บทที่ 6

การพ้องรูป

Polymorphism

สอนโดย อ.เรวดี พิพัฒน์สูงเนิน สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏอุดรธานี

หลักการเขียนโปรแกรมเชิงวัตถุ

วัตถุประสงค์การเรียนรู้

- เข้าใจแนวคิดของการพ้องรูป (Polymorphism) ในการเขียนโปรแกรมเชิงวัตถุ
- จำแนกความแตกต่างระหว่างการพ้องรูปที่เกิดขึ้นขณะคอมไพล์ (Compile-time Polymorphism) และขณะรันไทม์ (Runtime Polymorphism)
- สามารถใช้งานเมธอดโอเวอร์โหลด (Method Overloading) ได้อย่างถูกต้อง
- สามารถใช้งานเมธอดโอเวอร์ไรด์ (Method Overriding) ได้อย่างถูกต้อง
- นำหลักการพ้องรูปไปประยุกต์ใช้เพื่อทำให้โค้ดมีความยืดหยุ่นและบำรุงรักษาง่าย

ทบทวน : หลักการพื้นฐานของ OOP

- การเขียนโปรแกรมเชิงวัตถุ (OOP) คืออะไร?
- Class (คลาส) : แบบพิมพ์เขียวของวัตถุ
- Object (วัตถุ) : สิ่งที่ถูกสร้างขึ้นจากคลาส มีสถานะและพฤติกรรม
- 4 แนวคิดหลักของ OOP ได้แก่
  - Encapsulation (การห่อหุ้ม)
  - Inheritance (การสืบทอด)
  - **Polymorphism (การพ้องรูป)**
  - Abstraction (การซ่อนรายละเอียด)

Polymorphism คืออะไร?

- **Polymorphism** มาจากภาษากรีก หมายถึง “หลายรูปแบบ”
  - โพลี (Poly) แปลว่า "มากมาย" “หลาย”
  - มอร์ฟ (Morph) แปลว่า "รูปแบบ" หรือ "รูปร่าง"
- **Polymorphism** คือความสามารถของวัตถุในการรับรูปร่างที่หลากหลาย หรือแสดงพฤติกรรมที่แตกต่างกันในสถานการณ์ที่แตกต่างกัน

Polymorphism คืออะไร?

- **ในการเขียนโปรแกรม**
  - การที่ Object หนึ่งสามารถมีพฤติกรรมที่แตกต่างกันได้
  - การเรียกใช้ Method เดียวกัน แต่ได้ผลลัพธ์ที่แตกต่างกัน
  - ขึ้นอยู่กับประเภทของ Object ที่เรียกใช้

ประเภทของ Polymorphism

แบ่งออกเป็น 2 ประเภท

1. Compile-time Polymorphism

- หรือ Static Polymorphism
- เกิดขึ้นในระหว่างการคอมไพล์โค้ดจาวาตัดสินใจว่าจะเรียกเมธอดตัวไหนตั้งแต่ตอนคอมไพล์
- ตัวอย่าง: Method Overloading, Constructor Overloading

2. Runtime Polymorphism

- หรือ Dynamic Polymorphism
- เกิดขึ้นในขณะที่โปรแกรมกำลังทำงาน
- Java ตัดสินใจว่าจะเรียกเมธอดตัวไหนตอนรันโปรแกรม
- ตัวอย่าง: Method Overriding

Compile-time Polymorphism : Method Overloading

- คือการมีเมธอดหลายตัวในคลาสเดียวกันที่มีชื่อเดียวกัน แต่มีพารามิเตอร์ (parameter list) ที่แตกต่างกัน
  - จำนวนพารามิเตอร์ไม่เท่ากัน
  - ชนิดข้อมูลของพารามิเตอร์ไม่เหมือนกัน
  - ลำดับของชนิดข้อมูลของพารามิเตอร์ไม่เหมือนกัน
- Method Overloading จะเลือกใช้งานเมธอดที่ถูกต้องจากการตรวจสอบที่ชนิดข้อมูลและจำนวนของพารามิเตอร์ที่ส่งเข้ามาในขณะที่คอมไพล์

ตัวอย่างที่ 1 Method Overloading

```java
public class Calculator {
    // Method 1: บวกเลขจำนวนเต็ม 2 ตัว
    public int add(int a, int b) {
        return a + b;
    }

    // Method 2: บวกเลขจำนวนเต็ม 3 ตัว
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method 3: บวกเลขทศนิยม 2 ตัว
    public double add(double a, double b) {
        return a + b;
    }

    // Method 4: บวก String 2 ตัว
    public String add(String a, String b) {
        return a + b;
    }
}
```

ตัวอย่างที่ 1 การใช้งาน Method Overloading

```java
public class TestCalculator {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        // เรียกใช้ Method แต่ละแบบ
        System.out.println(calc.add(5, 10));           // 15
        System.out.println(calc.add(5, 10, 15));       // 30
        System.out.println(calc.add(5.5, 10.3));       // 15.8
        System.out.println(calc.add("Hello ", "World")); // Hello World

        // Compiler จะเลือก Method ที่เหมาะสมโดยอัตโนมัติ
    }
}
```

ตัวอย่างที่ 2 Method Overloading

```
public class Shape {
    // วาดรูปสี่เหลี่ยมผืนผ้า
    public void draw(int width, int height) {
        System.out.println("Drawing rectangle: " + width + "x" + height);
    }

    // วาดรูปวงกลม
    public void draw(double radius) {
        System.out.println("Drawing circle with radius: " + radius);
    }

    // วาดรูปสามเหลี่ยม
    public void draw(int base, int height, String type) {
        System.out.println("Drawing " + type + " triangle: base=" +
                base + ", height=" + height);
    }
}
```

ตัวอย่างที่ 3 Constructor Overloading

```java
public class Student {
    private String name;
    private int age;
    private String major;

    // Constructor 1: ไม่มี Parameter
    public Student() {
        this.name = "Unknown";
        this.age = 0;
        this.major = "Undeclared";
    }

    // Constructor 2: รับชื่อเท่านั้น
    public Student(String name) {
        this.name = name;
        this.age = 0;
        this.major = "Undeclared";
    }
```

ตัวอย่างที่ 3 Constructor Overloading (ต่อ)

```
    // Constructor 3: รับชื่อและอายุ
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
        this.major = "Undeclared";
    }

    // Constructor 4: รับข้อมูลครบ
    public Student(String name, int age, String major) {
        this.name = name;
        this.age = age;
        this.major = major;
    }
}
```

Runtime Polymorphism : Method Overriding

- คือการเขียน Method ใหม่คลาสลูก (Subclass) โดยเมธอดต้องมี Signature เหมือนกับเมธอดในคลาสแม่ (Superclass) ทุกประการ (ทั้งชื่อเมธอด, จำนวนและชนิดของพารามิเตอร์, และประเภทการคืนค่า)
- เพื่อให้คลาสลูกสามารถปรับปรุงหรือเปลี่ยนแปลงการทำงานของเมธอดที่สืบทอดมาจากคลาสแม่ได้ตามความต้องการ
- การตัดสินใจว่าจะเรียกเมธอดของคลาสแม่หรือคลาสลูกจะเกิดขึ้นในขณะที่โปรแกรมกำลังทำงาน

เงื่อนไขของ Method Overriding

1. เมธอดต้องอยู่ในคลาสลูก และเมธอดต้นฉบับต้องอยู่ในคลาสแม่
2. ชื่อเมธอดต้องเหมือนกันทุกประการ
3. จำนวนและชนิดของพารามิเตอร์ต้องเหมือนกันทุกประการ
4. ประเภทการคืนค่า (Return type) ต้องเหมือนกัน
5. ระดับการเข้าถึง (Access modifier) ของเมธอดในคลาสลูกต้องไม่แคบกว่าคลาสแม่ (เช่น ถ้าในคลาสแม่เป็น protected, ในคลาสลูกจะเป็น public ได้ แต่เป็น private ไม่ได้)
6. ไม่สามารถ override method ที่เป็น final, static, หรือ private
7. นิยมใช้ @Override annotation เพื่อบอกคอมไพเลอร์ว่าเราตั้งใจจะoverride method

ตัวอย่างที่ 4 Method Overriding

```java
public class Animal {
    public void makeSound() {
        System.out.println("The animal makes a sound");
    }
}

// Subclass
class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Bok! Bok!");
    }
}

// Subclass
class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow! Meow!");
    }
}
```

คลาส Dog เป็น Subclass ที่สืบทอดมาจากคลาส Animal

คลาส Cat เป็น Subclass ที่สืบทอดมาจากคลาส Animal

คลาส Dog และ Cat สืบทอดมาจาก Animal และโอเวอร์ไรด์เมธอด makeSound() ด้วยเสียงเฉพาะของตัวเอง

Runtime Polymorphism และการอ้างอิงวัตถุ

- กุญแจสำคัญของ Runtime Polymorphism คือ "การอ้างอิงของคลาสแม่ (Superclass reference) ไปยังอ็อบเจ็กต์ของคลาสลูก (Subclass object)“
- รูปแบบ

```
Superclass objectReference = new Subclass();
```

- เมื่อเรียกใช้เมธอดที่ถูกโอเวอร์ไรด์ผ่าน objectReference จาวาจะตรวจสอบประเภทของอ็อบเจ็กต์จริงๆ (ตอน Runtime) และเรียกใช้เมธอดของคลาสลูกนั้น

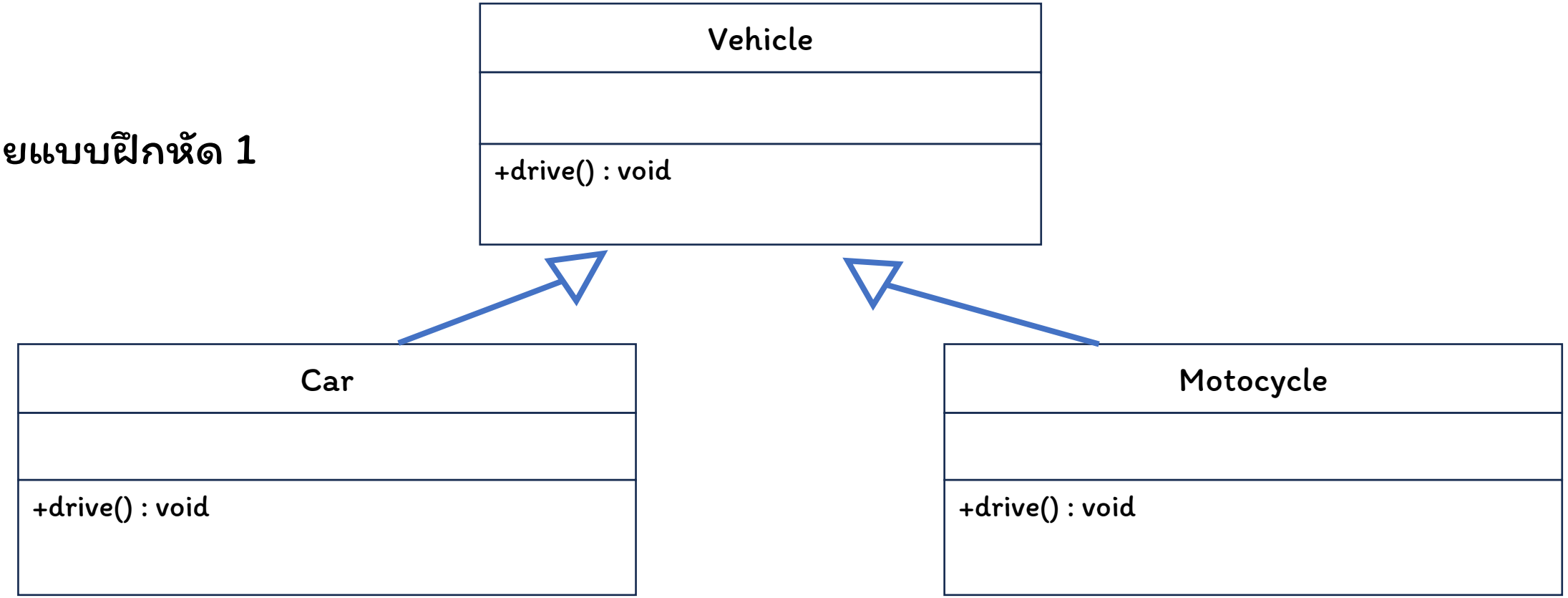
ตัวอย่างที่ 5 การใช้งาน Method Overriding

```java
public class TestAnimal {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Dog dog = new Dog();
        Cat cat = new Cat();

        System.out.println("=== Animal ===");
        animal.makeSound();  // The animal makes a sound

        System.out.println("\n=== Dog ===");
        dog.makeSound();     // The dog barks: Bok! Bok!

        System.out.println("\n=== Cat ===");
        cat.makeSound();     // The dog barks: Meow! Meow!

        // Runtime Polymorphism
        System.out.println("\n=== Polymorphism ===");
        Animal myDog = new Dog();
        Animal myCat = new Cat();
        myDog.makeSound();   // เรียกเมธอดของ Dog
        myCat.makeSound();   // รียกเมธอดของ Cat
    }
}
```

ประโยชน์ของ Polymorphism

1. **ลดความซับซ้อนของโค้ด** ไม่ต้องใช้คำสั่ง if-else หรือ switch เพื่อตรวจสอบประเภทของวัตถุ
2. **เพิ่มความยืดหยุ่น (Flexibility)** สามารถเพิ่มคลาสลูกใหม่ได้ง่ายๆ โดยไม่ต้องแก้ไขโค้ดเดิมที่เรียกใช้งาน
3. **บำรุงรักษาง่าย (Maintainability)** การเปลี่ยนแปลงการทำงานในคลาสลูกจะไม่กระทบกับส่วนอื่นๆ ของโค้ด
4. **เพิ่มความสามารถในการขยายระบบ (Extensibility)** สามารถสร้างคลาสลูกใหม่ที่สืบทอดพฤติกรรมจากคลาสแม่ได้ง่าย

เฉลยแบบฝึกหัด 1



ตัวอย่างที่ 6 Method Overriding

```
public class Vehicle {
    public void drive() {
        System.out.println("The vehicle is driving.");
    }
}

class Car extends Vehicle {
    @Override
    public void drive() {
        System.out.println("The car is driving on the road.");
    }
}

class Motorcycle extends Vehicle {
    @Override
    public void drive() {
        System.out.println("The motorcycle is riding on two wheels.");
    }
}
```

คลาส Dog และ Cat สืบทอดมาจาก Animal และโอเวอร์ไรด์เมธอด makeSound() ด้วยเสียงเฉพาะของตัวเอง

ตัวอย่างที่ 7 การใช้งาน Method Overriding

```
public class TestVehicle {
    public static void main(String[] args) {
        Vehicle myVehicle = new Vehicle();
        Vehicle myCar = new Car();
        Vehicle myMotorcycle = new Motorcycle();

        myVehicle.drive();
        myCar.drive();
        myMotorcycle.drive();
    }
}
```

ตัวอย่างที่ 8 การปรับการใช้งานแบบ Polymorphic

```
public class TestVehicle {
    // เมธอดที่รับอ็อบเจ็กต์ชนิด Vehicle
    public static void startEngine(Vehicle v) {
        v.drive(); // เรียกเมธอด drive() ของวัตถุที่ส่งเข้ามา
    }

    public static void main(String[] args) {
        Car car = new Car();
        Motorcycle motorcycle = new Motorcycle();

        startEngine(car);          // ส่งอ็อบเจ็กต์ Car
        startEngine(motorcycle);   // ส่งอ็อบเจ็กต์ Motorcycle
    }
}
```

เมธอด startEngine ไม่ต้องสนใจว่าวัตถุที่ส่งเข้ามาเป็น Car หรือ Motorcycle ขอแค่เป็น Vehicle หรือคลาสลูกของ Vehicle ก็พอ

END.

Q & A