

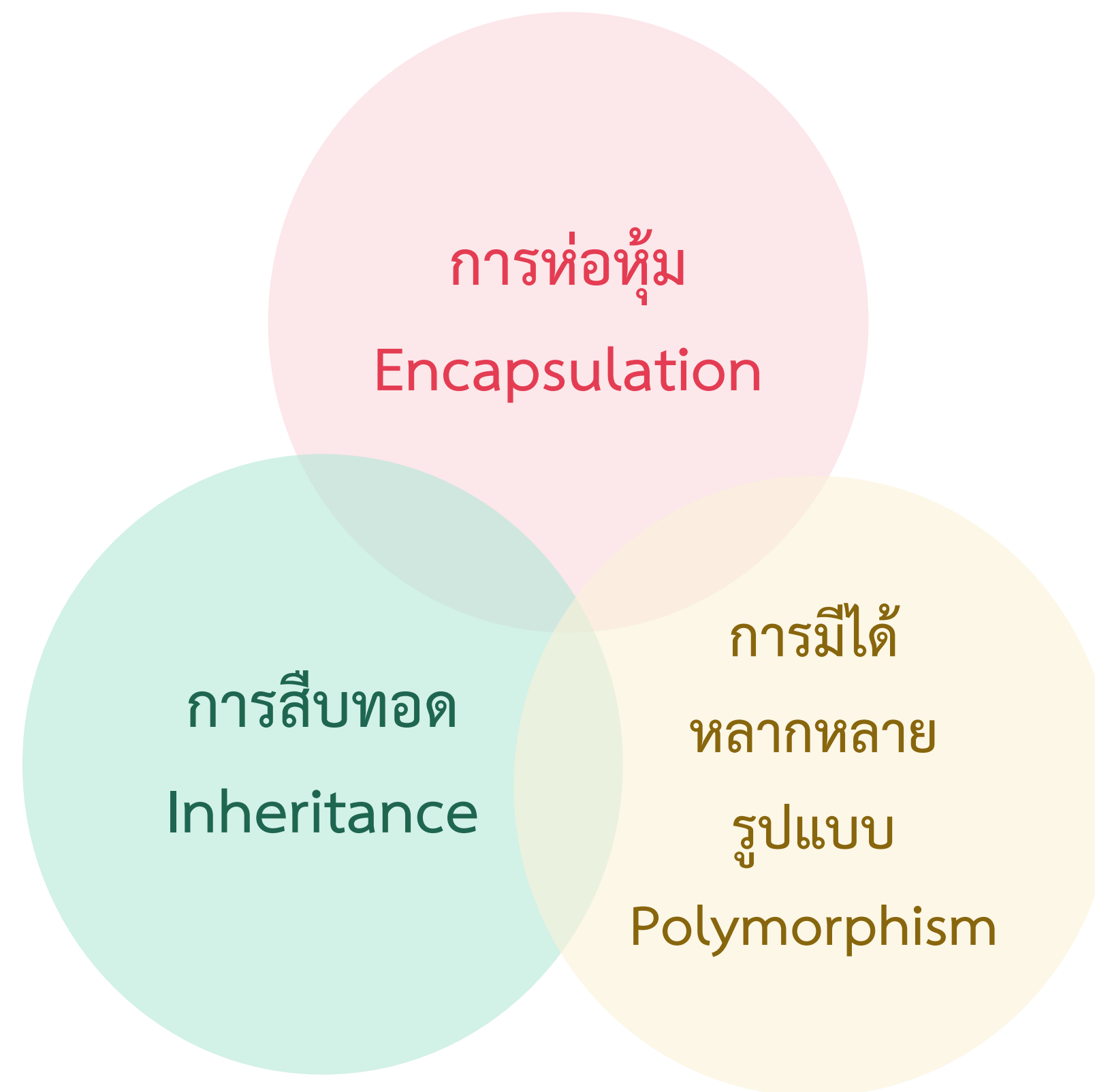
บทที่ 5 การเขียนโปรแกรมเชิงวัตถุ โดยอาศัยหลักการห่อหุ้มและสืบทอด

บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์

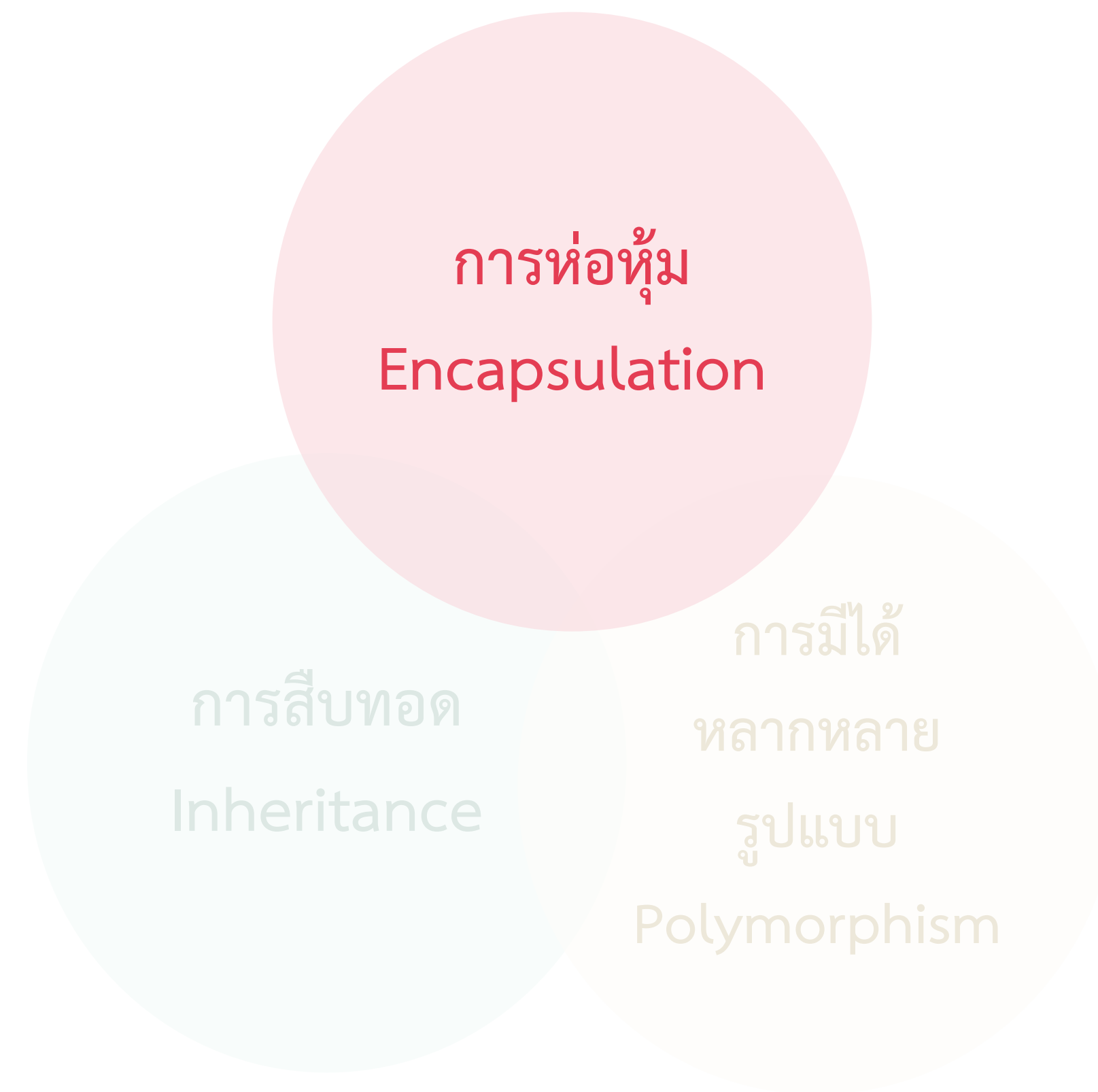
คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

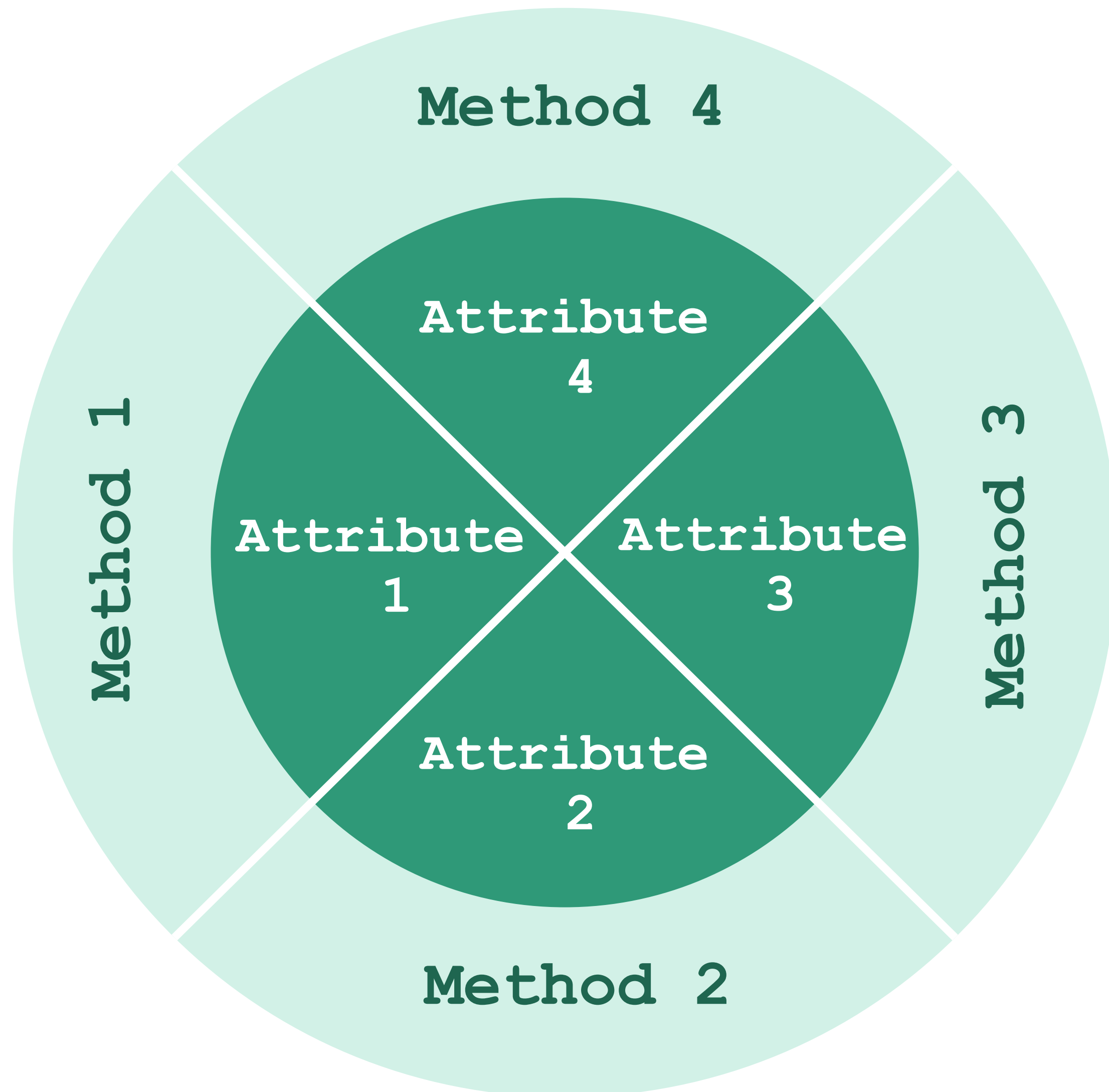
การเขียนโปรแกรมเชิงวัตถุ



การเขียนโปรแกรมเชิงวัตถุ



การห่อหุ้ม (Encapsulation)



การห่อหุ้ม คือ กระบวนการปกปิดโค้ดและข้อมูล
สำหรับการจัดการ หรือสามารถกล่าวได้ว่า
กระบวนการห่อหุ้ม เป็นวิธีการปกป้องข้อมูลจาก
การเข้าถึงจากภายนอกหรือโปรแกรมอื่น ๆ
(Data-Hiding)

การห่อหุ้ม (Encapsulation)

ทำไมเราควรกำหนดการห่อหุ้มแอตทริบิวต์

ข้อดี

ความเป็นส่วนตัวสามารถป้องกันวัตถุอื่น ๆ เข้ามาทำลายคุณสมบัติบางอย่าง

- เลขบัญชีธนาคารติดลบแทนที่จะเป็นเลขศูนย์หรือเลขบวก
- ความกว้างและความสูงของสี่เหลี่ยมน้อยกว่าหรือเท่ากับศูนย์

ข้อดี

ทำให้เราสามารถ
เปลี่ยนแปลงชนิดของตัวแปรเหล่านั้นได้ โดยไม่กระทบกระเทือนคลาสอื่นที่ใช้คลาสของเรา

ข้อดี

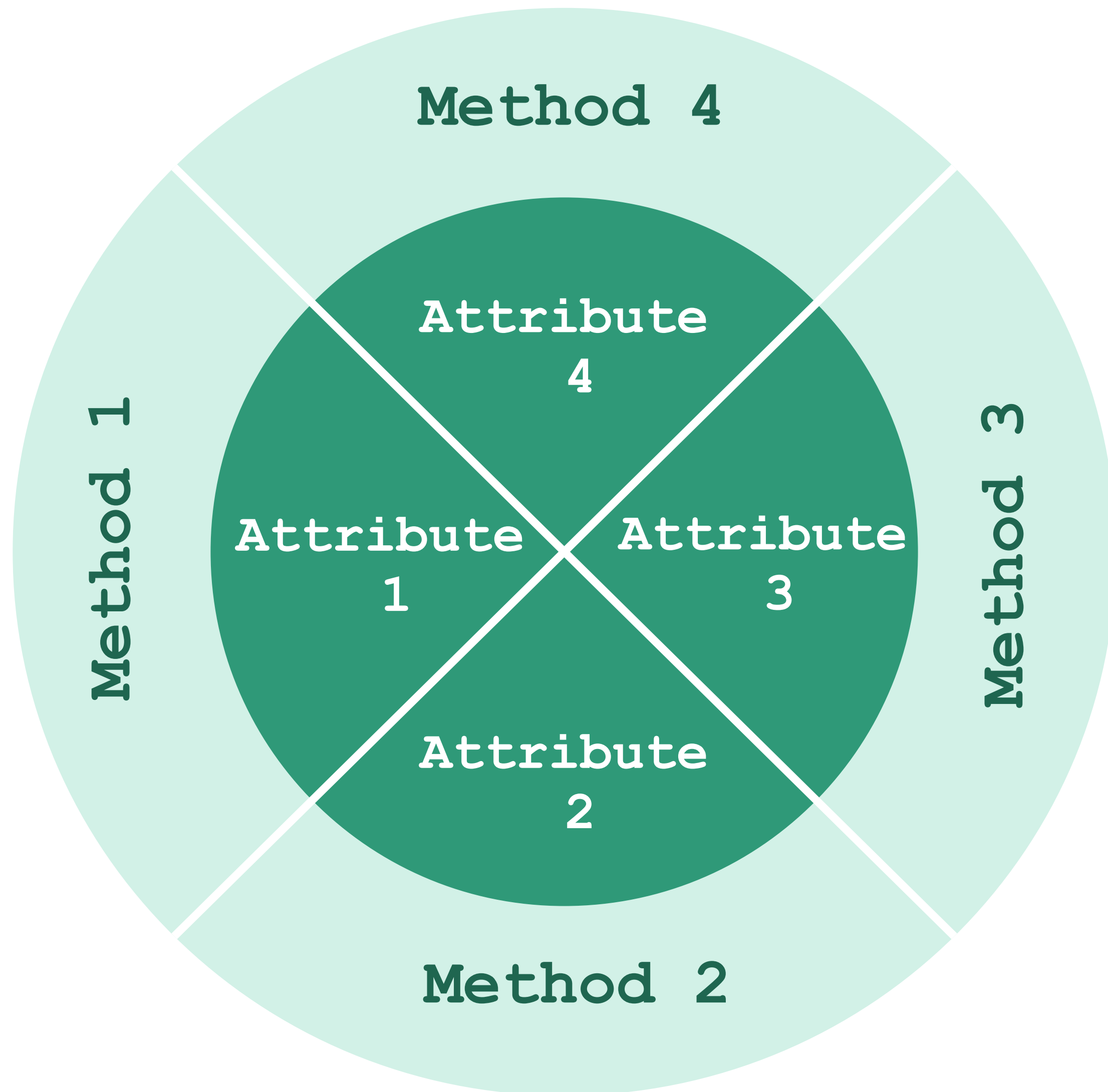
สามารถเปลี่ยนแปลง
วิธีการทำงานของวัตถุได้
โดยไม่กระทบต่อผู้ใช้วัตถุ

ข้อดี

เกิดความยืดหยุ่น

เช่น ถ้าเปรียบวัตถุเหมือนกับรถและผู้ใช้วัตถุคือคนขับรถ บริษัทผู้ผลิตรถสามารถเปลี่ยนเครื่องยนต์รุ่นใหม่ เปลี่ยนระบบเบรก ฯลฯ โดยที่คนขับยังสามารถขับรถได้เหมือนเดิม

การห่อหุ้ม (Encapsulation)



การห่อหุ้มนั้นสามารถทำได้โดย

- การกำหนดให้ทุกแอตทริบิวต์เป็น **private**
- สร้างเมธอดเพื่อใช้ในการเข้าถึงแอตทริบิวต์เป็น **public** ซึ่งเมธอดเหล่านี้จะเรียกว่า **set** และ **get**

การห่อหุ้ม (Encapsulation)

1

Modifier

คือ ตัวกำหนดขอบเขต
และสิทธิการเข้าถึงคลาส
แอสทริคิวิท์ และ เมธอด

public

ทุกคลาสสามารถเข้าถึงและเรียกใช้งานได้

อนุญาตให้คลาสอื่นใช้ได้ ส่วนใหญ่จะให้เมธอดมีการเข้าใช้แบบ public

private

ภายในคลาสเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

วัตถุต่างคลาสไม่สามารถเข้าใช้ได้ และ สามารถปกป้องข้อมูลได้ ส่วนใหญ่
จะให้แอสทริคิวิท์มีการเข้าใช้แบบ private

protected

คลาสแม่ลูกกันเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

เป็นคลาสที่เกี่ยวข้องกับการสืบทอดกันเท่านั้น

default

ทุกคลาสภายใน package เดียวกันที่สามารถเข้าถึงและ
เรียกใช้งานได้

แพ็คเกจเป็นที่เก็บรวบรวมคลาสที่เกี่ยวข้องกัน หรือเปรียบได้กับโฟลเดอร์
ซึ่งการเข้าใช้แบบแพ็คเกจไม่ต้องใช้คำประกอบใดๆ

2

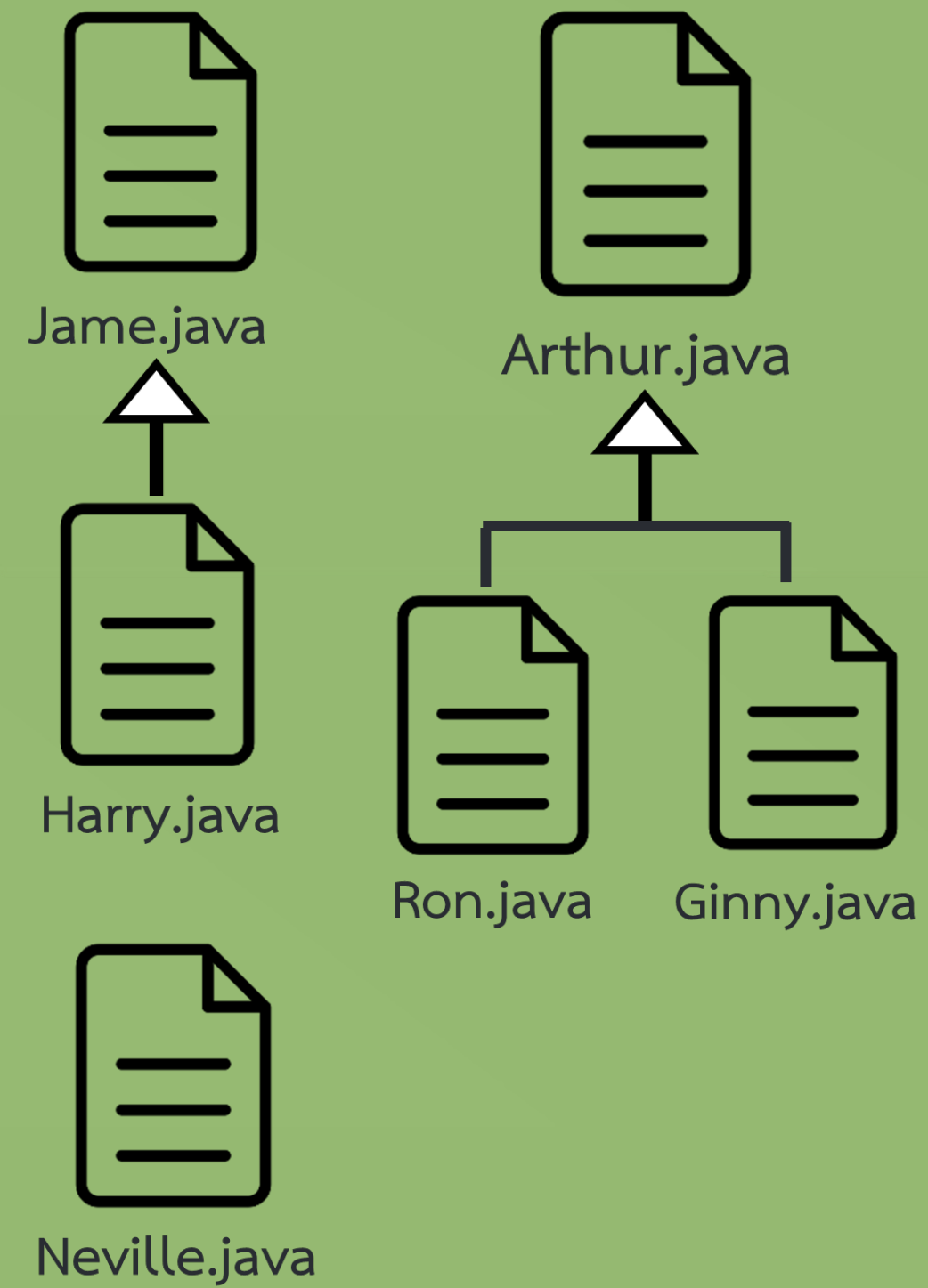
Method (set / get)

Access Modifier

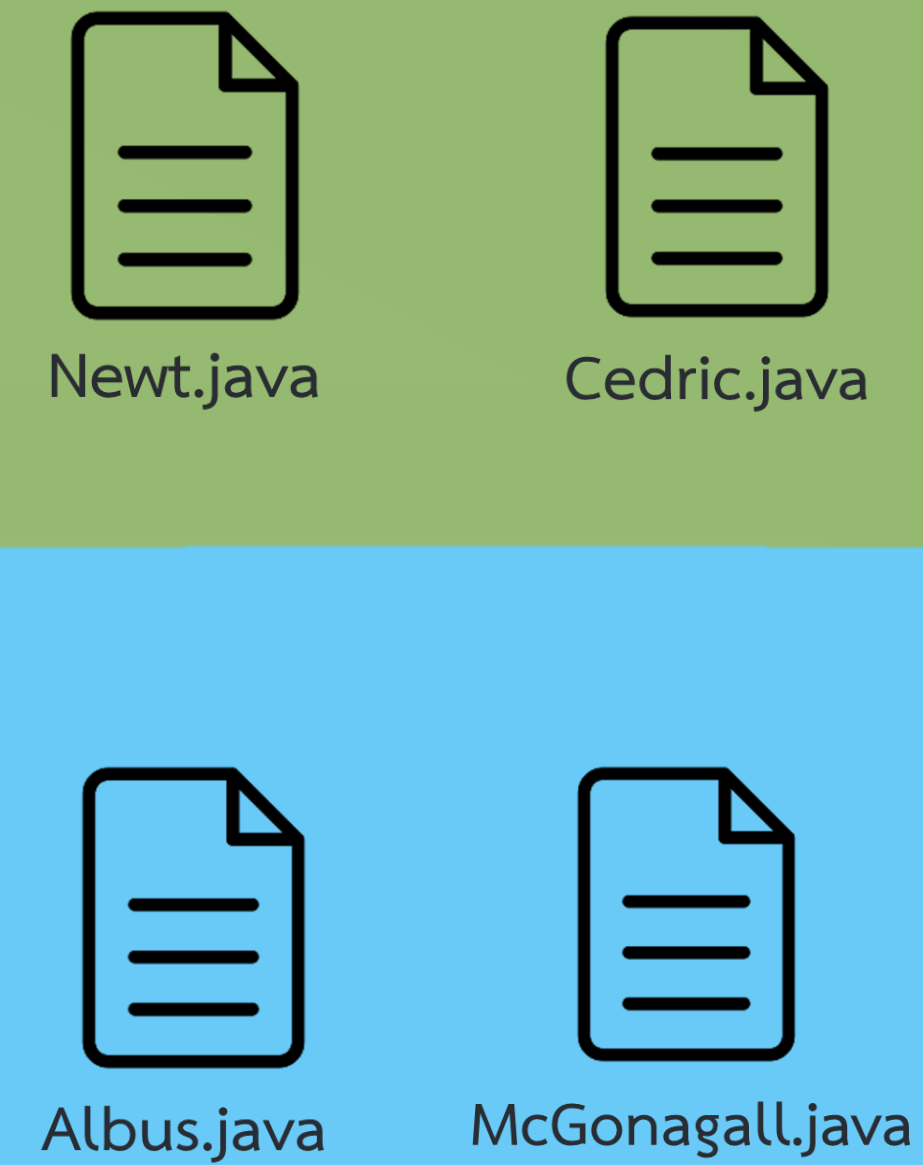


Hogwart

Gryffindor



Hufflepuffs

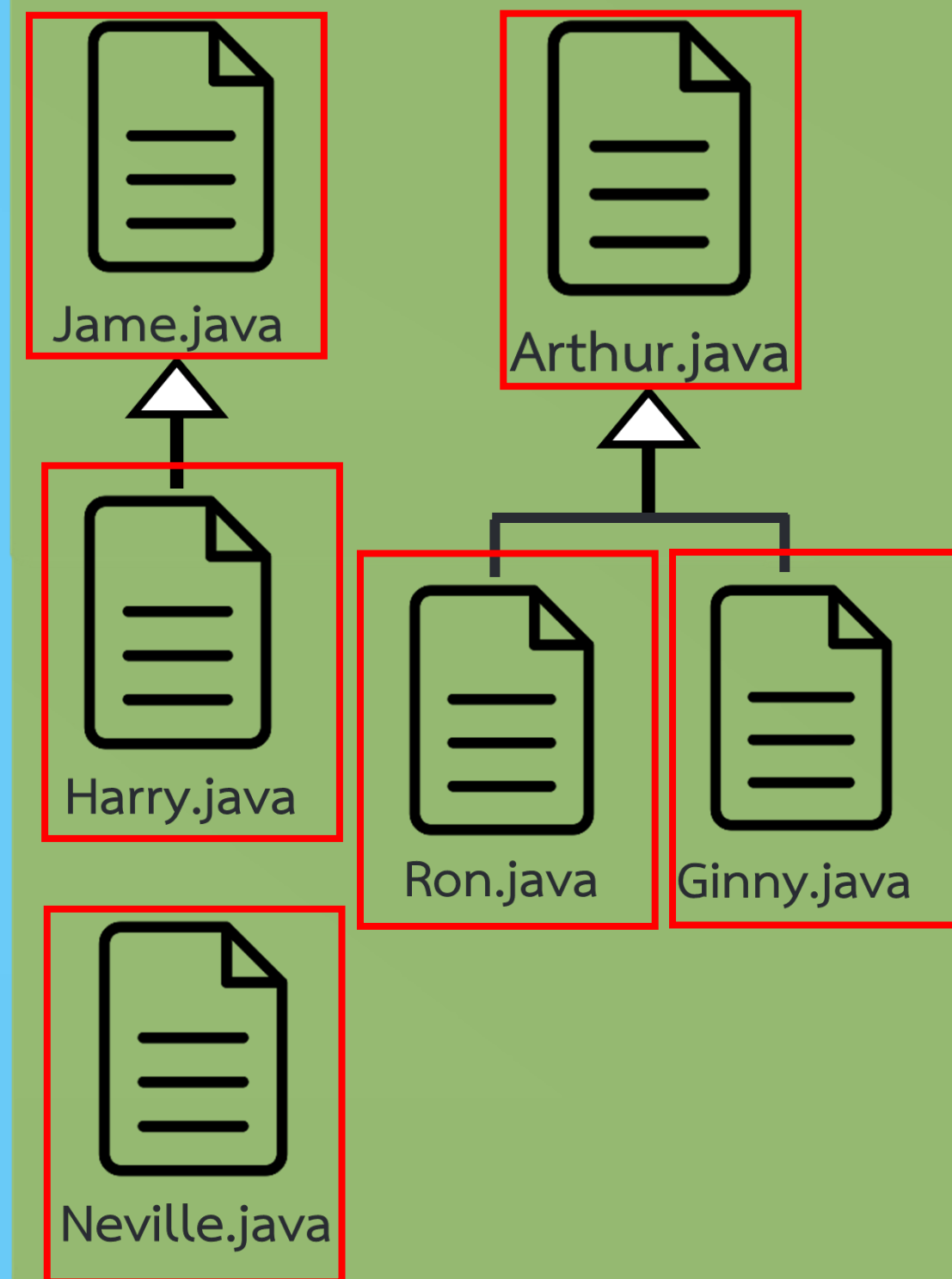


Access Modifier: **private**

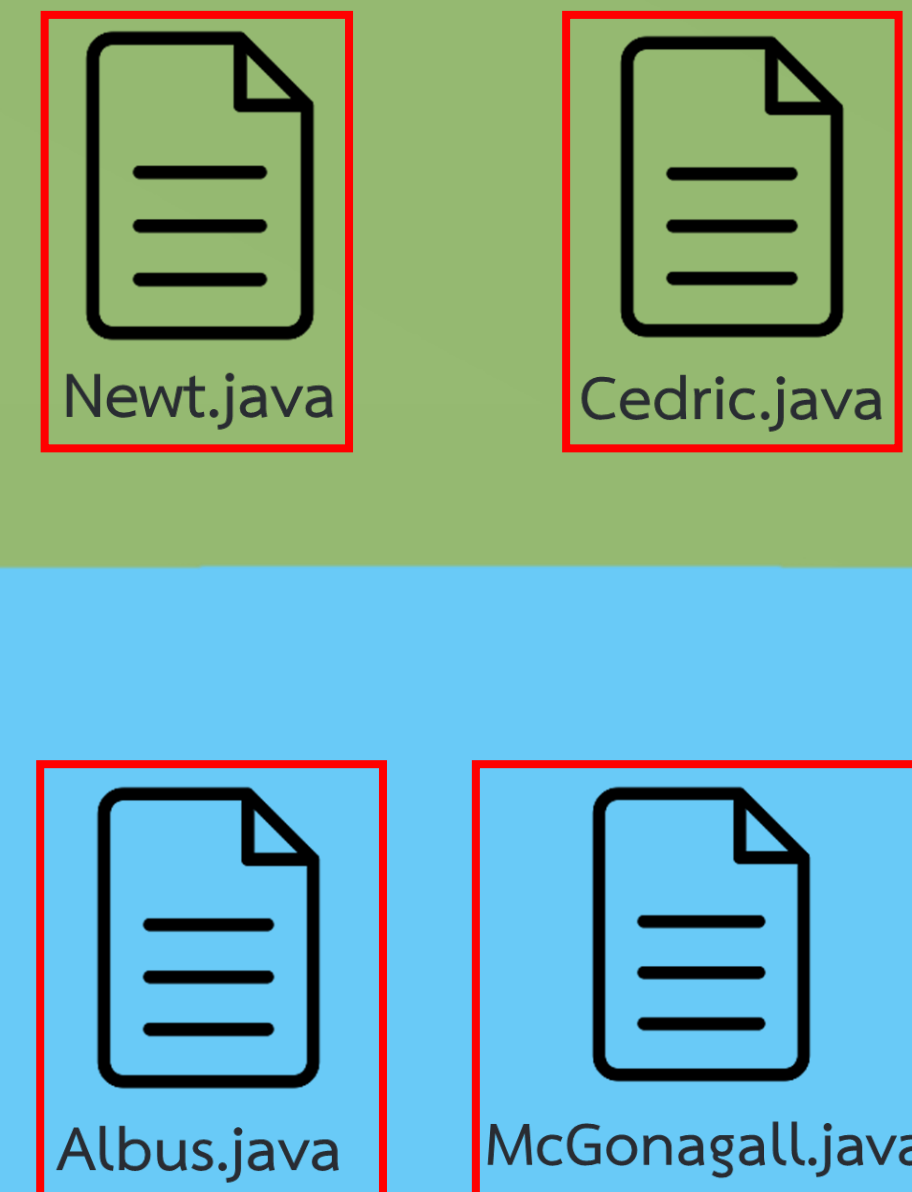


Hogwart

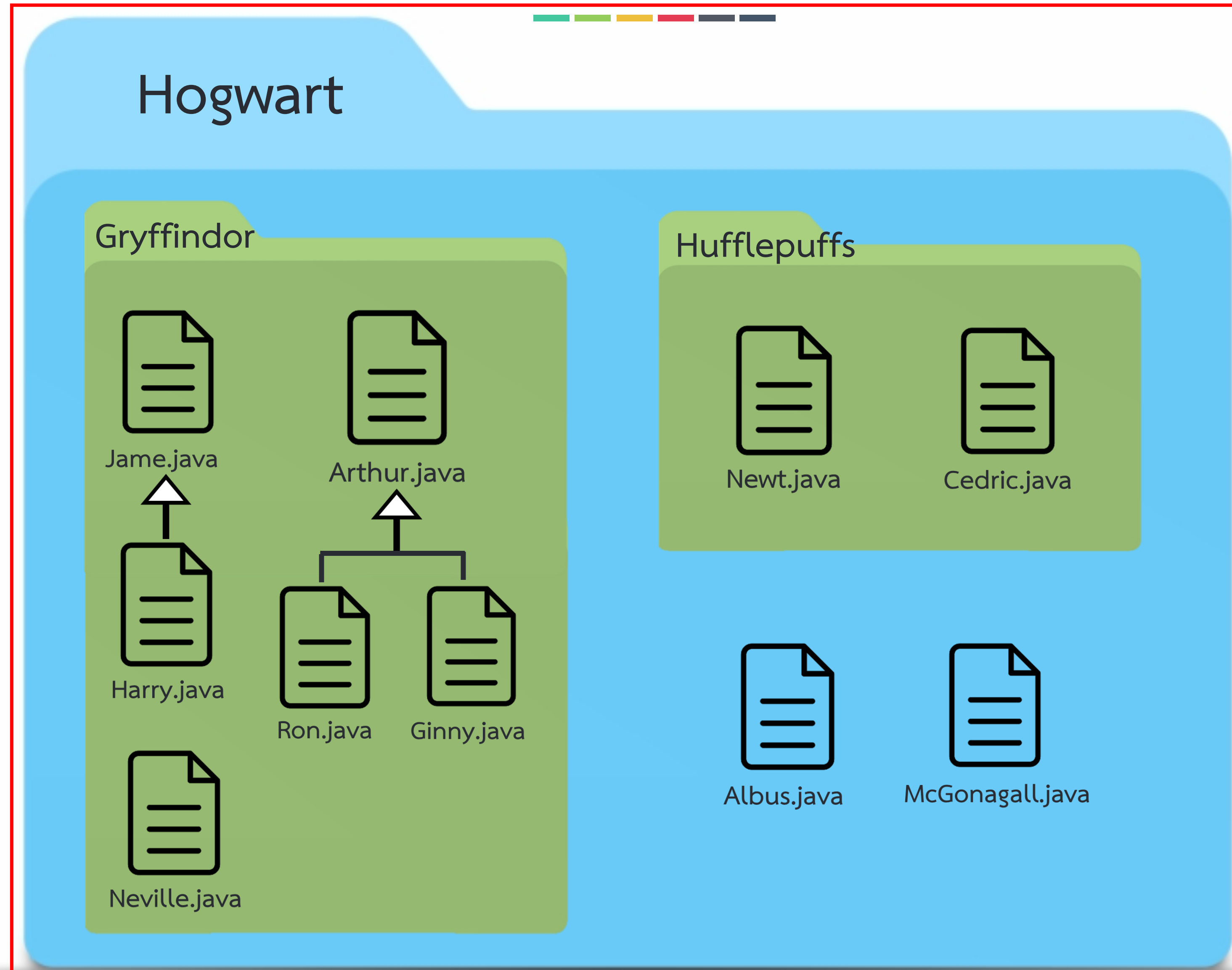
Gryffindor



Hufflepuffs



Access Modifier: **public**

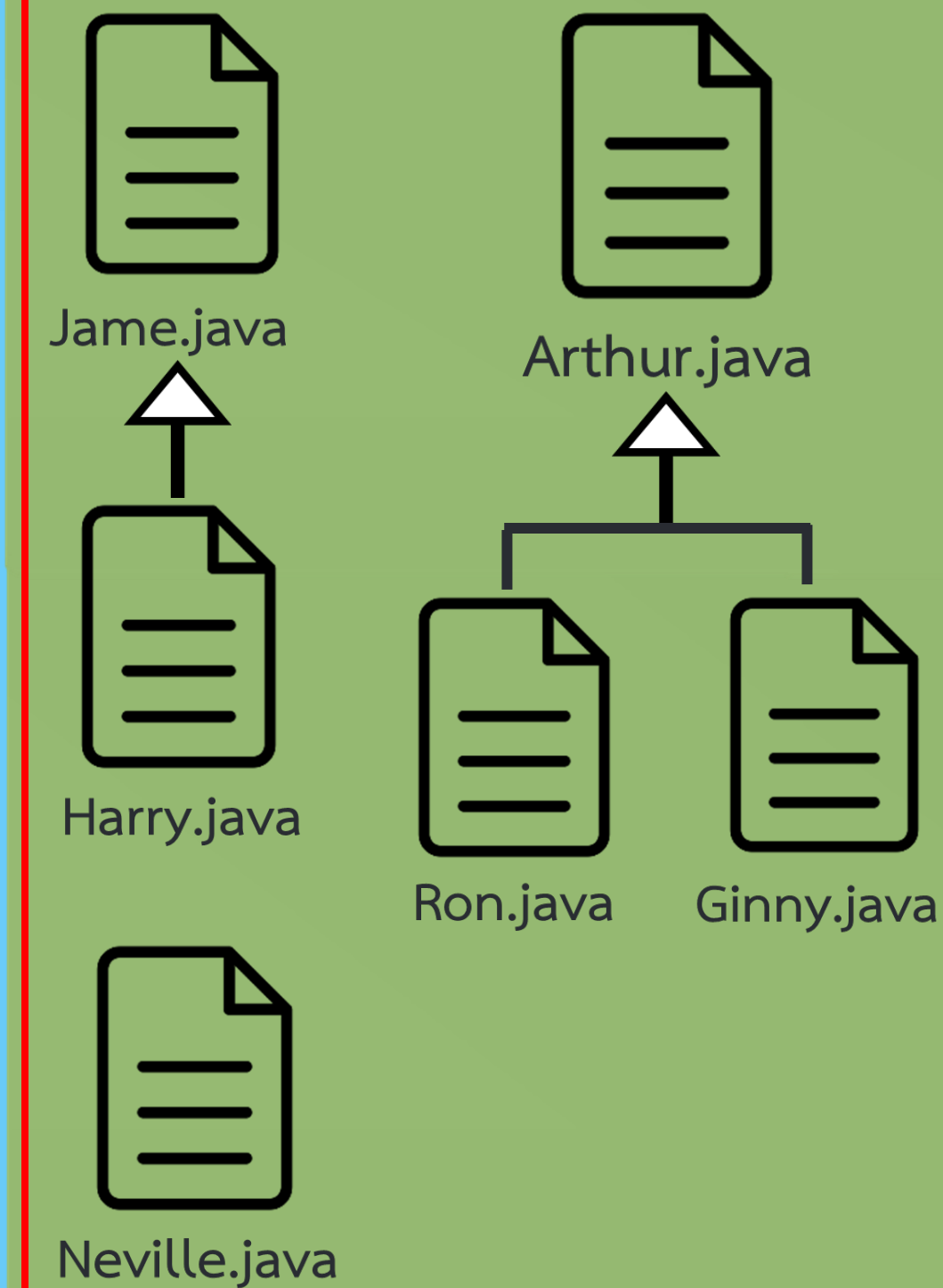


Access Modifier: **default**

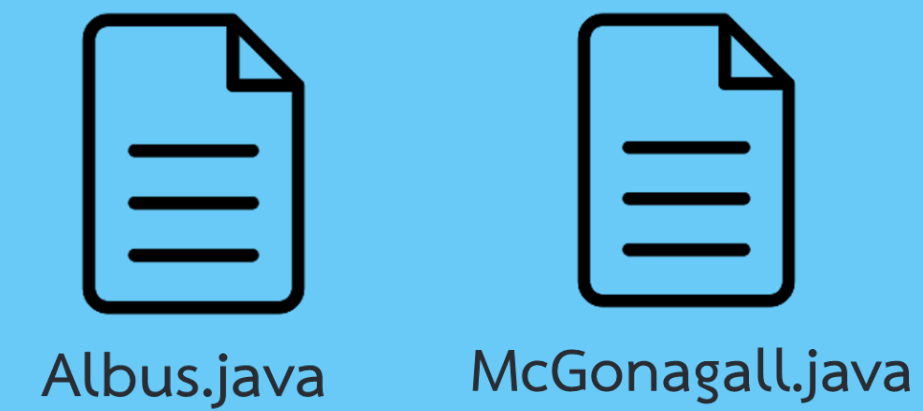
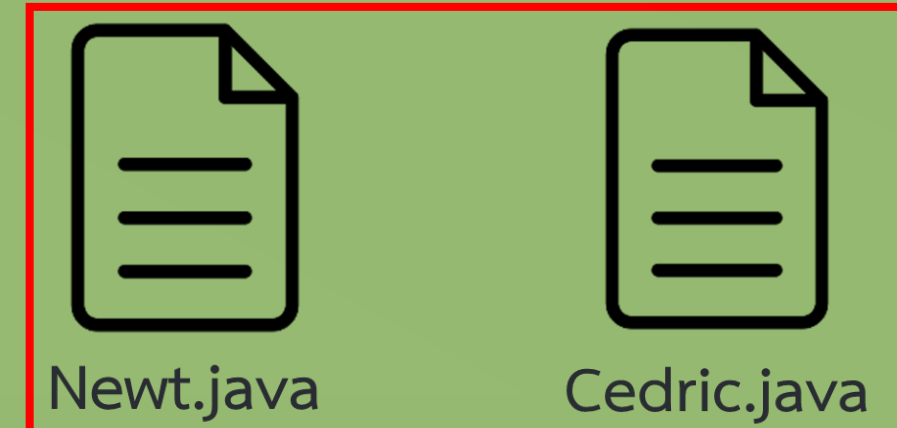


Hogwart

Gryffindor



Hufflepuffs



การห่อหุ้ม (Encapsulation)

1

Modifier

คือ ตัวกำหนดขอบเขต
และสิทธิการเข้าถึงคลาส
แอสทริคิวิท์ และ เมธอด

public

ทุกคลาสสามารถเข้าถึงและเรียกใช้งานได้

อนุญาตให้คลาสอื่นใช้ได้ ส่วนใหญ่จะให้เมธอดมีการเข้าใช้แบบ public

private

ภายในคลาสเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

วัตถุต่างคลาสไม่สามารถเข้าใช้ได้ และ สามารถปกป้องข้อมูลได้ ส่วนใหญ่จะให้แอสทริคิวิท์มีการเข้าใช้แบบ private

protected

คลาสแม่ลูกกันเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

เป็นคลาสที่เกี่ยวข้องกับการสืบทอดกันเท่านั้น

default

ทุกคลาภายใน package เดียวกันที่สามารถเข้าถึงและเรียกใช้งานได้

แพ็คเกจเป็นที่เก็บรวบรวมคลาสที่เกี่ยวข้องกัน หรือเปรียบได้กับโฟลเดอร์ ซึ่งการเข้าใช้แบบแพ็คเกจไม่ต้องใช้คำประกอบใดๆ

2

Method (set / get)

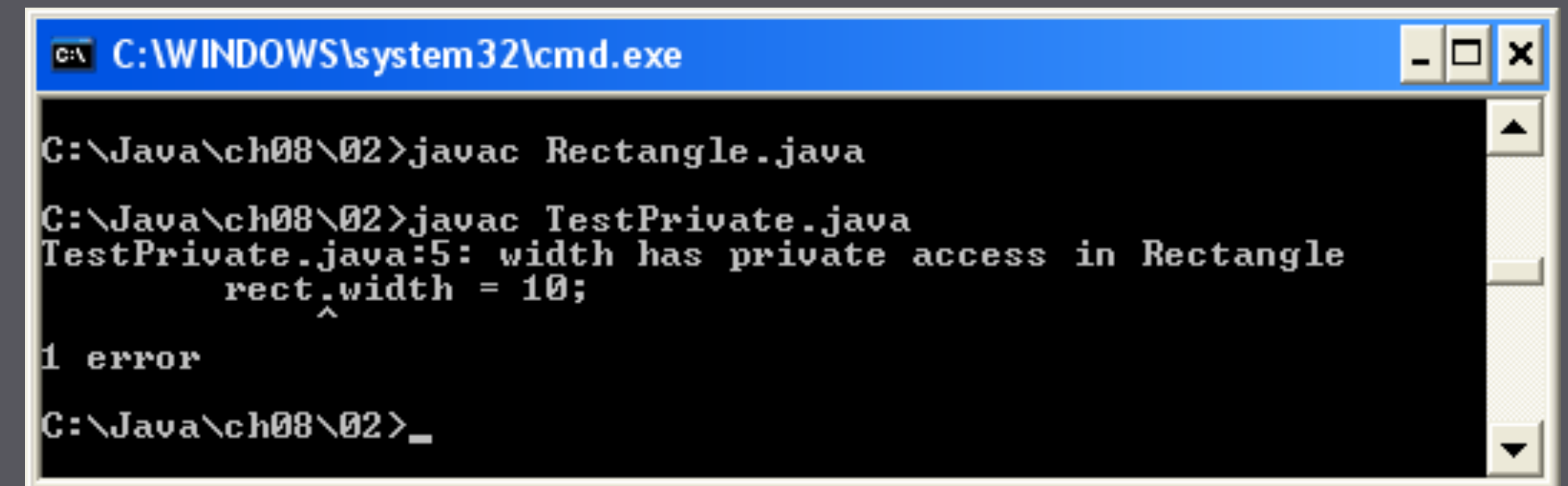
ตัวอย่างการห่อหุ้มที่เกิดข้อผิดพลาด

```
public class Student {  
    private String id;  
    private String name;  
    private double gpa;  
    public void setDetail(String i, String n, double g) {  
        id = i; name = n;  
        gpa = g;  
    }  
    public void showDetail() {  
        System.out.print("ID "+ id + " Name " + name + " GPA "+gpa);  
    }  
}  
public class Main {  
    public static void main( String[] args) {  
        Student s = new Student();  
        s.id = "9876";  
        s.name = "bank";  
        s.gpa = 3.8;  
        s.showDetail()  
    }  
}
```

} เกิดอะไรขึ้น ?

การห่อหุ้ม (Encapsulation)

```
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; error  
    }  
}
```



```
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>_
```

การห่อหุ้ม (Encapsulation) ข้อควรระวัง

```
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();  
        a1.name = "John";  
        Ant a2 = new Ant();  
        a2.name = "Alex";  
        a1.greeting(a2);  
        System.out.println(a1.name);  
    }  
}
```

อยู่ใน class

```
Exception in thread "main"  
java.lang.RuntimeException: Uncompilable  
code - name has private access in Ant  
    at Main.main(Ant.java:1)
```


การห่อหุ้ม (Encapsulation) ข้อควรระวัง

```
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) ;  
    }  
}
```

null

BUILD SUCCESSFUL (total time: 0 seconds)

การห่อหุ้ม (Encapsulation) ข้อควรระวัง

```
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";

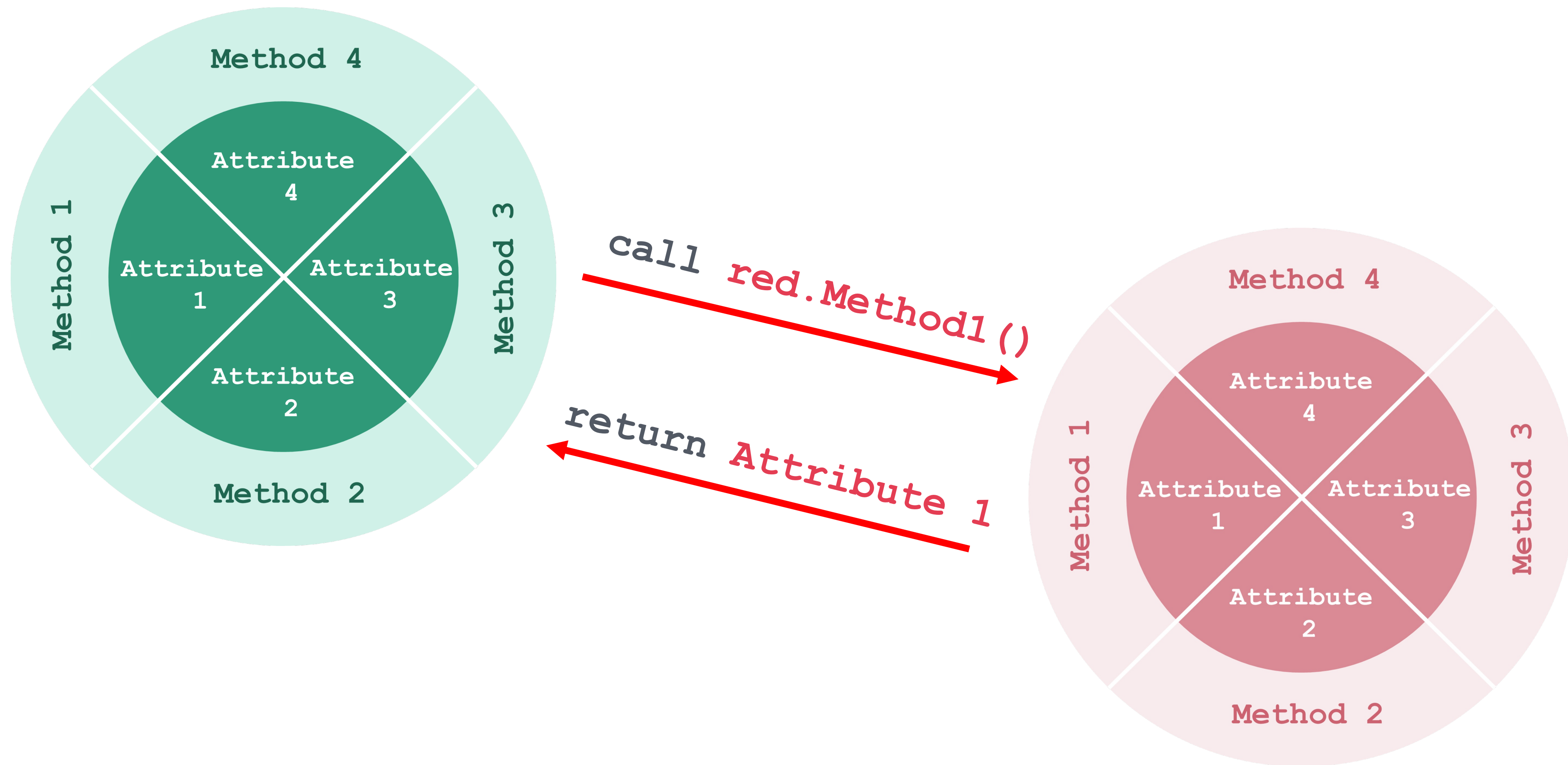
        a1.greeting(a2);
        System.out.println(a1.name);
    }
}
```

Alex

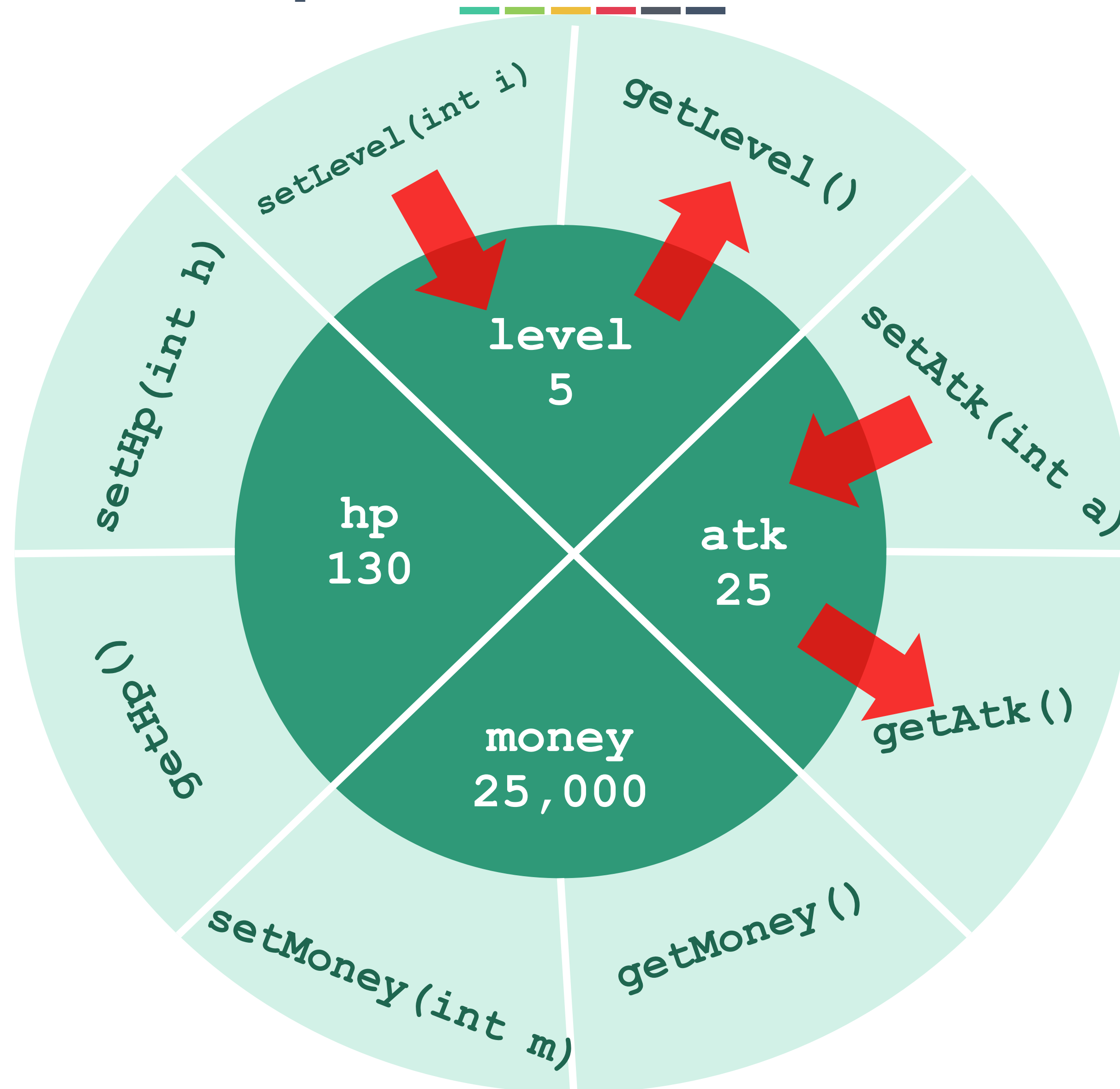
John

BUILD SUCCESSFUL (total time: 0 seconds)

การห่อหุ้ม (Encapsulation)



การห่อหุ้ม (Encapsulation)




method จำนวน = attr x 2

เมธอดแบบ setter



เมธอดแบบ accessor แบ่งออกได้เป็น 2 ประเภท ได้แก่ (1) เมธอดแบบ setter และ (2) เมธอดแบบ getter ซึ่งเมธอดแบบ setter จะใช้ในการกำหนดค่าของคุณลักษณะ โดยทั่วไปชื่อของเมธอดแบบ setter จะขึ้นต้นด้วยคำว่า set แล้วตามด้วยชื่อของคุณลักษณะ ซึ่งมีรูปแบบดังนี้

```
① public ② void ③ setAttributeName(dataType arg) {  
    attributeName = arg;  
}
```



เมธอดแบบ getter

เมธอดแบบ getter จะใช้ในการเรียกค่าของคุณลักษณะ โดยทั่วไปชื่อของเมธอดแบบ getter จะขึ้นต้นด้วยคำว่า get แล้วตามด้วยชื่อของคุณลักษณะ ซึ่งมีรูปแบบดังนี้

```
① public ② dataType ③ getAttributeName() {  
    return attributeName;  
}
```

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม



```
public class Student {  
    private double gpa;           → write only  
    public void setGpa(double g) { // หรือ public void setGPA(double g) {  
        gpa = g;  
    }                               → read only  
    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();  
        s1.setGpa(temp);  
        System.out.println("GPA: "+s1.getGpa());  
    }  
}
```


ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student {
    private double gpa;
    public void setGpa(double g) {
        if ((g < 0) || (g > 4.00)) {
            System.out.println("Incorrect Format!");
        } else {
            gpa = g;
        }
    }
    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();
        s1.setGpa(temp);
        System.out.println("GPA: "+s1.getGpa());
    }
}
```

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student{  
    private double gpa;  
    private String sex;  
  
    public void setGPA(double g){  
        gpa = g;  
    }  
    public double getGPA(){  
        return gpa;  
    }  
    public void setSex(String s){  
        sex = s;  
    }  
    public String getSex(){  
        return sex;  
    }  
}
```

ตัวอย่างโปรแกรมที่ใช้หลักการของการห่อหุ้ม

```
public class Student{
    private double gpa;
    private char sex;

    public void setGPA(double g){ gpa = g; }
    public double getGPA(){ return gpa; }
    public void setSex(String s){
        if (s.equals("female"))
            sex = 'f';
        else if (s.equals("male"))
            sex = 'm';
    }
    public String getSex(){
        if (sex == 'f')
            return "female";
        else if (sex == 'm')
            return "male";
    }
}
```

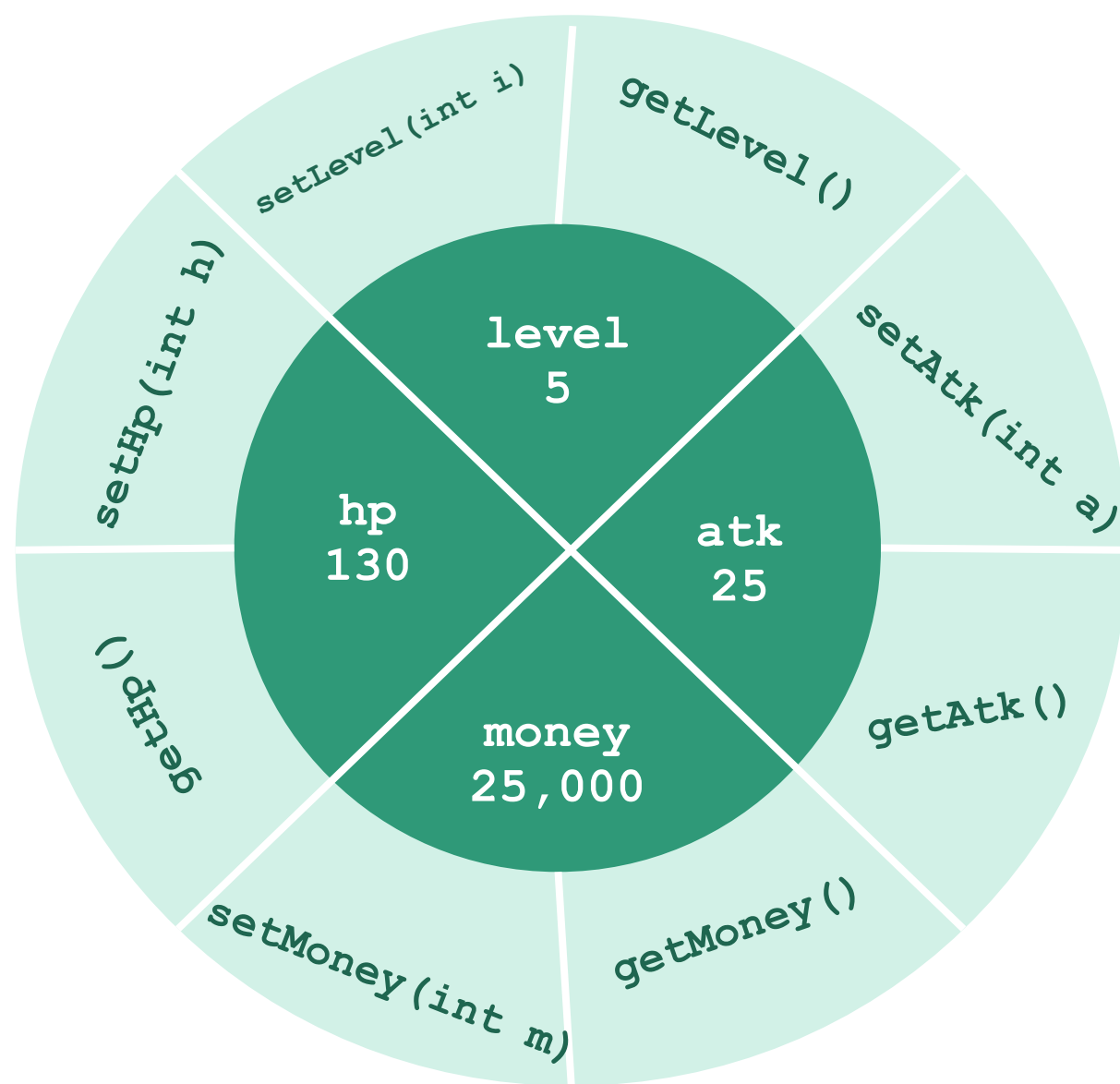
ตัวอย่างโปรแกรมที่*ไม่ใช้*หลักการของการห่อหุ้ม

```
public class Student {  
    public double 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 id, name;  
    private double gpa;  
    public void setID(String i) {  
        id = i;  
    } 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 getID() {  
        return id;  
    } 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;
    }

    .
    .
    .

}
  
```

ตัวอย่างการห่อหุ้มที่ 1

```
public class Student {
    private String id;
    private String name;
    private double gpa;
    public void setDetails(String i, String n, double g) {
        id = i;
        name = n;
        gpa = g;
    } public void showDetails() {
        System.out.println("ID: "+id + "\n Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}

public class Main {
    public static void main(String args[]) {
        Student ps = new Student();
        /*      ps.id = "12345";          illegal
               ps.name = "Thana";        illegal
               ps.gpa = 3.25;            illegal */
        ps.setDetails("12345", "Thana", 3.25);
        ps.showDetails();
    }
}
```

} error

ตัวอย่างการห่อหุ้มที่ 2

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

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) {  
        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);  
    }  
}
```

สามารถลดรูปได้

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

ผลลัพธ์:
A
A

ตัวอย่างการห่อหุ้มที่ 4

```
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()));  
    }  
}
```

Gamer.java

```
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);  
    }  
}
```

Main.java

ผลลัพธ์:

A

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

คีย์เวิร์ด `this`

- คีย์เวิร์ด `this` หมายถึง อ็อบเจกต์ของตัวเอง

ใช้เพื่อระบุตัวตนว่าฉันเป็นใคร

- เราสามารถที่จะเรียกใช้เมธอดหรือคุณลักษณะภายในคลาสได้ โดยใช้คีย์เวิร์ด `this` ซึ่งมีรูปแบบ ดังนี้

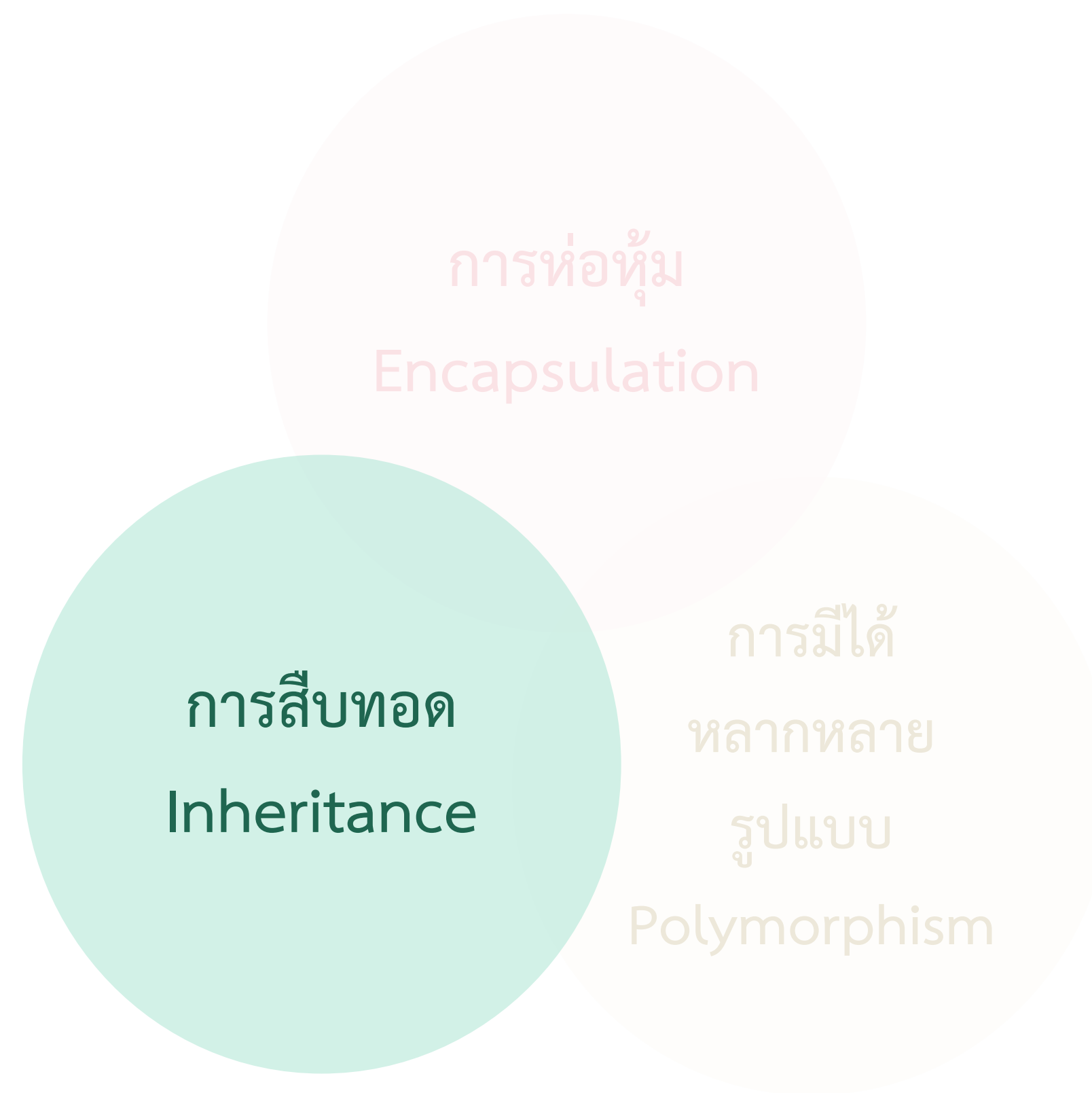
```
this.methodName () ;
```

```
this.attributeName
```

- โดยทั่วไปเราจะไม่ใช้คีย์เวิร์ด `this` ในคำสั่ง ยกเว้นในกรณีที่เป็น

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

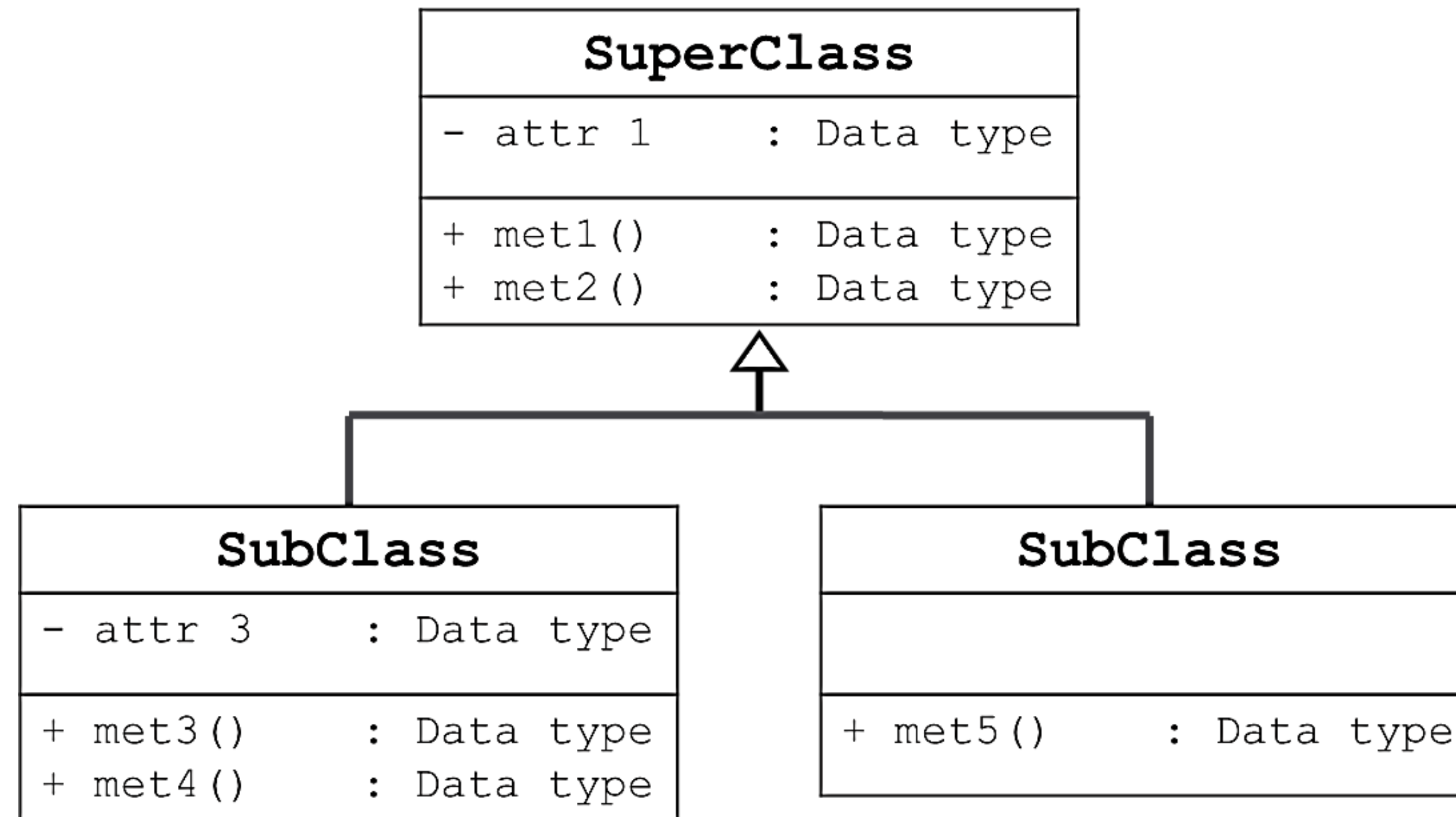
การเขียนโปรแกรมเชิงวัตถุ



การสืบทอด (Inheritance)

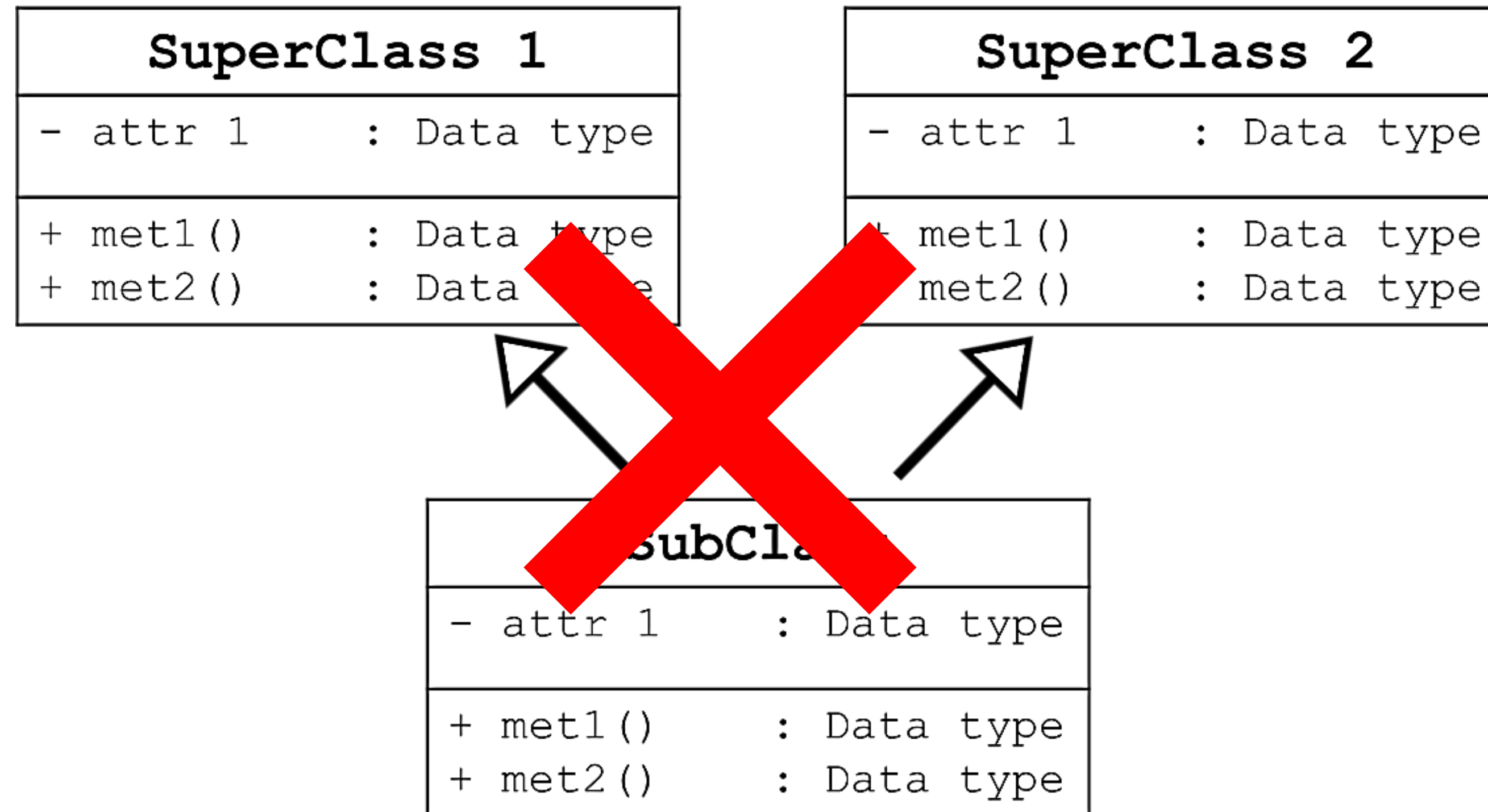
Law of Inheritance

1. Class แม่มีก็ Class ลูกก็ได้
2. Class ลูกมีได้แต่แม่เดียว
3. อะไรที่แม่มี → ลูกก็มี



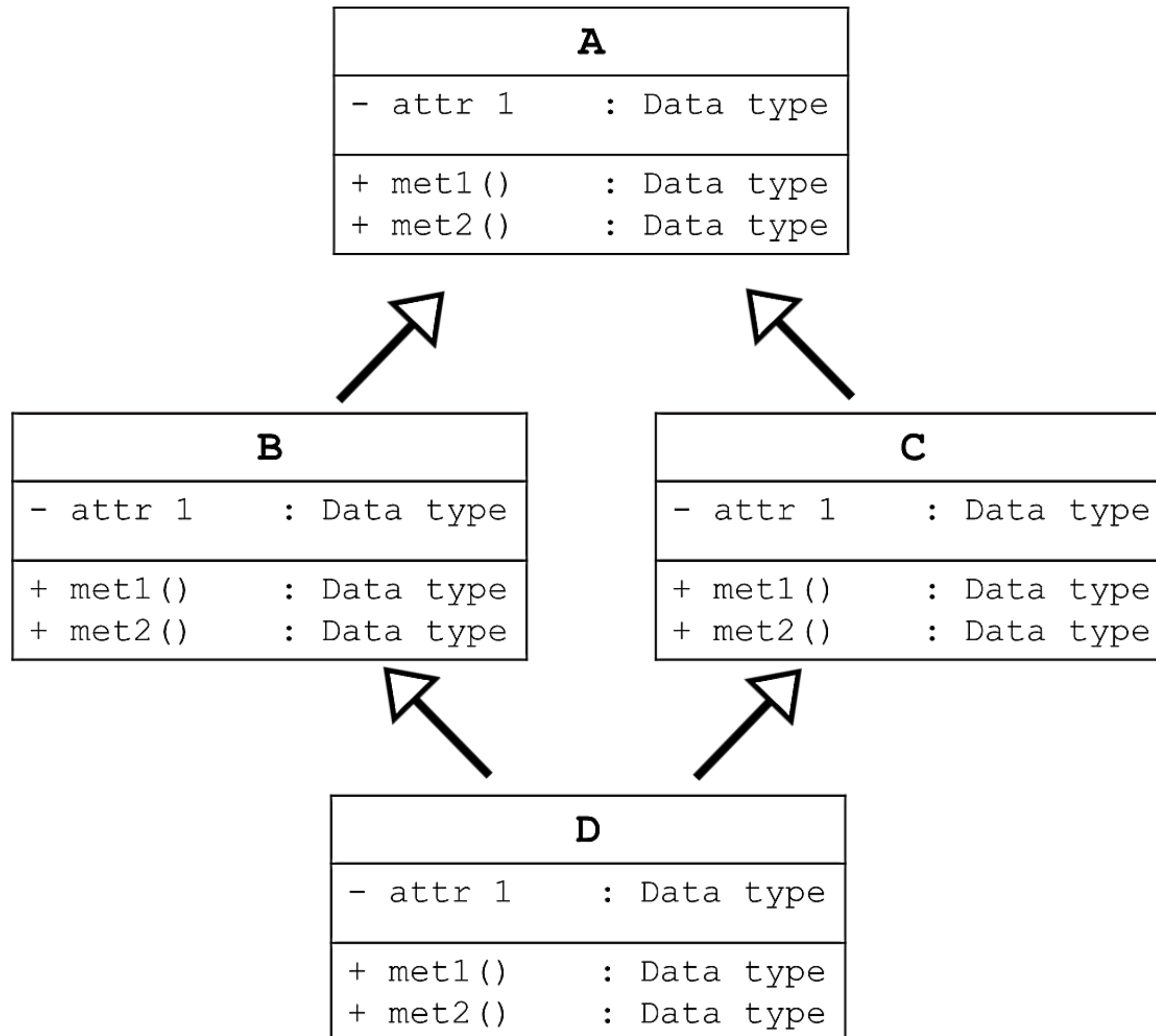
ในภาษาจาวาคลาสใหม่อาจจะได้แอททริบิวต์หรือเมธอดมาจากคลาสอื่น ๆ ซึ่งคลาสที่ได้รับแอททริบิวต์หรือเมธอดมาจากคลาสอื่นจะเรียกว่า **คลาสลูก (subclass)** และคลาสที่ถูกสืบทอดจะเรียกว่า **คลาสแม่ (superclass)** โดยทั่วไปแล้วคลาสลูกใด ๆ จะสืบทอดมาจากเพียงคลาสแม่เดียว ขณะที่คลาสแม่ใด ๆ จะมีลูกมาสืบทอดก็คลาสก็ได้ อย่างไรก็ตาม ถ้าคลาสนั้นไม่ได้สืบทอดมาจากคลาสใด ๆ หรือกล่าวคือ ไม่มีการกำหนดคลาสแม่แล้ว จาวาจะกำหนดให้คลาส Object เป็นคลาสแม่ทันที

การสืบทอด (Inheritance)



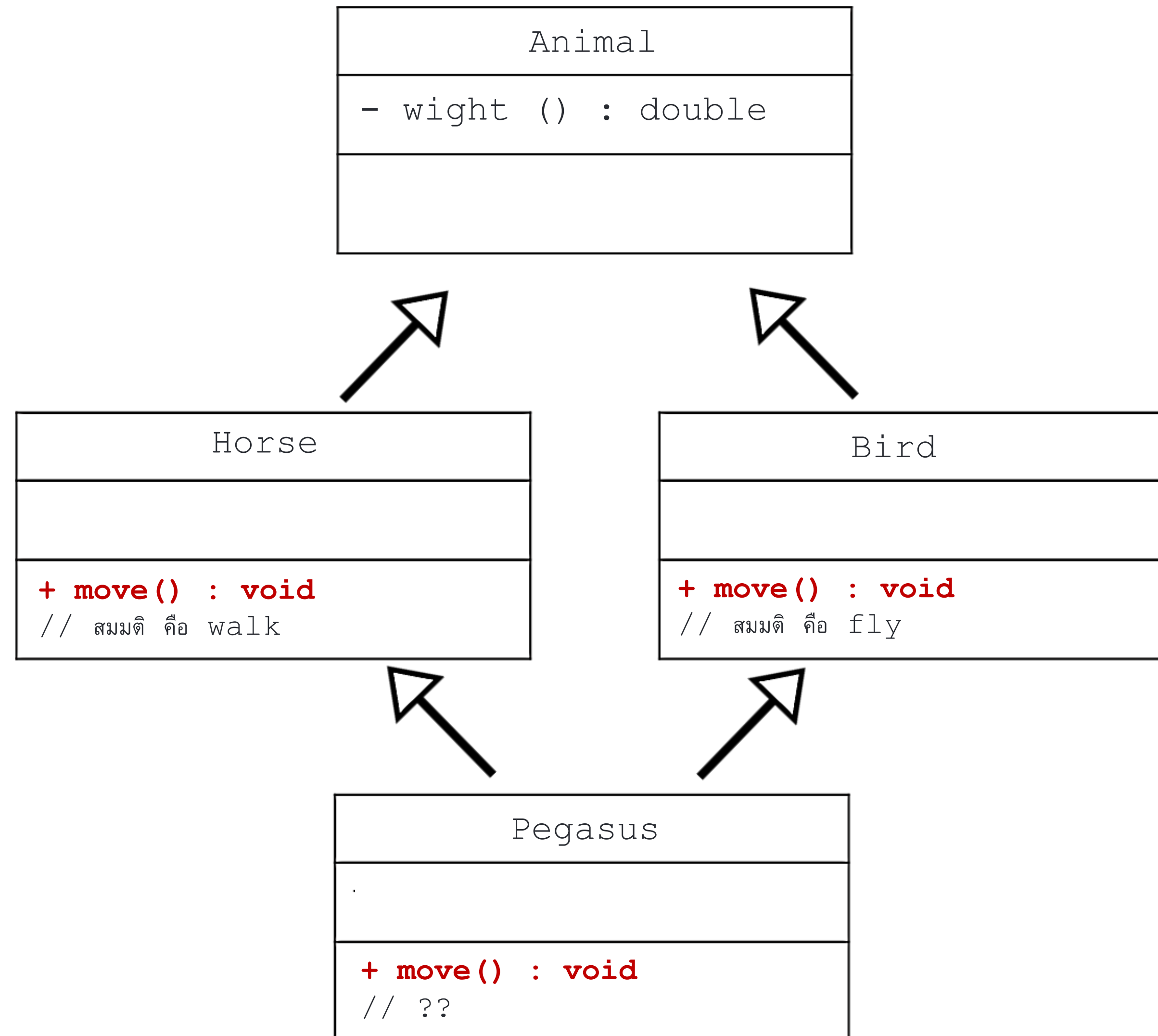
ในภาษาจาวาไม่อนุญาตให้มีการสืบทอดมากกว่า 1 คลาส หรือกล่าวคือ คลาสใด ๆ จะมี superclass ได้เพียง
 คลาสเดียวเพื่อหลีกเลี่ยงการเกิดปัญหา **“Diamond Problem”**

การสืบทอด (Inheritance)



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

การสืบทอด (Inheritance)



การสืบทอด (Inheritance)

การสืบทอด (Inheritance) สามารถลดปริมาณโค้ดที่มีความซ้ำซ้อนกันลง โดยการแชร์โค้ดหรือทรัพยากรในส่วนที่ใช้ร่วมกันกับคลาสลูกทั้งหลาย นอกจากนี้ เทคนิคการสืบทอดยังช่วยให้โปรแกรมของเรา**มีความยืดหยุ่นต่อการเปลี่ยนแปลง** ซึ่งสามารถสรุปได้ดังนี้



การสืบทอด (Inheritance)



- วิธีการออกแบบการสืบทอด ได้แก่



(1) Top-Down

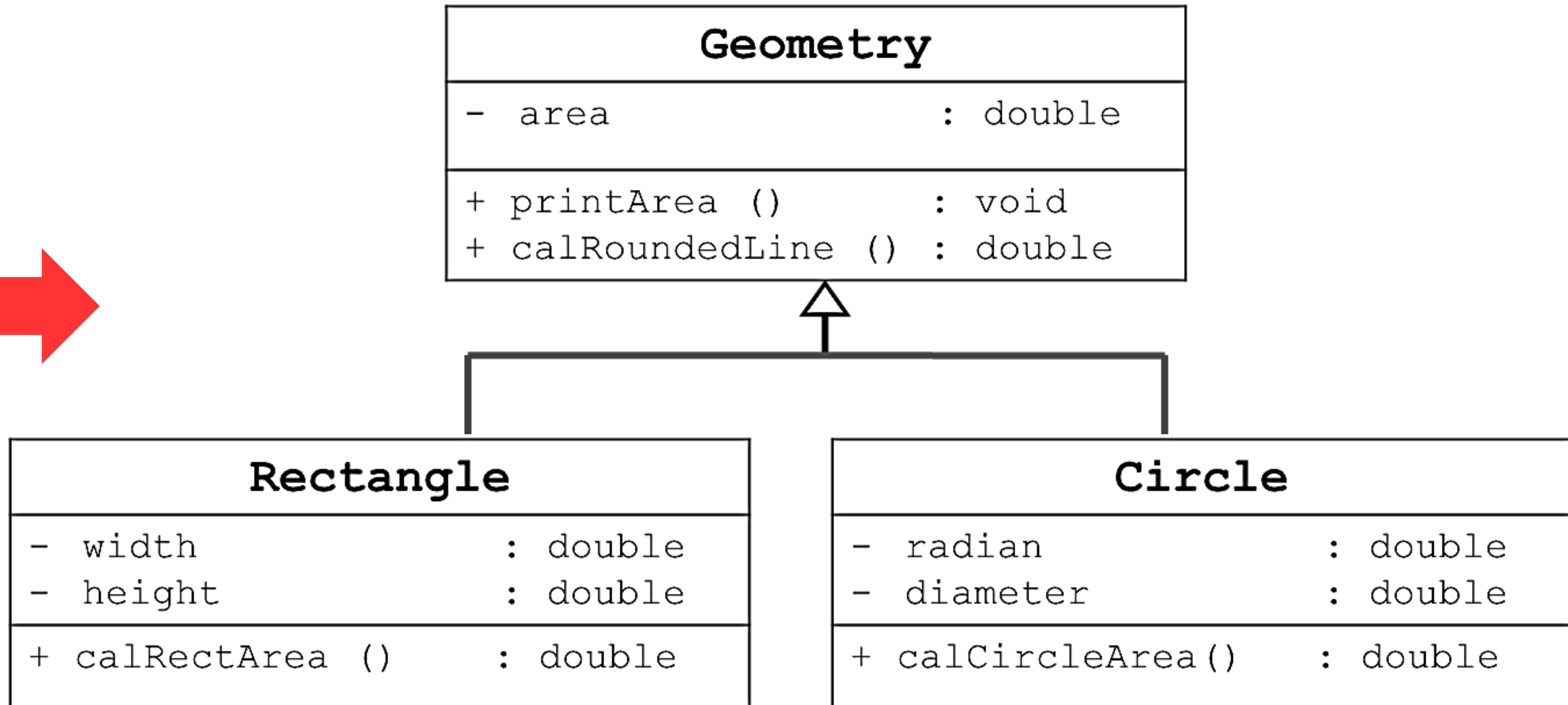
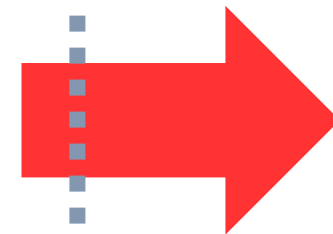


(2) Bottom-Up

การสืบทอด (Inheritance)

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

Geometry	
- width	: double
- height	: double
- area	: double
- radian	: double
- diameter	: double
+ printArea ()	: void
+ calRectArea ()	: double
+ calCircleArea ()	: double
+ calRoundedLine ()	: double

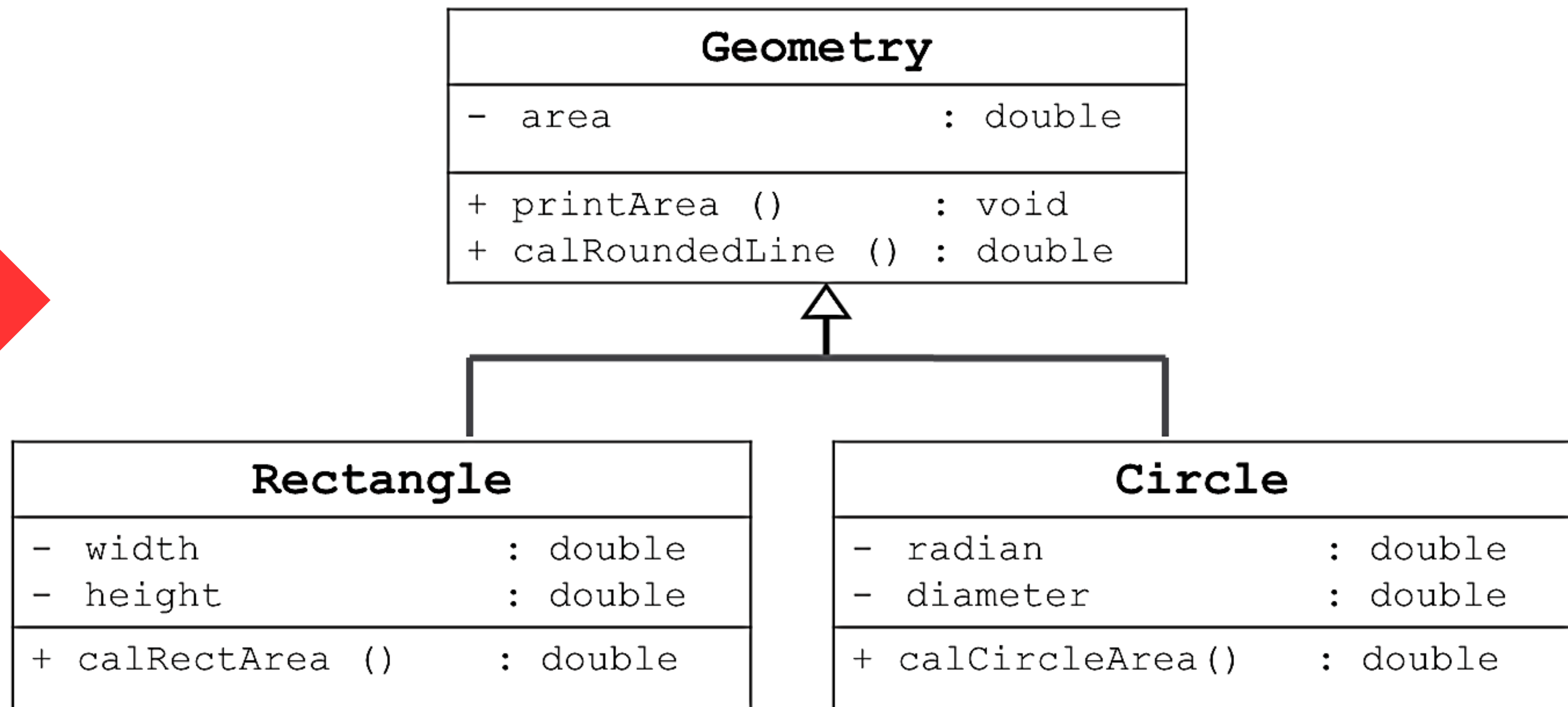
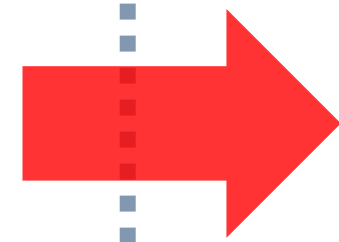


การสืบทอด (Inheritance)

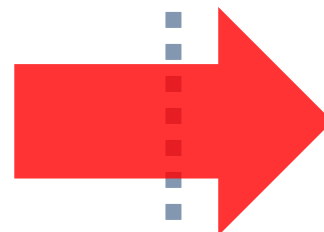
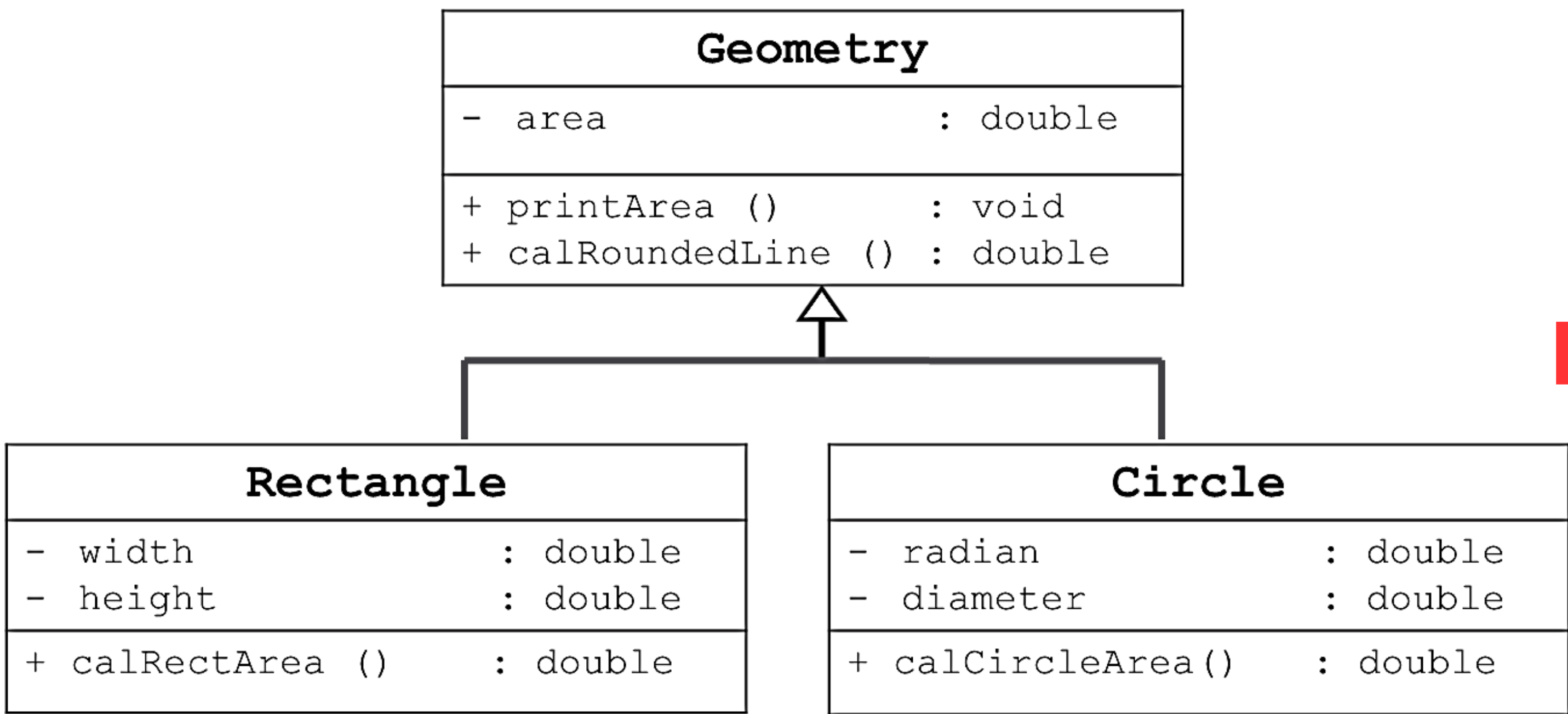
- วิธีการออกแบบการสืบทอด ได้แก่ (1) Top-Down และ (2) Bottom-Up

Circle	
- radian	: double
- diameter	: double
- area	: double
+ calCircleArea()	: double
+ printArea ()	: void
+ calRoundedLine ()	: double

Rectangle	
- width	: double
- height	: double
- area	: double
+ calRectArea ()	: double
+ printArea ()	: void
+ calRoundedLine ()	: double



การสืบทอด (Inheritance)

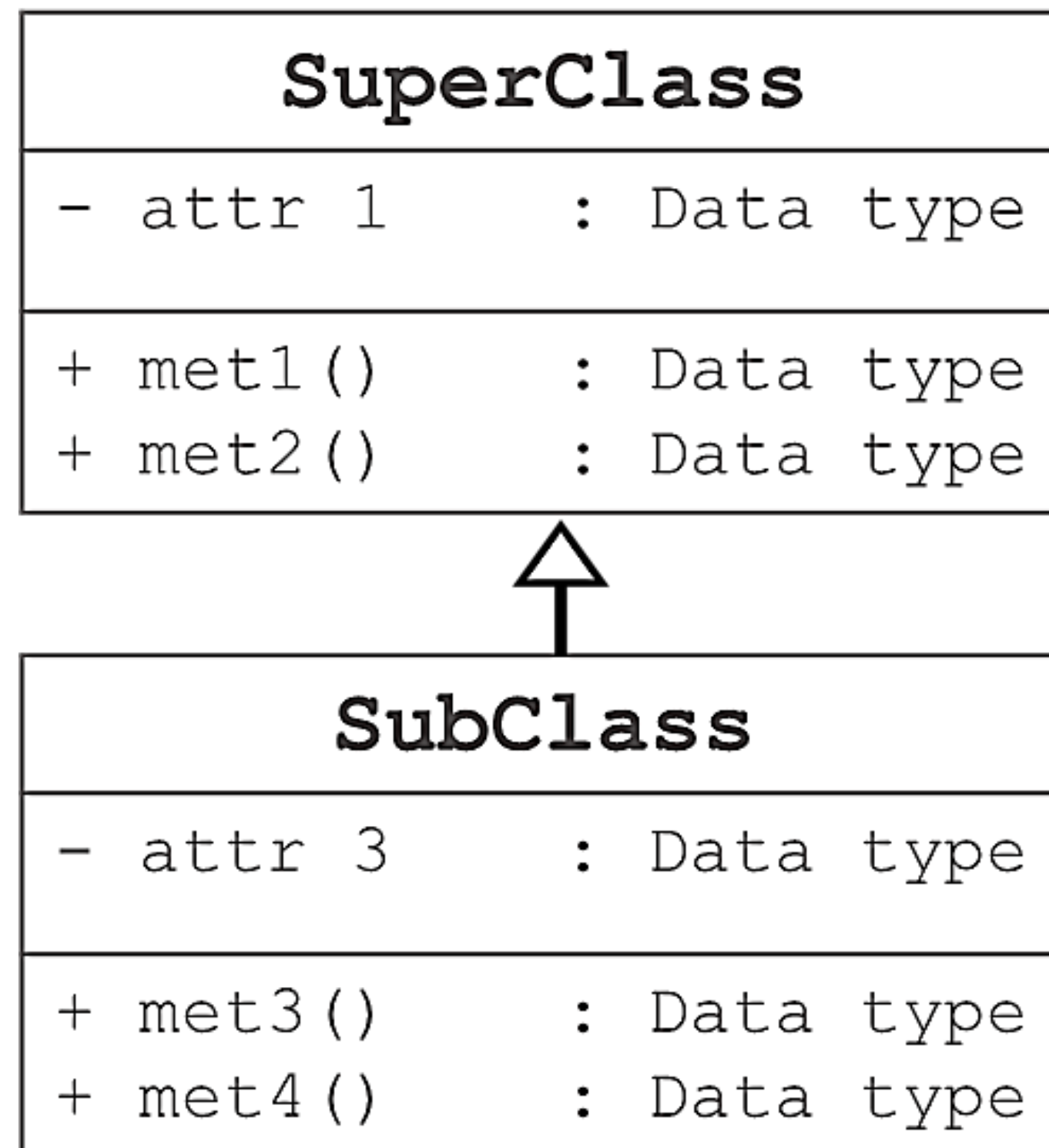


Geometry	
- area	: double
+ printArea ()	: void
+ calRoundedLine ()	: double

Circle	
- radian	: double
- diameter	: double
- area	: double
+ calCircleArea ()	: double
+ printArea ()	: void
+ calRoundedLine ()	: double

Rectangle	
- width	: double
- height	: double
- area	: double
+ calRectArea ()	: double
+ printArea ()	: void
+ calRoundedLine ()	: double

วิธีการสืบทอด



```

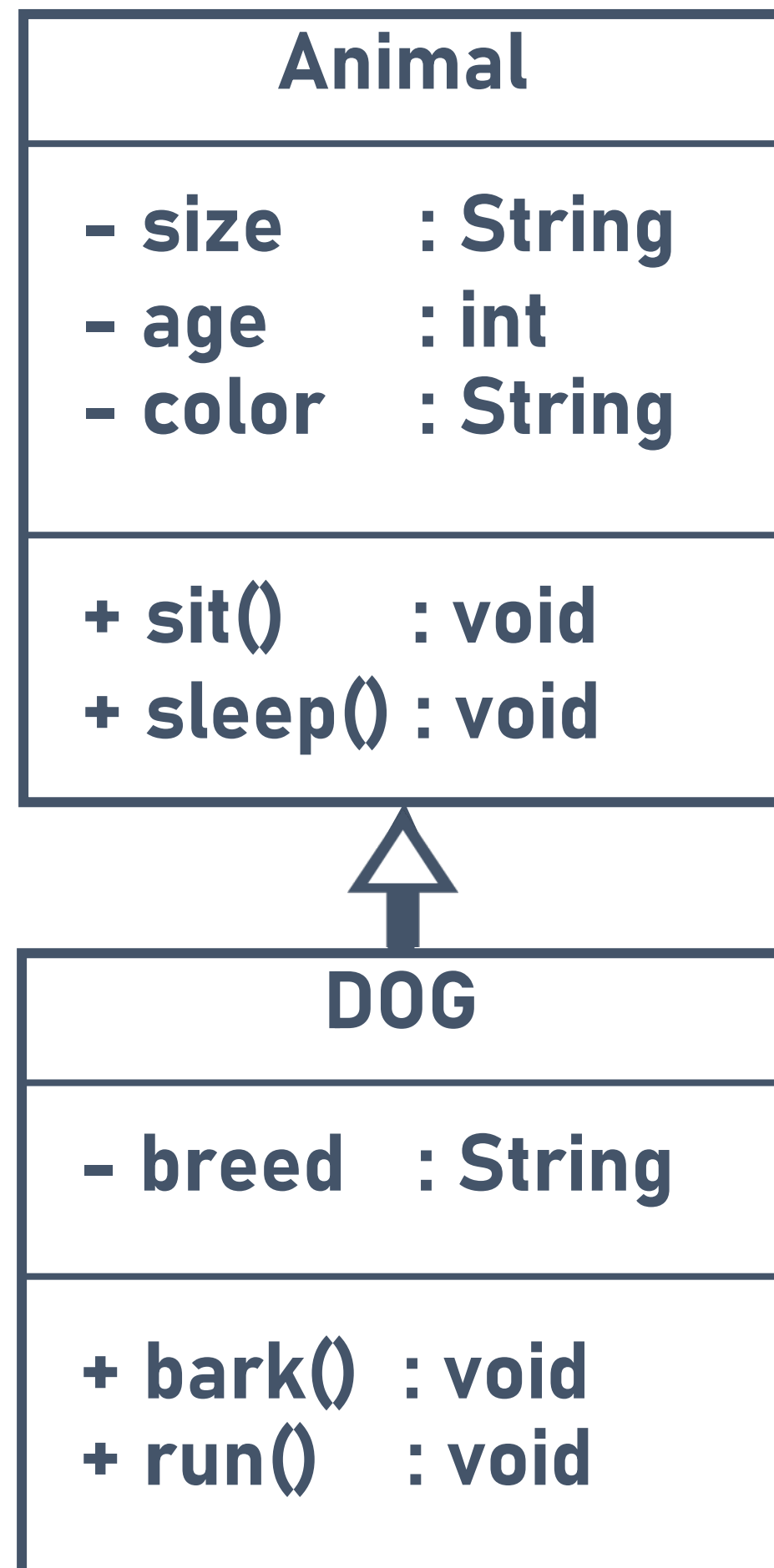
public class SuperClass{
    private datatype attr1;

    public void met1(){...}
    public int met2(){...}
}
public class SubClass extends SuperClass {
    private datatype attr3;

    public double met3(){...}
    public String met4(){...}
}
  
```

ตัวอย่างการสืบทอด

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



```

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(){...}
}
  
```

ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent



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

(1) สร้างคลาสขึ้นมาใหม่โดยไม่อ้างอิงกับคลาสเดิมที่ชื่อ Student

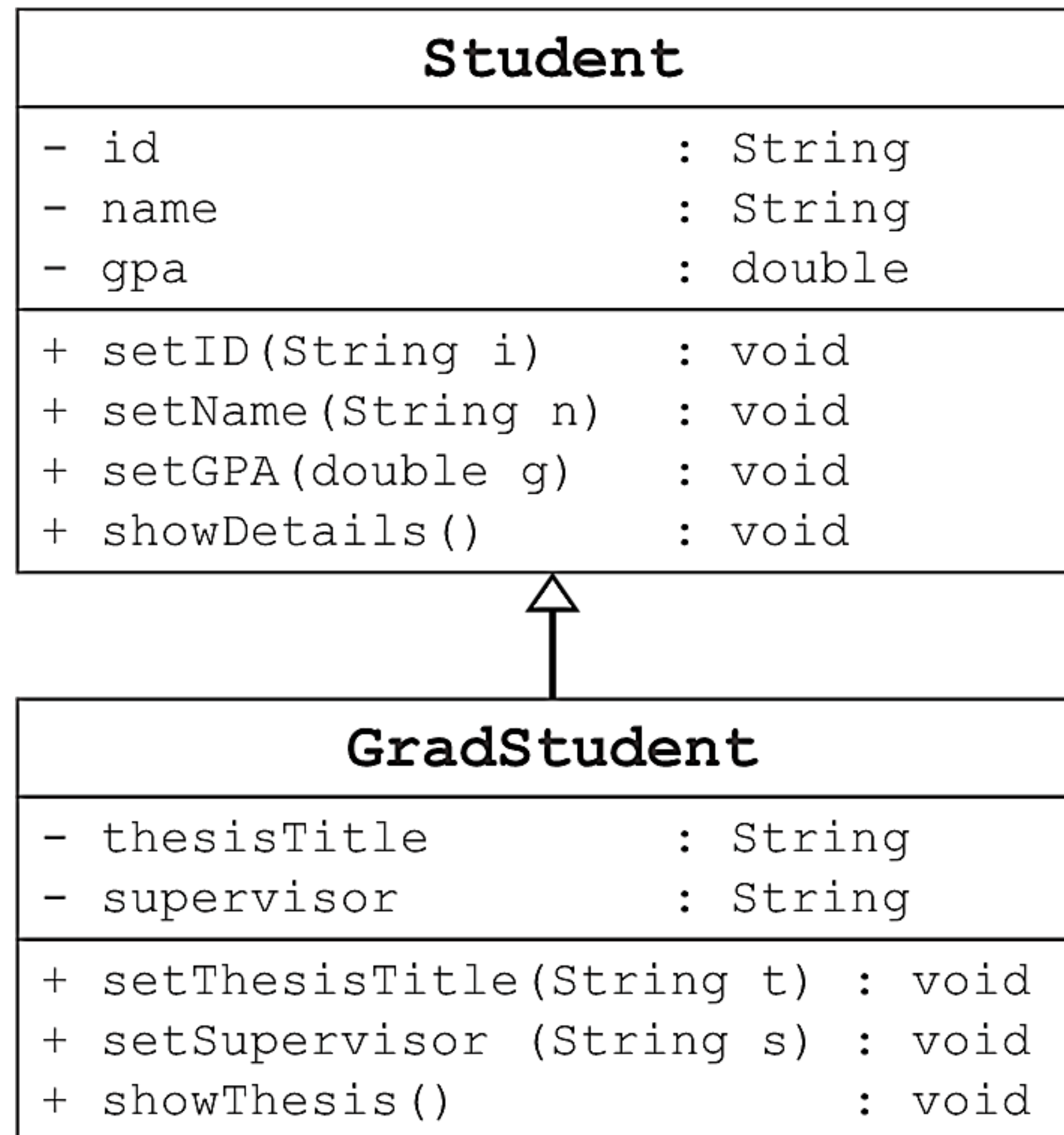
Student	
- id	: String
- name	: String
- gpa	: double
+ setID(String i)	: void
+ setName(String n)	: void
+ setGPA(double g)	: void
+ showDetails()	: void

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

ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent



(2) สร้างคลาสที่สืบทอดมาจากคลาสเดิมที่ชื่อ Student



ตัวอย่างกรณีศึกษา คลาส Student และ GradStudent

(1) สร้างคลาสขึ้นมาใหม่โดยไม่อ้างอิงกับคลาสเดิม

```
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

(2) สร้างคลาสที่สืบทอดมาจากคลาสเดิมที่ชื่อ Student

```
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 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



สำหรับการเขียนโปรแกรมเชิงวัตถุจะสามารถแบ่งแยกความสัมพันธ์ออกเป็น 2 แบบ ได้แก่

- **Is-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**เป็น**อีกสิ่งหนึ่งด้วย เช่น
 - แมวเป็นสัตว์
 - ต้นส้มเป็นต้นไม้
 - รถยนต์เป็นพาหนะ
- **Has-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**มี**อีกสิ่งหนึ่งด้วย เช่น
 - คนมีความสามารถว่ายน้ำ
 - ปลามีความสามารถว่ายน้ำ

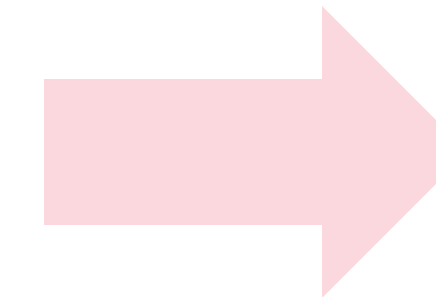
การพิจารณาการสืบทอดโดยอ้างอิงจากความสัมพันธ์



สำหรับการเขียนโปรแกรมเชิงวัตถุจะสามารถแบ่งแยกความสัมพันธ์ออกเป็น 2 แบบ ได้แก่

- **Is-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**เป็น**อีกสิ่งหนึ่งด้วย เช่น

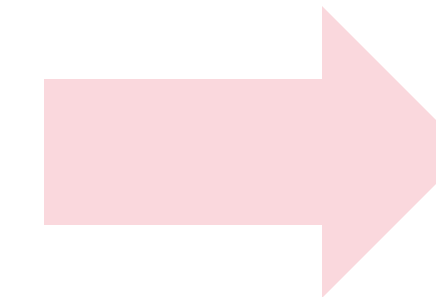
- แมวเป็นสัตว์
- ต้นส้มเป็นต้นไม้
- รถยนต์เป็นพาหนะ



การสืบทอด
Inheritance

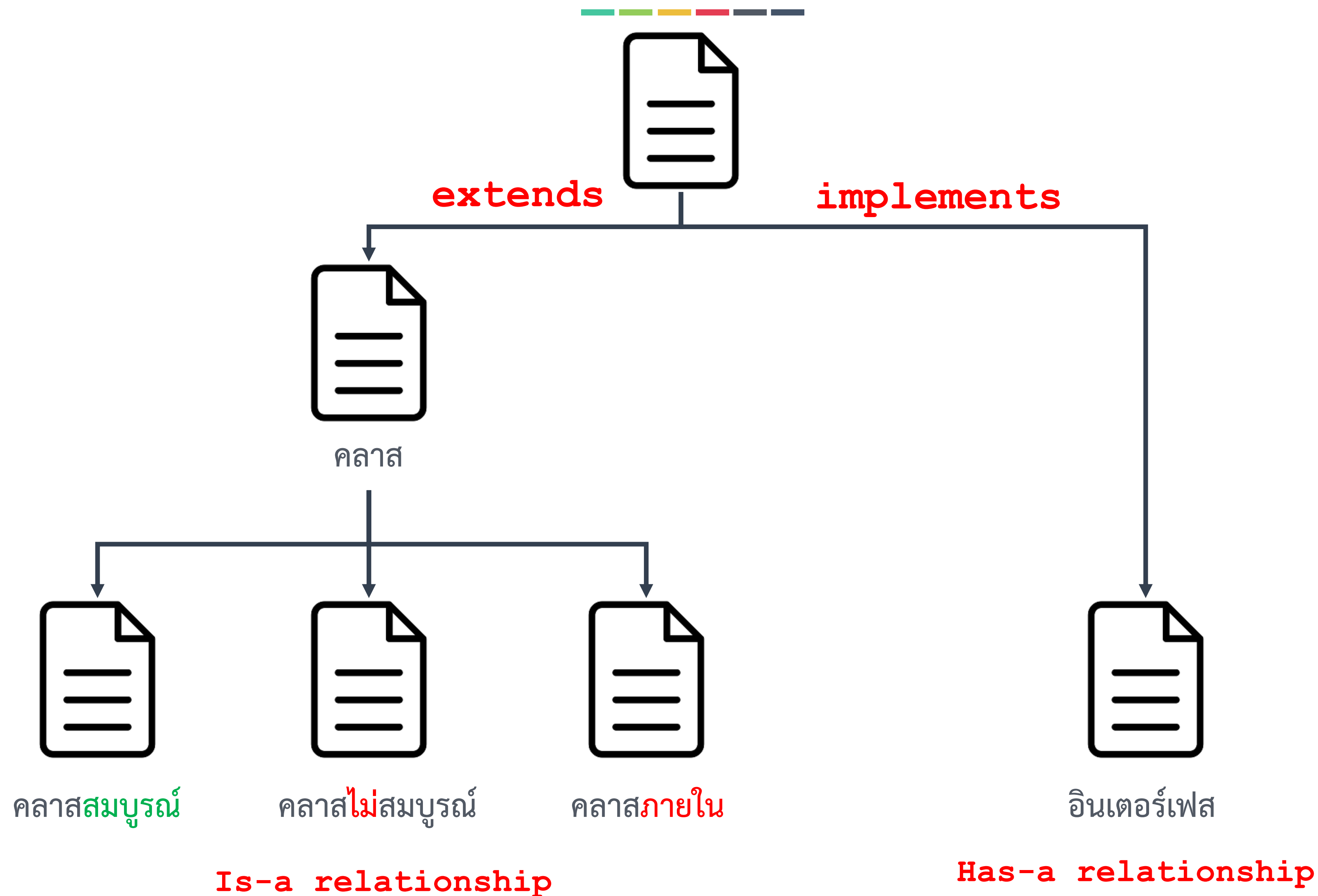
- **Has-a relationship** คือ ความสัมพันธ์ที่สิ่งหนึ่ง**มี**อีกสิ่งหนึ่งด้วย เช่น

- คนมีความสามารถว่ายน้ำ
- ปลามีความสามารถว่ายน้ำ



Implement

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;  
}
```

ทำไมถึง Error ?

```
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 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

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

วิธีการแก้ไขแบบที่ 1



```
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;
    }
}
```

```
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

การสืบทอด (Inheritance)



1

Modifier

คือ ตัวกำหนดขอบเขต
และสิทธิการเข้าถึงคลาส
แอสทริคิวิท์ และ เมธอด

public

ทุกคลาสสามารถเข้าถึงและเรียกใช้งานได้

อนุญาตให้คลาสอื่นใช้ได้ ส่วนใหญ่จะให้เมธอดมีการเข้าใช้แบบ public

private

ภายในคลาสเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

วัตถุต่างคลาสไม่สามารถเข้าใช้ได้ และ สามารถปกป้องข้อมูลได้ ส่วนใหญ่จะให้แอสทริคิวิท์มีการเข้าใช้แบบ private

protected

คลาสแม่ลูกกันเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

เป็นคลาสที่เกี่ยวข้องกับการสืบทอดกันเท่านั้น

default

ทุกคลาสภายใน package เดียวกันที่สามารถเข้าถึงและเรียกใช้งานได้

แพ็คเกจเป็นที่เก็บรวบรวมคลาสที่เกี่ยวข้องกัน หรือเปรียบได้กับโฟลเดอร์ ซึ่งการเข้าใช้แบบแพ็คเกจไม่ต้องใช้คำประกอบใดๆ

การสืบทอด (Inheritance)



1

Modifier

คือ ตัวกำหนดขอบเขต
และสิทธิการเข้าถึงคลาส
แอสทริคิวิท์ และ เมธอด

public

ทุกคลาสสามารถเข้าถึงและเรียกใช้งานได้

อนุญาตให้คลาสอื่นใช้ได้ ส่วนใหญ่จะให้เมธอดมีการเข้าใช้แบบ public

private

ภายในคลาสเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

วัตถุต่างคลาสไม่สามารถเข้าใช้ได้ และ สามารถปกป้องข้อมูลได้ ส่วนใหญ่จะให้แอสทริคิวิท์มีการเข้าใช้แบบ private

protected

คลาสแม่ลูกกันเท่านั้นที่สามารถเข้าถึงและเรียกใช้งานได้

เป็นคลาสที่เกี่ยวข้องกับการสืบทอดกันเท่านั้น

default

ทุกคลาสภายใน package เดียวกันที่สามารถเข้าถึงและเรียกใช้งานได้

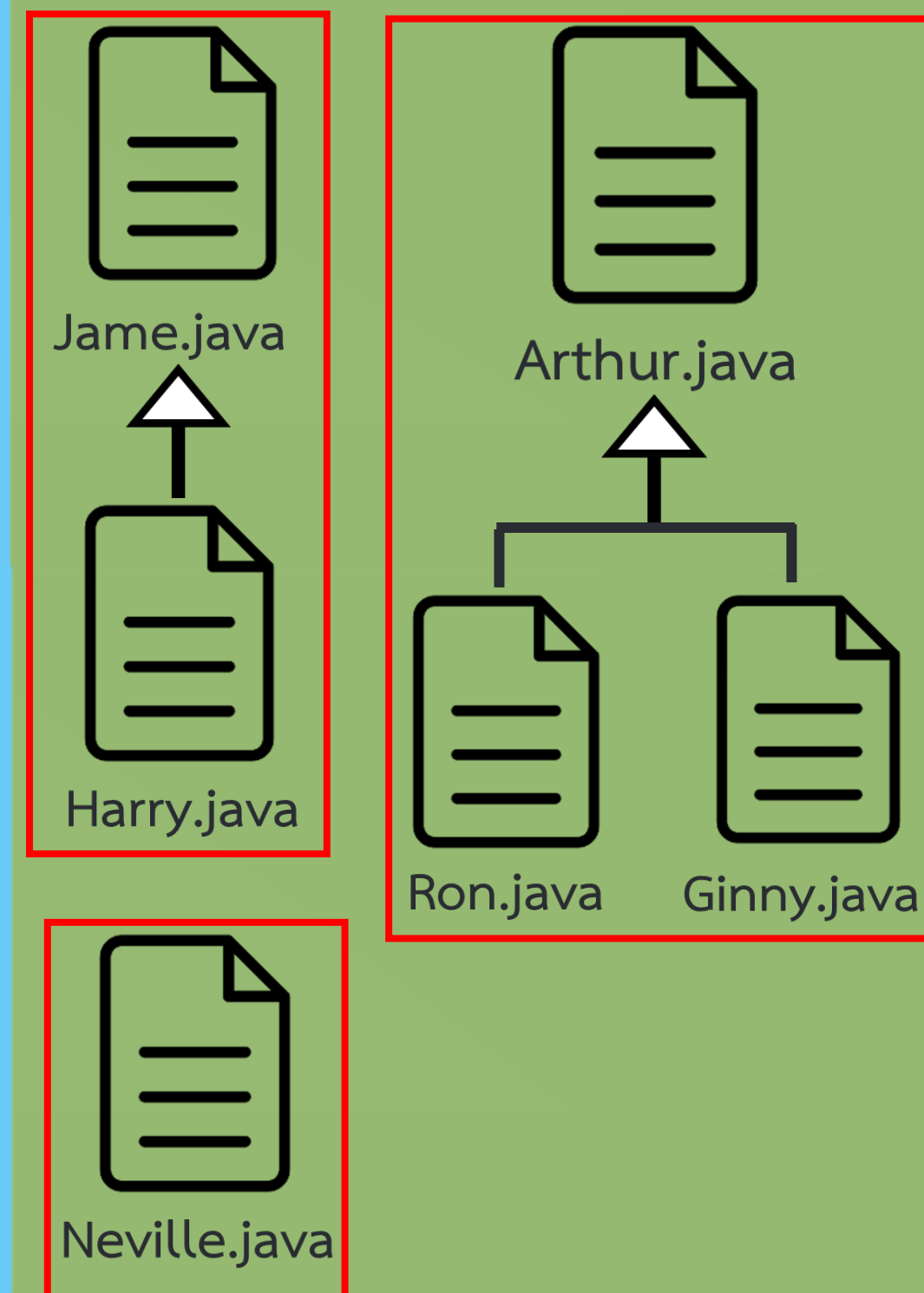
แพ็คเกจเป็นที่เก็บรวบรวมคลาสที่เกี่ยวข้องกัน หรือเปรียบได้กับโฟลเดอร์ ซึ่งการเข้าใช้แบบแพ็คเกจไม่ต้องใช้คำประกอบใดๆ

Access Modifier: **protected**

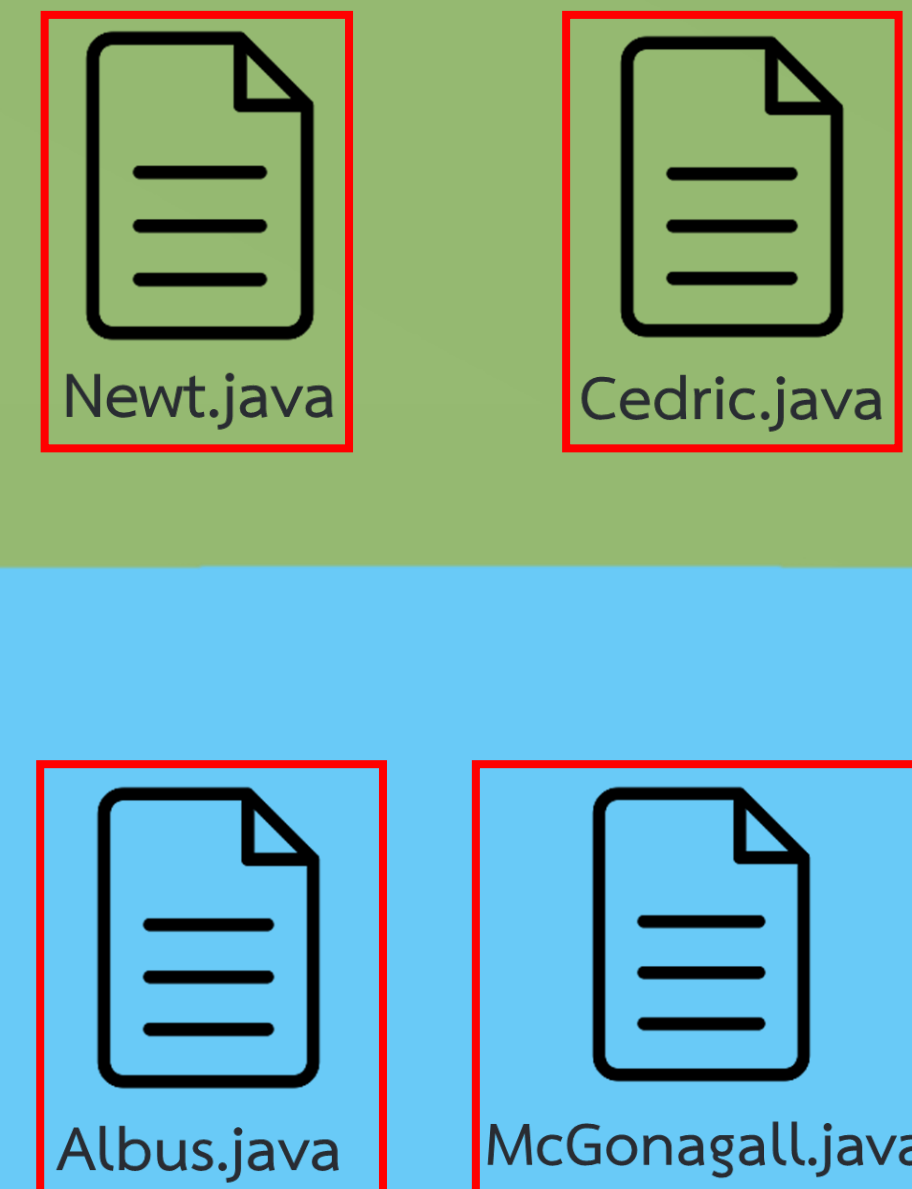


Hogwart

Gryffindor



Hufflepuffs



คีย์เวิร์ด protected



คุณลักษณะ หรือ **เมธอด** ของ superclass ที่มี modifier เป็นแบบ private จะทำให้ subclass ไม่สามารถที่จะเรียกใช้ได้ ดังนั้น ภาษาจาวากำหนดให้มี access modifier ที่ชื่อ protected ซึ่งจะทำให้ subclass สามารถเรียกใช้เมธอดหรือคุณลักษณะของ superclass ได้

ตัวอย่างเช่น

```
protected String name;
```

สรุป Access Modifier



ภาษาจาวากำหนดให้มี access modifier ที่ชื่อ protected ซึ่งจะทำให้คลาสที่เป็น subclass สามารถที่จะเรียกใช้เมธอดหรือคุณลักษณะของ superclass ได้ แม้ว่าจะอยู่ต่างคลาสหรืออยู่ต่างแพคเกจกันก็ตาม

modifier	คลาสเดียวกัน	คลาสที่อยู่ใน แพคเกจเดียวกัน	คลาสที่เป็น subclass	คลาสใด ๆ
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

วิธีการแก้ไขแบบที่ 2



```
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);  
    }  
}
```

```
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

ภาษาจาวาได้กำหนดให้คลาสใดๆ สามารถจะสืบทอดคลาสอื่นได้เพียงคลาสเดียวเท่านั้น ดังนั้น คลาสทุกคลาสในภาษาจาวา ถ้าไม่ได้สืบทอดจากคลาสใดเลยจะถือว่าสืบทอดจากคลาสที่ชื่อ **Object**

ตัวอย่างเช่น

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

คลาสที่ชื่อ Object จะมีเมธอดที่สำคัญคือ

①

`public String toString()`

และ

`public boolean equals(Object o)`

②

คีย์เวิร์ด `super` *(optional)*

`super` เป็นคีย์เวิร์ดที่ใช้ในการอ้างอิงถึง `superclass` เพื่อเรียกใช้ เมธอดของ superclass เท่านั้น โดยมีรูปแบบคำสั่งดังนี้

ใช้อ้างอิงถึง method ใน class แม่เท่านั้น

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


คีย์เวิร์ด super

```
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);  
    }  
}
```

```
ID: null  
Name: null  
GPA: 0.0  
Title: null  
Supervisor: null  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
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 ();  
    }  
}
```

คีย์เวิร์ด **super**

ตัวอย่างเช่น คลาส **GradStudent** อาจมีเมธอดที่ชื่อ **showDetailAll()** โดยมีคำสั่งที่เรียกใช้เมธอด **showDetails()** ของคลาส **Student** และ **showThesis()** ของคลาส **GradStudent**

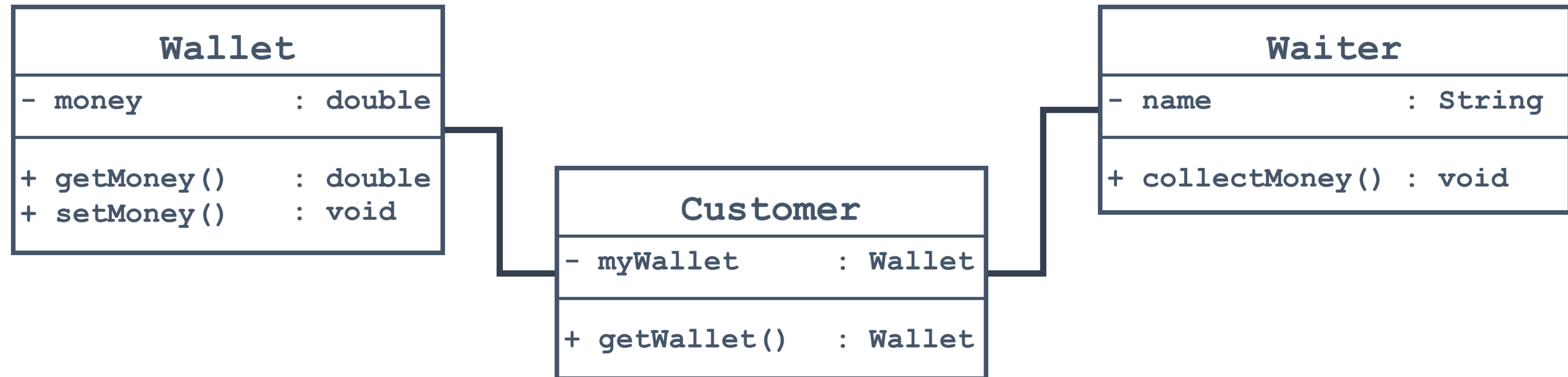
```
public class Student {  
    private String id;  
    private String name;  
    private double gpa;  
    public void showDetails() {  
        System.out.println("ID: "+id);  
        System.out.println("Name: "+name);  
        System.out.println("GPA: "+gpa);  
    }  
}
```

```
public class GradStudent extends Student {  
    private String thesisTitle;  
    private String supervisor;  
  
    public void showDetailAll() {  
        super.showDetails(); // หรือ showDetails()  
  
        System.out.println("Title: "+ thesisTitle);  
        System.out.println("Supervisor: "+supervisor);  
    }  
}
```

Design Pattern: “*the Law of Demeter*”



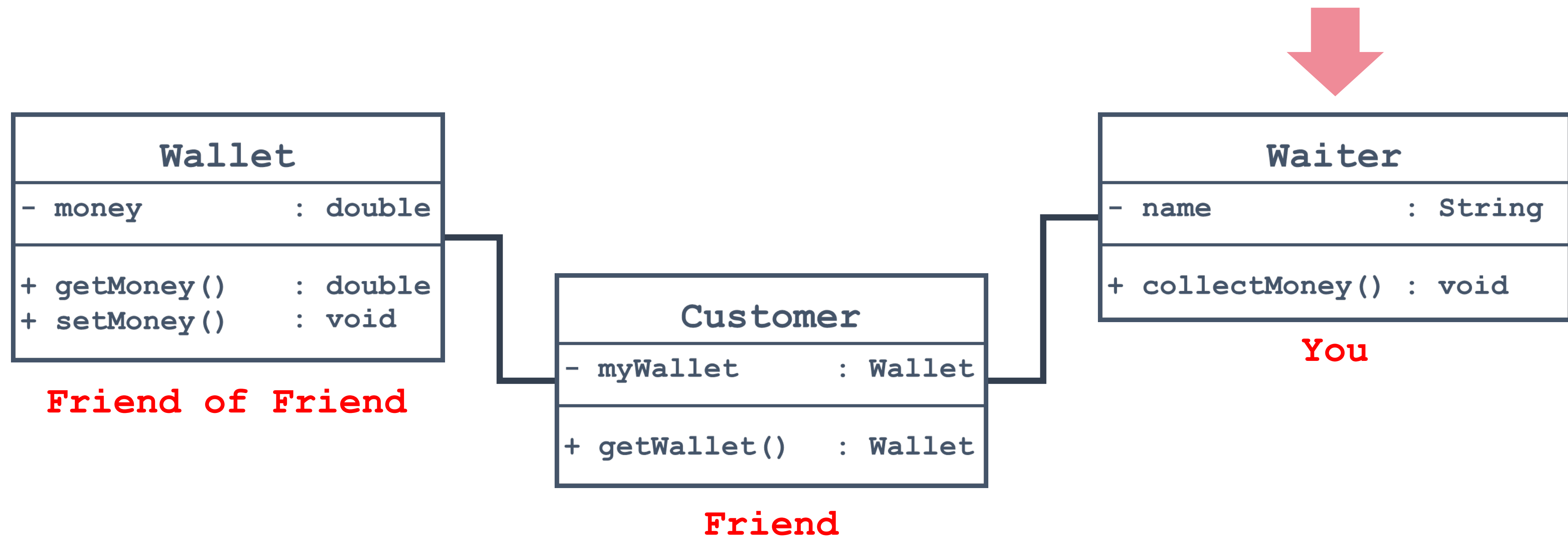
the Law of Demeter คือ Design Pattern รูปแบบหนึ่งซึ่งช่วยให้การออกแบบเชิงวัตถุไม่ผูกติดซึ่งกันและกันมากเกินไป อีกทั้งยังสอดคล้องกับ Information Hiding ที่จะซ่อนข้อมูลจากวัตถุอื่น ๆ ที่ไม่ควรทราบหรือเกี่ยวข้องกัน (แค่เพื่อนเท่านั้น)



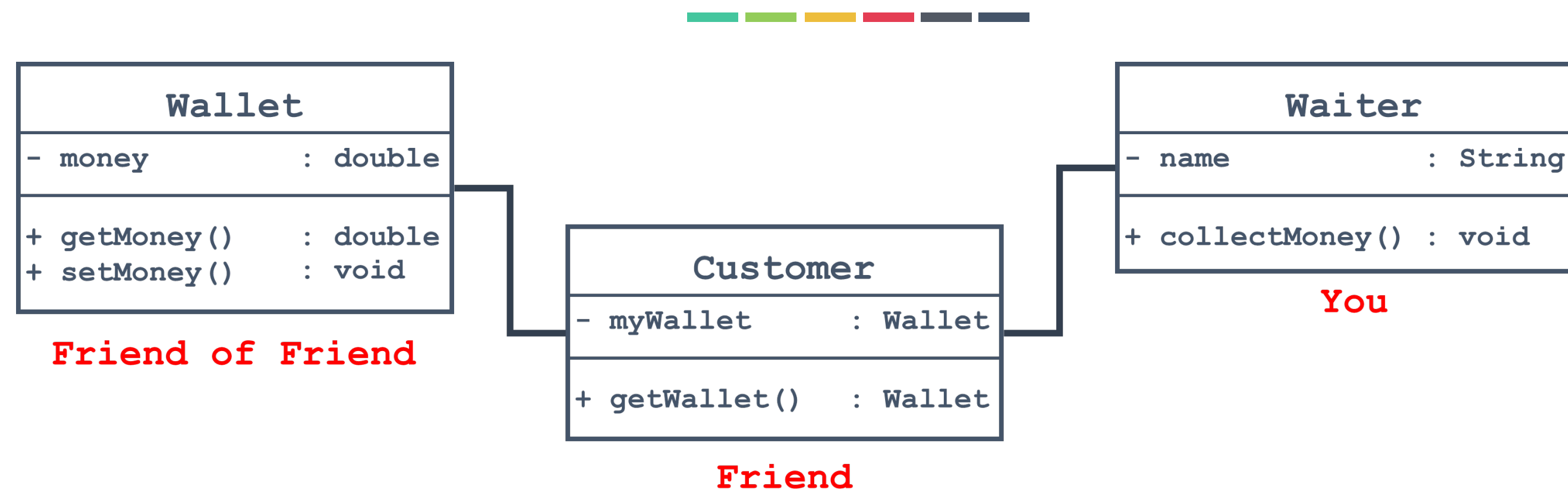
Design Pattern: “*the Law of Demeter*”



the Law of Demeter คือ Design Pattern รูปแบบหนึ่งซึ่งช่วยให้การออกแบบเชิงวัตถุไม่ผูกติดซึ่งกันและกันมากเกินไป อีกทั้งยังสอดคล้องกับ Information Hiding ที่จะซ่อนข้อมูลจากวัตถุอื่น ๆ ที่ไม่ควรทราบหรือเกี่ยวข้องกัน (แค่เพื่อนเท่านั้น)



Design Pattern: “*the Law of Demeter*”



“แต่ละ Object ควรเข้าถึงเพียง Object ที่มันรู้จักเท่านั้น (เพื่อนของมันเท่านั้น) ซึ่งเพื่อนของเพื่อนต้องไม่สามารถเข้าถึงได้” ได้แก่


1. การอ้างอิง**ตัวเอง** เช่น การเรียกใช้ this เป็นต้น
2. การอ้างอิง**ภายในตัวมันเอง** เช่น การเรียกแอตทริบิวต์ การเรียกใช้เมธอด เป็นต้น
3. การ**เรียกใช้ object ที่สัมพันธ์กัน** (โดยตรง) เช่น การใช้พารามิเตอร์ที่ส่งเข้ามาในเมธอด เป็นต้น


```
public class Wallet {
    private double money;
    public double getMoney(){ return this.money; }
    public void setMoney( int m ){ this.money = m; }
}

public class Customer{
    private Wallet myWallet;
    public Wallet getWallet(){ return this.myWallet; }
    public void setWallet(Wallet w){ this.myWallet = w; }
}

public class Waiter{
    private String name;
    private double money;
    public void collectMoney(Customer c, double money){
        if (c.getWallet().getMoney() >= money){
            customer c.getWallet().setMoney(c.getWallet().getMoney() - money);
            this.money += money;
        }else{
            new Exception("Not Enough!!");
        }
    }
}

public class Restaurant{
    public static void main(String args[]) {
        Wallet w = new Wallet();
        w.setMoney(1000);
        Customer c = new Customer();
        c.setWallet(w);
        Waiter wa = new Waiter();
        wa.collectMoney(c, 840);
    }
}
```




```

public class Wallet {
    private double money;
    public double getMoney(){ return this.money; }
    public void setMoney( int m ){ this.money = m; }
}

public class Customer{
    private Wallet myWallet;
    public Wallet getWallet(){ return this.myWallet; }
    public void setWallet(Wallet w){ this.myWallet = w; }
}

public class Waiter{
    private String name;
    private double money;
    public void collectMoney(Customer c, double money){
        if (c.getWallet().getMoney() >= money){
            c.getWallet().setMoney(c.getWallet().getMoney() - money);
            this.money += money;
        }else{
            new Exception("Not Enough!!");
        }
    }
}

```

ภายในตัวมันเอง และ วัตถุที่สัมพันธ์กัน ✓

ภายในตัวมันเอง และ วัตถุที่สัมพันธ์กัน ✓

วัตถุที่สัมพันธ์กัน ✓

เพื่อนของเพื่อน ✗

ไม่ควรเนื่องจาก

- (1) ปัญหาการผูกติดกันของวัตถุมากเกินไป >> ทำให้โค้ดซับซ้อนเกินไป
- (2) ผิดหลักการ Information hiding เนื่องจาก Waiter รู้ข้อมูล Wallet ของ Customer >> รู้มากเกินไป

```
public class Wallet {
    private double money;
    public double getMoney(){ return this.money; }
    public void setMoney( int m ){ this.money = m; }
}

public class Customer{
    private Wallet myWallet;
    public Wallet getWallet(){ return this.myWallet; }
    public void setWallet(Wallet w){ this.myWallet = w; }
}

public class Waiter{
    private String name;
    private double money;
    public void collectMoney(Customer c, double money){
        if (c.getWallet().getMoney() >= money){
            c.getWallet().setMoney(c.getWallet().getMoney() - money);
            this.money += money;
        }else{
            new Exception("Not Enough!!");
        }
    }
}
```

ภายในตัวมันเอง และ วัตถุที่สัมพันธ์กัน ✓

ภายในตัวมันเอง และ วัตถุที่สัมพันธ์กัน ✓

วัตถุที่สัมพันธ์กัน ✓

เพื่อนของเพื่อน ✗

ไม่ควรเนื่องจาก

- (1) ปัญหาการผูกติดกันของวัตถุมากเกินไป >> ทำให้โค้ดซับซ้อนเกินไป
- (2) ผิดหลักการ Information hiding เนื่องจาก Waiter รู้ข้อมูล Wallet ของ Customer >> รู้มากเกินไป

แก้ไขโดย

ไม่ทำการเรียกใช้หรือเข้าถึงวัตถุที่ไม่ติดกัน หรือไม่เรียกใช้เพื่อนของเพื่อน

```
public class Wallet {
    private double money;
    public double getMoney(){ return this.money; }
    public void setMoney( int m ){ this.money = m; }
    public double withdraw(double m) {
        if (money >= m) {
            money -= m;
        } else { new Exception("Not Enough!!"); }
        return m;
    }
}

public class Customer{
    private Wallet myWallet;
    public Wallet getWallet(){ return this.myWallet; }
    public void setWallet(Wallet w){ this.myWallet = w; }
    public double pay (int m) { return myWallet.withdraw(m); }
}

public class Waiter{
    private String name;
    private double money;
    public void collectMoney(Customer c, double money){
        money += c.pay(money);
    }
}
```