การพัฒนาซอฟต์แวร์เชิงวัตถุ
บทที่ 7
Class Modeling

อ.สกรณ์ บุษบง
สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
มหาวิทยาลัยราชภัฏบุรีรัมย์

Class Modeling

- องค์ประกอบที่สำคัญที่สุดใน Object Oriented Analysis and Design คือ Class ซึ่งทำหน้าที่เป็นตัวแทนของกลุ่มของวัตถุ (Object) ที่อยู่ใน Problem ที่เราสนใจ
- Class Diagram ใช้ในการอธิบายโครงสร้างและความสัมพันธ์ต่างๆ ของ Class ได้

CLASS

Class

- Class เป็นส่วนประกอบที่สำคัญที่สุดใน Class Diagram
- Class คือสิ่งที่อธิบายแนวคิดกลุ่มของวัตถุ (Object) ที่มี Attribute, Method และความหมายที่เหมือนๆ กัน
- ส่วนประกอบของ Class ประกอบด้วย
  - Attribute
  - Method
  - Constructor

Class : Attribute

- Attribute หมายถึง คุณสมบัติของ Class ซึ่ง Object ทุกตัวของ Class ต้องมีคุณสมบัติตามที่กำหนดไว้ใน Class
  - Class หนึ่งๆ จะมี Attribute จำนวนเท่าไรก็ได้
  - Attribute ทุกตัวต้องมีชื่อ และชื่อของ Attribute ของ Class เดียวกันต้องไม่ซ้ำกัน

Class : Attribute : Visibility

- Visibility : Attribute ทุกตัว Class จะต้องถูกกำกับด้วย Visibility เสมอ
  - Information Hiding เป็นการกำหนดการปกปิดแบ่งได้ 3 ระดับ
    - Private(-)
    - Protected(#)
    - Public(+)
    - Default(~)

Class : Attribute : Visibility

- **Private (-)** : เป็นการปกปิดไม่ถูกเปิดเผยแก่ภายนอกและไม่สามารถเข้าถึงได้โดยตรงจากภายนอก แต่สามารถเข้าถึงได้จากภายในตัว Class เอง
- **Protected (#)** : จะไม่ถูกเปิดเผยแก่ภายนอก และไม่สามารถเข้าถึงได้โดยตรงจากภายนอก Protected จะถูกถ่ายทอดไปให้ Subclass ซึ่งต่างกับ Private จะไม่สามารถเข้าถึงได้โดยตรงจาก Subclass
- **Public (+)** : จะถูกเปิดเผยและเข้าถึงได้โดยตรงจากภายนอก และสามารถถ่ายทอดสู่ Subclass ได้
- Default(~) : จะถูกมองเห็นจาก Package เดียวกันเท่านั้น

Class : Attribute : Type

- Type แบ่งออกเป็น 2 ประเภท คือ Primitive Type และ Class
  - **Primitive** หมายถึง คุณสมบัติของ Attribute ที่กำหนดรูปลักษณ์ของค่า Attribute ขอบเขตของค่าที่เป็นไปได้และการดำเนินการที่สามารถกระทำได้
  - **Class** หมายถึง Attribute ของ Class หนึ่ง อาจมีประเภทเป็นคลาสอื่นก็ได้

Method

- Method หมายถึง บริการที่ Object ของคลาสต้องมี เพื่อให้สิ่งแวดล้อม หรือ Object อื่นๆเรียกใช้บริการได้
- Method อาจเป็นการกระทำบางอย่างที่มีผลกระทบโดยตรงต่อความหมายหรือ Attribute ของ Object ก็ได้
- ในหนึ่ง Class มี Method เท่าไรก็ได้
- ทุกๆ Method ต้องมี Visibility เสมอ
- ทุกๆ Method ต้องมีชื่อ และชื่ออาจซ้ำกันได้ต้องพิจารณากฎการตั้งชื่อ Method

Method : กฎการตั้งชื่อ Method

1. ชื่อของ Method ใน Class เดียวกัน ไม่ควรซ้ำกัน
2. การซ้ำกันของชื่อ Method ใน Class เดียวกัน จะมีได้ก็ต่อมื่อ
    1. Method ที่มีชื่อซ้ำกัน มีจำนวนของ Parameter ต่างกัน
    2. Method ที่มีชื่อซ้ำกัน มีจำนวนของ Parameter เท่ากัน แต่ Type ของ Parameter ต่างกัน

Overloading

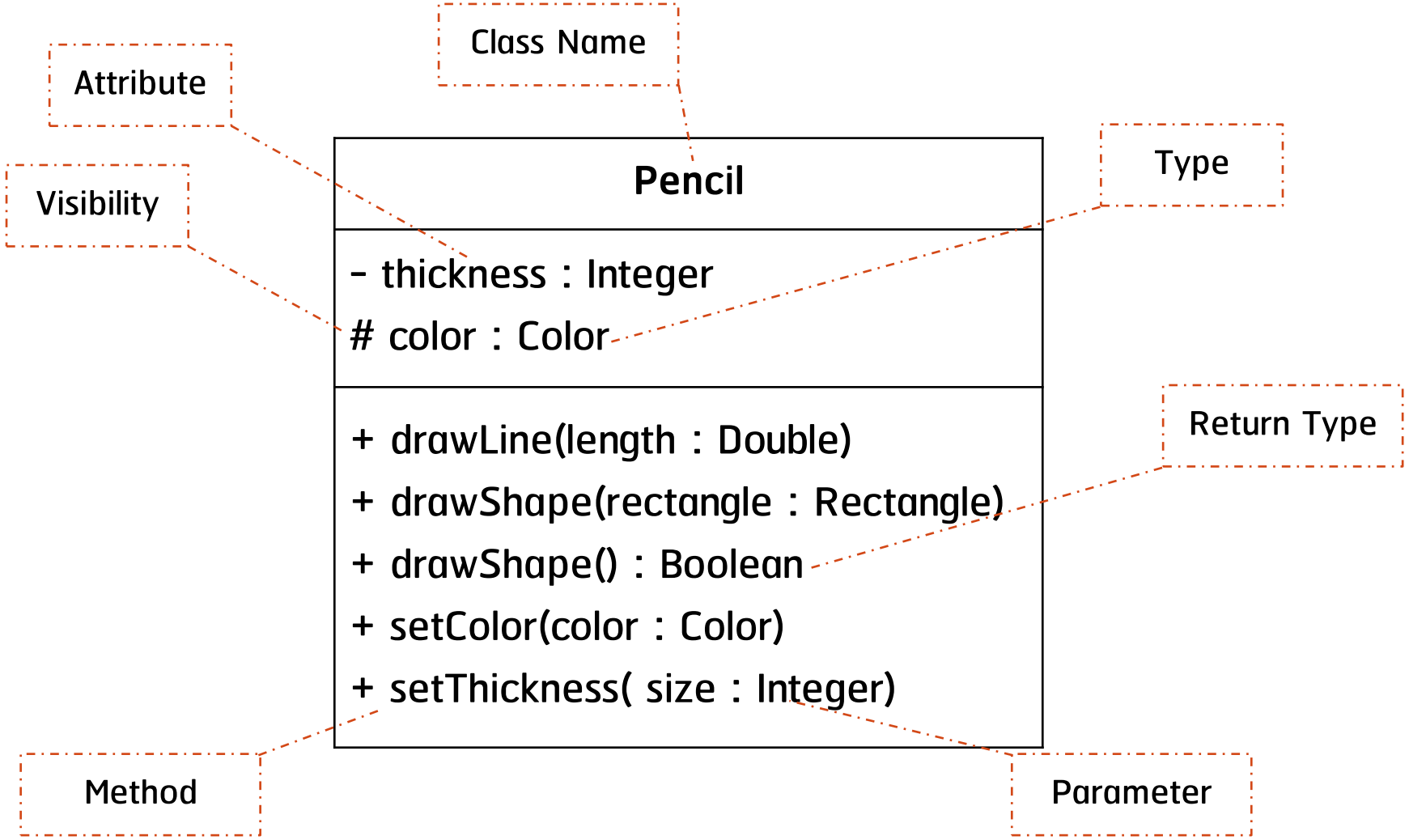
Method : Parameter

- Parameter คือ ตัวแปร หรือ Object ที่ถูกส่งเข้ามาที่ Method ซึ่งจะถูกใช้งานเพื่อการดำเนินการบางอย่าง
- Method อาจไม่มี Parameter เลยก็ได้
- Method มี Parameter เท่าไรก็ได้

Method : Return Type

- Return Type หมายถึง Type ของผลลัพธ์จากการดำเนินการของ Method ที่จะถูกส่งออกมา
- Method มี Return Type ได้สูงสุดค่าเดียว หรืออาจไม่มีเลยก็ได้

ตัวอย่าง Class Diagram



Constructor

- Constructor เป็น Method ที่ทำงานทันทีหลังจาก Object ถูกสร้างขึ้นมาจาก Class
- Constructor ต้องเป็น Method ที่มีชื่อเดียวกันกับ Class และ ไม่มี Return Type
- Constructor ของ Class สามารถมีมากกว่า 1 ตัวได้ แต่ใช้แนวคิดเดียวกันกับ Method ที่มีชื่อซ้ำกัน

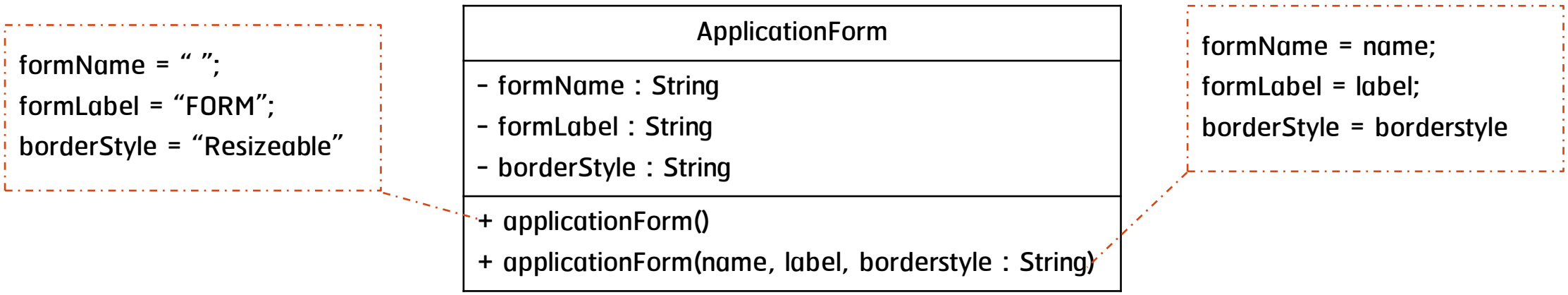
Constructor

- Constructor จะถูกเรียกใช้เมื่อสร้าง Object จาก Class จะมีรูปแบบการใช้งานดังนี้
  - การกำหนดค่าเริ่มต้นให้กับ Attribute ของ Object
  - การระบุ Class ที่แท้จริงของ Object

Constructor :

การกำหนดค่าเริ่มต้นให้กับ Attribute ของ Object

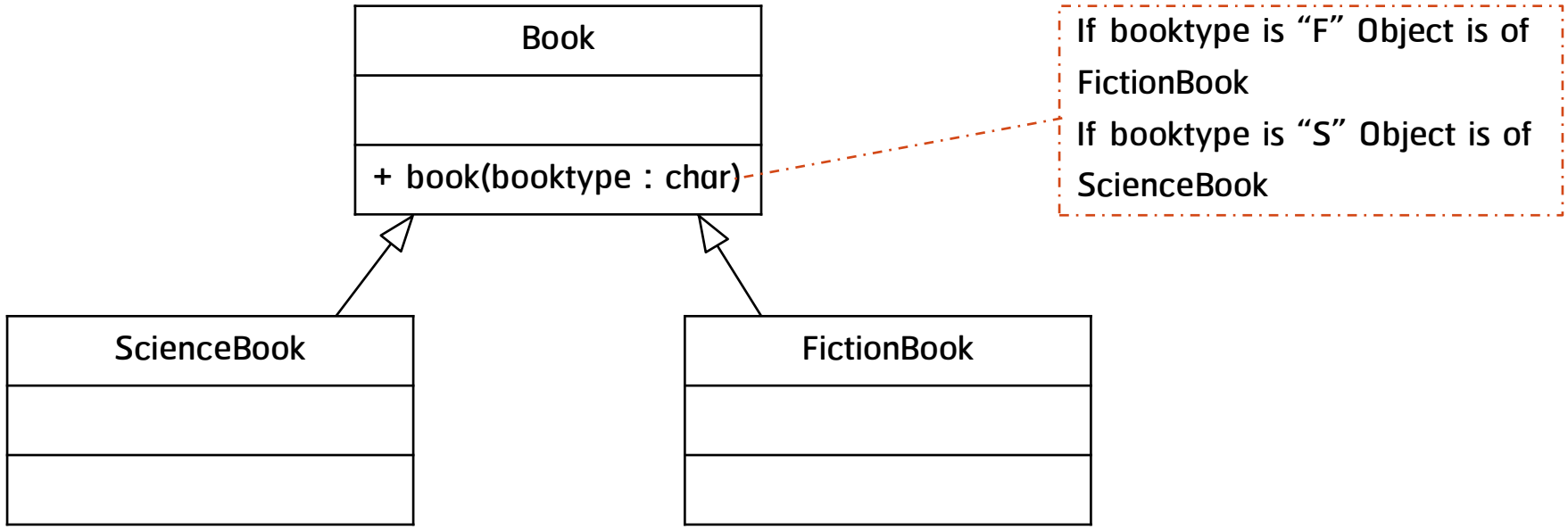
- Constructor มักถูกใช้ในการกำหนดค่าเริ่มต้น (Default) ให้กับ Object ของ Class
- Constructor จะส่งค่าของแต่ละ Attribute ของ Object ในรูปแบบของ Parameter ของ Constructor



Constructor :

การระบุ Class ที่แท้จริงของ Object

- ถ้า Class ทำ Inheritance จะสามารถใช้ Constructor ของ Superclass กำหนด Object ว่าเป็นของ Subclass ใด โดยกำหนดทาง Parameter ของ Constructor ของ Superclass



INTERFACE

Interface

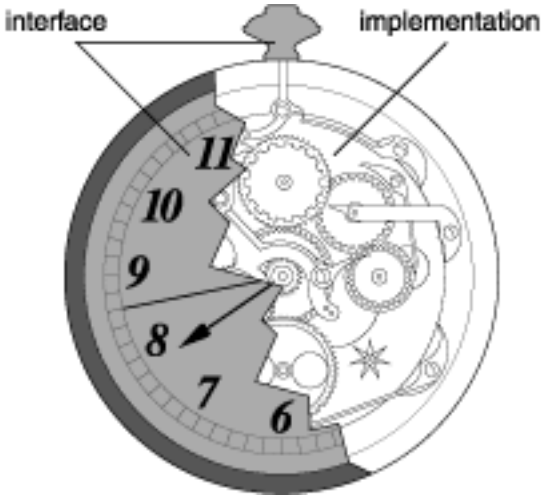
- Interface คือ ที่รวมของ Public Method Specification แต่ไม่ใช่ที่รวมของ Public Method Implementation
- Interface จะระบุเพียงว่าตนเองทำงานอะไร แต่ไม่ระบุว่าทำงานอย่างไร
- การใช้งาน Interface จะต้องใช้ Class มา Inherit Interface แล้วสร้าง Method ของ Interface เพื่อกำหนดการทำงาน

Interface

- เมื่อ Class ทำการ Inherit Interface แล้ว Class นั้นต้องทำหน้าที่ในการ Implement ตัว Method ของ Interface นั้นเสมอ
- Class ที่ต่างกัน ถ้า Inherit Interface ตัวเดียวกัน ใน Method เดียวกัน อาจทำงานแตกต่างกันได้
- แนวคิดนี้ยึดตามหลักการของ Polymorphism

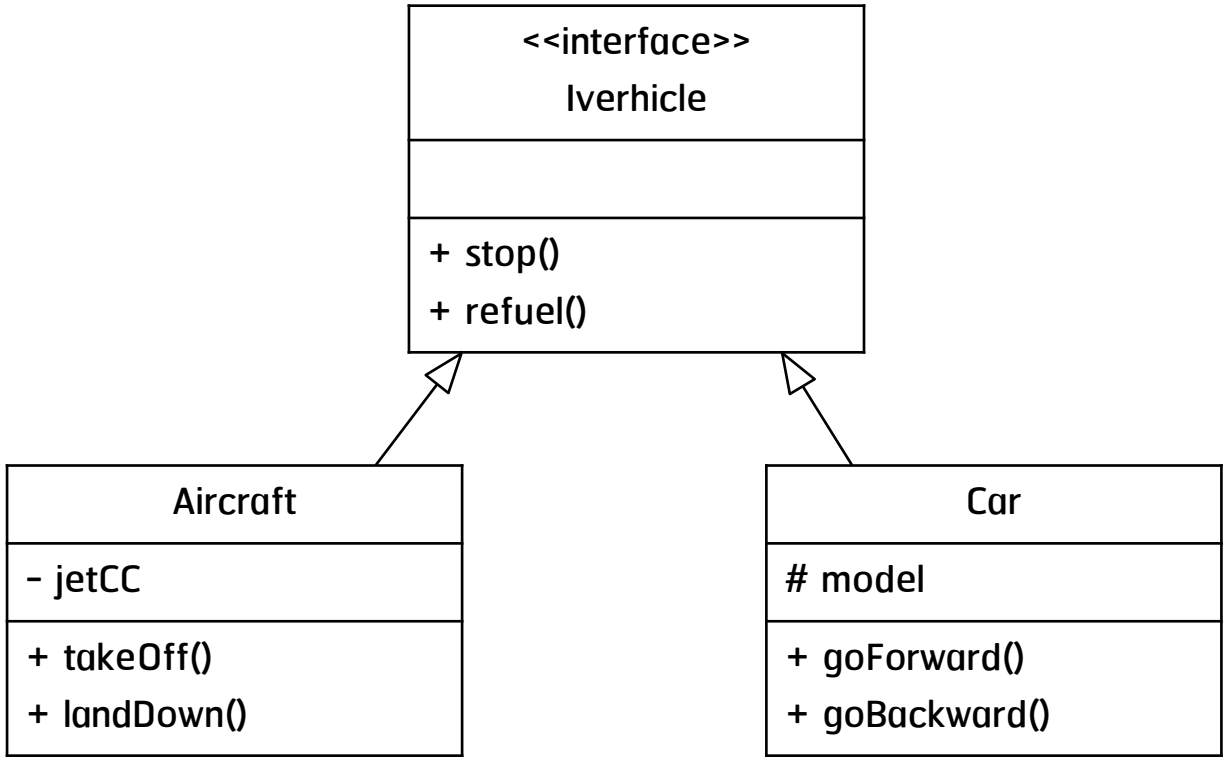
Interface

- ในทางปฏิบัติ Interface มีประโยชน์อย่างยิ่งในแง่ของการวิเคราะห์และออกแบบระบบ
- เนื่องจาก บางครั้งผู้ออกแบบและโปรแกรมเมอร์เป็นคนละกลุ่มกัน ผู้ออกแบบสามารถสร้าง Interface เพื่อกำหนด Method ต่างๆ ที่ต้องมี โดยไม่จำเป็นต้องระบุว่าแต่ละ Method ต้อง Implement อย่างไร



Interface

- การตั้งชื่อ Interface ให้ใส่ตัวอักษร I (ตัวพิมพ์ใหญ่) ไว้ข้างหน้าทุกครั้ง
- เช่น ชื่อ Interface ที่เป็นตัวแทนของ ยานพาหนะ (Vehicle) ควรตั้งชื่อเป็น IVechicle



ABSTRACT CLASS

Abstract Class

- Class โดยปกติเปรียบเสมือนพิมพ์เขียวในการสร้าง Object
- Class อีกประเภทที่ไม่สามารถสร้าง Object ได้ แต่สามารถถูก Inherit ได้โดย Class อื่นๆ
- เรียกคลาสเหล่านี้ว่า Abstract Class

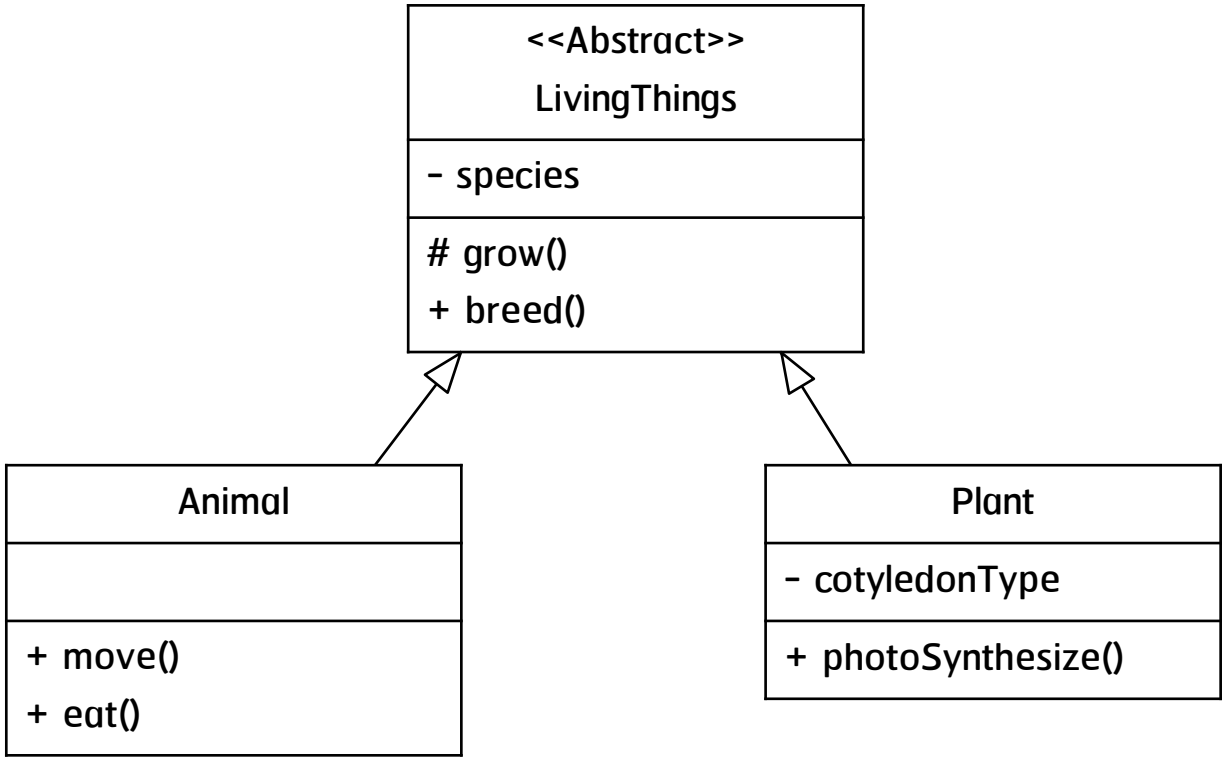
Abstract Class

- ในโลกความจริง Abstract Class มีมากมาย เช่น “สิ่งมีชีวิต (Living Things)”
- “สิ่งมีชีวิต (Living Things)” ถือว่าเป็น Abstract Class โดยเราไม่สามารถหา Object ของ Class ของสิ่งมีชีวิตบนโลกนี้ได้เลย เนื่องจากในโลกเรามีสิ่งมีชีวิต 2 ประเภท คือ พืช (Plant) และสัตว์ (Animal) ซึ่งจะมีแต่ Object ของพืช หรือสัตว์เท่านั้น

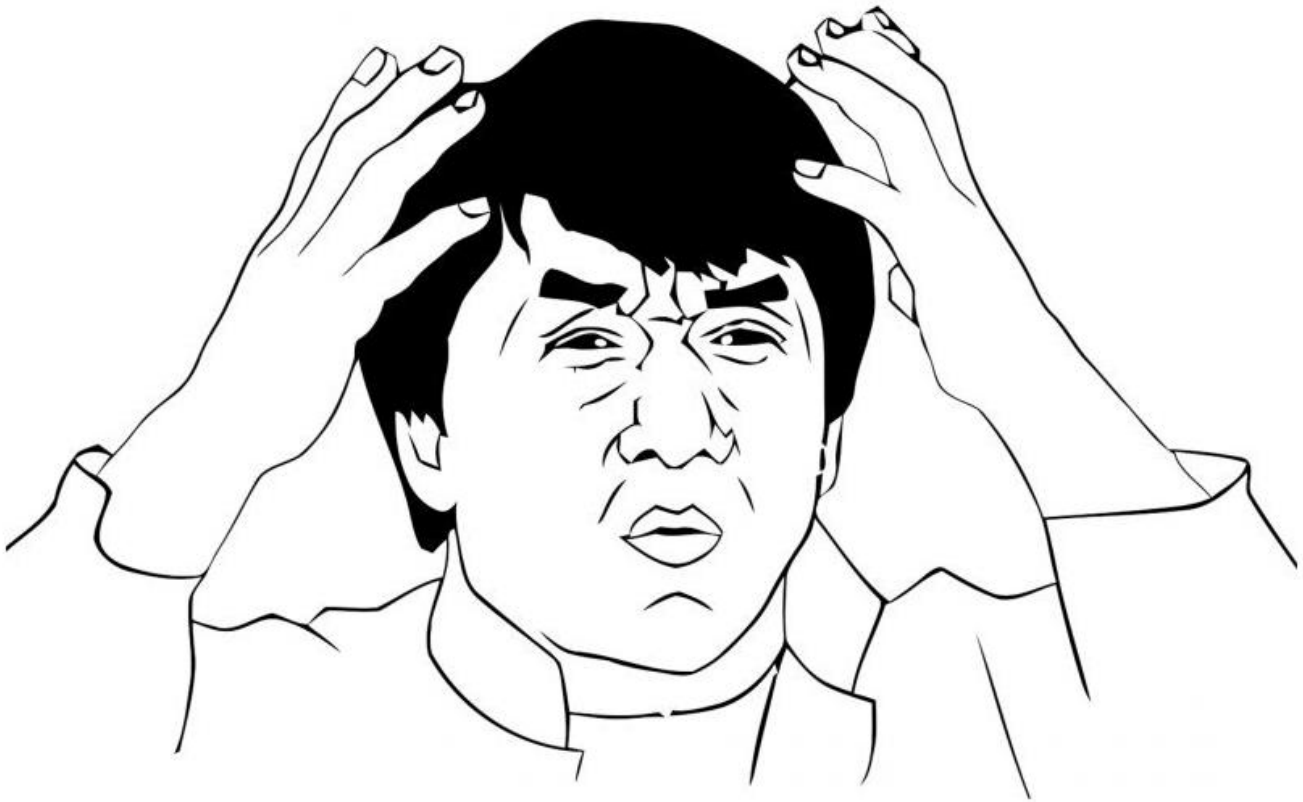
Abstract Class

- Abstract Class และ Interface มีความคล้ายกันมาก แต่มีข้อแตกต่างกันดังนี้
  - Abstract Class สามารถมี Attributes ได้ แต่ Interface ไม่สามารถมี Attributes ได้
  - Abstract Class สามารถมี Implementation ของ Method ได้ แต่ Interface ไม่มี
  - Methods และ Attribute ของ Abstract Class สามารถมี Visibility แบบใดก็ได้ แต่ Method ของ Interface ต้องเป็น Public เท่านั้น

Abstract Class



Abstract Class



Relationship ใน Class Diagram

Relationship ใน Class Diagram

- ความสัมพันธ์หรือ Relation ใน Class Diagram เป็น Relationship แบ่งออกเป็น
  - Association
  - Aggregation และ Composition

Association

- การพัฒนาซอฟต์แวร์เชิงวัตถุ หลีกเลี่ยงความสัมพันธ์รัหว่าง Class ไม่ได้เลย
- ดังนั้นการออกแบบคลาสจะต้องค้นหาความสัมพันธ์ ซึ่งเราเรียกว่า “Association Abstraction”

Role และ Association Name

- เมื่อ Class สอง Class มีความสัมพันธ์แบบ Association แล้ว จะต้องสามารถอธิบายได้ ด้วยชื่อของ Association (Association Name) เช่น
  - ความสัมพันธ์ของคนในที่ทำงาน คนและที่ทำงานมีความสัมพันธ์ ชื่อว่า “ทำงานที่ (work at)”
- Association Name จะต้องมีการกำหนดทิศทาง (Association Direction) เพื่อป้องกันความสับสน
- ใน Association Abstraction นั้น Class ทุก Class จะแสดงบทบาท (Role) เสมอ เช่น
  - ความสัมพันธ์ “คนทำงานในที่ทำงาน”
    - Class คน แสดงบทบามเป็น “ผู้ถูกจ้าง (Employee)”
    - Class ที่ทำงาน แสดงบทบามเป็น “ผู้จ้าง (Employer)”

Multiplicity

- การพิจารณา Association ระหว่าง Class ต้องคำนึงถึง ค่าที่เป็นไปได้ของจำนวนสมาชิกใน Class ที่มี Association ซึ่งจะเรียกค่าจำนวนสมาชิกที่เป็นไปได้ว่า “Multiplicity”
  - เรียกจำนวนน้อยที่สุดของสมาชิก Class ที่มีส่วนร่วมใน Association ว่า “Minimum Multiplicity (Min-card)”
  - เรียกจำนวนมากที่สุดของสมาชิก Class ที่มีส่วนร่วมใน Association ว่า “Maximum Multiplicity (Max-card)”

Multiplicity

- สมมติว่าจากการสอบถามเพื่อหาความสัมพันธ์ของ Class ในระบบห้องสมุดประชาชน พบว่า
  - 1. สมาชิกของห้องสมุดประชาชน (Member) มีบัตรสมาชิก (Member Card) ได้เพียงหนึ่งใบ
  - 2. ในการยืมหนังสือ (Book) แต่ละครั้งสมาชิกสามารถยืมได้อย่างน้อย 1 เล่ม อย่างมาก 5 เล่ม
  - 3. หนังสือบางเล่ม (โดยเฉพาะหนังสือเกี่ยวกับคอมพิวเตอร์) จะมีแผ่น CD เพื่อศึกษาคู่กับหนังสือเล่มนั้นๆ โดยแผ่น CD จะถูกแยกเก็บไว้ที่บรรณารักษ์ ซึ่งสมาชิกสามารถขอยืมหนังสือเพื่อนำไปประกอบการค้นคว้าได้โดยแผ่น CD ที่มากับหนังสือจะมีจำนวนเท่าใดก็ได้
  - 4. สมาชิกห้องสมุด สามารถแจ้งความจำนงขอให้ห้องสมุดจัดหาสื่อวิชาการต่างๆ (Media) ที่ต้องการได้ เช่น เทป แผ่นเสียง สไลด์ เป็นต้น โดยสมาชิกหนึ่งคนสามารถแจ้งความจำนงได้มากกว่าหนึ่งรายการ และเป็นไปได้ว่าสื่อหนึ่งรายการอาจถูกแจ้งความจำนงโดยสมาชิกมากกว่าหนึ่งคน

Multiplicity

- จาก Requirements ทั้ง 4 ข้อ สรุป Association และ Multiplicity ที่มีได้ดังนี้
- 1. Association ระหว่าง Member และ MemberCard

Association Name	Association Direction	Member			MemberCard		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
มี (has)	Member → MemberCard	เป็นเจ้าของ (owner)	1	1	ถูกเป็นเจ้าของ (owned item)	1	1

Multiplicity

- 2. Association ระหว่าง Member และ Book

Association Name	Association Direction	Member			Book		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
ยืม (borrows)	Member → Book	เป็นผู้ยืม (borrower)	1	1	ถูกยืม (borrowed item)	1	5

Multiplicity

- 3. Association ระหว่าง Book และ CompactDisk

Association Name	Association Direction	CompaactDisk			Book		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
ใช้ร่วมกับ (used with)	CompaactDisk → Book	ใช่ร่วม (used with)	1	1	ใช่ร่วม (used with)	0	n

Multiplicity

- 4. Association ระหว่าง Media และ Member

Association Name	Association Direction	Member			Media		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
ลงทะเบียน (Register)	Member → Media	ผู้ร้องขอ (requester)	0	n	สิ่งที่เป็นที่ต้องการ (required item)	0	n

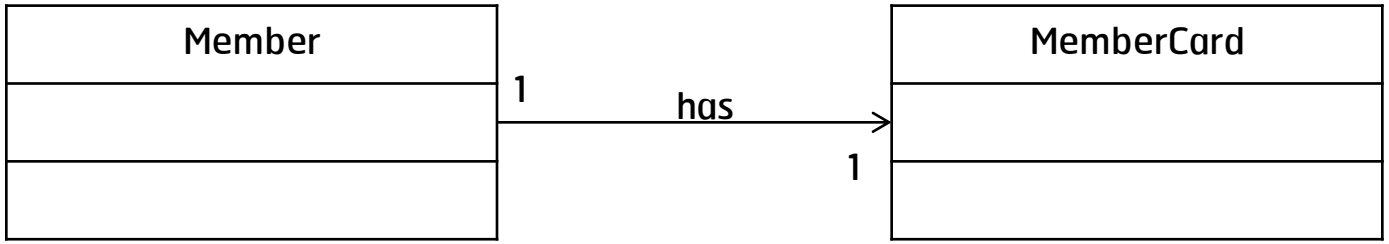
การจำลอง Association Abstract ด้วย UML

- การจำลอง Association ระหว่าง Class แสดงด้วยเส้นตรง ลากเชื่อมระหว่าง Class ทั้งสอง
- เส้นตรงต้องมี Association Name กำกับด้วยเสมอ
- เส้นตรงต้องมีหัวลูกศรก้างปลา เพื่อแสดง Association Direction
- แต่ละข้างของตัวอักษรต้องกำกับด้วย Min-card และ Max-card

การจำลอง Association Abstract ด้วย UML

- 1. Association ระหว่าง Member และ MemberCard

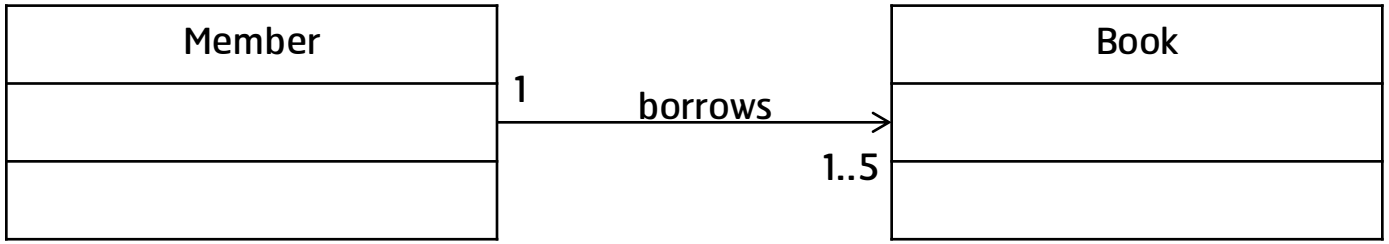
Association Name	Association Direction	Member			MemberCard		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
มี (has)	Member → MemberCard	เป็นเจ้าของ (owner)	1	1	ถูกเป็นเจ้าของ (owned item)	1	1



การจำลอง Association Abstract ด้วย UML

- 2. Association ระหว่าง Member และ Book

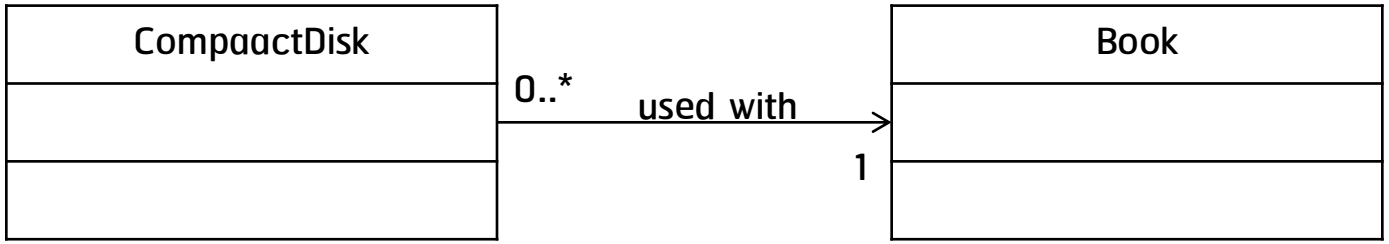
Association Name	Association Direction	Member			Book		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
ยืม (borrows)	Member → Book	เป็นผู้ยืม (borrower)	1	1	ถูกยืม (borrowed item)	1	5



การจำลอง Association Abstract ด้วย UML

- 3. Association ระหว่าง Book และ CompactDisk

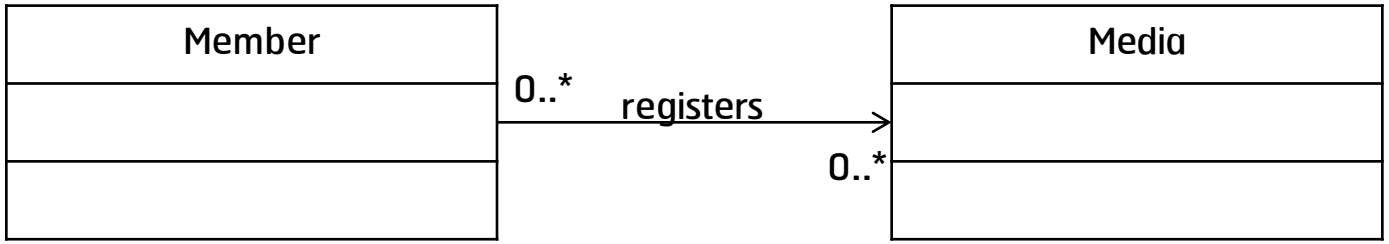
Association Name	Association Direction	CompaactDisk			Book		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
ใช้ร่วมกับ (used with)	CompaactDisk → Book	ใช่ร่วม (used with)	0	n	ใช่ร่วม (used with)	1	1



การจำลอง Association Abstract ด้วย UML

- 4. Association ระหว่าง Media และ Member

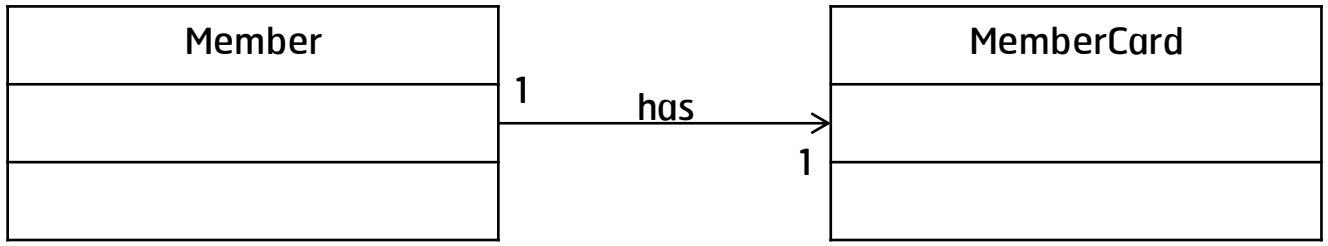
Association Name	Association Direction	Member			Media		
		Role Name	Multiplicity		Role Name	Multiplicity	
			Min	Max		Min	Max
ลงทะเบียน (Register)	Member → Media	ผู้ร้องขอ (requester)	0	n	สิ่งที่เป็นที่ต้องการ (required item)	0	n



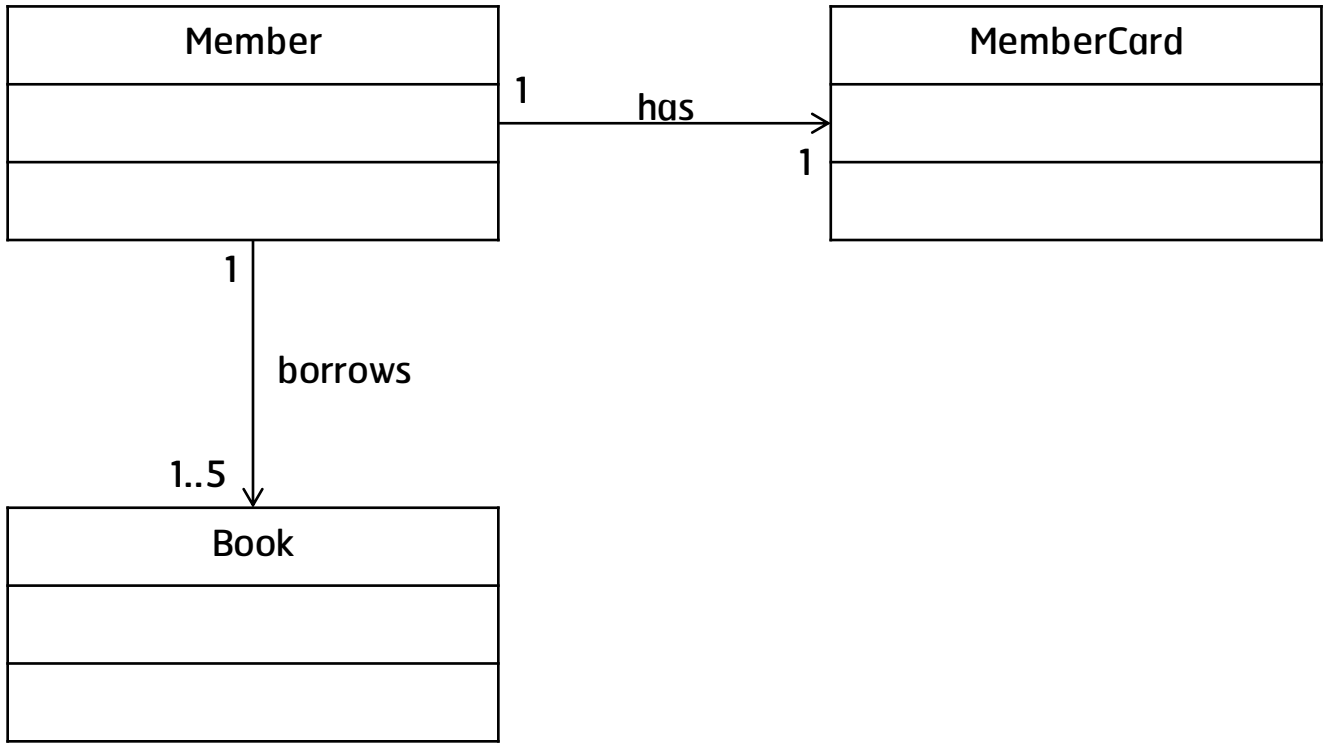
การจำลอง Association Abstract ด้วย UML

- Association แต่ละคู่มาจาก Problem Domain เดียวกัน ในการจำลองภาพ จะต้องนำ Association ทั้งหมดมารวมกัน

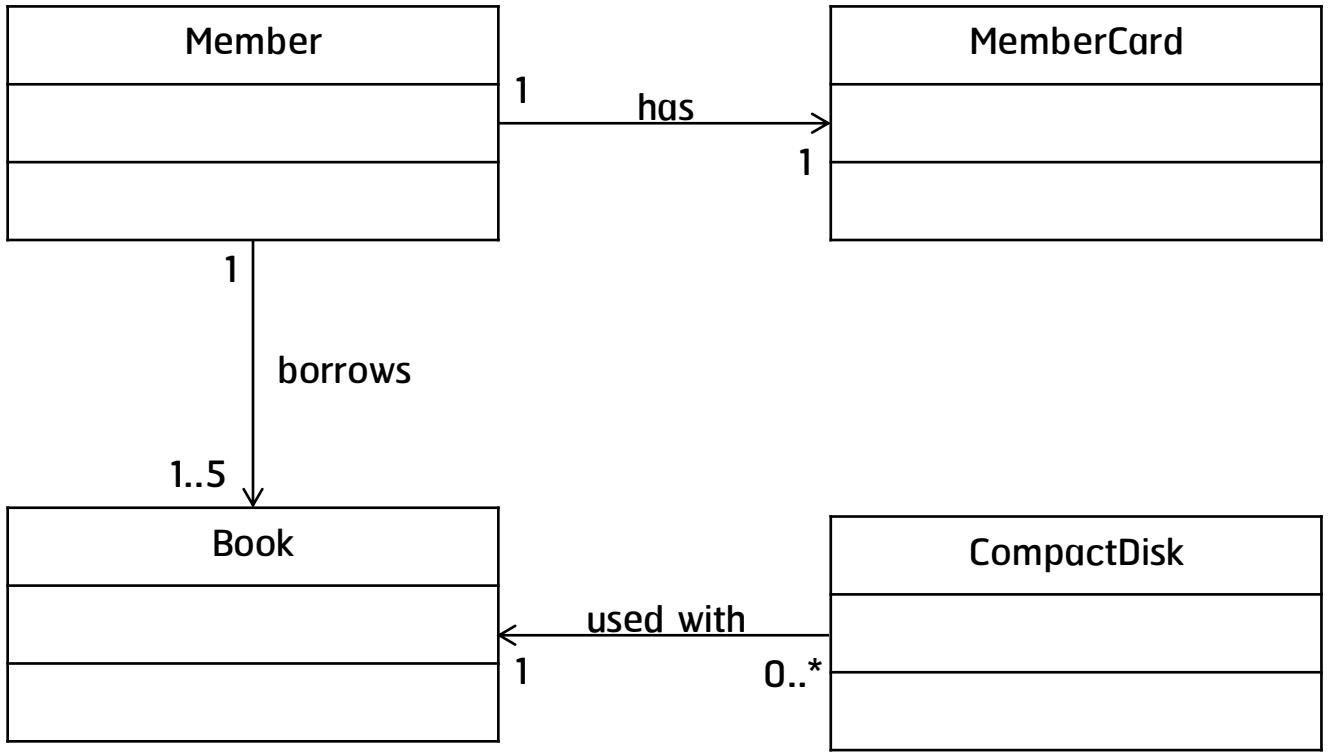
การจำลอง Association Abstract ด้วย UML



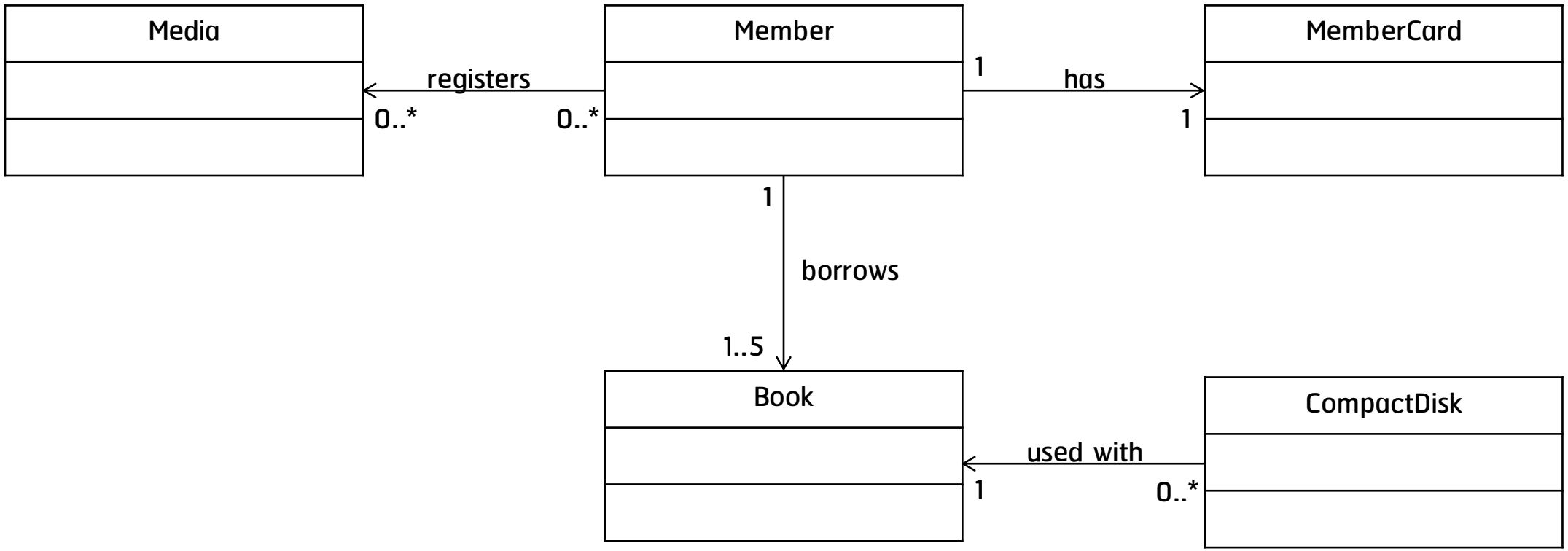
การจำลอง Association Abstract ด้วย UML



การจำลอง Association Abstract ด้วย UML



การจำลอง Association Abstract ด้วย UML



Mandatory และ Optional Class

- Mandatory Class
  - คือ Class ที่ต้องมีส่วนร่วมอยู่ใน Association เสมอ จะขาดไม่ได้ โดยสังเกตที่ Min-card ไม่เท่ากับ 0
- Optional Class
  - คือ Class ที่ไม่จำเป็นต้องมีส่วนร่วมใน Association โดย Optional Class สังเกตที่ Min-card เท่ากับ 0
- เช่น
  - จากตัวอย่าง Association ระหว่าง CompaactDisk กับ Book
    - Book ถือเป็น Mandatory Class
    - CompaactDisk ถือเป็น Optional Class

Association

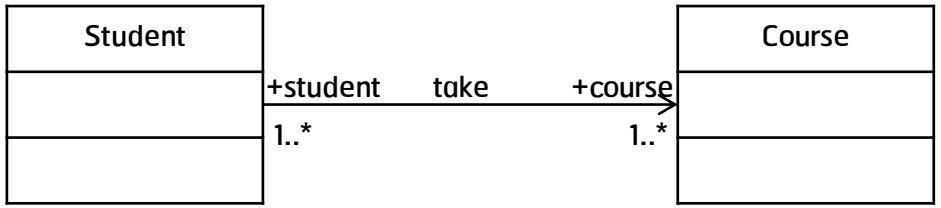
- Association นอกจากจะเป็นเรื่องของความสัมพันธ์ระหว่างคลาสแล้ว ยังมีองค์ประกอบอื่นๆ ที่ต้องคำนึงด้วย ดังนี้
  - Navigation
  - Visibility
  - Association Class

Association : Navigation

- การเขียนหัวลูกศรบนเส้น Association หมายถึง การใส่ Navigation ให้ Association
- Navigation หมายถึงการกำหนดการเข้าถึงระหว่าง Object ที่ต่าง Class กัน
- หัวลูกศรที่มีทิศทางเดียวจัดเป็น “Unidirectional Association”
- ถ้าทั้ง 2 Object เข้าถึงกันได้ (2 ทิศทาง) ไม่ต้องมีหัวลูกศร เรียกว่า “Bidirectional Association”

Association : Visibility

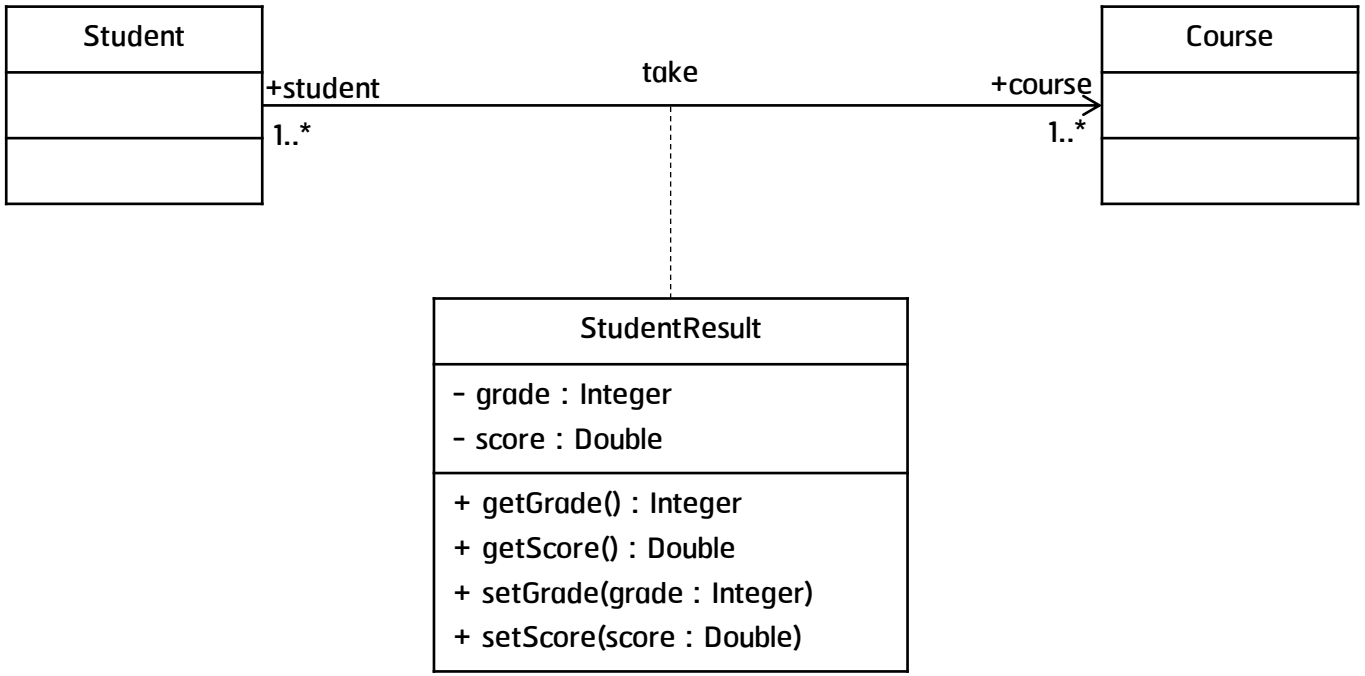
- ที่ผ่านมาได้กล่าวถึง Roles แล้ว
- เราสามารถใช้ Roles ในการกำหนดการเข้าถึงของ Class อื่นๆที่อยู่ภายนอก Association สามารถมองเห็นหรือเข้าถึง Association ได้หรือไม่ ด้วยการกำหนด Visibility ของ Role ซึ่งเป็นได้ทั้ง
  - Public Role
  - Protected Role
  - Private Role



- โดยทั่วไปใช้เฉพาะ Public และ Private Roles เท่านั้น

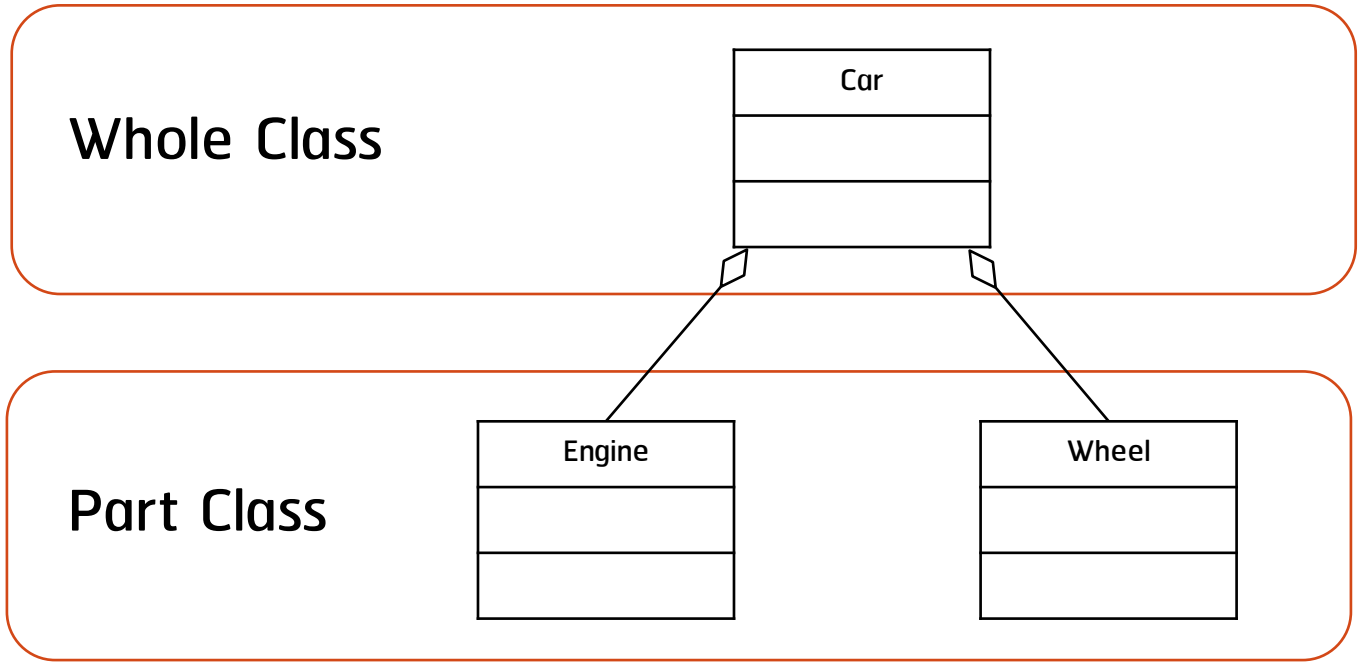
Association : Association Class

- คือ Class ที่ทำหน้าที่เป็นคุณสมบัติของ Association Abstraction
- มักใช้เพื่อแสดงรายละเอียดของความสัมพันธ์แบบ Many-to-Many Association



Association : Aggregation

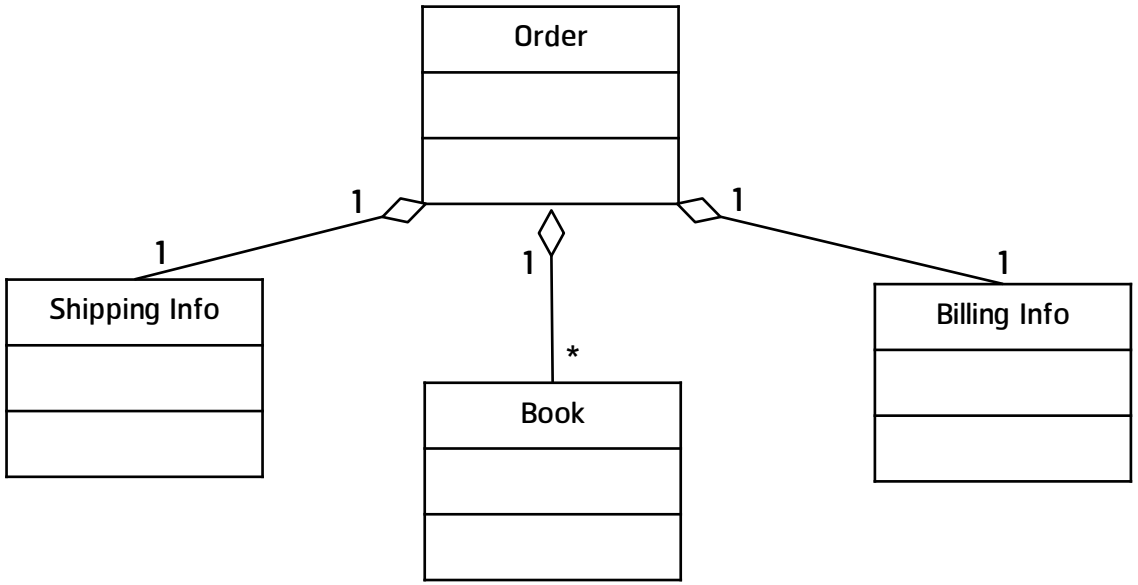
- คือ สิ่งที่กำหนดความสัมพันธ์ระหว่าง Class ในลักษณะขององค์ประกอบ
  - คลาสที่เป็นองค์ประกอบเรียกว่า “Part Class”
  - คลาสที่เกิดจากการรวมกันขององค์ประกอบ เรียกว่า “Whole Class”



Association : Aggregation

- ลักษณะสำคัญของ Aggregation คือ เมื่อลบ Whole Class ทิ้งไป Class ที่เป็น Part Classes จะยังสามารถคงอยู่ได้ โดยไม่ต้องพึ่งพา Whole Class เช่น
  - Computer กับ Harddisk ถ้า Computer เสีย Harddisk ก็ไม่จำเป็นต้องเสียด้วย
- ใน UML ความสัมพันธ์จะถูกตั้งสมมุติฐานไว้เป็นแบบ Bidirectional (ไม่มีหัวลูกศรเสมอ) เว้นแต่มีการกำหนดสัญลักษณ์ลูกศรเพื่อจำกัดไว้
- ถ้าไม่ระบุ Multiplicity ค่า default เป็น many parts and one whole.

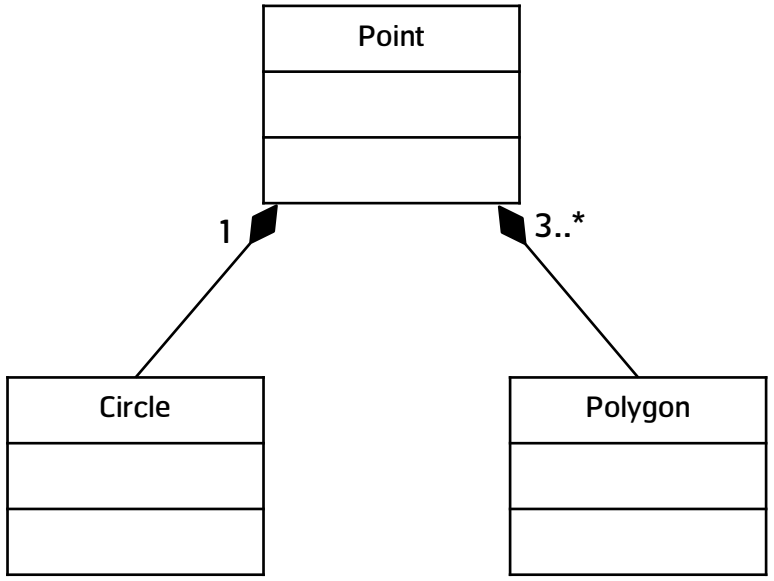
Association : Aggregation

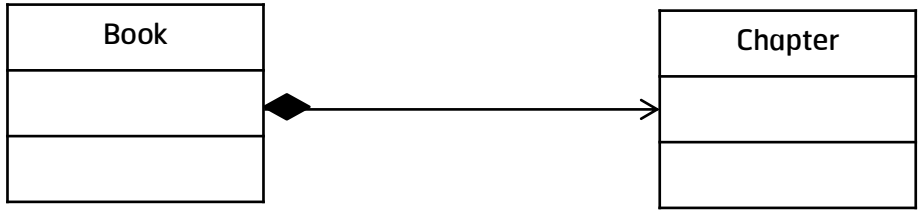


Association : Composition

- ส่วนที่เป็น Whole Class จะทำหน้าที่เป็นเจ้าของ Part Class
- ส่วนที่เป็น Part Class อาจเป็นของส่วนที่เป็น Whole Class เพียงหนึ่งเดียว
  - Multiplicity ของส่วนที่เป็น Whole ต้องเป็น 0 หรือ 1 เท่านั้น
- ช่วงชีวิตของส่วนที่เป็น Part Class จะขึ้นอยู่กับส่วนที่เป็น Whole Class เสมอ นั้นคือจะเกี่ยวข้องกับการสร้างหรือการลบส่วนที่เป็น Part Class เสมอ

Association : Composition





Aggregation & Composition

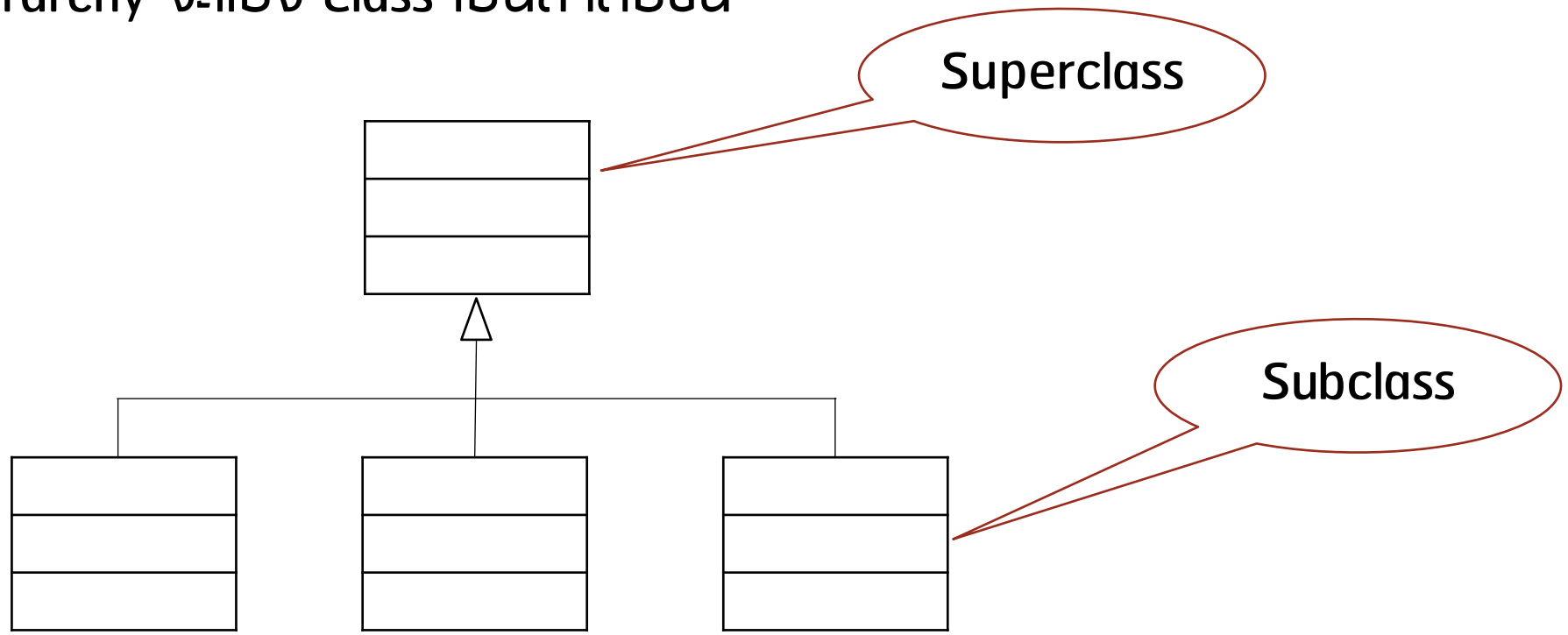
- เป็นความสัมพันธ์แบบ Transitive
  - ถ้าวัตถุ A เป็นส่วนหนึ่งของวัตถุ B และ B เป็นส่วนหนึ่งของ C แล้ว A เป็นส่วนหนึ่งของ C ด้วย เช่น
  - ถ้าที่จับประตูเป็นส่วนหนึ่งของประตู ประตูเป็นส่วนหนึ่งของรถยนต์ แล้วที่จับประตูเป็นส่วนหนึ่งของรถยนต์
- เป็นความสัมพันธ์แบบ Anti-symmetric
  - วัตถุใดๆ อาจไม่ต้องเป็นส่วนประกอบของตัวเองทั้งทางตรงและทางอ้อม เช่น
  - ถ้าประตูเป็นส่วนหนึ่งของรถยนต์ แต่รถยนต์ไม่จัดเป็นส่วนหนึ่งของประตู

Generalization(/Specialization)

- เป็นความสัมพันธ์แบบ Superclass และ Subclass ซึ่งก็คือ Inheritance
- Generalization เป็นการหาลักษณะร่วมของ Class ต่างๆ เพื่อสร้าง Class ที่เป็นตัวแทนของกลุ่ม Class นั้น (Superclass)
- Specialization เป็นการกำหนดลักษณะเฉพาะเจาะจงในรายละเอียด (Subclass)
- Specialization เป็นกระบวนการย้อนกลับของ Generalization

Generalization(/Specialization)

- ใน Class Hierarchy จะแบ่ง Class เป็นลำดับชั้น

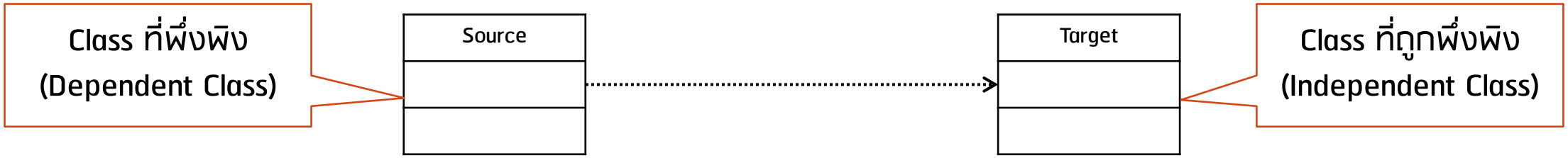


Inheritance

- Subclass จะรับคุณสมบัติ (Attribute and Methods) จาก Superclass มาโดยอัตโนมัติ
- Subclass สามารถเพิ่มเติมคุณสมบัติใหม่ (Attribute and Methods) จาก Superclass เรียกว่า Extending
- Subclass สามารถแก้ไขคุณสมบัติที่รับมา (Attribute and Methods) จาก Superclass เรียกว่า Overriding

Dependency

- เป็นความสัมพันธ์ระหว่าง Class แบบขึ้นต่อกัน
- การเปลี่ยนแปลงต่างๆ ที่เกิดขึ้นกับ Class หนึ่งจะมีผลกับอีก Class หนึ่ง



- Class Source จะขึ้นกับ Class Target
- เส้นแสดงความสัมพันธ์จะคล้ายกับ Association แต่เป็นเส้นประ

Class Diagram Modeling Techniques

- สรุป หลักการในการสร้าง Class Diagram
- 1. กำหนดกรอบของ Problem Domain ให้ชัดเจนและยึดถือ Problem Domain นี้เป็นมาตรฐานและบรรทัดฐานในการวิเคราะห์ระบบ เขียน Use case Diagram แล้วพิจารณาว่า ในแต่ละ Use case จะมี Object อะไรอยู่บ้าง ทำจนครบทุก Use case จะได้ Object ที่มีอยู่ใน Problem Domain ทั้งหมด

Class Diagram Modeling Techniques

- 2. พิจารณาหา Object ที่สามารถจับต้องได้ เรียกว่า Tangible Object หรือหาตัวแทนของ Tangible Object ในกรณีที่มี Tangible Object หลายๆ ตัวใน Problem Domain เดียวกันให้ครบทุกตัว
- 3. พิจารณาหา Object ที่ไม่สามารถจับต้องได้ เรียกว่า Intangible Object หรือหาตัวแทนของ Intangible Object ในกรณีที่มี Intangible Object หลายๆ ตัวใน Problem Domain เดียวกันให้ครบทุกตัว

Class Diagram Modeling Techniques

- 4. ใช้ Classification Abstraction เพื่อแยกแยะและสร้าง Class จาก Object ที่มีอยู่และพยายามหา Attribute และ Methods ที่มีอยู่ใน Class นั้นๆ เท่าที่จะหาได้ วาด Class ทั้งหมดลงใน Class Diagram
- 5. หา Aggregation Abstraction โดยพิจารณา Class ที่ได้จาก Classification Abstraction ว่ามี Class ใดที่ไม่มีความสัมพันธ์แบบเป็นส่วนหนึ่งหรือประกอบด้วย (whole-part) กับ Class อื่นๆ ถ้ามีพยายามหาว่า Aggregation ที่เกิดขึ้นนั้นเป็นแบบ One to One หรือ Many to One และใส่ Multiplicity ให้ถูกต้อง

Class Diagram Modeling Techniques

- 6. ใช้ Generalization มาพิจารณา Class ต่างๆ ใน Class Diagram หากเกิดความสัมพันธ์แบบ Generalization หรือ Specialization ให้เพิ่มเติมลงไปใน Class Diagram ซึ่งในขั้นตอนนี้อาจสร้าง Class ใหม่ได้
- 7. ใช้ Association มาพิจารณา Class ต่างๆ ใน Class Diagram เพิ่มเติมสัญลักษณ์ของ Association ลงใน Class Diagram และพิจารณาประเภทของความสัมพันธ์และ Multiplicity ให้ถูกต้อง

Class Diagram Modeling Techniques

- 8. พิจารณา Class Diagram ที่สร้างมาทั้งหมดว่า ทุก Class และทุกกลุ่มของ Class มีความสัมพันธ์ (Relationship) แบบใดแบบหนึ่งกับ Class หรือ กลุ่มของ Class อื่นหรือไม่
- หากว่ามี Class ใด Class หนึ่งหรือกลุ่มหนึ่ง ยังไม่มี Relationship ใดกับ Class อื่นเลย อาจเกิดจาก Class นั้น เกินความจำเป็น หรืออาจเกิดจากขาด Class อื่นๆที่จำเป็นต้องมีและมี Relationship กับ Class ดังกล่าว
- สิ่งที่ต้องทำถ้าเกิดกรณีแบบนี้คือ เริ่มต้นตั้งแต่ข้อ 1 ใหม่ โดยพิจารณาหา Object ที่น่าจะขาดหายหรือเกิน แล้วทำต่อมาจนจบหรือจนกว่าจะได้ Class Diagram ที่สมบูรณ์
- ข้อ 4 – 7 ทำสลับขั้นตอนกันได้

ตัวอย่าง

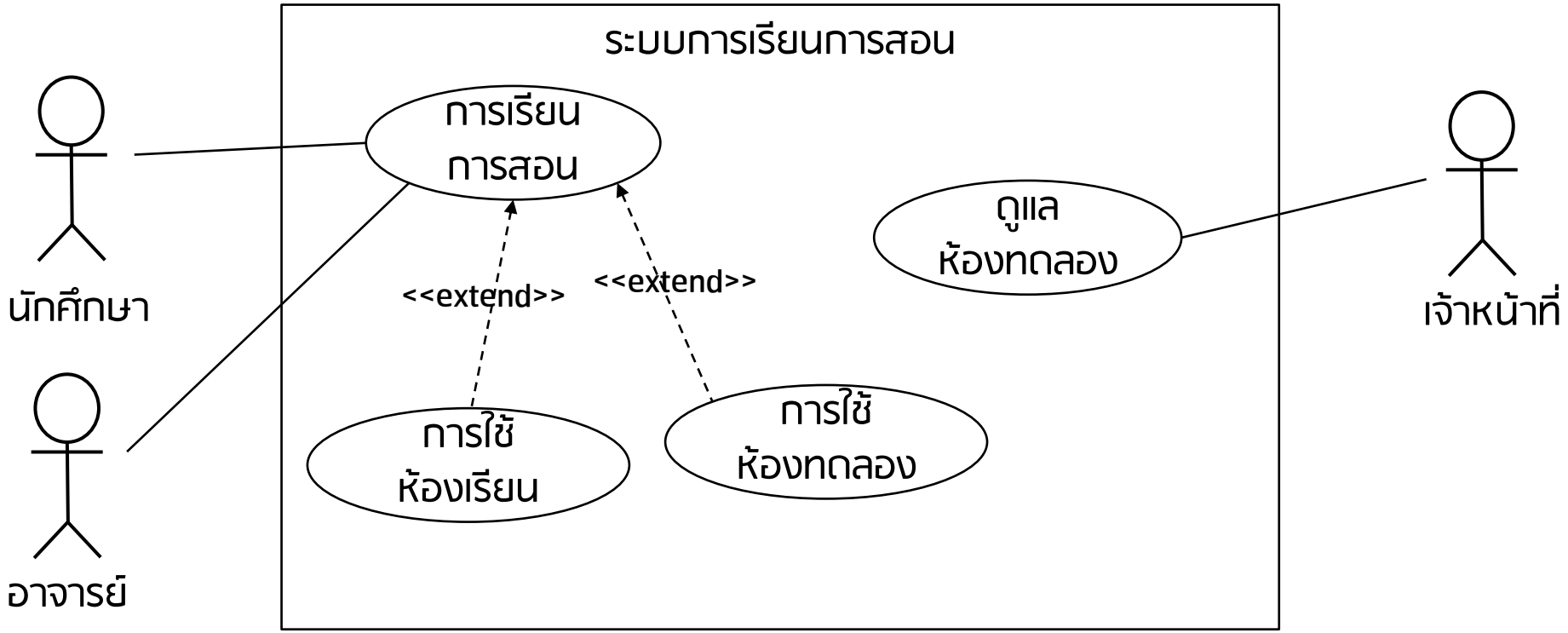
- จงสร้าง Class Diagram จาก Problem Domain ต่อไปนี้
- ในคณะวิทยาศาสตร์ของสถาบันการศึกษาแห่งหนึ่ง มีบุคลากรหลายประเภท ได้แก่ อาจารย์ นักศึกษา และเจ้าหน้าที่ โดยที่อาจารย์แต่ละท่านมีหน้าที่ในการสอนวิชาใดวิชาหนึ่ง หรือมากกว่า 1 วิชาก็ได้ และนักศึกษามีหน้าที่เรียนในรายวิชาใดวิชาหนึ่ง หรือมากกว่า 1 วิชาก็ได้ ในการเรียนแต่ละรายวิชาอาจมีการใช้ห้องเรียน หรือห้องทดลองก็ได้ เจ้าหน้าที่ของภาควิชา คือเจ้าหน้าที่ประจำห้องทดลองต่างๆ คอยดูแลห้องทดลอง โดยกำหนดให้ เจ้าหน้าที่ 1 คน ดูแลห้องทดลอง 1 ห้องเสมอ

ตัวอย่าง

- หา Use Case จาก Problem Domain
- Use case ของระบบคือ
  - การเรียนการสอน
    - ประกอบด้วย Actor คือ นักศึกษา อาจารย์
  - การใช้ห้องทดลอง
    - ประกอบด้วย Actor คือ นักศึกษา อาจารย์
  - การดูแลห้องทดลอง
    - ประกอบด้วย Actor คือ เจ้าหน้าที่

ตัวอย่าง

- เขียน Use Case Diagram

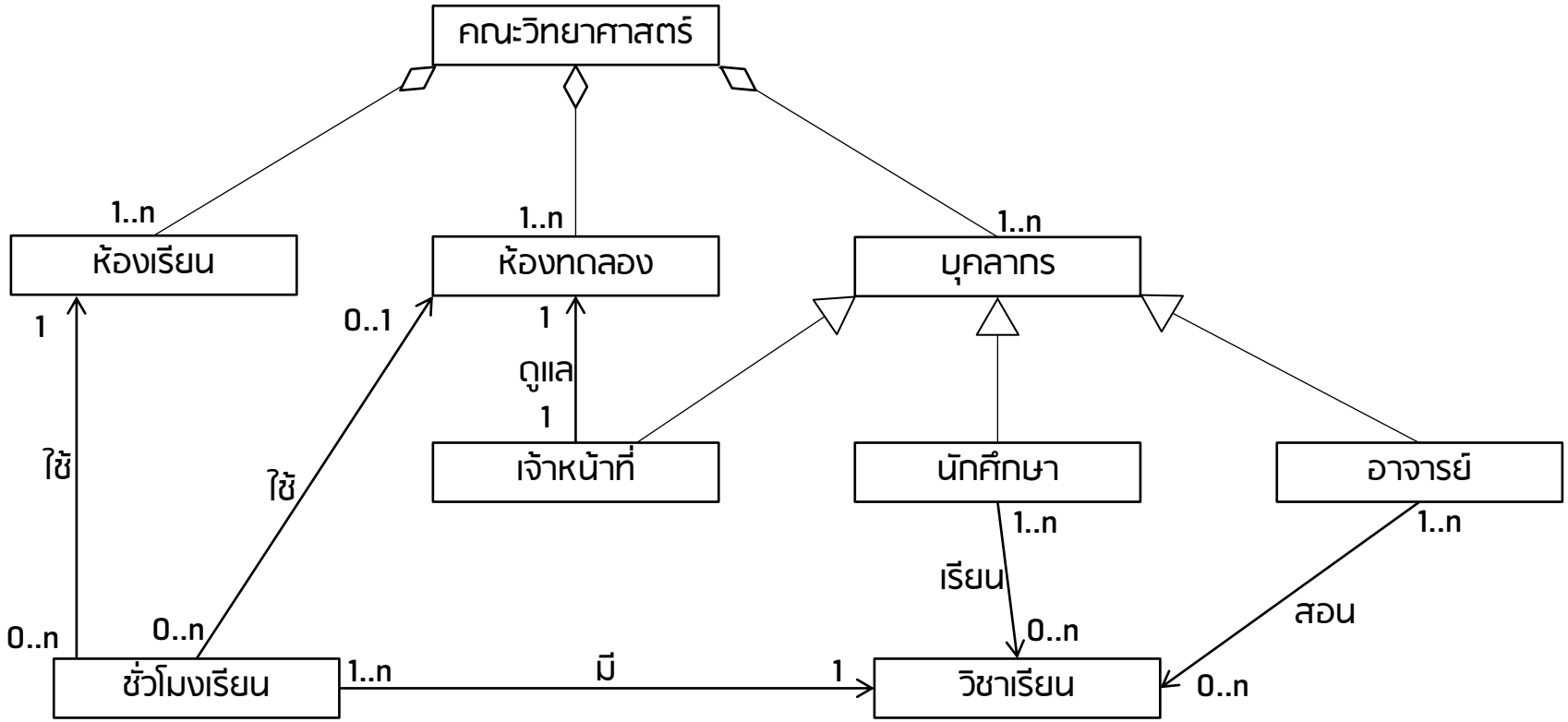


ตัวอย่าง

- จาก Use Case Diagram หา Objects/Classes ที่มีอยู่ในระบบ
  - Use case การเรียนการสอน
    - นักเรียน อาจารย์ // Object/Class Actor
    - ห้องเรียน วิชาเรียน ชั่วโมงเรียน // Object/Class อื่นๆ
  - Use case การใช้ห้องทดลอง
    - นักเรียน อาจารย์ // Object/Class Actor
    - ห้องทดลอง // Object/Class อื่นๆ
  - Use case การดูแลห้องทดลอง
    - เจ้าหน้าที่ // Object/Class Actor
    - ห้องทดลอง // Object/Class อื่นๆ

ตัวอย่าง

- เขียน Class Diagram เบื้องต้น (ไม่มี Attribute, Operations)



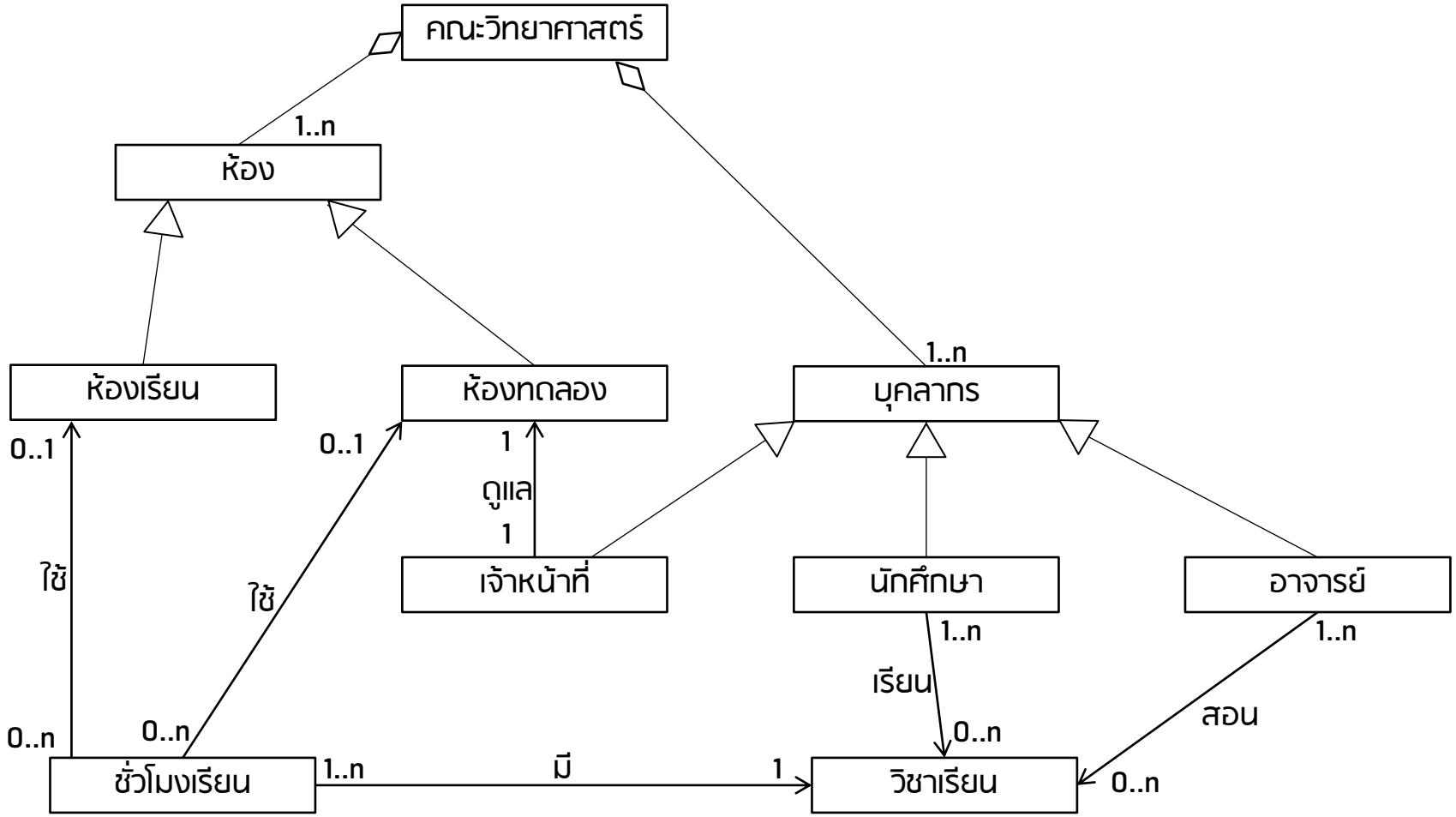
ตัวอย่าง

- จาก Class Diagram
- ได้มีการเพิ่มบาง Class
  - คณะวิทยาศาสตร์ เกิดจาก Aggregation Abstraction (ห้องเรียน ห้องทดลอง และ บุคลากร)
  - บุคลากร เกิดจาก Generalization (บุคลากร จำแนกเป็น เจ้าหน้าที่ นักเรียน อาจารย์)
- Tangible Class ได้แก่ บุคลากร ห้องเรียนและห้องทดลอง
- Intangible Class ได้แก่ คณะวิทยาศาสตร์ วิชาเรียน ชั่วโมงเรียน

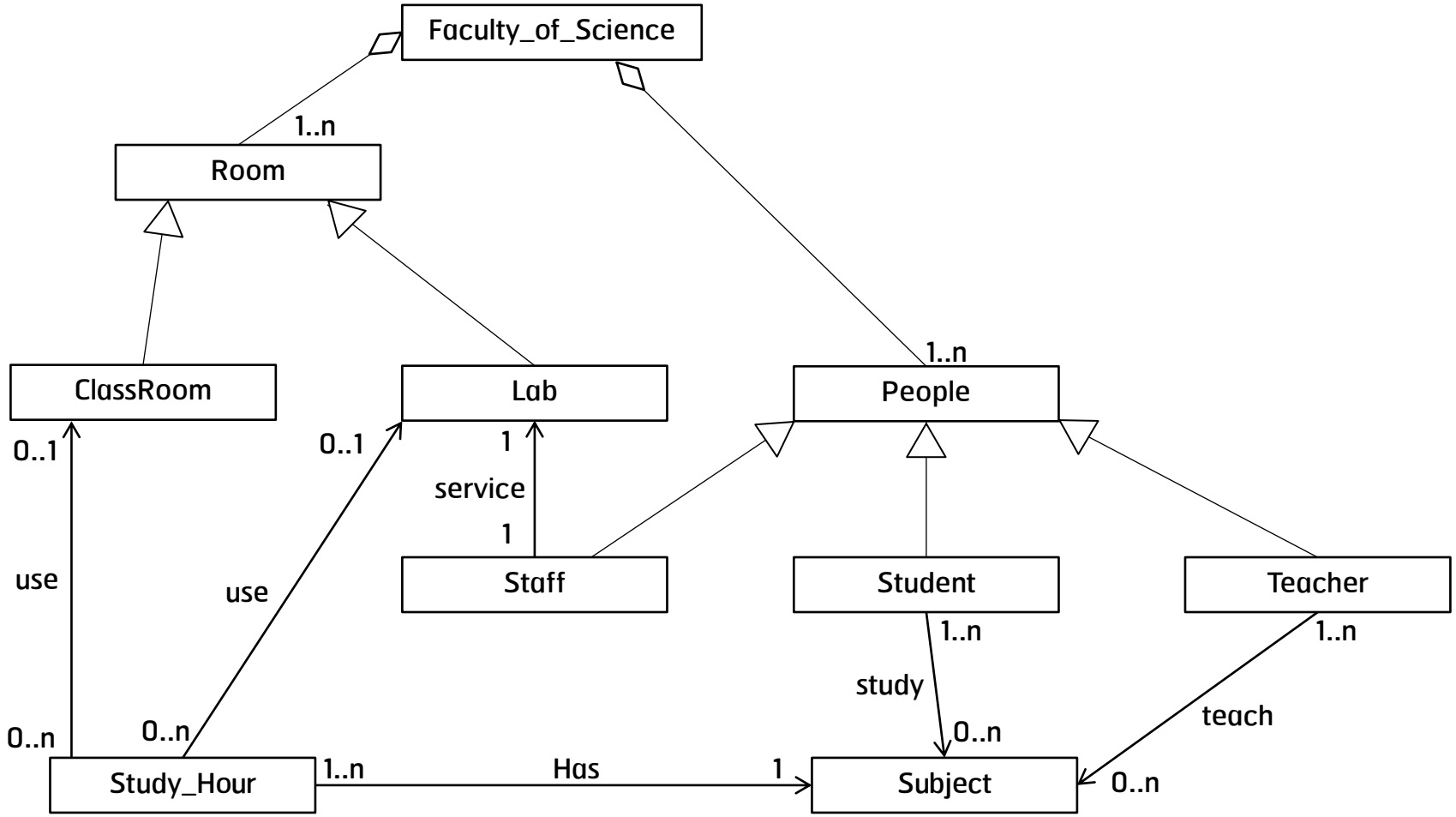
ตัวอย่าง

- เพิ่ม Class ชั่วโมงเรียน เพื่อเพิ่มความสมบูรณ์ (ให้เข้าใจง่ายขึ้น) แต่การเพิ่ม Class ต้องไม่ทำให้ Problem Domain เปลี่ยนไป
- จาก Class Diagram ข้างต้น จะเห็นว่า ห้องเรียน และ ห้องทดลอง มีคุณสมบัติที่คล้ายๆ กัน ดังนั้นควรทำ Generalization เพิ่ม Class ห้องขึ้นมาเพื่อให้สมบูรณ์ยิ่งขึ้น

ตัวอย่าง

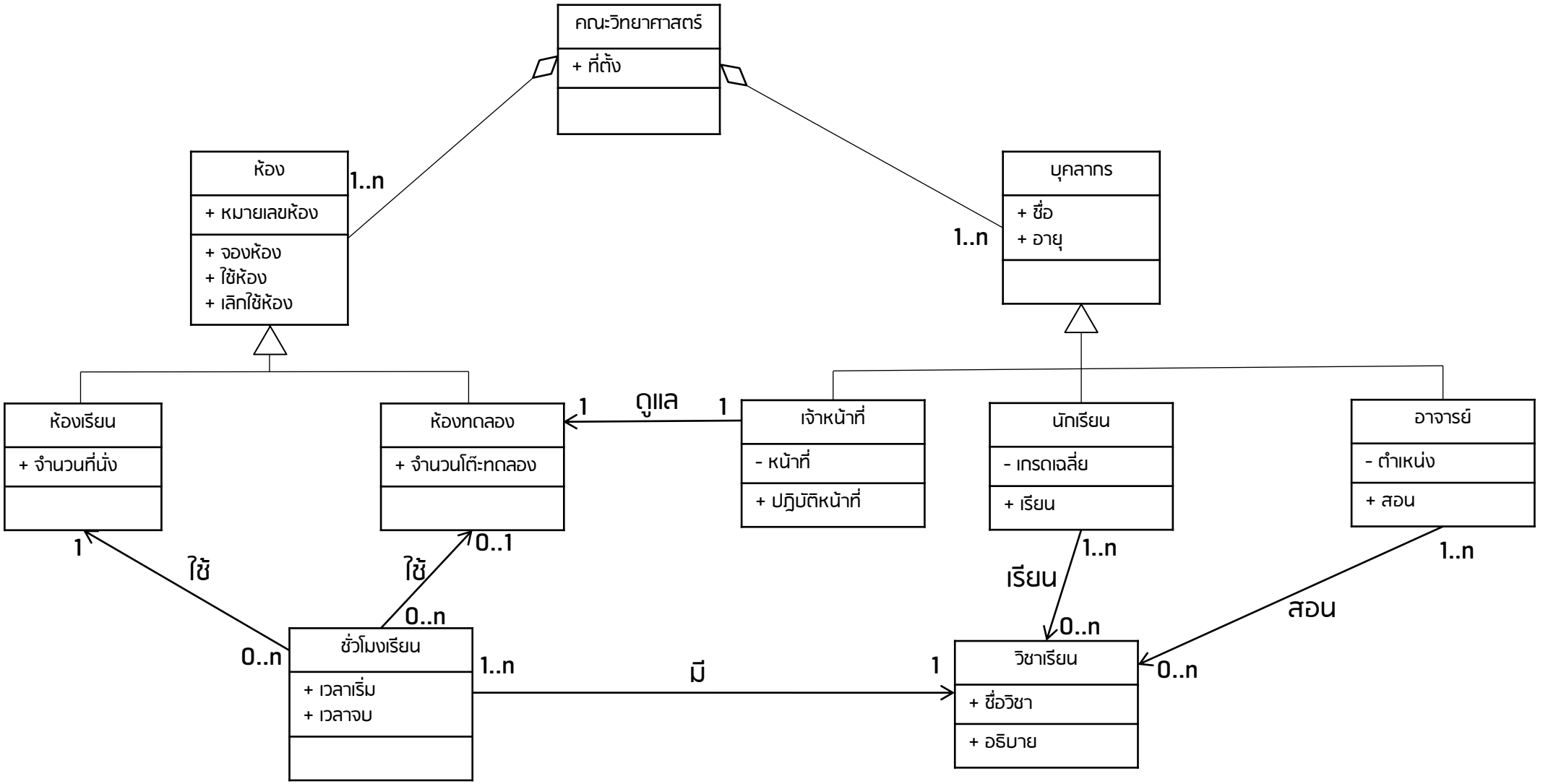

คณะวิทยาศาสตร์
1..n
ห้อง
ห้องเรียน
ห้องทดลอง
1..n
บุคลากร
0..1
0..1
1
ดูแล
1
ใช้
ใช้
เจ้าหน้าที่
นักศึกษา
อาจารย์
1..n
1..n
เรียน
สอน
0..n
0..n
0..n
ชั่วโมงเรียน
1..n
มี
1
วิชาเรียน
0..n

ตัวอย่าง

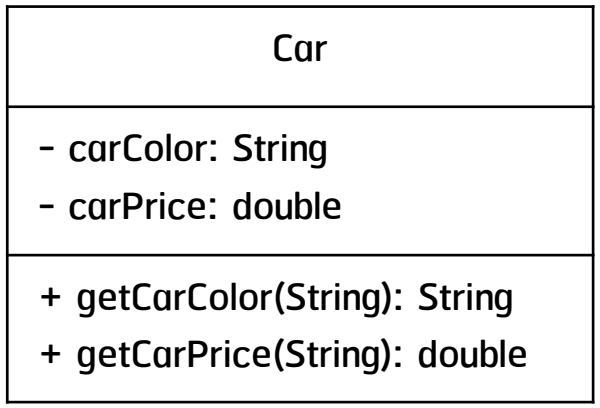

Faculty_of_Science
1..n
Room
ClassRoom
Lab
People
1..n
0..1
0..1
1
service
1
use
use
Staff
Student
Teacher
1..n
1..n
study
teach
0..n
0..n
0..n
Study_Hour
1..n
Has
1
Subject
0..n

ตัวอย่าง

- เมื่อแน่ใจกับโครงสร้างของ Class Diagram แล้ว สามารถเพิ่ม Attribute และ Operations เข้าไป



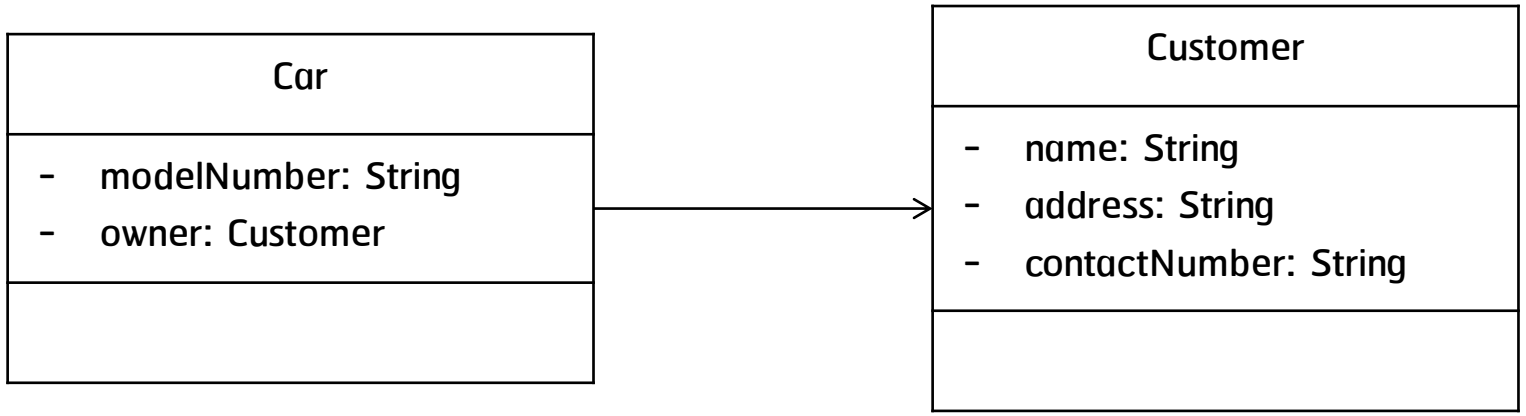
Class Diagram to Code in Java



```
public class Car {
    private String carColor;
    private double carPrice = 0.0;
    public String getCarColor(String model) {
        return carColor;
    }

    public double getCarPrice(String model) {
        return carPrice;
    }
}
```

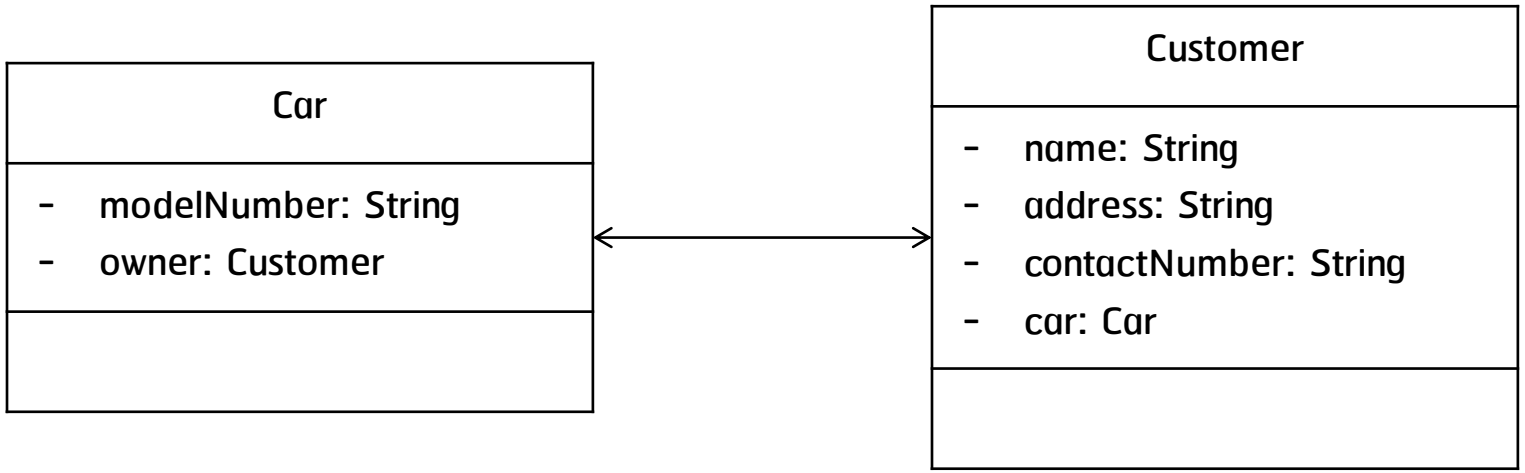
Class Diagram to Code in Java




```
public class Customer {
    private String name;
    private String address;
    private String contactNumber;
}

public class Car {
    private String modelNumber;
    private Customer owner;
}
```

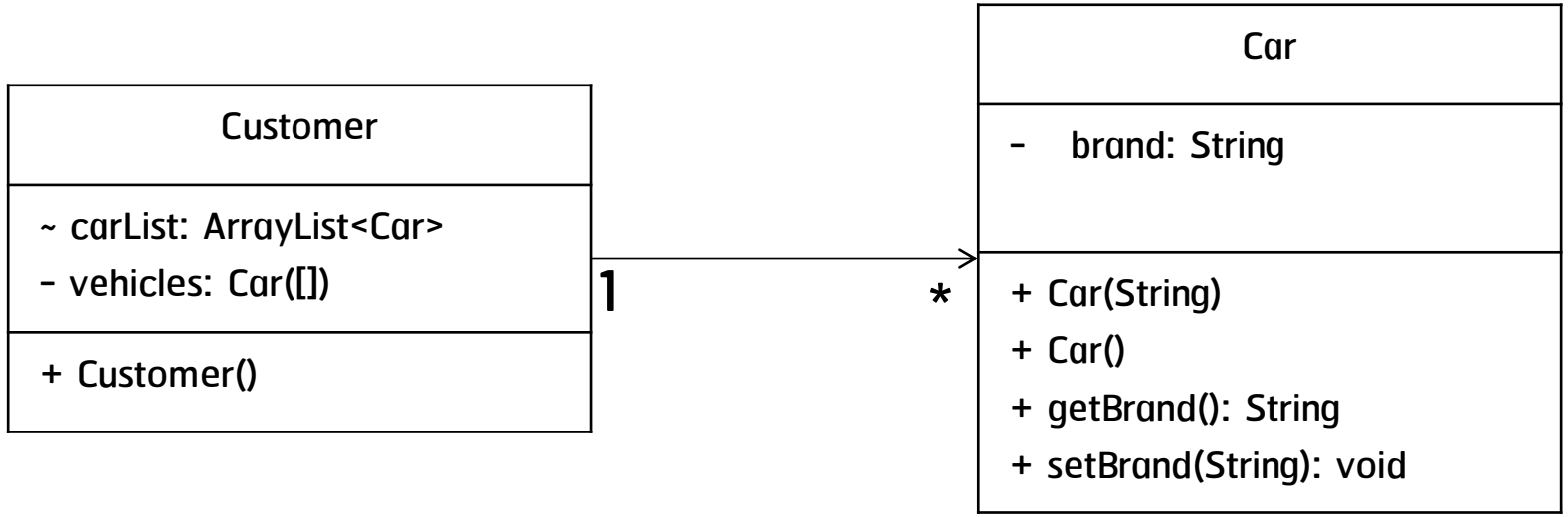
Class Diagram to Code in Java




```
public class Customer {
    private String name;
    private String address;
    private String contactNumber;
    private Car car;
}

public class Car {
    private String modelNumber;
    private Customer owner;
}
```

Class Diagram to Code in Java




```java
public class Customer {
    private Car[] vehicles;
    ArrayList<Car> carList = new ArrayList<Car>();
    public Customer(){
        vehicles = new Car[2];
        vehicles[0] = new Car("Audi");
        vehicles[1] = new Car("Mercedes");

        carList.add(new Car("BMW"));
        carList.add(new Car("Chevy"));
    }
}
```

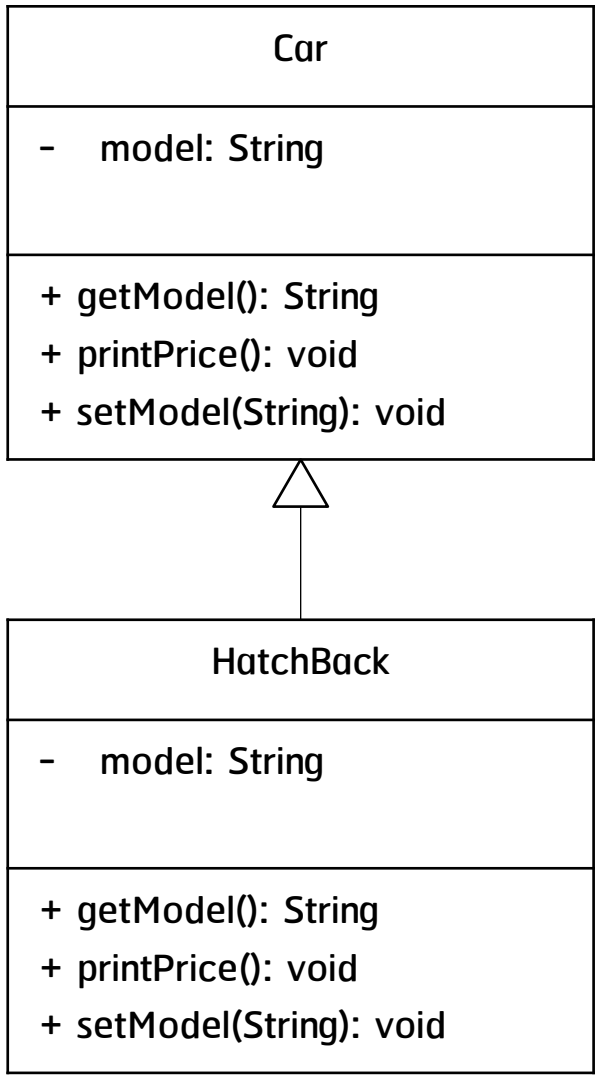
```java
public class Car {
    private String brand;

    public Car(String brands){
        this.brand = brands;
    }
    public Car() {
    }
    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

}
```

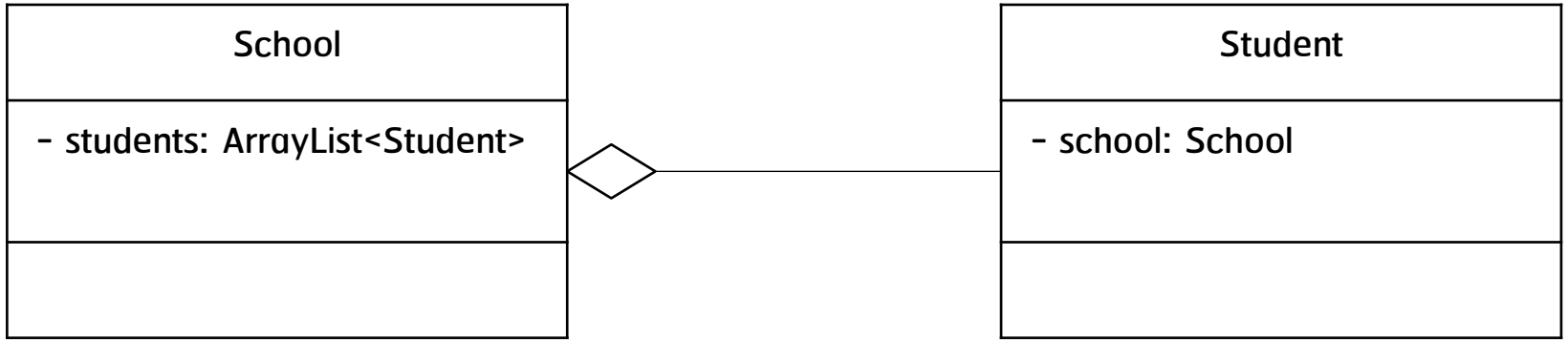
Class Diagram to Code in Java




```java
public class Car {
    private String model;
    public void printPrice() {
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
}

public class HatchBack extends Car {
    private String model;
    public void printPrice() {
        System.out.println("Hatchback Price");
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
}
```

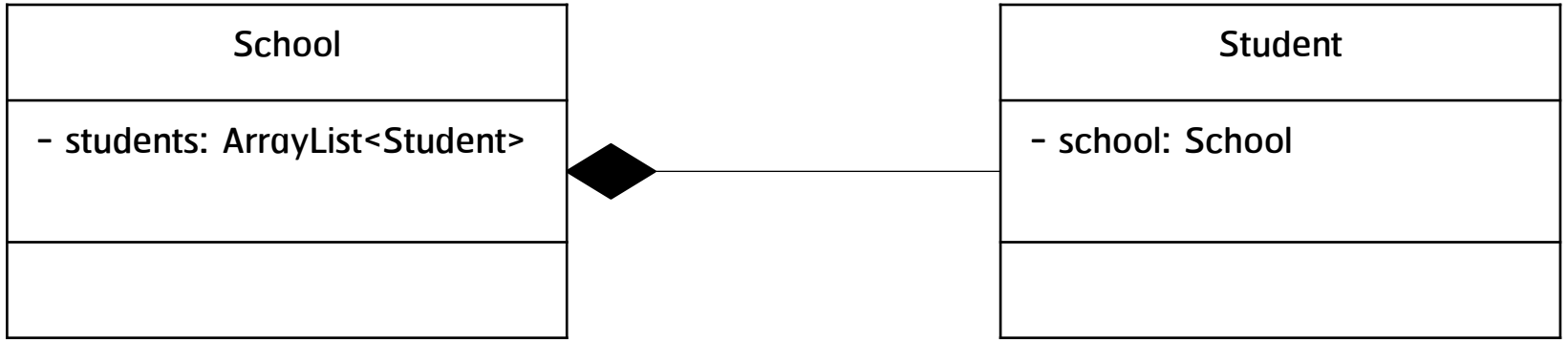
Class Diagram to Code in Java




```
public class Student {
    private School school;
}

public class School {
    ArrayList<Student> students;
}
```

Class Diagram to Code in Java




```
public class Student {
    private School school;
}

public class School {
    ArrayList<Student> students;
}
```

สรุป

แบบฝึกหัด

- จงเขียน Use Case Diagram -> Class Diagram -> Java Code จาก Requirement
  - ในระบบร้านค้าแห่งหนึ่งมีพนักงาน และผู้จัดการ ซึ่งร้านค้าแห่งนี้แบ่งการทำงานเป็น 2 รอบ คือ รอบกลางวันและรอบกลางคืนแต่ละรอบการทำงานจะมีพนักงานประจำเคาเตอร์ 2 คน ผู้จัดการมีหน้าที่ตรวจสอบการทำงานของพนักงานในแต่ละรอบการทำงาน วันละ 2 ครั้ง รอบการทำงานละ 1 ครั้ง ลูกค้าที่เข้ามาซื้อสินค้าสามารถซื้อสินค้าได้หลายชิ้น โดยการทำรายการขายต้องมีพนักงาน 1 คนเป็นผู้ทำรายการ เมื่อสิ้นสุดรอบการทำงานพนักงานประจำรอบการทำงานนั้นๆ 1 คน ต้องตรวจสอบ จำนวนสินค้าคงเหลือ และสรุปยอดขายในแต่ละรอบการทำงานได้