บทที่ 7

การซ่อนรายละเอียด

Abstraction

หลักการเขียนโปรแกรมเชิงวัตถุ : การซ่อนรายละเอียด

วัตถุประสงค์การเรียนรู้

- อธิบายแนวคิด Abstraction ได้
- แยกแยะ Abstract Class และ Interface
- เขียน Abstract Class และ Abstract Method
- สร้างและใช้งาน Interface
- ประยุกต์ใช้ Abstraction ในการออกแบบ
- เลือกใช้ Abstract Class หรือ Interface อย่างเหมาะสม

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)**

Abstraction คืออะไร?

- Abstraction หมายถึง การซ่อนรายละเอียดการทำงานภายใน และแสดงเฉพาะสิ่งที่จำเป็น

“แสดง สิ่งสำคัญ ซ่อน รายละเอียดไม่จำเป็น”

- ตัวอย่างในชีวิตจริง
  - **<u>รถยนต์</u>** เราใช้พวงมาลัยและคันเร่ง 🚗 โดยไม่จำเป็นต้องรู้ว่าเครื่องยนต์ทำงานอย่างไร
  - **<u>ตู้ ATM</u>** เรากดปุ่มเพื่อถอนเงิน 🏧 โดยไม่ต้องรู้ว่าระบบฐานข้อมูลธนาคารจัดการข้อมูลอย่างไร

ประโยชน์ของ Abstraction

1. **ลดความซับซ้อน** ซ่อนรายละเอียดที่ไม่จำเป็น
2. **เพิ่มความปลอดภัย** ป้องกันการเข้าถึงข้อมูลโดยตรง
3. **ง่ายต่อการบำรุงรักษา** เปลี่ยนการทำงานภายในได้โดยไม่กระทบโค้ดภายนอก
4. **เพิ่มความยืดหยุ่น** สามารถขยายและปรับปรุงได้ง่าย
5. **การใช้งานซ้ำ** สามารถนำไปใช้ในที่อื่นได้

ความแตกต่างระหว่าง Abstraction vs. Encapsulation

แบ่งออกเป็น 2 ประเภท

1. **Abstraction: เน้นการ ซ่อนรายละเอียด (What to do)** ➡️ ทำให้ผู้ใช้เห็นเฉพาะส่วนที่จำเป็น
2. **Encapsulation: เน้นการ รวมข้อมูลและพฤติกรรมเข้าด้วยกัน** และป้องกันการเข้าถึงที่ไม่เหมาะสม (How to do it) ➡️ ใช้ Access Modifiers (private, public)

ความแตกต่างระหว่าง Abstraction vs. Encapsulation

คุณสมบัติ	Abstraction	Encapsulation
เป้าหมาย	ซ่อนความซับซ้อน	รวมข้อมูลและป้องกันการเข้าถึง
สิ่งที่ซ่อน	การทำงานภายใน	ข้อมูลและพฤติกรรม (state and behavior)
วิธี	Abstract class, Interface	Access modifiers

Encapsulation คือกลไกที่ช่วยให้ Abstraction ทำงานได้

วิธีการทำ Abstraction ใน Java

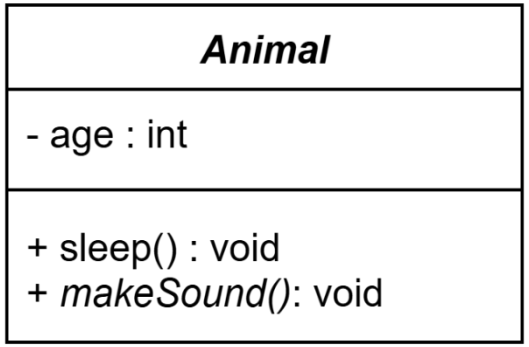
1. Abstract Classes ใช้สำหรับคลาสที่มีความสัมพันธ์แบบ "is-a" (เป็น... ประเภทหนึ่ง) และมีทั้งเมธอดที่มีการใช้งานแล้วและเมธอดที่ยังไม่มีการใช้งาน
2. Interfaces ใช้สำหรับกำหนด "สัญญา" หรือ "blueprint" ของพฤติกรรม โดยไม่มีการใช้งานใด ๆ ในตัวมันเลย

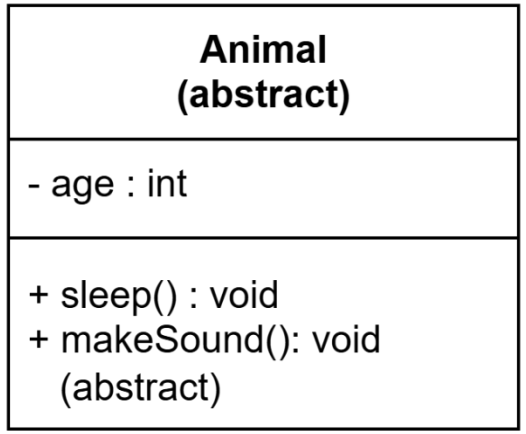
Abstract Class

- **Abstract Class** คือคลาสที่ประกาศด้วยคีย์เวิร์ด abstract
- ไม่สามารถสร้าง Object ของ Abstract Class ได้โดยตรง
- สามารถมี Abstract Method (เมธอดที่ไม่มีการ implement) และ Concrete Method (เมธอดที่มีการ implement)
- ใช้สำหรับเป็นแม่แบบให้คลาสลูก(Subclass)
- คลาสลูกที่สืบทอดจาก Abstract Class ต้อง Override (implement) เมธอดที่เป็น Abstract Method ทั้งหมด
- สามารถมี Constructor

สัญลักษณ์ของ Abstract Class ใน Class Diagram ของ UML

- ชื่อของ Abstract Class และ Abstract Method จะถูกเขียนด้วยตัวเอียงเพื่อแสดงว่ามันเป็นนามธรรมและไม่สามารถสร้าง Object ได้โดยตรง
- นอกจากใช้ตัวเอียงแล้ว บางครั้งยังอาจใช้คำว่า {abstract} เพิ่มเติมไว้ด้านบนหรือด้านล่างชื่อคลาสเพื่อความชัดเจนยิ่งขึ้น โดยเฉพาะในกรณีที่ฟอนต์ที่ใช้ไม่รองรับการเขียนตัวเอียง





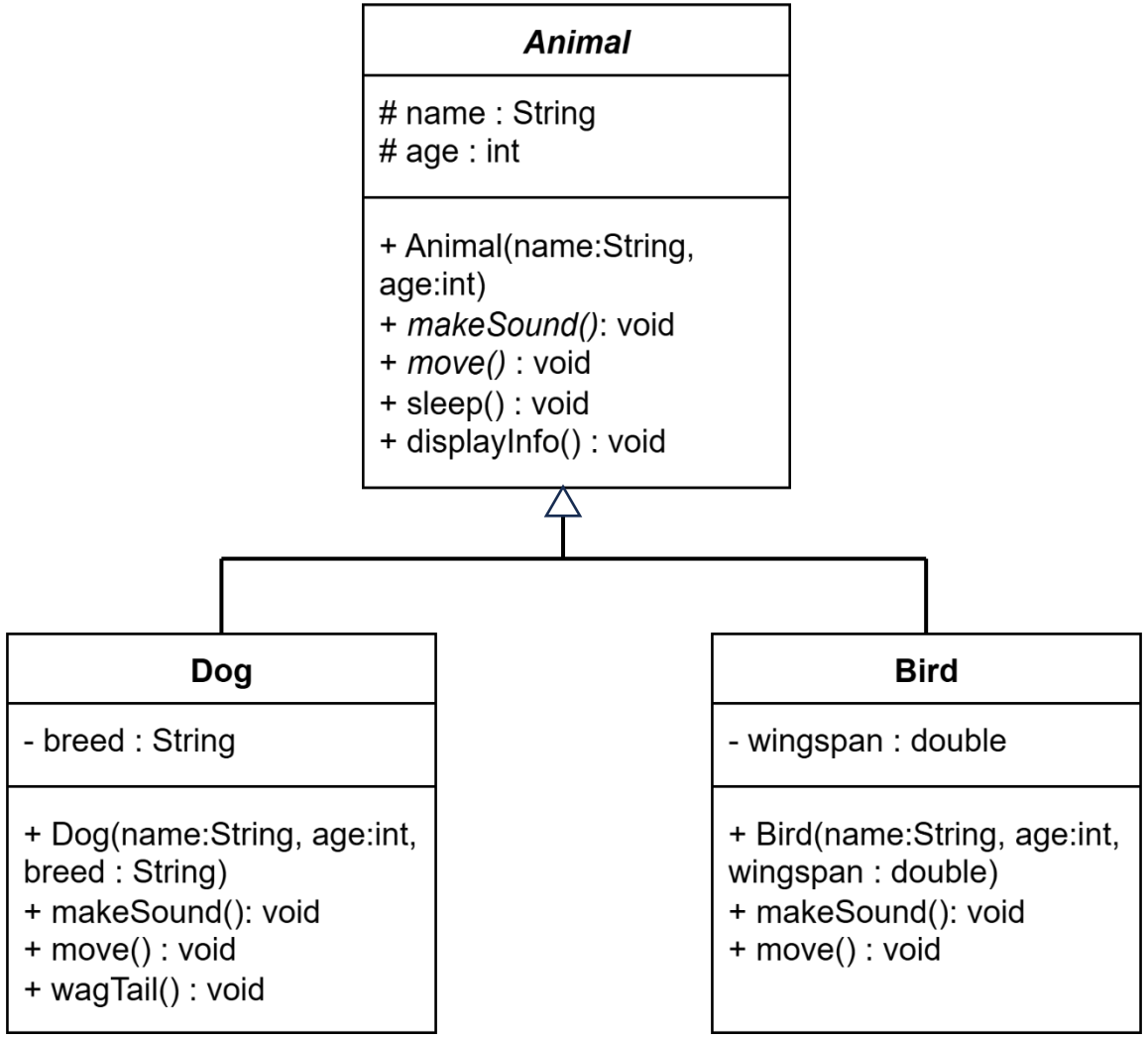

```java
abstract class Animal {
    private int  age;

    //Concrete method
    public void sleep() {
        System.out.println("Zzzzzz....");
    }

    //Abstract method
    public abstract void makeSound();
}
```

Abstract Class : ตัวอย่างพื้นฐาน

Animal
name : String # age : int
+ Animal(name:String, age:int) + *makeSound()*: void + *move()* : void + sleep() : void + displayInfo() : void

```java
abstract class Animal {
    protected String name;
    protected int  age;

    //Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
    //Abstract method
    public abstract void makeSound();
    public abstract void move();

    //Concrete method
    public void sleep() {
        System.out.println("Zzzzzz....");
    }

    public void displayInfo() {
        System.out.println(String.format("Name : %s\nAge  : %d",name,age));
    }
}
```

Abstract Class : การสืบทอด



Abstract Class : การสืบทอด

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

    public Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " barks: Bok! Bok!");
    }

    @Override
    public void move() {
        System.out.println(name + " runs on four legs");
    }

    public void wagTail() {
        System.out.println(name + " wags tail happily");
    }
}
```

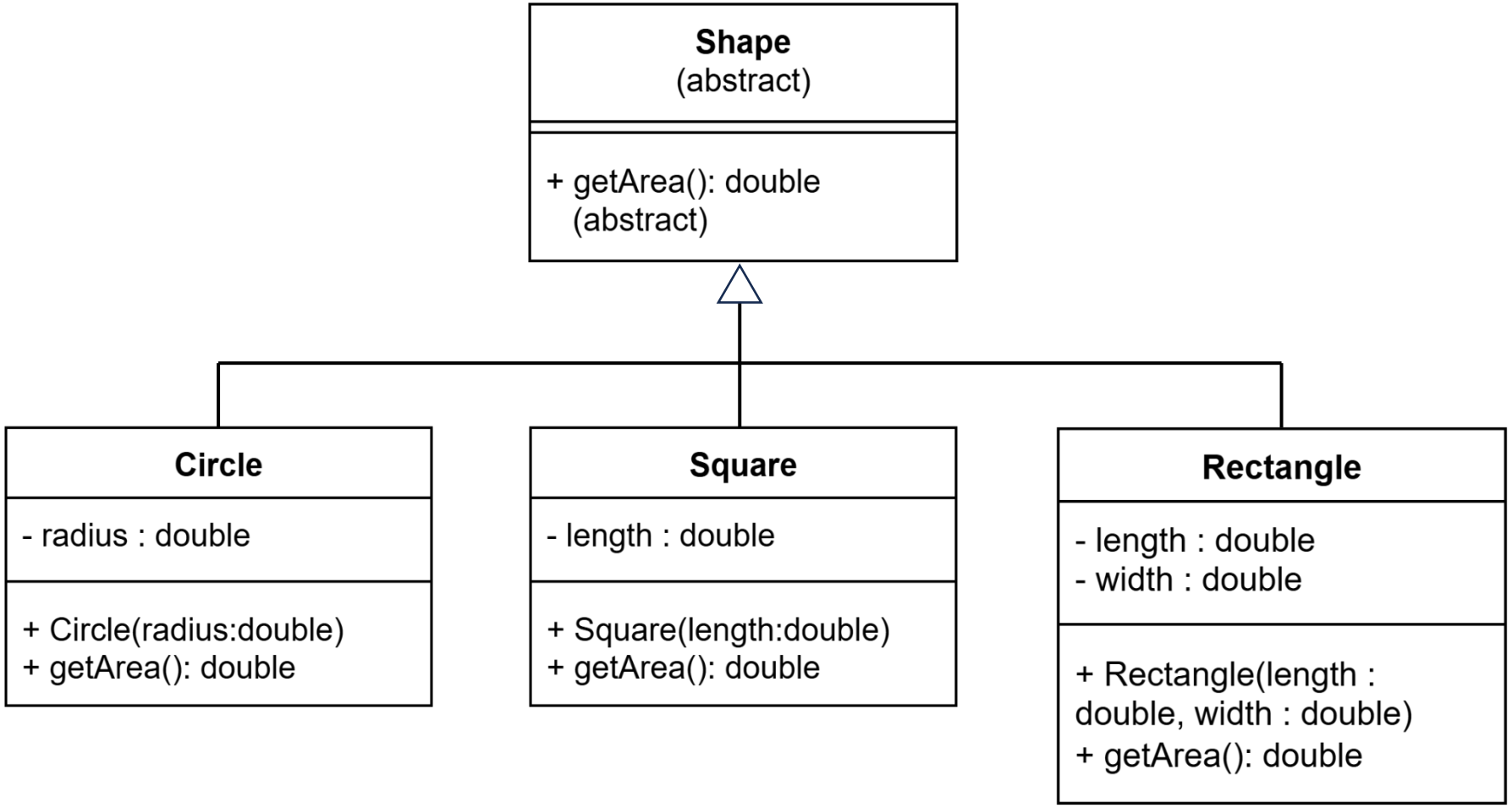
```java
class Bird extends Animal {
    private double wingspan;

    public Bird(String name, int age, double wingspan) {
        super(name, age);
        this.wingspan = wingspan;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " chirps: Jib! Jib!");
    }

    @Override
    public void move() {
        System.out.println(name + " flies with " + wingspan + "cm wingspan");
    }
}
```

Abstract Class : การใช้งาน

```java
public class AnimalTest {
    public static void main(String[] args) {
        Dog dog = new Dog("Soba", 5, "Shih Tzu");
        Bird bird = new Bird("Tweety", 2, 15.0);

        System.out.println("++++++ Dog ++++++");
        dog.displayInfo();
        dog.makeSound();
        dog.move();
        dog.sleep();
        dog.wagTail();

        System.out.println("\n++++++ Bird ++++++");
        bird.displayInfo();
        bird.makeSound();
        bird.move();
        bird.sleep();

        //Polymorphism กับ Abstract Class
        System.out.println("\n++++++ Polymorphism ++++++");
        Animal[] animals = {dog, bird};
        for(Animal animal : animals) {
            animal.makeSound();
            animal.move();
        }
    }
}
```

```
++++++ Dog ++++++
Name : Soba
Age  : 5
Soba barks: Bok! Bok!
Soba runs on four legs
Zzzzzz....
Soba wags tail happily

++++++ Bird ++++++
Name : Tweety
Age  : 2
Tweety chirps: Jib! Jib!
Tweety flies with 15.0cm wingspan
Zzzzzz....

++++++ Polymorphism ++++++
Soba barks: Bok! Bok!
Soba runs on four legs
Tweety chirps: Jib! Jib!
Tweety flies with 15.0cm wingspan
```

Ex1. จงเขียนโค้ด Java สร้าง class ตาม class diagram



Ex2. จงเขียนโค้ด Java สร้าง class ใช้งาน shape ให้แสดงผลลัพธ์ดังนี้

```
+++ Shape Menu +++
  1 : Circle
  2 : Square
  3 : Rectangle
  4 : Exit program.
+++++++++++++++++
```

แสดงเมนู

```
Enter menu ==> 5
*** Invalid menu choice!!!
---------------------------
Enter menu ==> 0
*** Invalid menu choice!!!
---------------------------
Enter menu ==>
```

หากป้อนเมนูไม่ใช่ 1-4 ให้แสดงข้อความ *** Invalid menu choice!!! และให้วนรับค่าเมนูใหม่ จนกว่าจะถูกต้อง

Ex2. จงเขียนโค้ด Java สร้าง class ใช้งาน shape ให้แสดงผลลัพธ์ดังนี้

```
---------------------------
Enter menu ==> 1
Enter radius ==> 5
Circle area : 78.5
---------------------------------------
Start new?(Y/N) ==> Y

+++ Shape Menu +++
  1 : Circle
  2 : Square
  3 : Rectangle
  4 : Exit program.
+++++++++++++++++++
```

กรณีป้อนเมนู Shape ถูกต้องให้รับค่าตาม Shape นั้นๆ และแสดงพื้นที่ของ Shape นั้นๆ

ถ้าตอบ Y ให้วนลูปทำงานตั้งแต่ต้นใหม่อีกรอบ เริ่มตั้งแต่ แสดงเมนู Shape

Ex2. จงเขียนโค้ด Java สร้าง class ใช้งาน shape ให้แสดงผลลัพธ์ดังนี้

```
Enter menu ==> 2
Enter length ==> 10
Square area : 100.0
-----------------------------------
Start new?(Y/N) ==>
```

กรณีป้อนเมนู Shape เป็น 2

```
Enter menu ==> 3
Enter length ==> 10
Enter width ==> 5
Rectangle area : 50.0
-----------------------------------
Start new?(Y/N) ==>
```

กรณีป้อนเมนู Shape เป็น 3

Ex2. จงเขียนโค้ด Java สร้าง class ใช้งาน shape ให้แสดงผลลัพธ์ดังนี้

```
Enter menu ==> 4
**** Program terminated ****
```

กรณีป้อนเมนู Shape เป็น 4
แสดงข้อความ
*** Program terminated ***
และหยุดการทำงานของโปรแกรม

```
Enter menu ==> 1
Enter radius ==> 5
Circle area : 78.5
-----------------------------------
Start new?(Y/N) ==> N
```

กรณีป้อน Start New? ไม่ใช่ Y ให้จบ
การทำงานของโปรแกรม

Interface

- Interface คือพิมพ์เขียวของ Class
- ประกอบด้วย เมธอดที่เป็น Abstract Method ทั้งหมด
- ไม่สามารถมี instance variable ได้ (เฉพาะ constants)
- Method เป็น public abstract โดยอัตโนมัติ
- Variable เป็น public static final โดยอัตโนมัติ

สัญลักษณ์ของ Interface ใน Class Diagram ของ UML

- Interface จะแสดงเป็นกล่องสี่เหลี่ยมคล้ายกับ Class แต่จะมีคำว่า <<interface>> อยู่เหนือชื่อ Interface เพื่อระบุว่าเป็นประเภท Interface
- ชื่อของ Interface จะถูกเขียนด้วยตัวเอียงเช่นเดียวกับ Abstract Class
- เมธอดทั้งหมดที่อยู่ใน Interface จะเป็น public abstract โดยปริยาย ทำให้ไม่จำเป็นต้องเขียนด้วยตัวเอียงหรือระบุ {abstract}

สัญลักษณ์ของ Interface ใน Class Diagram ของ UML

<<interface>> **Flyable**
+ *fly()* : void

<<interface>> **Swimmable**
+ *swim()* : void

```java
public interface Flyable {
    void fly();
}
```

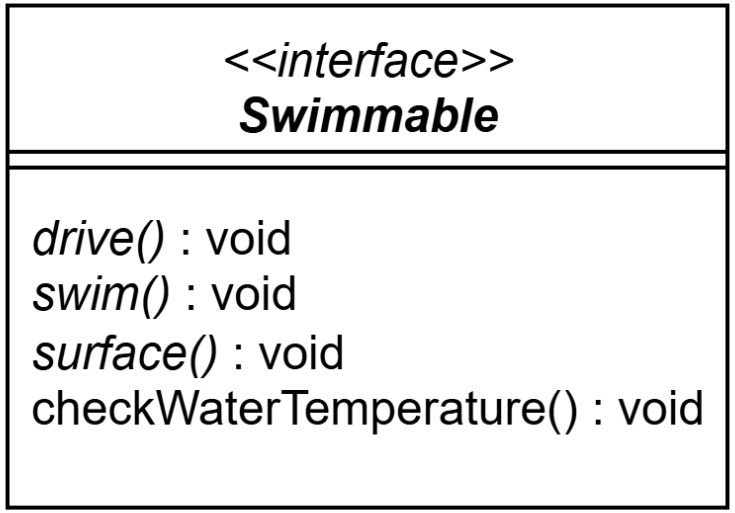
```java
interface Swimmable {
    void swim();
}
```

Interface : ตัวอย่างพื้นฐาน

<<interface>> **Flyable**
MAX_ALTITUDE = 10000.0 FLIGHT_MODE = "FLYING"
takeoff() : void *fly()*: void *land()* : void checkWeather() : void

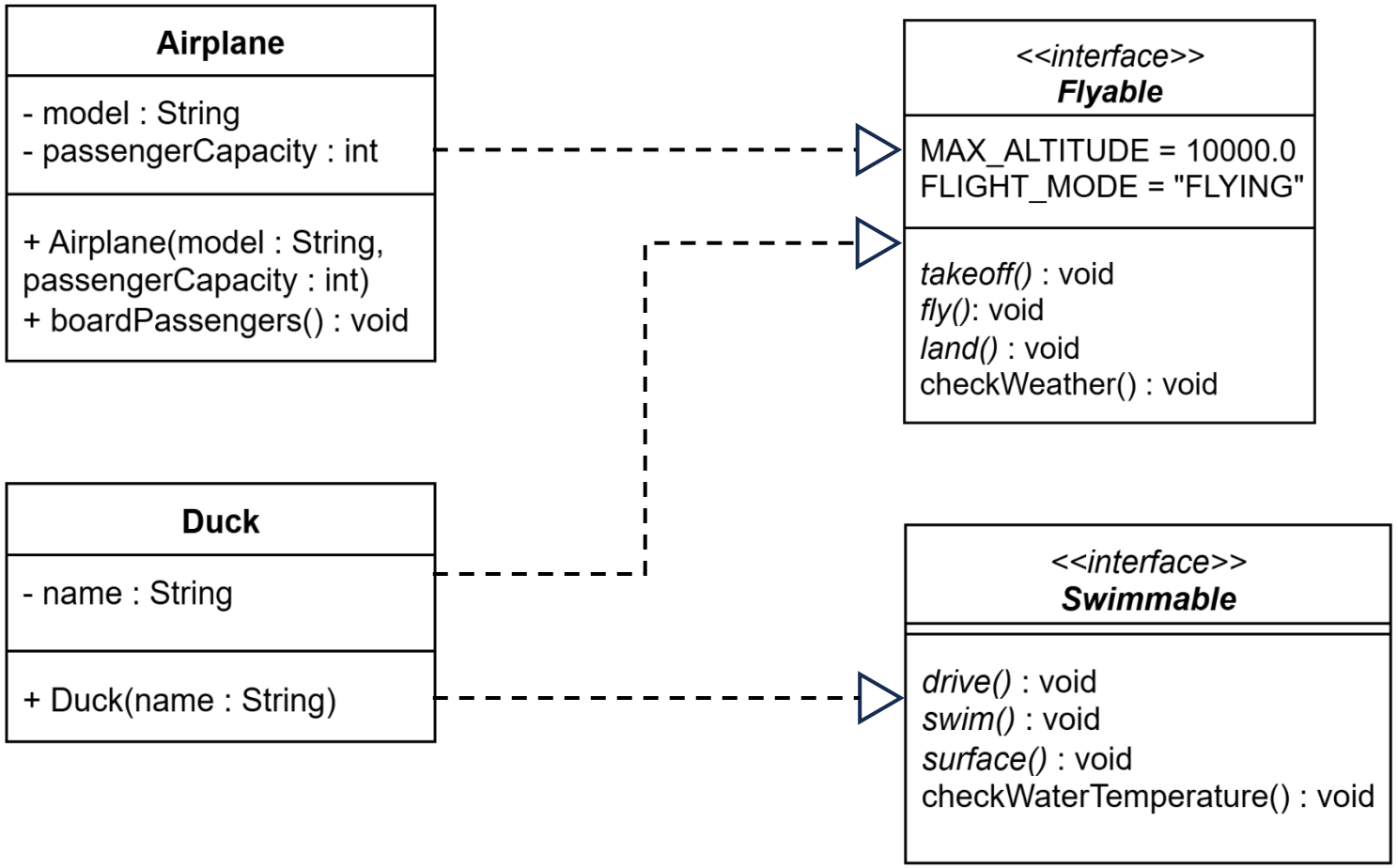
```java
interface Flyable {
    //Constant (public static final อัตโนมัติ
    double MAX_ALTITUDE = 10000.0;
    String FLIGHT_MODE = "FLYING";

    //abstract method (public abstract โดยอัตโนมัติ
    void takeoff();
    void fly();
    void land();

    //default method (java 8+)
    default void checkWeather() {
        System.out.println("Checking weather condition.....");
    }
}
```

Interface : ตัวอย่างพื้นฐาน




```java
interface Swimmable {
    void drive();
    void swim();
    void surface();

    default void checkWaterTemperature() {
        System.out.println("Water temperature is suitable for swimming.");
    }
}
```

Interface Implementation



Interface Implementation

Airplane
- model : String - passengerCapacity : int
+ Airplane(model : String, passengerCapacity : int) + takeoff() : void + fly(): void + land() : void + boardPassengers() : void

```java
public class Airplane implements Flyable {
    private String model;
    private int passengerCapacity;

    public Airplane(String model, int passengerCapacity) {
        this.model = model;
        this.passengerCapacity = passengerCapacity;
    }

    @Override
    public void takeoff() {
        System.out.println(model + " is taking off from runway.");
    }

    @Override
    public void fly() {
        System.out.println(model + " is flying at altitude up to " + MAX_ALTITUDE + " feet.");
    }

    @Override
    public void land() {
        System.out.println(model + " is landing safely.");
    }

    public void boardPassengers() {
        System.out.println("Boarding " + passengerCapacity + " passengers.");
    }
}
```

Interface Implementation

Duck
- name : String
+ Duck(name : String) + takeoff() : void + fly(): void + land() : void + drive() : void + swim() : void + surface() : void

```java
public class Duck implements Flyable, Swimmable{
    private String name;

    public Duck(String name) {
        this.name = name;
    }

    //Flyable methods
    @Override
    public void takeoff() {
        System.out.println(name + " flaps wings and takes off.");
    }

    @Override
    public void fly() {
        System.out.println(name + " flies gracefully in the sky.");
    }

    @Override
    public void land() {
        System.out.println(name + " lands gently on water or ground.");
    }
```

Interface Implementation

Duck
- name : String
+ Duck(name : String) + takeoff() : void + fly(): void + land() : void + drive() : void + swim() : void + surface() : void

```java
    //Swimmable methods
    @Override
    public void drive() {
        System.out.println(name + " dives underwater to find food.");
    }

    @Override
    public void swim() {
        System.out.println(name + " swims on the water surface.");
    }

    @Override
    public void surface() {
        System.out.println(name + " surfaces from underwater.");
    }
}
```

Interface : การใช้งาน

```java
public class InterfaceTest {
    public static void main(String[] args) {
        Airplane boeing = new Airplane("Boeing 737", 180);
        Duck mallard = new Duck("Mallard");

        System.out.println("++++++ Airplane Flight ++++++");
        boeing.checkWeather();
        boeing.takeoff();
        boeing.fly();
        boeing.land();
        boeing.boardPassengers();

        System.out.println("\n++++++ Duck Activities ++++++");
        //Flying activities
        mallard.checkWeather();
        mallard.takeoff();
        mallard.fly();
        mallard.land();
        System.out.println("----------------------------------------");
```

```java
System.out.println("\n++++++ Duck Activities ++++++");
//Flying activities
mallard.checkWeather();
mallard.takeoff();
mallard.fly();
mallard.land();
System.out.println("-----------------------------------------------");

//Swimming activities
mallard.checkWaterTemperature();
mallard.drive();
mallard.swim();
mallard.surface();
System.out.println("===============================================");
```

```java
        //Interface Polymorphism
        System.out.println("\n++++++ Interface Polymorphism ++++++");
        Flyable[] flyingObjects = {boeing, mallard};
        for(Flyable obj : flyingObjects) {
            obj.takeoff();
            obj.fly();
            obj.land();
            System.out.println("----------------------------------------");
        }
    }
}
```

```
++++++ Airplane Flight ++++++
Checking weather condition.....
Boeing 737 is taking off from runway.
Boeing 737 is flying at altitude up to 10000.0 feet.
Boeing 737 is landing safely.
Boarding 180 passengers.

++++++ Duck Activities ++++++
Checking weather condition.....
Mallard flaps wings and takes off.
Mallard flies gracefully in the sky.
Mallard lands gently on water or ground.
-----------------------------------------------
Water temperature is suitable for swimming.
Mallard dives underwater to find food.
Mallard swims on the water surface.
Mallard surfaces from underwater.
===============================================
```

```
++++++ Interface Polymorphism ++++++
Boeing 737 is taking off from runway.
Boeing 737 is flying at altitude up to 10000.0 feet.
Boeing 737 is landing safely.
----------------------------------------
Mallard flaps wings and takes off.
Mallard flies gracefully in the sky.
Mallard lands gently on water or ground.
----------------------------------------
```

Abstract Class vs Interface

```java
abstract class Vehicle {
    protected String brand;      // Instance variable ได้
    protected int year;

    public Vehicle(String brand, int year) {  // Constructor ได้
        this.brand = brand;
        this.year = year;
    }

    public void displayInfo() {  // Concrete method ได้
        System.out.println(brand + " " + year);
    }

    public abstract void start();  // Abstract method
    public abstract void stop();
}
```

Abstract Class vs Interface

```java
interface Drivable {
    String LICENSE_REQUIRED = "YES";  // Constants เท่านั้น

    // ไม่มี Constructor
    void accelerate();     // Abstract method (อัตโนมัติ)
    void brake();
    void steer();

    default void honk() {  // Default method (Java 8+)
        System.out.println("Beep! Beep!");
    }
}
```

ควรใช้ Abstract Class หรือ Interface เมื่อใด

- ใช้ Abstract Class เมื่อ

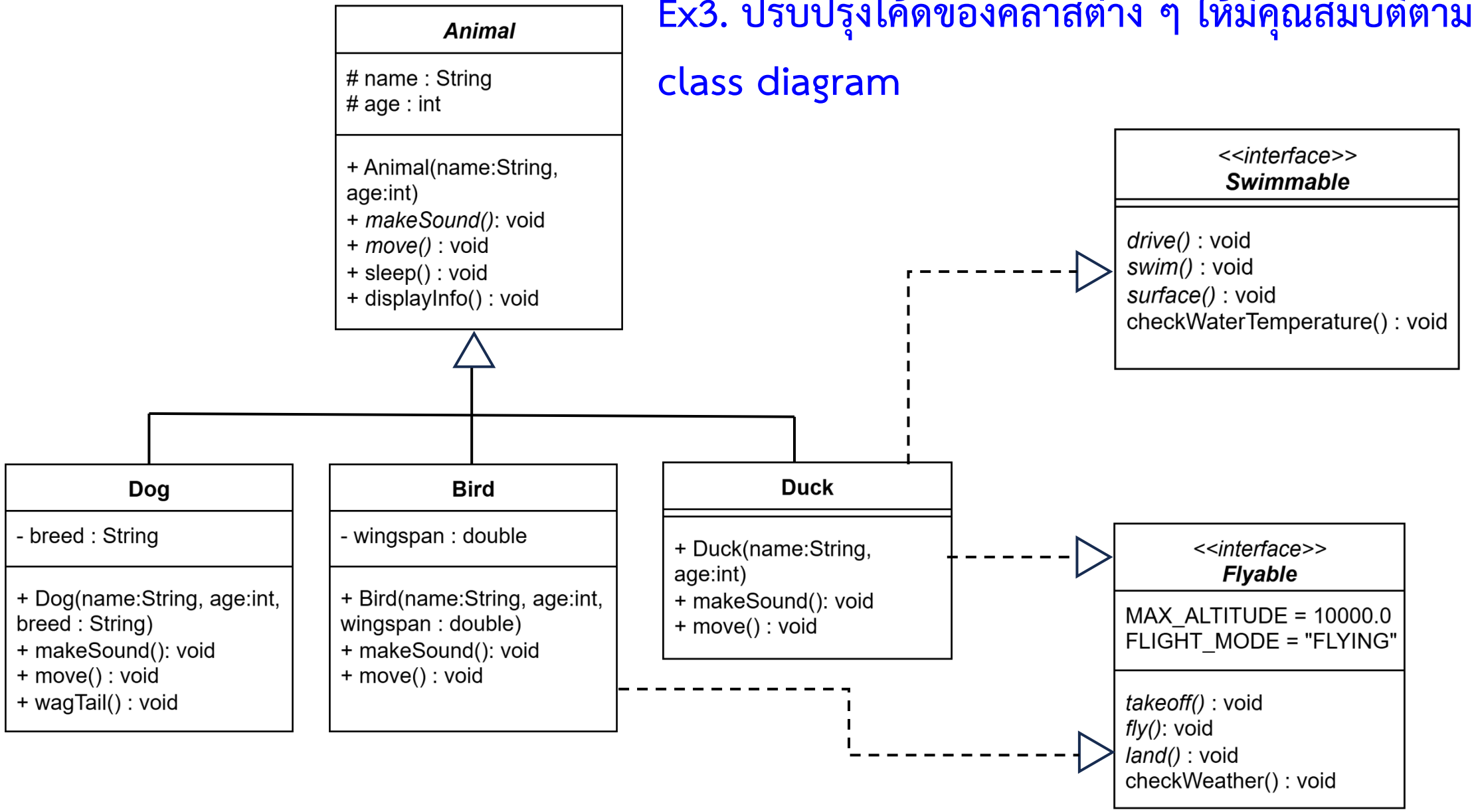
1. **มีโค้ดที่ใช้ร่วมกัน** - มี method ที่มีการ implement แล้ว
2. **มี instance variable** - ต้องเก็บ state
3. **ต้องการ Constructor** – กำหนดค่าเริ่มต้น
4. **มี access modifier ที่หลากหลาย** - protected, private
5. **เป็นแนวคิด "IS-A"** - Dog IS-A Animal

ควรใช้ Abstract Class หรือ Interface เมื่อใด

- ใช้ Interface เมื่อ

1. **กำหนดสัญญา (Contract)** - บอกว่าต้องมี method อะไรบ้าง
2. **Multiple Implementation** - Class หนึ่งทำได้หลายอย่าง
3. **เป็นแนวคิด "CAN-DO"** - Duck CAN-DO Flying, Swimming
4. **ไม่ต้องการ state** - เฉพาะ behavior
5. **ต้องการ loose coupling** - ลดการพึ่งพาแม่แบบเดียว

Ex3. ปรับปรุงโค้ดของคลาสต่าง ๆ ให้มีคุณสมบัติตาม class diagram

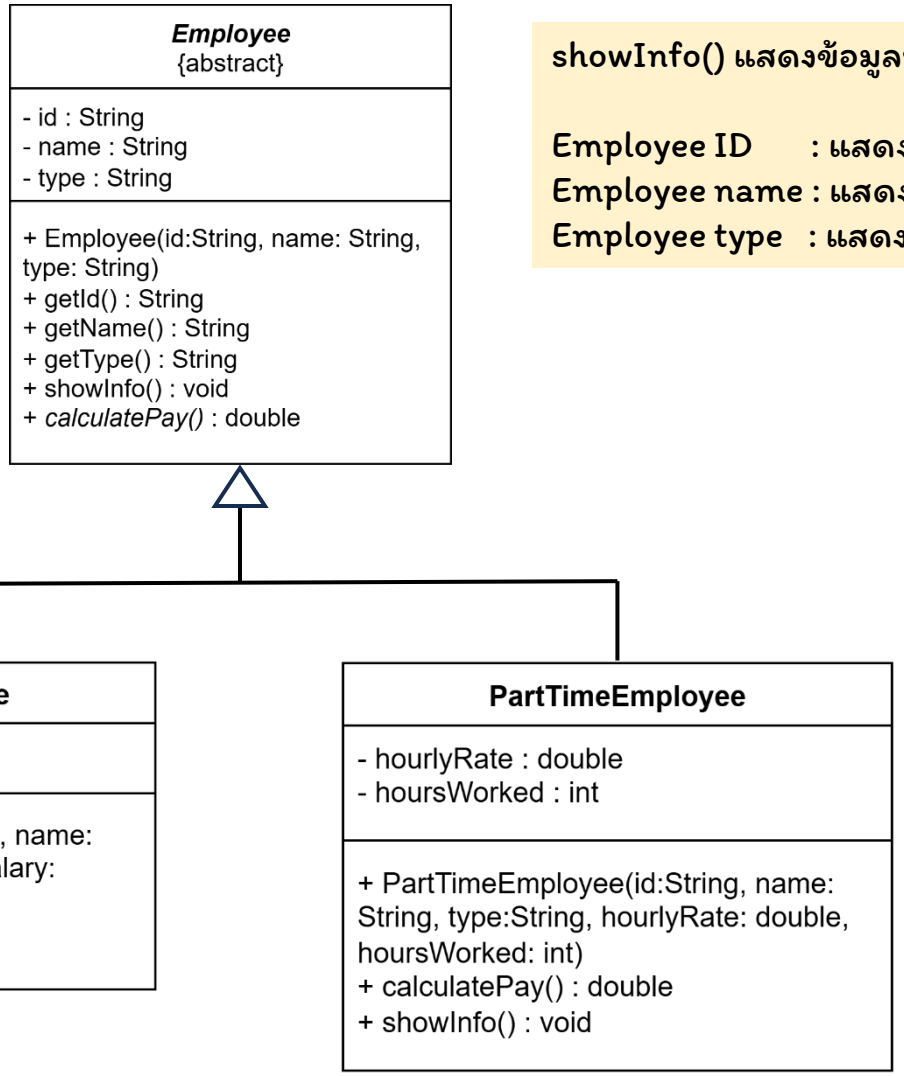


Ex4. ปรับปรุงการทำงานของไฟล์ AnimalTest ให้ได้ผลลัพธ์ดังนี้

```
++++++ Dog ++++++
Name : Soba
Age  : 5
Soba barks: Bok! Bok!
Soba runs on four legs
Zzzzzz....
Soba wags tail happily

++++++ Bird ++++++
Name : Tweety
Age  : 2
Tweety chirps: Jib! Jib!
Tweety flies with 15.0cm wingspan
Zzzzzz....
Tweety flaps wings and takes off.
Tweety flies gracefully in the sky.
Tweety lands gently on tree or ground.
```

```
++++++ Duck ++++++
Name : Mallard
Age  : 2
Mallard quacks: Gabb! Gabb!
Mallard move by walking or floating.
Zzzzzz....
Mallard flaps wings and takes off.
Mallard flies gracefully in the sky.
Mallard lands gently on water or ground.
Mallard dives underwater to find food.
Mallard swims on the water surface.
Mallard surfaces from underwater.
========================================

++++++ Polymorphism ++++++
Soba barks: Bok! Bok!
Soba runs on four legs
Tweety chirps: Jib! Jib!
Tweety flies with 15.0cm wingspan
Mallard quacks: Gabb! Gabb!
Mallard move by walking or floating.
```

Ex5. เขียนโค้ด Java ตาม UML Class Diagram ข้างล่างนี้



showInfo() แสดงข้อมูลพนักงาน โดยมีรูปแบบ

Employee ID : แสดงรหัสพนักงาน
Employee name : แสดงชื่อพนักงาน
Employee type : แสดงประเภทพนักงาน

showInfo() แสดงข้อมูลพนักงาน โดยมีรูปแบบ

++++ Full Time Employee ++++
Employee ID : แสดงรหัสพนักงาน
Employee name : แสดงชื่อพนักงาน
Employee type : แสดงประเภทพนักงาน
Salary : แสดงค่าจ้างที่ได้รับ

calculatePay() ให้ส่งค่าเป็น monthlySalary

showInfo() แสดงข้อมูลพนักงาน โดยมีรูปแบบ

++++ Part Time Employee ++++
Employee ID : แสดงรหัสพนักงาน
Employee name : แสดงชื่อพนักงาน
Employee type : แสดงประเภทพนักงาน
Wages : แสดงค่าจ้างที่ได้รับ

calculatePay() ให้ส่งค่าเป็น hourlyRate * hoursWorked

Ex6. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

1. รับจำนวนพนักงานทั้งหมด
2. รับข้อมูลพนักงานทีละคนตามจำนวนที่ป้อนในข้อ 1 ใส่อาร์เรย์
3. แสดงข้อมูลพนักงานทีละคน

```
+++++ Payroll System +++++
Enter number of employees ==> 4
Enter employee no.1 data.....
    Employee ID ==> E01
    Employee name ==> Rewadee
    Employee type(1:Full time / 2: Part time) ==> 1
    Salary ==> 52000
++++ Full Time Employee ++++
Employee ID      : E01
Employee name    : Rewadee
Employee type    : Full time
Salary           : 52000.00
-------------------------------------------------------
```

1. ป้อนจำนวนพนักงานทั้งหมด

2. กรณีป้อน Employee type เป็น 1 ให้รับข้อมูลเงินเดือน(Salary)

3. แสดงข้อมูลโดยใช้ showInfo()

Ex6. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

```
Enter employee no.2 data.....
    Employee ID ==> E02
    Employee name ==> Khwanchai
    Employee type(1:Full time / 2: Part time) ==> 2
    Hourly rate ==> 1500
    Hours worked ==> 30
++++ Part Time Employee ++++
Employee ID      : E02
Employee name    : Khwanchai
Employee type    : Part time
Wages            : 45000.00
-----------------------------------------------------
```

2. กรณีป้อน Employee type เป็น 2 ให้รับข้อมูล อัตราค่าจ้างรายชม.(hourlyRate) และ จำนวนชม.ที่ทำงาน (hoursWorked)

3. แสดงข้อมูลโดยใช้ showInfo()

Ex6. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

```
Enter employee no.3 data.....
    Employee ID ==> E05
    Employee name ==> Wannasiri
    Employee type(1:Full time / 2: Part time) ==> 3
--- Invalid employee type!! ---
    Employee type(1:Full time / 2: Part time) ==> 5
--- Invalid employee type!! ---
    Employee type(1:Full time / 2: Part time) ==> 1
    Salary ==> 48000
++++ Full Time Employee ++++
Employee ID      : E05
Employee name    : Wannasiri
Employee type    : Full time
Salary           : 48000.00
-----------------------------------------------------
```

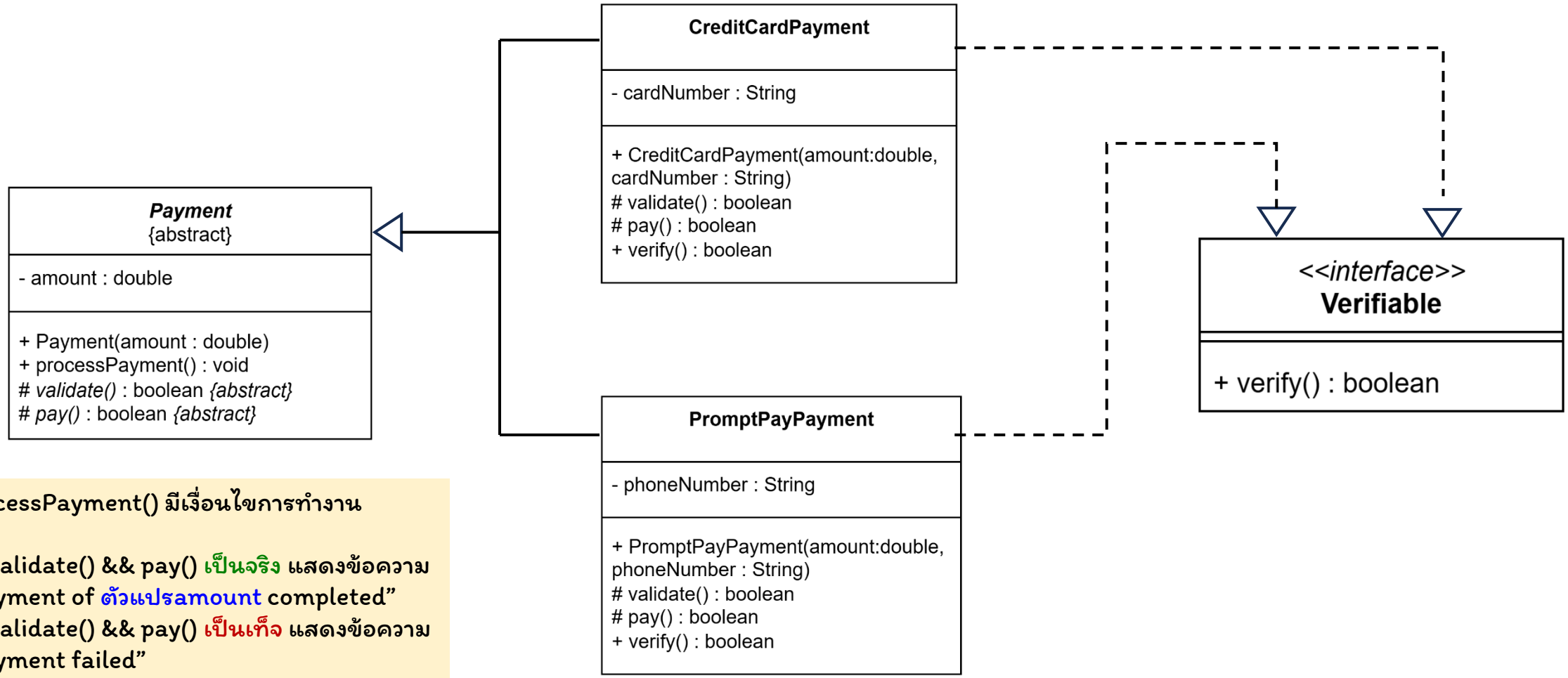
2. กรณีป้อน Employee type เป็น ค่าอื่น(ไม่ใช่ 1 หรือ 2) ให้แสดงข้อความ Invalid employee type และให้วนรับรหัสประเภทพนักงานใหม่

Ex6. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

4. หลังจากรับข้อมูลครบตามจำนวนที่ระบุในข้อ 1 จะแสดงข้อมูลเป็นตารางตามรูปแบบข้างล่าง

```
******* Employee payroll Report *******

---------------------------------------------
ID     Name              Type          Earnings
---------------------------------------------
E01    Rewadee           Full time     52000.00
E02    Khwanchai         Part time     45000.00
E05    Wannasiri         Full time     48000.00
E08    Wilaiporn         Full time     57000.00
```

Ex7. เขียนโค้ด Java ตาม UML Class Diagram ข้างล่างนี้



processPayment() มีเงื่อนไขการทำงาน

ถ้า validate() && pay() เป็นจริง แสดงข้อความ
"Payment of ตัวแปรamount completed"
ถ้า validate() && pay() เป็นเท็จ แสดงข้อความ
"Payment failed"

Verifying OTP for Credit Card Payment...

Ex7. เขียนโค้ด Java ตาม UML Class Diagram ข้างล่างนี้

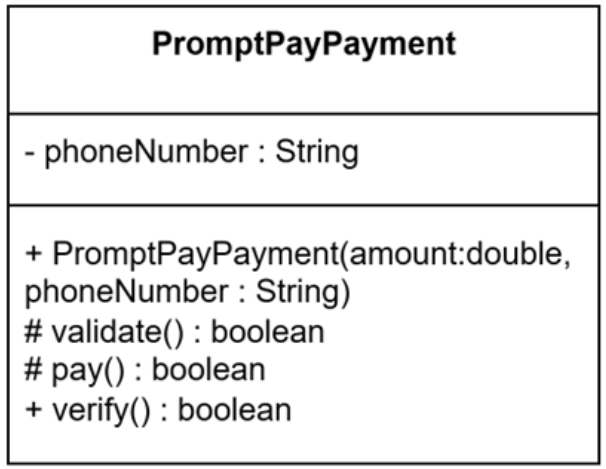
CreditCardPayment
- cardNumber : String
+ CreditCardPayment(amount:double, cardNumber : String) # validate() : boolean # pay() : boolean + verify() : boolean

validate() มีการทำงานดังนี้
- แสดงข้อความ Validating credit card: แสดงค่าcardNumber
- ถ้าความยาวของ cardNumber == 16 ส่งค่า true
- ถ้าความยาวของ cardNumber != 16 ส่งค่า false

pay() มีการทำงานดังนี้
- ส่งค่ากลับเป็นค่าจาก verify()

verify() มีการทำงานดังนี้
- แสดงข้อความ Verifying OTP for Credit Card Payment...
- ส่งค่ากลับเป็น true (สมมุติว่าตรวจสอบถูกต้อง)

Ex7. เขียนโค้ด Java ตาม UML Class Diagram ข้างล่างนี้



validate() มีการทำงานดังนี้
- แสดงข้อความ Validating PromptPay number : แสดงค่าphoneNumber
- ถ้าความยาวของ phoneNumber == 10 ส่งค่า true
- ถ้าความยาวของ phoneNumber != 10 ส่งค่า false

pay() มีการทำงานดังนี้
- ส่งค่ากลับเป็นค่าจาก verify()

verify() มีการทำงานดังนี้
- แสดงข้อความ Verifying phone number with OTP...
- ส่งค่ากลับเป็น true (สมมุติว่าตรวจสอบถูกต้อง)

Ex8. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

1. รับข้อมูลราคาต่อหน่วยและจำนวนที่ซื้อของสินค้า
2. แสดงราคารวมของสินค้าที่ซื้อ
3. รับประเภทการชำระเงิน (1 : Credit card / 2 : Prompt Pay)
4. แสดงผลการชำระเงิน

Ex8. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

```
Enter unit price of product ==> 1000
Enter the purchase quantity ==> 5
Sum price = 5000.0
-----------------------------
Payment method
1 : Credit card
2 : Prompt Pay
Enter payment method(1/2) ==> 1
Enter credit card number ==> 1234567890123456
*******************************************
Validating credit card: 1234567890123456
Verifying OTP for Credit Card Payment...
Payment of 5000.0 completed.
*******************************************

Purchase new(Y/N)? ==> Y
```

กรณีเลือกการชำระเงินแบบ Credit card

```
Enter unit price of product ==> 1000
Enter the purchase quantity ==> 5
Sum price = 5000.0
-----------------------------
Payment method
1 : Credit card
2 : Prompt Pay
Enter payment method(1/2) ==> 2
Enter phone number ==> 0819754012
*******************************************
Validating PromptPay number: 0819754012
Verifying phone number with OTP...
Payment of 5000.0 completed.
*******************************************

Purchase new(Y/N)? ==> Y
```

กรณีเลือกการชำระเงินแบบ Prompt Pay

Ex8. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

```
Enter unit price of product ==> 1000
Enter the purchase quantity ==> 5
Sum price = 5000.0
-----------------------------
Payment method
1 : Credit card
2 : Prompt Pay
Enter payment method(1/2) ==> 1
Enter credit card number ==> 123456890
*****************************************
Validating credit card: 123456890
Payment failed.
*****************************************

Purchase new(Y/N)? ==>
```

กรณีป้อนหมายเลขบัตรเครดิตไม่ครบ 16 หลัก

```
Enter unit price of product ==> 1000
Enter the purchase quantity ==> 5
Sum price = 5000.0
-----------------------------
Payment method
1 : Credit card
2 : Prompt Pay
Enter payment method(1/2) ==> 2
Enter phone number ==> 08176549
*****************************************
Validating PromptPay number: 08176549
Payment failed.
*****************************************

Purchase new(Y/N)? ==>
```

กรณีป้อนหมายเลขโทรศัพท์ไม่ครบ 10 หลัก

Ex8. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

```
Enter unit price of product ==> 1000
Enter the purchase quantity ==> 5
Sum price = 5000.0
-----------------------------
Payment method
1 : Credit card
2 : Prompt Pay
Enter payment method(1/2) ==> 5
+++ Invalid payment method!!! +++
---------------------------------
Enter payment method(1/2) ==> 8
+++ Invalid payment method!!! +++
---------------------------------
Enter payment method(1/2) ==> -1
+++ Invalid payment method!!! +++
---------------------------------
Enter payment method(1/2) ==>
```

กรณีป้อนรหัสวิธีการชำระเงินไม่ถูกต้อง แสดงข้อความ
+++ Invalid payment method +++
และให้วนรับใหม่ จนกว่าจะป้อนถูกต้อง

Ex8. เขียนโค้ด Java ตามเงื่อนไขต่อไปนี้

```
*****************************************
Validating PromptPay number: 0815674022
Verifying phone number with OTP...
Payment of 5000.0 completed.
*****************************************

Purchase new(Y/N)? ==> Y
```

- กรณีป้อน Y จะวนไปทำงานข้อ 1 ใหม่
- กรณีป้อนค่าอื่น ๆ ให้หยุดการทำงานของโปรแกรม

END.

Q & A