บทที่ 4

การห่อหุ้ม

Encapsulation

สอนโดย อ.เรวดี พิพัฒน์สูงเนิน สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏอุดรธานี

หลักการเขียนโปรแกรมเชิงวัตถุ

วัตถุประสงค์การเรียนรู้

- สามารถอธิบายความหมายและประโยชน์ของการห่อหุ้มได้
- สามารถระบุและใช้งานตัวระบุการเข้าถึง (Access Modifiers) ในภาษาจาวาได้
- สามารถออกแบบคลาสที่ใช้หลักการห่อหุ้มได้อย่างถูกต้อง
- สามารถสร้างเมธอด Getter และ Setter เพื่อเข้าถึงข้อมูลที่ถูกห่อหุ้มได้

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)

Encapsulation คืออะไร?

- **Encapsulation (การห่อหุ้ม)** คือ การรวมข้อมูล (Data) และเมธอด (Methods) ที่ทำงานกับข้อมูลนั้นไว้ในคลาสเดียวกัน พร้อมทั้งควบคุมการเข้าถึงข้อมูลจากภายนอก
- เปรียบเทียบในชีวิตจริง เช่น
  - แคปซูลยา: ห่อหุ้มยาภายในเปลือกแข็ง ไม่ให้สารภายในสัมผัสโดยตรง
  - โทรศัพท์มือถือ: ซ่อนวงจรซับซ้อนภายใน แสดงเพียงปุ่มกดที่จำเป็น
  - รถยนต์: ผู้ขับใช้พวงมาลัย เบรก คันเร่ง โดยไม่ต้องรู้รายละเอียดเครื่องยนต์

หลักการสำคัญของ Encapsulation

1. **Data Hiding (การซ่อนข้อมูล)**
   - ป้องกันการเข้าถึงข้อมูลโดยตรงจากภายนอกคลาส
   - ลดการผิดพลาดจากการแก้ไขข้อมูลผิดวิธี
2. **Controlled Access (การควบคุมการเข้าถึง)**
   - กำหนดวิธีการเข้าถึงและแก้ไขข้อมูล
   - ตรวจสอบความถูกต้องก่อนการแก้ไขข้อมูล
3. **Code Organization (การจัดระเบียบโค้ด)**
   - จัดกลุ่มข้อมูลและฟังก์ชันที่เกี่ยวข้องไว้ด้วยกัน
   - ทำให้โค้ดง่ายต่อการดูแลรักษา

กุญแจสำคัญสู่ Encapsulation : Access Modifiers

- **Access Modifiers (ตัวระบุการเข้าถึง)** คือ คำสั่งที่ใช้กำหนดระดับการเข้าถึงของคลาส, ตัวแปร, เมธอด และ Constructor
- **มี 4 ประเภทหลักใน Java** ได้แก่
  - private
  - (default) หรือ package-private
  - protected
  - public

Access Modifier : private

- **การเข้าถึง :** สามารถเข้าถึงได้เฉพาะภายในคลาสที่ประกาศเท่านั้น
- **การใช้งานทั่วไป :** ใช้กับ Fields (Attributes/ข้อมูล) เพื่อซ่อนข้อมูล
- **ตัวอย่าง :**

```
class Person {
    private String name; // สามารถเข้าถึงได้แค่ภายในคลาส Person
    private int age;     // สามารถเข้าถึงได้แค่ภายในคลาส Person

    // ... methods
}
```

Access Modifier : public

- **การเข้าถึง :** สามารถเข้าถึงได้จากทุกที่
- **การใช้งานทั่วไป :** ใช้กับ Methods (พฤติกรรม) ที่ต้องการให้คลาสอื่นเรียกใช้งานได้
- **ตัวอย่าง :**

```
class Person {
    private String name;
    private int age;

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

Access Modifier : default (Package-Private)

- **การเข้าถึง :** สามารถเข้าถึงได้เฉพาะภายในแพ็คเกจ (Package) เดียวกันเท่านั้น
- **การใช้งานทั่วไป :** หากไม่ระบุ Access Modifier ใดๆ จะถือว่าเป็น default
- **ตัวอย่าง :**

```
package com.example.model;

class Person { // default access (package-private)
    String address; // default access (package-private)

    void doSomething() { // default access (package-private)
        // ...
    }
}
```

Access Modifier : default (Package-Private)

```java
package com.example.model;

class AnotherClass {
    void test() {
        Person p = new Person();
        p.address = "123 Main St."; // เข้าถึงได้
        p.doSomething();            // เข้าถึงได้
    }
}
```

```java
package com.example.app;

public class Main {
    public static void main(String[] args) {
        Person p = new Person(); // Error: Person is not public
        // p.address = "Test"; // Error
    }
}
```

Access Modifier : protected

- **การเข้าถึง :** สามารถเข้าถึงได้ภายในแพ็คเกจเดียวกัน และคลาสลูก (Subclass) แม้จะอยู่ต่างแพ็คเกจก็ตาม
- **การใช้งานทั่วไป :** มักใช้กับตัวแปรหรือเมธอดที่ต้องการให้คลาสลูกเข้าถึงได้โดยตรง
- **ตัวอย่าง :** (จะเห็นอธิบายชัดเจนในเรื่อง Inheritance)

Access Modifier : protected

```java
package com.example.vehicles;

public class Vehicle {  1 inheritor
    protected String brand; // เข้าถึงได้จากคลาสลูก และคลาสใน package เดียวกัน

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

```java
package com.example.cars;

import com.example.vehicles.Vehicle;
class Car extends Vehicle {
    // Car เป็นคลาสลูกของ Vehicle
    public Car(String brand) {
        super(brand);
        System.out.println("Car brand: " + this.brand); // สามารถเข้าถึง brand ได้
    }
}
```

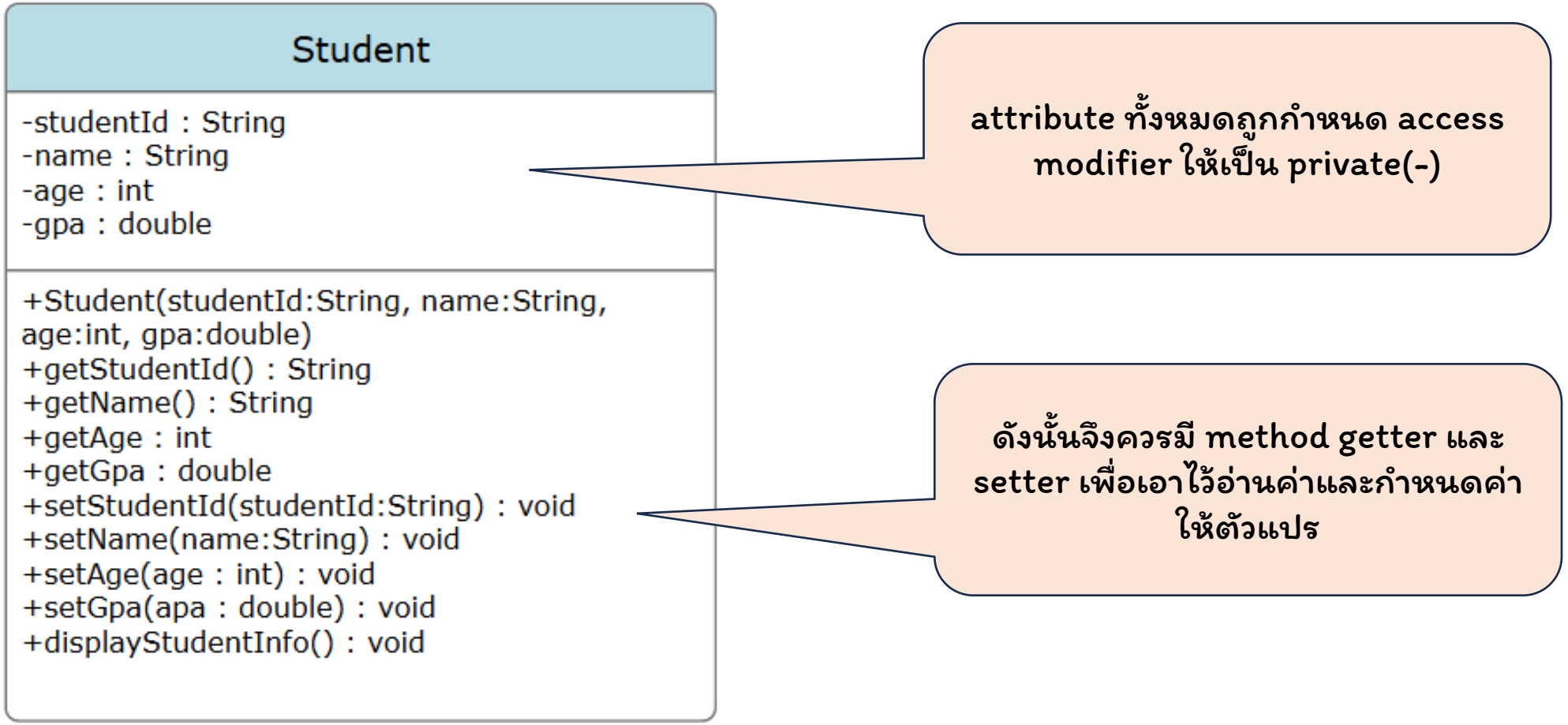
ตารางสรุป Access Modifiers

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

Getter และ Setter Methods

- เนื่องจากข้อมูล (attribute) มักถูกประกาศเป็น private จึงไม่สามารถเข้าถึงได้โดยตรงจากภายนอก
- เราจึงต้องมีเมธอดสาธารณะ (public) เพื่อ "เข้าถึง" หรือ "แก้ไข" ข้อมูลเหล่านั้นอย่างมีขั้นตอนและควบคุมได้
- **Getter (Accessor Method)** : เมธอดที่ใช้สำหรับ "อ่าน" ค่าของตัวแปร
  - ชื่อมักขึ้นต้นด้วย get ตามด้วยชื่อตัวแปร หรือ is สำหรับ Boolean (เช่น getName(), getAge())
  - คืนค่าเป็นชนิดข้อมูลเดียวกับตัวแปร
- **Setter (Mutator Method)** : เมธอดที่ใช้สำหรับ "กำหนด" หรือ "เปลี่ยนแปลง" ค่าของตัวแปร
  - ชื่อมักขึ้นต้นด้วย set ตามด้วยชื่อตัวแปร (เช่น setName(), setAge())
  - รับพารามิเตอร์เป็นค่าที่ต้องการจะกำหนด
  - มักไม่มีการคืนค่า (เป็น void)

การออกแบบคลาส Student แบบมี Encapsulation



ตัวอย่างโค้ดคลาส Student แบบมี Encapsulation

ไฟล์ Student.java

```java
public class Student {
    // 1. กำหนด attributes ให้เป็น private เพื่อซ่อนข้อมูล
    private String studentId;
    private String name;
    private int age;
    private double gpa;

    // 2. Constructor สำหรับสร้าง Object
    public Student(String studentId, String name, int age, double gpa) {
        this.studentId = studentId;
        this.name = name;
        // ใช้ Setter เพื่อให้มีการตรวจสอบค่าเบื้องต้นได้
        setAge(age);
        setGpa(gpa);
    }
```

ตัวอย่างโค้ดคลาส Student แบบมี Encapsulation

ไฟล์ Student.java (ต่อ)

```java
// 3. Getter Methods สำหรับอ่านค่า
public String getStudentId() {
    return studentId;
}

public String getName() {
    return name;
}

public int getAge() {
    return age;
}

public double getGpa() {
    return gpa;
}
```

ตัวอย่างโค้ดคลาส Student แบบมี Encapsulation

ไฟล์ Student.java (ต่อ)

```java
// 4. Setter Methods สำหรับกำหนดค่า (พร้อม Validation)
public void setStudentId(String studentId) {
    // อาจมี logic ตรวจสอบความถูกต้องของ studentId
    if (studentId != null && !studentId.isEmpty()) {
        this.studentId = studentId;
    } else {
        System.out.println("Student ID cannot be empty.");
    }
}

public void setName(String name) {
    if (name != null && !name.isEmpty()) {
        this.name = name;
    } else {
        System.out.println("Name cannot be empty.");
    }
}
```

ตัวอย่างโค้ดคลาส Student แบบมี Encapsulation

ไฟล์ Student.java (ต่อ)

```
    public void setAge(int age) {
        // การตรวจสอบความถูกต้อง (Validation Logic)
        if (age >= 15 && age <= 100) {
            this.age = age;
        } else {
            System.out.println("Invalid age. Age must be between 15 and 100.");
        }
    }

    public void setGpa(double gpa) {
        // การตรวจสอบความถูกต้อง (Validation Logic)
        if (gpa >= 0.0 && gpa <= 4.0) {
            this.gpa = gpa;
        } else {
            System.out.println("Invalid GPA. GPA must be between 0.0 and 4.0.");
        }
    }
```

ตัวอย่างโค้ดคลาส Student แบบมี Encapsulation

ไฟล์ Student.java (ต่อ)

```
    // เมธอดอื่นๆ ที่เกี่ยวข้องกับ Student (เช่น displayInfo)
    public void displayStudentInfo() {
        System.out.println("ID: " + studentId + ", Name: " + name +
                ", Age: " + age + ", GPA: " + gpa);
    }
}
```

การใช้งานคลาส Student จากภายนอก

ไฟล์ StudentApp.java

```java
public class StudentApp {
    public static void main(String[] args) {
        // สร้าง Object Student
        Student student1 = new Student("IT66001", "สมชาย ใจดี", 20, 3.55);
        student1.displayStudentInfo(); // ID: IT66001, Name: สมชาย ใจดี, Age: 20, GPA: 3.55

        // การเข้าถึงข้อมูลผ่าน Getter
        System.out.println("Student 1's Name: " + student1.getName()); // สมชาย ใจดี
        System.out.println("Student 1's GPA: " + student1.getGpa());   // 3.55

        // การแก้ไขข้อมูลผ่าน Setter (มีการตรวจสอบ)
        student1.setAge(12); // Invalid age. Age must be between 15 and 100.
        student1.setGpa(4.5); // Invalid GPA. GPA must be between 0.0 and 4.0.
        student1.setName(""); // Name cannot be empty.
```

การใช้งานคลาส Student จากภายนอก

ไฟล์ StudentApp.java (ต่อ)

```
        student1.setAge(21);
        student1.setGpa(3.80);
        student1.setName("สมศรี มีสุข");

        student1.displayStudentInfo(); // ID: IT66001, Name: สมศรี มีสุข, Age: 21, GPA: 3.8

        // ลองพยายามเข้าถึง private field โดยตรง (จะเกิด Error)
        // student1.name = "ชื่อใหม่"; // Compile-time Error
    }
}
```

ข้อดีของ Encapsulation จากตัวอย่าง

- **ควบคุมการเข้าถึง** age และ gpa ไม่สามารถกำหนดค่าผิดพลาดจากภายนอกได้โดยตรง เพราะมี if-else ใน Setter
- **ง่ายต่อการบำรุงรักษา** หากมีการเปลี่ยนแปลงเงื่อนไขของอายุหรือ GPA (เช่น มหาวิทยาลัยเปลี่ยนเกณฑ์อายุต่ำสุด) เราแก้ไขแค่ในเมธอด setAge() หรือ setGpa() เท่านั้น ไม่ต้องแก้โค้ดภายนอก
- **ซ่อนรายละเอียด** ผู้ใช้คลาส Student ไม่จำเป็นต้องรู้ว่า age ถูกเก็บเป็น int หรือมีการตรวจสอบค่าอย่างไร เพียงแค่เรียกใช้ setAge()

หลักการ "ข้อมูลเป็นส่วนตัว, เมธอดเป็นสาธารณะ" (Data Hiding, Method Public)

- โดยทั่วไปแล้ว **Attribute (ข้อมูล)** ควรเป็น private
- **Methods (เมธอด)** ที่เป็นพฤติกรรมของวัตถุและต้องการให้คลาสอื่นเรียกใช้ ควรเป็น public
- นี่คือหัวใจของการห่อหุ้ม เพื่อให้ข้อมูลปลอดภัย และควบคุมการทำงานของวัตถุ

เมื่อไหร่ที่อาจไม่จำเป็นต้องมี Setter?

- บางครั้ง ข้อมูลบางอย่างอาจต้องการให้กำหนดได้เพียงครั้งเดียวตอนสร้างวัตถุ (ผ่าน Constructor) และไม่สามารถเปลี่ยนแปลงได้ในภายหลัง
- ตัวอย่าง: studentId ในคลาส Student อาจต้องการให้เป็น final และกำหนดค่าได้แค่ครั้งเดียว เพื่อป้องกันการเปลี่ยนแปลงรหัส

เมื่อไหร่ที่อาจไม่จำเป็นต้องมี Setter?

```java
public class Student {
    // 1. กำหนด attributes ให้เป็น private เพื่อซ่อนข้อมูล
    private final String studentId;
    private String name;
    private int age;
    private double gpa;

    // 2. Constructor สำหรับสร้าง Object
    public Student(String studentId, String name, int age, double gpa) {
        this.studentId = studentId;
        this.name = name;
        // ใช้ Setter เพื่อให้มีการตรวจสอบค่าเบื้องต้นได้
        setAge(age);
        setGpa(gpa);
    }
```

สรุป Encapsulation

- **Encapsulation** คือการรวมข้อมูลและเมธอดไว้ในคลาสเดียวกัน พร้อมควบคุมการเข้าถึง
- **Private attributes** ป้องกันการเข้าถึงข้อมูลโดยตรงจากภายนอก
- **Getter/Setter methods** ใช้สำหรับการเข้าถึงและแก้ไขข้อมูลอย่างปลอดภัย
- **Data validation** ในส่วนของ setter methods ช่วยป้องกันข้อมูลผิด
- **Business logic methods** จัดกลุ่มฟังก์ชันที่เกี่ยวข้องไว้ในคลาส

ประโยชน์ของ Encapsulation

- ความปลอดภัยของข้อมูล (Data Security)
- การตรวจสอบข้อมูล (Data Validation)
- ง่ายต่อการดูแลรักษา (Maintainability)
- ลดความซับซ้อน (Complexity Reduction)

แบบฝึกหัดที่ 1 : ออกแบบ Class สำหรับบัญชีธนาคาร

- สร้างคลาส SavingsAccount ที่มีคุณสมบัติดังนี้
  - หมายเลขบัญชี (accountNumber)
  - ชื่อเจ้าของบัญชี (accountHolder)
  - ยอดเงิน (balance)
  - อัตราดอกเบี้ย (interestRate)
- ข้อกำหนด
  - ทุก attribute ต้องเป็น private
  - สร้าง getter methods สำหรับทุก attribute
  - สร้าง methods: deposit(), withdraw(), calculateInterest()
  - เพิ่ม validation ที่เหมาะสม

แบบฝึกหัดที่ 2 : ระบบจัดการนักเรียน

- สร้างคลาส Student สำหรับจัดเก็บข้อมูลนักเรียน
  - รหัสนักเรียน (studentId)
  - ชื่อ-สกุล (fullName)
  - อายุ (age)
  - เกรดเฉลี่ย (gpa)
  - สถานะการศึกษา (status: "Active", "Graduated", "Suspended")
- ข้อกำหนด
  - ใช้ Encapsulation อย่างเหมาะสม
  - เพิ่ม validation สำหรับทุกข้อมูล
  - สร้าง method สำหรับคำนวณเกรด และเปลี่ยนสถานะ

END.

Q & A