บทที่ 5

การสืบทอด

Inheritance

สอนโดย อ.เรวดี พิพัฒน์สูงเนิน สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏอุดรธานี

หลักการเขียนโปรแกรมเชิงวัตถุ

วัตถุประสงค์การเรียนรู้

- สามารถอธิบายความหมายและประโยชน์ของการสืบทอดได้
- สามารถระบุความสัมพันธ์แบบ "is-a" สำหรับการสืบทอดได้
- สามารถใช้คีย์เวิร์ด extends เพื่อสร้างคลาสลูกได้
- สามารถเรียกใช้ Constructor ของคลาสแม่ด้วย super() ได้
- สามารถ Overriding เมธอดในคลาสลูกได้
- สามารถเข้าใจและใช้คีย์เวิร์ด super ในการเข้าถึงเมธอดของคลาสแม่ได้
- สามารถเข้าใจข้อจำกัดและประโยชน์ของ final คีย์เวิร์ดกับการสืบทอดได้

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)

Inheritance คืออะไร?

- **Inheritance (การสืบทอด)** คือ กลไกที่ทำให้คลาสใหม่ (คลาสลูก หรือ Subclass) สามารถรับคุณสมบัติ (Attributes) และพฤติกรรม (Methods) ของคลาสเดิม (คลาสแม่ หรือ Superclass) มาใช้งานได้

- เป็นความสัมพันธ์แบบ “is-a” (เป็น) เช่น
  - รถยนต์ **เป็น** ยานพาหนะ
  - สุนัข **เป็น** สัตว์

ประโยชน์ของ Inheritance

1. ลดการเขียนโค้ดซ้ำ ส่งเสริมการนำกลับมาใช้ซ้ำ(Code Reusability)

- เมื่อหลายๆ คลาสมีคุณสมบัติและพฤติกรรมที่เหมือนกัน แทนที่จะเขียนโค้ดซ้ำๆ เราสามารถสร้างคลาสแม่ขึ้นมาเก็บสิ่งเหล่านั้นไว้
- คลาสลูกสามารถ "สืบทอด" คุณสมบัติและพฤติกรรมเหล่านั้นมาใช้ได้ทันที

2. บำรุงรักษาง่ายขึ้น(Maintainability)

- หากมีการแก้ไขโค้ดที่ใช้ร่วมกันในคลาสแม่ จะส่งผลต่อคลาสลูกทุกคลาสที่สืบทอดไป

3. ง่ายต่อการขยาย(Extensibility)

- สามารถเพิ่มคลาสลูกใหม่ๆ ได้โดยไม่ต้องแก้ไขคลาสแม่เดิม เพิ่มความสามารถใหม่ๆ ได้อย่างเป็นระบบ

คำศัพท์ที่สำคัญ

- Parent Class / Super Class / Base Class: คลาสแม่ที่ให้คลาสอื่นสืบทอด
- Child Class / Sub Class / Derived Class: คลาสลูกที่สืบทอดจากคลาสแม่
- extends: คำสั่งที่ใช้ในการสืบทอด
- super: คำสั่งที่ใช้เรียกใช้งานจากคลาสแม่
- Override: การเขียนเมธอดใหม่ทับเมธอดเดิมในคลาสแม่

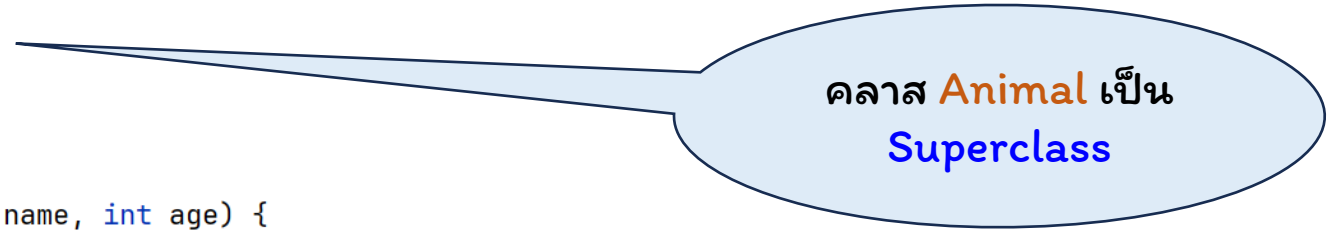
ไวยกรณ์(syntax) พื้นฐานของ Inheritance

```
class Parent {
    // fields และ methods
}

class Child extends Parent {
    // เพิ่มหรือเปลี่ยนพฤติกรรมได้
}
```

- คลาส Parent คือ Superclass
- คลาส Child คือ Subclass
- extends คือ **คีย์เวิร์ด** ที่ใช้ในการระบุว่าคลาสใดสืบทอดมาจากคลาสใด

ตัวอย่างโค้ด Inheritance

```java
//คลาสแม่ (Superclass)
public class Animal {
    String name;
    int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
        System.out.println("Animal created: " + name);
    }

    public void eat() {
        System.out.println(name + " is eating.");
    }

    public void sleep() {
        System.out.println(name + " is sleeping.");
    }

    public void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```



```java
//คลาสลูก (Subclass)
public class Dog extends Animal{
    String breed;

    public Dog(String name, int age, String breed) {
        // ต้องเรียกใช้ constructor ของคลาสแม่ (Animal) ก่อน
        super(name, age);
        this.breed = breed;
        System.out.println("Dog created: " + name + " (" + breed + ")");
    }

    public void bark() {
        System.out.println(name + " says Bok Bok!");
    }

    Rename usages
    // เมธอดเฉพาะของ Dog
    public void waggingTail() {
        System.out.println(name + " is wagging tail.....");
    }

    // Overriding displayInfo method
    @Override
    public void displayInfo() {
        super.displayInfo(); // เรียกใช้ displayInfo ของ Animal ก่อน
        System.out.println("Breed: " + breed); // แล้วเพิ่มข้อมูลเฉพาะของ Dog
    }
}
```

คลาส Dog เป็น Subclass ที่สืบทอดมาจากคลาส Animal

ข้อควรรู้เกี่ยวกับ Inheritance ใน Java

- Java ไม่รองรับ Multiple Inheritance คลาสลูกสามารถสืบทอดจากคลาสแม่ได้เพียงคลาสเดียวเท่านั้น

```
class A {}
class B {}
class C extends A, B {} // ❌ error
```

- **แต่สามารถสืบทอดได้หลายระดับ** เช่น Class C extends Class B, Class B extends Class A

```
class Animal {}
class Mammal extends Animal {}
class Dog extends Mammal {}
```

- **คลาสแม่สามารถสืบทอดไปยังคลาสลูกได้หลายคลาส**

```
class Animal {}
class Dog extends Animal {}
class Cat extends Animal {}
```

ข้อควรรู้เกี่ยวกับ Inheritance ใน Java

- **Constructor ไม่ถูกสืบทอด** คลาสลูกจะต้องเรียกใช้ Constructor ของคลาสแม่เสมอ

```java
public class Dog extends Animal{
    String breed;

    public Dog(String name, int age, String breed) {
        // ต้องเรียกใช้ constructor ของคลาสแม่ (Animal) ก่อน
        super(name, age);
        this.breed = breed;
        System.out.println("Dog created: " + name + " (" + breed + ")");
    }
```

- **private members ไม่สามารถสืบทอดได้โดยตรง** แต่สามารถเข้าถึงผ่าน public หรือ protected methods ของคลาสแม่ได้ (จากหลัก Encapsulation)

Access Modifiers และการสืบทอด

- ระดับการเข้าถึงในการสืบทอด
  - public: เข้าถึงได้จากทุกที่ รวมถึงคลาสลูก
  - protected: เข้าถึงได้จากคลาสลูกและคลาสใน package เดียวกัน
  - default (package-private): เข้าถึงได้จากคลาสใน package เดียวกันเท่านั้น
  - private: เข้าถึงไม่ได้จากคลาสลูก

```java
class Parent {
    public int publicVar = 1;
    protected int protectedVar = 2;
    int defaultVar = 3;
    private int privateVar = 4;
}

class Child extends Parent {
    public void testAccess() {
        System.out.println(publicVar);    // OK
        System.out.println(protectedVar); // OK
        System.out.println(defaultVar);   // OK (same package)
        // System.out.println(privateVar); // Error!
    }
}
```

ตารางสรุป Access Modifiers

Access Modifier	Same Class	Same Package	Subclass (diff. package)	Anywhere (diff. package)
private (-)	✓	✗	✗	✗
(default)	✓	✓	✗	✗
protected (#)	✓	✓	✓	✗
public (+)	✓	✓	✓	✓

คำสั่ง super()

การใช้งาน super keyword

- **เรียก constructor ของคลาสแม่**: super(parameters)
- **เรียกใช้เมธอดของคลาสแม่**: super.methodName()
- **เรียกใช้ attribute ของคลาสแม่**: super.variableName

ตัวอย่างการใช้คำสั่ง super()

```java
class Vehicle {
    protected String brand;

    public Vehicle(String brand) {
        this.brand = brand;
    }

    public void start() {
        System.out.println(brand + " vehicle is starting");
    }
}
```

```java
class Car extends Vehicle {
    private int doors;

    public Car(String brand, int doors) {
        super(brand); // เรียก constructor ของคลาสแม่
        this.doors = doors;
    }

    @Override
    public void start() {
        super.start(); // เรียกเมธอดของคลาสแม่
        System.out.println("Car with " + doors + " doors is ready");
    }
}
```

การเรียกใช้ super() ใน constructor ของ Subclass

- super() ใช้สำหรับเรียก Constructor ของคลาสแม่
- **กฎสำคัญ** ต้องเป็นคำสั่งแรกสุดใน Constructor ของคลาสลูกเสมอ (หากไม่ระบุ Java จะพยายามเรียก super() แบบไม่มีพารามิเตอร์ให้เอง)
- **ทำไมต้องเรียก?** เพื่อให้แน่ใจว่าส่วนประกอบของคลาสแม่ได้รับการเริ่มต้นอย่างถูกต้องก่อนที่คลาสลูกจะเริ่มต้นส่วนประกอบของตัวเอง

```java
public class Dog extends Animal{
    String breed;

    public Dog(String name, int age, String breed) {
        // ต้องเรียกใช้ constructor ของคลาสแม่ (Animal) ก่อน
        super(name, age);
        this.breed = breed;
        System.out.println("Dog created: " + name + " (" + breed + ")");
    }
```

ตัวอย่างโค้ดการใช้งานคลาสที่สืบทอด

```
public class TestInheritance {
    public static void main(String[] args) {
        Animal myAnimal = new Animal("Generic Animal", 5);
        myAnimal.displayInfo();
        myAnimal.eat();
        myAnimal.sleep();
        System.out.println("-----------------------------------");

        Dog myDog = new Dog("Soba", 3, "Shih Tzu");
        myDog.displayInfo(); // จะเรียก displayInfo() ที่ถูก Overriding ใน Dog
        myDog.eat();         // สืบทอดมาจาก Animal
        myDog.sleep();       // สืบทอดมาจาก Animal
        myDog.bark();        // เมธอดเฉพาะของ Dog
        myDog.waggingTail(); // เมธอดเฉพาะของ Dog
    }
}
```

ตัวอย่างผลลัพธ์การใช้งานคลาสที่สืบทอด

```java
public class TestInheritance {
    public static void main(String[] args) {
        Animal myAnimal = new Animal("Generic Animal", 5);
        myAnimal.displayInfo();
        myAnimal.eat();
        myAnimal.sleep();
        System.out.println("-----------------------------------");

        Dog myDog = new Dog("Soba", 3, "Shih Tzu");
        myDog.displayInfo(); // เรียก displayInfo() ที่ถูก Overriding ใน Dog
        myDog.eat();         // สืบทอดมาจาก Animal
        myDog.sleep();       // สืบทอดมาจาก Animal
        myDog.bark();        // เมธอดเฉพาะของ Dog
        myDog.waggingTail(); // เมธอดเฉพาะของ Dog
    }
}
```

```
Animal created: Generic Animal
Name: Generic Animal, Age: 5
Generic Animal is eating.
Generic Animal is sleeping.
-----------------------------------
Animal created: Soba
Dog created: Soba (Shih Tzu)
Name: Soba, Age: 3
Breed: Shih Tzu
Soba is eating.
Soba is sleeping.
Soba says Bok Bok!
Soba is wagging tail.....
```

Method overriding คืออะไร?

- **Method Overriding** คือ การที่คลาสลูกมีเมธอดที่มี ชื่อ, ประเภทพารามิเตอร์, และประเภทค่าคืน เหมือนกับเมธอดในคลาสแม่ทุกประการ
- **วัตถุประสงค์** เพื่อให้คลาสลูกสามารถมีพฤติกรรมที่แตกต่างไปจากคลาสแม่สำหรับเมธอดนั้นๆ
- **กฎของ Overriding**
  - method ต้องมีชื่อเดียวกัน
  - parameter ต้องเหมือนกัน
  - access modifier ต้องเท่าหรือกว้างขึ้น

ตัวอย่างโค้ด Method Overriding (ใน Dog คลาส)

```java
//คลาสแม่ (Superclass)
public class Animal {
    String name;
    int age;



    public void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

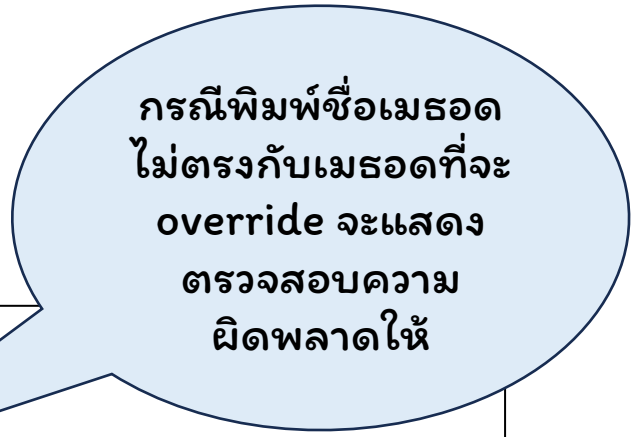
```java
//คลาสลูก (Subclass)
public class Dog extends Animal{
    String breed;


    // Overriding displayInfo method
    @Override
    public void displayInfo() {
        super.displayInfo(); // เรียกใช้ displayInfo ของ Animal ก่อน
        System.out.println("Breed: " + breed); // แล้วเพิ่มข้อมูลเฉพาะของ Dog
    }
}
```

การใช้ @Override annotation

- ช่วยตรวจสอบความถูกต้องในการ override
- ลดความผิดพลาดจากชื่อผิด / parameter ไม่ตรง

```java
public class Animal {

    public void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

```java
public class Dog extends Animal{
    // Overriding displayInfo method
    @Override
    public void DisplayInfo() {
        super.displayInfo(); // เรียกใช้ displayInfo ของ Animal ก่อน
        System.out.println("Breed: " + breed); // แล้วเพิ่มข้อมูลเฉพาะของ Dog
    }
}
```



final คีย์เวิร์ดกับการสืบทอด

- **final class :** คลาสที่ถูกประกาศเป็น final ไม่สามารถถูกสืบทอดได้

```
public final class MyFinalClass {
    // ...
}
// class ChildClass extends MyFinalClass { } // Error!
```

- **final method :** เมธอดที่ถูกประกาศเป็น final ไม่สามารถถูก Overriding ได้

```
public class Parent {
    public final void cannotOverride() {
        System.out.println("This method cannot be overridden.");
    }
}
// class Child extends Parent {
//     @Override
//     public void cannotOverride() { // Error!
//         // ...
//     }
// }
```

ทำไมต้องใช้ final กับ class/method

- final class
  - เพื่อป้องกันการสืบทอดที่ไม่พึงประสงค์ (Security, Performance Optimization)
  - เมื่อต้องการให้คลาสมีพฤติกรรมคงที่ ไม่ต้องการให้ใครมาเปลี่ยนแปลง
- final method
  - เพื่อป้องกันการเปลี่ยนแปลงพฤติกรรมหลักของเมธอดนั้นๆ ในคลาสลูกเมื่อเมธอดมีการทำงานที่สำคัญและเป็นแกนหลัก ไม่ควรถูกแก้ไข

ลำดับชั้นการสืบทอด (Inheritance hierarchy)

- **คลาสสามารถสืบทอดกันเป็นลำดับชั้นได้**
- **Object Class:** คลาส java.lang.Object เป็น Superclass ของทุกคลาสใน Java (แม้ไม่ระบุ extends ก็จะสืบทอดโดยปริยาย)
- **ตัวอย่างลำดับชั้น: Animal -> Dog -> Labrador**
  - **Labrador เป็น Dog และ Dog เป็น Animal**

ตัวอย่างโค้ดการใช้งานคลาสที่สืบทอดแบบลำดับชั้น

```java
public class Labrador extends Dog{
    String color;

    public Labrador(String name, int age, String color) {
        // เรียก Constructor ของ Dog (ซึ่งจะเรียก Constructor ของ Animal ต่อไป)
        super(name, age, "Labrador"); // กำหนด breed เป็น Labrador โดยตรง
        this.color = color;
        System.out.println("Labrador created: " + name + " (" + color + ")");
    }

    // Labrador มีพฤติกรรมเฉพาะเพิ่มเติม
    public void retrieve() {
        System.out.println(name + " is retrieving.");
    }

    @Override
    public void displayInfo() {
        super.displayInfo(); // เรียก displayInfo ของ Dog
        System.out.println("Color: " + color); // เพิ่มข้อมูลเฉพาะของ Labrador
    }
}
```

ตัวอย่างโค้ดการใช้งาน Object ในลำดับชั้น

```java
public class PetStore {
    public static void main(String[] args) {
        Animal myAnimal = new Animal("Kitty", 2);
        myAnimal.displayInfo();
        System.out.println("---------------------------------");

        Dog myDog = new Dog("Max", 5, "German Shepherd");
        myDog.displayInfo();
        myDog.bark();
        System.out.println("+++++++++++++++++++++++++++++++++");

        Labrador myLab = new Labrador("Charlie", 1, "Yellow");
        myLab.displayInfo(); // จะเรียก displayInfo ของ Labrador -> Dog -> Animal
        myLab.bark();        // สืบทอดมาจาก Dog
        myLab.eat();         // สืบทอดมาจาก Animal
        myLab.retrieve();    // เมธอดเฉพาะของ Labrador
    }
}
```

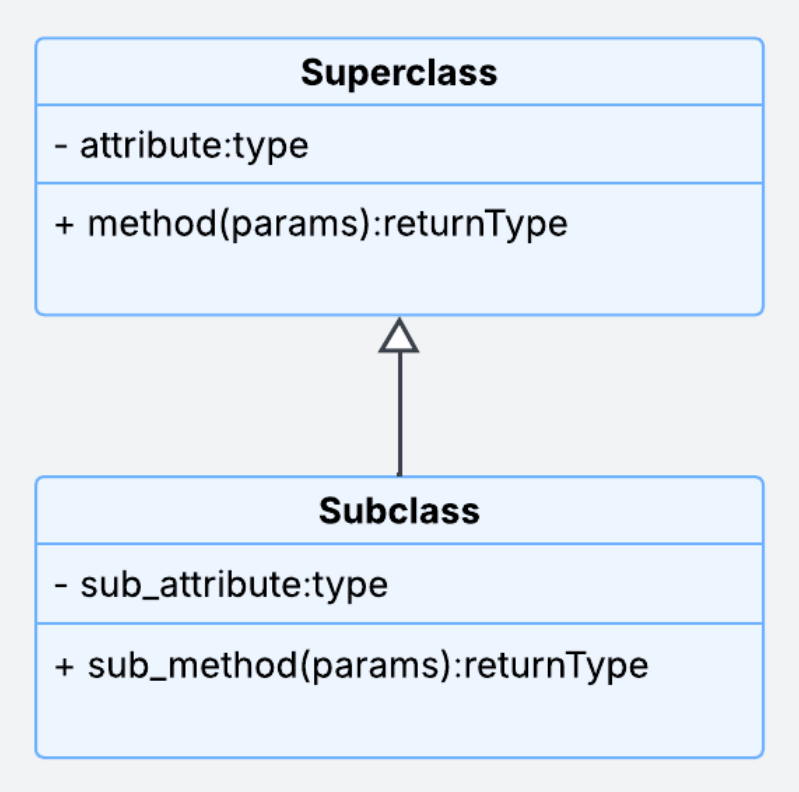
ตัวอย่างผลลัพธ์การใช้งานคลาสที่สืบทอด

```java
public class PetStore {
    public static void main(String[] args) {
        Animal myAnimal = new Animal("Kitty", 2);
        myAnimal.displayInfo();
        System.out.println("---------------------------------");

        Dog myDog = new Dog("Max", 5, "German Shepherd");
        myDog.displayInfo();
        myDog.bark();
        System.out.println("+++++++++++++++++++++++++++++++++");

        Labrador myLab = new Labrador("Charlie", 1, "Yellow");
        myLab.displayInfo(); // จะเรียก displayInfo ของ Labrador -> Dog -> Animal
        myLab.bark();        // สืบทอดมาจาก Dog
        myLab.eat();         // สืบทอดมาจาก Animal
        myLab.retrieve();    // เมธอดเฉพาะของ Labrador
    }
}
```

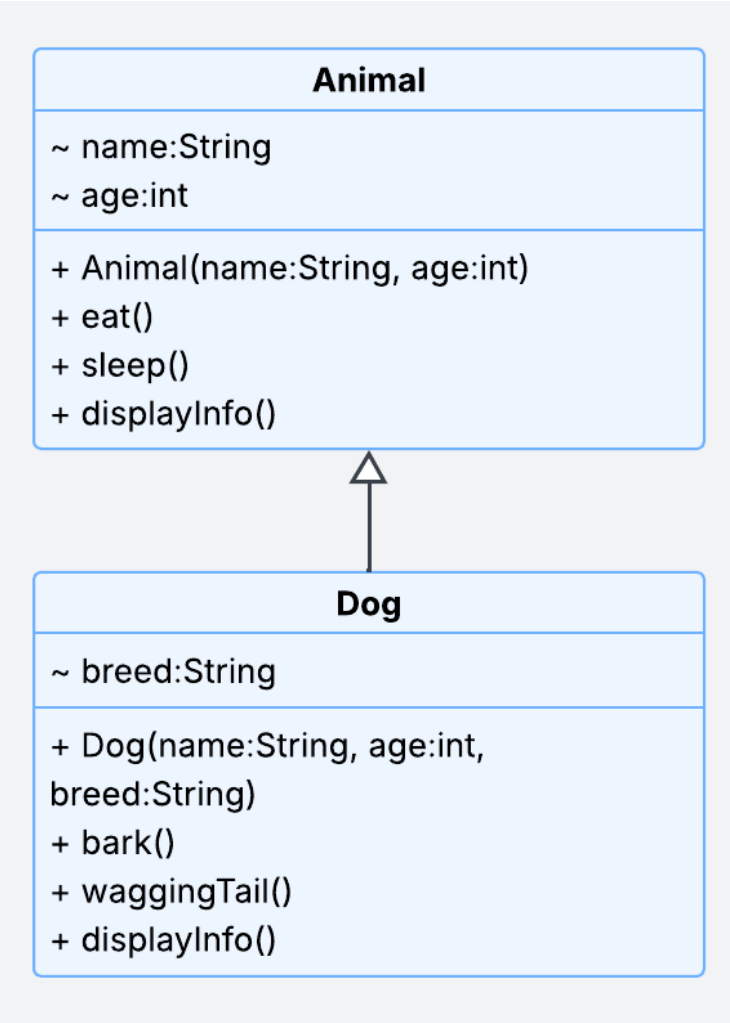
```
Animal created: Kitty
Name: Kitty, Age: 2
---------------------------------
Animal created: Max
Dog created: Max (German Shepherd)
Name: Max, Age: 5
Breed: German Shepherd
Max says Bok Bok!
+++++++++++++++++++++++++++++++++
Animal created: Charlie
Dog created: Charlie (Labrador)
Labrador created: Charlie (Yellow)
Name: Charlie, Age: 1
Breed: Labrador
Color: Yellow
Charlie says Bok Bok!
Charlie is eating.
Charlie is retrieving.
```

UML Class Diagram สำหรับ Inheritance

รูปแบบ



ตัวอย่าง

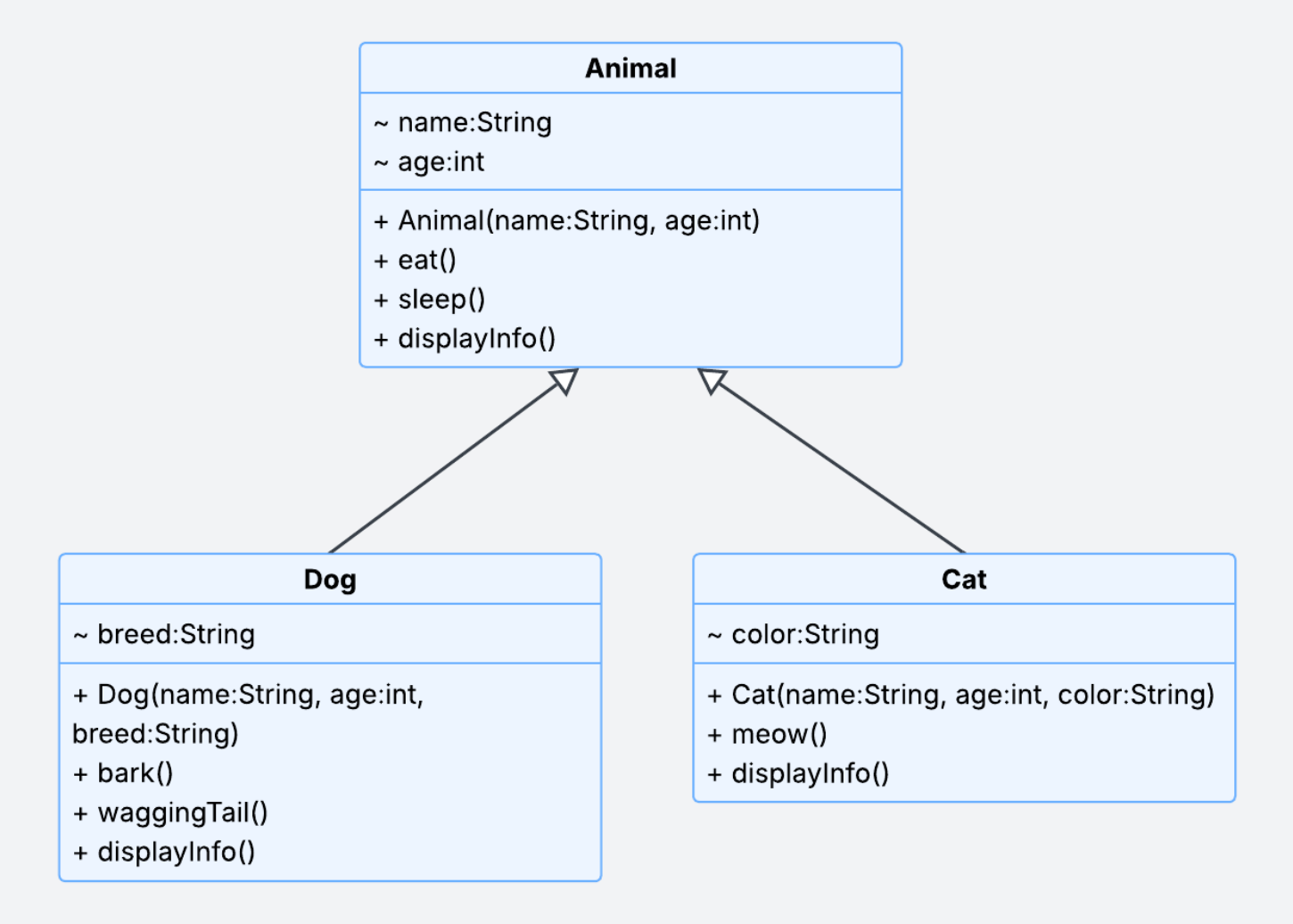


UML Class Diagram สำหรับ Inheritance

- สัญลักษณ์ระดับการเข้าถึง

สัญลักษณ์	ความหมาย	Java Keyword
+	Public	`public`
-	Private	`private`
#	Protected	`protected`
~	Package	(default)

UML Class Diagram สำหรับ Inheritance



สรุป Inheritance

- เป็นกลไกที่ทำให้คลาสลูกสามารถนำคุณสมบัติและพฤติกรรมของคลาสแม่มาใช้ได้
- ใช้ extends คีย์เวิร์ดเพื่อสร้างความสัมพันธ์
- ประโยชน์หลักคือการนำโค้ดกลับมาใช้ใหม่, ลดความซ้ำซ้อน, และเพิ่มความสามารถในการบำรุงรักษาและขยายระบบ
- ต้องเรียกใช้ Constructor ของคลาสแม่ด้วย super()
- สามารถ Overriding เมธอดของคลาสแม่ได้ เพื่อให้มีพฤติกรรมที่แตกต่างไป
- final คีย์เวิร์ดใช้ป้องกันการสืบทอดหรือ Overriding

แบบฝึกหัดที่ 1 : ออกแบบ Class ระบบการทำงาน

- จงวาดแผนภาพ Class diagram ของระบบการทำงานโดยมีส่วนประกอบต่อไปนี้
- สร้างคลาส Employee ที่มีคุณสมบัติดังนี้ (ทุก attribute ต้องเป็น protected)
  - ชื่อพนักงาน(name)
  - รหัสพนักงาน(id)
  - เงินเดือน(baseSalary)
- คอนสตรักเตอร์รับพารามิเตอร์เพื่อกำหนดค่าให้ทุก attribute ในการสร้าง object
- เมธอด displayInfo() : ทำหน้าที่แสดงข้อมูลพนักงาน ในรูปแบบ
  - “Name : *ชื่อพนักงาน*”
  - “ID : *รหัสพนักงาน*”
  - “Base salary : *เงินเดือน*”
- เมธอด calculateSalary() : ส่งค่ากลับเป็นเงินเดือนของพนักงาน
- เมธอด work() : แสดงข้อความ “*ชื่อพนักงาน* is working.”

แบบฝึกหัดที่ 1 : ออกแบบ Class ระบบการทำงาน

- สร้างคลาส Manager เป็นคลาสที่สืบทอดจาก Employee ที่มีคุณสมบัติดังนี้ (ทุก attribute ต้องเป็น private)
  - โบนัส(bonus)
  - จำนวนพนักงานในทีม(teamSize)
- คอนสตรักเตอร์รับพารามิเตอร์เพื่อกำหนดค่าให้ทุก attribute ในการสร้าง object
- เมธอด displayInfo() : ทำหน้าที่แสดงข้อมูลพนักงาน ในรูปแบบ
  - “Name : *ชื่อพนักงาน*”
  - “ID : *รหัสพนักงาน*”
  - “Base salary : *เงินเดือน*”
  - “Bonus : *โบนัส*”
  - “Team size : *จำนวนพนักงานในทีม*”
- เมธอด calculateSalary() : ส่งค่ากลับเป็นเงินเดือนของพนักงาน + โบนัส
- เมธอด work() : แสดงข้อความ “*ชื่อพนักงาน* is also managing the team.”
- เมธอด manageTeam() : แสดงข้อความ “*ชื่อพนักงาน* is managing a team of *จำนวนพนักงานในทีม* people.”

แบบฝึกหัดที่ 1 : ออกแบบ Class ระบบการทำงาน

- สร้างคลาส Developer เป็นคลาสที่สืบทอดจาก Employee ที่มีคุณสมบัติดังนี้ (ทุก attribute ต้องเป็น private)
  - ชื่อภาษาการเขียนโปรแกรมที่ถนัด (programmingLanguage)
  - ชื่อโครงการที่ทำงาน(projects[])
- คอนสตรักเตอร์รับพารามิเตอร์เพื่อกำหนดค่าให้ทุก attribute ในการสร้าง object
- เมธอด displayInfo() : ทำหน้าที่แสดงข้อมูลพนักงาน ในรูปแบบ
  - “Name : *ชื่อพนักงาน*”
  - “ID : *รหัสพนักงาน*”
  - “Base salary : *เงินเดือน*”
  - “Programming language : *ชื่อภาษาการเขียนโปรแกรมที่ถนัด*”
  - “Projects : *จำนวนโครงการที่ทำงาน*”
- เมธอด calculateSalary() : ส่งค่ากลับเป็นเงินเดือนของพนักงาน + (จำนวนโครงการที่ทำงาน) * 10000
- เมธอด work() : แสดงข้อความ “*ชื่อพนักงาน* is developing software.”
- เมธอด code() : แสดงข้อความ “*ชื่อพนักงาน* is coding in *ชื่อภาษาการเขียนโปรแกรมที่ถนัด*.”

แบบฝึกหัดที่ 2 : เขียนโปรแกรมสร้างคลาส

- จาก class diagram ที่ได้ออกแบบในแบบฝึกหัดที่ 1 ให้เขียนโค้ดภาษาจาวาสร้างคลาส Employee, Manager และ Developer ตามเงื่อนไขที่กำหนดของแต่ละคลาส

แบบฝึกหัดที่ 3 : เขียนโปรแกรมทดสอบการทำงาน

- ให้เขียนโปรแกรมสร้างคลาสเพื่อทดสอบการทำงานของคลาสที่ออกแบบและสร้างขึ้นทั้ง 3 คลาส โดยมีเงื่อนไขการทำงานดังนี้
- สร้าง object ของคลาส Manager
- แสดงข้อความ “+++++ Manager Information +++++”
- แสดงข้อมูลโดยใช้เมธอด displayInfo()
- แสดงข้อความ “Total salary : *แสดงค่าเงินเดือนของmanagerโดยใช้เมธอด calculateSalary()*”
- แสดงข้อมูลการทำงานโดยใช้เมธอด work()
- แสดงข้อมูลจำนวนทีมงานโดยใช้เมธอด manageTeam()

แบบฝึกหัดที่ 3 : เขียนโปรแกรมทดสอบการทำงาน

- **(ต่อ)**
- แสดงข้อความ “+++++ Developer Information +++++”
- ประกาศและกำหนดค่าให้ตัวแปร projects[] ดังนี้ {“E-commerce”, “Mobile Application”, “AI Systems”}
- สร้าง Object ของคลาส Developer
- แสดงข้อมูลโดยใช้เมธอด displayInfo()
- แสดงข้อความ “Total salary : *แสดงค่าเงินเดือนของdeveloperโดยใช้เมธอด calculateSalary()*”
- แสดงข้อมูลการทำงานโดยใช้เมธอด work()
- แสดงข้อมูลภาษาการเขียนโปรแกรมที่ถนัดโดยใช้เมธอด code()

แบบฝึกหัดที่ 3 : เขียนโปรแกรมทดสอบการทำงาน

- ตัวอย่างผลลัพธ์

```
+++++ Manager Information +++++
Name: Kunawut Boonkhwang
ID: 1001
Base Salary: 75000.0
Bonus: 15000.0
Team Size: 5
Total Salary: 90000.0
Kunawut Boonkhwang is working.
Kunawut Boonkhwang is also managing the team.
Kunawut Boonkhwang is managing a team of 5 people.

+++++ Developer Information +++++
Name: Rewadee  Piputsoongnern
ID: 2001
Base Salary: 50000.0
Programming Language: Java
Projects: 3
Total Salary: 80000.0
Rewadee  Piputsoongnern is working.
Rewadee  Piputsoongnern is developing software.
Rewadee  Piputsoongnern is coding in Java.
```

END.

Q & A