**บทที่ 5**

**แผนภาพคลาส (Class Diagram )**

**จุดประสงค์**

1. เพื่อให้นิสิตสามารถค้นหาคลาสที่มีอยู่ในระบบ
2. เพื่อให้นิสิตสามารถใช้แผนภาพคลาสนำเสนอคลาสที่มีอยู่ในระบบ
3. เพื่อให้นิสิตสามารถแสดงความสัมพันธ์ของคลาสได้

**Relationship**

Class diagram จัดเป็น Structural Model (Static Model) คือ มุมมองของระบบที่เน้นโครงสร้างของวัตถุ รวมทั้งคลาสของวัตถุ (Class) ความสัมพันธ์ระหว่างคลาส (Relationship) แอททริบิวต์ (Attribute) และโอเปอเรชัน (Operation) ความสัมพันธ์ที่แสดงเป็นความสัมพันธ์เชิงสถิตย์ (คงที่ Static ) ไม่ใช่ความสัมพันธ์ที่เกิดขึ้นเนื่องจากกิจกรรม (Dynamic) จึงไม่สามารถมองเห็นการเปลี่ยนแปลงต่างๆที่เกิดขึ้น แม้จะเกิดเหตุการณ์ใดๆ

**Class Diagram**

**Static relationship**

เจ้าของบัญชีเป็นเจ้าของบัญชีเงินฝาก

**Dynamic relationship**

เจ้าของบัญชีฝากเงินเข้าบัญชีเงินฝาก

เจ้าของบัญชีถอนเงินจากบัญชีเงินฝาก

เจ้าของบัญชีปรับปรุงยอดบัญชีเงินฝาก

Class Diagram เป็น Static relationship

**องค์ประกอบของ Class Diagram**

Class

โครงสร้างของ Class

พฤติกรรมของ Class

ตัวบ่งชี้

Multiplicity

Navigation

ชื่อของ Role

**ความสัมพันธ์**

Association

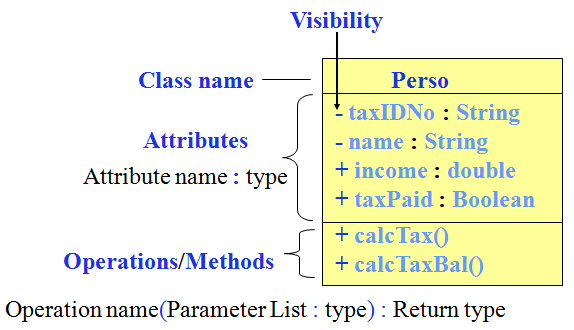
Generalization (Inheritance)

Aggregation

Composition

Dependency

**Class ใน Class Diagram**



**UML Syntax for Class Name**

ชื่อ Class ไม่มีช่องว่าง และขึ้นต้นด้วยตัวพิมพ์ใหญ่ เช่น Bank

ถ้ามีมากกว่า 1 คำตัวแรกของแต่ละคำเป็นตัวพิมพ์ใหญ่ เช่น BankAccount

**UML Syntax for Attributes**

Visibility Name : Type [ = ค่าเริ่มต้น ]

Visibility ประกอบด้วย

+ (public) , - (private), # (protected)

Name คือ ชื่อAttribute ไม่มีช่องว่าง ขึ้นต้นด้วยตัวพิมพ์เล็ก

ตัวแรกของคำถัดไปเป็นตัวพิมพ์ใหญ่เช่น userID

Type คือ ชนิดข้อมูล (ขึ้นกับแต่ละภาษา)

Primitive Type

ข้อมูลพื้นฐาน เช่น integer, boolean Class Type

Attribute ของ Class หนึ่งอาจมีชนิดเป็น Class อื่นได้

UML Syntax for Operations

Visibility Name**(**Parameter List **:** Type**)** : Return type

Visibility ประกอบด้วย

+ (public) , - (private), # (protected)

Name คือ ชื่อ เหมือนกับชื่อของ Attribute

แต่เป็นคำกริยา

เช่น withdrawMoney

Parameter List คือ รายการตัวแปรหรือ Objects ที่ถูกส่ง

เข้าไปใน Operation (Method) เพื่อ

ใช้ดำเนินการอย่างใดอย่างหนึ่ง

( อาจไม่มีก็ได้ )

Type , Return type คือ ชนิดข้อมูล เหมือน Attribute

( อาจไม่มีก็ได้ )

**Visibility**

private ( - )

มักใช้กับ Attributes มากกว่า Operations เห็นได้ภายในเฉพาะตัว class ไม่สามารถเห็นได้ จากภายนอก การเข้าถึงจากภายนอกจะกระทำผ่าน Operation ที่เตรียมไว้

public ( + )

มักใช้กับ Operations มากกว่า Attributesมองเห็นและเข้าถึงได้โดยตรงจากภายนอก

protected (#)

สงวนไว้สำหรับการทำ Inheritance โดยเฉพาะโดยปกติจะเป็นของ Super class

เมื่อทำ inheritance แล้ว Attributes และ Operations เหล่านี้จะเป็นได้ทั้ง Private หรือ Protect ซึ่งขึ้นอยู่กับภาษาที่ใช้

**ความสัมพันธ์ระหว่าง Class**

ความสัมพันธ์ระหว่าง Class จะแสดงโดยใช้เส้นตรงลากเชื่อมระหว่าง Class ซึ่งแบ่งได้เป็น

* + - Associations
    - Generalization
    - Aggregation

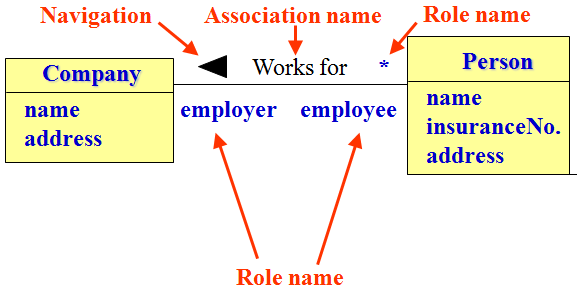
ได้กล่าวถึงมาบ้างแล้วในเรื่อง Abstraction (การให้ Conceptกับ Object จนเกิดเป็น Class และความสัมพันธ์ระหว่าง Class)และบนเส้นความสัมพันธ์ระหว่าง Class อาจมีการระบุ

Association name (ชื่อความสัมพันธ์)

Role name (บทบาท)

Navigation (ทิศทางความสัมพันธ์)

Multiplicity (จะกล่าวถึงในส่วนถัดไป)



**Association name และ Role name**

Association name เป็นคำกริยา

Role name เป็นคำนาม

ปกติ จะเลือกใช้ Association name หรือ Role name เพียงอย่างใดอย่างหนึ่ง และมักจะใช้ Association name เท่านั้น

**Navigation**

หัวลูกศรแสดงถึงทิศทางของความสัมพันธ์ เป็นการแสดงถึงความสามารถของ object ของ Class หนึ่งในการเข้าถึง (ส่ง Message)object ของอีก Class หนึ่งและหมายถึงคือ object ของ Class ทางด้านที่ไม่มีหัวลูกศร สามารถเข้าถึง object ของ Class ทางด้านที่มีหัวลูกศร( แต่ในทางตรงกันข้ามไม่สามารถทำได้ )

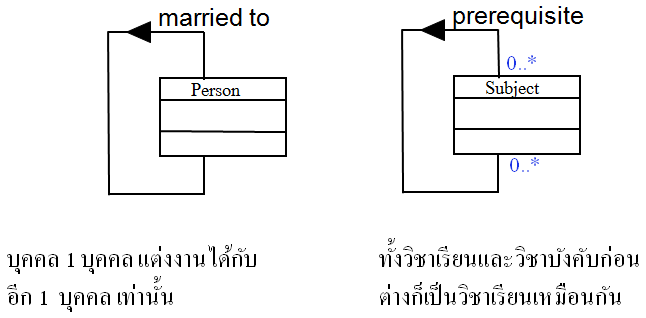
จากตัวอย่าง มีหัวลูกศรทิศทางเดียวจัดเป็น Unidirectional Association

แต่ถ้าทั้งสอง objects สามารถเข้าถึงกันและกันได้ (2 ทิศทาง) ไม่ต้องมีหัวลูกศร ซึ่งเรียกว่า Bidirectional Association

ระดับความสัมพันธ์ระหว่าง Class (Degree of Relationship)

เป็นการมองในเรื่องจำนวนของ Class ที่มีความสัมพันธ์ร่วมกัน ซึ่งแบ่งได้ 3 ระดับคือ

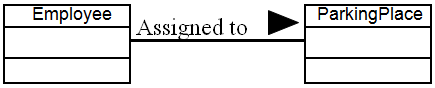
* + - Unary Relationship
    - Binary Relationship
    - Ternary Relationship
    - Unary Relationship
    - ความสัมพันธ์ที่เกิดขึ้นใน Class เดียว (สัมพันธ์กับตัวเอง) จะเกิดในกรณีที่ Attribute ของ Class นั้น สามารถสร้างความสัมพันธ์กับอีก Attribute หนึ่งภายใน Class เดียวกัน



**Binary Relationship**

เป็นความสัมพันธ์ระหว่างสอง Class ซึ่งพบเห็นได้มากที่สุด

**ตัวอย่าง** ความสัมพันธ์ระหว่าง 2 Class

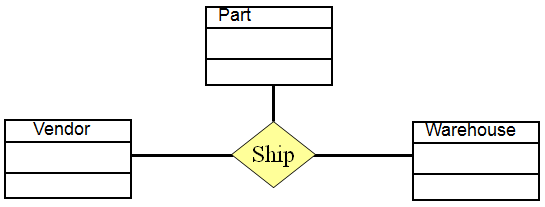


พนักงาน 1 คนถูกกำหนดให้มีที่จอดรถได้ 1 คัน

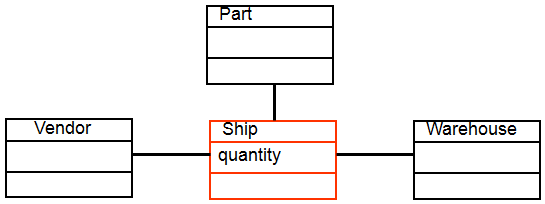
หมายเหตุ กรณี 1 ไม่ระบุได้

**Ternary Relationship**

เป็นความสัมพันธ์ระหว่าง 3 Classes



ซึ่งจะมีการนำชื่อความสัมพันธ์ แปลงให้เป็น Class



จากตัวอย่างก่อนหน้า

Class Part(อะไหล่), Vender (ผู้จำหน่าย), Warehose(คลังสินค้า) มีความสัมพันธ์แบบ Ship(จัดส่ง) และเนื่องจากจำนวนสินค้าที่ส่งให้ Vender และ Warehouse นั้นต่างกัน จึงนำความสัมพันธ์ Ship มาสร้างเป็น Class ตัวกลาง

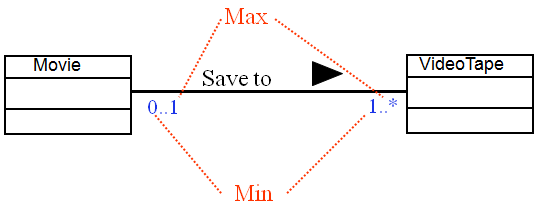
**Multiplicity ในความสัมพันธ์ (Relationship)**

เป็นการแสดงจำนวน Object ของ Class หนึ่ง ซึ่งมีความสัมพันธ์กับ Objects ของอีก Class หนึ่ง บนเส้นความสัมพันธ์

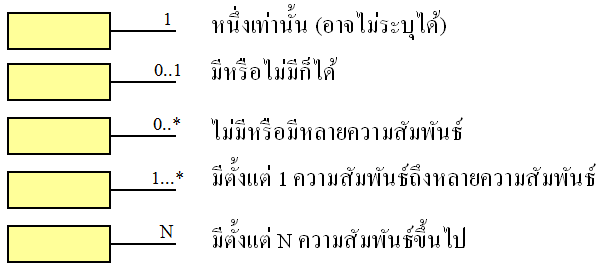
โดยใช้รูปแบบ

Minumum Cardinality **..** Maximum Cardinality

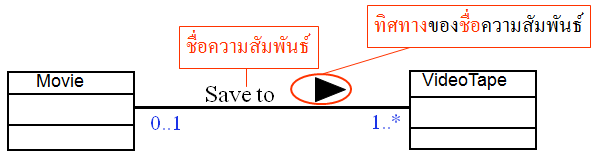
**ตัวอย่าง**



**Association Multiplicity**

****

**วิธีการอ่านค่าความสัมพันธ์**

****

1. อ่านด้านที่มี Min เป็น 1 เพื่อกำหนด Min..Max ของฝั่งตรงข้าม

เช่น Video 1 ม้วน อาจไม่บันทึกหนัง (0) หรือ บันทึกได้ 1 เรื่อง

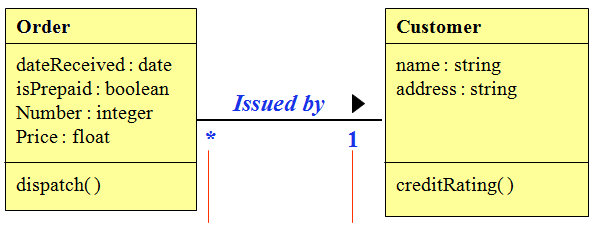
2. เปลี่ยนไปอ่านฝั่งตรงกันข้าม โดยกำหนดให้สมาชิกเริ่มต้นเป็น 1

เช่น หนัง 1 เรื่องบันทึกโดยใช้ Video ต่ำสุด 1 ม้วน และสูงสุด 1 ม้วน

หมายเหตุ ไม่พิจารณาทิศทางของชื่อความสัมพันธ์ เพราะต้องอ่านทั้ง 2 ด้านอยู่แล้ว

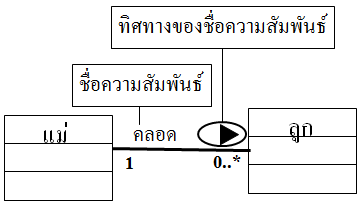
**Associations**

ความสัมพันธ์ทั่วไป (Associate) เป็นความสัมพันธ์ที่ไม่มีการระบุผลของการสืบทอด และการเป็นส่วนหนึ่งของ Class ที่สัมพันธ์ด้วย แต่ Class นั้นจะสัมพันธ์กันในด้านอื่นๆ และเป็นความสัมพันธ์ในระดับเดียวกัน (ไม่มี Class ใด สำคัญกว่า Class ใด )



ลูกค้าอาจมีหลายใบสั่งซื้อ ใบสั่งซื้อเกิดจากลูกค้าคนเดียว

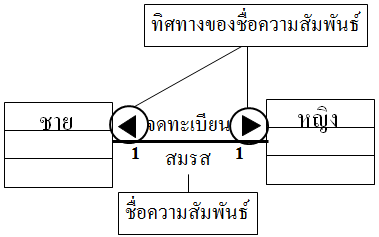
**ตัวอย่าง** ความสัมพันธ์ Association



ความสัมพันธ์ทิศทางเดียว

แม่สามารถคลอดลูกได้

แต่ลูกไม่สามารถ



ความสัมพันธ์แบบสองทิศทาง

ชาย 1 คน จดทะเบียนสมรสกับหญิงได้ 1 คนเท่านั้น และ

หญิง 1 คน จดทะเบียนสมรสกับชายได้ 1 คนเท่านั้น

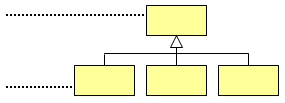
**Generalization(/Specialization)**

เป็นความสัมพันธ์แบบ Superclass และ Subclass ซึ่งก็คือ Inheritance ในแนวคิดหลักเชิงวัตถุGeneralization หาลักษณะร่วมของ Class ต่างๆ เพื่อสร้าง Class ที่เป็นตัวแทนของกลุ่ม Class นั้น (Superclass)

Specialization กำหนดลักษณะเฉพาะเจาะจงในรายละเอียด (Subclass)

Specialization เป็นกระบวนการย้อนกลับของ Generalization Generalization อาจเรียกว่า คือประเภทของ (kind-of relationship )

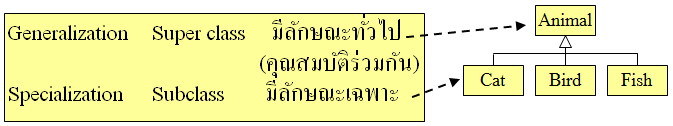
ใน Class Hierachy จะแบ่ง Class เป็นลำดับชั้น

Class ที่มีระดับสูงกว่าเรียกว่า Superclass

Class ที่อยู่ในระดับต่ำกว่าเรียกวา Subclass

**ในการสืบทอด (Inheritance)**

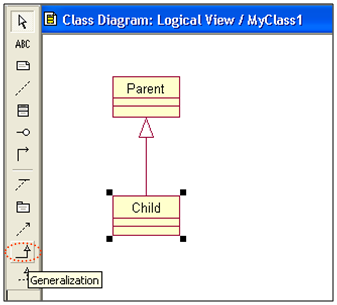
Subclass จะรับคุณสมบัติ (Attributes and Methods) จาก Superclass มาไว้ในตัวเองโดยอัตโนมัติและ Subclass สามารถเพิ่มเติมคุณสมบัติใหม่ (Attributes and Methods) จาก Superclass ซึ่งเรียกว่า Extending หรือแก้ไขคุณสมบัติที่รับมา ซึ่งเรียกว่า Overriding



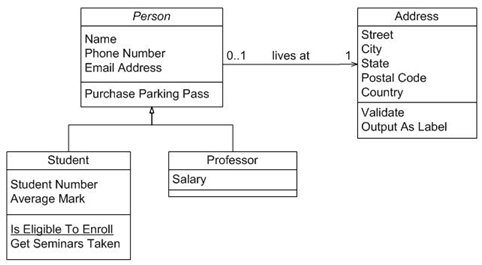
**การสร้างความสัมพันธ์แบบ Generalization**

- เลือก Generalization จาก Diagram Toolbar

- ทำการลากเส้น Generalization จาก Subclass ไปยัง Super class

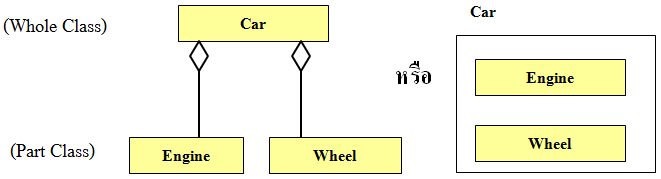


**จงสร้าง Class Diagram ตามตัวอย่างต่อไปนี้ พร้อมทั้งอธิบาย**



**Aggregation Relationship**

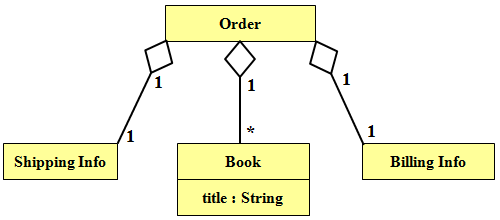
เป็นความสัมพันธ์ Association แบบหนึ่ง (แบบพิเศษ)คือ Object หนึ่ง มี Objects อื่นๆ เป็นส่วนประกอบอาจเรียกว่า  **whole-part** relationship (ทั้งหมด-ส่วนประกอบ) ซึ่งมักจะใช้ค่ำว่า consists of , contains , is part of เช่น Car มี Engine และwheels เป็นส่วนประกอบ



**ลักษณะสำคัญของ Aggregation Relationship**

เมื่อลบ Whole class ทิ้ง Class ที่เป็น Part Classes จะยังคงอยู่ได้ โดยไม่ต้องพึ่งพา Whole class

เช่น Computer กับ Harddisk ถ้าComputer เสีย Harddisk ก็ไม่จำเป็นต้องเสียด้วย ใน UML ความสัมพันธ์จะถูกตั้งสมมุติฐานไว้เป็นแบบ Bidirectional (ไม่มีลูกศร) เสมอ เว้นแต่ว่าจะมีการกำหนดสัญลักษณ์ลูกศรเพื่อจำกัดไว้ และถ้าไม่ระบุ Multiplicity default เป็น many (\*) parts and one whole.



จาก Class Diagram ข้างต้น จงอธิบาย

Object ของ Class ....

และยกตัวอย่าง Aggregation Relationship มา 1 ตัวอย่าง

**Composition Relationship**

เป็นความสัมพันธ์แบบ Aggregation ที่เข้มงวด(Strong)ขึ้นซึ่งจะมีช่วงชีวิตร่วมกัน

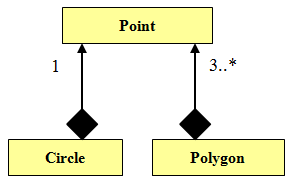
- ส่วนที่เป็น whole จะทำหน้าที่เป็นเจ้าของส่วนที่เป็น part

- ส่วนที่เป็น part อาจเป็นของส่วนที่เป็น whole ได้เพียงหนึ่งเดียว

- Multiplicity ของส่วนที่เป็น whole จะต้องมีค่าเป็น 0 หรือ 1 เท่านั้น

- ช่วงชีวิตของส่วนที่เป็น part จะขึ้นอยู่กับส่วนที่เป็น whole เสมอ นั่นคือจะเกี่ยวข้องการสร้างหรือ

ลบส่วนที่เป็น part เสมอ

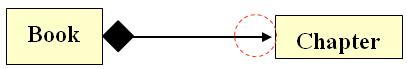


**Composition Relationship**



ใช้สัญลักษณ์

เช่น



กำหนดไว้ทางด้าน Book เนื่องจาก Book จะประกอบไปด้วย Chapter

สัญลักษณ์ลูกศรแสดงถึงความสัมพันธ์เป็นแบบทางเดียว นั่นคือ Chapter ไม่รู้จักส่วนที่เป็น Book

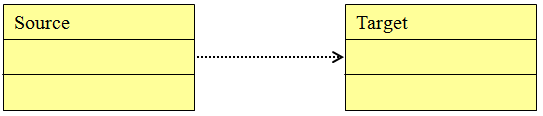
**Aggregation & Composition**

เป็นความสัมพันธ์แบบ Transitive ถ้าวัตถุ A เป็นส่วนหนึ่งของ B และ B เป็นส่วนหนึ่งของ C แล้ว A เป็นส่วนหนึ่งของ C ด้วย เช่น ถ้าที่จับประตูเป็นส่วนหนึ่งของประตู ประตูเป็นส่วนหนึ่งของรถยนต์ แล้ว ที่จับประตูเป็นส่วนหนึ่งของรถยนต์ เป็นความสัมพันธ์แบบ Anti-symmetric วัตถุใดๆ อาจไม่จำเป็นต้องเป็นส่วนประกอบของตัวเองทั้งทางตรงและทางอ้อม เช่น ถ้าประตูเป็นส่วนหนึ่งของรถยนต์ แต่รถยนต์ไม่จัดเป็นส่วนหนึ่งของประตู

**ความสัมพันธ์ระหว่าง Class**

**Dependency**

เป็นความสัมพันธ์ระหว่าง Class แบบขึ้นต่อกัน การเปลี่ยนแปลงต่างๆที่เกิดขึ้นกับ Class หนึ่งจะมีผลกับอีก Class หนึ่งที่ขึ้นต่อกัน



Class Source จะขึ้นกับ Class Target

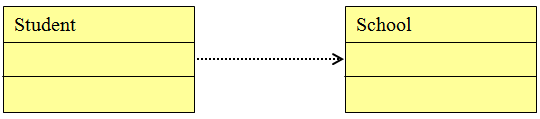
การเปลี่ยนแปลงที่เกิดขึ้นกับ Target จะกระทบกับ Source เสมอ

หมายเหตุ Target อาจเรียกว่า Class ที่ถูกพึ่งพิง (Independent Class)

Source อาจเรียกว่า Class ที่พึ่งพิง (Dependent Class)

เส้นความสัมพันธ์จะคล้ายกับ Association แต่เป็นเส้นประ

**ตัวอย่าง Dependency Relationship**



การเปลี่ยนแปลงของ School จะมีผลกระทบต่อ Student

**หลักการในการสร้าง Class Diagram (Class Diagram Modeling Techniques)**

ในการจำลอง (Model) Class และ Relationship ต่างๆ ใน Class Diagram ใน OOA นั้น สิ่งที่ต้องคำนึงถึงก็คือ Class ทั้งหมดที่ต้องมีอยู่ในระบบหรือใน Real World ( ต้องมีอยู่ครบไม่ขาดหายและไม่มีมากจนเกินความจำเป็น) โดยยังไม่ต้องคำนึง Class ต่าง ๆ ที่ต้องเพิ่มขึ้นเมื่อทำงานในคอมพิวเตอร์ อาทิ เช่น User Interface หรือหน้าจอต่าง ๆ เป็นต้น เพื่อการสร้าง Class Diagram ที่ถูกต้องและสมบูรณ์ใน Analysis Phase เราสามารถสรุปเป็นหลักการเพื่อใช้ในการสร้าง Class Diagram ได้ดังนี้

(จาก กิตติ ภักดีวัฒนะกุล, กิตติพงษ์ กลมกล่อม. **UML วิเคราะห์และออกแบบระบบเชิงวัตถุ**)

1. กำหนดกรอบของ Problem Domain ให้ชัดเจนและยึดถือ Problem Domain นี้เป็นมาตรฐานและบรรทัดฐานในการวิเคราะห์ระบบ (ขอบเขตของ Problem Domain นี้ สามารถเปลี่ยนแปลงได้ตามช่วงเวลาแต่ไม่ควรเปลี่ยนแปลงบ่อยจนเกินความจำเป็น) เขียน Use Case Diagram ของ Problem Domain ที่กำหนดไว้แล้วพิจารณาว่า ในแต่ละ Use Case จะมี Objects อะไรอยู่บ้าง เมื่อทำเช่นนี้จนครบทุก Use Case แล้วจะได้ Object ที่มีอยู่ใน Problem Domain ทั้งหมด
2. พิจารณาหา Objects ที่สามารถจับต้องได้ เห็นได้ สัมผัสได้ซึ่งเรียกว่า Tangible Objects หรือหาตัวแทนของ Tangible Objects ในกรณีที่มี Tangible Objects หลาย ๆ ตัวใน Problem Domain เดียวกันให้ครบทุกตัว
3. พิจารณา Objects ที่ไม่สามารถจับต้องได้ซึ่งเรียกว่า Intangible Objects หรือหาตัวแทนของ Intangible Objects ในกรณีที่มี Intangible Objects หลาย ๆ ตัวใน Problem Domain เดียวกัน ที่มีอยู่หรือที่น่าจะมีอยู่ใน Problem Domain ให้ครบทุกตัว

4. ใช้ Classification Abstraction เพื่อแยกแยะและสร้าง Class

จาก Objects ที่มีอยู่ และพยายามหา Attributes และ Functions ที่มีอยู่ใน Class นั้น ๆ เท่าที่จะหา มาได้ วาด Class ที่ได้ทั้งหมดลงใน Class Diagram

1. หา Aggregation Abstraction

โดยพิจารณา Class ที่ได้จาก Classification Abstraction ว่ามี Class ใดหรือไม่ที่มีความสัมพันธ์ แบบเป็นส่วนหนึ่งหรือประกอบด้วย (whole-part) กับ Class อื่น ๆ

ถ้ามีพยายามหาว่า Aggregation ที่เกิดขึ้นนั้นเป็นแบบ One to One หรือ Many to One และใส่ Cardinality ให้ถูกต้อง

1. ใช้ Generalization มาพิจารณา Class ต่างๆ ใน Class Diagram

หากเกิดมีความสัมพันธ์แบบ Generalization หรือ Specialization เกิดขึ้น ให้เพิ่มเติมลงใน Class Diagram ซึ่งในขั้นตอนนี้อาจมีการสร้าง Class ใหม่ได้

1. ใช้ Association มาพิจารณา Class ต่าง ๆ ใน Class Diagram

เพิ่มเติมสัญลักษณ์ของ Association ลงใน Class Diagram และพิจารณาประเภทของความสัมพันธ์ และ Cardinality (Multiplicity) ให้ถูกต้อง

8. พิจารณา Class Diagram ที่สร้างมาทั้งหมดว่า ทุก Class และทุกกลุ่มของ Class มีความสัมพันธ์ (Relationship) แบบใดแบบหนึ่งกับ Class หรือกลุ่มของ Class อื่นหรือไม่

หากว่ามี Class ใด Class หนึ่งหรือกลุ่มหนึ่ง ยังไม่มี Relationship ใดกับ Class อื่นเลย อาจเกิดเนื่องจาก Class นั้นเป็น Class ที่เกินความจำเป็นจริง ๆ ไม่ต้องมีในระบบก็ได้ หรืออาจเกิดเนื่องจาก ยังขาด Class อื่นๆ ที่จำเป็นต้องมีและต้องมี Relationship กับ Class ดังกล่าว สิ่งที่ต้องทำหากเกิดกรณีนี้ขึ้น คือเริ่มต้นใหม่ตั้งแต่ขั้นตอนที่ 1 โดยพิจารณาหา Objects ที่น่าจะขาดหายหรือเกินขึ้นมา แล้วทำต่อมาจนจบหรือจนกว่าจะได้ Class Diagram ที่สมบูรณ์

**หมายเหตุ**  ข้อ 4 ถึง 7 สามารถทำสลับขั้นตอนกันได้

**ตัวอย่าง**

จงสร้าง Class Diagram จาก Problem Domain ที่กำหนดให้ต่อไปนี้

ในคณะวิทยาศาสตร์ของสถาบันการศึกษาแห่งหนึ่งมีบุคลากรหลายประเภทด้วยกัน ได้แก่ อาจารย์ นักศึกษา และเจ้าหน้าที่ โดยที่อาจารย์แต่ละท่านมีหน้าที่ในการสอนวิชาใดวิชาหนึ่งหรือมากกว่า 1 วิชาก็ได้ และนักศึกษาก็มีหน้าที่ในการศึกษาวิชาใดวิชาหนึ่ง หรือมากกว่า 1 วิชาก็ได้ในเวลาเดียวกันเจ้าหน้าที่ของภาควิชา คือเจ้าหน้าที่ประจำห้องทดลองต่างๆ โดยกำหนดว่าใน 1 ห้องทดลองจะต้องมีเจ้าหน้าที่ 1 คนเสมอ

**หา Use Case จาก Problem Domain**

use case ของระบบคือ

การเรียนการสอน

ประกอบด้วย Actor คือ นักเรียน อาจารย์

การใช้ห้องทดลอง

ประกอบด้วย Actor คือ นักเรียน อาจารย์

การดูแลห้องทดลอง

ประกอบด้วย Actor คือ เจ้าหน้าที่

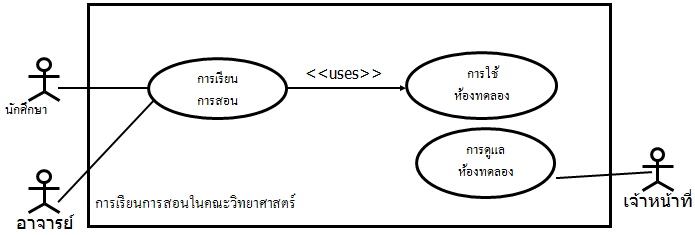
**Actors ที่มีจาก use case คือ**

นักเรียน

อาจารย์

เจ้าหน้าที่

**เขียน Use Case Diagram**



จาก Use Case ที่ได้ เราสามารถหา Objects/Classes ที่มีอยู่ในระบบ

**U**se case การเรียนการสอน

นักเรียน อาจารย์ // Object/Class Actor

ห้องเรียน วิชาเรียน ชั่วโมงเรียน // Object/Class อื่นๆ

**U**se case การใช้ห้องทดลอง

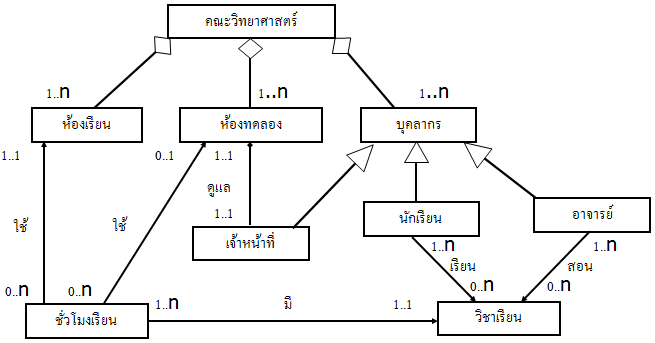
นักเรียน อาจารย์ // Object/Class Actor

ห้องทดลอง // Object/Class อื่นๆ **U**se case การดูแลห้องทดลอง

เจ้าหน้าที่ // Object/Class Actor

ห้องทดลอง // Object/Class อื่นๆ

**เขียน Class Diagram เบื้องต้น** (ไม่มี Attributes, Operations) **จาก Scenario**



ได้เพิ่ม Class บาง Class ที่จำเป็นได้แก่

คณะวิทยาศาสตร์ เกิดจาก Aggregation Abstraction (ของห้องเรียน ห้องทดลองและบุคลากร)

บุคลากร เกิดจาก Generalization (บุคลากร จำแนกเป็นเจ้าหน้าที่ นักเรียนและอาจารย์)

จาก Class Diagram เบื้องต้น

ประกอบด้วย Class ที่เป็น Tangible ได้แก่บุคลากร (จำแนกเป็นเจ้าหน้าที่ นักเรียนและอาจารย์) ห้องเรียน และห้องทดลอง

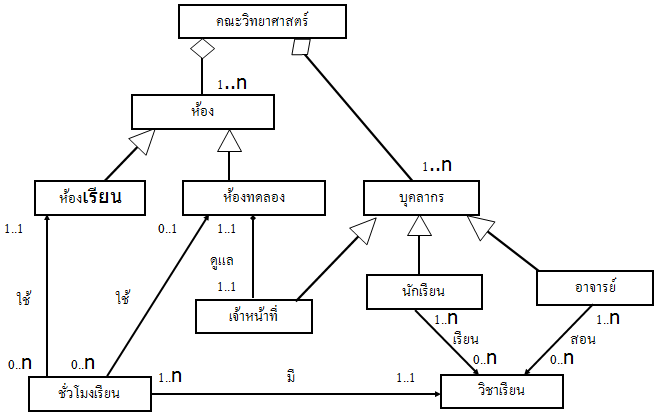
ส่วน Class ที่เป็น Intangible ได้แก่ คณะวิทยาศาสตร์ วิชาเรียน และชั่วโมงเรียน

นอกจากนี้จาก Problem Domain ที่กำหนดให้ เราจะพบว่า Class Diagram จะมีความสัมพันธ์กันอันเกิดจากการใช้ Abstraction แบบต่าง ๆ และมีการสร้าง Class เพิ่มขึ้นด้วย เพื่อเพิ่มความสมบูรณ์ให้กับ Class Diagram ได้แก่ Class ชั่วโมงเรียน เพื่อช่วยในการโยงความสัมพันธ์ระหว่างห้องเรียนกับวิชาเรียน

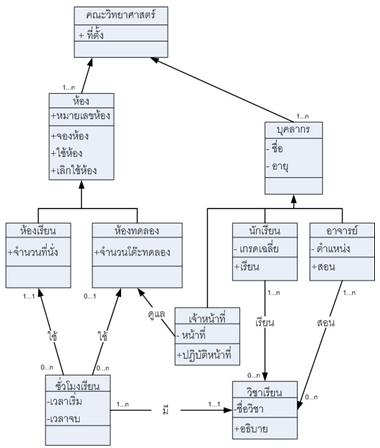
การปรับเปลี่ยน Class Diagram ให้สมบูรณ์ขึ้น (คำว่า สมบูรณ์ขึ้น คือทำให้ Class Diagram เข้าใจได้ง่ายและครบถ้วนขึ้น อาจมีการเพิ่ม Class หรือความสัมพันธ์ขึ้น แต่ต้องไม่ทำให้เรื่องราวที่มีเปลี่ยนไป)

โดยการเพิ่ม Class ห้อง ซึ่งเป็น Generalized Class หรือ Super class ของห้องเรียนกับห้องทดลองนั่นเอง

เมื่อปรับเปลี่ยน Class Diagram ให้สมบูรณ์มากยิ่งขึ้น



สามารถปรับเปลี่ยน Class Diagram อีกครั้ง เพื่อให้ได้ Class Diagram ที่สมบูรณ์ขึ้น โดยเพิ่มเติม Attributes และ Operations ลงไป



อย่างไรก็ตาม Class Diagram ที่ได้ จะถูกทำ Refinement ในขั้นตอนของ Object-Oriented Design (OOD) เพื่อเพิ่มความสมบูรณ์ จนกระทั่งสามารถนำมาใช้เป็นต้นแบบในการพัฒนาระบบงานในคอมพิวเตอร์ได้ในที่สุด

**แบบฝึกหัด**

1. จงสร้าง Class Diagram ของระบบลงทะเบียน Online ผ่าน Internet ของมหาวิทยาลัย
2. จงสร้าง Class Diagram ถ้าใน 1 ปีประกอบด้วยเดือน 12 เดือน ได้แก่ มกราคม กุมภาพันธ์ - ธันวาคม ในแต่ละเดือนจะประกอบไปด้วย วันจำนวนหนึ่ง ซึ่งวันที่มีในเดือนนั้น จำแนกได้เป็น วันธรรมดา (Working Days) วันหยุดปลายสัปดาห์ (Weekends) และวันหยุดนักขัตฤกษ์ (Holidays) ซึ่งวันหยุดนักขัตฤกษ์สามารถจำแนกได้เป็น

1) วันหยุดสากล เช่น วันแรงงาน วันปีใหม่

2) วันหยุดทางศาสนา เช่น วันมาฆบูชา

3) วันหยุดที่ประกาศโดยรัฐ เช่น วันจักรี