

# บทที่ 7: เมธอด Constructor, คลาสไม่สมบูรณ์ (Abstract class), และอินเตอร์เฟส (Interface)

---

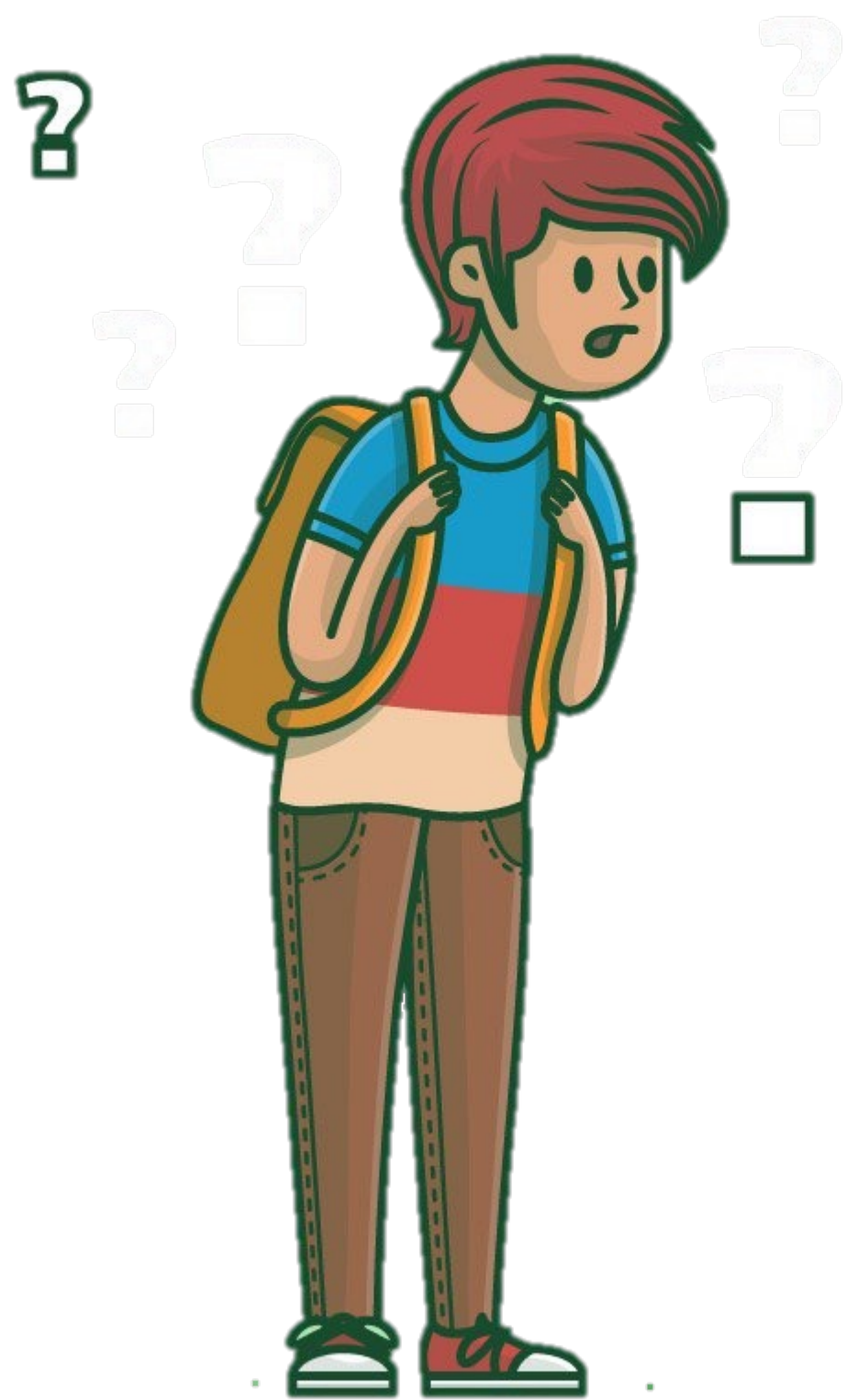
บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

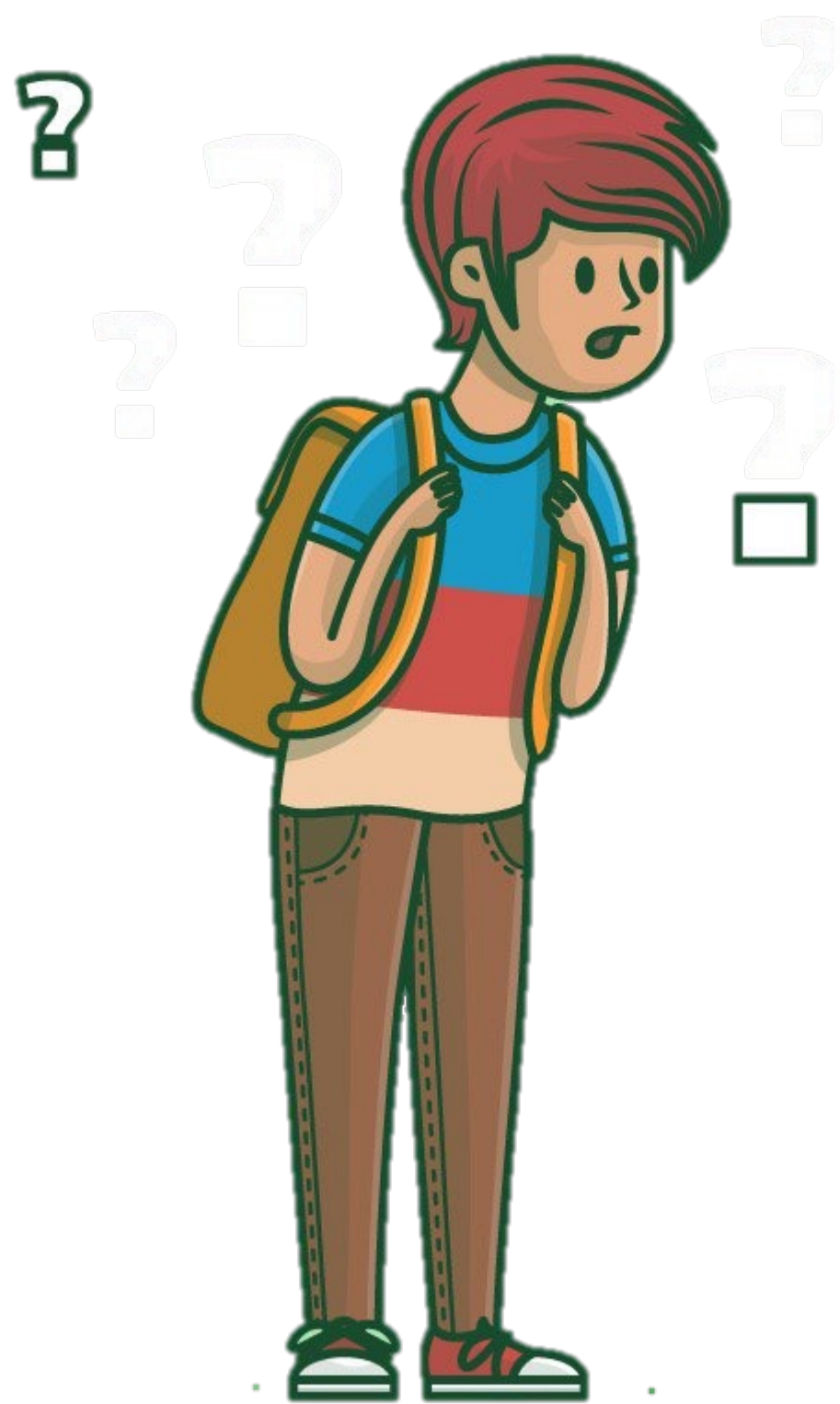
# หัวข้อ

- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)



# หัวข้อ

- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)



# คอนสตรัคเตอร์ (constructor)

คอนสตรัคเตอร์ คือ เมธอดที่มีชื่อเหมือนกับคลาส ซึ่งใช้กำหนดค่าเริ่มต้นให้กับแอตทริบิวต์หรือการทำงานเบื้องต้น เราสามารถสร้าง constructor ในคลาสใดก็ได้ อาทิเช่น  ไม่มีการกำหนด return type

```
public class Student{  
    protected String id, name;  
    protected double gpa;  
    public Student (){} >>> Default constructor  
    public Student (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.gpa = 0.0;  
    } public Student (String id, String name, double gpa) {  
        this.id = id;  
        this.name = name;  
        this.gpa = gpa;  
    }  
}
```

# คอนสตรัคเตอร์ (constructor)



```
public class Student{  
    protected String id, name;  
    protected double gpa;  
    public Student () { → Default Constructor  
        id = "unknown";  
        name = "unknown";  
        gpa = 0.0;  
    } public Student (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.gpa = 0.0;  
    } public Student (String id, String name, double gpa) {  
        this.id = id;  
        this.name = name;  
        this.gpa = gpa;  
    }  
}
```



# คอนสตรัคเตอร์ (constructor)

คอนสตรัคเตอร์จะถูกเรียกใช้เมื่อเราสร้างวัตถุด้วยคำสั่ง

**new**

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

```
    public Player () {} → Default constructor
```

```
    public Player (int h){
        hp = h;
    }
```

```
    public Player (int h, int a){
        hp = h;
        atk = a;
```

```
    }
    public int getHp(){
        return hp;
    }
```

```
}
```

```
public class Main{
    public static void main( String[] args) {
        Player p1 = new Player(); ← constructor
        Player p2 = new Player(10);
        Player p3 = new Player(10,100);
```

```
        System.out.println("p1 "+ p1.getHp() );
        System.out.println("p2 "+ p2.getHp() );
        System.out.println("p3 "+ p3.getHp() );
```

```
}
```

```
}
```

ผลลัพธ์

```
p1 0
p2 10
p3 10
```

# คอนสตรัคเตอร์ (constructor)

**Default constructor** จะถูกสร้างโดยอัตโนมัติในกรณีที่ คลาสนั้นไม่มี constructor นั้นหมายความว่า ถ้ามีการสร้าง constructor มาแล้ว Default constructor จะไม่ถูกสร้าง อาทิเช่น

```
public class Player{  
    private int hp;  
    private int atk;  
    private int level;  
    private int money;  
    public Player(){  
    public int getHp(){  
        return hp;  
    }  
}
```

```
public class Main{  
    public static void main( String[] args) {  
        Player p1 = new Player();  
        System.out.println("p1 "+ p1.getHp() );  
    }  
}
```

ผลลัพธ์  
p1 0

# คอนสตรัคเตอร์ (constructor)

**Default constructor** จะถูกสร้างโดยอัตโนมัติในกรณีที่ คลาสนั้นไม่มี constructor นั้นหมายความว่า **ถ้ามีการสร้าง constructor มาแล้ว Default constructor จะไม่ถูกสร้าง** อาทิเช่น

```
public class Player{
    private int hp;
    private int atk;
    private int level;
    private int money;
    public Player (int h){
        hp = h;
    }
    public Player (int h, int a){
        hp = h;
        atk = a;
    }
    public int getHp(){
        return hp;
    }
}
```

```
public class Main{
    public static void main( String[] args) {
        Player p1 = new Player();
        Player p2 = new Player(10);
        Player p3 = new Player(10,100);

        System.out.println("p1 "+ p1.getHp() );
        System.out.println("p2 "+ p2.getHp() );
        System.out.println("p3 "+ p3.getHp() );
    }
}
```

ผลลัพธ์

Main.java:3: error: no suitable constructor found for Player(no arguments)

```
    Player p1 = new Player();
                  ^
```

constructor Player.Player(int) is not applicable

constructor Player.Player(int,int) is not applicable



# หลักการทำงานของคอนสตรัคเตอร์กับการสร้างวัตถุ

ประกอบด้วย 4 ขั้นตอนดังต่อไปนี้

```
public class Student{  
    private String id = "0" ;  
    private String name = "unknown";  
    private double gpa = 0.0;  
    public Student (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.gpa = 0.0;  
    }  
}  
  
public class Main{  
    public static void main( String[] args) {  
        Student s2 = new Student("203", "Tara");  
    }  
}
```

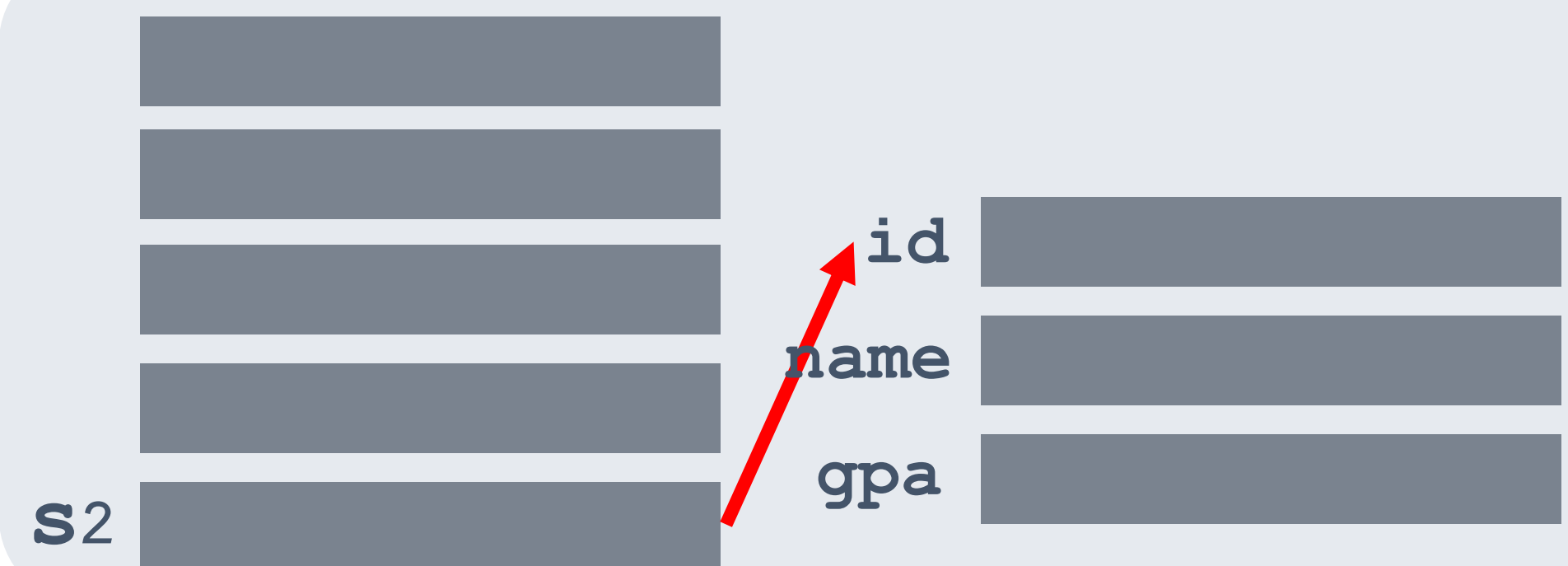
(1) จองพื้นที่หน่วยความจำให้กับวัตถุ

(2) กำหนดค่าเริ่มต้นให้กับแอตทริบิวต์ของวัตถุ

(3) กำหนดค่าแอตทริบิวต์ของวัตถุตามที่ได้ประกาศไว้

(4) เรียกใช้งาน constructor

## หน่วยความจำ



# หลักการทำงานของคอนสตรัคเตอร์กับการสร้างวัตถุ

ประกอบด้วย 4 ขั้นตอนดังต่อไปนี้

```
public class Student{
    private String id = "0" ;
    private String name = "unknown";
    private double gpa = 0.0;
    public Student (String id, String name) {
        this.id = id;
        this.name = name;
        this.gpa = 0.0;
    }
}

public class Main{
    public static void main( String[] args) {
        Student s2 = new Student("203", "Tara");
    }
}
```

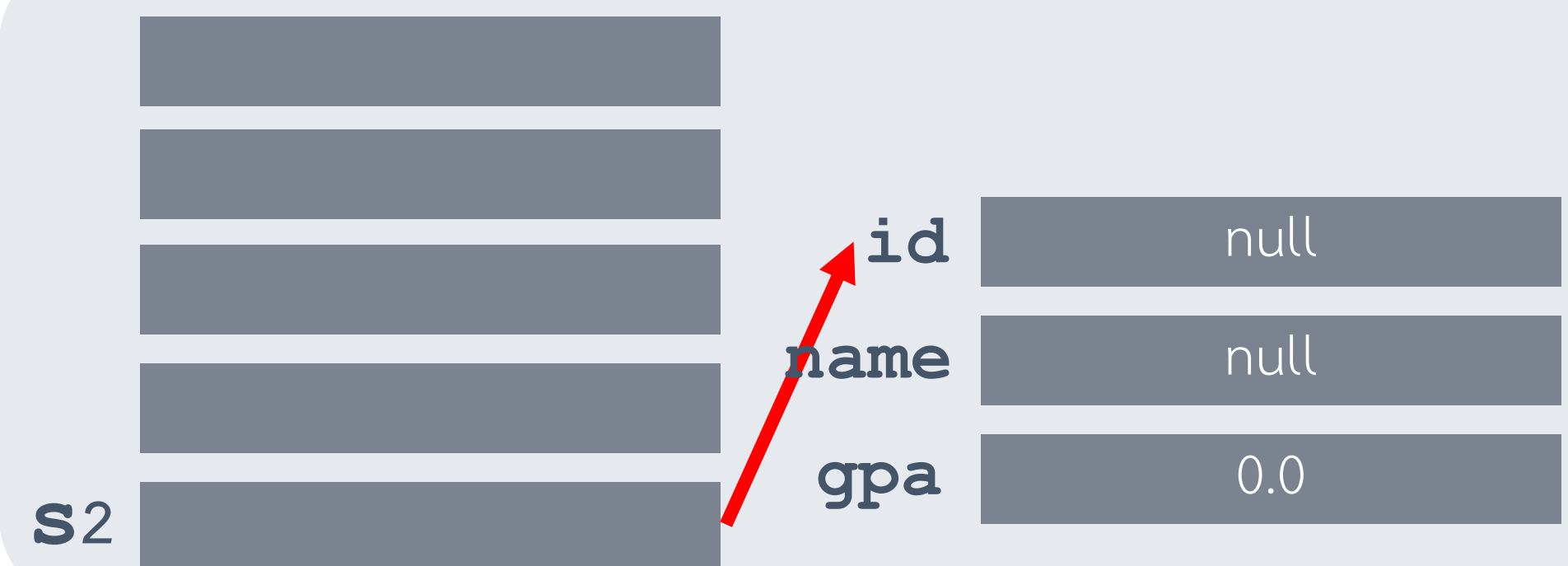
(1) จองพื้นที่หน่วยความจำให้กับวัตถุ

(2) กำหนดค่าเริ่มต้นให้กับแอตทริบิวต์ของวัตถุ

(3) กำหนดค่าแอตทริบิวต์ของวัตถุตามที่ได้ประกาศไว้

(4) เรียกใช้งาน constructor

## หน่วยความจำ



# หลักการทำงานของคอนสตรัคเตอร์กับการสร้างวัตถุ

ประกอบด้วย 4 ขั้นตอนดังต่อไปนี้

```
public class Student{  
    private String id = "0" ;  
    private String name = "unknown";  
    private double gpa = 0.0;  
    public Student (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.gpa = 0.0;  
    }  
}  
  
public class Main{  
    public static void main( String[] args) {  
        Student s2 = new Student("203", "Tara");  
    }  
}
```

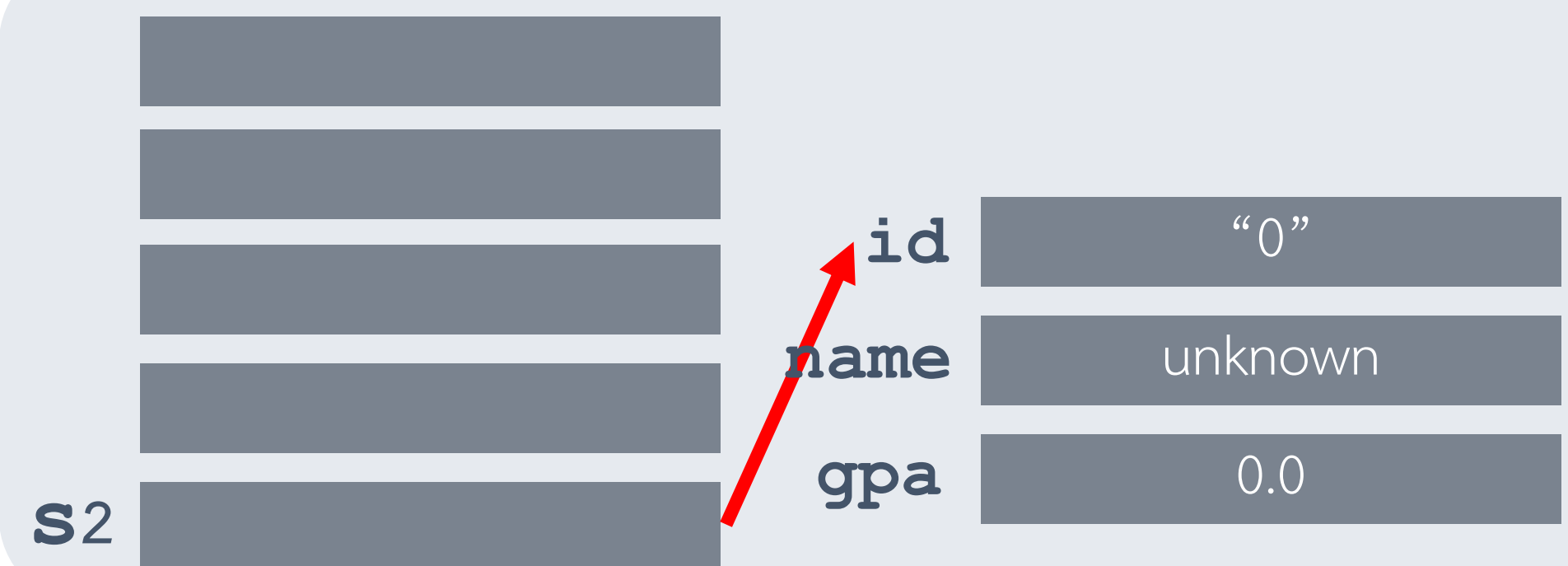
(1) จองพื้นที่หน่วยความจำให้กับวัตถุ

(2) กำหนดค่าเริ่มต้นให้กับแอตทริบิวต์ของวัตถุ

(3) กำหนดค่าแอตทริบิวต์ของวัตถุตามที่ได้ประกาศ

(4) เรียกใช้งาน constructor

## หน่วยความจำ



# หลักการทำงานของคอนสตรัคเตอร์กับการสร้างวัตถุ

ประกอบด้วย 4 ขั้นตอนดังต่อไปนี้

```
public class Student{  
    private String id = "0" ;  
    private String name = "unknown";  
    private double gpa = 0.0;  
    public Student (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.gpa = 0.0;  
    }  
}  
  
public class Main{  
    public static void main( String[] args) {  
        Student s2 = new Student("203", "Tara");  
    }  
}
```

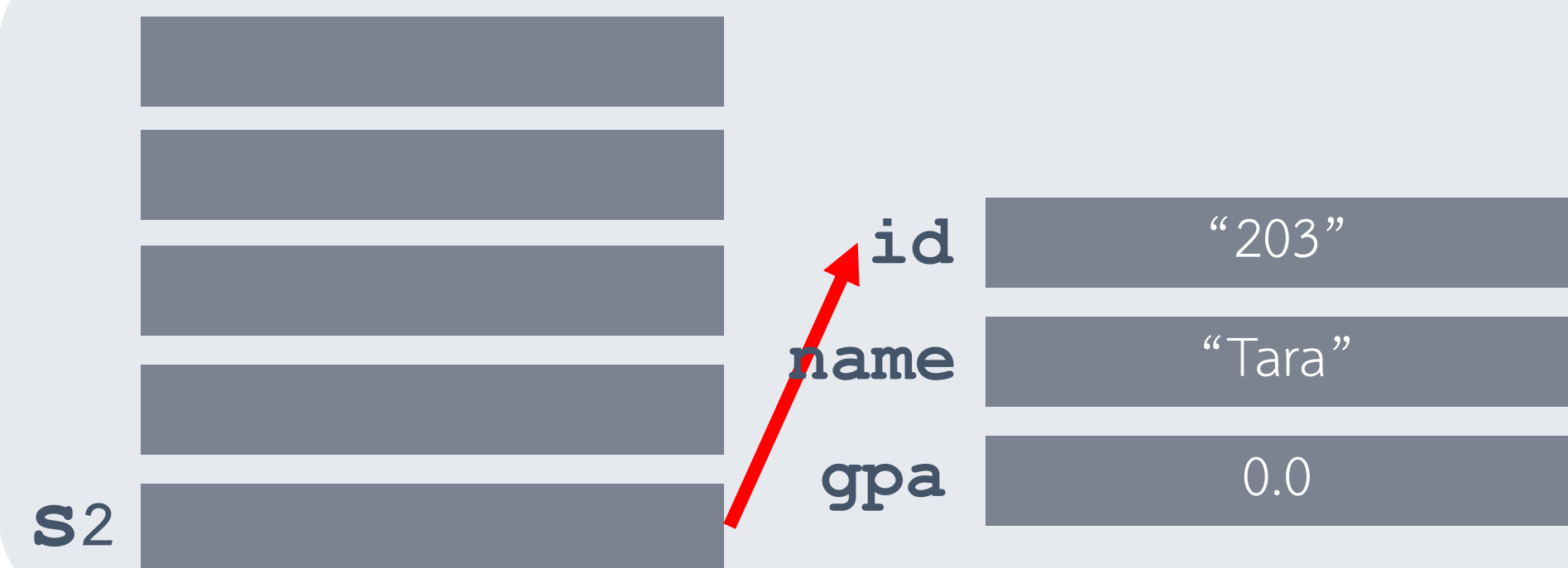
(1) จองพื้นที่หน่วยความจำให้กับวัตถุ

(2) กำหนดค่าเริ่มต้นให้กับแอตทริบิวต์ของวัตถุ

(3) กำหนดค่าแอตทริบิวต์ของวัตถุตามที่ได้ประกาศไว้

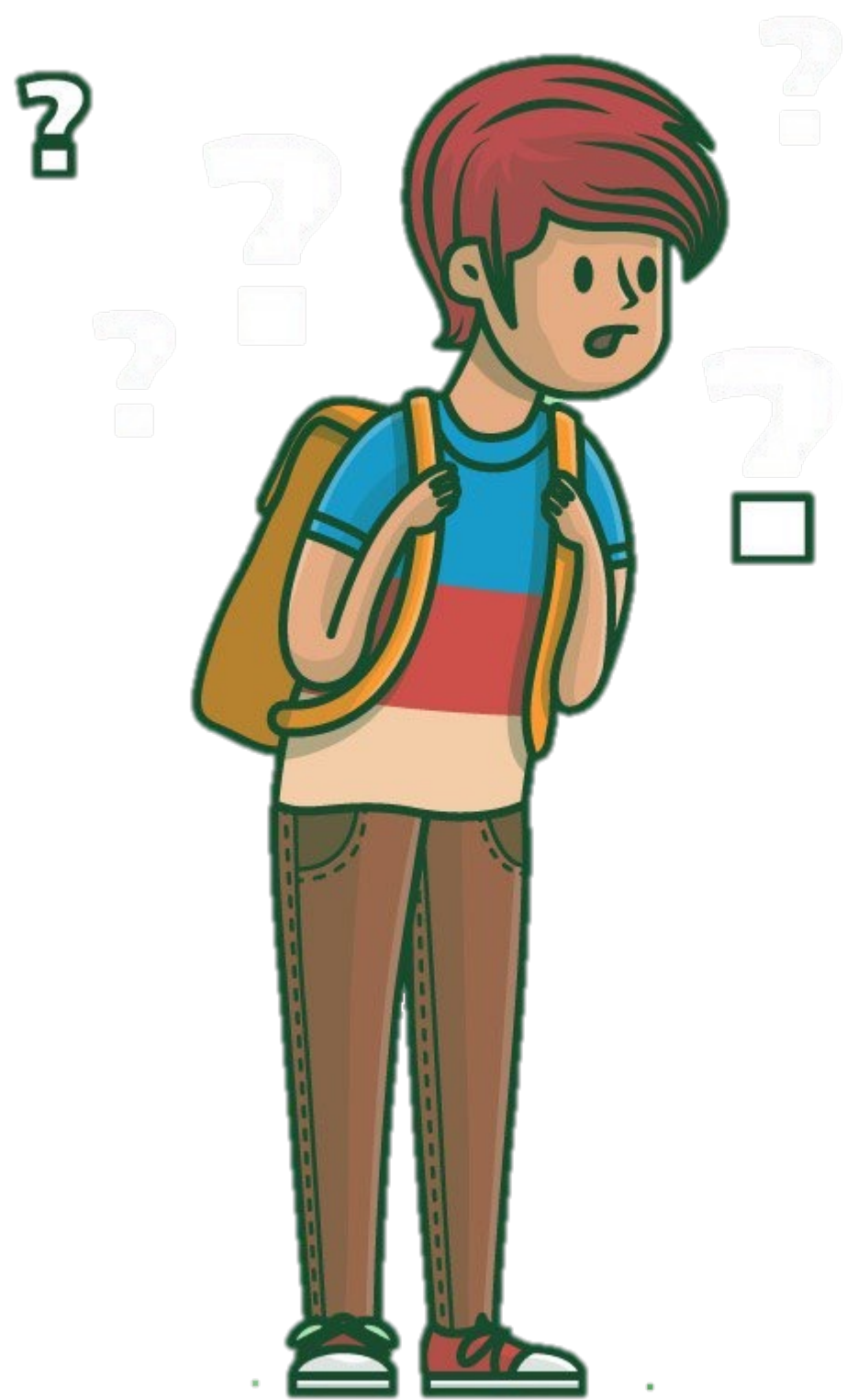
(4) เรียกใช้งาน constructor

## หน่วยความจำ



# หัวข้อ

- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)





# การใช้งานคีย์เวิร์ด this และ super



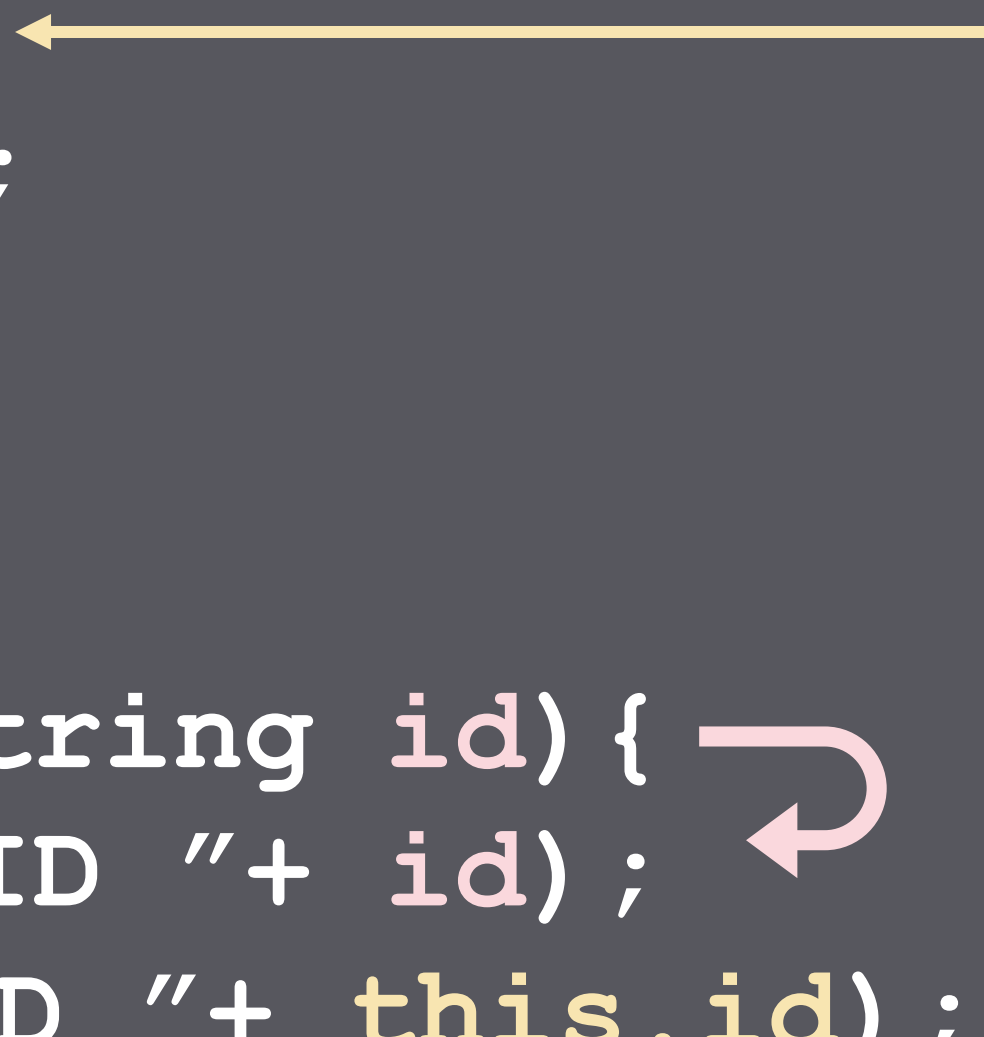
โดยปกติแล้วนักศึกษาสามารถใช้คีย์เวิร์ด this และ super เพื่อใช้ระบุว่าเป็นของคลาสแม่หรือคลาสลูกสำหรับกรณีที่มีความคลุมเครือ นอกจากนี้ ในภาษาจาวายังมีวิธีการเรียกคอนสตรัคเตอร์ระหว่างกันอยู่ โดยอาศัยเมธอด this(...) และ super(...) ซึ่งสามารถสรุปความแตกต่างได้ดังนี้

	ประเภท	บ่งบอกถึง	ความหมาย
this	คีย์เวิร์ด	คลาสตนเอง	ใช้เพื่อเรียกใช้งานแอททริบิวต์หรือเมธอด <u>ภายในคลาส</u> ซึ่งนิยมเรียกใช้งานเมื่อเกิดความคลุมเครือเท่านั้น และควรใช้กับแอททริบิวต์
super	คีย์เวิร์ด	คลาสแม่	ใช้เพื่อเรียกใช้งานแอททริบิวต์และเมธอดของ <u>คลาสแม่</u>
this(...)	เมธอด	คลาสตนเอง	เมธอดที่เรียกใช้งานเพื่อเรียกคอนสตรัคเตอร์ <u>ภายในคลาส</u>
super(...)	เมธอด	คลาสแม่	เมธอดที่เรียกใช้งานเพื่อเรียกคอนสตรัคเตอร์ของ <u>คลาสแม่</u>

# ตัวอย่างการใช้งานคีย์เวิร์ด **this**

นักศึกษาจะพบว่าเกิดความคลุมเครือของตัวแปร `id` และแอททริบิวต์ `id` ขึ้นในเมธอด `printID(..)`

```
public class CreditCard{  
    protected String id;  
    protected String name;  
    protected double gpa;  
    CreditCard  
    public Student () {}  
    public void printID(String id) {  
        System.out.print("ID "+ id);  
        System.out.print("ID "+ this.id);  
    }  
}
```



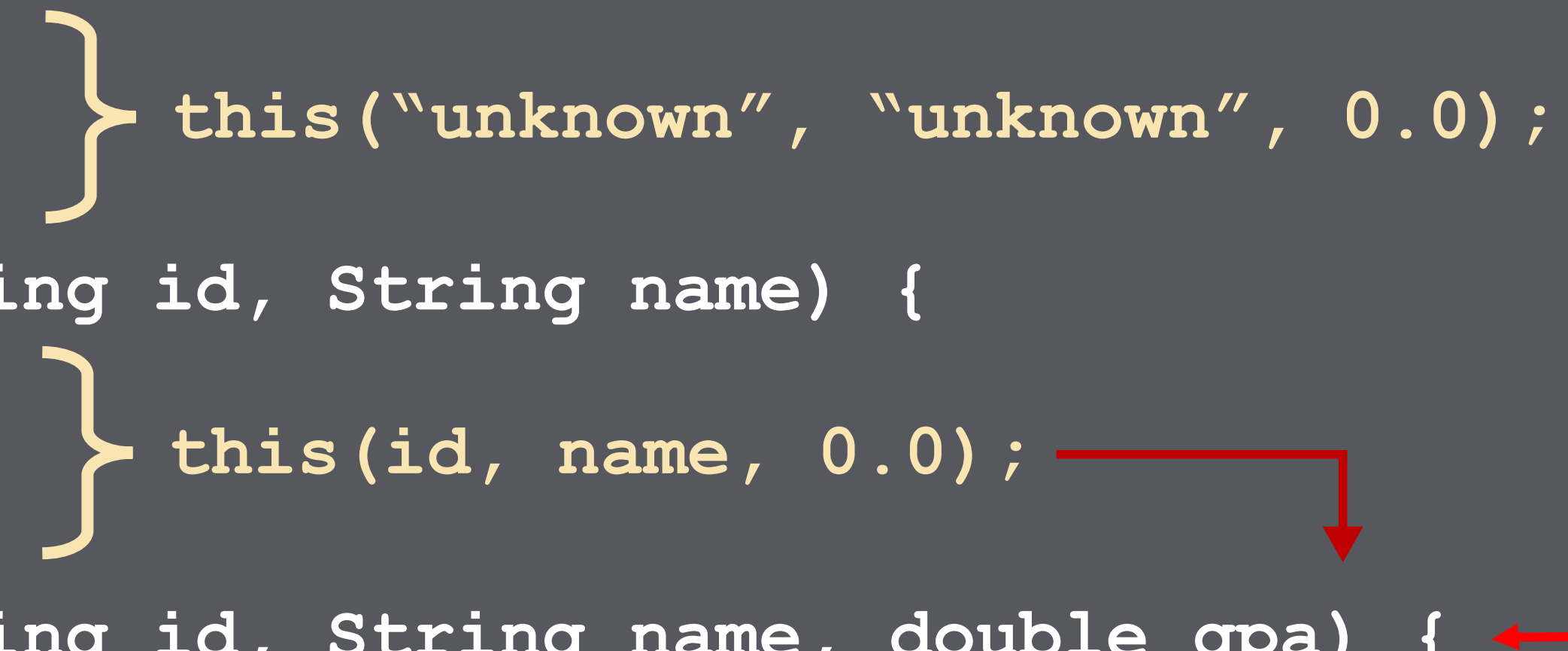
# ตัวอย่างการใช้งาน **คีย์เวิร์ด super**

```
public class Student{
    protected String id, name;
    protected double gpa;
    public void printDetail(){
        System.out.println("ID "+id);
        System.out.println("Name "+name);
    }
}

public class GradStudent extends Student{
    protected String thesisTitle;
    public void printDetail(){
        super.printDetail();
        System.out.println("GPA "+ gpa);
        System.out.println("Title "+ thesisTitle);
    }
}
```

# ตัวอย่างการใช้งานเมธอด **this()**

```
public class Student{  
    protected String id, name;  
    protected double gpa;  
    public Student () {  
        id = "unknown";  
        name = "unknown";  
        gpa = 0.0;  
    }  
    public Student (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.gpa = 0.0;  
    }  
    public Student (String id, String name, double gpa) {  
        this.id = id;  
        this.name = name;  
        this.gpa = gpa;  
    }  
}
```



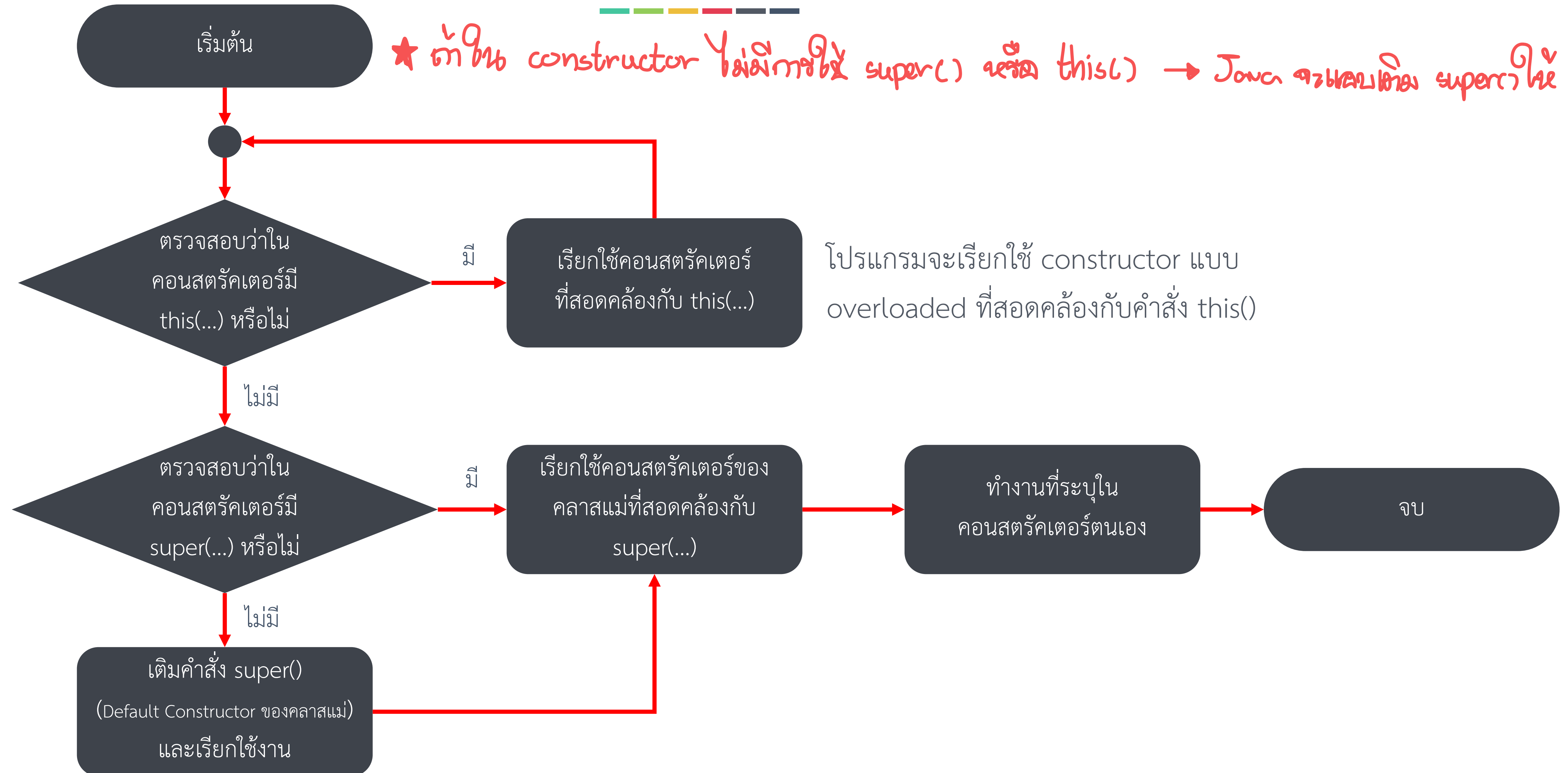
# ตัวอย่างการใช้งานเมธอด `super()`

```
public class Animal{  
    protected double weight;  
    public Animal (double weight) {  
        this.weight = weight;  
    }  
}
```

```
public class Pig extends Animal{  
    protected double height;  
    public Pig () {  
        super(0);  
        this.height = 0; } หรือ this(0,0);  
    } public Pig (double height, double weight) {  
        super(weight);  
        this.height = height;  
    }  
}
```



# ขั้นตอนการทำงานของคอนสตรัคเตอร์



# ตัวอย่างการทำงานของคอนสตรัคเตอร์



```
public class Animal {  
    public Animal(){  
        System.out.println("Hi, Animal");  
    }  
}  
  
public class Ant extends Animal {  
    public Ant(){  
        System.out.println("Hi, Ant");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Ant a1 = new Ant();  
    }  
}
```

ผลลัพธ์

Hi, Animal

Hi, Ant

# ตัวอย่างการทำงานของคอนสตรัคเตอร์ (ต่อ)



```
public class Animal {  
    public Animal() {  
        System.out.println("Hi, Animal");  
    }  
}  
  
public class Ant extends Animal {  
    public Ant() {  
        this("Ant");  
    }  
    public Ant(String n) {  
        System.out.println("Hi, " + n);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Ant a1 = new Ant("Sara");  
    }  
}
```


ผลลัพธ์

Hi, Animal

Hi, Sara

# ตัวอย่างที่ 1

```
public class Main {  
    public static void main(String[] args) {  
        Duck d2 = new Duck();  
    }  
}  
  
public class Animal{  
    protected String name;  
    public Animal(String n){  
        this.name = n;  
    }  
}  
  
public class Duck extends Animal{  
    public Duck(String n){  
        super(n);  
    }  
    public Duck(){}  
}
```

  
super()

ผลลัพธ์

Main.java:14: error: constructor Animal in class  
Animal cannot be applied to given types;

```
    public Duck(){}  
                ^
```

required: String

found: no arguments

# ตัวอย่างที่ 2

```
public class Main {  
    public static void main(String[] args) {  
        Duck d1 = new Duck("Pedd");  
        Duck d2 = new Duck();  
        System.out.print("Done it.");  
    }  
}  
  
public class Animal{  
    protected String name;  
    public Animal(String n){ this.name = n; }  
}  
  
public class Duck extends Animal{  
    public Duck(String n){  
        super(n);  
    }  
    public Duck(){  
        super("");  
    }  
}
```

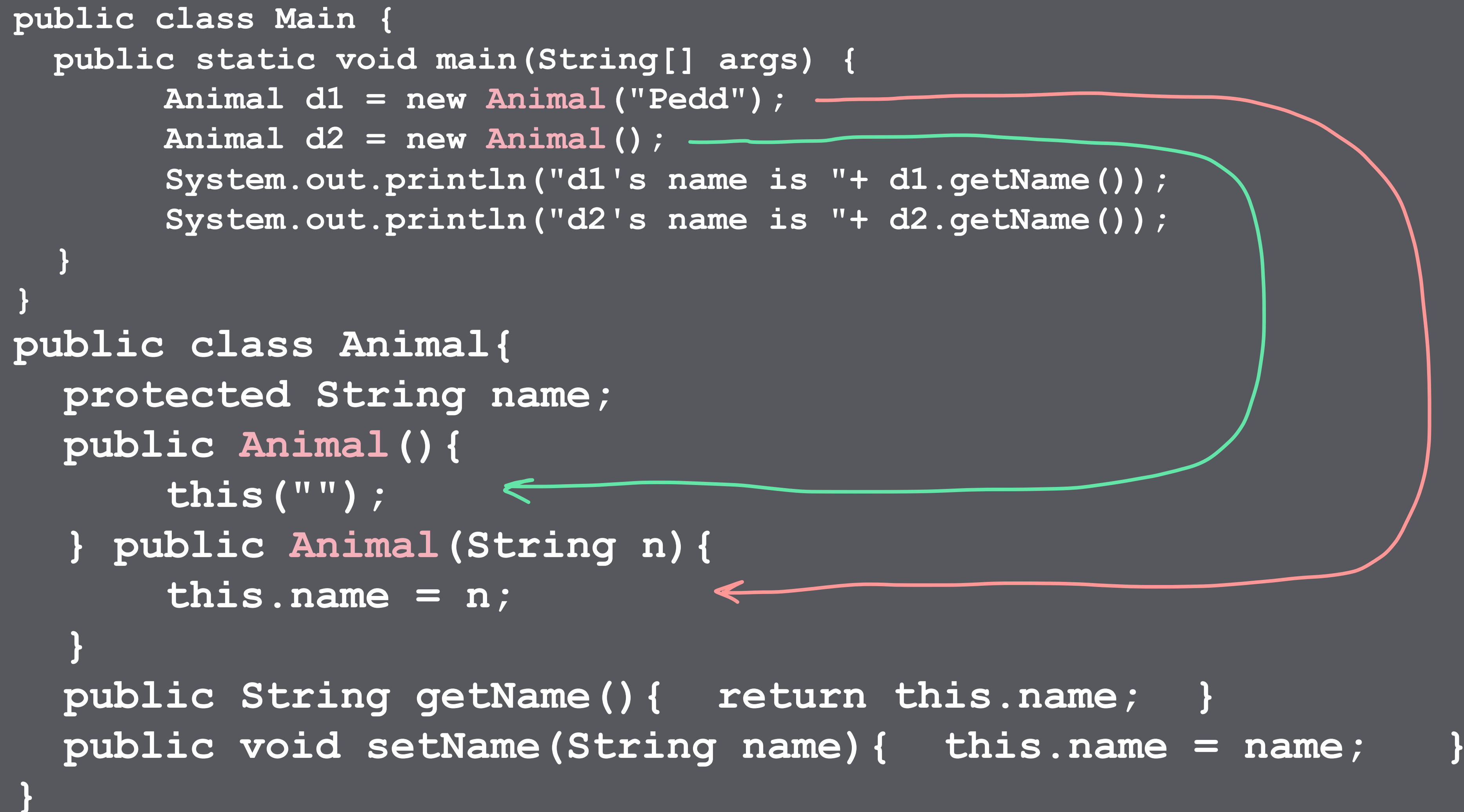
ผลลัพธ์

Done it.



# ตัวอย่างที่ 3

```
public class Main {  
    public static void main(String[] args) {  
        Animal d1 = new Animal("Pedd");  
        Animal d2 = new Animal();  
        System.out.println("d1's name is " + d1.getName());  
        System.out.println("d2's name is " + d2.getName());  
    }  
}  
  
public class Animal{  
    protected String name;  
    public Animal(){  
        this("");  
    } public Animal(String n){  
        this.name = n;  
    }  
    public String getName(){ return this.name; }  
    public void setName(String name){ this.name = name; }  
}
```



ผลลัพธ์

d1's name is Pedd  
d2's name is

# ตัวอย่างที่ 4



```
public class Main {
    public static void main(String[] args) {
        Duck d1 = new Duck("Pedd");
        Duck d2 = new Duck();
        System.out.println("d1's name is " + d1.getName());
        System.out.println("d2's name is " + d2.getName());
    }
}

public class Animal{
    protected String name;
    public Animal(){ this(""); }
    public Animal(String n){ this.name = n; }
    public String getName(){ return this.name; }
}

public class Duck extends Animal{
    public Duck(String n){
        super(n);
    }
    public Duck(){
        super();
    }
}
```

ผลลัพธ์

```
d1's name is Pedd
d2's name is
```

# คีย์เวิร์ด final

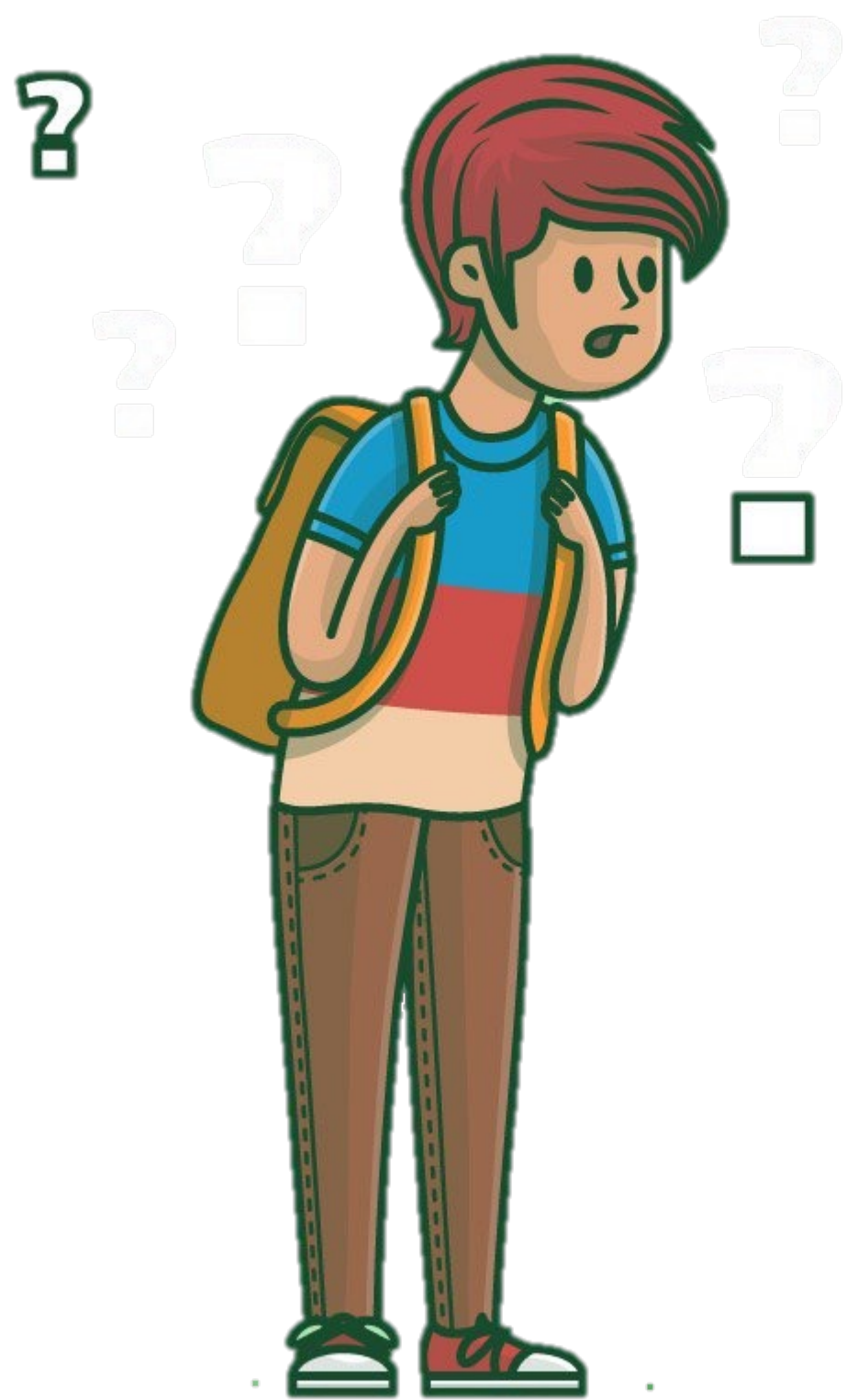


- คีย์เวิร์ด **final** สามารถใช้ได้กับคลาส ตัวแปร และเมธอด

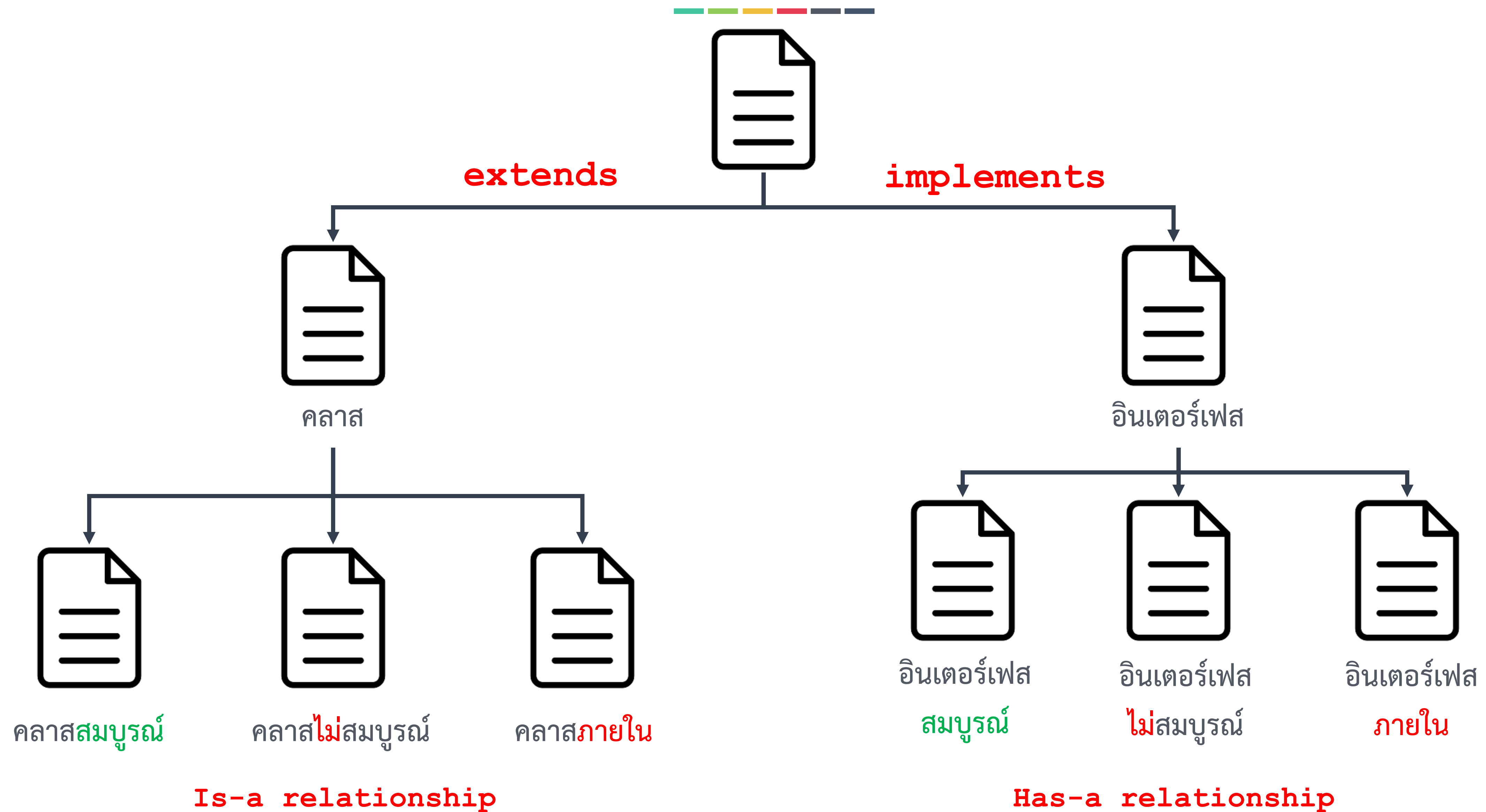
ตำแหน่ง	ความหมาย
คลาส	ทำให้คลาสอื่นไม่สามารถสืบทอดคลาสนี้ได้
เมธอด	เมธอดที่จะไม่สามารถมีเมธอดแบบ overridden ได้
ตัวแปร	ค่าคงที่ ซึ่งจะทำได้กำหนดค่าได้เพียงครั้งเดียวเท่านั้น

# หัวข้อ

- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)

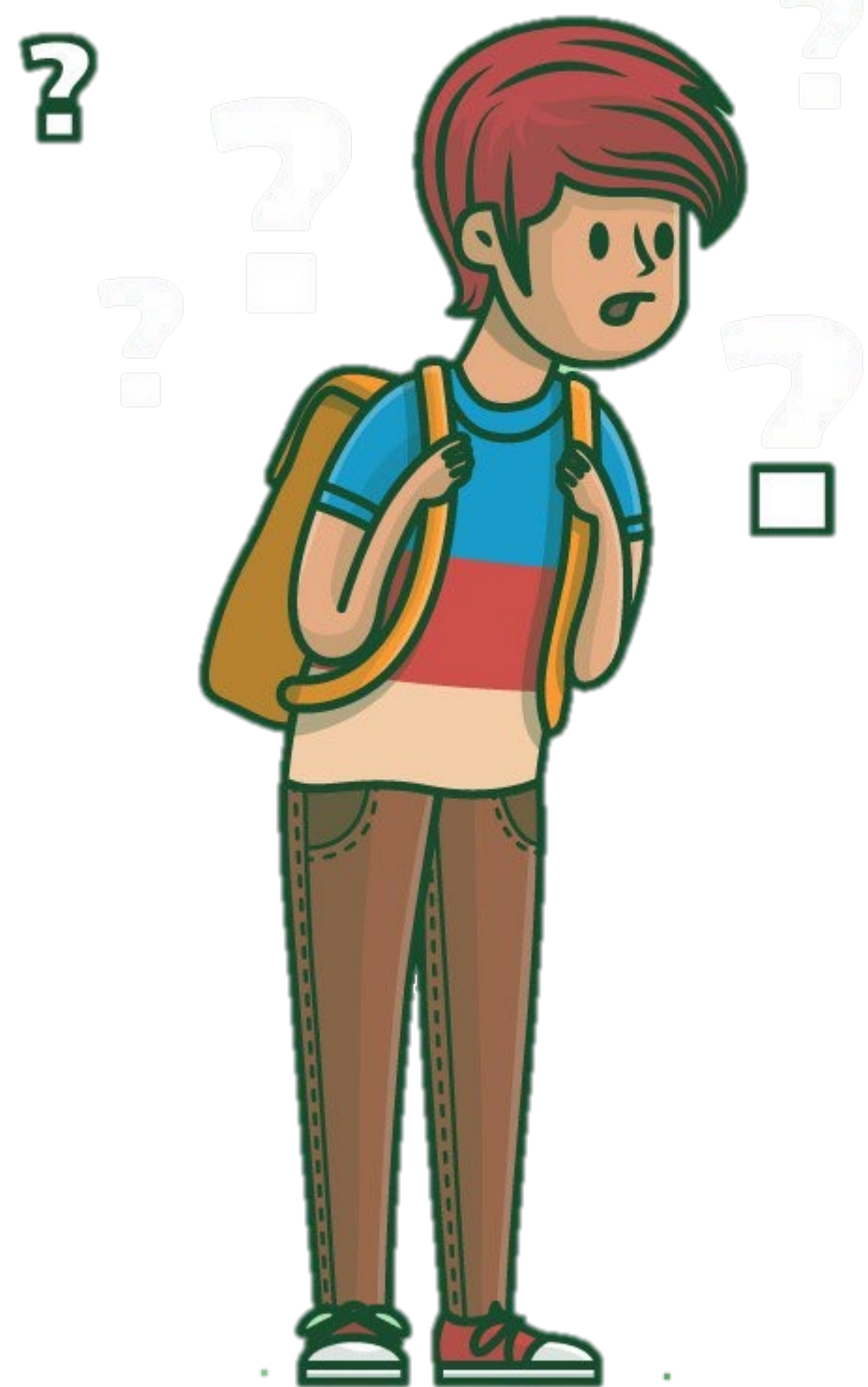


# การพิจารณาการสืบทอดโดยอ้างอิงจากความสัมพันธ์



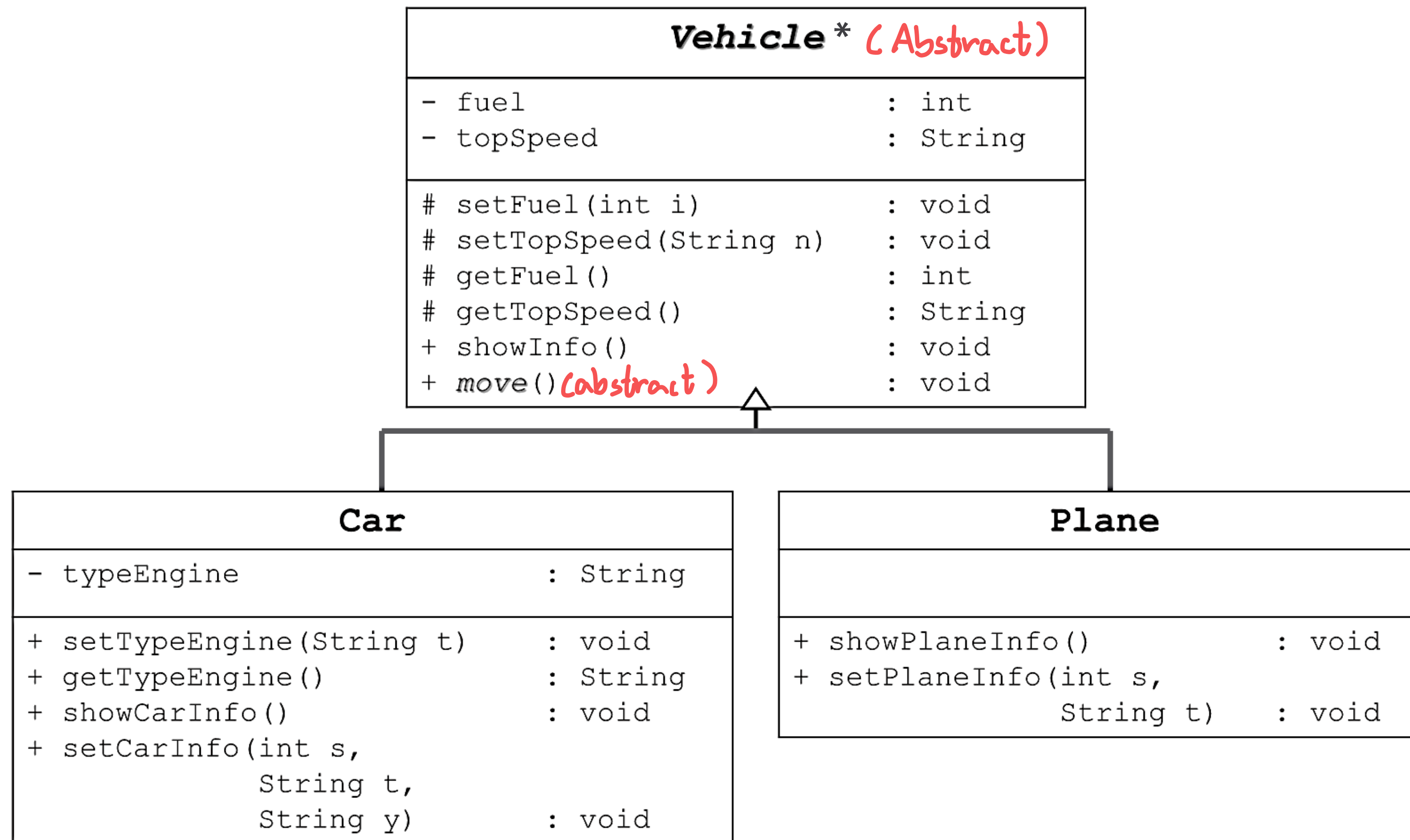


# หัวข้อ



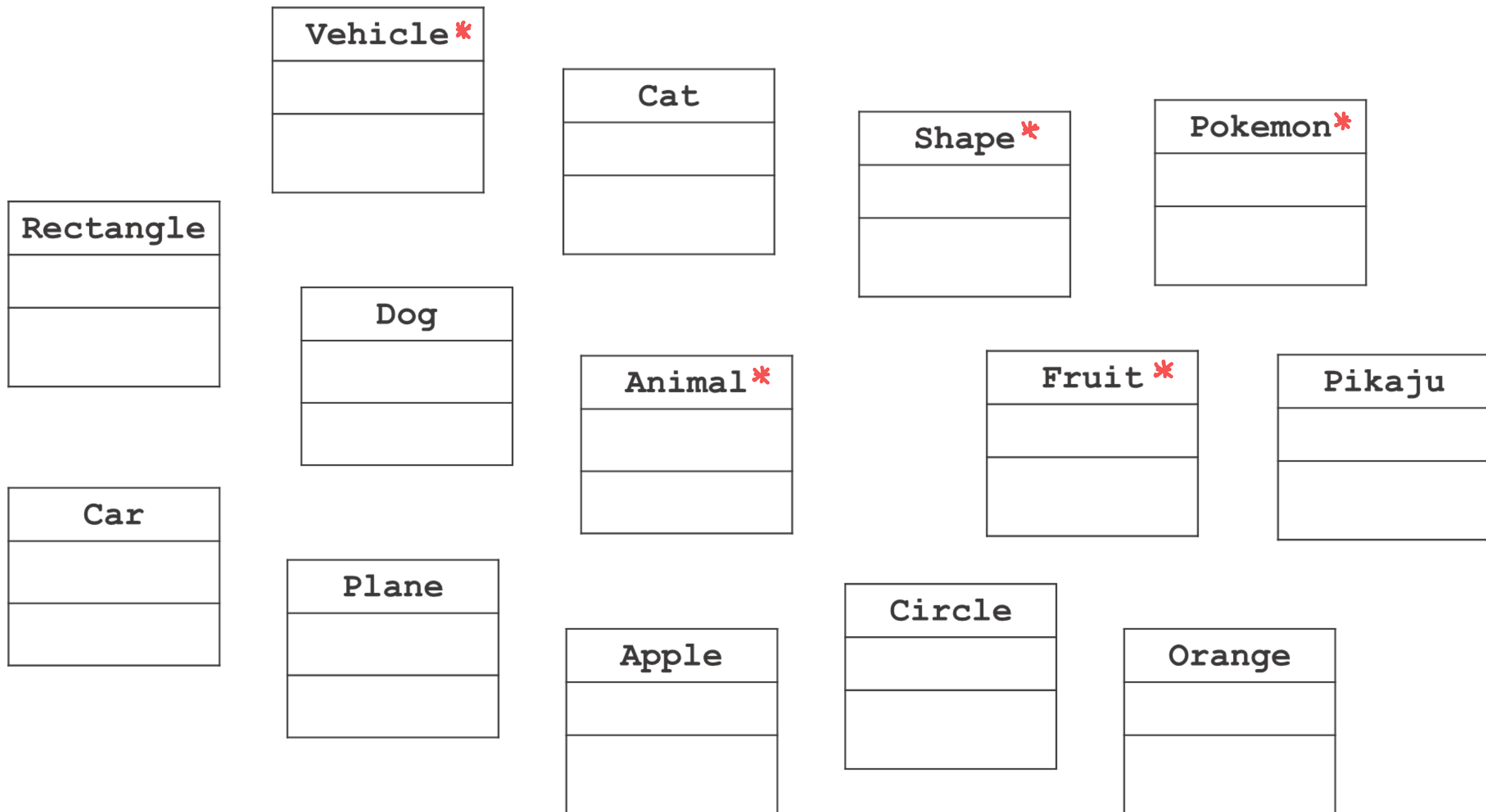
- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คุณลักษณะและเมธอดของคลาสและของอ็อบเจกต์
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)

# คลาสแบบ abstract



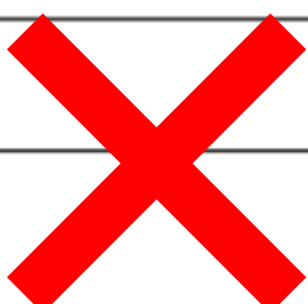
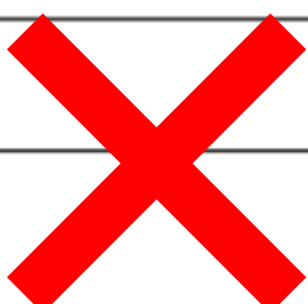





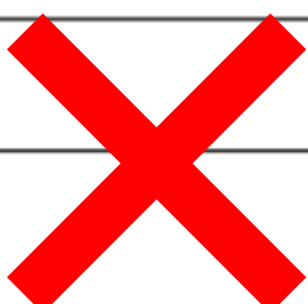


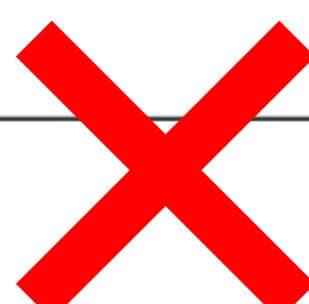
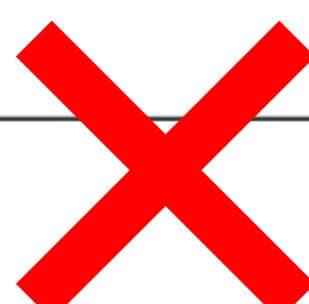






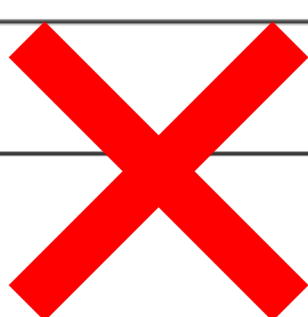
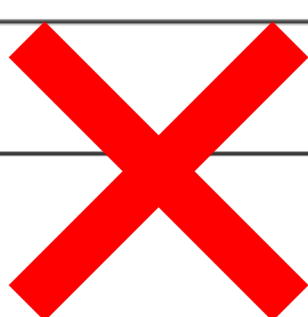
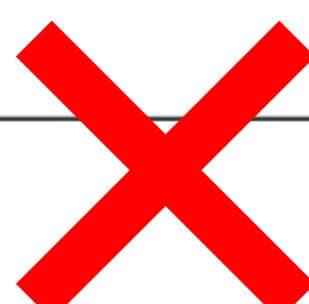



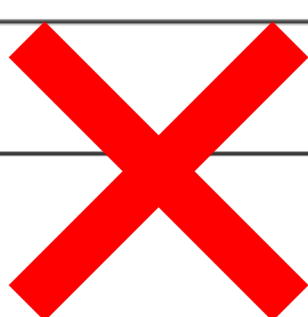
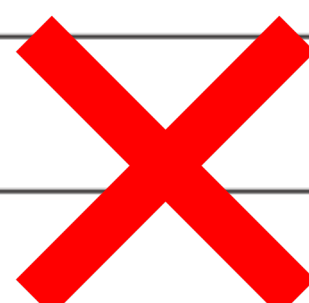
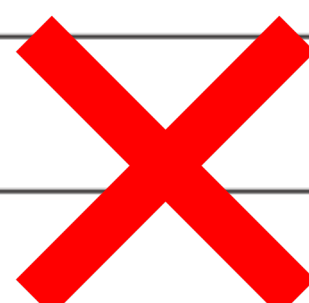
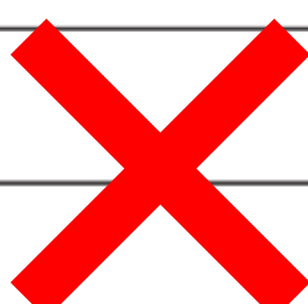
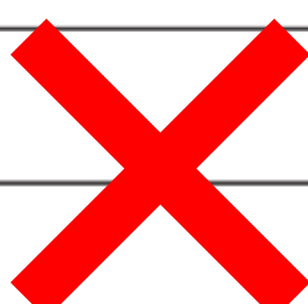
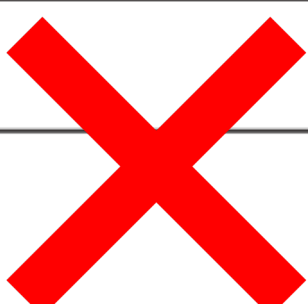
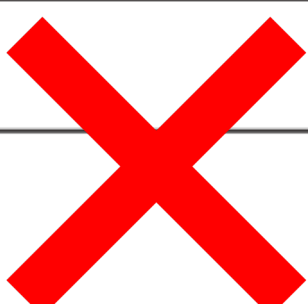
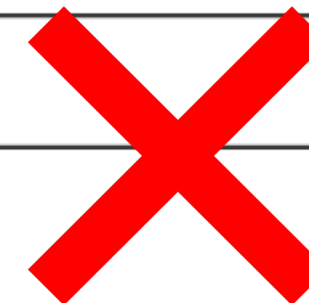
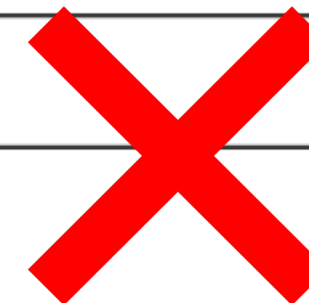


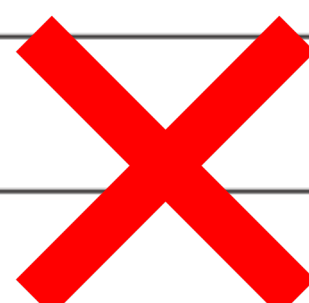
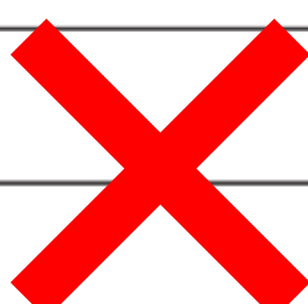
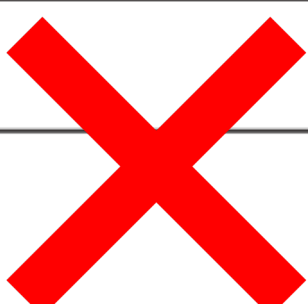
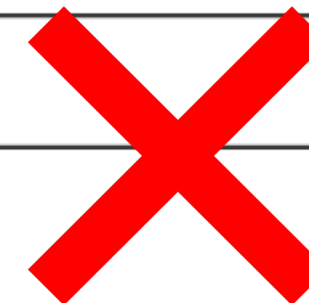



คลาสแบบ abstract คือ คลาสที่ไม่สมบูรณ์ คลาสที่มีเมธอดที่ยังไม่สามารถระบุการกระทำ (เมธอด) ให้ชัดเจนได้

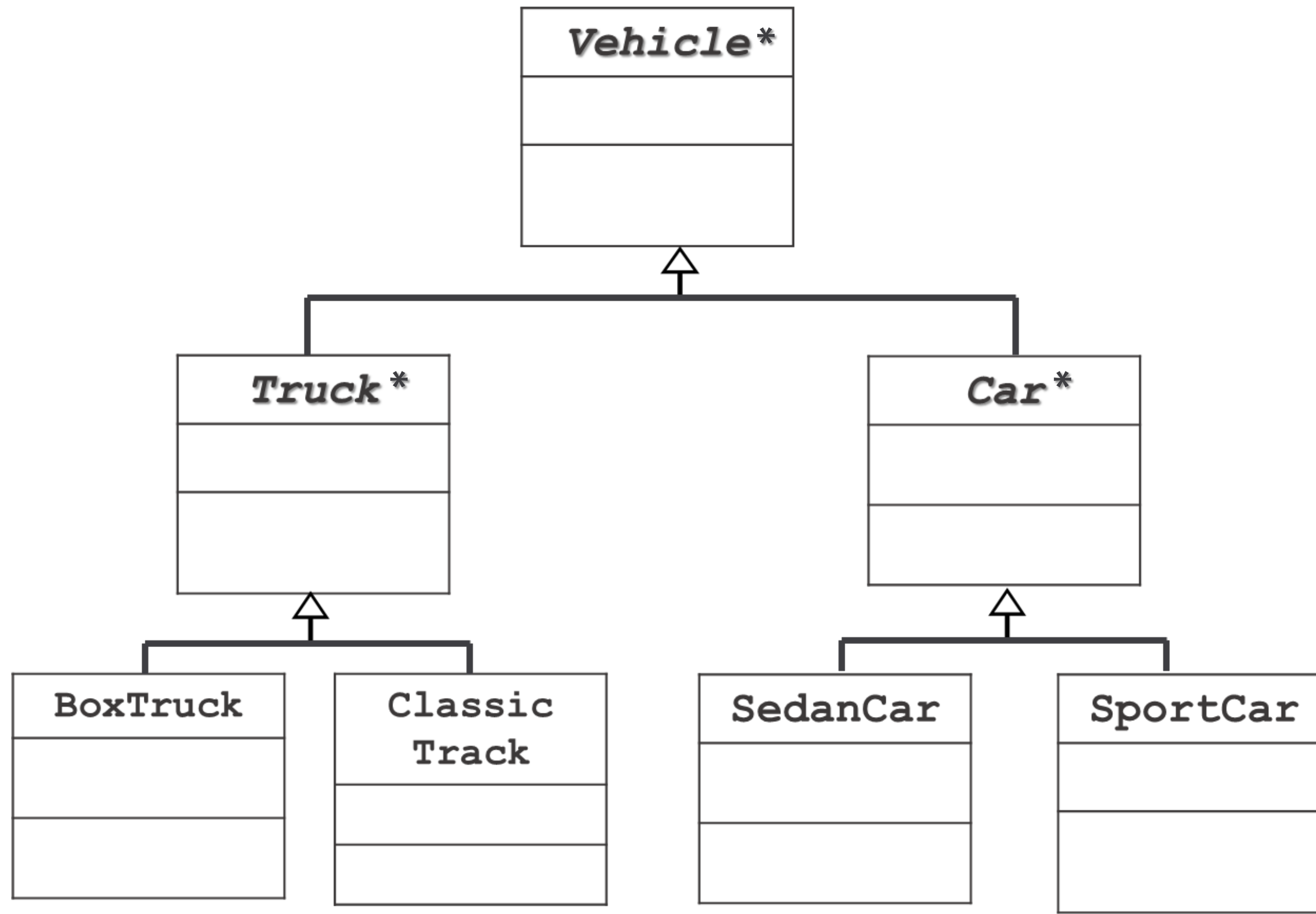
# คลาสแบบ abstract



# คลาสแบบ abstract

	<table><tr><td>Vehicle</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Vehicle			<table><tr><td>Cat</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Cat			<table><tr><td>Shape</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Shape			<table><tr><td>Pokemon</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pokemon					
Vehicle																			
																			
Cat																			
																			
Shape																			
																			
Pokemon																			
																			
<table><tr><td>Rectangle</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Rectangle			<table><tr><td>Dog</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Dog			<table><tr><td>Animal</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Animal			<table><tr><td>Fruit</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Fruit			<table><tr><td>Pikaju</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pikaju		
Rectangle																			
																			
Dog																			
																			
Animal																			
																			
Fruit																			
																			
Pikaju																			
																			
<table><tr><td>Car</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Car			<table><tr><td>Plane</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Plane			<table><tr><td>Apple</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Apple			<table><tr><td>Circle</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Circle			<table><tr><td>Orange</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Orange		
Car																			
																			
Plane																			
																			
Apple																			
																			
Circle																			
																			
Orange																			
																			

# คลาสแบบ abstract



# คลาสแบบ abstract

คลาสที่มี modifier เป็น abstract หมายความว่าคลาสนั้นยังเป็น **คลาสที่ไม่สมบูรณ์** โดยมีเมธอดแบบ abstract ซึ่งเป็น **เมธอดที่ยังไม่สมบูรณ์** อย่างน้อยหนึ่งเมธอดอยู่ในคลาส ในทางปฏิบัติอาจจะมี **เมธอดที่ยังไม่สมบูรณ์** เลยก็ได้

รูปแบบของเมธอดแบบ abstract

```
[modifier] abstract return_type methodName ([arguments]) ;
```

คลาสแบบ abstract กำหนดขึ้นมา **เพื่อให้คลาสอื่นสืบทอด** โดยคลาสที่มาสืบทอดจะต้องกำหนดบล็อกรหัสคำสั่งในเมธอดที่ยังไม่สมบูรณ์ (**บังคับ**) นอกจากนี้ **เราไม่สามารถสร้างอ็อบเจกต์ของคลาสแบบ abstract ได้**



# ตัวอย่างโปรแกรมคลาสแบบ abstract ที่ 1

```
public abstract class Student {
    protected String id;
    protected String name;
    public void setID(String ID) {    id = ID;    }
    public void setName(String n) {    name = n;    }
    public abstract void showDetails(); ★
}

public class FullTimeStudent extends Student {
    private int credit;
    private final int MAX_YEAR = 4;
    public FullTimeStudent(int c) {    credit = c;    }
    public void showDetails() {
        ★ {
            System.out.println("ID: "+id);
            System.out.println("Name: "+name);
            System.out.println("Credit: "+credit);
        } ★
    }
}

public class Main {
    public static void main(String[] args) {
        FullTimeStudent p1 = new FullTimeStudent(72);
        p1.showDetails();
    }
}
```

ผลลัพธ์

ID: null  
Name: null  
Credit: 72

# ตัวอย่างโปรแกรมคลาสแบบ abstract ที่ 2

```
public abstract class Student {
    protected String id;
    protected String name;
    public void setID(String ID) {          id = ID;          }
    public void setName(String n) {         name = n;         }
    public abstract void showDetails();
}

public class FullTimeStudent extends Student {
    private int credit;
    private final int MAX_YEAR = 4;
    public FullTimeStudent(int c) {         credit = c;      }
    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("Credit: "+credit);
    }
}

public class Main {
    public static void main(String[] args) {
        Student p0 = new Student();
        p0.showDetails();
    }
}
```

ผลลัพธ์

```
Main.java:3: error: Student is abstract;
cannot be instantiated
        Student p0 = new Student();
```

# ตัวอย่างโปรแกรมคลาสแบบ abstract ที่ 3

```
public abstract class Student {
    protected String id;
    protected String name;
    public void setID(String ID) {          id = ID;          }
    public void setName(String n) {         name = n;         }
    public abstract void showDetails();
}

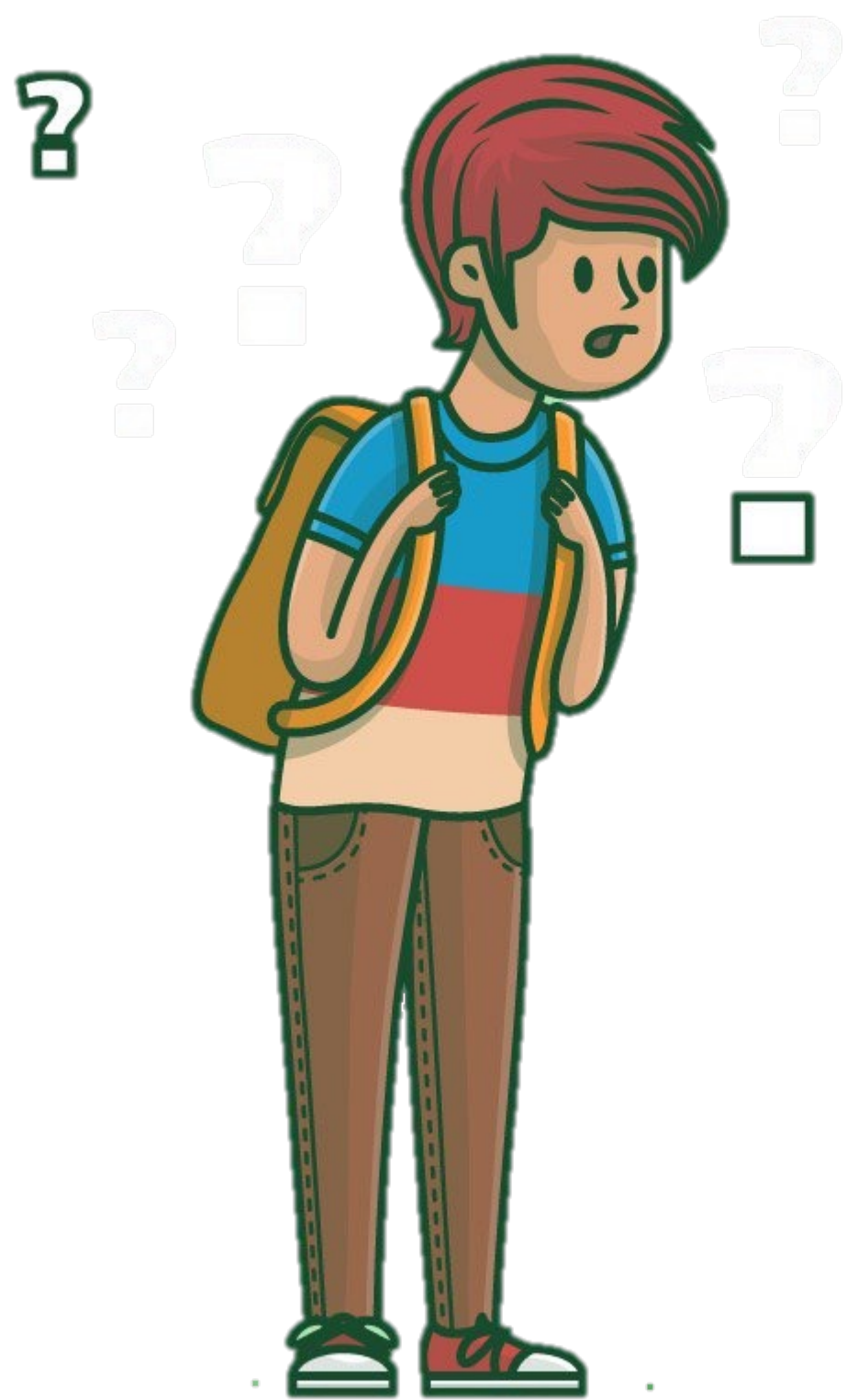
public class FullTimeStudent extends Student {
    private int credit;
    private final int MAX_YEAR = 4;
    public FullTimeStudent() {}
    public FullTimeStudent(int c) {          credit = c;      }
    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("Credit: "+credit);
    }
}

public class Main {
    public static void main(String[] args) {
        Student p0 = new FullTimeStudent();
        p0.showDetails();
    }
}
```

ผลลัพธ์  
????

# หัวข้อ

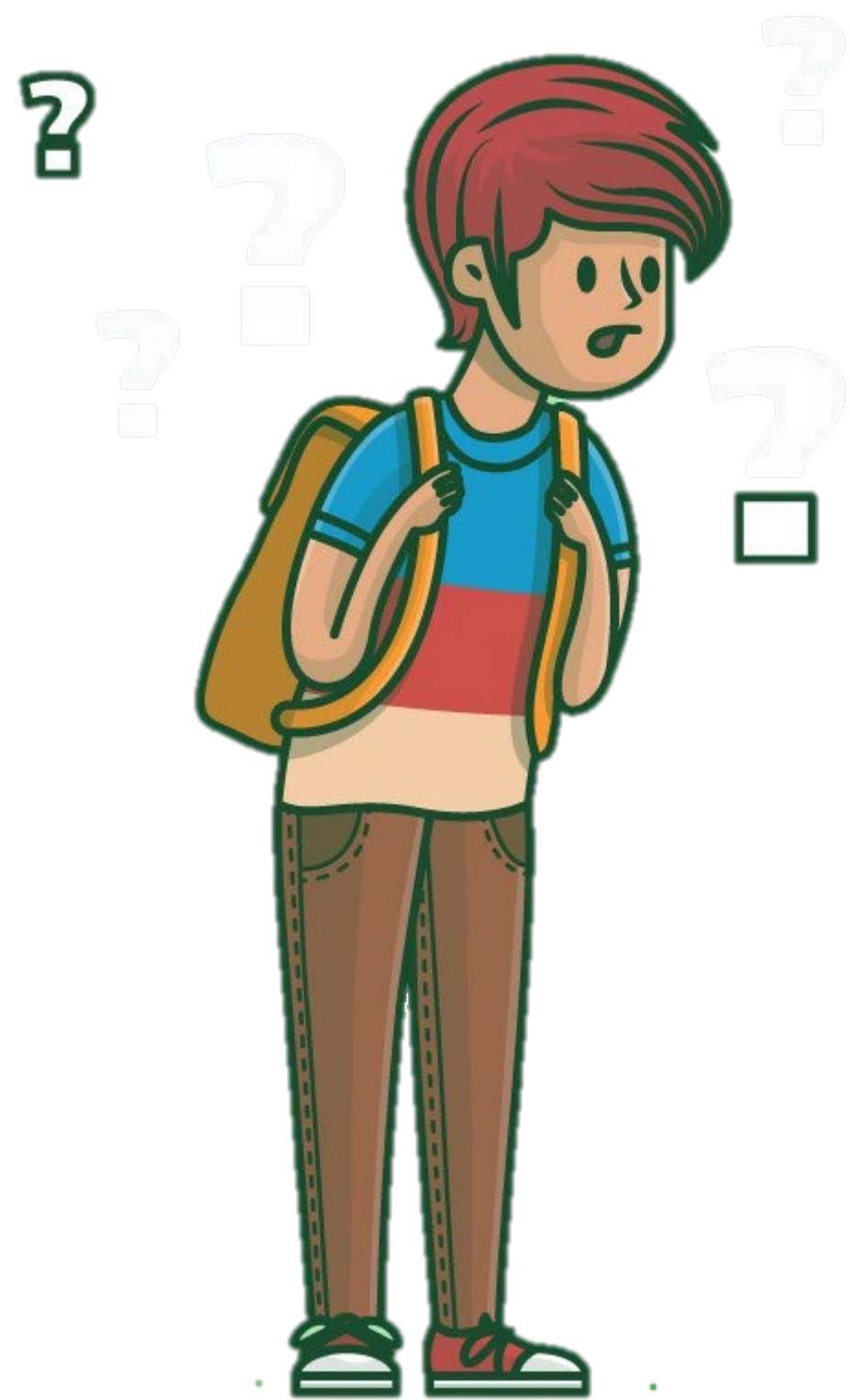
- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)



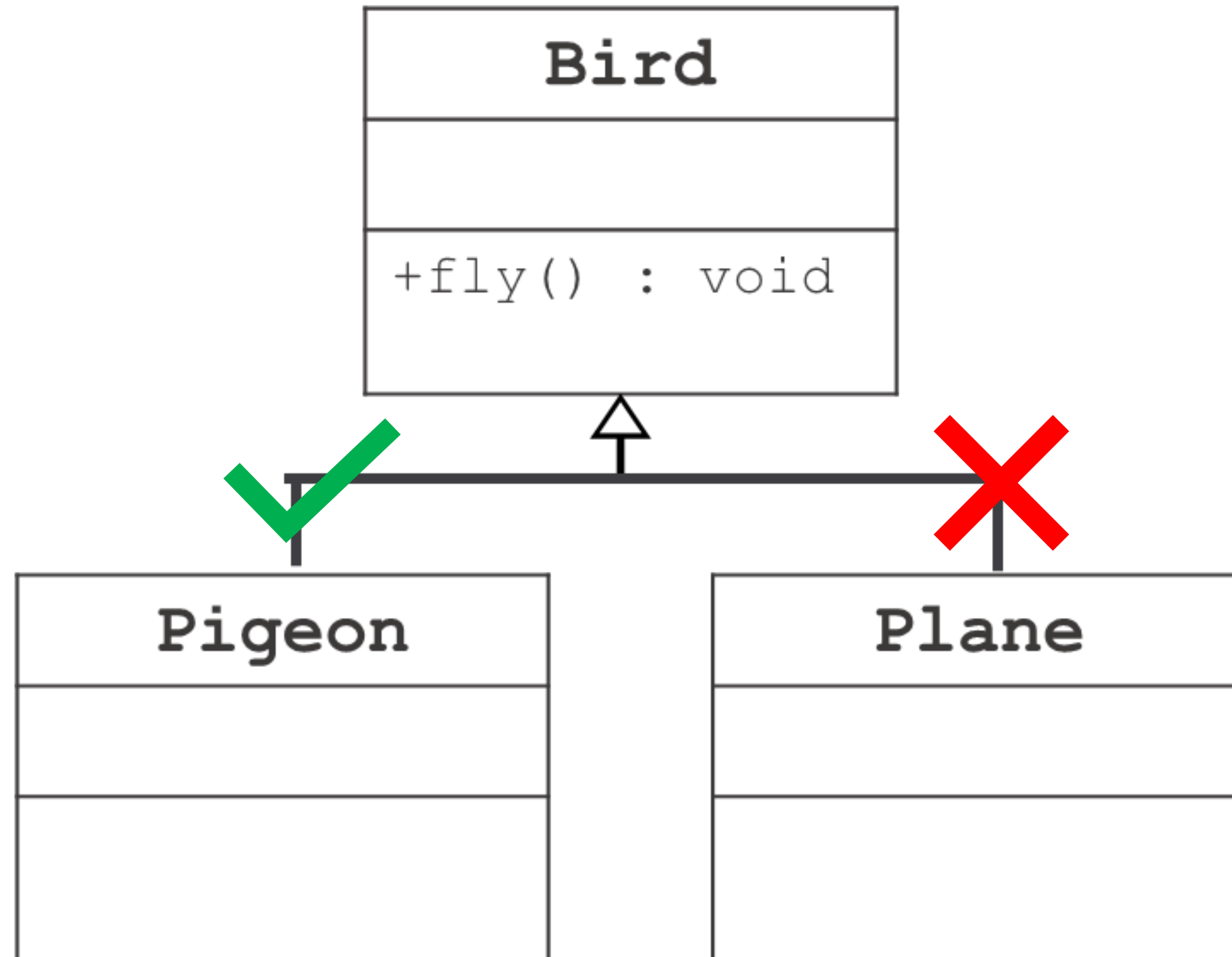


# หัวข้อ

- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)

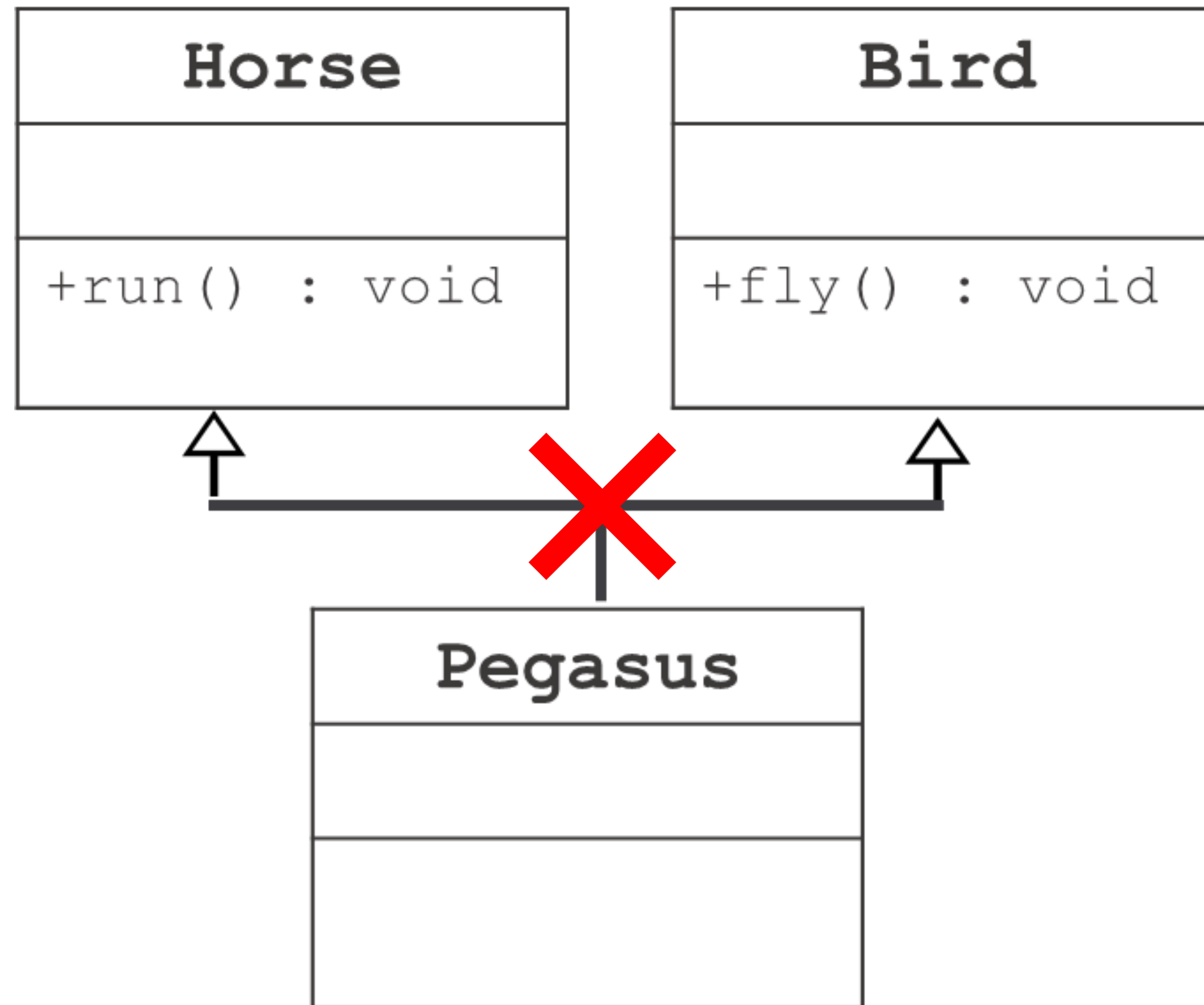


# อินเทอร์เฟส (interface)





# อินเทอร์เฟส (interface)



# อินเทอร์เฟส (interface)



อินเทอร์เฟส (interface) มีลักษณะคล้ายกับคลาสแบบ abstract แต่จะประกอบด้วย **เมธอดที่ยังไม่สมบูรณ์เท่านั้น** ซึ่งอินเทอร์เฟสจะเหมือนกับคลาสแบบ abstract ตรงที่เราจะไม่สามารถสร้างอ็อบเจกต์ของอินเทอร์เฟสได้

รูปแบบของอินเทอร์เฟส

```
[modifier] interface InterfaceName {  
    [methods () ;]  
}
```

อินเทอร์เฟสกำหนดขึ้นมาเพื่อให้คลาสอื่นนำไปใช้งาน โดยใช้คีย์เวิร์ด **implements** โดยมีรูปแบบดังนี้

```
[modifier] class ClassName implements InterfaceName {  
    [methods () ;]  
}
```

# ประโยชน์ของอินเทอร์เฟส



- **การกำหนดรูปแบบ**ของเมธอดต่าง ๆ ที่คลาสอื่น ๆ จะต้อง **implements ไว้ล่วงหน้า** ซึ่งสามารถอาศัยหลักการของการมีได้หลายรูปแบบมาเรียกใช้เมธอดเหล่านั้นได้จากคลาสที่ implements อินเทอร์เฟส
- ภาษาจาวากำหนดให้คลาสใด ๆ สามารถ**สืบทอดคลาสอื่นได้เพียงคลาสเดียวเท่านั้น** แต่จะสามารถ **implements อินเทอร์เฟสได้หลายอินเทอร์เฟส**

# แอททริบิวต์และเมธอดของอินเตอร์เฟส



เมื่อประกาศอินเตอร์เฟสดังนี้

```
public interface Moveable {  
    int AVERAGE_SPEED = 40;  
    void move();  
}
```

แอททริบิวต์ของอินเตอร์เฟสจะกำหนดให้เป็น **public static final** และเมธอดของอินเตอร์เฟสจะกำหนดให้เป็น **public abstract**

```
public interface Moveable {  
    public static final int AVERAGE_SPEED = 40;  
    public abstract void move();  
}
```

# แอททริบิวต์และเมธอดของอินเตอร์เฟส



ในทางปฏิบัติแล้ว ถ้าผู้พัฒนาอยากได้แอททริบิวต์ของอินเตอร์เฟสที่ไม่ใช่ `public static final` อาจจะสามารถหลีกเลี่ยงได้ดังนี้

```
public interface Walkable {  
    //private double distance;  
    public double getDistance();  
    public void setDistance(double distance);  
}
```

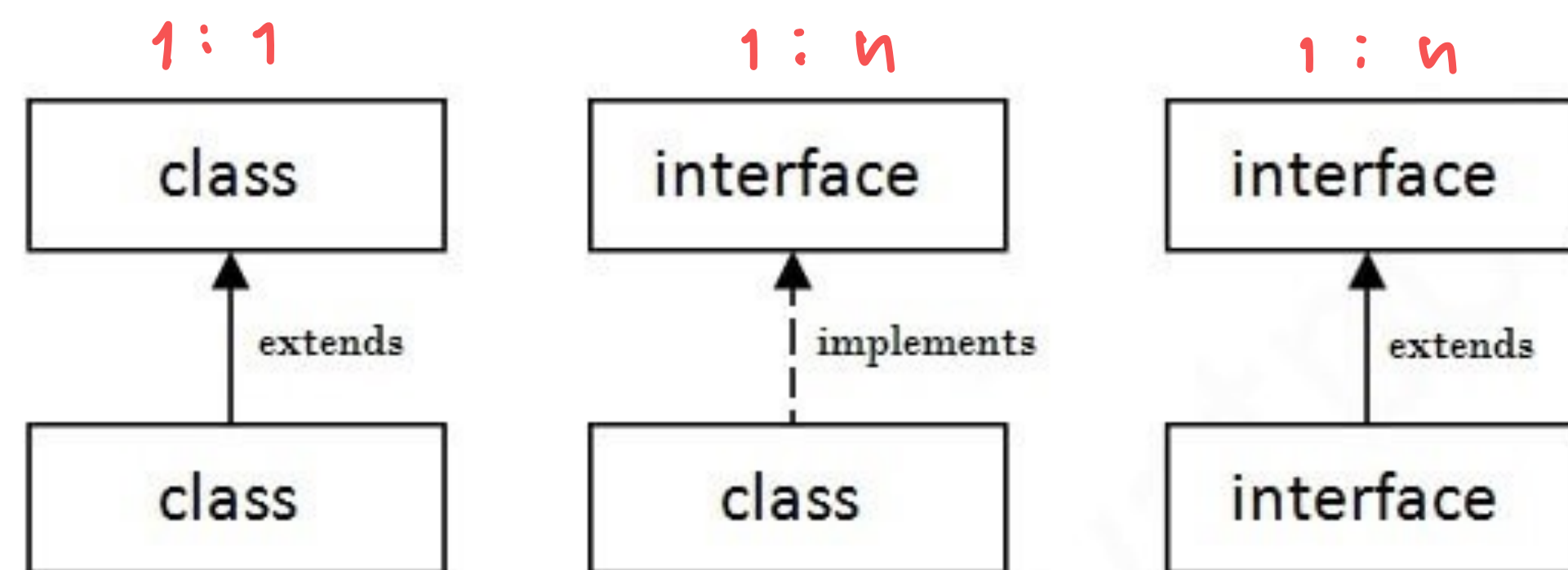
# แอททริบิวต์และเมธอดของอินเตอร์เฟส



- ทุกเมธอดภายในอินเตอร์เฟส**ไม่สามารถ**กำหนดให้เป็น **static** และ **final** ได้
- ทุกเมธอดภายในอินเตอร์เฟส**ถูกบังคับให้เป็น** **public abstract** โดยอัตโนมัติถึงแม้ว่าเราจะได้กำหนดก็ตาม
- ทุกแอททริบิวต์ในอินเตอร์เฟสจะ**ถูกบังคับให้เป็น** **public static final** หรือกล่าวได้ว่าเป็นค่าคงที่ (**constants**)
- อินเตอร์เฟส กับ อินเตอร์เฟสจะ **extends** และสามารถ **extends** ได้**มากกว่า 1** อินเตอร์เฟส เช่น

```
public interface Hockey extends Sports, Event {...}
```

- อินเตอร์เฟส**ไม่สามารถ** **implement** คลาสได้
- อินเตอร์เฟสสามารถซ้อนกันได้





# ตัวอย่างอินเทอร์เฟซที่ 1

```
public interface Student {  
    public void setID(String ID);  
    public void setName(String n);  
    public void showDetails();  
}  
  
public class PartTimeStudent implements Student {  
    private String id;  
    private String name;  
    private int credit;  
    public PartTimeStudent(int c) {        credit = c;        }  
    public void setID(String ID) {        id = ID;        }  
    public void setName(String n) {        name = n;        }  
    public void showDetails() {  
        System.out.println("ID: "+id);  
        System.out.println("Name: "+name);  
        System.out.println("Credit: "+credit);  
    }  
}
```

# ตัวอย่างอินเทอร์เฟซที่ 2

```
public interface Runnable {
    int x = 10;
    void move(int n);
    void move();
} public interface Flyable {
    public static int x = 100;
    public abstract void move();
}

public class Pegasus implements Runnable, Flyable {
    public static void main(String[] args) {
        // System.out.println(x); >>> error: reference to x is ambiguous
        System.out.println(Runnable.x);
        System.out.println(Flyable.x);
        new Pegasus().move();
    } public void move(){
        move(1);
    } public void move(int i) {
        for (int j = 1; j <= i; j++)
            System.out.println("M m M m");
    }
}
```

} ถ้า attr ไม่ชัดเจนไม่ได้บอกให้ก็ได้

ผลลัพธ์

10

100

M m M m

# คลาสไม่สมบูรณ์ vs อินเตอร์เฟส

Component		Interface	Abstract Class
	Constructor	×	✓
แอมพลิฟาย	Static	✓	✓
	Non-Static	×	✓
	Final	✓	✓
	Non-final	×	✓
เมธอด	Abstract	✓	✓
	Static	✓	✓
	Non-static	✓	✓
	Final	×	✓
	Non-final	✓	✓
Access Modifier	private	×	✓
	default	✓	×
	protected	×	✓
	public	✓	✓

# หัวข้อ

- เมธอด Constructor
- คีย์เวิร์ด this และ super และเมธอด this() และ super()
- คลาส (Class)
  - คลาสสมบูรณ์ (Class)
  - คลาสไม่สมบูรณ์ (Abstract class)
  - คลาสภายใน (Inner class หรือ Nested class)
- อินเตอร์เฟส (Interface)

