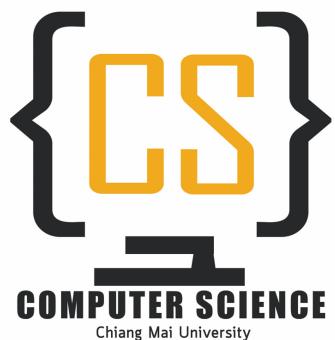


# CS204362 – Object-Oriented Design

## L2: Concept of Object-Oriented (OO) Design

Kamonphop Srisophha



Faculty of Science, Chiang Mai University  
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

# Learning Objectives

- The fundamental concepts of object orientation and OO design, including:
  - Objects and classes
  - Abstraction, Encapsulation, Information Hiding
  - Modularity, Association
  - Hierarchy, Generalization, Specialization
  - Inheritance, Polymorphism
- Why OO Design?

# Motivation for Objects

ถ้าเราจะเดินทางไปเที่ยวต่างประเทศหรือต่างจังหวัด, ถ้าให้เลือกชนของแบบ 1) ชนทุกอย่างไว้ในมือเป็นสิ่งย่อยๆไป หรือ 2) ใส่ของทุกอย่างในกระเป๋าเดินทางแล้วขนกระเป๋าเดินทาง นักศึกษาจะเลือกอะไร?



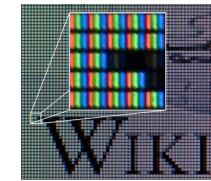
```
void process(char name[],  
int id, int major,...){  
...  
}
```

```
void process(Student  
s1){  
...  
}
```

# Objects

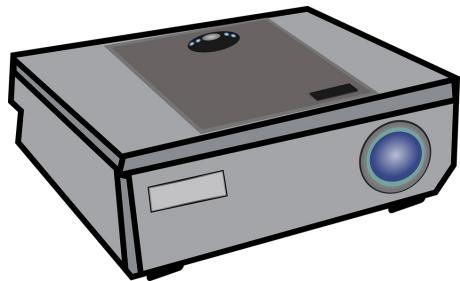
ปอยครั้งที่เราไม่สามารถแทนสิ่งบางสิ่งในโลกของความเป็นจริงด้วยตัวแปรแค่ 1 ตัวแปรในซอฟท์แวร์อย่างพวก **integer**, **character**, หรือ **double/float** ได้ เช่น

- A **pixel** เป็นการรวมกันของ **int red, green, blue**
- A **circle** เป็นการรวมกันของ **Cx, Cy, Radius, Color**
- A **student** เป็นการรวมกันของ **name, age, ID, major** และอื่นๆ



**Object** นั้นก็คือหน่วยในซอฟท์แวร์เกิดจากการที่เราจำลักษณะ (**attributes**) และฟังก์ชันการทำงาน (**methods**) มาด้วยกันเพื่อที่จะใช้มันเป็นแทนของบางสิ่งในโลกของความเป็นจริง และทำให้เรา **interact** กับมันในโลกของซอฟท์แวร์ได้

# Objects in this room



A projector

Resolution  
Brightness  
Display()  
TakeVideoInput()



A chair

Number of Legs  
Dimension  
Material  
isOccupied()



A person

Name  
Age  
Gender  
Teach()  
Talk()

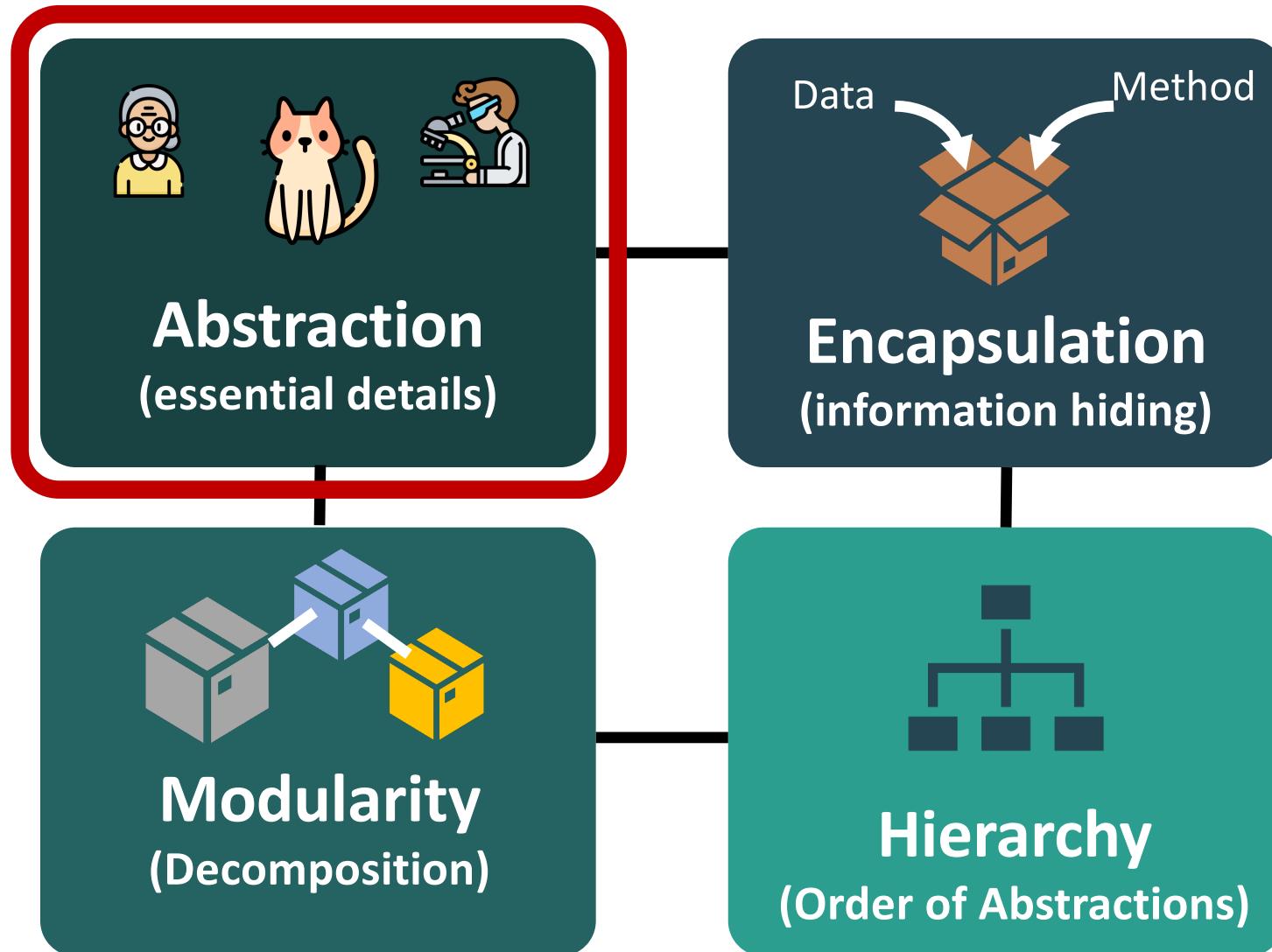
# Object Orientation (แนวคิดเชิงวัตถุ)



ประเด็นของแนวคิด OO Design คือการที่เราไม่เดล software ของเราให้:

- 1) เป็นระบบที่ประกอบด้วย objects ต่างๆ
- 2) Objects เหล่านั้นมีปฏิสัมพันธ์ระหว่างกัน

# Principle of OO Design



# Abstraction (ลดการซับซ้อนของความเป็นจริง)

- **Abstraction** เป็นการมุ่งโฟกัสไปที่ลักษณะสำคัญของบางสิ่ง โดยไม่สนใจรายละเอียดปลีกย่อย ทำให้ความซับซ้อนของสิ่งนั้นลดลง สำหรับมุมมองใดมุมมองหนึ่ง

เช่น

คนๆหนึ่ง (Person)

ถ้าเราไม่กำหนดมุมมองเราจะโฟกัสไปที่ลักษณะสำคัญได้  
ของคนๆหนึ่ง(และเพื่ออะไร)? (Hint: กว้างมาก)

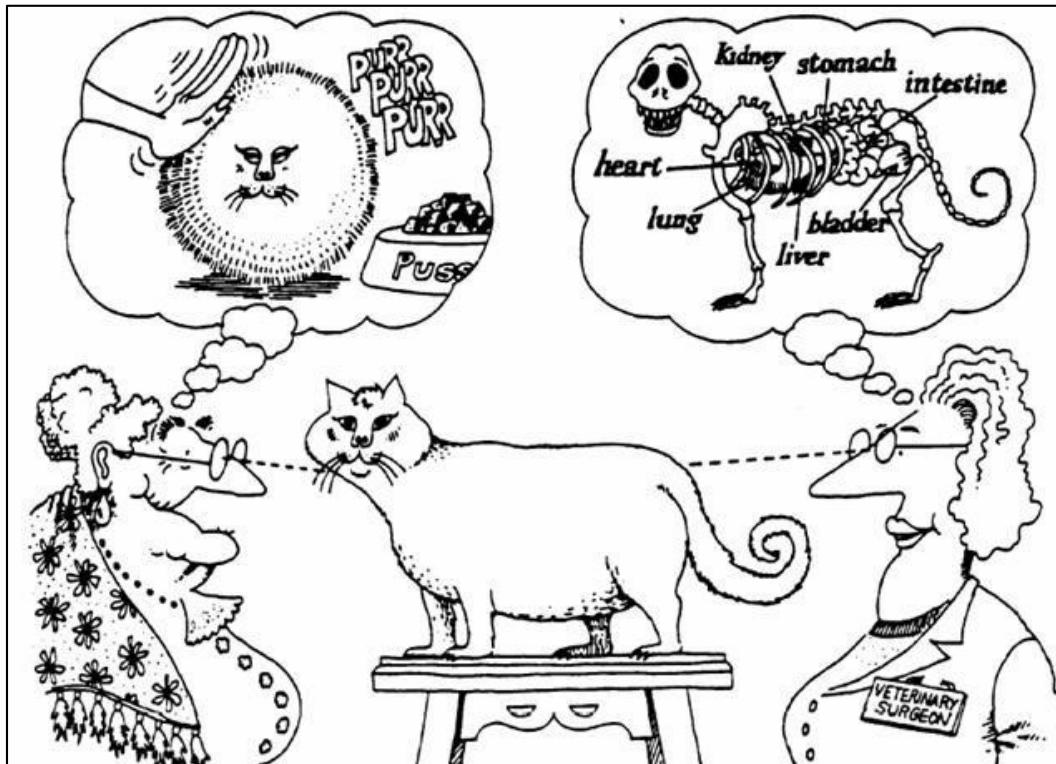


ถ้าเราบอกว่าเราจะสร้าง  
**Driving App (Grab)**  
ลักษณะสำคัญของ  
**Person** ต้องมีอะไรบ้าง?



ถ้าเราจะสร้าง  
**Restaurant App,**  
ลักษณะสำคัญของ  
**Person** ต้องมีอะไรบ้าง?

# The Cat as a Metaphor for Abstraction



“Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer”

- Grady Booch

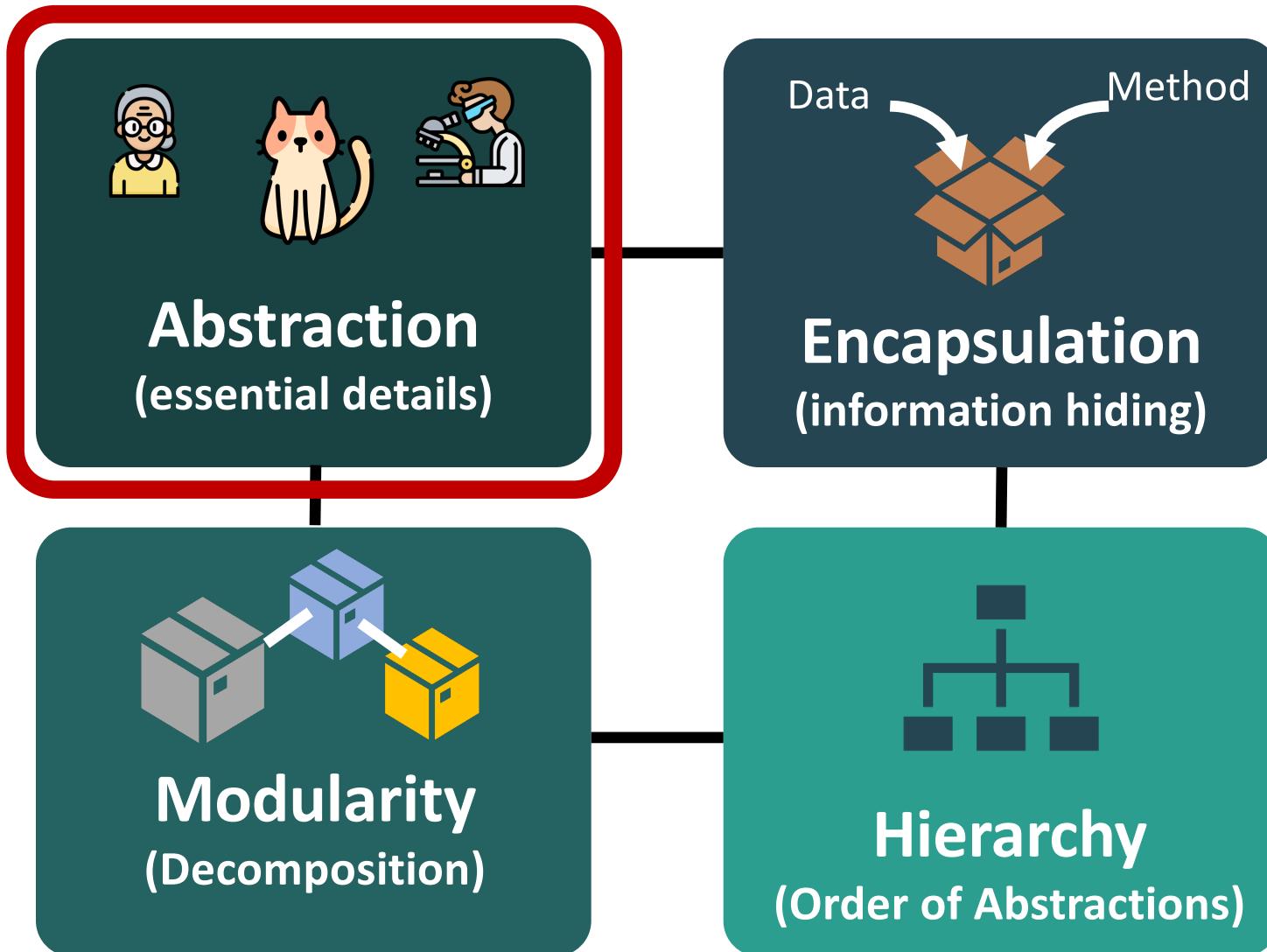
ฉะนั้น **Context (Perspective of the viewer)**  
สำคัญมากสำหรับ **abstraction**

“Object Oriented Analysis and Design with Applications”, Booch et al., 3<sup>rd</sup> Edition

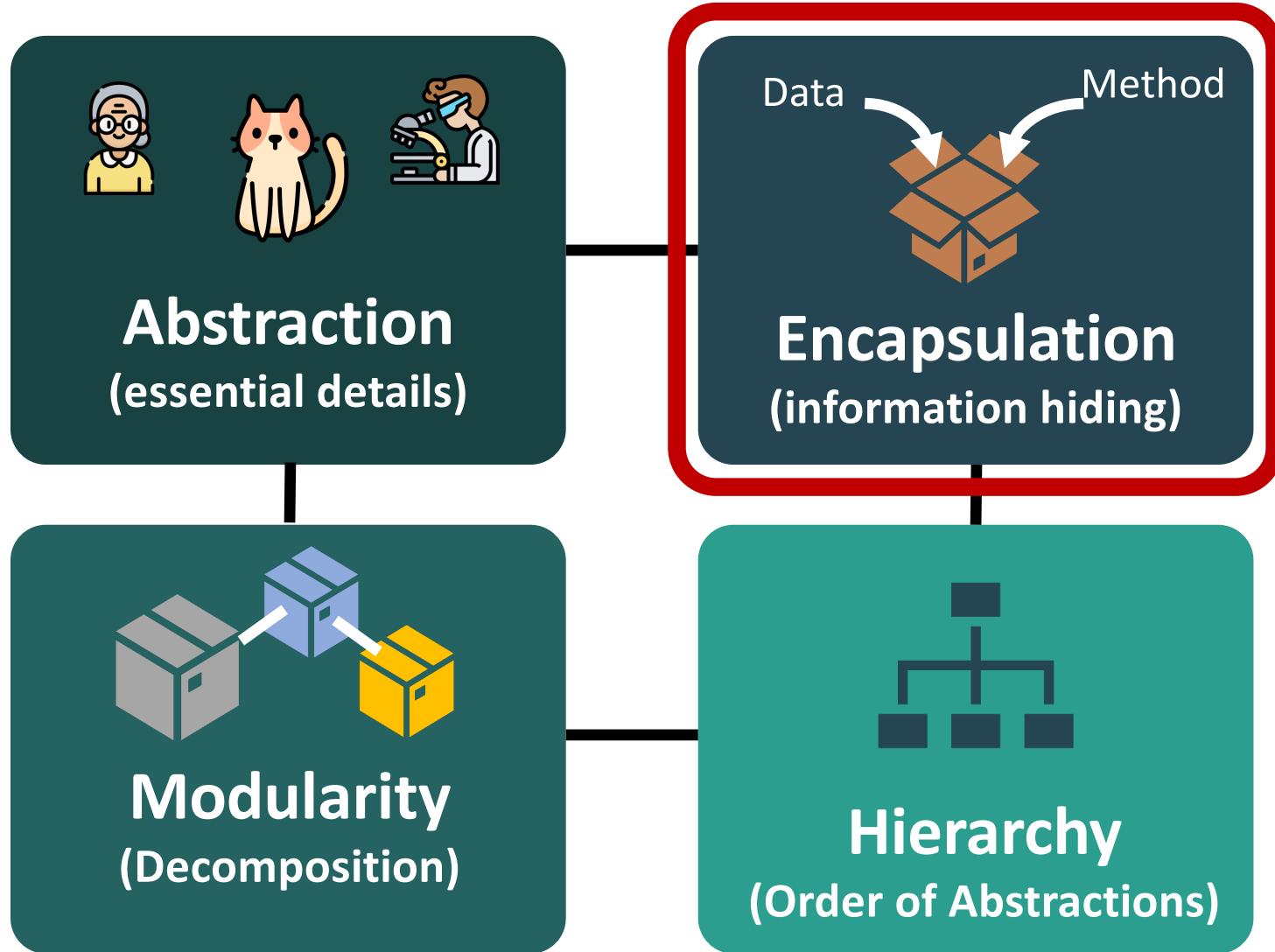
# Check your understanding

Q: ถ้าเราอยากระสร้าง Abstraction สำหรับ นักศึกษา (Student) ในมุ่มนองของ spanning ที่มีอยู่ในโครงสร้างแบบตัวอย่าง ลักษณะสำคัญที่ต้องโฟกัส มีอะไรบ้าง?

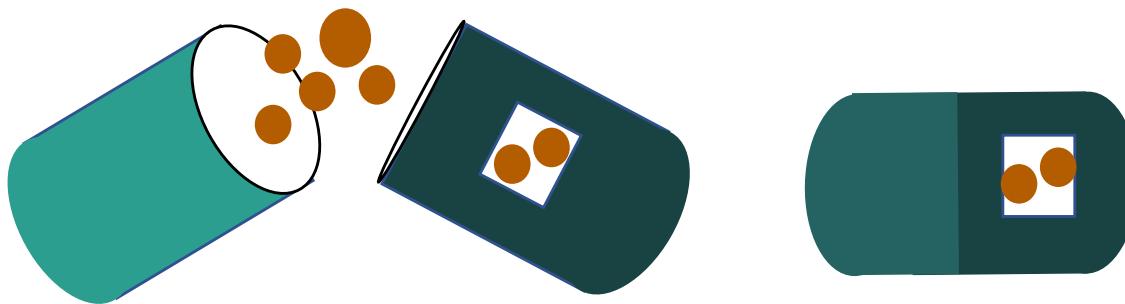
# Principle of OO Design



# Principle of OO Design



# Encapsulation (การห่อหุ้มและซ่อนข้อมูล)



## Encapsulation คือการ

- 1) นำเอา คุณสมบัติ หรือลักษณะสำคัญ และ พฤติกรรมสำคัญ (เรียกว่าข้อมูล) ที่ระบุมาได้จาก abstraction มาห่อหุ้มรวมกัน
- 2) เปิดให้ข้อมูลบางข้อมูลเข้าถึงได้จากภายนอก
- 3) จำกัดการเข้าถึงของข้อมูลบางข้อมูลจากภายนอก ให้ข้อมูลนั้นใช้สำหรับภายนอกเท่านั้น

Information Hiding

# Example of Encapsulation

**A Coke Vending Machine:** เครื่องมีช่องให้เราหยดเหรียญ รับเหรียญ  
รับของ มีปุ่มให้เรากดเลือกน้ำ มีไฟบอกสถานะว่าโค้กหมดมั้ย เราไม่ต้องสนใจ  
เครื่องมันจ่ายโค้กมาให้เราอย่างไร หรือ มีโค้กเหลือกี่กระป๋อง (ถ้าไม่หมด)



มันอาจมีคนอยู่  
ภายใน คอยจ่าย  
โค้กให้เรา ก็เป็นได้

# Another Example of Encapsulation

จะมาเรียนคลาสกับผู้สอนได้

เกรดเฉลี่ยเท่าไหร่

3.95 ครับ



นักศึกษาอาจจะ

- 1 สงเรื่องขอ transcript ไปที่สำนักงานทะเบียน
2. เข้าไปตรวจใน ระบบส่วนกลางของมหาวิทยาลัย
3. คำนวนเองจากเกรดของวิชาที่เรียนมา
- 4 .....

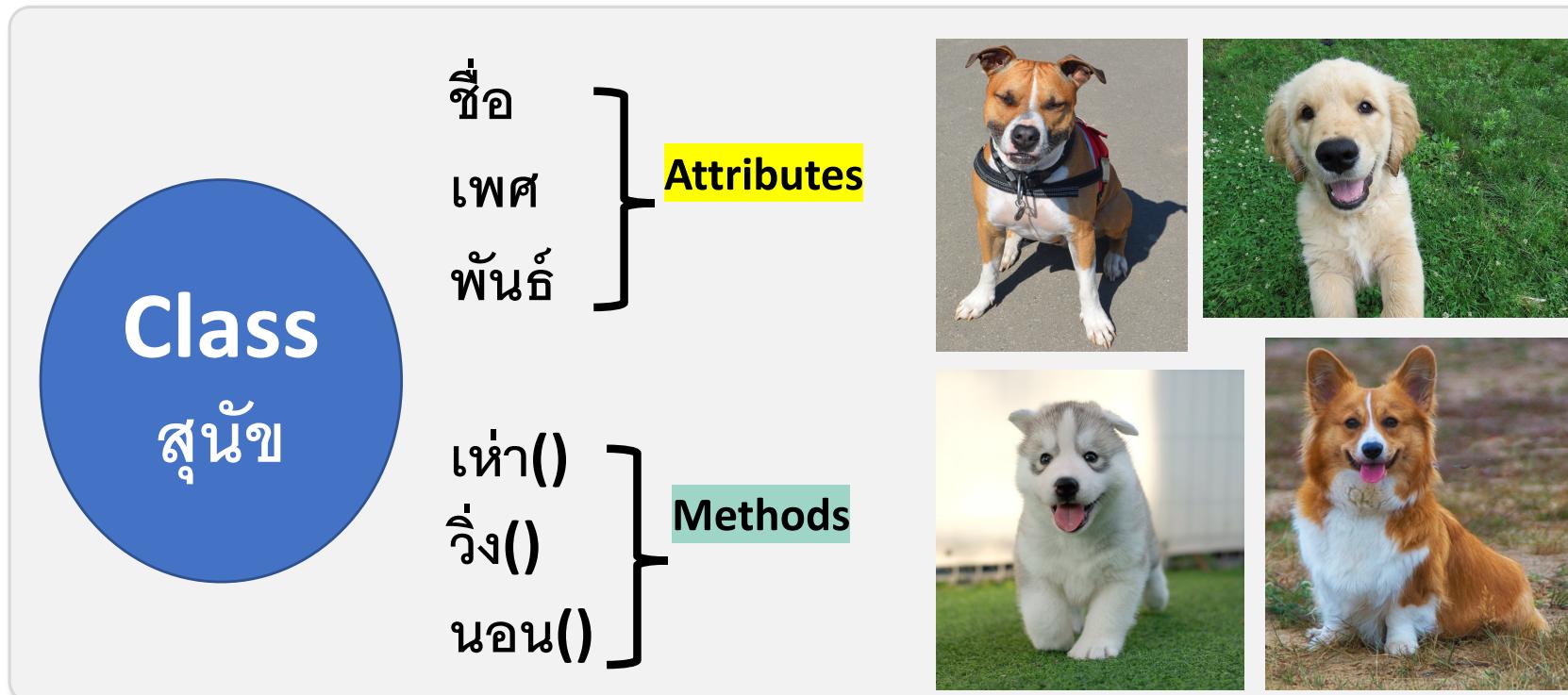
อาจารย์ไม่จำเป็นต้องรู้ว่านักศึกษาได้รวมคะแนน

GPA มาได้อย่างไร หรือรู้ว่านักเรียนเรียนวิชาอะไร  
มากบ้างแล้วและแต่ละวิชาได้เกรดเท่าไหร่

- ไม่ว่าจะวิธีใดก็ตาม ผลลัพธ์สุดท้ายก็คือ บอกอาจารย์ว่าตนได้เกรดเฉลี่ยเท่าไหร่
- นักศึกษาเป็นคนเดียวที่รู้ว่าคำนวน GPA ด้วยวิธีไหน

# Class (ผลลัพธ์จากการ abstraction + encapsulation)

- คือแม่พิมพ์ (Blueprint) ในการสร้าง objects ที่จะบอกว่า objects ต้องมีลักษณะอะไร (attributes) และมีพฤติกรรมอะไร (methods)
- Objects คือวัตถุที่สร้างมาจากการ Class จะนับ Object จาก Class เดียวกันอาจมีลักษณะต่างกัน



# What are instances?

กระบวนการที่ทำให้เกิด Object จาก Class ในทาง CS คือ  
กระบวนการที่เรียกว่า **Instantiation**



ฉะนั้น Object ก็คือ Instance หนึ่งของ Class นั้นเอง

ดังนั้น Instance (object) ของ Class ทั้งหมดจะมีความเหมือนกัน

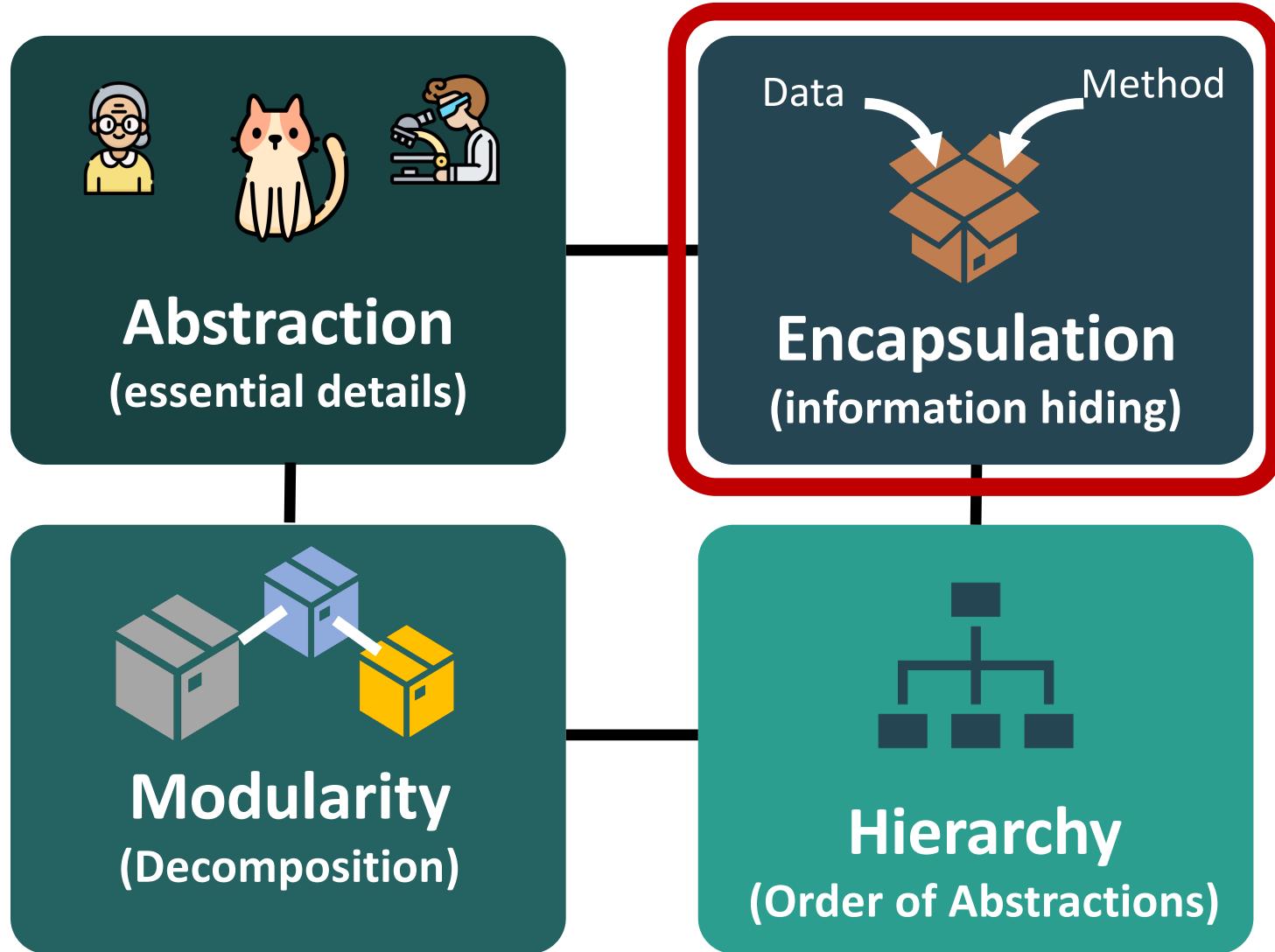
- ในทาง Structure: (same attributes) e.g., what they know, what information they hold, และ what *links* they have to other objects.
- ในทาง Behavior: (same methods) e.g., what they can do

# Should It Be a Class or an Instance?

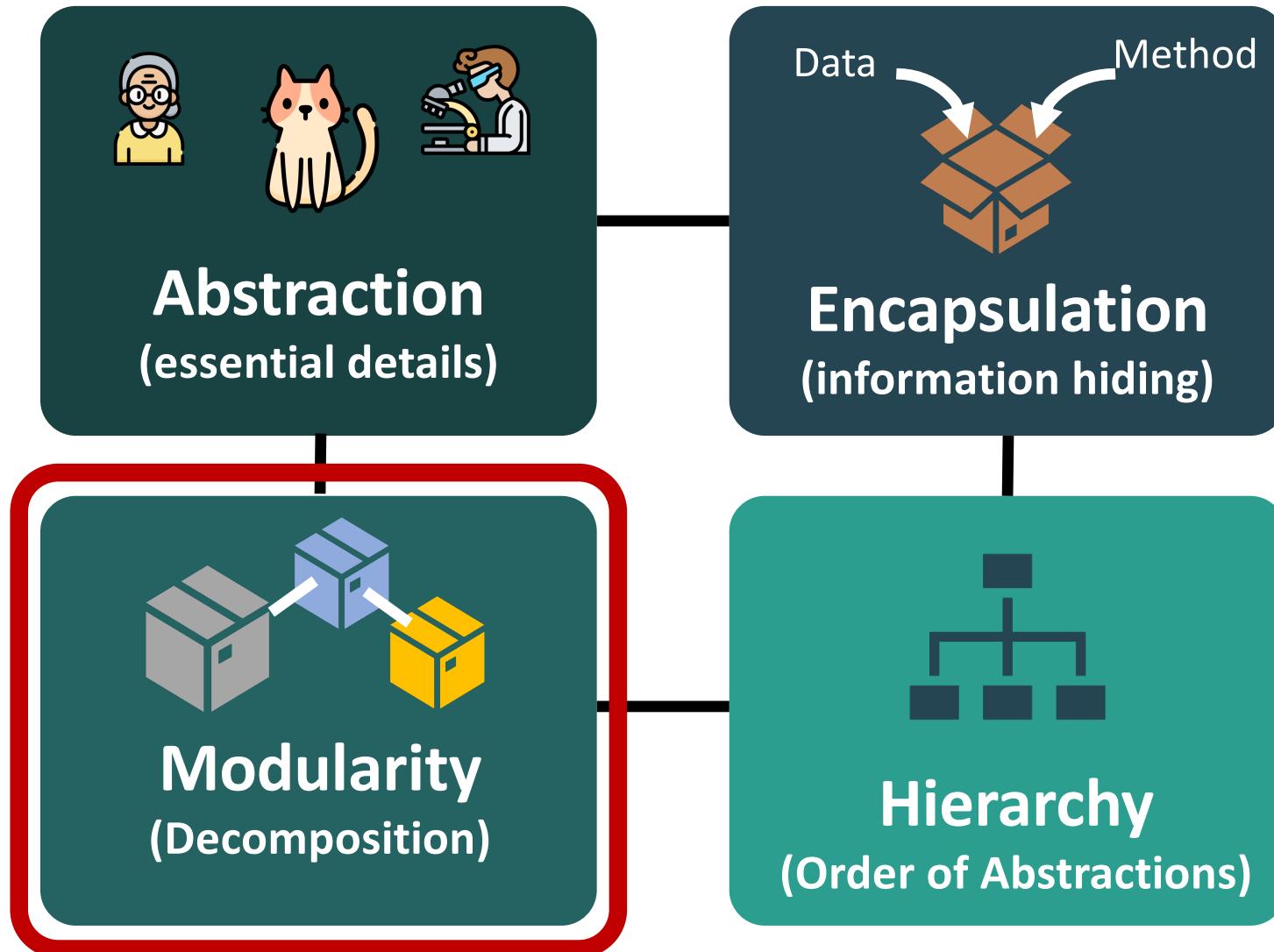
**Q:** จาก List ด้านล่างนี้ นักศึกษาคิดว่า อันไหนควรเป็น class อันไหนควรเป็น instance. ถ้าคิดว่าเป็น instance ให้บอก class ที่เหมาะสมของมัน. ถ้า นักเรียนคิดว่ามันเป็นได้ทั้ง class และ instance ขึ้นอยู่กับสถานการณ์ ก็ให้อธิบายเสริม

- a) Automobile Company
- b) Boeing 777
- c) Computer Science Student
- d) Game
- e) Albert Einstein
- f) Board Game
- g) Chess
- h) CMU Course CS362
- i) The game of chess between Tom and Jane which will start at 2:30pm today
- j) The car with serial number JM198765T4
- k) Thai baht

# Principle of OO Design

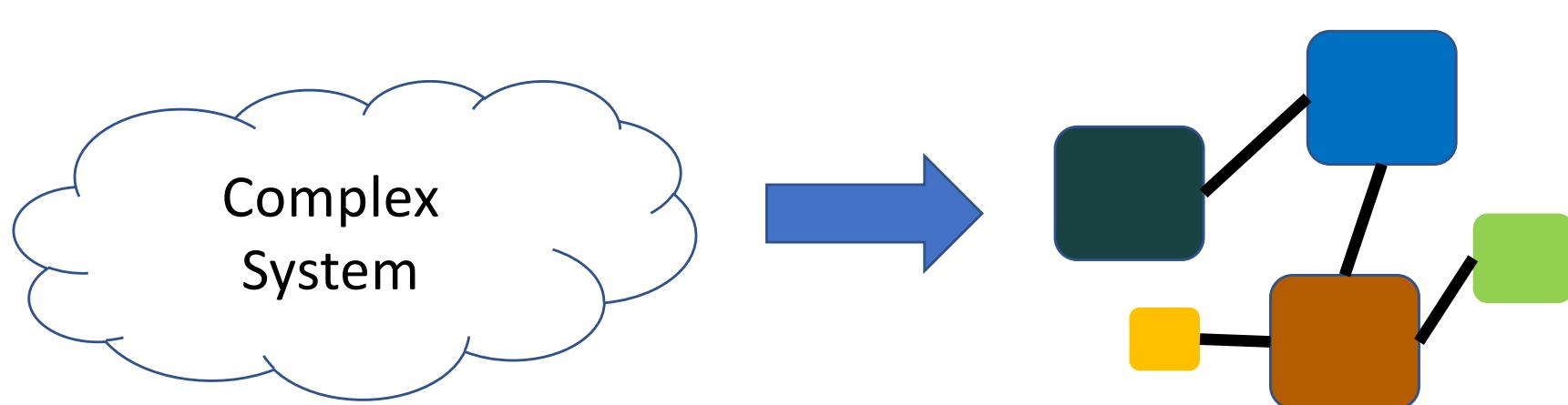


# Principle of OO Design



# Modularity (การแยกส่วนจำเพาะ)

การแตกระบบที่ซับซ้อน ให้เป็นหน่วยย่อยๆ ที่ความสัมพันธ์ภายในแข็งแรง (ความเกี่ยวข้องในตัวของแต่ละหน่วยสูง) และ ความสัมพันธ์ระหว่างหน่วยต่างๆ โดยไม่แต่ละหน่วยมีหน้าที่ของมันเองอย่างชัดเจน



# Principles of Modularity

High Cohesion

หน่วยแต่ละหน่วยประกอบไปด้วย **data** และ **method** ที่ไปในทางเดียวกัน

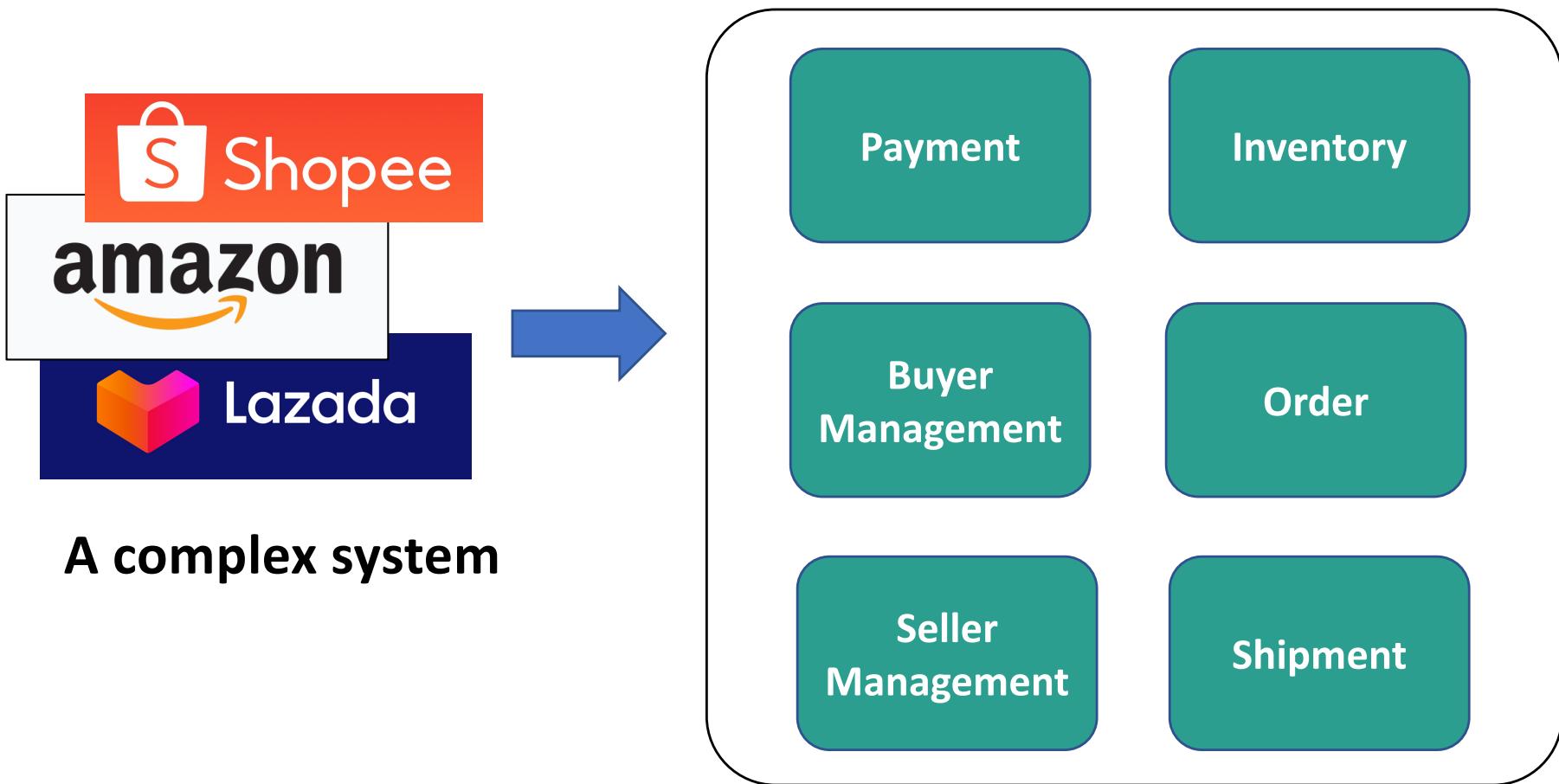
Loose Coupling

เปลี่ยนหน่วยได้หน่วยหนึ่งไม่ควรจะมีผลกระทบกับหน่วยอื่นๆ

Law of Demeter

สื่อสารกับหน่วยที่มันรู้จักได้แค่นั้น (คุณได้แต่กับเพื่อน ไม่สนใจเพื่อนของเพื่อน)

# Example of Modularity



# Relationship ความสัมพันธ์กันของหน่วย

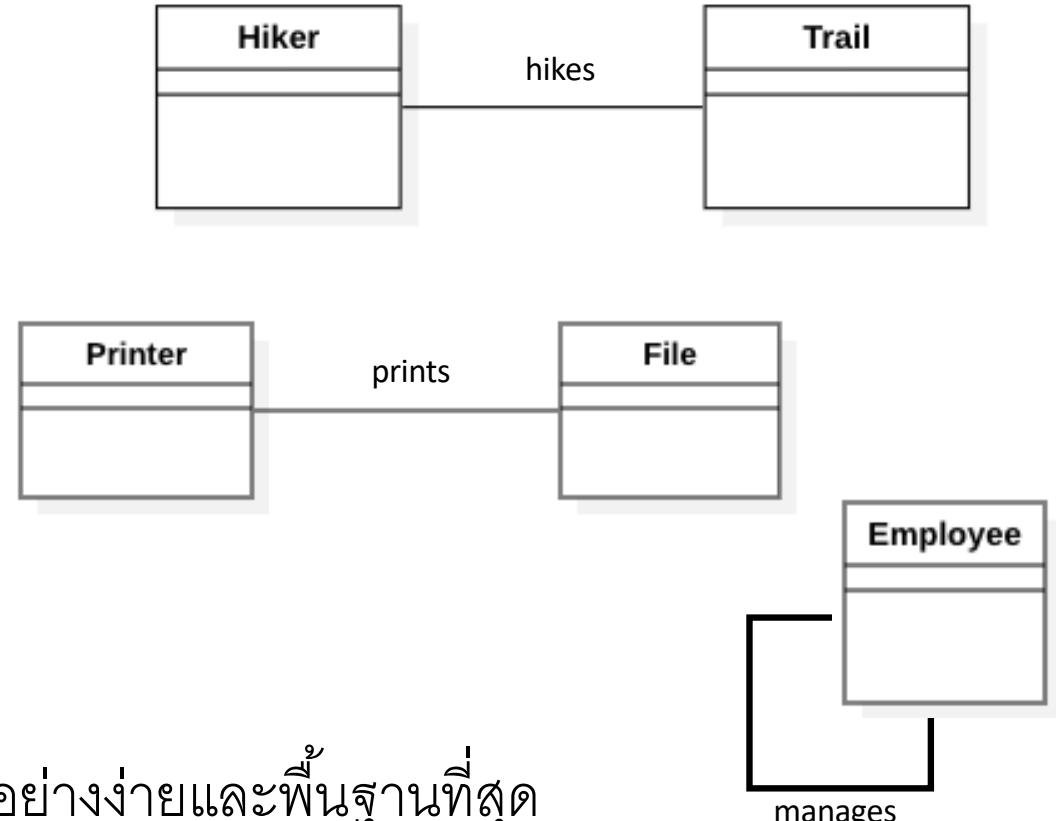
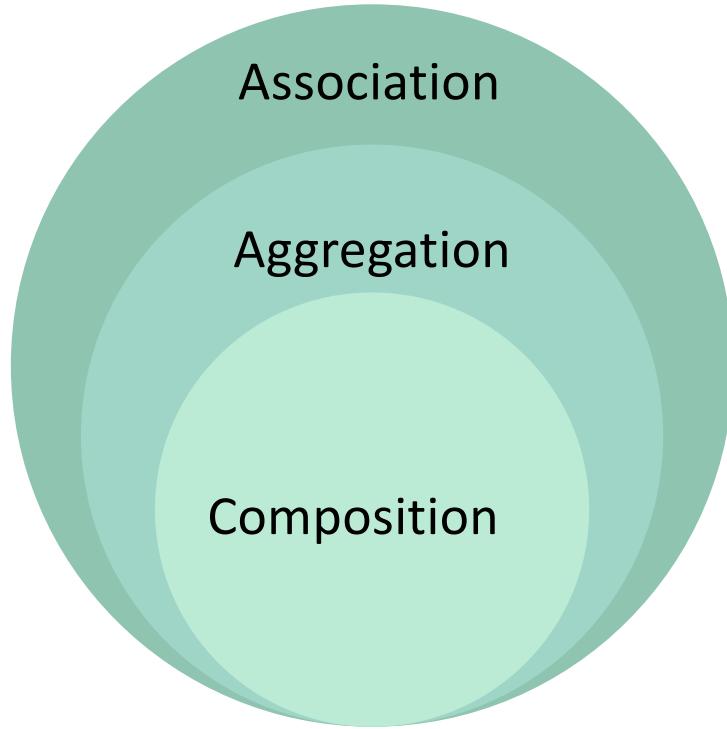
Units ในที่นี่อาจเป็นได้ทั้ง Objects และ Class

- ความสัมพันธ์ระหว่าง Objects เราจะเรียกมันว่า **Link**
- ความสัมพันธ์ระหว่าง Classes เราจะเรียกมันว่า **Association**

ความสัมพันธ์ระหว่างคลาสมี:

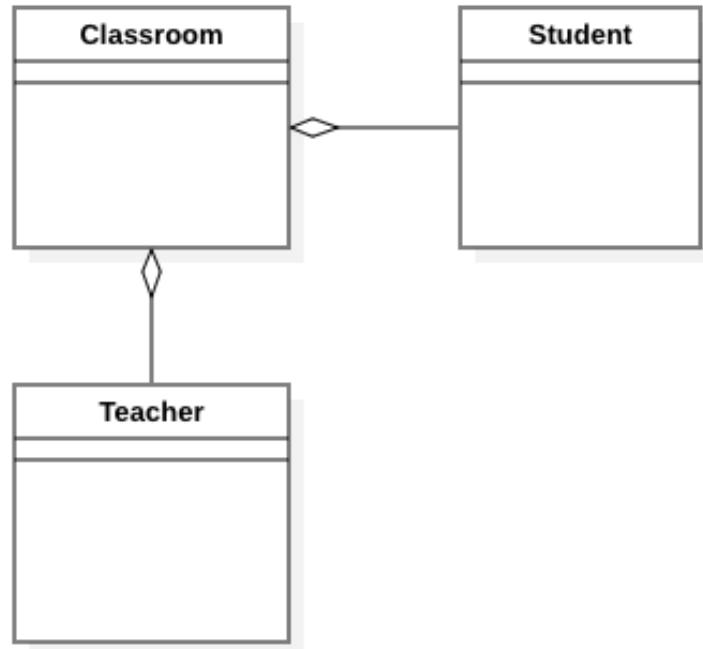
1. Association
2. Aggregation (คล้าย association แต่มีความจำเพาะ)
3. Composition (คล้าย aggregation แต่มีความจำเพาะลงไปอีก)

# Relationship: Association



- ความสัมพันธ์ระหว่างคลาสอย่างง่ายและพื้นฐานที่สุด
- ความสัมพันธ์ **peer-to-peer**
- ความสัมพันธ์แบบ **Aggregation** กับ **Composition** ก็เรียกว่าเป็น **Association** เช่นกัน

# Relationship: Aggregation



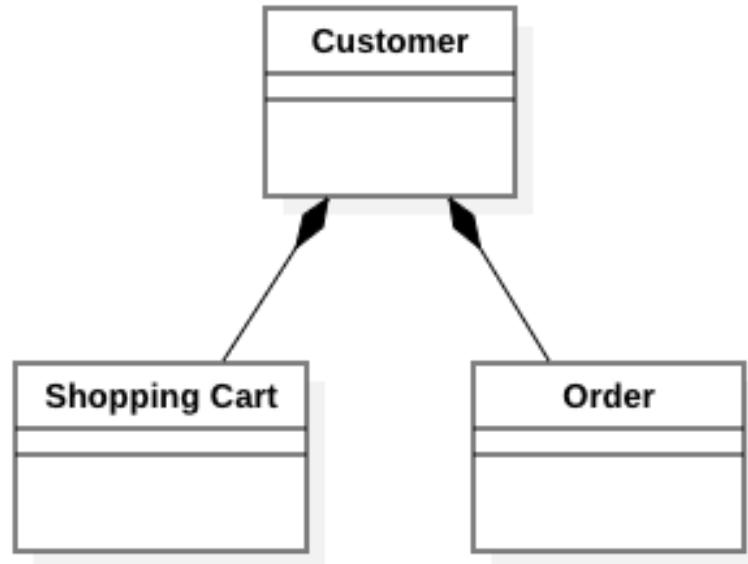
A Classroom has a Student(s)  
A Classroom has a Teacher(s)

A Student and a Teacher can exist  
independently from a classroom

ความสัมพันธ์ระหว่างกันที่บ่งบอกความ  
มีหรือความเป็นส่วนประกอบ ประหนึ่งว่า  
คลาสนี้มีหรือประกอบไปด้วยส่วนย่อยๆ  
อะไรบาง

แต่ถึงแม้ถ้าไม่มีคลาสอย่อย คลาสหลักก็ไม่  
จำเป็นว่าต้องสูญหายไปหรืออยู่ไม่ได้

# Relationship: Composition



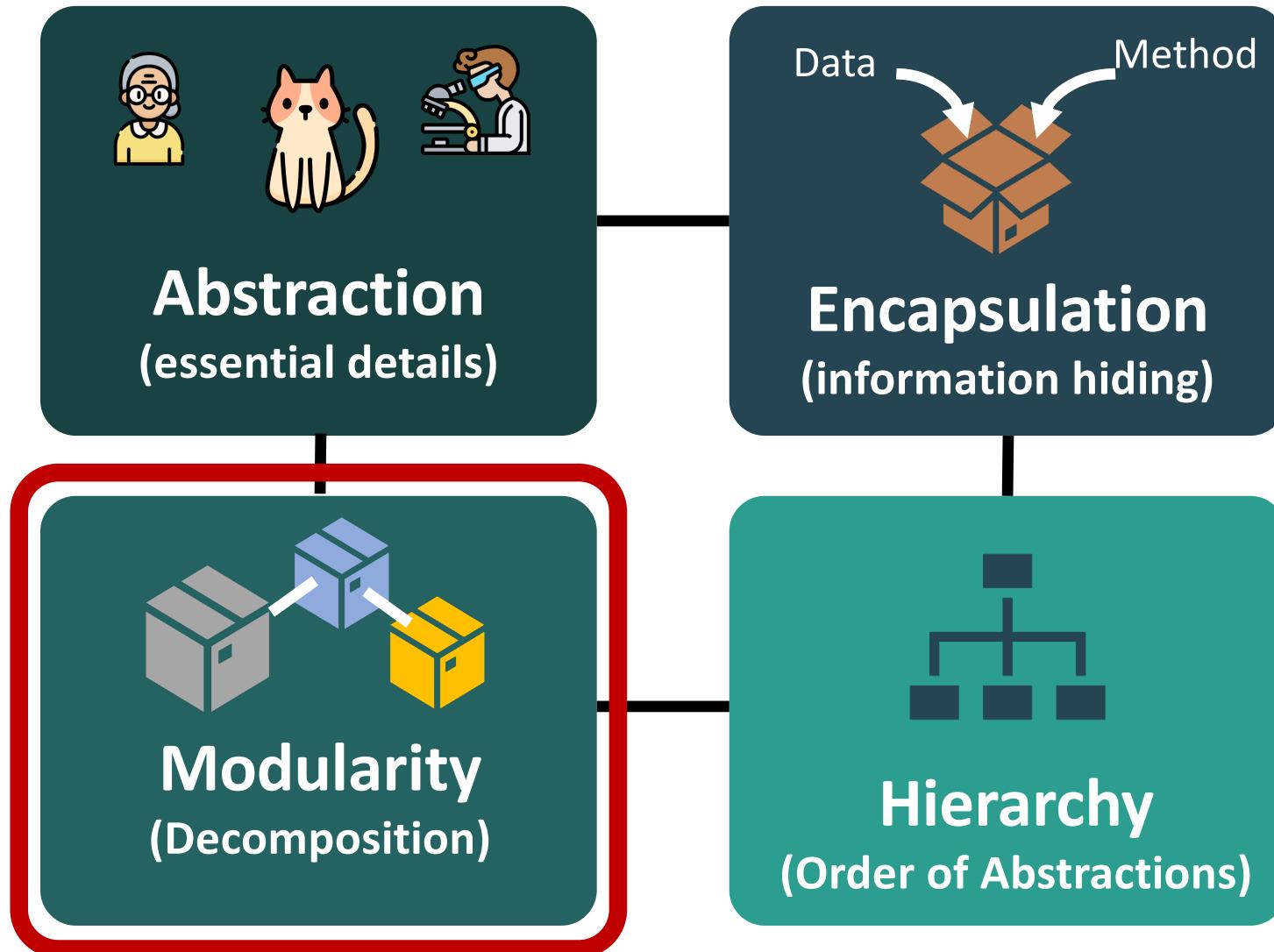
A customer “owns” a shopping cart and an order(s)

The shopping cart and order have no meaning in the model without a customer

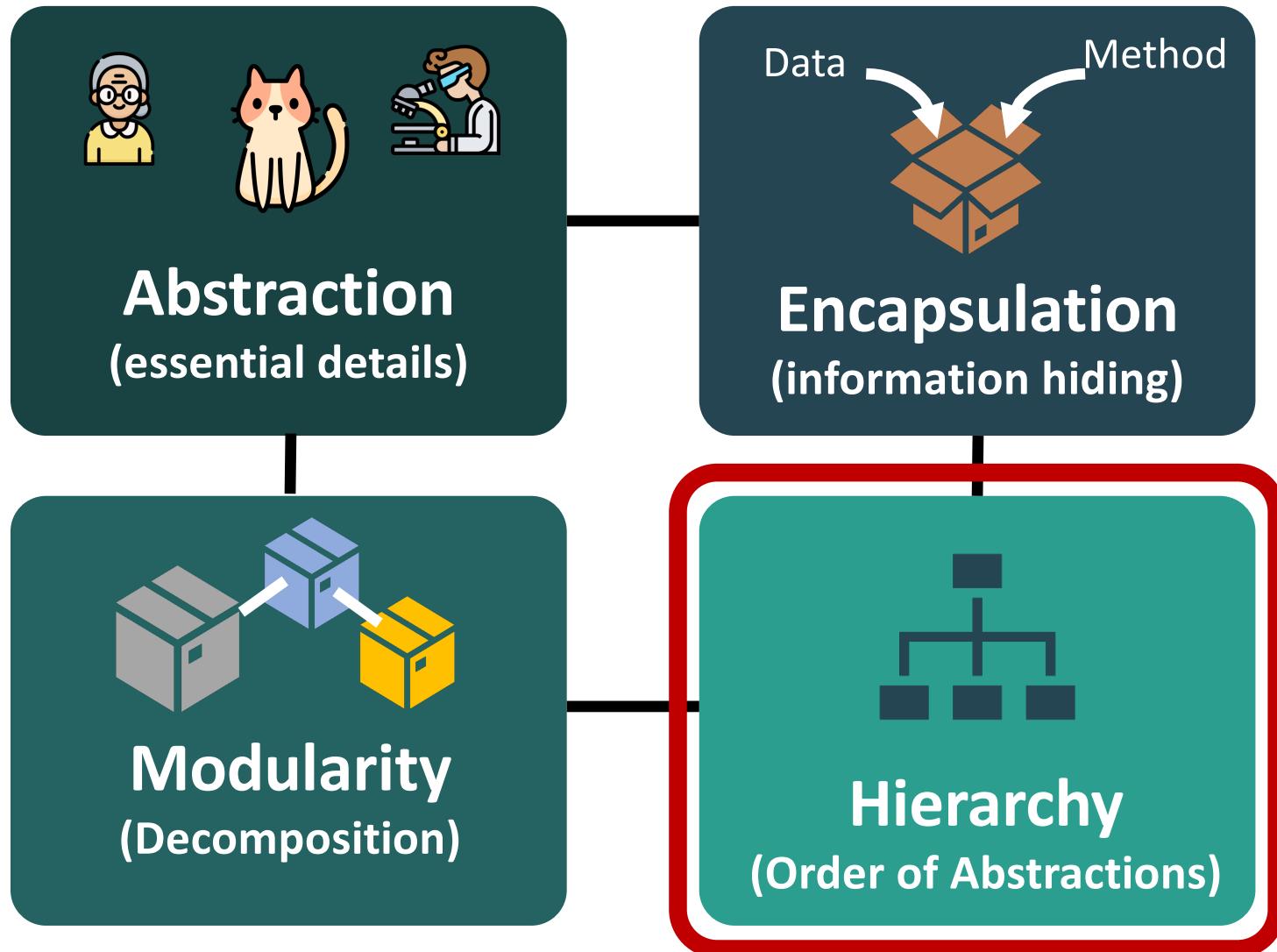
ความสัมพันธ์คล้ายกับ Aggregation แต่เป็นแบบที่จำเพาะกว่า เพราะมีการแสดง ความเป็นเจ้าของระหว่างความสัมพันธ์

เมื่อคลาสหลักสูญหายไป คลาஸย่อยก็อยู่ไม่ได้ (คลาஸย่อยอยู่ไม่ได้หรือสูญหายไปถ้าไม่มีคลาสหลัก)

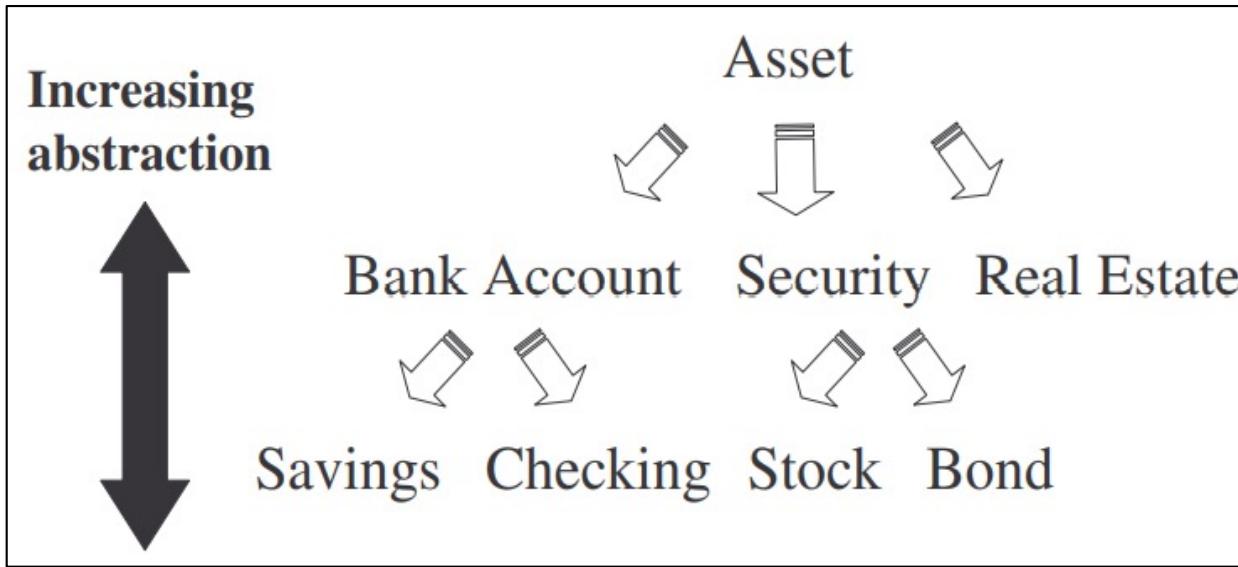
# Principle of OO Design



# Principle of OO Design



# Hierarchy (การจัดวางลำดับชั้น)



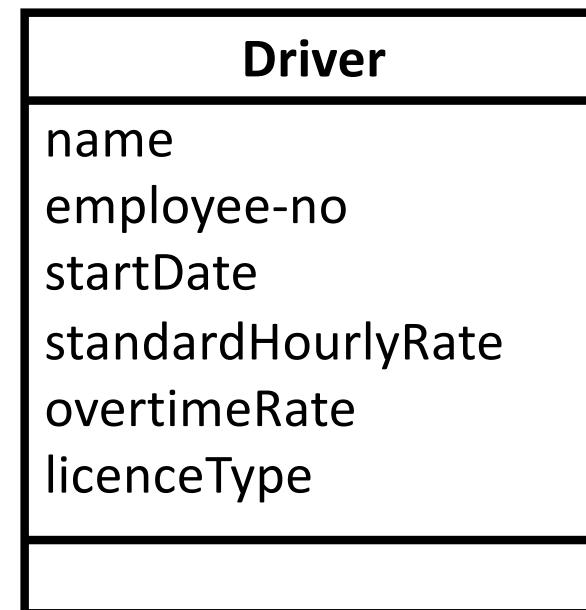
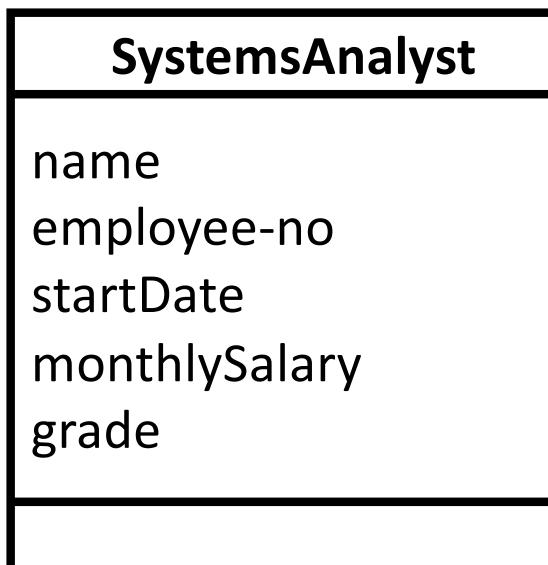
หลังจากที่เรานำ **Modularity principle** มาใช้ เราอาจจะได้ คลาส ให้บูรณาการคลาส การจัดวางลำดับชั้น (Hierarchy) ก็เป็นอีก หลักการสำคัญของ OOD หนึ่งที่ให้เราจัดคลาส และ ความสัมพันธ์ของ คลาสให้เป็นระเบียบและง่ายต่อการเข้าใจมากขึ้น

สองเทคนิคหลักของการจัดวางลำดับชั้นคือ **Generalization** และ **Specialization**

# Generalization and Specialization

**Generalization** คือ การนำเอาคุณสมบัติที่เหมือนหรือคล้ายกันของหลายๆ คลาส มาสร้างเป็นคลาสใหม่ เพื่อที่จะลดการใช้และประมวลซ้ำของข้อมูล

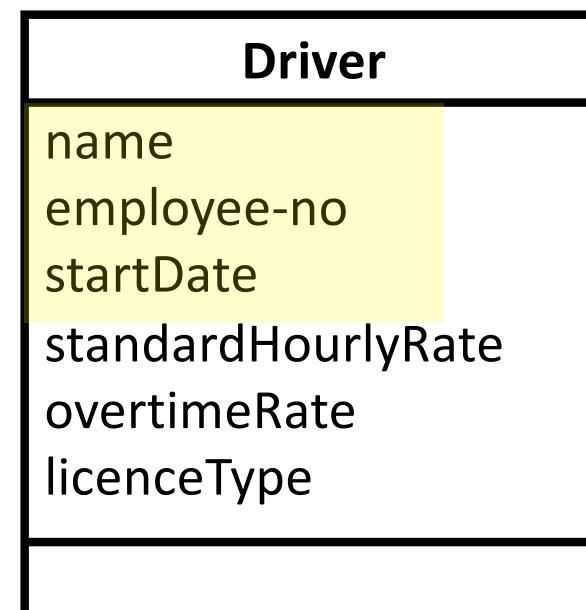
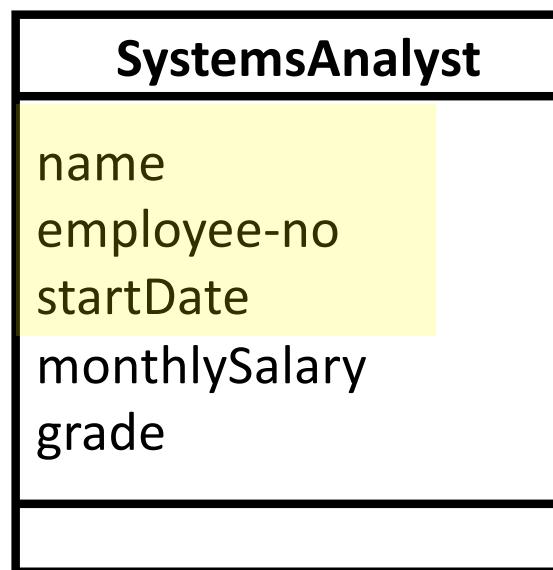
(More general bits of description are *abstracted out* from specialized classes)



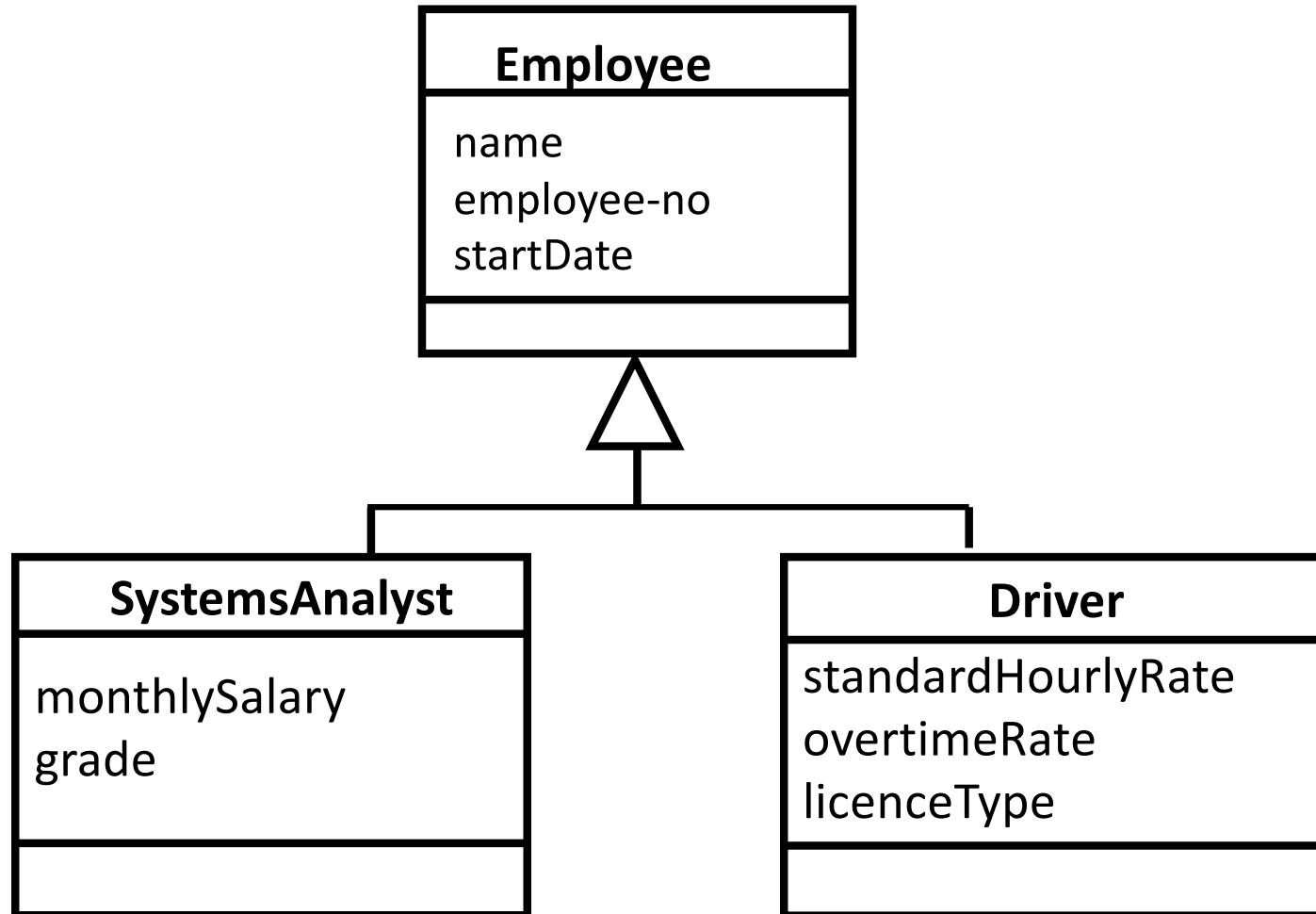
# Generalization and Specialization

**Generalization** คือ การนำเอาคุณสมบัติที่เหมือนหรือคล้ายกันของหลายๆ คลาส มาสร้างเป็นคลาสใหม่ เพื่อที่จะลดการใช้และประมวลซ้ำของข้อมูล

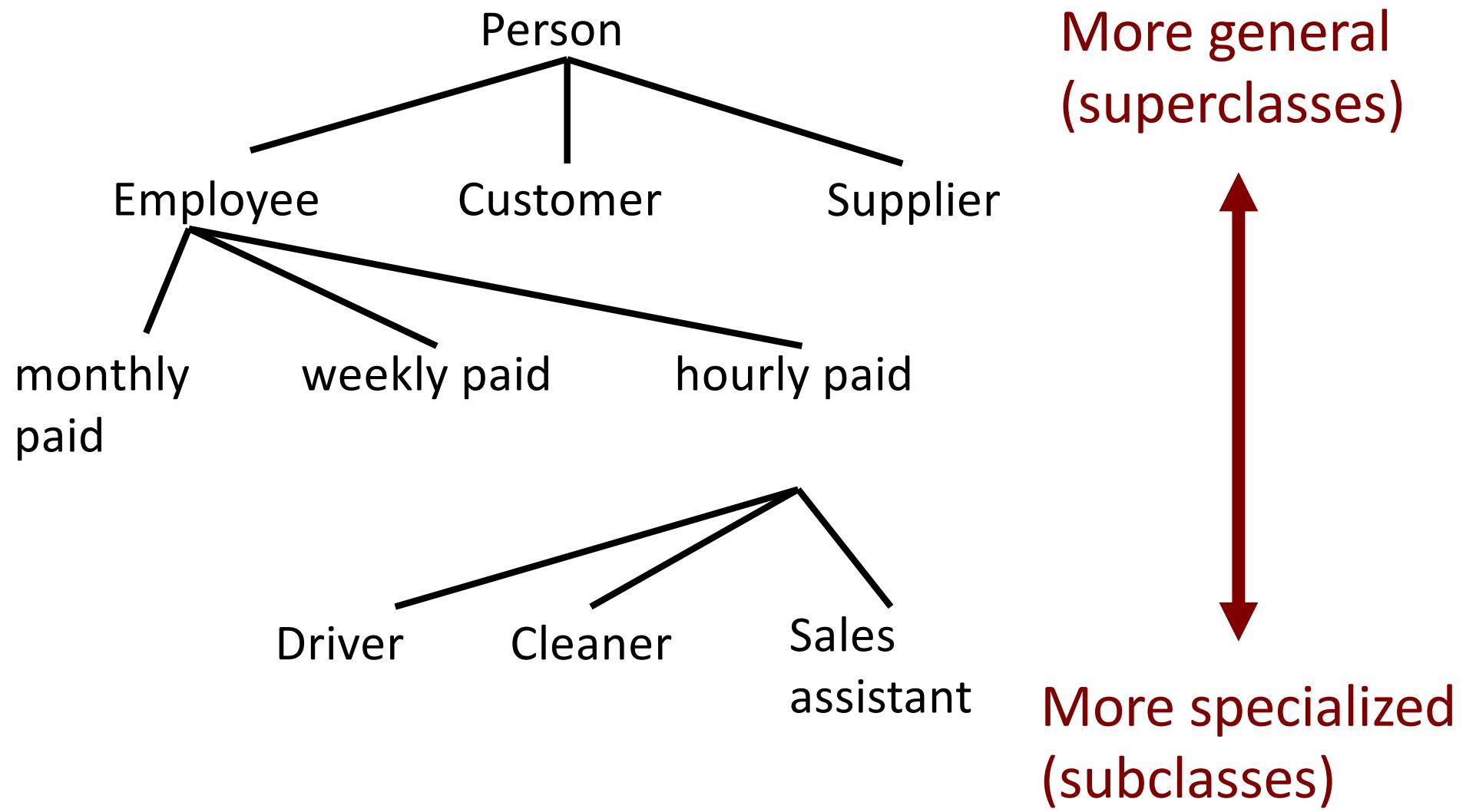
(More general bits of description are *abstracted out* from specialized classes)



# Generalization and Specialization



# Generalization and Specialization Hierarchy



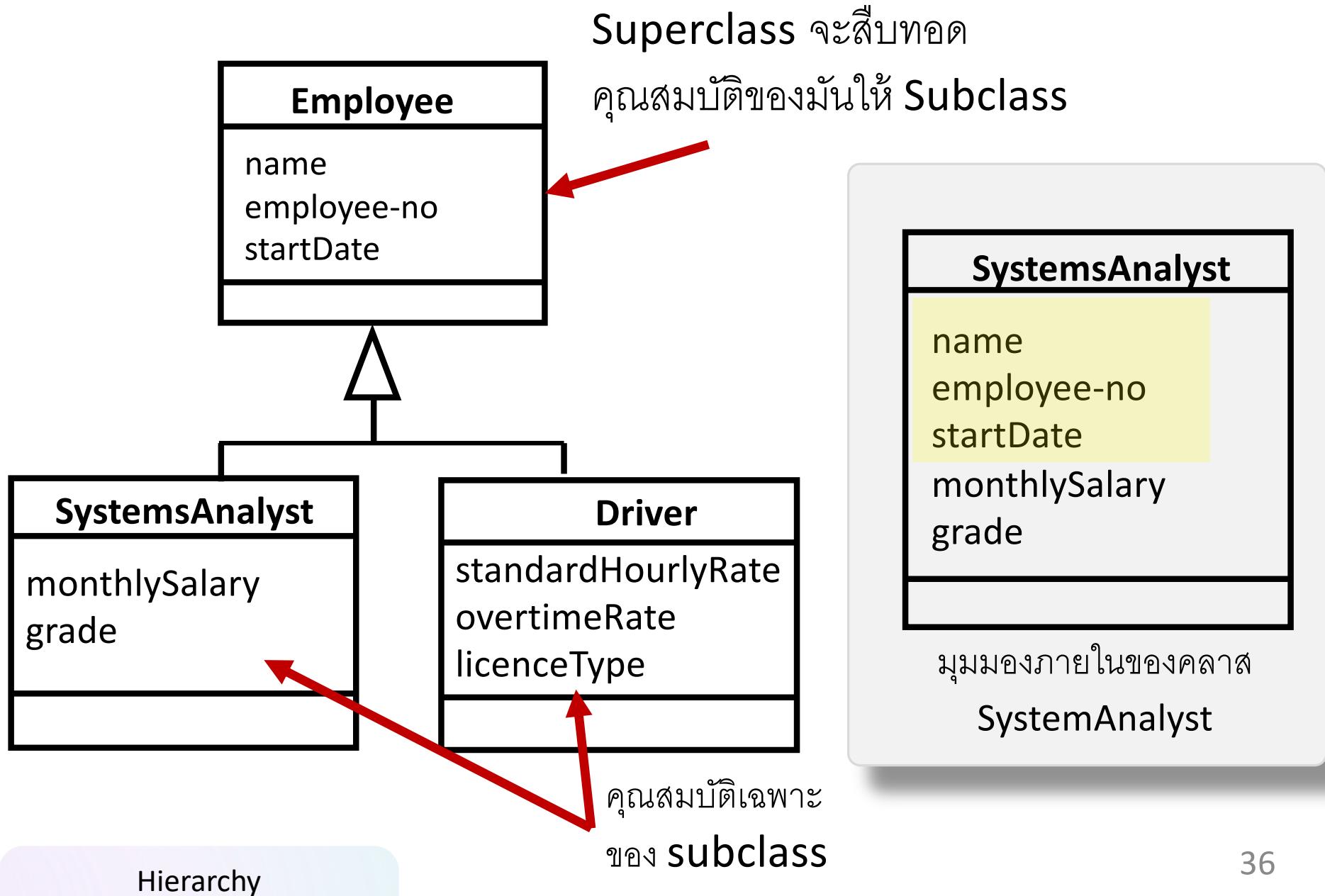
# Inheritance (การสืบทอดคุณสมบัติ)

เป็นการสร้างคลาสที่นำคุณสมบัติของคลาสพื้นฐาน (Generalization) ที่มีอยู่แล้ว มาสืบทอดต่อ แต่ทั้งนี้คลาสที่นำมาสืบทอดคุณสมบัติยังจะสามารถมีคุณสมบัติเฉพาะเพิ่มเติมได้ (Specialization)

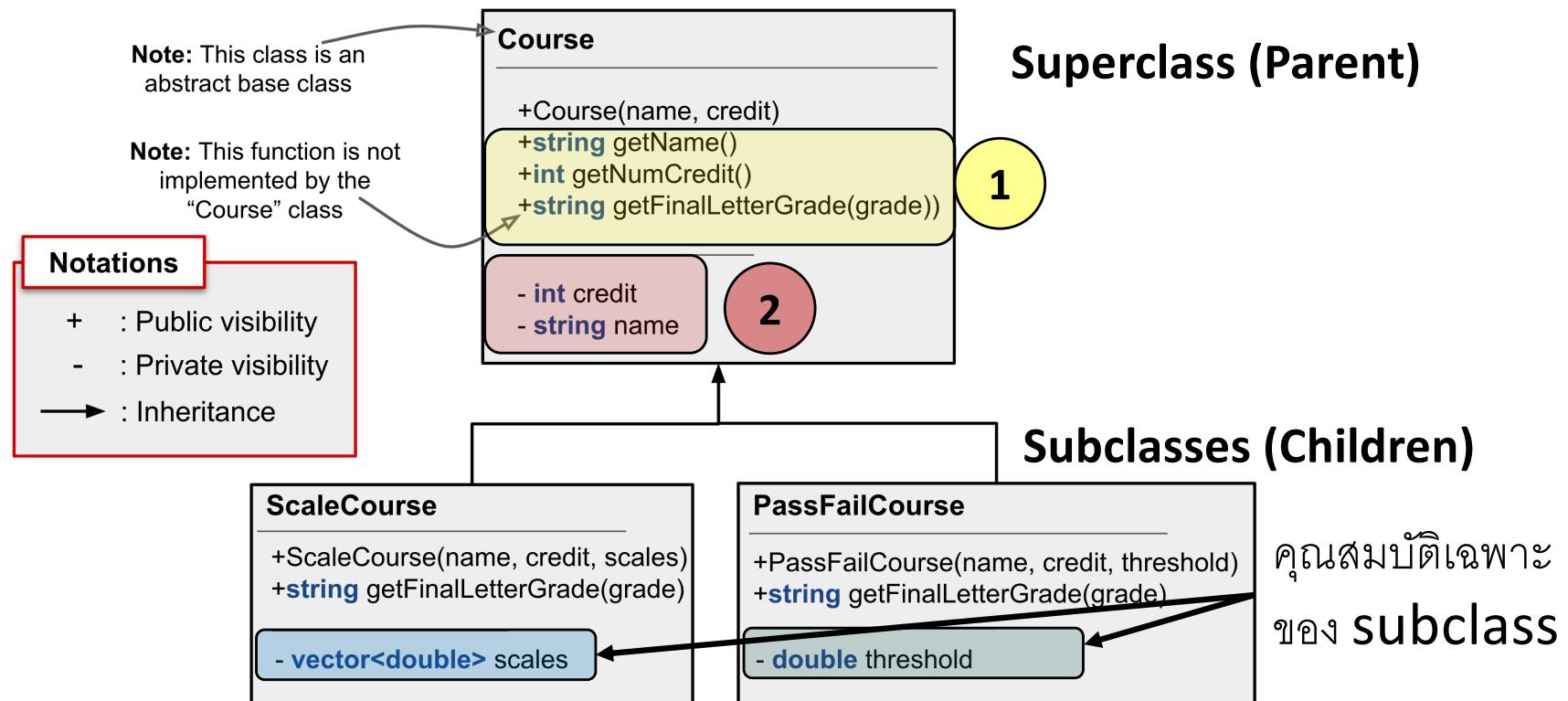
เราจะเรียกคลาสพื้นฐานที่เป็น Generalization ว่า  
**“Superclass”** (หรือ Parent)

และคลาสที่มาสืบทอดที่เป็น Specialization ว่า  
**“Subclass”** (หรือ Child)

# Inheritance (การสืบทอดคุณสมบัติ)



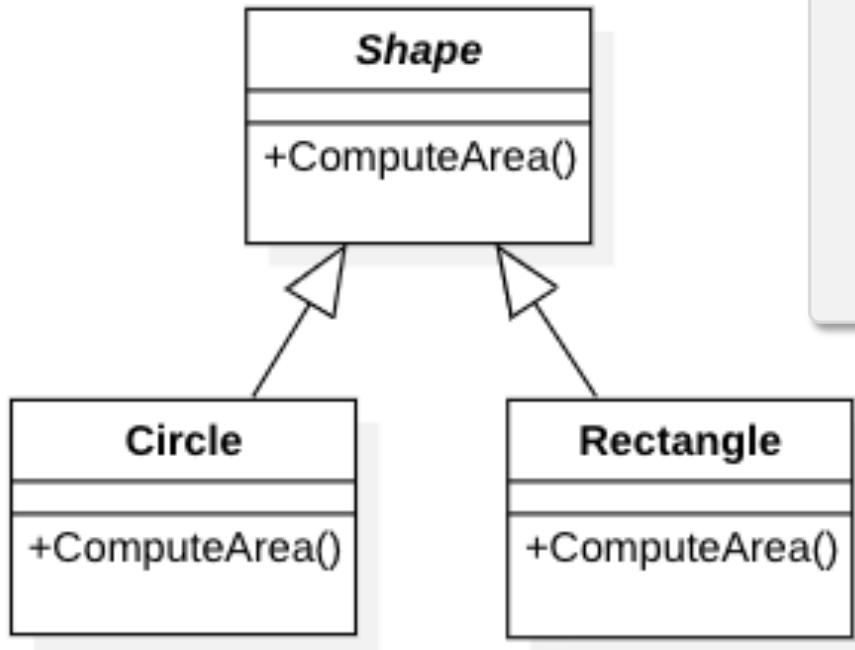
# Inheritance (การสืบทอดคุณสมบัติ)



1 คุณสมบัติต่างๆ ของ Course จะถูกได้รับการถ่ายทอดคุณสมบัติไปที่ คลาสลูกๆ Scale Course และ PassFail Course

2 แต่ทั้งนี้ก็ขึ้นอยู่กับการจำกัดข้อมูลของ Superclass ด้วย เช่น credit กับ name ไม่ได้รับการถ่ายทอดและสามารถเข้าถึงได้โดย subclass เพราะถูกตั้งเป็น private

# Polymorphism (การมีหลายรูปแบบ)



นอกจาก subclass จะสืบทอดคุณสมบัติและพฤติกรรมต่างๆของ superclass พวกมันยังจะมีอิสตระในการตอบสนองที่ต่างกันจากพฤติกรรม (method) เดียวกันได้อีก!

```
Shape *p = new Circle();
p -> ComputeArea();
```

```
Shape *q = new Rectangle();
q -> ComputeArea();
```

เพราะฉะนั้น คลาส Shape นั้น เปิด ให้คลาสอื่นมาสืบทอดคุณสมบัติ (inherit) ได้ และตัวมันก็ไม่ต้องถูกแก้หรือเปลี่ยนแปลงเพื่อที่จะรองรับพฤติกรรมที่แต่งต่างกันไปของคลาสที่มาสืบทอดมัน ตรงตาม OO principle ที่ชื่อว่า **Open/Closed principle**

# Check your understanding

Q: ความสัมพันธ์ของคลาสแบบ Inheritance มีประโยชน์หรือข้อดีอย่างไรบ้าง จงเลือกข้อที่ถูก

- ก. subclasses ที่ inherit จาก superclass เดียวกันจะได้รับคุณสมบัติต่างๆ ของ superclass เหมือนกัน
- ข. เราสามารถเปลี่ยนแปลงสิ่งต่างๆ ของ subclass ได้ง่ายขึ้นโดยการเปลี่ยนแปลงบางอย่างในตัว superclass ที่ subclass พากนั้น inherit มา
- ค. Inheritance สามารถทำให้เรานำโค้ดบางอย่างมาใช้ซ้ำโดยที่ไม่ต้อง implement โค้ดนั้นอีกรอบ

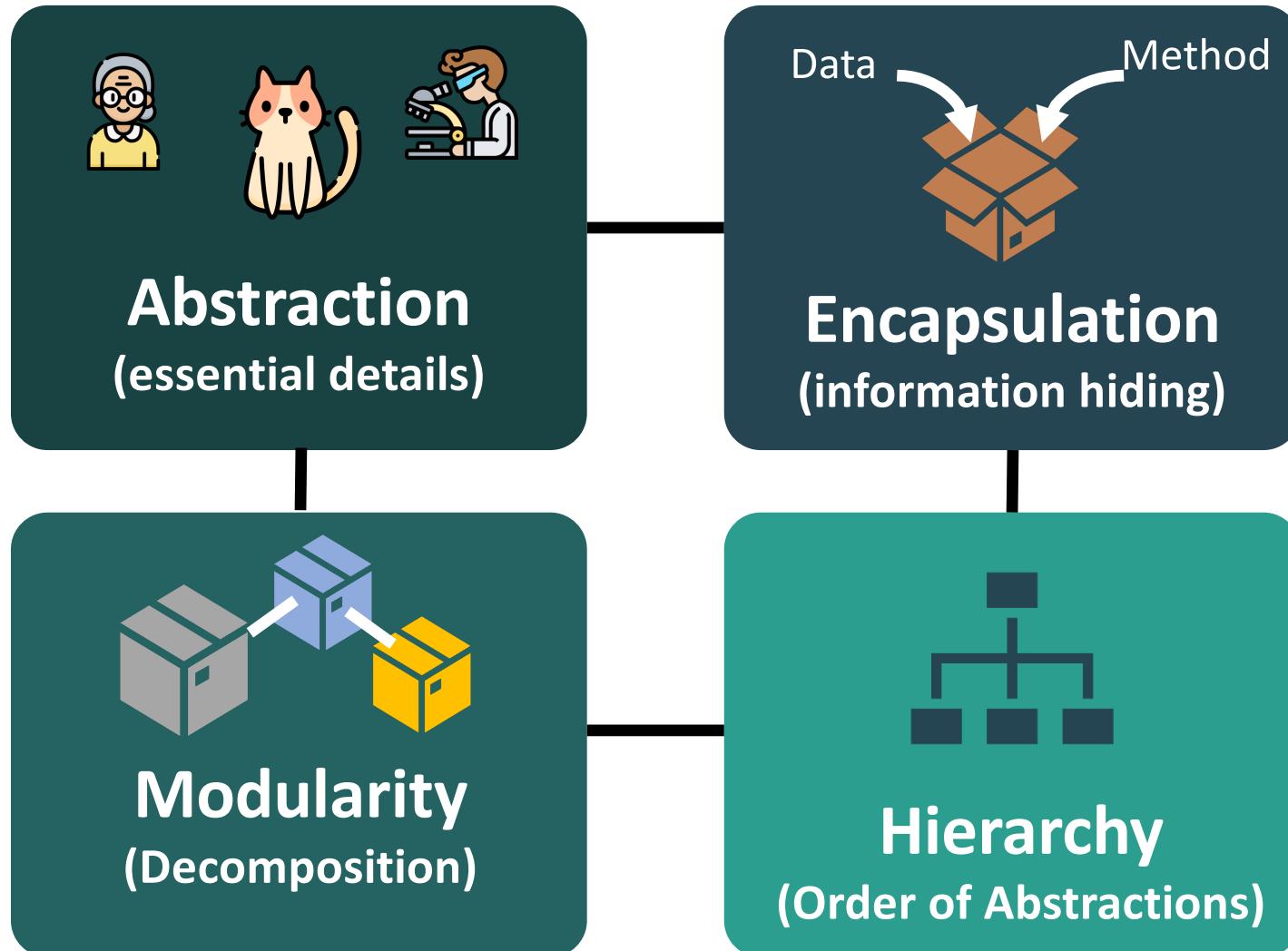
# Class Hierarchy?

**Q:** คุณได้ต่อไปนี้ที่นักศึกษาคิดว่าไม่เหมาะสมที่จะเป็นคู่ Superclass–Subclass (Parent-Child)?

- a) Bank – Account
- b) Money – US Dollars
- c) Savings Account – Checking Account
- d) Account – Account#204362
- e) Person – Customer
- f) Student – Graduate Student
- g) Course – Pass/NoPass Course
- h) Circle – Ellipse

“Object-Oriented Software Engineering” by Lethbridge and Laganiere, 2nd Ed

# Principle of OO Design



# Let's Check Your Understanding

นายวินัย ถูกจ้างให้ทำแอป ทำสำหรับ Grab เข้าตัดสินใจโมเดล Person ให้เหลือแต่ลักษณะที่สำคัญของคนขับรถส่งของ เช่น พื้นที่ที่ใช้ส่งของ (DeliveryArea), ช่วงเวลาทำงาน (WorkHour), พาหนะ (Vehicle) และพฤติกรรมสำคัญ เช่น takeOrder() หรือ drive() และตัดลักษณะและพฤติกรรมที่ไม่จำเป็นออก เช่น ไซส์รองเท้า, สีตา, ความสูง, น้ำหนัก, sleep() นายวินัยกำลังนำ Principle of OO Design ข้อไหนมาใช้ ?

TakeOrder() ✓

Shoe-size ✗

WorkHour ✓

DeliveryArea ✓

Sleep() ✗



Vehicle ✓

Eye Color ✗

Height ✗

Drive() ✓

# Let's Check Your Understanding

นายวินัยนำเข้า พฤติกรรมที่สำคัญและลักษณะสำคัญของ Person ที่เขากิด เมื่อครู่นี้ มารวม เป็น Class ชื่อว่า GrabDriver โดยที่เขากิดว่าจะเปิดให้ class อื่น เข้าถึงข้อมูลบางข้อมูล และปิดกันข้อมูลบางข้อมูลให้ใช้ได้แค่ ภายใน เอกำลังนำ **Principle of OO Design** ข้อใดมาใช้ ?

# Let's Check Your Understanding

ตอนนี้นายวินัยคิดว่าการดีไซด์ระบบ Grab ของเขารึมที่จะซับซ้อนมากขึ้น  
เขาจึงคิดวิธีแบ่งลักษณะสำคัญหรือพฤติกรรมของ GrabDriver ออกไป  
เป็นหลายคลาสที่เกี่ยวข้องกันแต่ไม่แชร์ลักษณะหรือพฤติกรรมกับ  
GrabDriver เช่น class Vehicle หรือ class Location  
เขาがらังนำ **Principle of OO Design** ข้อใดมาใช้?

# Let's Check Your Understanding

และแล้ว นายวินัยคิดว่าคลาส **GrabDriver** ของเขามีพัฒนาระบบและคุณลักษณะคล้ายๆกับ คลาสอื่นๆที่มีอยู่ในโมเดลก่อนแล้ว เช่น **FoodPandaDriver**, **LineManDriver** เขายเลยนำคุณลักษณะหรือพัฒนาระบบที่เหมือนกันของห้องสมุดมารวมกันและสร้างคลาสใหม่ชื่อว่า **Driver** เพื่อให้ที่จะให้ห้องสมุดสามารถสืบทอดพัฒนาระบบและคุณลักษณะไปเข้ากำลังนำเทคนิคของ **OO Design** ที่เรียกว่าอะไรมาใช้?

# Object Orientation (แนวคิดเชิงวัตถุ)

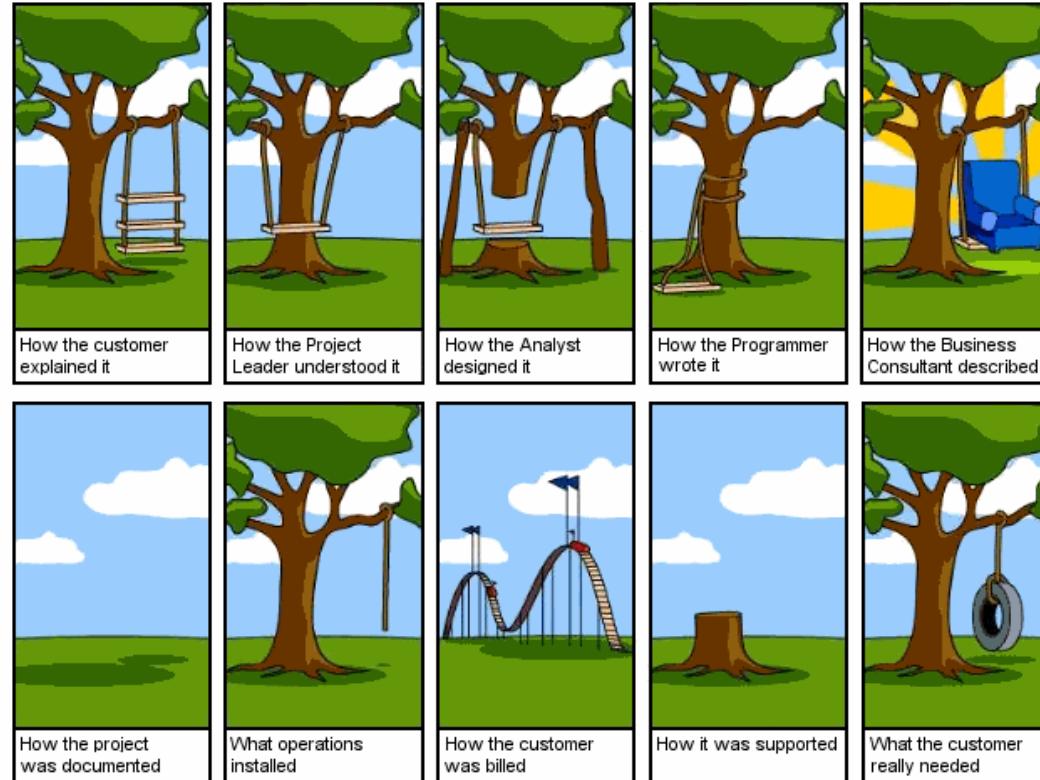


ประเด็นของแนวคิด OO Design คือเราอยากที่จะโมเดล software ของเราให้เป็น:

- 1) ระบบที่ประกอบด้วย objects (classes) ต่างๆ
- 2) Objects (classes) เหล่านั้นมีปฏิสัมพันธ์ระหว่างกัน

ว่าแต่ข้อดีของแนวคิดหรือ  
การทำแบบนี้คืออะไร?

# Why OOD?



## Manage Complexity

นำมาใช้วิเคราะห์ user requirements  
ให้เข้าใจเห็นส่วนต่างๆ ของความซับซ้อน  
ของซอฟแวร์ที่จะสร้างได้เด่นชัด

## Reduce Communication Difficulties

ใช้ Diagram ต่างๆ มาช่วยในการสื่อสาร  
กับ stakeholders ต่างๆ เพื่อยืนยัน  
ถูกต้องและความต้องการของผู้ใช้

# Why OOD?

Software ที่สร้างขึ้นจากการฝ่านการวิเคราะห์และดีไซน์เชิงวัตถุจะ:

## More Maintainable

ง่ายต่อการบำรุงรักษาขึ้น  
ผลกระทบที่เกิดจากการ  
เปลี่ยนแปลงน้อยลง เอา  
software ไปต่ออยอดได้ง่ายขึ้น

## More Reusable

ง่ายต่อการนำส่วนประกอบต่างๆมา  
ใช้ซ้ำ ไม่ต้อง implement ใหม่ ซึ่ง  
ก็จะสามารถลดเวลาและแรงงานใน  
การเขียน software ลง

# Summary

ใน Lecture นี้ นักศึกษาได้เรียนเกี่ยวกับ

a) **The fundamental concepts of object orientation and OO design:**

- Objects and classes
- Abstraction, Encapsulation, Information Hiding
- Modularity, Association
- Hierarchy, Generalization, Specialization
- Inheritance, Polymorphism

b) **Why OO Design?**

# References

- “Object Oriented Analysis and Design with Applications”, Booch et al., 3<sup>rd</sup> Edition
- “Object-Oriented Software Engineering” by Lethbridge and Laganiere, 2nd Ed
- “Object Oriented Design” – Kenny Wong, University of Alberta
- Icons from Flaticon.com