หน่วยที่ 3

การให้นิยามวัตถุและ UML Diagram (1)

Abstraction and UML Diagram (1)

เรื่องที่จะศึกษา

3.1 Classification Abstraction

3.2 Aggregation Abstraction

Concept หลักของ OOP

- Encapsulation
- Inheritance
- Polymorphism

3.1 Classification Abstraction

Concept: Encapsulation

Classification Abstraction

- อธิบายหลักการในการกำหนด Problem Domain ได้
- สามารถหา Object ใน Domain
- สามารถใช้หลักการของ Classification Abstraction ในการสร้าง Class จาก Object ที่กำหนดให้ได้
- สามารถบอกหลักการ Encapsulation และ Information Hiding ของ Class ได้

การกำหนด Problem Domain

- Problem Domain คือ ขอบเขตของสิ่งที่กำลังจะพิจารณา
- Problem Domain สามารถกำหนดได้จากการสอบถามความต้องการ (Requirement) จากผู้ใช้ระบบงานนั้น ๆ
- ถึงแม้ว่าในขั้นตอนการวิเคราะห์ระบบจะยังไม่สามารถกำหนด Problem Domain ที่ชัดเจน ก็ขอให้กำหนด Big Picture ของ Problem Domain ให้ได้ออกมาก่อน
- ถ้ากำหนดภาพรวมของ Problem Domain ไม่ได้ ก็จะมี class จำนวนมากมายมหาศาลเกิดขึ้น จนยากที่จะออกแบบ software ที่ดีได้

การค้นหา Object ใน Domain

- ทำได้โดยการค้นหาคำนามทั้งหมดที่มีใน Problem Domain
- แยกแยะว่าสิ่งใดคือ Object และสิ่งใดคือ Attribute
- ระวัง!! คำนามบางคำก็เป็น Object แต่บางคำเป็น Attribute

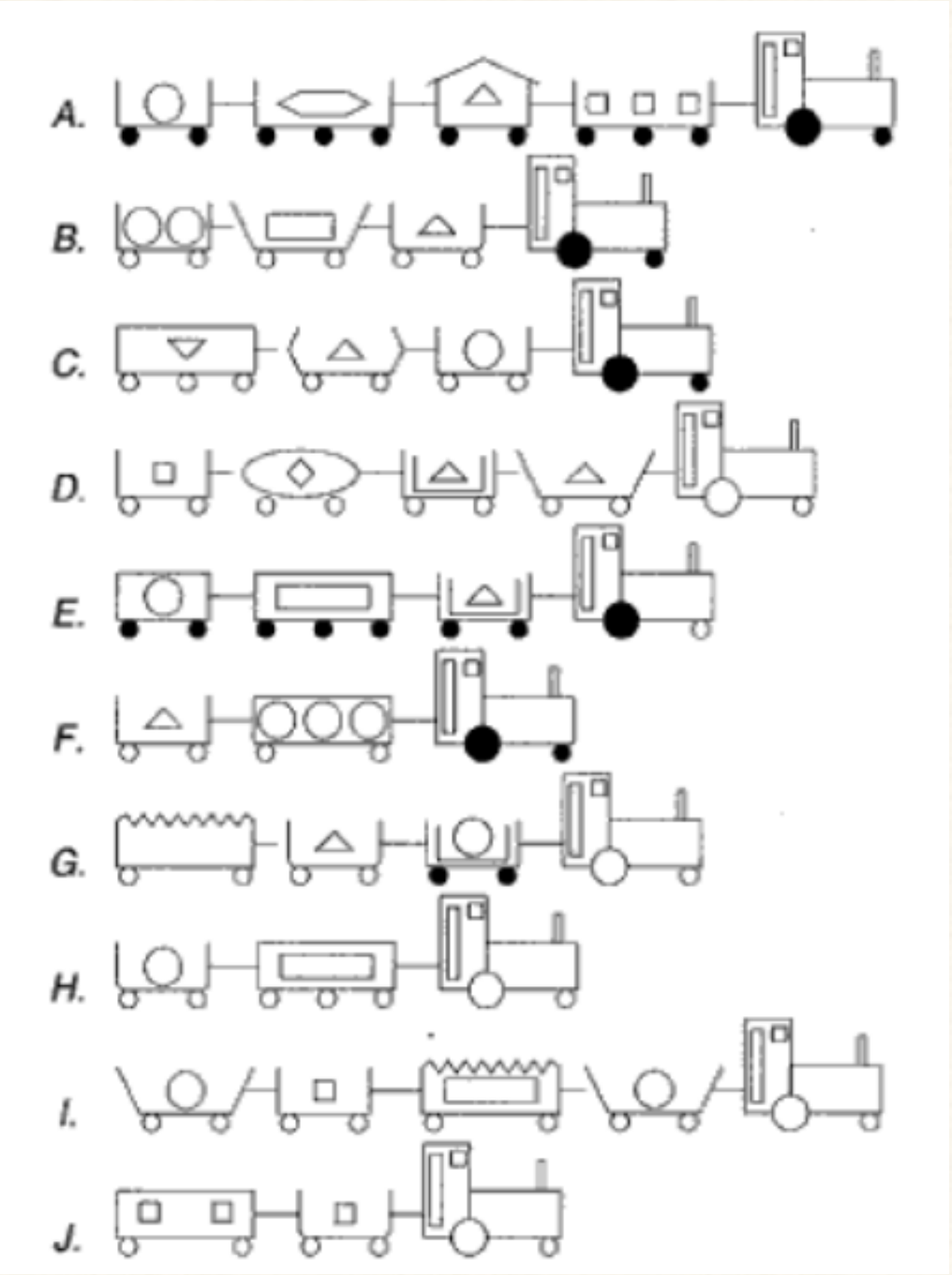
ความยากในการทำ Classification

- โดยทั่วไป เราแยกแยะวัตถุโดยใช้เงื่อนไขบางอย่างมากำหนดเป็นกรอบ
  - แล้ว ขอบเขตของแขน ขา อยู่ตรงไหน???
  - เราแยกเสียงเครื่องดนตรีในเพลงได้ไหม???
  - ในเอกสารหนึ่งชิ้น เราใช้อะไรแยกแยะประเภทของเอกสาร????

มะเขือ เป็น ผัก หรือ ผลไม้ ???

- ในทางพฤกษศาสตร์
- ในทางโภชนาการ

จงทำ classification ของรถไฟในภาพ



ประเภทของ Object

- สิ่งที่มีตัวตนสามารถจับต้องได้ (Tangible Objects)
  - คน สุนัข รถยนต์
  - อื่น ๆ (ให้นักศึกษายกตัวอย่าง)
- สิ่งที่ไม่มีตัวตนและไม่สามารถจับต้องได้ (Intangible Objects)
  - บทบาท เหตุการณ์ ปฏิสัมพันธ์
  - อื่น ๆ (ให้นักศึกษายกตัวอย่าง)

ตัวอย่างที่ 1

- “หนังสือเล่มหนึ่ง ปกสีเหลือง ภายในประกอบด้วยเนื้อหาเกี่ยวกับ Object Orientation หนังสือเล่มนี้มีจำนวน 50 หน้า”

ตัวอย่างที่ 1 : การวิเคราะห์ (1)

- ขั้นตอนที่ 1 : หาคำนาม
  - หนังสือเล่มหนึ่ง
  - ปกสีเหลือง
  - เนื้อหาเกี่ยวกับ Object Orientation
  - หน้า

ตัวอย่างที่ 1 : การวิเคราะห์ (2)

- ขั้นตอนที่ 2 : แยกประเภทของคำนาม
  - หนังสือเล่มหนึ่ง : Object
  - ปกสีเหลือง : Attribute
  - เนื้อหาเกี่ยวกับ Object Orientation : Attribute
  - หน้า : Attribute

การระบุ Class และ Object

- ในบาง Problem Domain อาจจะได้ทั้ง Class และ Object ในเวลาเดียวกัน ดังนั้น จำเป็นต้องระบุให้แน่ชัดว่าสิ่งใดคือ Class และสิ่งใดคือ Object

ตัวอย่างที่ 2

- “พยาบาลชื่อ ปราณี ฉีดยาป้องกันโรคบาดทะยักให้แก่คนไข้ชื่อ กิตติ”

ตัวอย่างที่ 2 : การวิเคราะห์ (1)

- ขั้นตอนที่ 1 : หาคำนาม
  - พยาบาลชื่อปราณี
  - ยาป้องกันโรคบาดทะยัก
  - คนไข้ชื่อกิตติ

ตัวอย่างที่ 2 : การวิเคราะห์ (2)

- ขั้นตอนที่ 2 : แยกประเภทของคำนาม
  - พยาบาล : Class
  - ปราณี : Object หนึ่งใน Class พยาบาล
  - คนไข้ : Class
  - กิตติ : Object หนึ่งใน Class คนไข้
  - ยาป้องกันโรคบาดทะยัก : Class / Object

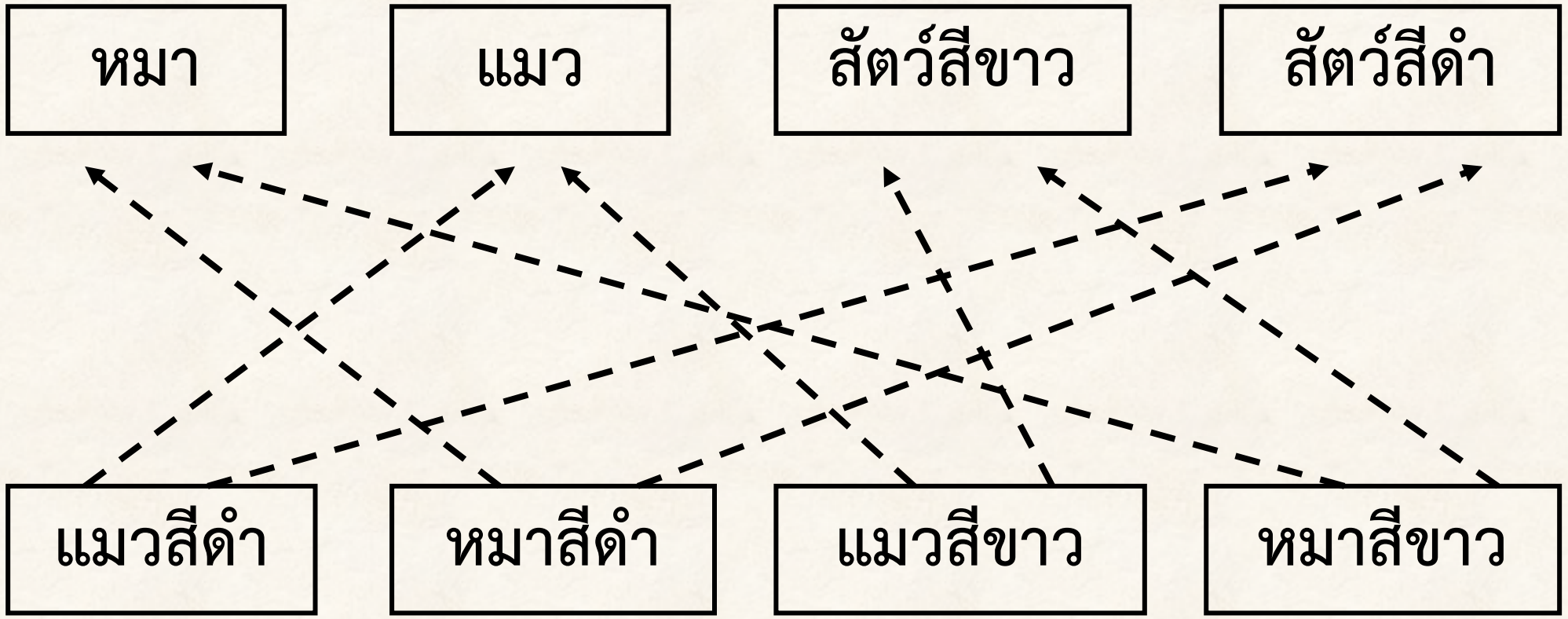
ตัวอย่างที่ 2 : การวิเคราะห์ (3)

- ยาป้องกันโรคบาดทะยัก : Class
  - เพราะเป็นการบอกอย่างกว้าง ๆ ว่าเป็นยาป้องกันบาดทะยัก แต่ไม่ได้ระบุยี่ห้อยา
- ยาป้องกันโรคบาดทะยัก : Object
  - ถือเป็น Object หนึ่งใน Class ยา

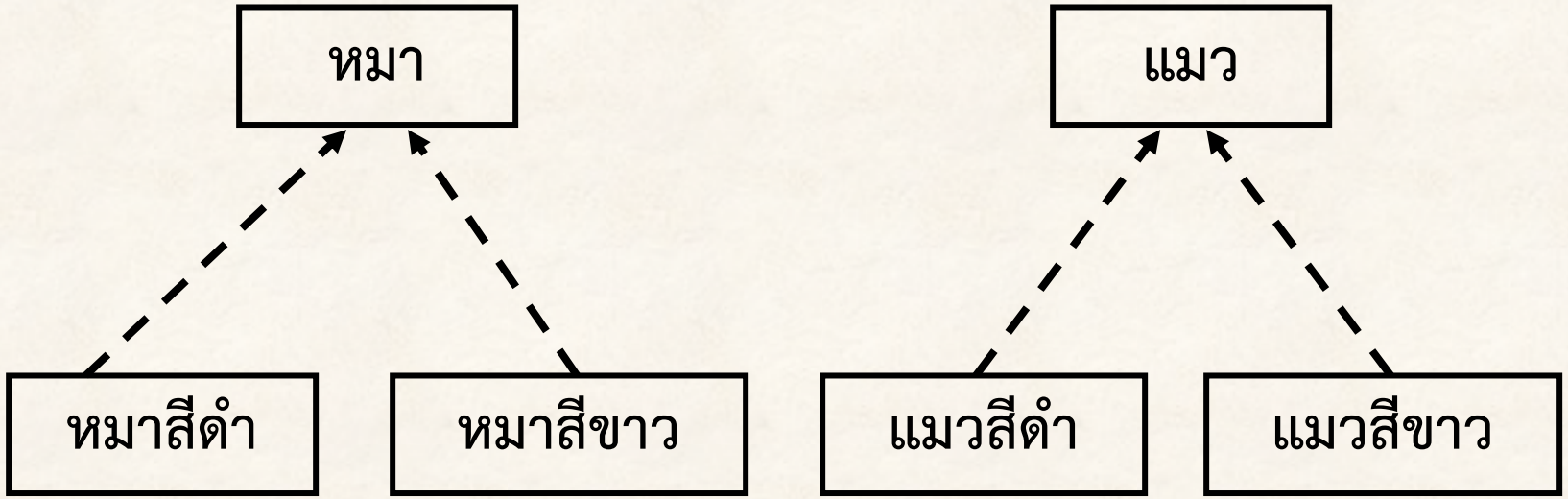
Classification Abstraction

- เป็นกระบวนการในการค้นหาว่ามี Object ใดบ้างใน Problem Domain
- เป็นการจำแนกแยกแยะว่า Object แต่ละตัวจัดอยู่ใน Class ใดบ้าง
- การทำ Classification Abstraction แสดงด้วยสัญลักษณ์ ลูกศรประที่ลากจาก Objects ไปยัง Class

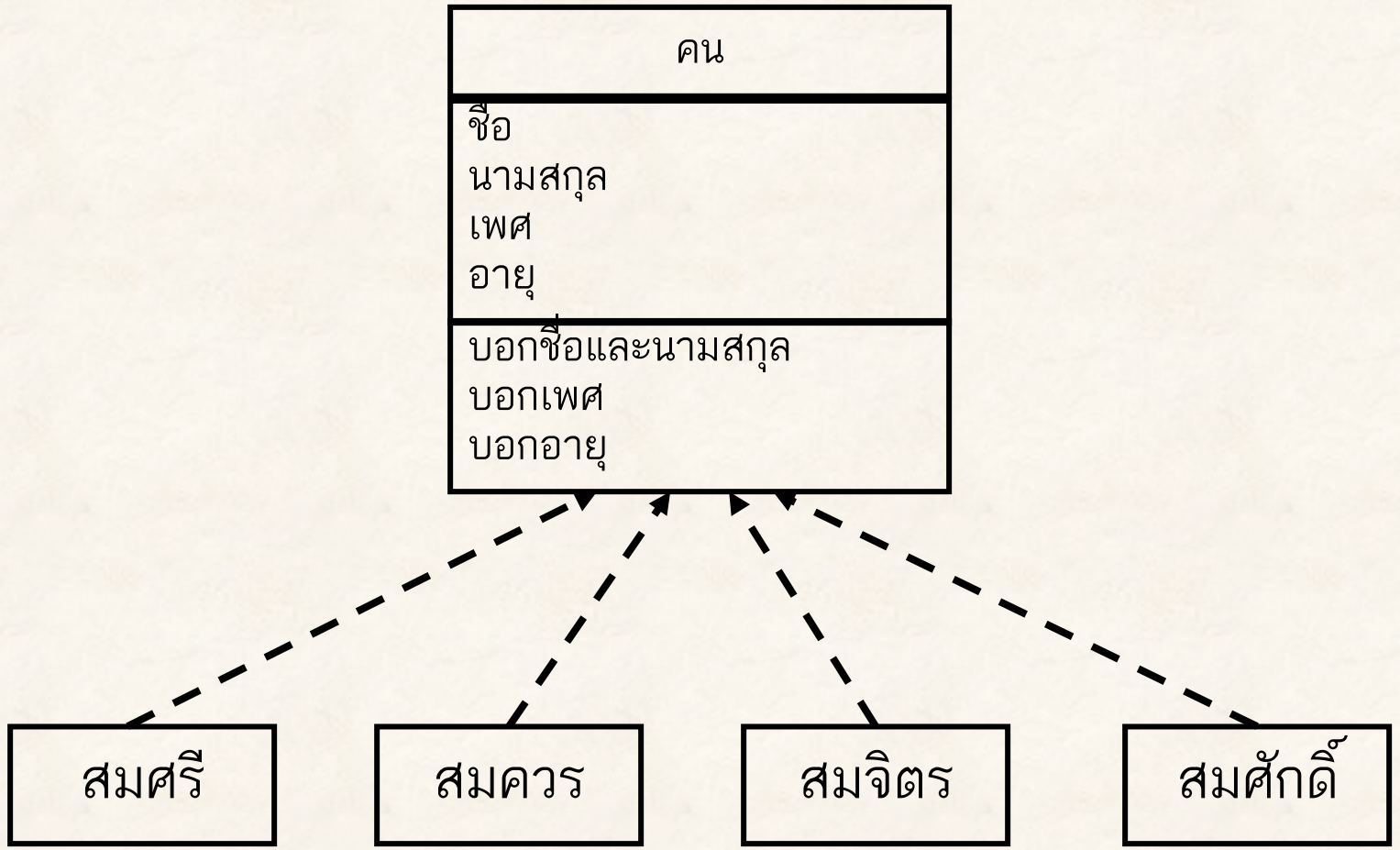
ตัวอย่างที่ 3 การทำ Classification



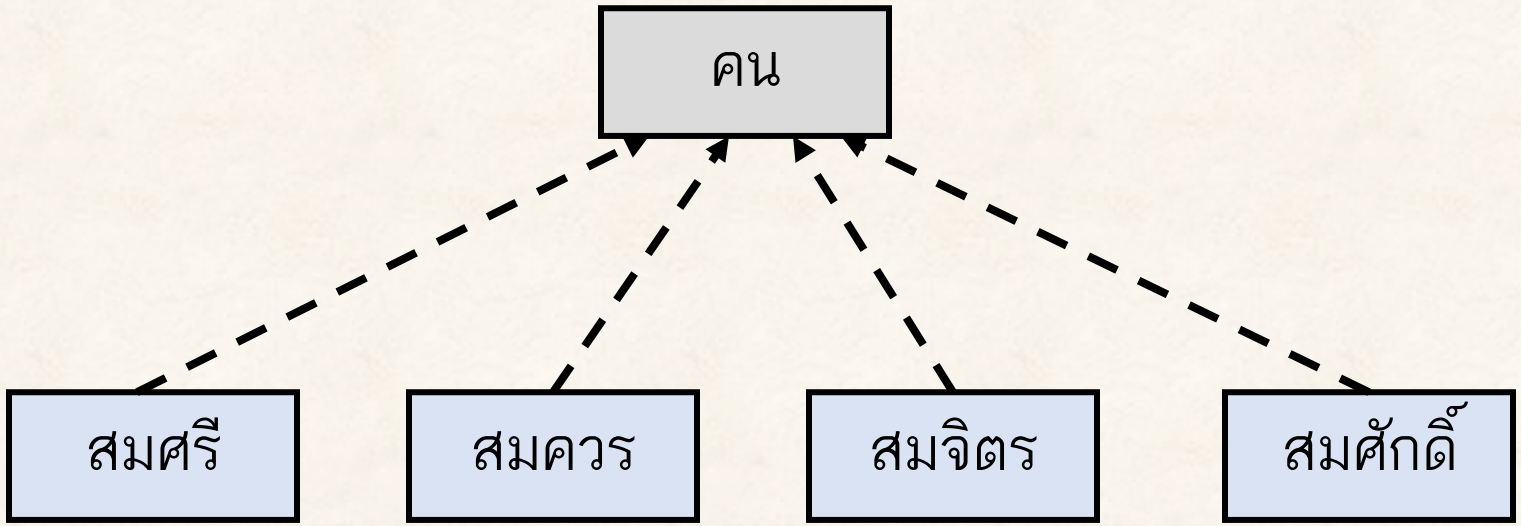
ปรับปรุงการทำ Classification ของหมาและแมว



ตัวอย่างที่ 4 Classification ของ Class คน



การสร้างวัตถุจาก Class คน

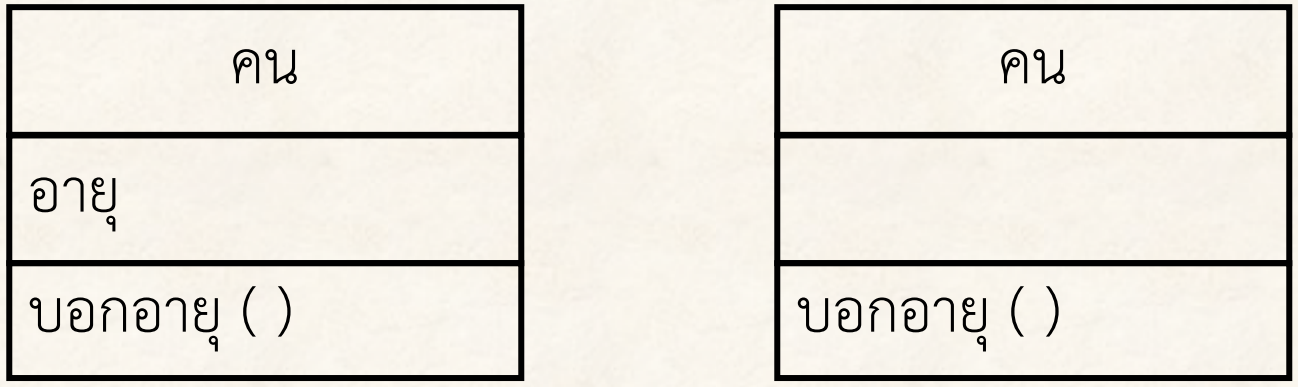


Encapsulation

Encapsulation

- Encapsulation เปรียบเสมือนกับการนำเปลือกมาครอบ Attributes และ Function ของ Class เอาไว้
  - ลักษณะของเปลือก
    - เปลือกใส จะสามารถมองได้จากภายนอก
    - เปลือกทึบ จะไม่สามารถเห็นได้จากภายนอก
- ภาพของ Class ที่มองเห็นได้จากภายนอกนั้นเรียกว่า Outside View

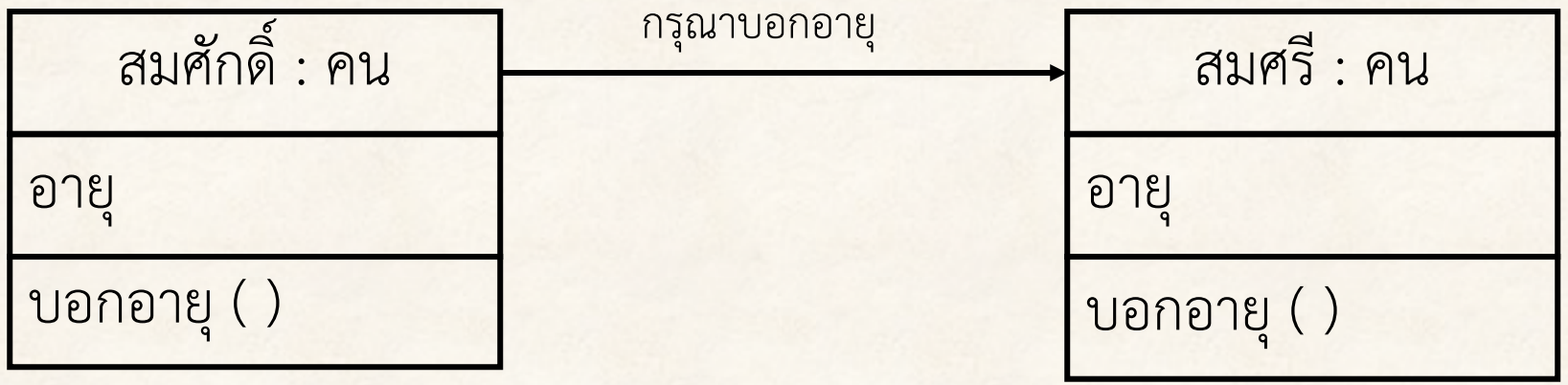
Outside View ของ Class คน



Class คน

Outside View ของ Class คน

การขอดู Attribute โดยการใช้ Method



ในบางภาษาเช่น C++ จะเรียก Method ว่า Member Function

การเขียน Function ที่ถูกใช้งาน

สมศักดิ์ : คน
อายุ
บอกอายุ ( )

สมศรี : คน
อายุ
บอกอายุ ( )

ส่วนของ Function

นิยมเขียนด้วยตัวเอียง

Information Hiding

- การซ่อนคุณสมบัติของ Object
- เมื่อต้องการเข้าถึง Attribute บางตัวของ Object นั้น จะต้องทำผ่าน method ที่สามารถมองเห็นและเรียกใช้ได้เท่านั้น
  - Attributes/Properties ควรหุ้มด้วยเปลือกทึบ
  - Functions/Methods ควรหุ้มด้วยเปลือกใส

ประเภทของ Attribute และ Methods

- จำแนกตามความสามารถในการเห็นและเข้าถึง Attributes และ Methods เหล่านั้น (Visibility) ได้ 3 ประเภท
  - Private Attributes and Functions
  - Protected Attributes and Functions
  - Public Attributes and Functions

Private Attributes and Methods

- เป็น Attributes และ Method ที่ไม่สามารถเห็นได้เลยจากภายนอก
- การเข้าถึง Attribute เหล่านี้ได้ต้องผ่านทาง Method ที่มีไว้เท่านั้น
- จะใช้เครื่องหมาย (-) กำกับไว้หน้า Private Attribute และ Private Method
  - เช่น อายุของคน

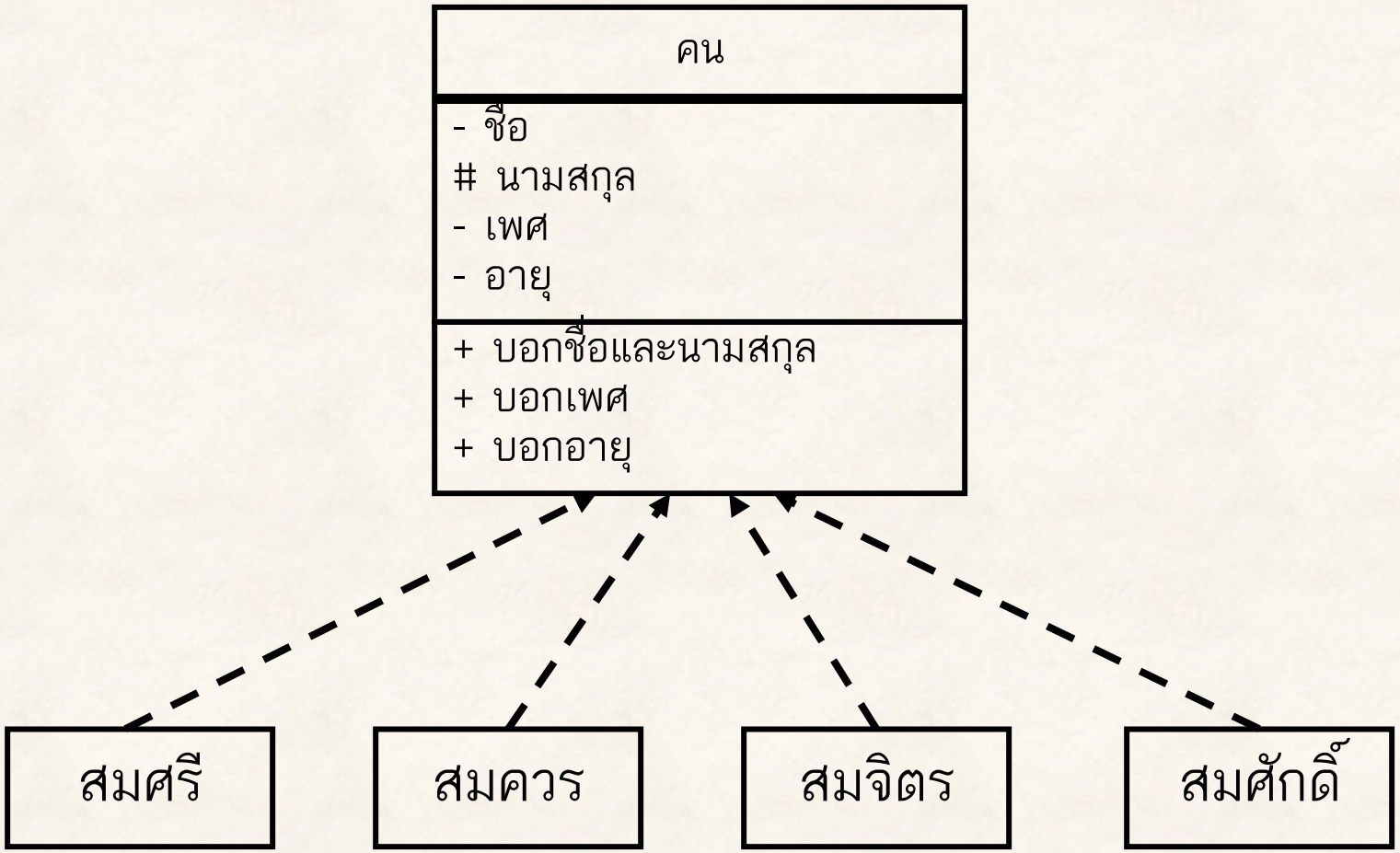
Protected Attributes and Functions

- เป็น Attributes และ Methods ที่ไม่สามารถเห็นได้จากภายนอกแต่เป็นส่วนที่สามารถส่งต่อให้ Inherited Class ได้เท่านั้น
- จะใช้เครื่องหมาย (#) กำกับไว้หน้า Protected Attribute และ Protected Methods
  - เช่น ลักษณะทางกรรมพันธุ์ที่ลูกสืบทอดมาจากพ่อแม่

Public Attributes and Functions

- เป็น Attributes และ Methods ที่สามารถมองเห็นได้และสามารถเรียกใช้ได้โดยตรงจากภายนอก
- จะใช้เครื่องหมาย (+) กำกับไว้หน้า Public Attribute และ Public Method
  - เช่น สีผม สีผิว

Classification ของ Class คน



3.2 Aggregation Abstraction

3.2 Aggregation Abstraction

- อธิบายหลักการแยกและประกอบคลาสด้วยวิธีการ Aggregation Association ได้
- อธิบายและใช้งาน Cardinality, Required และ Optional Components ได้

Aggregation Abstraction

- ในโลกความจริง วัตถุจะเกิดจากการประกอบกันเข้าของวัตถุหลายๆ ชนิด
  - ประกอบแบบไม่สามารถแยกชิ้นส่วน (มาใช้งาน) ได้
    - เช่น คอนกรีต ประกอบด้วยหิน ทราย ปูนซิเมนต์ และ น้ำ (เราไม่สามารถเอาปูนซิเมนต์ออกมาจากคอนกรีต เพื่อใช้งานใหม่ได้)
  - ประกอบแบบแยกชิ้นส่วน (มาใช้งาน) ได้
    - เช่น โคมไฟ ประกอบหลอดไฟ สวิตช์ สายไฟ สตาร์ทเตอร์ บัลลาสต์ (เราสามารถแยกส่วนประกอบต่างๆ ไปใส่ในโคมไฟอื่น หรือนำไปใช้ที่อื่นได้ หากมีขนาดเท่ากัน)

กรณีศึกษา

- ให้นักศึกษา List วัตถุที่เกิดจากการรวมกันของวัตถุอื่น
  - แบบแยกส่วนนำมาใช้ใหม่ได้
  - แบบไม่สามารถแยกส่วนนำมาใช้ใหม่ได้

Concept ของวัตถุแบบ Aggregation

- เมื่อนำ Class มาประกอบกันแบบ Aggregation จะทำให้เกิด Concept ที่ต่างออกไปแก่ Class ใหม่
  - การนำ ทราย หิน ปูน น้ำ มาประกอบเป็นคอนกรีต จะได้วัตถุที่มี Concept ต่างไปโดยสิ้นเชิง
  - การนำ โต๊ะ เก้าอี้ กระดาน มาประกอบเป็นห้องเรียน จะต่างจากการนำโต๊ะ เก้าอี้ มาประกอบกันเป็นห้องรับประทานอาหาร

Composition VS. Decomposition

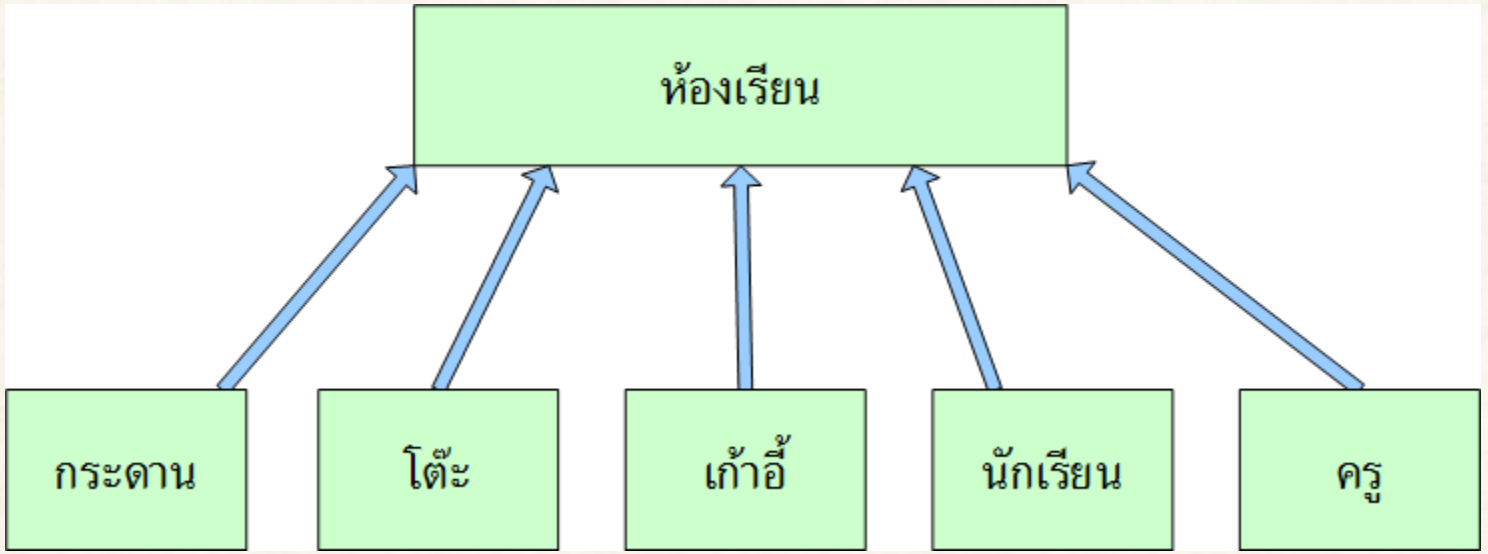
- Composition : การนำ Class มาประกอบกันเพื่อให้ได้ Class ใหม่ตาม Concept ที่กำหนด
  - การนำ ล้อรถ เครื่องยนต์ ตัวถัง ระบบขับเคลื่อน มารวมกัน จะทำให้ได้คลาส รถยนต์
- Decomposition : การจำแนก Class เพื่อให้รู้ว่า Class ที่มี Concept นั้น ประกอบด้วยคลาสอะไรบ้าง
  - เมื่อกำหนด Concept ของรถยนต์ เราก็จะทราบว่า ควรมีล้อ เครื่องยนต์ ฯลฯ

ตัวอย่าง 5

- “ห้องเรียนประกอบไปด้วย กระดานดำ 1 กระดาน มีโต๊ะและเก้าอี้จำนวนหนึ่ง มีนักเรียน มีครู”

Composition

- จากข้อความข้างต้น สามารถสรุปได้ว่า class กระดานดำ class โต๊ะ class เก้าอี้ class นักศึกษา class อาจารย์ เมื่อนำมารวมกันจะได้ class ใหม่ คือ class ห้องเรียน (Concept ต่างไปจากเดิม)



Decomposition

- Class ห้องเรียนสามารถแบ่งออกได้เป็น
  - Class กระดานดำ
  - Class โต๊ะ
  - Class เก้าอี้
  - Class นักศึกษา
  - Class อาจารย์

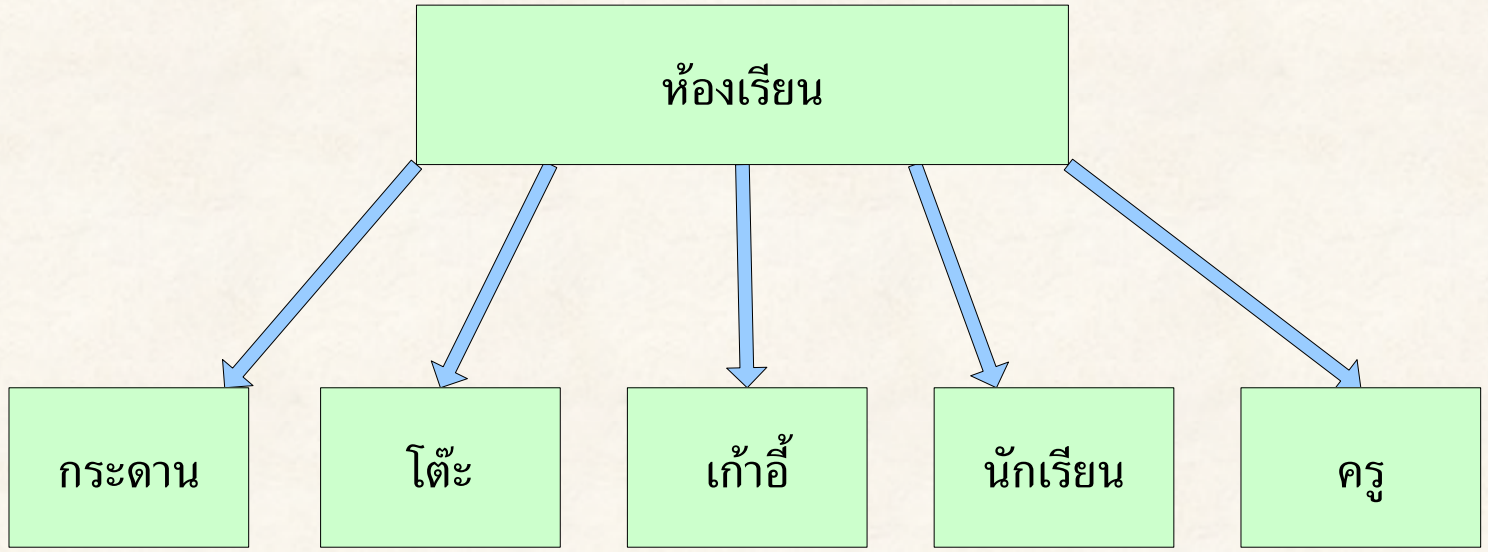
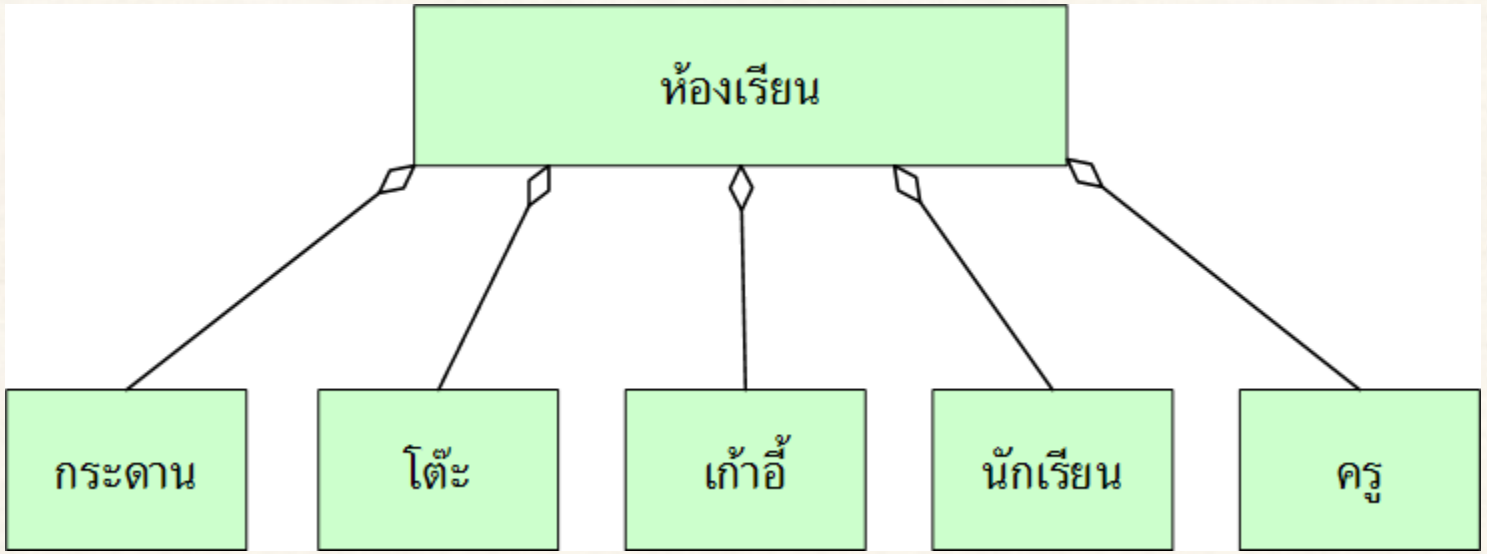


Diagram ของ Aggregation Abstraction

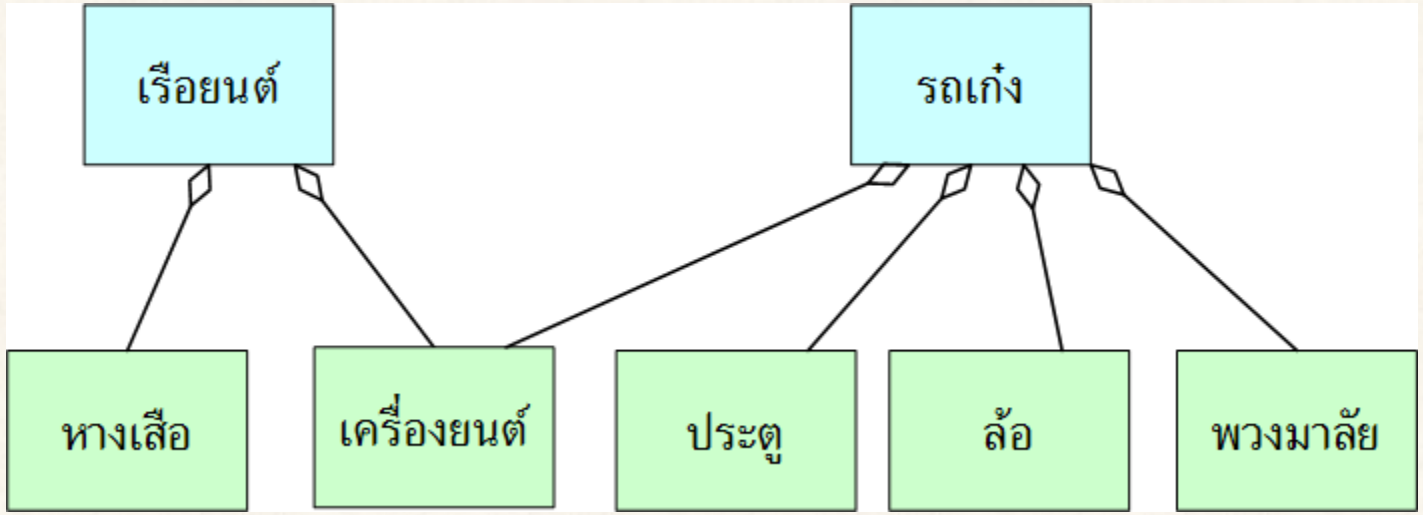
- ใช้เส้นตรงที่มีหัวสี่เหลี่ยมขนมเปียกปูน

- ลากจาก Class ย่อย ไปยัง Class หลัก



Advances Aggregation Abstraction

- อาจมี Class ที่เป็น Class ย่อยของหลายๆ คลาสใหญ่ซึ่งมี Concept ต่างกัน



สรุป Aggregation Abstraction

- คือ การ พยายามตอบคำถามที่ว่า มี class ใดเป็นส่วนประกอบ (Is part of) ของ class อื่นหรือไม่ และที่สำคัญ “การประกอบกันของ class ต้องทำให้เกิด class ใหม่ ซึ่งมี concept ใหม่ด้วย”
- ในทาง object orientation นั้น การแสดงสัญลักษณ์เพื่อแสดง Aggregation Abstraction ของ class นั้น ทำได้โดยการโยงลูกศรเป็นสี่เหลี่ยมขนมเปียกปูน จาก class ย่อยหรือ class ที่เป็นส่วนประกอบ (Composite class) ไปยัง Class หลัก (Main Class)

กิจกรรม

- ให้นักศึกษาวาดแผนภาพ Composition ของเครื่องคอมพิวเตอร์
- ให้นักศึกษาวาดแผนภาพ Decomposition ของหนังสือ 1 เล่ม

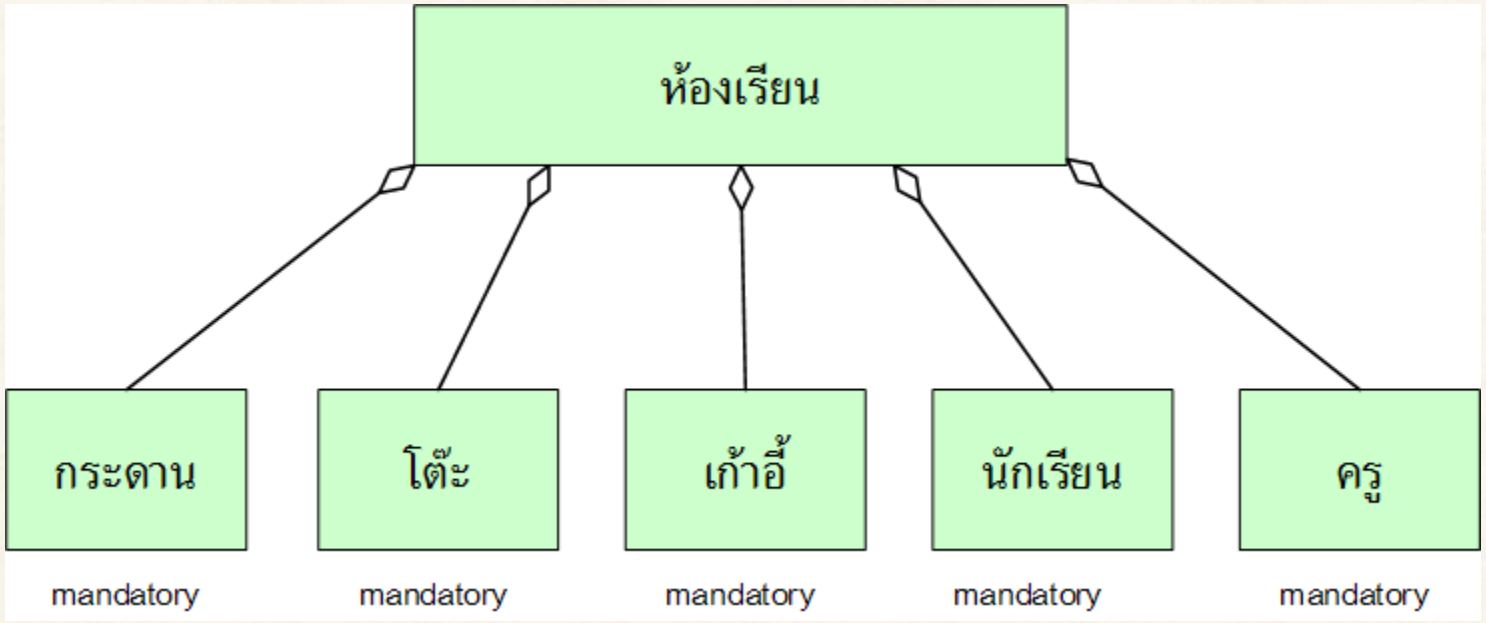
Cardinality, Required & Optional Components

- การประกอบกันของ class หรือความสัมพันธ์เชิง is part of
  - อาจจะประกอบไปด้วย class ย่อย (Composite class) ชนิดที่หนึ่ง เพียงชิ้นเดียว
  - class ย่อยชนิดที่สอง จำนวน 4 ชิ้นขึ้นไป
  - Class ย่อยชนิดที่สาม ไม่จำกัดจำนวน (หรืออาจไม่มีเลยก็ได้)
- สิ่งที่ใช้ในการแสดงจำนวนสมาชิกของ Object ในความสัมพันธ์ ดังกล่าวนี้เรียกว่า Cardinality

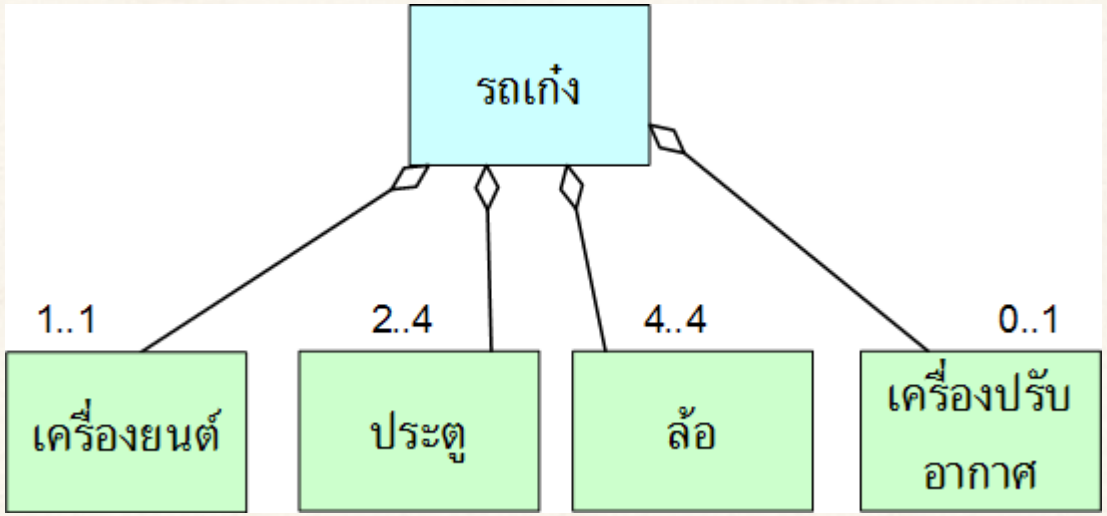
Cardinality, Required & Optional Components

- ในทาง Object-Oriented นิยมเรียก Class ย่อย ว่า Component
  - ส่วนประกอบที่จำเป็นต้องมี เรียกว่า Required หรือ Mandatory Component
- รถยนต์จำเป็นต้องมีเครื่องยนต์ ถ้าไม่มีเครื่องยนต์ รถยนต์ก็ไม่สามารถวิ่งได้
  - ส่วนประกอบที่ไม่จำเป็นต้องมี เรียกว่า Optional Component
  - เครื่องปรับอากาศในรถยนต์ไม่จำเป็นต้องมีก็ได้ ถึงไม่มีเครื่องปรับอากาศรถยนต์ก็ยังสามารถวิ่งได้

Cardinality, Required & Optional Components



Cardinality



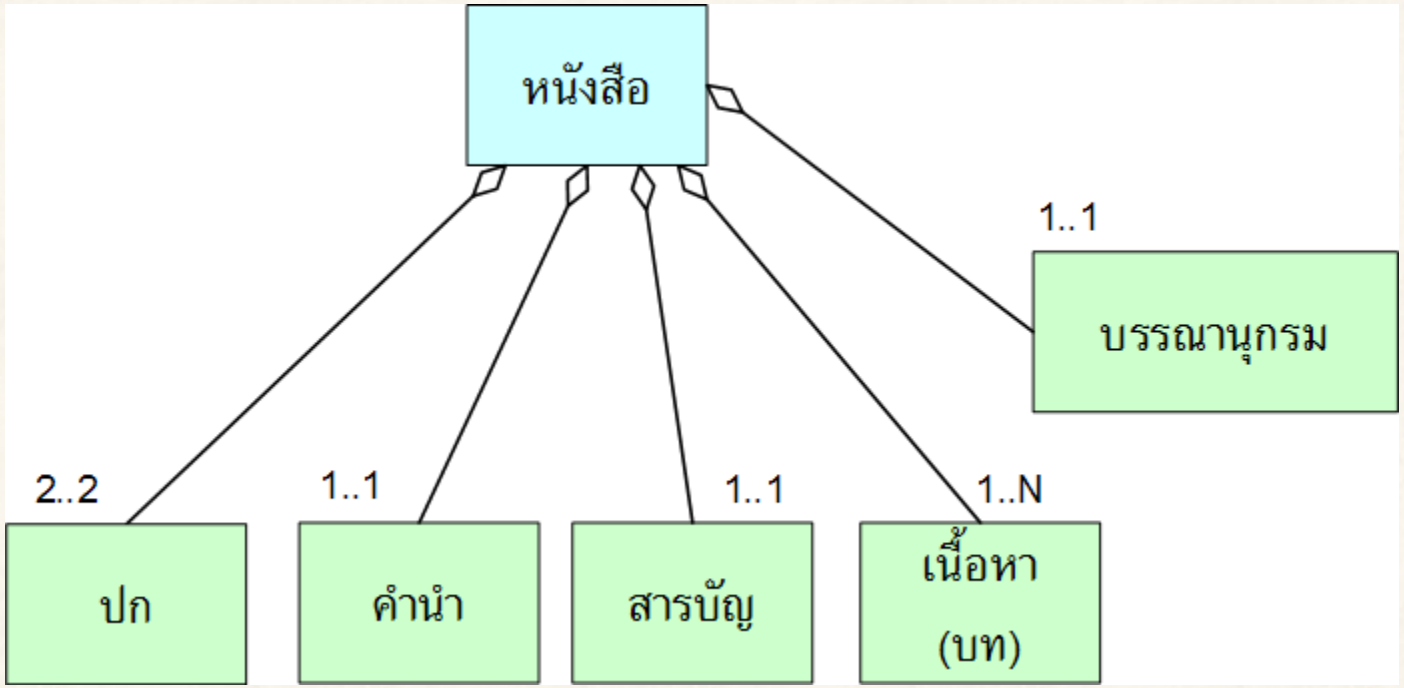
Maximum & Minimum Cardinality

- Maximum Cardinality (Max-card): จำนวนมากที่สุดของ Components ที่สามารถมีได้
  - เท่ากับ N
- Minimum Cardinality (Min-card): จำนวนน้อยที่สุดของ Components ที่สามารถมีได้
  - เท่ากับ 0 (ศูนย์)

การอ่าน Cardinality

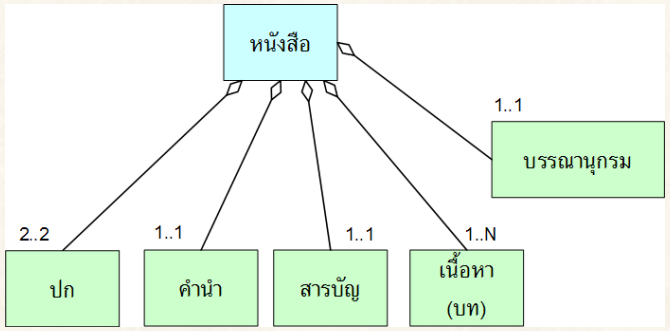
- <maximum | minimum> Cardinality ของ <ชื่อ Component> ใน aggregation <ชื่อ Class หลัก>-<ชื่อ component> มีค่าเท่ากับ <ค่าของ cardinality> เช่น
  - Minimum Cardinality ของ ประตู ใน Aggregation รถ-ประตู, มีค่าเป็น 2 และ Maximum Cardinality ของ ประตู ใน Aggregation รถ-ประตู, มีค่าเป็น 2
  - Minimum Cardinality ของ นักเรียน ใน aggregation ห้องเรียน-นักเรียน มีค่าเป็น 0 และ Maximum Cardinality ของ นักเรียน ใน aggregation ห้องเรียน-นักเรียน มีค่าเป็น n เมื่อ n เป็นจำนวนใดๆ

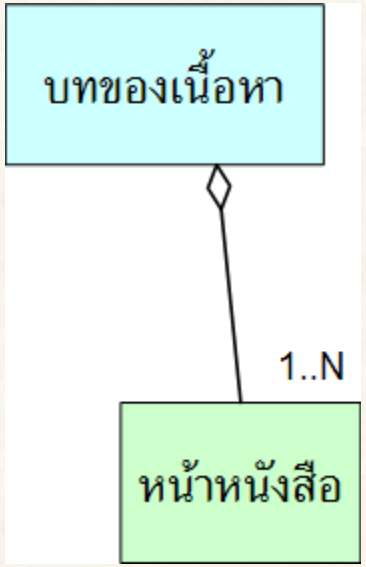
ตัวอย่าง 6 Aggregation ของคลาส หนังสือ (1)

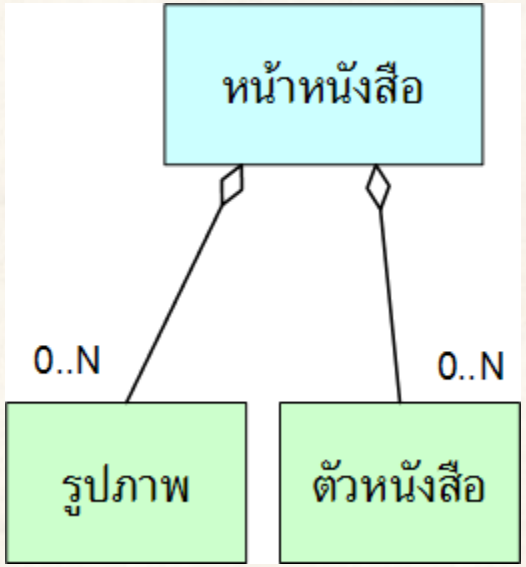


ให้นักศึกษาอธิบาย cardinality เป็นประโยคคำพูด

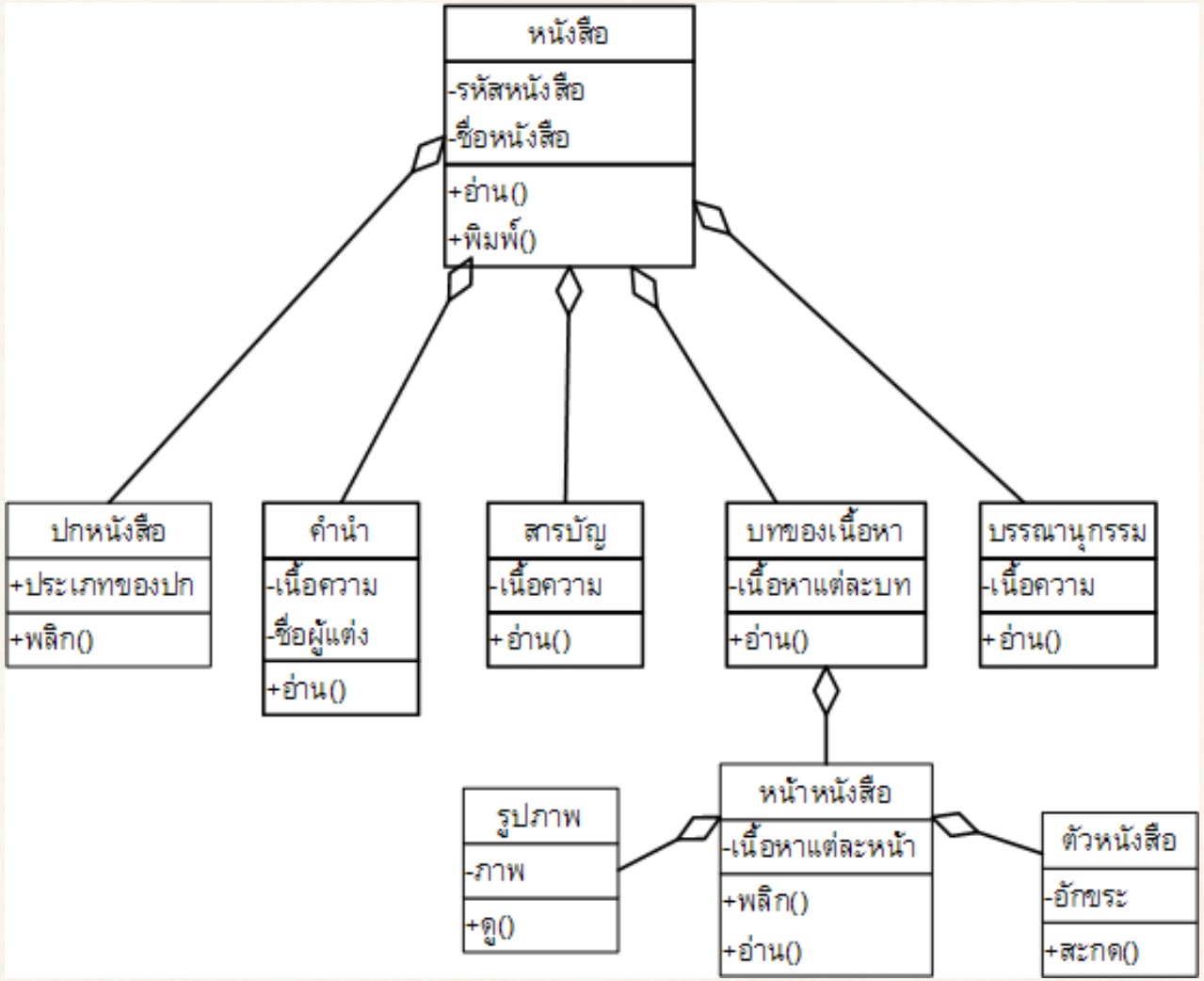
ตัวอย่าง Aggregation ของคลาส หนังสือ (2)







เพิ่ม Attribute และ Method ให้กับ Class หนังสือ



การบ้าน

References

- กิตติพงษ์ กลมกล่อม, "พื้นฐานการวิเคราะห์และออกแบบระบบเชิงวัตถุด้วย UML", สำนักพิมพ์ เคทีพี, 2552.
- พนิดา พานิชกุล, "การพัฒนาระบบเชิงวัตถุด้วย UML", สำนักพิมพ์ เคทีพี, 2552.
- พนิดา พานิชกุล, "Object-Oriented ฉบับพื้นฐาน" , สำนักพิมพ์ เคทีพี, 2548.