เข้าใจง่าย ใช้ได้จริง!
ลดเวลาสอน เพิ่มผลลัพธ์การเรียนรู้

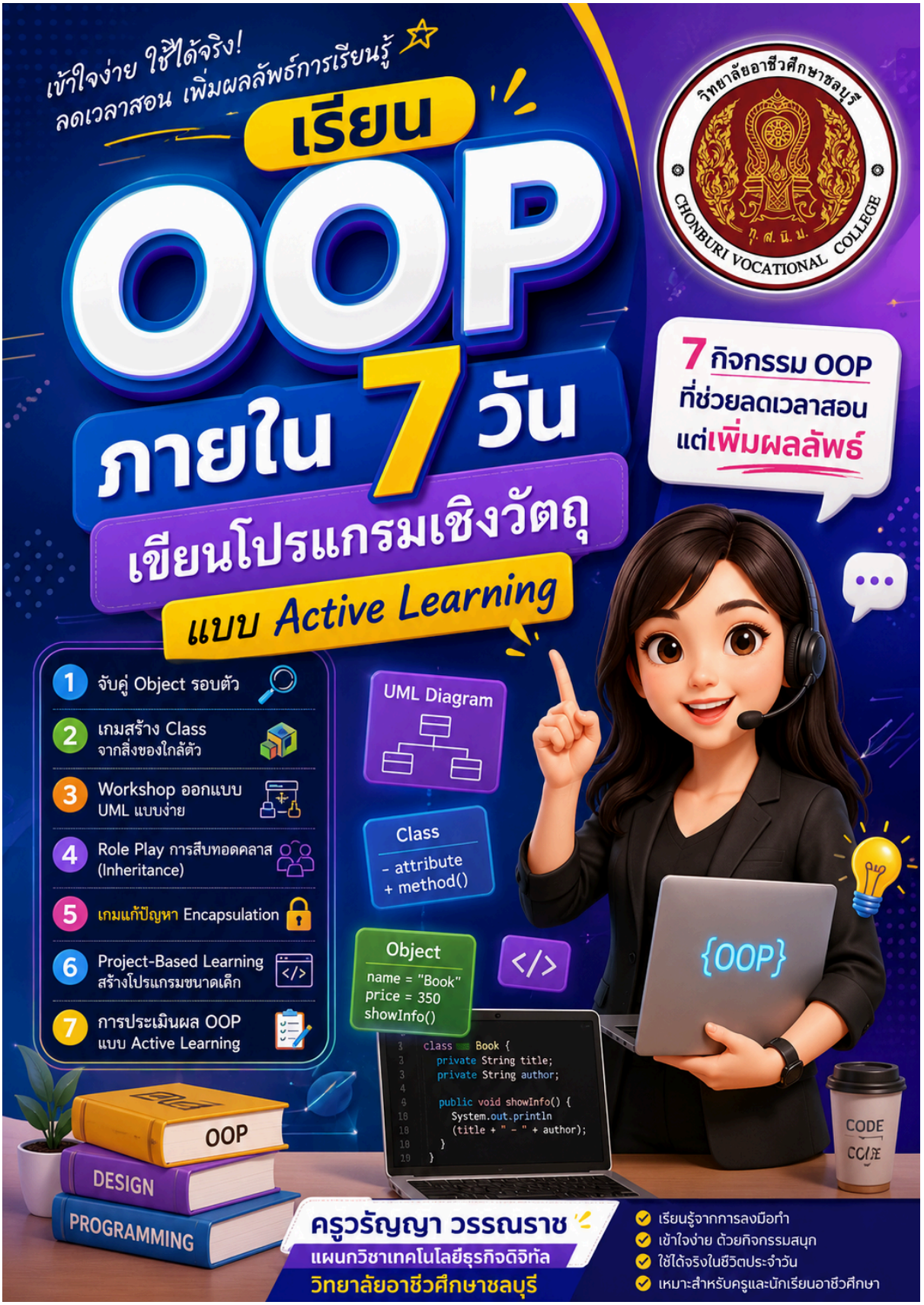
เรียน OOP ภายใน 7 วัน

เขียนโปรแกรมเชิงวัตถุ แบบ Active Learning

วิทยาลัยอาชีวศึกษาชลบุรี
CHONBURI VOCATIONAL COLLEGE

7 กิจกรรม OOP ที่ช่วยลดเวลาสอน แต่เพิ่มผลลัพธ์

1. จับคู่ Object รอบตัว
2. เกมสร้าง Class จากสิ่งของใกล้ตัว
3. Workshop ออกแบบ UML แบบง่าย
4. Role Play การสืบทอดคลาส (Inheritance)
5. เกมแก้ปัญหา Encapsulation
6. Project-Based Learning สร้างโปรแกรมขนาดเล็ก
7. การประเมินผล OOP แบบ Active Learning



ครูวรัญญา วรรณราช
แผนกวิชาเทคโนโลยีธุรกิจดิจิทัล
วิทยาลัยอาชีวศึกษาชลบุรี

- เรียนรู้จากการลงมือทำ
- เข้าใจง่าย ด้วยกิจกรรมสนุก
- ใช้ได้จริงในชีวิตประจำวัน
- เหมาะสำหรับครูและนักเรียนอาชีวศึกษา

เรียน OOP ภายใน 7 วัน

7 กิจกรรม OOP ที่ช่วยลดเวลาสอน แต่เพิ่มผลลัพธ์

คำนำ

- ทำไม OOP จึงเป็นเรื่องยากสำหรับผู้เรียน
- แนวคิด Teach Less, Learn More
- วิธีใช้ eBook เล่มนี้ให้เกิดประโยชน์สูงสุด

บทที่ 1 กิจกรรมจับคู่ Object รอบตัว

1.1 OOP คืออะไร เข้าใจใน 10 นาที

1.2 มองหา Object จากสิ่งรอบตัว

1.3 แยกคุณลักษณะ (Attribute) และพฤติกรรม (Method)

1.4 เกมจับคู่ Object ให้ถูกต้อง

1.5 สรุปแนวคิดเชิงวัตถุจากชีวิตประจำวัน

บทที่ 2 เกมสร้าง Class จากสิ่งของใกล้ตัว

2.1 ความสัมพันธ์ระหว่าง Class และ Object

2.2 ออกแบบ Class จากโทรศัพท์มือถือ

2.3 เกมสร้าง Class แข่งขันเป็นทีม

2.4 เขียน Class Diagram อย่างง่าย

2.5 แบบฝึกหัดสร้าง Class ด้วยตนเอง

บทที่ 3 Workshop ออกแบบ UML แบบง่าย

3.1 ทำความรู้จัก UML

3.2 ส่วนประกอบของ Class Diagram

3.3 ออกแบบ UML จากสถานการณ์จริง

3.4 Workshop วาด UML เป็นกลุ่ม

3.5 ตรวจสอบและปรับปรุงแบบจำลอง

บทที่ 4 กิจกรรม Role Play การสืบทอดคลาส (Inheritance)

4.1 แนวคิด Inheritance แบบเข้าใจง่าย

4.2 เกมบทบาทสมมติครอบครัวของคลาส

4.3 การสร้าง Parent Class และ Child Class

4.4 Workshop การสืบทอดคุณสมบัติ

4.5 สรุปข้อดีของ Inheritance

บทที่ 5 เกมแก้ปัญหา Encapsulation

5.1 ทำไมต้องซ่อนข้อมูล

5.2 เข้าใจ Encapsulation ผ่านสถานการณ์จริง

5.3 การใช้ Getter และ Setter

5.4 เกมปกป้องข้อมูลสำคัญ

5.5 แบบฝึกหัดตรวจสอบความเข้าใจ

บทที่ 6 Project-Based Learning สร้างโปรแกรมขนาดเล็ก

6.1 วางแผนโครงงาน OOP

6.2 วิเคราะห์ปัญหาและออกแบบคลาส

6.3 พัฒนาโปรแกรมด้วยแนวคิด OOP

6.4 ทดสอบและแก้ไขข้อผิดพลาด

6.5 นำเสนอผลงานและแลกเปลี่ยนเรียนรู้

บทที่ 7 การประเมินผล OOP แบบ Active Learning

7.1 การประเมินระหว่างเรียน

7.2 การใช้ Rubric วัดสมรรถนะ

7.3 การประเมินจากชิ้นงานจริง

7.4 การสะท้อนผลการเรียนรู้ (Reflection)

7.5 แนวทางพัฒนาการสอน OOP ในอนาคต

ภาคผนวก

- ใบกิจกรรมทั้ง 7 กิจกรรม
- ตัวอย่าง Rubric การประเมิน
- ตัวอย่างโครงงาน OOP
- แหล่งเรียนรู้เพิ่มเติม

บทที่ 1 : กิจกรรมจับคู่ Object รอบตัว

ปัญหา

เคยไหมครับ...

สอนเรื่อง OOP ไปตั้งนาน นักเรียนยังสับสนว่า Object คืออะไร? Class คืออะไร? บางคนจำคำศัพท์ได้ แต่พอให้ยกตัวอย่างกลับตอบไม่ได้

ปัญหานี้เกิดขึ้นบ่อยมาก เพราะ OOP เป็นเรื่องที่ค่อนข้างนามธรรม นักเรียนมักมองไม่เห็นภาพและรู้สึกว่าเป็นเรื่องไกลตัว ทำให้การเรียนกลายเป็นการท่องจำมากกว่าการเข้าใจ

ผลลัพธ์คือ...

- นักเรียนเบื่อ
- ครูต้องอธิบายซ้ำ
- เสียเวลาเรียน
- พื้นฐานไม่แน่น ส่งผลต่อบทเรียนถัดไป

แล้วจะทำอย่างไรให้นักเรียน "เห็น" Object ได้จริง?

ทำไมต้องแก้

ก่อนจะเขียนโค้ด OOP ได้ นักเรียนต้องเข้าใจแนวคิดพื้นฐานก่อนว่า

"ทุกสิ่งรอบตัวสามารถมองเป็น Object ได้"

ถ้านักเรียนเข้าใจจุดนี้ เขาจะสามารถเชื่อมโยงไปสู่

- Class
- Attribute
- Method
- Inheritance

- Encapsulation

ได้ง่ายขึ้นหลายเท่า

การเริ่มจากสิ่งใกล้ตัวจึงเป็นวิธีที่ช่วยให้นักเรียนเรียนรู้ได้เร็ว และครูไม่ต้องเสียเวลาอธิบายซ้ำหลายรอบ

นี่คือแนวคิดของ Teach Less, Learn More

ครูพูดน้อยลง แต่ผู้เรียนค้นพบด้วยตัวเองมากขึ้น

ขั้นตอนกิจกรรม

ขั้นที่ 1 แจกบัตรภาพ

เตรียมภาพสิ่งของต่าง ๆ เช่น

- โทรศัพท์มือถือ
- รถยนต์
- นักเรียน
- สุนัข
- พัดลม
- คอมพิวเตอร์

แจกให้นักเรียนแต่ละกลุ่ม

ขั้นที่ 2 ระดมความคิด

ให้นักเรียนตอบคำถาม 2 ข้อ

1. สิ่งนี้มีลักษณะอะไรบ้าง?
2. สิ่งนี้ทำอะไรได้บ้าง?

ขั้นที่ 3 แยก Attribute และ Method

ให้นักเรียนเขียนข้อมูลลงตาราง

Attribute	Method
สี	โทรออก
ยี่ห้อ	ถ่ายรูป
ราคา	เล่นเพลง

ขั้นที่ 4 จับคู่คำตอบ

แต่ละกลุ่มนำเสนอผลงาน

ครูช่วยชี้ให้เห็นว่า

Attribute = คุณสมบัติ

Method = ความสามารถ

ขั้นที่ 5 สรุปเป็น Object

เมื่อกิจกรรมจบ ให้นักเรียนสรุปว่า

"โทรศัพท์มือถือ 1 เครื่อง คือ Object"

และข้อมูลที่ค้นพบคือส่วนประกอบของ Object นั่นเอง

ตัวอย่าง

ตัวอย่างที่ 1 : โทรศัพท์มือถือ

Object :
โทรศัพท์มือถือ

Attribute :

- สี
- ยี่ห้อ
- ราคา
- ความจุ

Method :

- โทรออก
- ส่งข้อความ
- ถ่ายรูป
- เล่นเพลง

ตัวอย่างที่ 2 : นักเรียน

Object :
นักเรียน

Attribute :

- รหัสนักศึกษา
- ชื่อ
- อายุ
- สาขาวิชา

Method :

- เรียนหนังสือ
- ส่งงาน
- สอบ
- ทำกิจกรรม

ตัวอย่างที่ 3 : รถยนต์

Object :
รถยนต์

Attribute :

- **สี**
- **ยี่ห้อ**
- **รุ่น**
- **ทะเบียนรถ**

Method :

- **ขับเคลื่อน**
- **เบรก**
- **เปิดไฟ**
- **บีบแตร**

การบ้าน

Mission : นักสืบ Object

ให้นักเรียนเลือกสิ่งของรอบตัว 5 อย่าง

เช่น

- **กระเป๋า**
- **โน้ตบุ๊ก**
- **จักรยาน**
- **โทรทัศน์**
- **เครื่องปรับอากาศ**

แล้ววิเคราะห์

1. Object **คืออะไร**
2. **มี** Attribute **อะไรบ้างอย่างน้อย** 3 **ข้อ**
3. **มี** Method **อะไรบ้างอย่างน้อย** 3 **ข้อ**

จัดทำเป็น Mind Map หรือ Infographic 1 หน้า

สรุปบทเรียน

OOP ไม่ได้เริ่มต้นจากการเขียนโค้ด

แต่เริ่มจากการมองโลกเป็น "วัตถุ"

เมื่อผู้เรียนสามารถมองเห็น Object รอบตัวได้ เขาจะเข้าใจ Class, Attribute และ Method ได้ง่ายขึ้น และพร้อมก้าวสู่การพัฒนาโปรแกรมเชิงวัตถุในบทต่อไป

บทที่ 2 : เกมสร้าง Class จากสิ่งของใกล้ตัว

ปัญหา

หลังจากนักเรียนเริ่มเข้าใจว่า Object คืออะไรแล้ว ปัญหาที่มักเกิดขึ้นต่อมาคือ

"Object กับ Class ต่างกันอย่างไร?"

นักเรียนหลายคนเข้าใจว่า

- โทรศัพท์มือถือ = Class
- โทรศัพท์มือถือเครื่องนี้ = Object

แต่บางคนยังสับสนว่าทำไมต้องมี Class ก่อนถึงจะมี Object ได้

เมื่อไม่เข้าใจความสัมพันธ์ของ Class และ Object การเรียนเรื่องการสร้างโปรแกรมเชิงวัตถุจะยิ่งซับซ้อนขึ้น

ครูหลายคนจึงต้องใช้เวลานานในการอธิบายซ้ำหลายรอบ

ทำไมต้องแก้

Class ถือเป็นหัวใจสำคัญของ OOP

ถ้าเปรียบเทียบง่าย ๆ

Class คือ "แบบพิมพ์เขียว"

Object คือ "สิ่งที่สร้างขึ้นจากแบบพิมพ์เขียว"

เหมือนกับ

- แบบบ้าน = Class
- บ้านที่สร้างเสร็จแล้ว = Object

ถ้านักเรียนเข้าใจแนวคิดนี้ตั้งแต่ต้น

เขาจะสามารถออกแบบโปรแกรมได้ง่ายขึ้น

และพร้อมเรียนรู้เรื่อง

- Constructor
- Inheritance
- Encapsulation
- Polymorphism

ในบทต่อ ๆ ไป

ขั้นตอนกิจกรรม

ขั้นที่ 1 แบ่งกลุ่ม

แบ่งนักเรียนออกเป็นกลุ่มละ 4-5 คน

แจกภาพสิ่งของให้แต่ละกลุ่ม เช่น

- รถยนต์
- โทรศัพท์มือถือ
- โน้ตบุ๊ก
- นักเรียน
- สัตว์เลี้ยง

ขั้นที่ 2 ระดมสมอง

ให้แต่ละกลุ่มตอบคำถาม

"ถ้าจะสร้างสิ่งนี้ในโปรแกรม ต้องเก็บข้อมูลอะไรบ้าง?"

เช่น โทรศัพท์มือถือ

- ยี่ห้อ
- รุ่น
- สี
- ราคา

ขั้นที่ 3 สร้าง Class

ให้แต่ละกลุ่มออกแบบ Class ลงในกระดาษ

ตัวอย่าง

Class : Smartphone

Attributes

- brand
- model
- color
- price

Methods

- call()
- takePhoto()
- playMusic()

ขั้นที่ 4 สร้าง Object

หลังจากได้ Class แล้ว

ให้นักเรียนสร้าง Object อย่างน้อย 3 ตัว

เช่น

Object 1

iPhone 15

Object 2

Samsung Galaxy S25

Object 3

Xiaomi 15

ขั้นที่ 5 นำเสนอผลงาน

แต่ละกลุ่มอธิบายว่า

- Class ของตนคืออะไร
- Attribute มีอะไรบ้าง
- Method มีอะไรบ้าง
- Object ที่สร้างขึ้นมีอะไรบ้าง

ครูช่วยเสริมและเชื่อมโยงแนวคิด OOP ให้ชัดเจน

ตัวอย่าง

ตัวอย่างที่ 1 : Class รถยนต์

Class : Car

Attributes

- color
- brand

- model
- speed

Methods

- startEngine()
- accelerate()
- brake()

Objects

- Toyota Yaris สีขาว
- Honda City สีดำ
- Mazda 2 สีแดง

ตัวอย่างที่ 2 : Class นักศึกษา

Class : Student

Attributes

- studentID
- name
- major
- age

Methods

- study()
- submitWork()
- takeExam()

Objects

- นายสมชาย
- นางสาวสมหญิง
- นายธนพล

ตัวอย่างที่ 3 : Class สุนัข

Class : Dog

Attributes

- name
- breed
- age
- color

Methods

- bark()
- run()
- eat()

Objects

- โชคดี
- เจ้าดำ
- มะลิ

การบ้าน

Mission : ออกแบบ Class รอบตัว

ให้นักเรียนเลือกสิ่งของหรือบุคคลรอบตัว 3 อย่าง

ตัวอย่าง

- ครู
- รถจักรยานยนต์
- ร้านกาแฟ
- คอมพิวเตอร์
- โทรศัพท์มือถือ

สำหรับแต่ละสิ่ง ให้สร้าง

1. ชื่อ Class
2. Attribute อย่างน้อย 5 รายการ
3. Method อย่างน้อย 3 รายการ
4. Object อย่างน้อย 2 ตัว

จัดทำเป็นแผนภาพ Class Diagram แบบง่าย

Challenge พิเศษ

ให้นักเรียนลองเขียนโค้ดจำลอง Class ที่ออกแบบด้วยภาษาใดก็ได้ เช่น

- Java
- C#
- Python
- PHP

เพื่อเตรียมพร้อมสำหรับการเขียนโปรแกรมจริงในบทถัดไป

สรุปบทเรียน

Object คือสิ่งที่เราใช้งานจริง

แต่ก่อนจะมี Object ได้ เราต้องมี Class ก่อนเสมอ

Class เปรียบเสมือนแม่พิมพ์หรือแบบพิมพ์เขียว

ส่วน Object คือสิ่งที่ถูกสร้างขึ้นจากแม่พิมพ์นั้น

เมื่อเข้าใจความสัมพันธ์ระหว่าง Class และ Object แล้ว การเรียน OOP จะง่ายขึ้นมาก และพร้อมต่อยอดไปสู่การออกแบบระบบและการเขียนโปรแกรมในโลกจริง

บทที่ 3 : Workshop ออกแบบ UML แบบง่าย

ปัญหา

เคยสังเกตไหมว่า...

นักเรียนหลายคนสามารถบอกได้ว่า Class มีอะไรบ้าง แต่พอให้เริ่มเขียนโปรแกรมจริงกลับไม่รู้ว่าจะเริ่มตรงไหน

บางคนเขียนโค้ดก่อนคิด

บางคนเขียนไปแก้ไป

บางคนเพิ่ม Class ใหม่ตลอดเวลา จนโปรแกรมเริ่มสับสน

สาเหตุสำคัญคือ

"นักเรียนยังมองภาพรวมของระบบไม่ออก"

เปรียบเหมือนการสร้างบ้านโดยไม่มีแบบแปลน

สุดท้ายอาจสร้างได้ แต่เสียเวลา แก้งานบ่อย และเกิดข้อผิดพลาดจำนวนมาก

ทำไมต้องแก้

โปรแกรมเมอร์มืออาชีพส่วนใหญ่ไม่ได้เริ่มจากการเขียนโค้ด

แต่เริ่มจากการออกแบบก่อน

UML (Unified Modeling Language)

คือเครื่องมือที่ช่วยให้เราเห็นภาพของระบบก่อนลงมือเขียนโปรแกรมจริง

ข้อดีของ UML คือ

✅ มองเห็นโครงสร้างระบบชัดเจน

✅ ลดความผิดพลาดในการเขียนโปรแกรม

✅ สื่อสารกับทีมได้ง่าย

✅ ช่วยวางแผนก่อนเริ่มพัฒนา

เมื่อผู้เรียนเข้าใจ UML

จะสามารถเชื่อมโยงแนวคิด

- Class
- Object
- Attribute
- Method

เข้าด้วยกันได้อย่างเป็นระบบ

ขั้นตอนกิจกรรม

ขั้นที่ 1 เลือกสถานการณ์ใกล้ตัว

ให้แต่ละกลุ่มเลือกหัวข้อ 1 เรื่อง

ตัวอย่าง

- ระบบร้านกาแฟ
- ระบบห้องสมุด

- ระบบเช่ายานพาหนะ
- ระบบร้านอาหาร
- ระบบยืมอุปกรณ์การเรียน

เลือกเรื่องที่ทุกคนเข้าใจง่ายและพบเห็นในชีวิตประจำวัน

ขั้นที่ 2 ระดมสมองหา Class

ให้สมาชิกช่วยกันคิดว่า

ในระบบนี้มีอะไรบ้าง

เช่น ระบบห้องสมุด

อาจมี

- หนังสือ
- สมาชิก
- บรรณารักษ์
- การยืมหนังสือ

เขียนทุกอย่างลงบนกระดาษ Post-it

ขั้นที่ 3 แยก Attribute และ Method

สำหรับแต่ละ Class

ให้นักเรียนระบุ

Attribute

คือข้อมูลที่เก็บ

Method

คือสิ่งที่ทำได้

ตัวอย่าง

Class : Book

Attribute

- bookID
- title
- author

Method

- borrow()
- returnBook()

ขั้นที่ 4 วาด UML Diagram

ให้นักเรียนวาดกล่อง UML

แบ่งออกเป็น 3 ส่วน

ส่วนที่ 1

ชื่อ Class

ส่วนที่ 2

Attribute

ส่วนที่ 3

Method

จากนั้นเชื่อมความสัมพันธ์ระหว่าง Class

ขั้นที่ 5 นำเสนอและแลกเปลี่ยนความคิดเห็น

แต่ละกลุ่มอธิบาย UML ของตนเอง

พร้อมตอบคำถาม

- **ทำไมเลือก Class นี้**
- **มีความสัมพันธ์กันอย่างไร**
- **ถ้าพัฒนาต่อจะเพิ่มอะไรได้อีก**

ครูทำหน้าที่เป็นโค้ช มากกว่าผู้บรรยาย

ตัวอย่าง

ตัวอย่าง : ระบบห้องสมุด

Class : Book

Attribute

- bookID
- title
- author

Method

- borrow()
- returnBook()

Class : Member

Attribute

- memberID
- name
- phone

Method

- register()

- borrowBook()

Class : Librarian

Attribute

- employeeID
- name

Method

- addBook()
- removeBook()

ตัวอย่าง UML แบบง่าย

Book

bookID
title
author

borrow()
returnBook()

Member

memberID
name
phone

register()
borrowBook()

Librarian

employeeID
name

addBook()
removeBook()

การบ้าน

Mission : ออกแบบ UML จากชีวิตจริง

ให้นักเรียนเลือก 1 สถานการณ์ต่อไปนี้

- ร้านสะดวกซื้อ
- ร้านกาแฟ
- โรงเรียน
- โรงพยาบาล
- ระบบจองตั๋วภาพยนตร์

จากนั้นออกแบบ

1. Class อย่างน้อย 4 Class
2. Attribute ของแต่ละ Class อย่างน้อย 3 รายการ
3. Method ของแต่ละ Class อย่างน้อย 2 รายการ
4. วาด UML Diagram ให้สมบูรณ์

Challenge พิเศษ

ให้นักเรียนนำ UML ที่ออกแบบ

ไปทดลองสร้างเป็นโค้ดจริงด้วยภาษา

- Java
- Python

- C#
- PHP

เพื่อเปรียบเทียบว่า

"สิ่งที่ออกแบบไว้" กับ

"สิ่งที่เขียนจริง"

สอดคล้องกันมากน้อยแค่ไหน

สรุปบทเรียน

UML เปรียบเสมือนแบบแปลนก่อนสร้างบ้าน

ยิ่งวางแผนดีเท่าไร

การเขียนโปรแกรมก็ยิ่งง่ายขึ้นเท่านั้น

นักพัฒนาที่เก่งไม่ได้เริ่มจากการเขียนโค้ดเร็วที่สุด

แต่เริ่มจากการออกแบบที่ชัดเจนที่สุด

เมื่อผู้เรียนสามารถสร้าง UML ได้ด้วยตนเอง

เขาจะมองเห็นภาพรวมของระบบ เข้าใจความสัมพันธ์ของ Class และพร้อมก้าวสู่การพัฒนาโปรแกรมเชิงวัตถุอย่างมืออาชีพในบทต่อไป

บทที่ 4 : กิจกรรม Role Play การสืบทอดคลาส (Inheritance)

ปัญหา

เมื่อเริ่มเรียนเรื่อง Inheritance

นักเรียนมักเกิดคำถามว่า

- ทำไมต้องสืบทอดคลาส?
- สร้าง Class ใหม่ไม่ได้หรือ?
- Parent Class กับ Child Class ต่างกันอย่างไร?
- ทำไมข้อมูลบางอย่างถึงใช้ร่วมกันได้?

หลายคนจำคำศัพท์ได้ แต่ไม่เข้าใจแนวคิดจริง ๆ

เมื่อถึงเวลาลงมือเขียนโค้ด

ก็ไม่รู้ว่าควรใช้ Inheritance ตอนไหน

ผลลัพธ์คือ

- เขียนโค้ดซ้ำจำนวนมาก
- ออกแบบ Class ไม่เป็นระบบ

- ใช้ OOP ได้ไม่เต็มประสิทธิภาพ

ทำไมต้องแก้

ลองนึกภาพว่า

เรามีข้อมูลเกี่ยวกับสัตว์หลายชนิด

- สุนัข
- แมว
- นก

ทุกตัวมีคุณสมบัติเหมือนกัน เช่น

- ชื่อ
- อายุ
- น้ำหนัก

ถ้าสร้างข้อมูลซ้ำทุก Class

จะเสียเวลาและดูแลยาก

Inheritance จึงเข้ามาช่วย

โดยให้ Class ลูกนำคุณสมบัติและความสามารถจาก Class แม่ไปใช้ต่อได้

ข้อดีคือ

✅ ลดการเขียนโค้ดซ้ำ

✅ จัดระบบข้อมูลได้ดีขึ้น

✅ แก้ไขง่าย

✅ ขยายระบบในอนาคตได้สะดวก

นี่คือแนวคิดสำคัญที่โปรแกรมเมอร์มืออาชีพใช้กันทุกวัน

ขั้นตอนกิจกรรม

ขั้นที่ 1 เลือกครอบครัว Class

ครูแบ่งนักเรียนเป็นกลุ่ม

แต่ละกลุ่มได้รับหัวข้อ เช่น

- สัตว์
- ยานพาหนะ
- บุคลากรในโรงเรียน
- อุปกรณ์อิเล็กทรอนิกส์

ขั้นที่ 2 สร้าง Parent Class

ให้นักเรียนช่วยกันคิดว่า

สิ่งใดเป็นคุณสมบัติร่วมกัน

ตัวอย่าง

Class : Animal

Attribute

- name
- age
- weight

Method

- eat()
- sleep()

ขั้นที่ 3 เล่นบทบาทสมมติ

ให้นักเรียน 1 คนรับบทเป็น Parent Class

เช่น Animal

จากนั้นให้นักเรียนอีก 3 คนรับบท

- Dog
- Cat
- Bird

นักเรียนที่เป็น Parent Class จะถือป้ายแสดงคุณสมบัติพื้นฐาน

ส่วน Child Class จะได้รับคุณสมบัติเหล่านั้นไปใช้

พร้อมเพิ่มความสามารถเฉพาะของตนเอง

ขั้นที่ 4 เพิ่มคุณสมบัติพิเศษ

ตัวอย่าง

Dog

- bark()

Cat

- climb()

Bird

- fly()

ให้นักเรียนแสดงท่าทางประกอบ

เพื่อสร้างความเข้าใจผ่านการเคลื่อนไหว

ขั้นที่ 5 สรุปแนวคิด

ครูถามคำถาม

- Child Class ได้อะไรจาก Parent Class?
- Child Class เพิ่มอะไรได้บ้าง?
- ถ้าไม่มี Parent Class จะเกิดอะไรขึ้น?

ให้นักเรียนเป็นผู้ตอบและสรุปเอง

ตัวอย่าง

ตัวอย่างที่ 1 : บุคลากรในโรงเรียน

Parent Class

Person

Attribute

- name
- age
- gender

Method

- walk()
- speak()

Child Class

Teacher

Method เพิ่มเติม

- teach()

Student

Method เพิ่มเติม

- study()

Director

Method เพิ่มเติม

- manage()

ตัวอย่างที่ 2 : ยานพาหนะ

Parent Class

Vehicle

Attribute

- brand
- color
- speed

Method

- move()

Car

Method เพิ่มเติม

- openDoor()

Motorcycle

Method เพิ่มเติม

- balance()

Truck

Method เพิ่มเติม

- loadGoods()

ตัวอย่าง UML แบบง่าย

Vehicle

brand
color
speed

move()

↑

Car

openDoor()

Motorcycle

balance()

Truck

loadGoods()

การบ้าน

Mission : สร้างครอบครัว Class

ให้นักเรียนเลือก 1 หัวข้อ

- สัตว์
- กีฬา
- อาชีพ
- เครื่องใช้ไฟฟ้า
- เกมออนไลน์

จากนั้นออกแบบ

1. Parent Class 1 Class
2. Child Class อย่างน้อย 3 Class
3. Attribute ของ Parent Class อย่างน้อย 3 รายการ
4. Method ของ Parent Class อย่างน้อย 2 รายการ
5. Method เฉพาะของ Child Class อย่างน้อย 2 รายการ

พร้อมวาด UML Diagram

Challenge พิเศษ

ให้นักเรียนเขียนโค้ดจำลอง Inheritance

ตัวอย่างภาษา

- Java
- Python
- PHP
- C#

โดยแสดงให้เห็นว่า

Child Class สามารถเรียกใช้ข้อมูลจาก Parent Class ได้จริง

สรุปบทเรียน

Inheritance คือการสืบทอดคุณสมบัติและพฤติกรรมจาก Class แม่ไปยัง Class ลูก

แนวคิดนี้ช่วยลดการเขียนโค้ดซ้ำ

ทำให้ระบบเป็นระเบียบ

และสามารถขยายโปรแกรมในอนาคตได้ง่ายขึ้น

จำไว้ว่า

"โปรแกรมเมอร์ที่เก่ง ไม่ได้เขียนโค้ดเยอะที่สุด

แต่เขียนโค้ดซ้ำน้อยที่สุด"

และ Inheritance คือหนึ่งในเครื่องมือสำคัญที่ช่วยให้เราทำสิ่งนั้นได้

บทที่ 4 : กิจกรรม Role Play การสืบทอดคลาส (Inheritance)

ปัญหา

เมื่อเริ่มเรียนเรื่อง Inheritance

นักเรียนมักเกิดคำถามว่า

- ทำไมต้องสืบทอดคลาส?
- สร้าง Class ใหม่ไม่ได้หรือ?
- Parent Class กับ Child Class ต่างกันอย่างไร?
- ทำไมข้อมูลบางอย่างถึงใช้ร่วมกันได้?

หลายคนจำคำศัพท์ได้ แต่ไม่เข้าใจแนวคิดจริง ๆ

เมื่อถึงเวลาลงมือเขียนโค้ด

ก็ไม่รู้ว่าควรใช้ Inheritance ตอนไหน

ผลลัพธ์คือ

- เขียนโค้ดซ้ำจำนวนมาก
- ออกแบบ Class ไม่เป็นระบบ
- ใช้ OOP ได้ไม่เต็มประสิทธิภาพ

ทำไมต้องแก้

ลองนึกภาพว่า

เรามีข้อมูลเกี่ยวกับสัตว์หลายชนิด

- สุนัข
- แมว
- นก

ทุกตัวมีคุณสมบัติเหมือนกัน เช่น

- ชื่อ
- อายุ
- น้ำหนัก

ถ้าสร้างข้อมูลซ้ำทุก Class

จะเสียเวลาและดูแลยาก

Inheritance จึงเข้ามาช่วย

โดยให้ Class ลูกนำคุณสมบัติและความสามารถจาก Class แม่ไปใช้ต่อได้

ข้อดีคือ

✅ ลดการเขียนโค้ดซ้ำ

✅ จัดระบบข้อมูลได้ดีขึ้น

✅ แก้ไขง่าย

✅ ขยายระบบในอนาคตได้สะดวก

นี่คือแนวคิดสำคัญที่โปรแกรมเมอร์มืออาชีพใช้กันทุกวัน

ขั้นตอนกิจกรรม

ขั้นที่ 1 เลือกครอบครัว Class

ครูแบ่งนักเรียนเป็นกลุ่ม

แต่ละกลุ่มได้รับหัวข้อ เช่น

- สัตว์
- ยานพาหนะ
- บุคลากรในโรงเรียน
- อุปกรณ์อิเล็กทรอนิกส์

ขั้นที่ 2 สร้าง Parent Class

ให้นักเรียนช่วยกันคิดว่า

สิ่งใดเป็นคุณสมบัติร่วมกัน

ตัวอย่าง

Class : Animal

Attribute

- name
- age
- weight

Method

- eat()
- sleep()

ขั้นที่ 3 เล่นบทบาทสมมติ

ให้นักเรียน 1 คนรับบทเป็น Parent Class

เช่น Animal

จากนั้นให้นักเรียนอีก 3 คนรับบท

- Dog
- Cat
- Bird

นักเรียนที่เป็น Parent Class จะถือป้ายแสดงคุณสมบัติพื้นฐาน

ส่วน Child Class จะได้รับคุณสมบัติเหล่านั้นไปใช้

พร้อมเพิ่มความสามารถเฉพาะของตนเอง

ขั้นที่ 4 เพิ่มคุณสมบัติพิเศษ

ตัวอย่าง

Dog

- bark()

Cat

- climb()

Bird

- fly()

ให้นักเรียนแสดงท่าทางประกอบ

เพื่อสร้างความเข้าใจผ่านการเคลื่อนไหว

ขั้นที่ 5 สรุปแนวคิด

ครูถามคำถาม

- Child Class ได้อะไรจาก Parent Class?
- Child Class เพิ่มอะไรได้บ้าง?
- ถ้าไม่มี Parent Class จะเกิดอะไรขึ้น?

ให้นักเรียนเป็นผู้ตอบและสรุปเอง

ตัวอย่าง

ตัวอย่างที่ 1 : บุคลากรในโรงเรียน

Parent Class

Person

Attribute

- name
- age
- gender

Method

- walk()
- speak()

Child Class

Teacher

Method เพิ่มเติม

- teach()

Student

Method เพิ่มเติม

- study()

Director

Method เพิ่มเติม

- manage()

ตัวอย่างที่ 2 : ยานพาหนะ

Parent Class

Vehicle

Attribute

- brand
- color
- speed

Method

- move()

Car

Method เพิ่มเติม

- openDoor()

Motorcycle

Method เพิ่มเติม

- balance()

Truck

Method เพิ่มเติม

- loadGoods()

ตัวอย่าง UML แบบง่าย

Vehicle

brand
color
speed

move()

↑

Car

openDoor()

Motorcycle

balance()

Truck

loadGoods()

การบ้าน

Mission : สร้างครอบครัว Class

ให้นักเรียนเลือก 1 หัวข้อ

- **สัตว์**
- **กีฬา**
- **อาชีพ**
- **เครื่องใช้ไฟฟ้า**
- **เกมออนไลน์**

จากนั้นออกแบบ

1. Parent Class 1 Class
2. Child Class อย่างน้อย 3 Class
3. Attribute ของ Parent Class อย่างน้อย 3 รายการ
4. Method ของ Parent Class อย่างน้อย 2 รายการ
5. Method เฉพาะของ Child Class อย่างน้อย 2 รายการ

พร้อมวาด UML Diagram

Challenge พิเศษ

ให้นักเรียนเขียนโค้ดจำลอง Inheritance

ตัวอย่างภาษา

- Java
- Python
- PHP
- C#

โดยแสดงให้เห็นว่า

Child Class สามารถเรียกใช้ข้อมูลจาก Parent Class ได้จริง

สรุปบทเรียน

Inheritance คือการสืบทอดคุณสมบัติและพฤติกรรมจาก Class แม่ไปยัง Class ลูก

แนวคิดนี้ช่วยลดการเขียนโค้ดซ้ำ

ทำให้ระบบเป็นระเบียบ

และสามารถขยายโปรแกรมในอนาคตได้ง่ายขึ้น

จำไว้ว่า

"โปรแกรมเมอร์ที่เก่ง ไม่ได้เขียนโค้ดเยอะที่สุด

แต่เขียนโค้ดซ้ำน้อยที่สุด"

และ Inheritance คือหนึ่งในเครื่องมือสำคัญที่ช่วยให้เราทำสิ่งนั้นได้

บทที่ 5 : เกมแก้ปัญหา Encapsulation

ปัญหา

เมื่อเริ่มเรียน OOP หลายคนมักสงสัยว่า

"ทำไมต้องซ่อนข้อมูล?"

"ทำไมไม่ให้ทุกคนเข้าถึงข้อมูลได้เลย?"

"Getter และ Setter มีไว้ทำอะไร?"

นักเรียนจำนวนมากมองว่า Encapsulation เป็นเรื่องยุ่งยาก

เพราะต้องกำหนด Private, Public และสร้าง Method เพิ่มเติม

ทำให้รู้สึกว่าเขียนโค้ดมากขึ้นโดยไม่เห็นประโยชน์

ผลที่เกิดขึ้นคือ

- เข้าถึงข้อมูลโดยตรง
- แก้ไขข้อมูลผิดพลาด

- โปรแกรมเกิดข้อผิดพลาดง่าย
- ข้อมูลสำคัญขาดความปลอดภัย

ทำไมต้องแก้

ลองนึกถึงบัญชีธนาคารของเรา

ถ้าทุกคนสามารถเข้าไปเปลี่ยนยอดเงินในบัญชีได้โดยตรง

จะเกิดอะไรขึ้น?

แน่นอนว่าเกิดปัญหาแน่นอน

ในโลกของโปรแกรมก็เช่นกัน

ข้อมูลบางอย่างไม่ควรถูกแก้ไขโดยตรง

Encapsulation จึงถูกสร้างขึ้นมาเพื่อ

✅ ปกป้องข้อมูลสำคัญ

✅ ควบคุมการเข้าถึงข้อมูล

✅ ลดข้อผิดพลาดจากผู้ใช้งาน

✅ ทำให้โปรแกรมมีความปลอดภัยมากขึ้น

แนวคิดสำคัญคือ

"ซ่อนสิ่งที่ไม่จำเป็นต้องรู้ และเปิดเฉพาะสิ่งที่ควรรู้"

ขั้นตอนกิจกรรม

ขั้นที่ 1 เกมกล่องปริศนา

ครูเตรียมกล่องปิดทึบจำนวน 1 กล่อง

ภายในใส่กระดาษคำตอบไว้

เช่น

- จำนวนเงิน
- รหัสผ่าน
- คะแนนสอบ

นักเรียนจะไม่สามารถเปิดกล่องได้โดยตรง

ขั้นที่ 2 กำหนดกติกา

นักเรียนสามารถสอบถามข้อมูลได้

ผ่านคำถามที่ครูกำหนดเท่านั้น

เช่น

- คะแนนมากกว่า 80 หรือไม่?
- มีเงินเพียงพอหรือไม่?

แต่ไม่สามารถเปิดดูข้อมูลจริงได้

ขั้นที่ 3 เชื่อมโยงกับ OOP

ครูอธิบายว่า

กล่องปิด = Private

ช่องทางสอบถาม = Public Method

ข้อมูลภายใน = Attribute

นักเรียนจะเริ่มมองเห็นว่า

Encapsulation คือการป้องกันข้อมูลไม่ให้ถูกเข้าถึงโดยตรง

ขั้นที่ 4 เกมผู้พิทักษ์ข้อมูล

แบ่งนักเรียนเป็นกลุ่ม

แต่ละกลุ่มได้รับสถานการณ์

- **ระบบธนาคาร**
- **ระบบโรงพยาบาล**
- **ระบบทะเบียนนักเรียน**
- **ระบบร้านค้าออนไลน์**

ให้ออกแบบว่า

ข้อมูลใดควรเป็น Private

ข้อมูลใดควรเป็น Public

ขั้นที่ 5 สรุปบทเรียน

ให้แต่ละกลุ่มนำเสนอเหตุผล

ว่าทำไมข้อมูลบางอย่างจึงต้องถูกปกป้อง

และถ้าไม่มี Encapsulation จะเกิดปัญหาอะไร

ตัวอย่าง

ตัวอย่างที่ 1 : บัญชีธนาคาร

Class : BankAccount

Attribute

- accountNumber
- ownerName
- balance

Method

- deposit()
- withdraw()
- getBalance()

ในกรณีนี้

balance ควรถูกกำหนดเป็น Private

เพื่อไม่ให้ใครเปลี่ยนยอดเงินโดยตรง

ตัวอย่างที่ 2 : ระบบนักเรียน

Class : Student

Attribute

- studentID
- name
- GPA

Method

- getGPA()
- updateGPA()

GPA ควรถูกป้องกัน

เพราะไม่ควรให้ผู้ใช้งานแก้ไขคะแนนเฉลี่ยเองได้

ตัวอย่างที่ 3 : ร้านค้าออนไลน์

Class : Product

Attribute

- productID
- productName

- stock

Method

- addStock()
- reduceStock()
- getStock()

Stock ไม่ควรถูกเปลี่ยนค่าโดยตรง

เพื่อป้องกันข้อมูลคลาดเคลื่อน

การบ้าน

Mission : นักออกแบบระบบรักษาความปลอดภัย

ให้นักเรียนเลือก 1 ระบบ

- ระบบธนาคาร
- ระบบโรงเรียน
- ระบบร้านอาหาร
- ระบบโรงพยาบาล
- ระบบร้านค้าออนไลน์

จากนั้นออกแบบ

1. Class อย่างน้อย 2 Class
2. Attribute อย่างน้อย 5 รายการ
3. ระบุว่า Attribute ใดเป็น Private
4. สร้าง Getter และ Setter ที่เหมาะสม
5. อธิบายเหตุผลในการปกป้องข้อมูล

Challenge พิเศษ

เขียนโปรแกรมจำลอง Encapsulation

โดยใช้ภาษา

- Java
- Python
- PHP
- C#

ให้สามารถ

- อ่านข้อมูลผ่าน Getter
- แก้ไขข้อมูลผ่าน Setter
- ป้องกันการเข้าถึงข้อมูลโดยตรง

สรุปบทเรียน

Encapsulation คือการห่อหุ้มและปกป้องข้อมูลภายใน Class

ไม่ใช่เพื่อทำให้การเขียนโปรแกรมยากขึ้น

แต่เพื่อทำให้ระบบปลอดภัยและน่าเชื่อถือมากขึ้น

จำไว้ว่า

"ข้อมูลที่สำคัญ ไม่ควรเปิดให้ทุกคนแก้ไขได้"

โปรแกรมที่ดีไม่ใช่โปรแกรมที่เปิดทุกอย่างให้ใช้งาน

แต่คือโปรแกรมที่ควบคุมการเข้าถึงข้อมูลได้อย่างเหมาะสม

และนี่คือหัวใจสำคัญของ Encapsulation ในโลก OOP

บทที่ 6 : Project-Based Learning สร้างโปรแกรมขนาดเล็ก

ปัญหา

เคยไหม...

นักเรียนเรียนเรื่อง OOP มาหลายสัปดาห์

- รู้จัก Object
- รู้จัก Class
- เข้าใจ Inheritance
- เข้าใจ Encapsulation

แต่พอถามว่า

"สามารถสร้างโปรแกรมอะไรได้บ้าง?"

กลับตอบไม่ได้

หลายคนจำทฤษฎีได้

แต่ยังไม่สามารถเชื่อมโยงความรู้ทั้งหมดเข้าด้วยกัน

ผลที่เกิดขึ้นคือ

- เรียนเพื่อสอบ
- จำได้ชั่วคราว
- ไม่เห็นประโยชน์ของ OOP
- ขาดความมั่นใจในการเขียนโปรแกรม

ทำไมต้องแก้

การเรียนรู้ที่ดีที่สุด

ไม่ใช่การฟัง

ไม่ใช่การอ่าน

แต่คือการลงมือทำจริง

Project-Based Learning หรือ PBL

คือการเรียนรู้ผ่านการสร้างผลงาน

นักเรียนจะได้

✅ คิดวิเคราะห์

✅ วางแผน

✅ แก้ปัญหา

✅ ทำงานร่วมกัน

✅ สร้างโปรแกรมจริง

เมื่อผู้เรียนได้สร้างผลงานของตนเอง

ความรู้จะไม่ใช่แค่ทฤษฎีอีกต่อไป

แต่จะกลายเป็นทักษะที่ใช้งานได้จริง

ขั้นตอนกิจกรรม

ขั้นที่ 1 เลือกหัวข้อโครงงาน

ให้นักเรียนเลือกปัญหาหรือสิ่งที่สนใจ

ตัวอย่าง

- ระบบจัดการรายชื่อนักเรียน
- ระบบร้านกาแฟ
- ระบบห้องสมุด
- ระบบเช็คชื่อเข้าเรียน
- ระบบขายสินค้าออนไลน์

เลือกหัวข้อที่สามารถทำได้ภายในเวลาเรียน

ขั้นที่ 2 วิเคราะห์และออกแบบระบบ

ให้นักเรียนตอบคำถาม

- ระบบนี้มีใครใช้งานบ้าง?
- ต้องเก็บข้อมูลอะไร?
- มี Class อะไรบ้าง?
- แต่ละ Class ทำหน้าที่อะไร?

จากนั้นออกแบบ UML Diagram

ก่อนเริ่มเขียนโปรแกรม

ขั้นที่ 3 พัฒนาโปรแกรม

เริ่มสร้าง Class ต่าง ๆ

เช่น

- Student
- Product
- Customer
- Order

ใช้แนวคิด OOP ที่เรียนมา

- Object
- Class
- Inheritance
- Encapsulation

ให้ครบถ้วน

ขั้นที่ 4 ทดสอบระบบ

ให้นักเรียนทดลองใช้งานโปรแกรม

พร้อมตอบคำถาม

- โปรแกรมทำงานถูกต้องหรือไม่?
- มีข้อผิดพลาดตรงไหน?
- ควรปรับปรุงอะไรเพิ่มเติม?

ขั้นที่ 5 นำเสนอผลงาน

แต่ละกลุ่มนำเสนอ

- แนวคิดของระบบ
- UML Diagram
- ตัวอย่างการทำงาน
- ปัญหาที่พบ
- สิ่งที่ได้เรียนรู้

เน้นให้ผู้เรียนเป็นผู้อธิบาย

ครูเป็นผู้ตั้งคำถามและให้คำแนะนำ

ตัวอย่าง

ตัวอย่างโครงงาน : ระบบจัดการรายชื่อนักเรียน

Class : Student

Attribute

- studentID
- name
- major

Method

- addStudent()
- editStudent()
- deleteStudent()

Class : Course

Attribute

- courseID
- courseName

Method

- addCourse()
- removeCourse()

Class : Teacher

Attribute

- teacherID
- teacherName

Method

- assignCourse()

คุณสมบัติของระบบ

- **เพิ่มข้อมูลนักเรียน**
- **แก้ไขข้อมูลนักเรียน**
- **ลบข้อมูลนักเรียน**
- **ค้นหาข้อมูลนักเรียน**
- **แสดงรายชื่อนักเรียนทั้งหมด**

แนวคิด OOP ที่ใช้

Object

- **นักเรียนแต่ละคน**

Class

- Student

Encapsulation

- **ป้องกันการแก้ไขข้อมูลโดยตรง**

Inheritance

- Student และ Teacher สืบทอดจาก Person

การบ้าน

Mission : Mini OOP Project

ให้นักเรียนสร้างโปรแกรมขนาดเล็ก 1 โปรแกรม

เลือกหัวข้อใดหัวข้อหนึ่ง

- **ระบบร้านอาหาร**
- **ระบบจองห้องประชุม**
- **ระบบจัดการหนังสือ**
- **ระบบเช็คชื่อเรียน**
- **ระบบขายสินค้า**

เงื่อนไข

1. มีอย่างน้อย 3 Class
2. มี Object อย่างน้อย 5 Object
3. มี Encapsulation อย่างน้อย 1 จุด
4. มี Inheritance อย่างน้อย 1 จุด
5. มี UML Diagram ประกอบ

Challenge พิเศษ

ให้นักเรียนเพิ่ม

- **เมนูใช้งาน**
- **การค้นหาข้อมูล**
- **การบันทึกข้อมูล**
- **ส่วนติดต่อผู้ใช้ (GUI)**

เพื่อพัฒนาระบบให้ใกล้เคียงกับโปรแกรมจริงมากขึ้น

สรุปบทเรียน

การเรียน OOP จะสมบูรณ์ก็ต่อเมื่อ

ผู้เรียนสามารถนำความรู้ไปสร้างผลงานได้จริง

Project-Based Learning ช่วยให้ผู้เรียน

จาก "คนที่รู้"

กลายเป็น

"คนที่ทำได้"

จำไว้ว่า

"โปรแกรมเมอร์ไม่ได้เก่งขึ้นจากการอ่านโค้ดเพียงอย่างเดียว

แต่เก่งขึ้นจากการลงมือสร้างโปรแกรมจริง"

และทุกโครงงานที่นักเรียนสร้าง

คืออีกหนึ่งก้าวสำคัญสู่การเป็นนักพัฒนาซอฟต์แวร์ในอนาคต

บทที่ 7 : การประเมินผล OOP แบบ Active Learning

ปัญหา

เมื่อสอน OOP จบแล้ว

ครูหลายคนมักใช้วิธีประเมินแบบเดิม

- สอบปรนัย
- สอบข้อเขียน
- ท่องจำคำศัพท์

ผลคือ

นักเรียนบางคนทำข้อสอบได้ดี

แต่เขียนโปรแกรมไม่ได้

ในขณะที่บางคนสร้างโปรแกรมได้จริง

แต่กลับทำคะแนนสอบไม่สูง

คำถามสำคัญคือ

"เรากำลังวัดความรู้ หรือกำลังวัดความสามารถ?"

ในโลกการทำงานจริง

ไม่มีใครถามว่า

"คุณจำความหมายของ Encapsulation ได้ไหม?"

แต่ทุกคนถามว่า

"คุณสามารถสร้างโปรแกรมที่ใช้งานได้จริงหรือไม่?"

ทำไมต้องแก้

เป้าหมายของการเรียน OOP

ไม่ใช่การจำทฤษฎี

แต่คือการนำความรู้ไปใช้แก้ปัญหาได้

Active Learning จึงให้ความสำคัญกับ

✅ การลงมือทำ

✅ การคิดวิเคราะห์

✅ การทำงานร่วมกัน

✅ การสร้างผลงานจริง

ดังนั้นการประเมินผล

ก็ควรวัดสิ่งเหล่านี้เช่นกัน

เมื่อเปลี่ยนวิธีประเมิน

ผู้เรียนจะเปลี่ยนวิธีเรียน

จาก

"เรียนเพื่อสอบ"

เป็น

"เรียนเพื่อทำได้จริง"

ขั้นตอนกิจกรรม

ขั้นที่ 1 กำหนดเป้าหมายการเรียนรู้

ก่อนประเมิน

ครูควรถามตนเองก่อนว่า

ต้องการให้ผู้เรียนทำอะไรได้

ตัวอย่าง

- ออกแบบ Class ได้
- สร้าง UML ได้
- เขียนโปรแกรม OOP ได้
- ทำงานเป็นทีมได้

เมื่อเป้าหมายชัด

การประเมินก็จะตรงจุด

ขั้นที่ 2 ใช้ภาระงานจริง

แทนที่จะสอบเพียงอย่างเดียว

ให้นักเรียนสร้างผลงาน

เช่น

- ออกแบบ UML
- สร้างโปรแกรม
- นำเสนอระบบ
- วิเคราะห์ปัญหา

สิ่งเหล่านี้สะท้อนความสามารถจริงมากกว่า

ขั้นที่ 3 ประเมินระหว่างเรียน

ไม่ต้องรอสอบปลายภาค

ครูสามารถประเมินได้ตลอดเวลา

เช่น

- การตอบคำถาม
- การทำกิจกรรม
- การทำงานกลุ่ม
- การแก้ปัญหา

ช่วยให้เห็นพัฒนาการของผู้เรียนอย่างต่อเนื่อง

ขั้นที่ 4 ใช้ Rubric

Rubric คือเกณฑ์การให้คะแนนที่ชัดเจน

ช่วยให้นักเรียนรู้ว่า

ต้องทำอะไรจึงจะประสบความสำเร็จ

และช่วยลดความคลุมเครือในการให้คะแนน

ขั้นที่ 5 สะท้อนผลการเรียนรู้

หลังจบกิจกรรม

ให้นักเรียนตอบคำถาม

- วันนี้ได้เรียนรู้อะไร?
- สิ่งใดที่ยังไม่เข้าใจ?
- ถ้าทำใหม่จะปรับปรุงอะไร?

การ Reflection

ช่วยให้การเรียนรู้เกิดขึ้นอย่างลึกซึ้ง

ตัวอย่าง

ตัวอย่างที่ 1 : ประเมินจากโครงงาน

หัวข้อ

ระบบจัดการร้านกาแฟ

สิ่งที่ประเมิน

- การออกแบบ Class
- การใช้ Inheritance
- การใช้ Encapsulation
- การทำงานของโปรแกรม
- การนำเสนอ

คะแนนรวม 100 คะแนน

ตัวอย่างที่ 2 : ประเมินจาก UML

เกณฑ์

รายการ	คะแนน
กำหนด Class ถูกต้อง	20

รายการ	คะแนน
Attribute ครบถ้วน	20
Method เหมาะสม	20
ความสัมพันธ์ของ Class	20
ความถูกต้องของ UML	20

รวม 100 คะแนน

ตัวอย่างที่ 3 : ประเมินการทำงานกลุ่ม

พิจารณา

- การมีส่วนร่วม
- ความรับผิดชอบ
- การสื่อสาร
- การช่วยเหลือเพื่อน

ซึ่งเป็นทักษะสำคัญในโลกการทำงานจริง

การบ้าน

Mission : สร้าง Rubric ของคุณเอง

ให้นักเรียนเลือกกิจกรรม 1 กิจกรรม

เช่น

- การสร้าง UML
- การเขียนโปรแกรม
- การนำเสนอผลงาน

จากนั้นออกแบบ Rubric ของตนเอง

โดยกำหนด

1. หัวข้อที่ต้องประเมิน
2. ระดับคุณภาพ 4 ระดับ
3. คะแนนแต่ละระดับ
4. เหตุผลในการกำหนดเกณฑ์

Challenge พิเศษ

ให้นักเรียนแลกเปลี่ยน Rubric กับเพื่อน

แล้วทดลองใช้ประเมินผลงานจริง

จากนั้นเปรียบเทียบผลการประเมิน

เพื่อเรียนรู้มุมมองที่แตกต่าง

สรุปบทเรียน

การประเมินผลที่ดี

ไม่ใช่การจับผิดผู้เรียน

แต่คือการช่วยให้ผู้เรียนเห็นพัฒนาการของตนเอง

ในวิชา OOP

สิ่งที่ควรวัดไม่ใช่แค่ความจำ

แต่คือ

- ความเข้าใจ
- การคิดวิเคราะห์
- การออกแบบระบบ
- การสร้างโปรแกรม

- **การทำงานร่วมกับผู้อื่น**

จำไว้ว่า

"สิ่งที่เราวัด คือสิ่งที่ผู้เรียนจะให้ความสำคัญ"

หากเราวัดการท่องจำ

ผู้เรียนก็จะท่องจำ

แต่หากเราวัดการลงมือทำ

ผู้เรียนก็จะลงมือทำ

และนั่นคือหัวใจของการเรียนรู้แบบ Active Learning อย่างแท้จริง