
}
```

```java
public class GradStudent extends Student {

   private String thesisTitle;
   private String supervisor;

   public void showDetails() {
     super.showDetails();  // showDetails()
     System.out.println("Title: "+ thesisTitle);
     System.out.println("Supervisor: "+supervisor);
   }
   public static void main(String[] args) {
       GradStudent g = new GradStudent();
       g.showDetails();
   }
}
```

ผลลัพธ์

ID: null

Name: null

GPA: 0.0

Title: null

Supervisor: null

ตัวอย่างเช่น คลาส GradStudent อาจมีเมธอดที่ชื่อ showDetail All() โดยมีคำสั่งที่เรียกใช้เมธอด showDetails() ของคลาส Stud ent และ showThesis() ของคลาส GradStudent

Thanks !