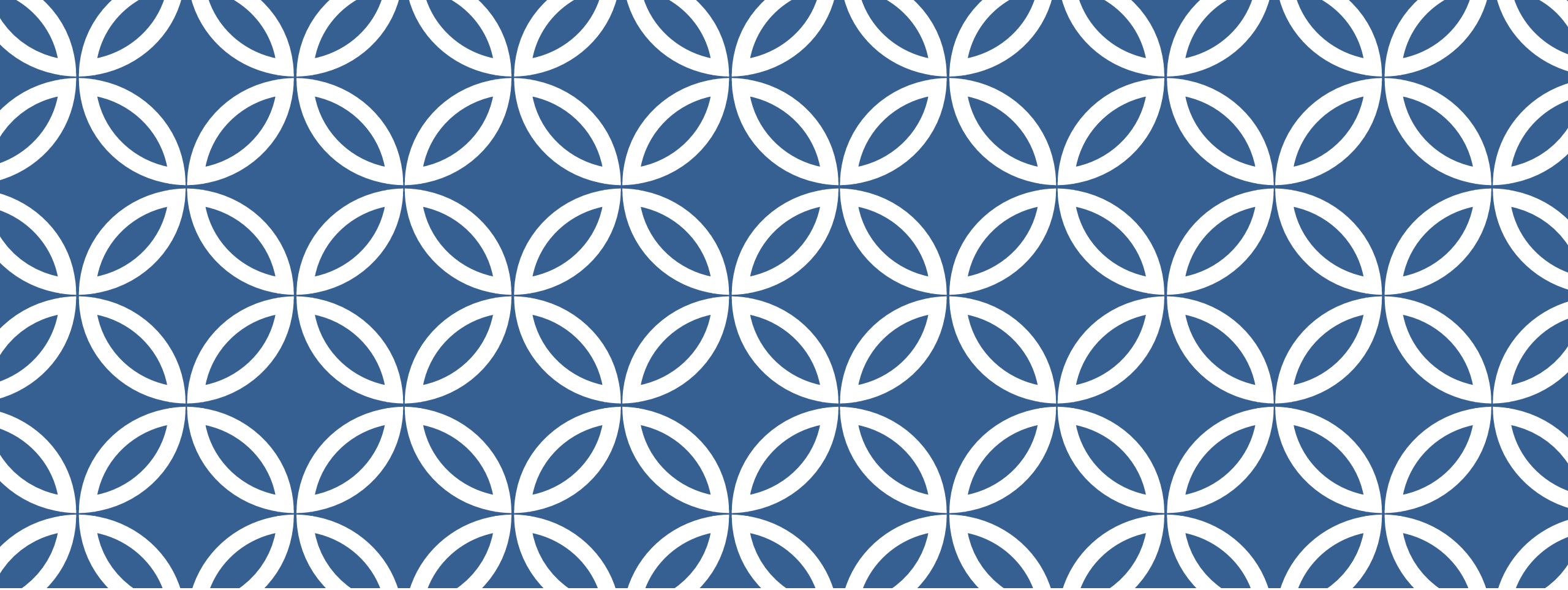


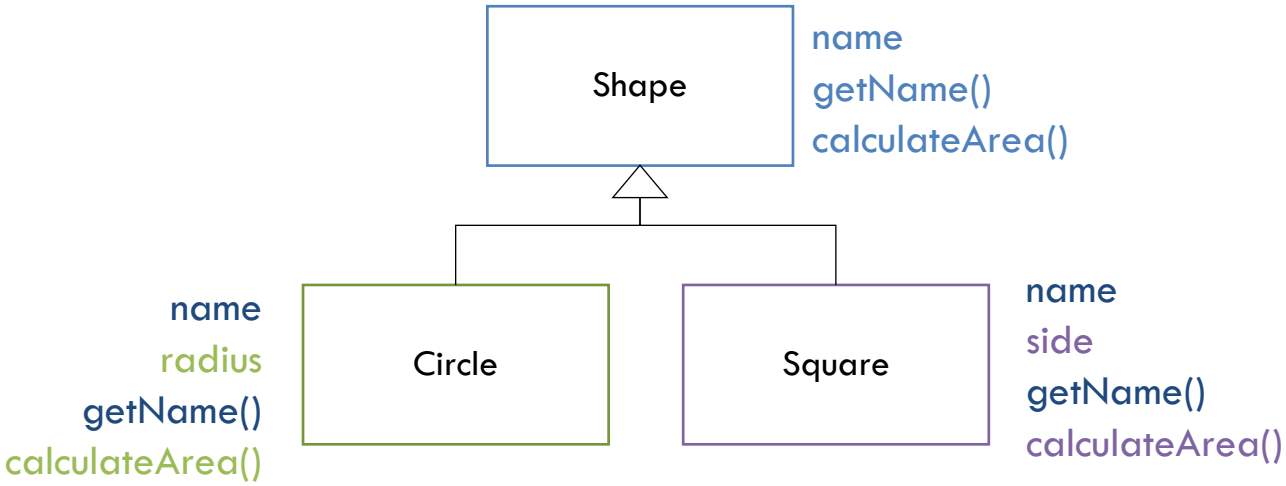
บทที่ 4
INHERITANCE PART 2

อ.สกรณ์ บุษบง
สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
มหาวิทยาลัยราชภัฏบุรีรัมย์

ADDING/DELETING A CLASS

- การเพิ่ม **class** เข้ามายัง **hierarchy** ที่มีอยู่แล้วนั้นมีผลต่อเสถียรภาพของ **hierarchy** เป็นอย่างมาก
- ควรเพิ่มในลักษณะของ **subclass** จะไม่ส่งผลต่อ เสถียรภาพของ **hierarchy**
- ดังตัวอย่างต่อไปนี้

ADDING/DELETING A CLASS



ADDING/DELETING A CLASS

```
public class Shape {
    private String name;
    public Shape(String aName) {
        name=aName;
    }
    public String getName() {
        return name;
    }
    public float calculateArea() {
        return 0.0f;
    }
}
```

```
public class Circle extends Shape {
    private int radius;
    public Circle(String aName) {
        super(aName);
        radius = 3;
    }
    public float calculateArea() {
        float area;
        area = (float) (3.14 * radius * radius);
        return area;
    }
}
```

```
public class Square extends Shape{
    private int side;
    public Square(String aName) {
        super(aName);
        side = 3;
    }
    public float calculateArea() {
        int area;
        area = side * side;
        return area;
    }
}
```

ADDING/DELETING A CLASS

จะกล่าวในบทถัดไป เนื่องจากเป็นเรื่องเกี่ยวกับ **Polymorphism**

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Shape shapeArray[] = {c, s};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
             shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

```
The area of Circle C is 28.26 sq. cm.
The area of Square S is 9.0 sq. cm.
```

ADDING/DELETING A CLASS

Circle
Attribute
- name : Circle c
- radius : 3

Operations
- getName()
- calculateArea()

Square
Attributes
- name : Square S
- side : 3

Operations
- getName()
- calculateArea()

ADDING/DELETING A CLASS

- สมมุติว่าเราต้องการเพิ่ม class เข้าไปใน Shape class hierarchy
- Class ใหม่ คือ Triangle

ADDING/DELETING A CLASS

```
public class Triangle extends Shape {
    private int base, height;
    Triangle(String aName) {
        super(aName);
        base = 4;
        height = 5;
    }
    public float calculateArea() {
        float area = 0.5f * base * height;
        return area;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Triangle t = new Triangle("Triangle T");
        Shape shapeArray[] = {c, s, t};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
              shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

```
The area of Circle C is 28.26 sq. cm.
The area of Square S is 9.0 sq. cm.
The area of Triangle T is 10.0 sq. cm.
```

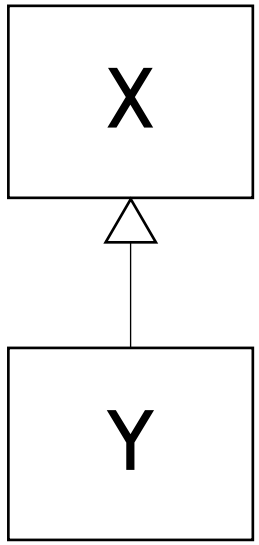
ADDING/DELETING A CLASS

- จากการเขียน **code** ดังกล่าวจะเห็นได้ชัดว่าการเพิ่ม **subclass** นั้นส่งผลกระทบต่อ **class hierarchy** น้อยมากหรือไม่ส่งผลเลย
- ดังนั้น การลบ **subclass** ที่ไม่ได้เป็น **superclass** ของ **class** อื่นๆก็จะเกิดผลกระทบต่อ **class hierarchy** น้อยมากหรือไม่ส่งผลเลย

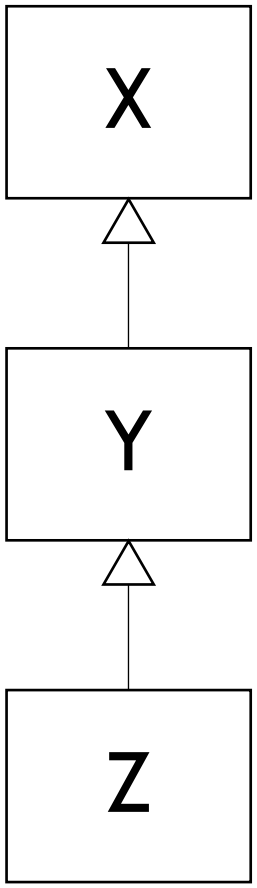
INHERITANCE CHAIN

- **class hierarchie** เป็นลักษณะโครงสร้างที่ได้กล่าวไปแล้วในบทที่ผ่านมา
- เส้นทางที่เชื่อมระหว่าง **class** ต่างๆ ใน **class hierarchie** เรียกว่า **“inheritance chain “**
- **inheritance chain 1** เส้น สามารถเป็นแบบ ***single-level*** หรือ ***multi-level*** ก็ได้

INHERITANCE CHAIN



Single-level Single Inheritance Chain

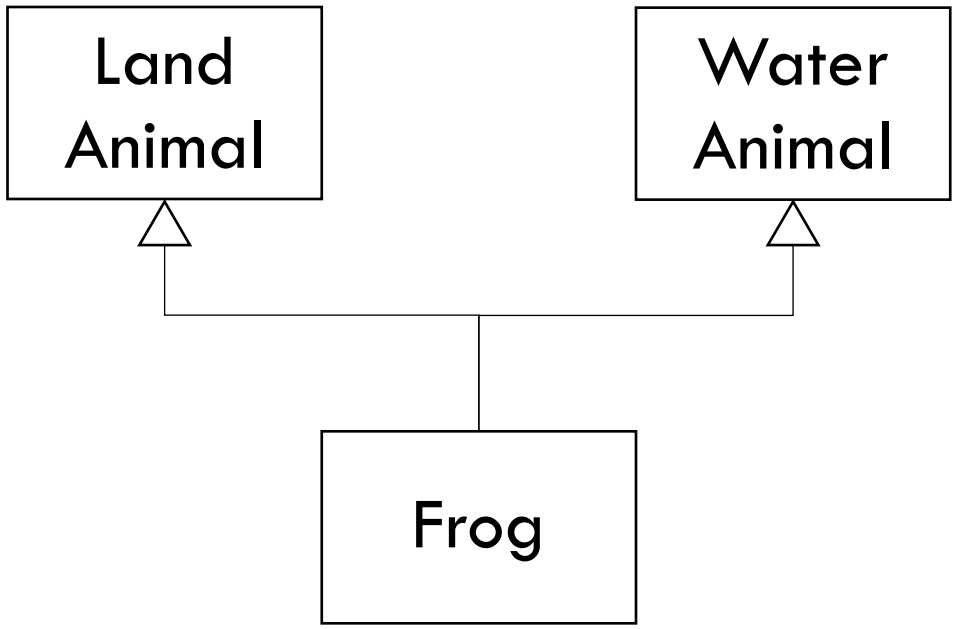


Multiple-level Single Inheritance Chain

MULTIPLE INHERITANCE

- พิจารณากรณีของ กบ
  - กบเป็นสัตว์ครึ่งบกครึ่งน้ำ (amphibian)
  - สัตว์ครึ่งบกครึ่งน้ำ มีคุณสมบัติร่วมกันระหว่าง
    - สัตว์บก (land animal)
    - สัตว์น้ำ (water animal)

MULTIPLE INHERITANCE

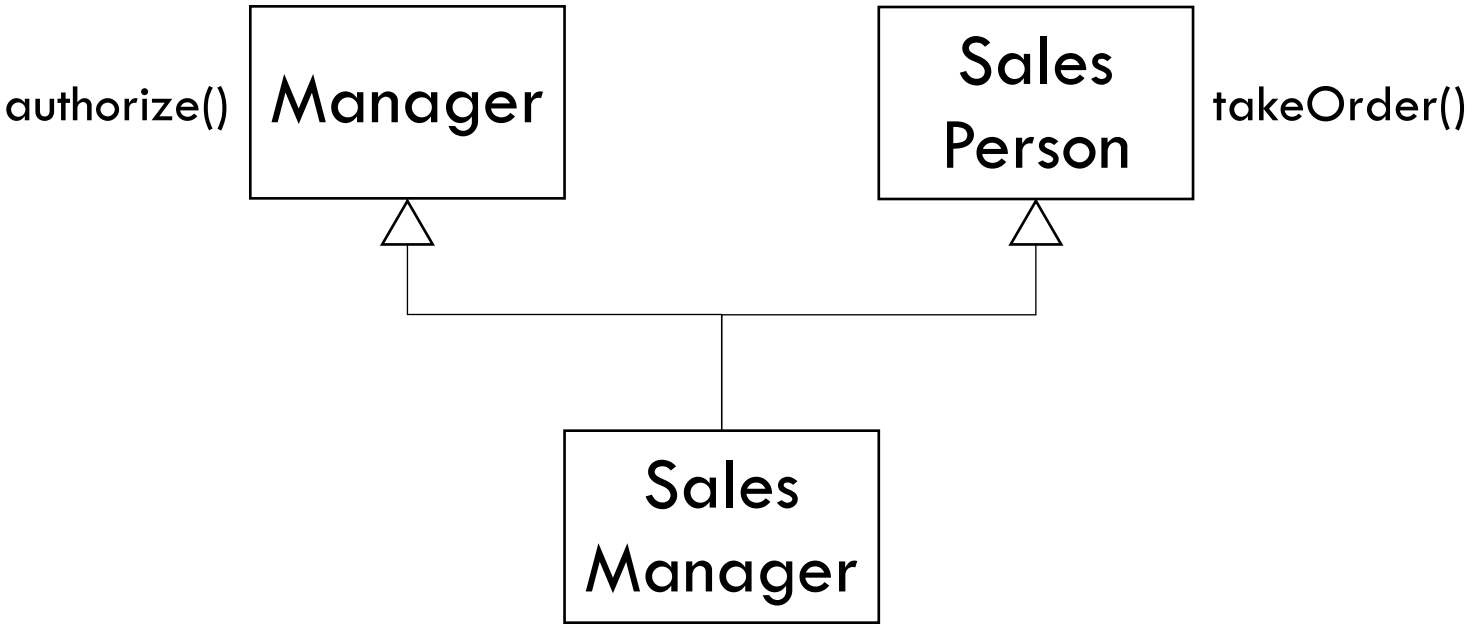


- จากการสืบทอดมากกว่า 1 **superclass**
- Class Frog เรียกว่า **multiple inheritance**
- **Attribute** และ **Method** ทั้งหมดของ **class Land Animal** และ **Water Animal** กลายเป็นส่วนหนึ่งของ class Frog

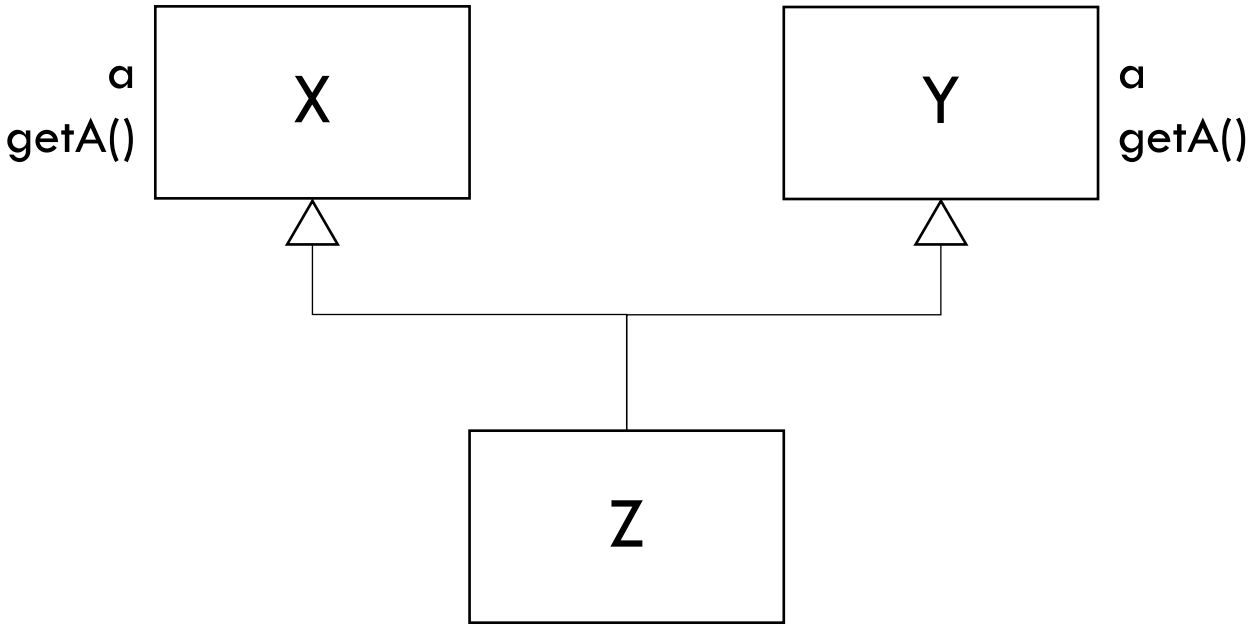
PROBLEMS ASSOCIATED WITH MULTIPLE INHERITANCE

- พิจารณาตัวอย่างของผู้ตัวอย่างของ ผู้จัดการฝ่ายขาย (sales manager)
- ผู้จัดการฝ่ายขายสามารถรับรายการสั่งซื้อ (takeOrder) จากลูกค้าได้เหมือนที่พนักงานขายทำได้ และสามารถอนุมัติการสั่งซื้อ เหมือนกับที่ผู้จัดการสามารถทำได้
- ดังนั้น class SalesManager จึงเป็น subclass ของ 2 superclass คือ class SalesPerson และ class Manager

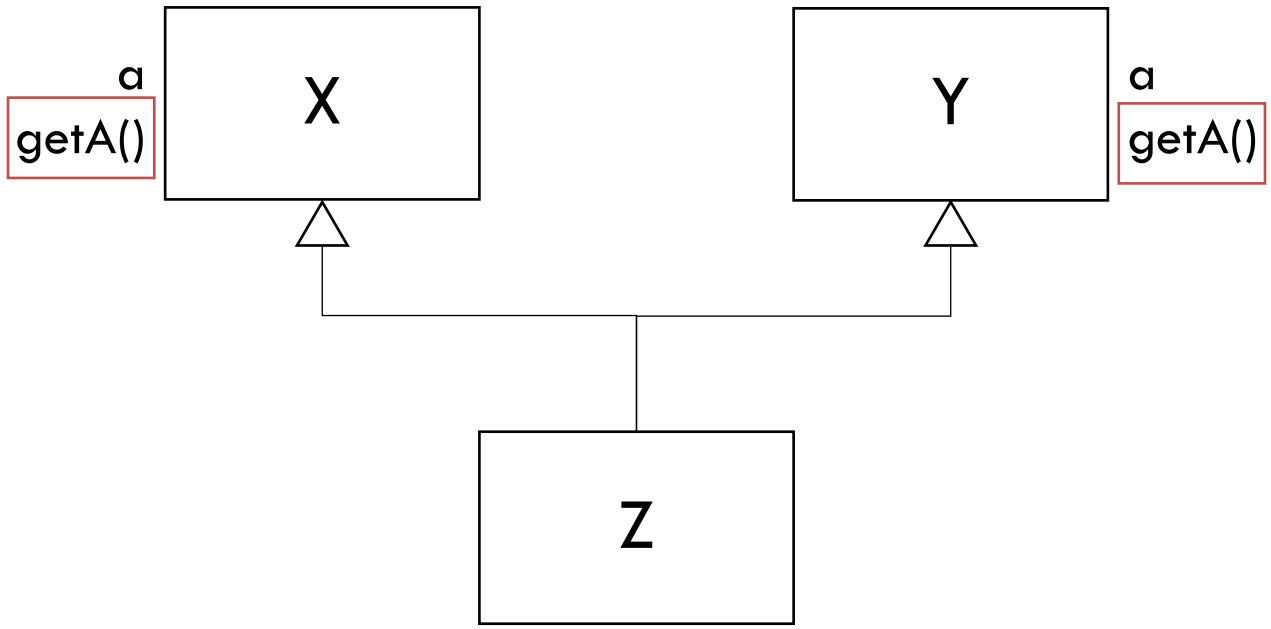
PROBLEMS ASSOCIATED WITH MULTIPLE INHERITANCE



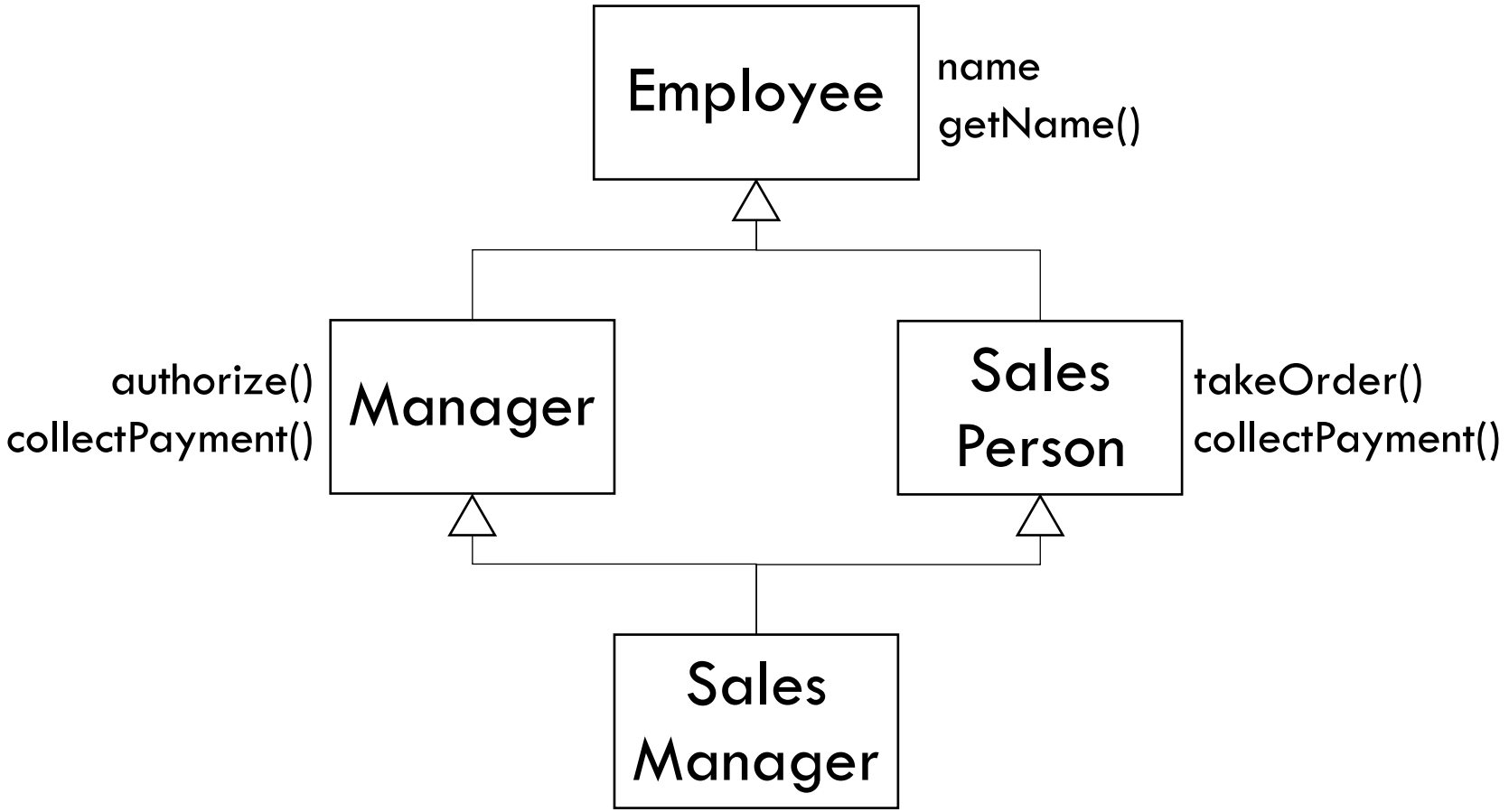
PROBLEMS ASSOCIATED WITH MULTIPLE INHERITANCE



PROBLEMS ASSOCIATED WITH MULTIPLE INHERITANCE



PROBLEMS ASSOCIATED WITH MULTIPLE INHERITANCE

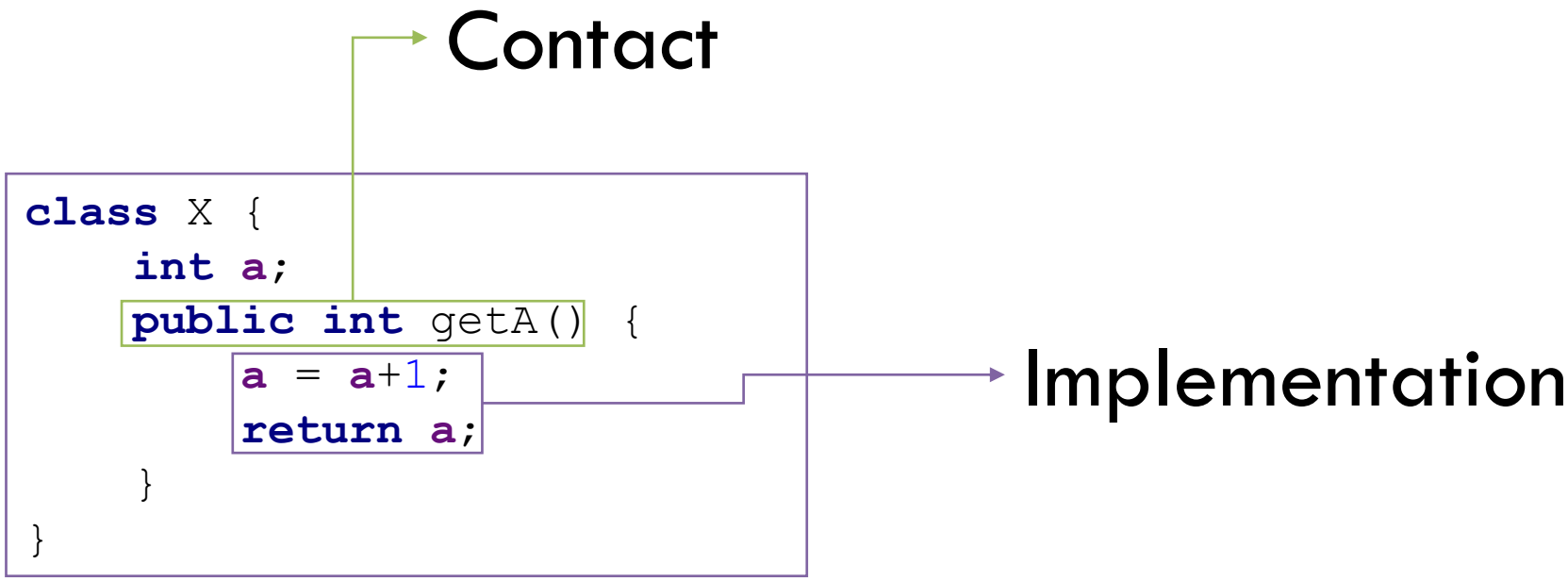


- SalesManager จะใช้ Method collectPayment() ของใคร ????

CONTRACT AND IMPLEMENTATION PARTS

- ปกติแล้ว **method** จะประกอบไปด้วย 2 ส่วน
- Contract : method signature and specifies the name of the method, its return type and formal parameters.
- Implementation : contains statements that are executed when a method is called

CONTRACT AND IMPLEMENTATION PARTS



CONTRACT AND IMPLEMENTATION INHERITANCE

- เพื่อหลีกเลี่ยงปัญหาความไม่ชัดเจนของ **method** ที่ **inherit** มาจาก หลาย **superclass** ภาษาจาวาจึงไม่รองรับ **Multiple Inheritance**
- จาวาจึงมีวิธีการที่การยอมให้ **subclass inherit Multiple Contact** ได้ แต่ไม่สามารถ **inherit Multiple Implementation**
- **Subclass** จะต้องดำเนินการส่วน **Implementation** เอง
- จากวิธีดังกล่าวทำให้ **Multiple Inheritance** ยังคงใช้ได้ในภาษาจาวา แต่ไม่สามารถใช้ได้ตรงๆ

INTERFACE

- ภาษาจาวารองรับ Contract inheritance ผ่านการสร้าง interface
- การสร้าง **interface** นั้นเหมือนกับการสร้าง **class** เว้นแต่ใช้ keyword “interface”

INTERFACE

```
interface I {
    void j();
    int k();
}
```

- Method ทั้งหมดของ interface เห็น abstract method
- Method จะมีการประกาศแต่ส่วน Contact ไม่มีส่วน Implement
- ส่วน Implement ให้ subclass ประกาศเอง

MULTIPLE INHERITANCE USING INTERFACE

- การสร้าง **interface** ในภาษา **Java** ใช้ **single inheritance** เพื่อให้เกิด **multiple inheritance**
- ดังตัวอย่างต่อไปนี้

MULTIPLE INHERITANCE USING INTERFACE

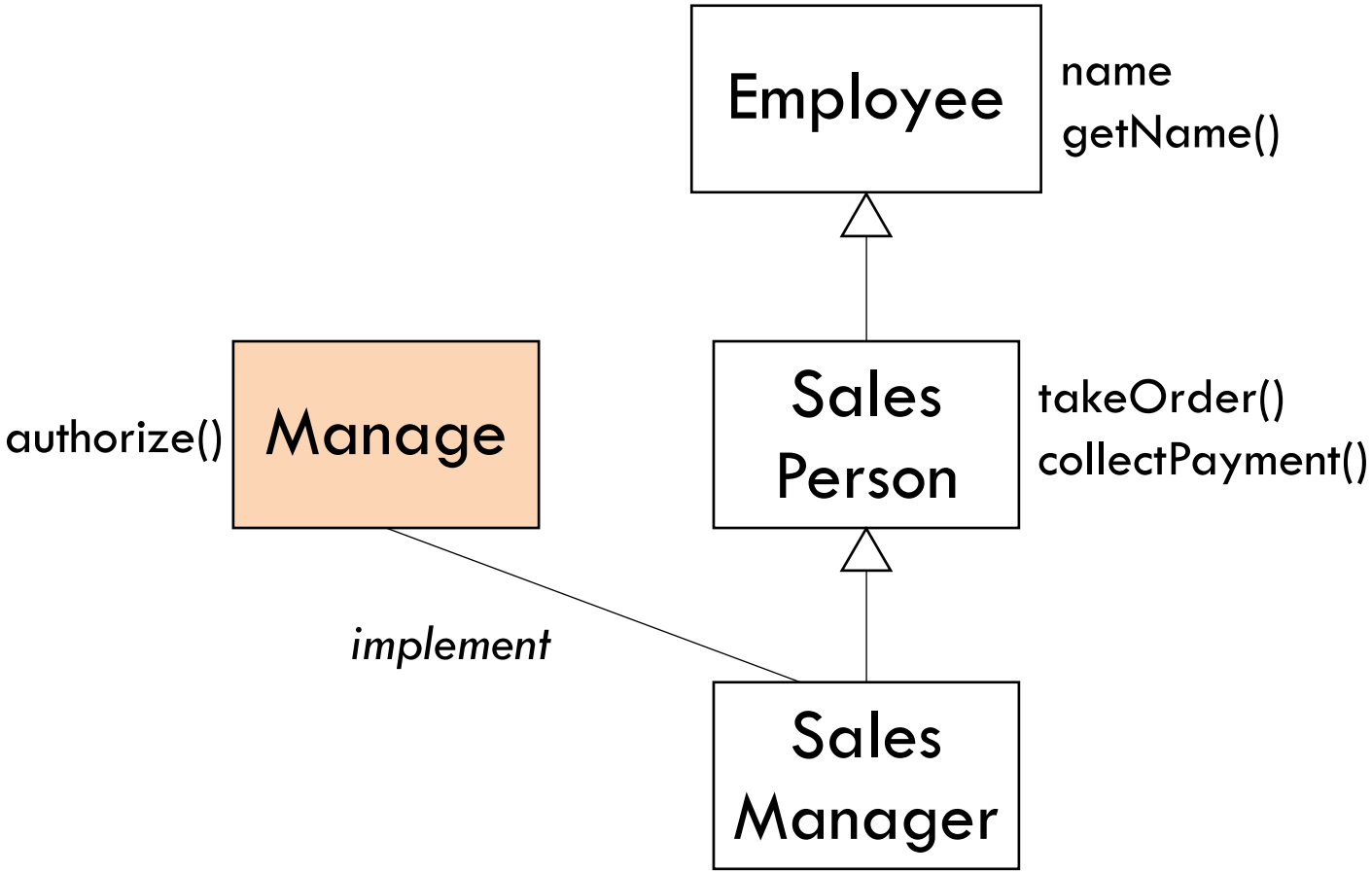
- Class SalesManager เป็น subclass ของ SalesPerson ใน single inheritance chain.
- SalesPerson เป็น subclass ของ class Employee
- ทำให้ class SalesManager ได้ inherit จาก SalesPerson และ Employee เท่านั้น ไม่มีความสามารถในการจัดการ

MULTIPLE INHERITANCE USING INTERFACE

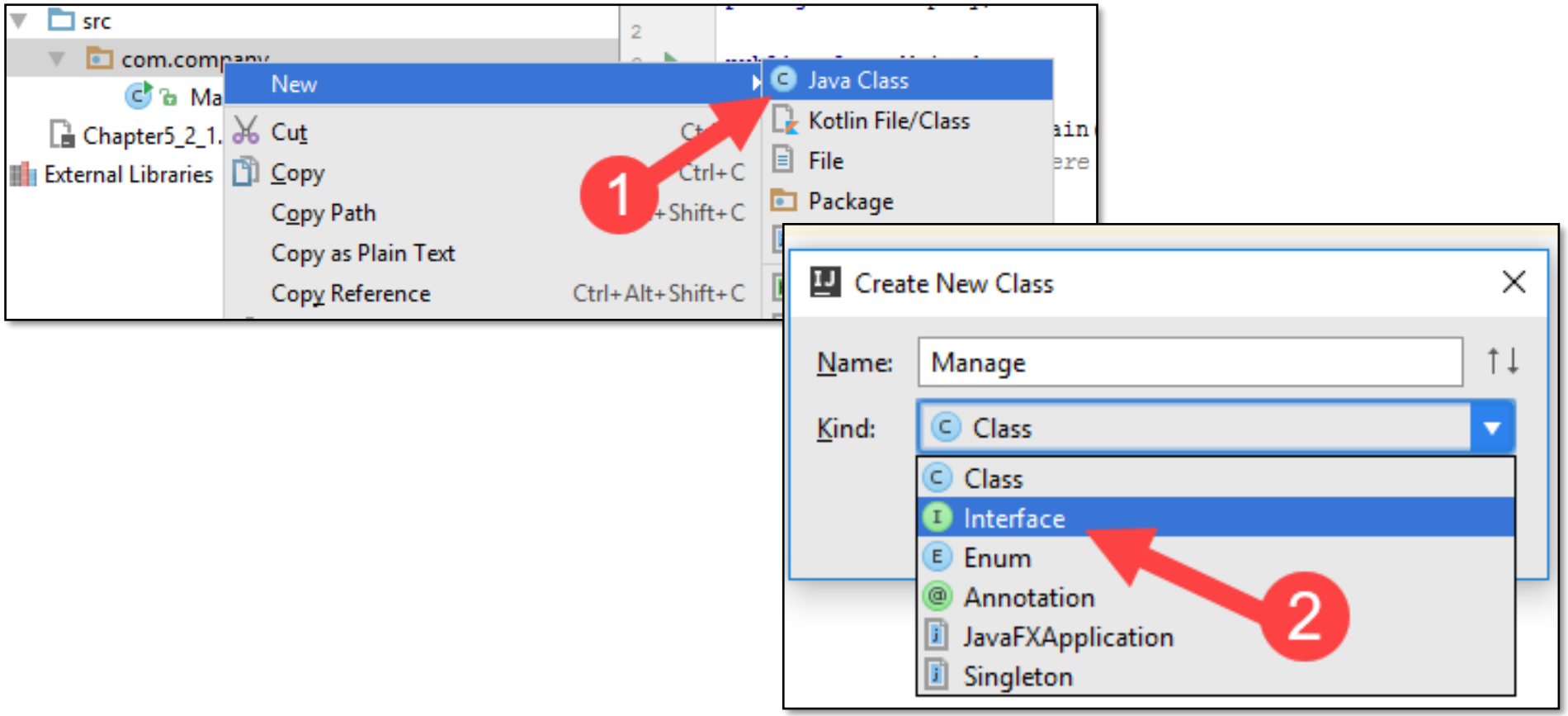
- เพื่อเพิ่มความสามารถในการจัดการ (manage) ให้ class SalesManager จึงใช้ interface ที่เก็บความสามารถในการ Manage ไว้
- เรียกว่า interface Manage

```
interface Manage {
    public boolean authorize();
}
```

MULTIPLE INHERITANCE USING INTERFACE



MULTIPLE INHERITANCE USING INTERFACE



MULTIPLE INHERITANCE USING INTERFACE

```
public interface Manage {
    public boolean authorize();
}
```

```
public class Employee {
    private String name;
    public Employee() {}
    public String getName() {
        return name;
    }
}
```

```
public class SalesPerson extends Employee {
    public boolean takeOrder() {
        System.out.println("Order taken");
        return true;
    }
    public void collectPayment() {
        System.out.println("Payment collected");
    }
}
```

MULTIPLE INHERITANCE USING INTERFACE

```
public class SalesManager extends SalesPerson implements Manage {
    public SalesManager(String n) {
        super.name = n;
    }
    public boolean authorize() {
        // authorisation by a sales manager
        System.out.println("Order authorized");
        return true;
    }
}
```

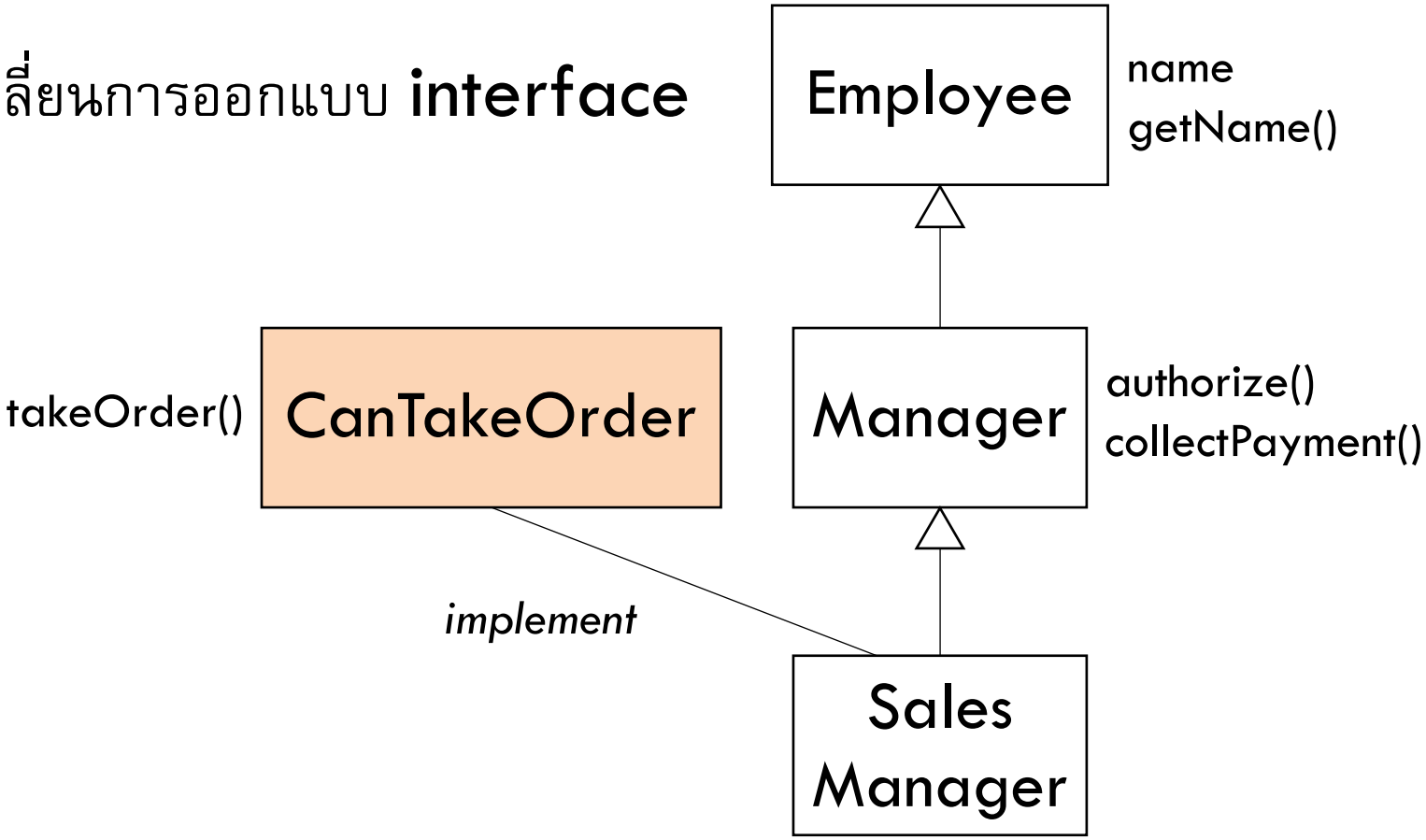
MULTIPLE INHERITANCE USING INTERFACE

```
public class Main {
    public static void main(String[] args) {
        SalesManager sm = new SalesManager("Sara");
        if (sm.takeOrder()) {
            if (sm.authorize()) {
                sm.collectPayment();
                System.out.println("SalesManager " + sm.getName()+
                  " took order, authorized it and collected payment.");
            }
            else {
                System.out.println("SalesManager "+sm.getName()+
                  " did not authorize order. No payment collected. ");
            }
        }
    }
}
```

Order taken
Order authorized
Payment collected
SalesManager Sara took order, authorized it and collected payment.

MULTIPLE INHERITANCE USING INTERFACE

- เปลี่ยนการออกแบบ **interface**



ATTRIBUTES IN AN INTERFACE

- Attribute ใน interface จะเป็น static และ final เสมอ
- Static เพราะ attribute จะมีเพียงหนึ่งเดียว
- Final เพราะไม่สามารแก้ไขได้

ATTRIBUTES IN AN INTERFACE

```
public interface Colourable {
    int RED = 1;
    int GREEN = 2;
    int BLUE = 3;

    public void setColour (int c);
    public int getColour();
}
```

```
public class Color implements Colourable {
    public void setColour (int c){
        RED = c;
    }
    public int getColour(){
        return RED;
    }
}
```

METHODS IN AN INTERFACE

- Method ทั้งหมดใน interface เป็น abstract method และทุกๆ class ที่ใช้ interface จะต้อง implement method เหล่านี้ด้วย
- Method ใน interface เป็น public เสมอ แม้ไม่ใส่ keyword ในส่วน Access

ABSTRACT CLASS AND INTERFACE

- Class ที่ใช้งาน **interface** จะต้อง **implement** ทุกๆ **method** ที่ประกาศใน **interface** ไม่งั้น **class** จะกลายเป็น **abstract class**

```
public abstract class ColourTest implements Colourable {
    int i;
    public ColourTest() {}

    public void setColour (int c) {
        i=c;
    }
}
```

ABSTRACT CLASS AND INTERFACE

Abstract Class	Interface
May have some methods declared abstract.	Can only have abstract methods.
May have protected properties and static methods.	Can only have public methods with no implementation.
May have final and nonfinal data attributes.	Limited to only static and final.

ABSTRACT CLASS AND INTERFACE

- Abstract class ไม่สามารถสร้าง object ได้

```
public abstract class LandVehicle {
    int doors;
    public  LandVehicle() {
        doors = 4;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        LandVehicle l = new LandVehicle();
    }
}
```

EXTENDING INTERFACE

- Subclass ของ class ที่ Implement interface จะได้รับการสืบทอดคุณสมบัติของ interface นั้นหรือไม่ ????

EXTENDING INTERFACE

```
public interface I {
    public void x();
}
```

```
public class A implements I {
    public void x() {
        System.out.println("in A.x");
    }
    public void y() {
        System.out.println("in A.y");
    }
}
```

```
public class B extends A {
    public void z() {
        x();
        y();
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        A aa = new A();
        B bb = new B();
        bb.z();
    }
}
```

```
in A.x
in A.y
```

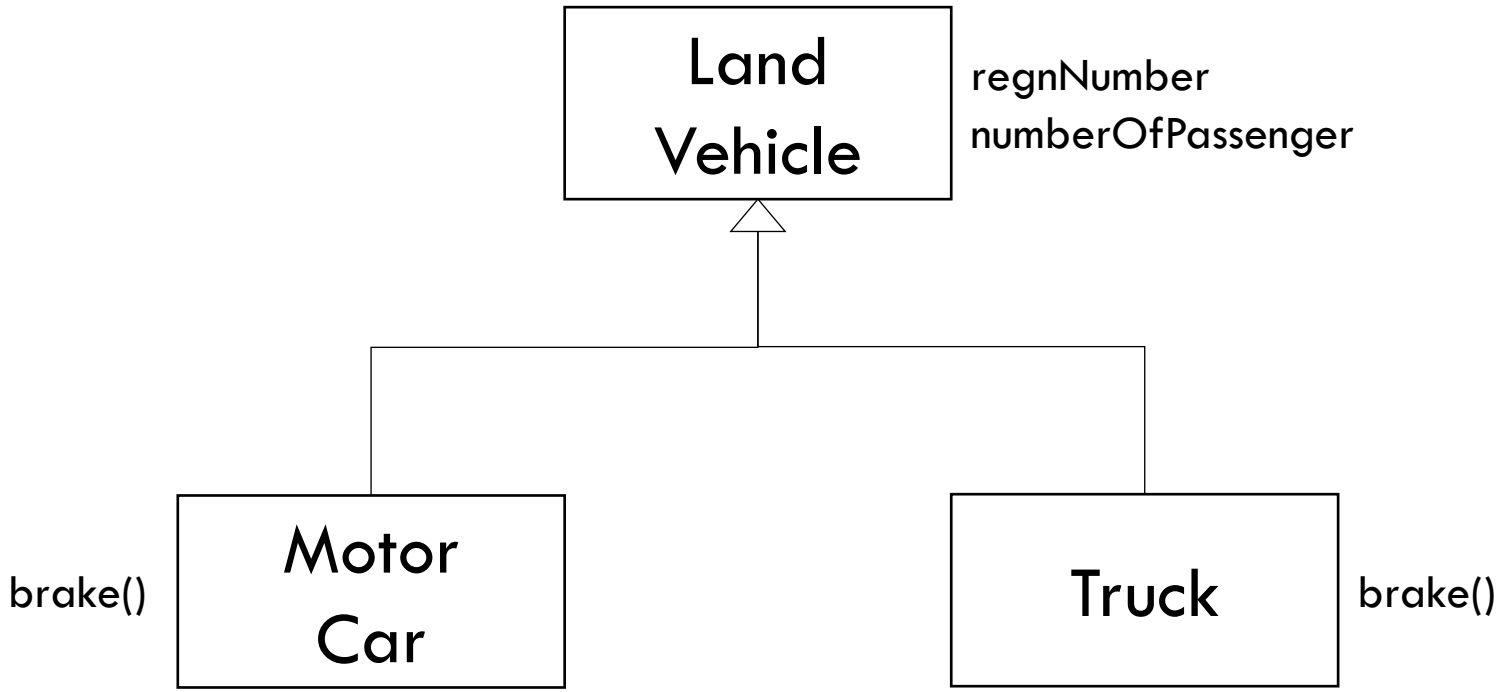
EXTENDING INTERFACE

- จากการทดลอง พบว่า **method x()** และ **y()** ถูกเรียกใช้งาน
- **Class B** เป็น **subclass** ของ **class A**
- **Class B** ได้รับการสืบทอดไม่เพียงแต่ **method y()** แต่ได้รับการสืบทอด **method x()** ซึ่งเป็น **method** ของ **interface I** ด้วย

LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE

- แม้ว่า **interface** จะช่วยในการจัดการปัญหา **multiple inheritance** ใน **class hierarchy**
- แต่การใช้งาน **interface** ก็มีข้อจำกัด

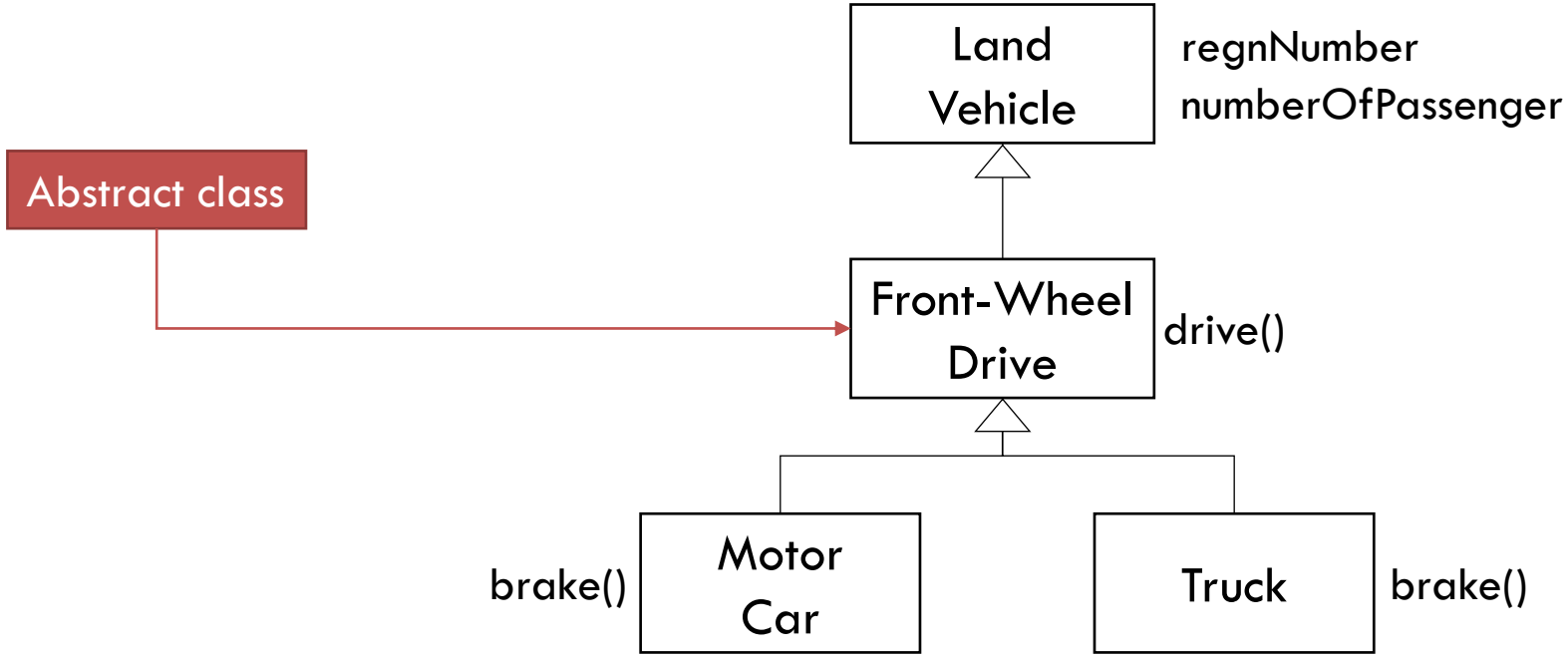
LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE



LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE

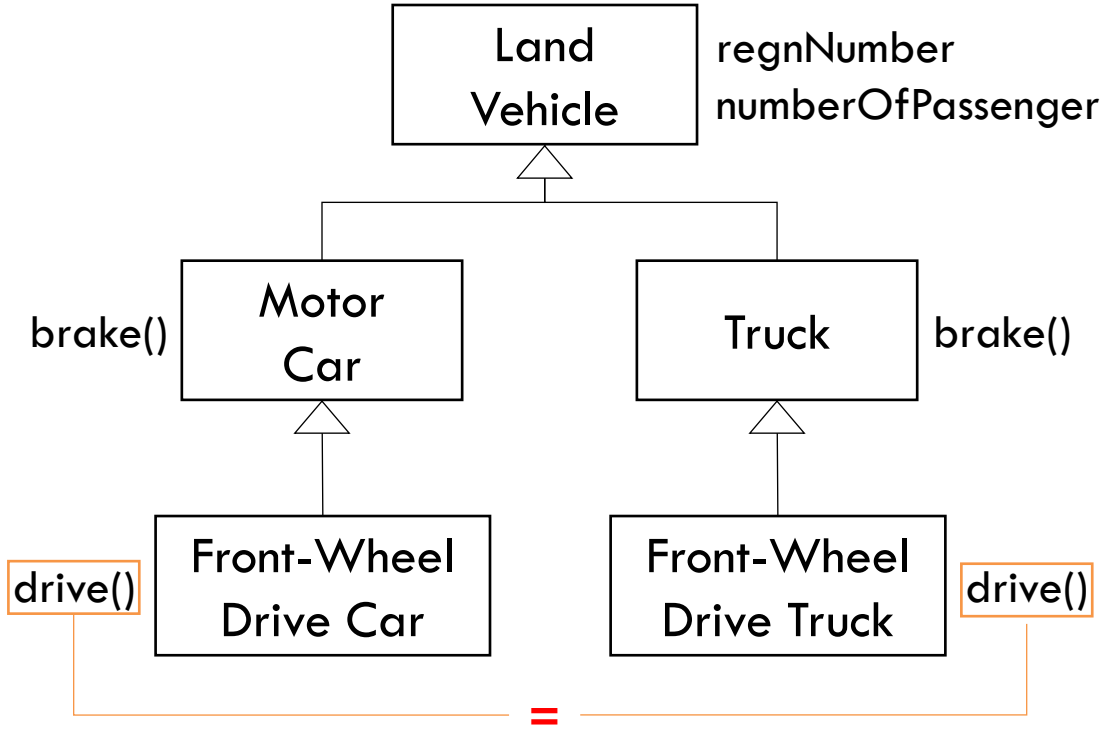
- ถ้า ทั้ง **class Motor Car** และ **class Truck** ต่างก็มีคุณลักษณะของการขับเคลื่อนแบบล้อหน้า (**Front-Wheel Drive**)
-

LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE



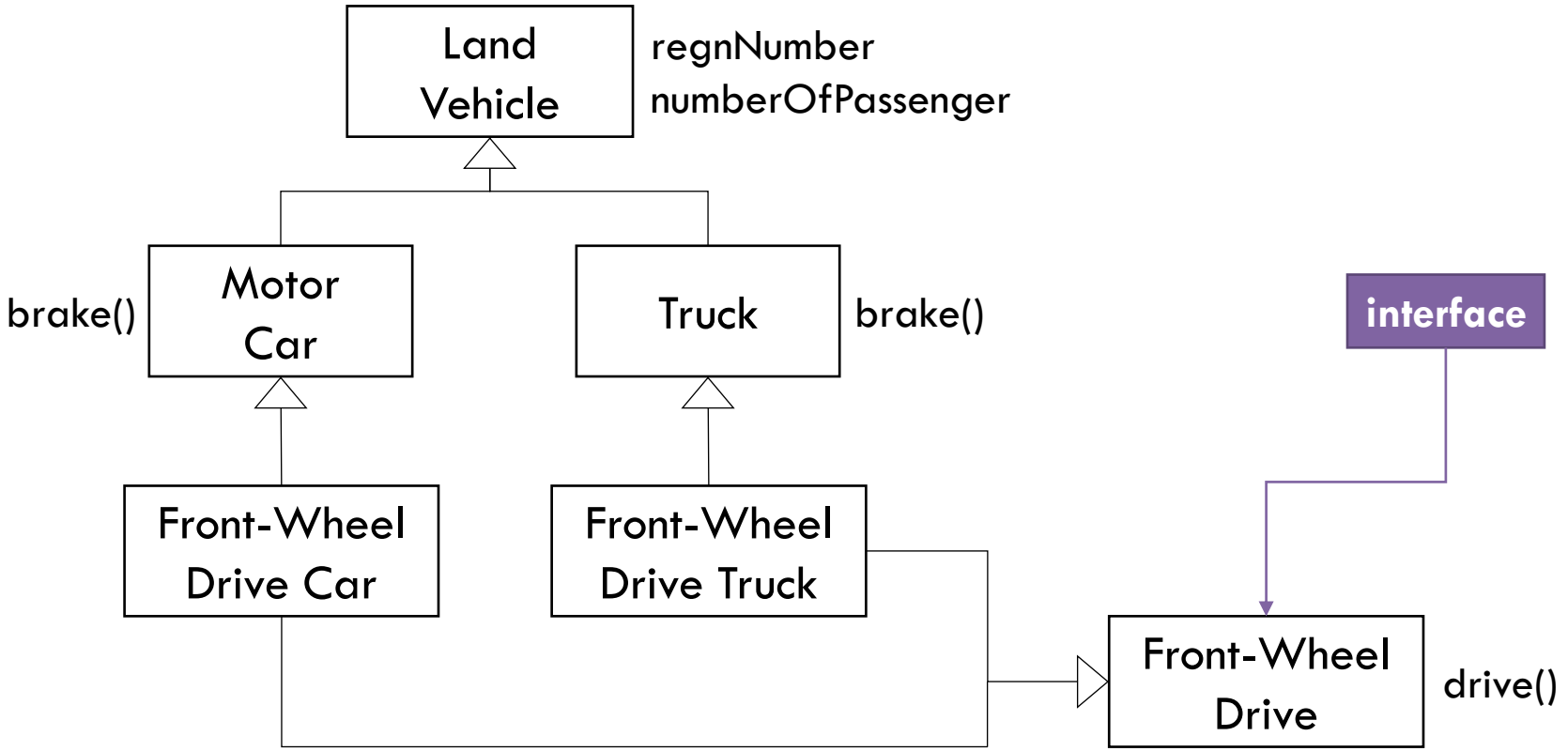
LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE

- ทำการปรับให้สอดคล้องกับความจริง



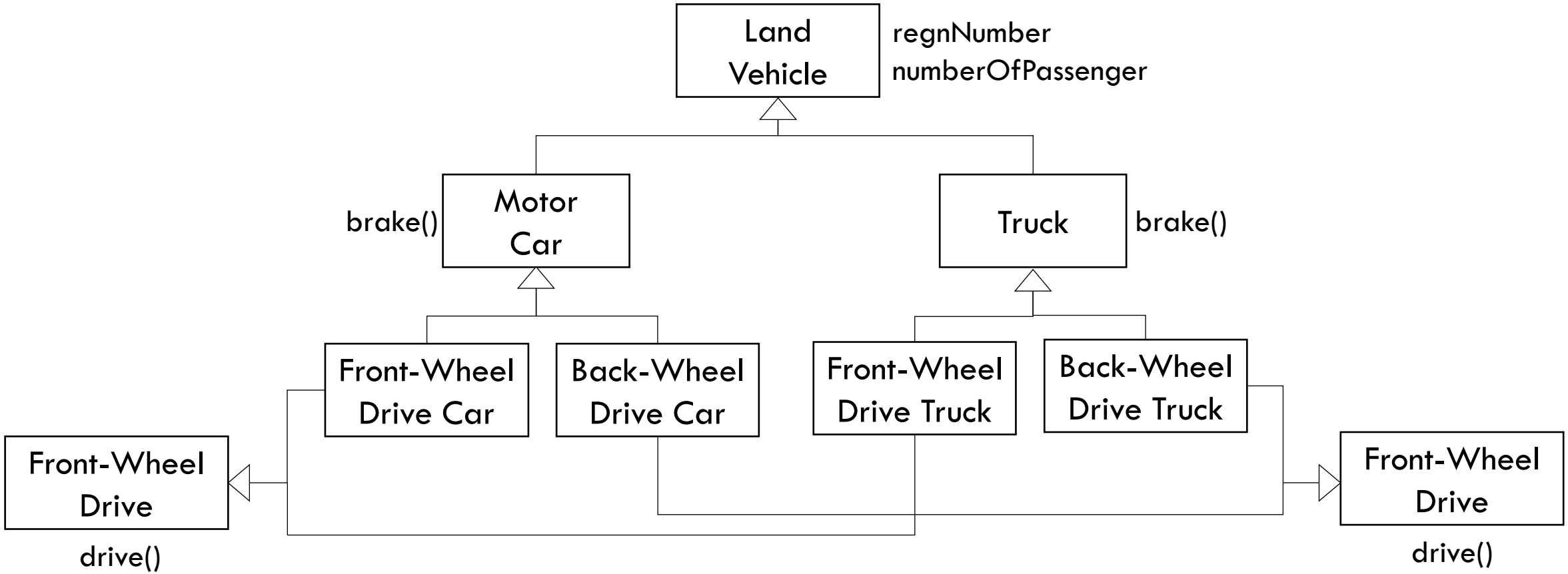
LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE

- แก้ไขปัญหา **method drive()** ที่ซ้ำกัน



LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE

- แก้ไขปัญหา **method drive()** ที่ซ้ำกัน



LIMITATIONS OF INTERFACE FOR MULTIPLE INHERITANCE > NO CODE REUSE

- จากตัวอย่างจะเห็นว่า การใช่ interface นั้นไม่มีคุณสมบัติ Code Reuse

สรุป