01076105, 01075106

Object Oriented Programming

Object Oriented Programming Project

UML

Object Oriented Analysis and Design Concepts

ขั้นตอนในการวิเคราะห์และออกแบบระบบ

1. รวบรวมความต้องการ กำหนดปัญหาที่จะแก้ ระบุว่าระบบงานที่จะทำต้องมีฟังก์ชันอะไรบ้าง (Requirement)
2. อธิบายระบบ ในมุมของผู้ใช้ อาจใช้ Wireframe, Mock-up หรือ Prototype (UI Design)
3. ระบุว่ามี Class อะไรบ้าง และ เขียน Diagram แสดงรายละเอียดและ Behavior ของการทำงานระหว่าง Class โดยใช้ UML

Requirement

- ความต้องการของระบบ แบ่งออกเป็น 2 อย่าง คือ
  - Functional Requirement คือ เป็นสิ่งที่ระบุว่าโปรแกรมต้องมี หรือ ต้องทำได้ รวมถึงการระบุว่า เมื่อได้รับ Input แล้วโปรแกรมจะตอบสนองอย่างไร
  - ตัวอย่าง App เพื่อการวิ่ง
    - ความเร็วในการวิ่งต้องแสดงในหน้าจอหลัก (show speed)
    - เลือกหน่วยความเร็วได้
    - จับเวลาในการวิ่งได้
    - ฯลฯ

Requirement

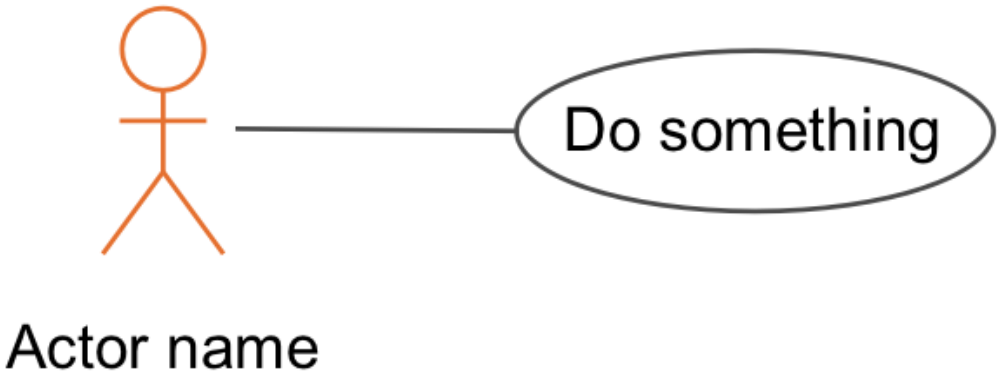
- อีกส่วนคือ
  - Non-Functional Requirement คือ เป็นความต้องการ แต่จะไม่เป็น Feature ของโปรแกรม
  - ตัวอย่าง Non-Functional Requirement สำหรับ App เพื่อการวิ่ง
    - การตอบสนองของ App (Performance)
    - การเก็บข้อมูลผู้ใช้ (Sensitive Data)
    - ความง่ายในการใช้งาน
    - คู่มือการใช้งาน

Example : Travel Expense App

- Functional
  - สามารถเก็บค่าใช้จ่ายตลอดการเดินทางได้
  - ต้องสามารถเก็บได้หลายสกุลเงิน (กรณีไปต่างประเทศ) และสามารถกำหนดสกุลเงินฐานได้
  - App จะต้องสามารถแปลงค่าใช้จ่ายเป็นสกุลเงินฐานได้
  - App ต้องแยกประเภทค่าใช้จ่ายได้
- Non-Functional
  - ต้องรันใน iOS v. xxx หรือ Android v. xxx ขึ้นไป
  - App ต้องไม่ใช้ข้อมูลเครือข่ายโดยไม่อนุญาต
  - App ต้องมี Support (เช่น E-mail หรืออื่นๆ)

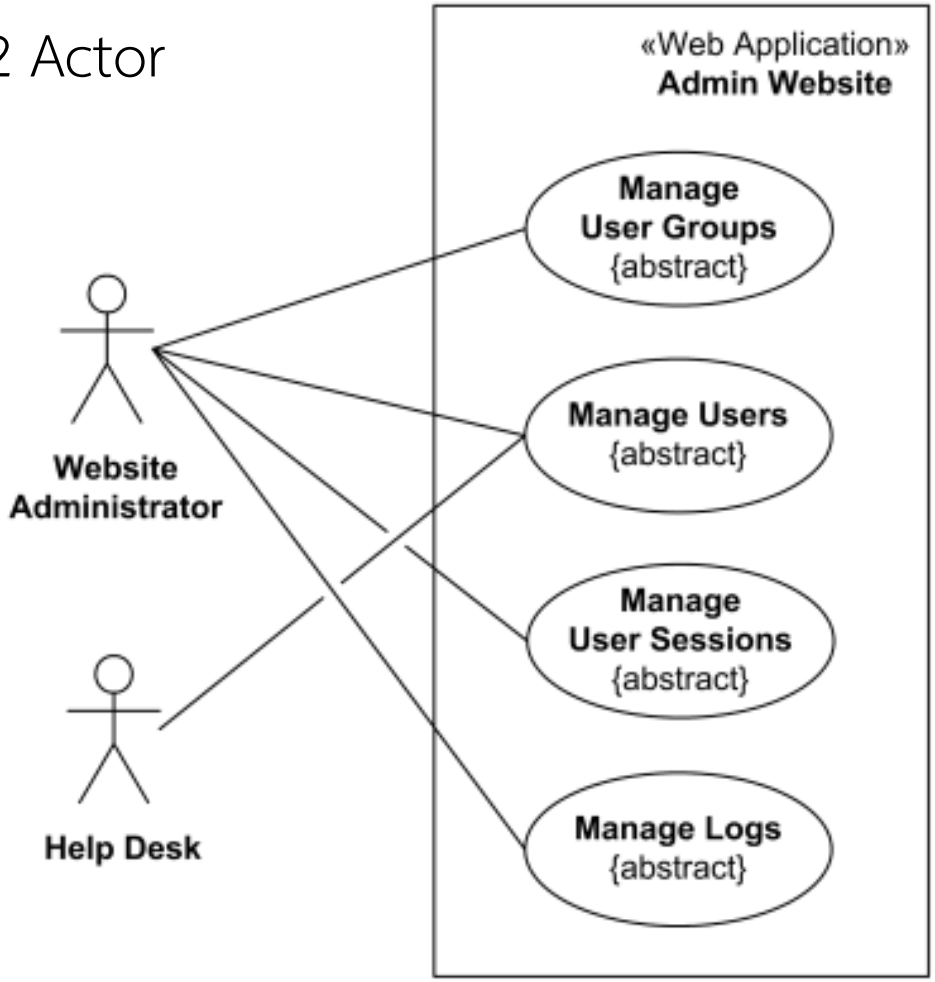
Requirement

- ปกติเราจะเขียน Requirement ใน 2 รูปแบบ คือ แบบตัวอักษร และ โดยใช้ Use Case Diagram
- สำหรับ Use Case มักจะมีองค์ประกอบเบื้องต้น 2 ส่วน คือ
  - Actor ซึ่งหมายถึง ผู้ใช้แต่ละกลุ่ม
  - Use Case หมายถึง การทำงานที่ผู้ใช้สามารถทำได้ มักใช้เป็นคำกริยา



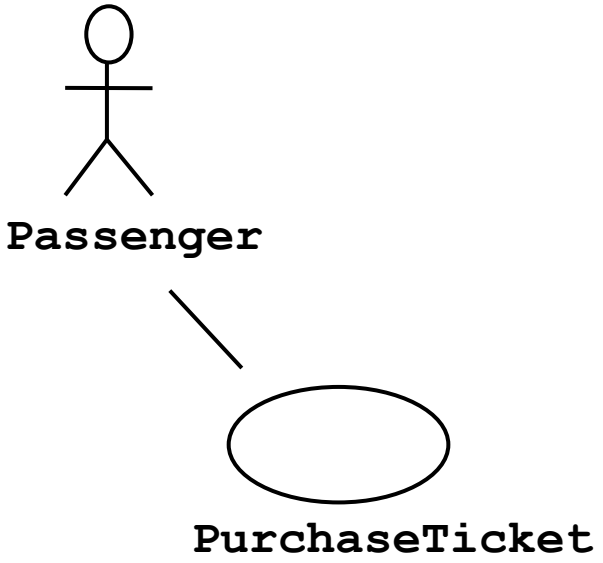
Use Case Diagram

- จากรูป ส่วนของ admin web มี 2 Actor
  - ผู้ดูแลเว็บไซต์ สามารถ
    - จัดการกลุ่มผู้ใช้ได้
    - จัดการผู้ใช้ได้
    - จัดการ user sessions
    - จัดการ logs
  - Help Desk สามารถ
    - จัดการผู้ใช้



Use Case Diagram

- ในการสร้าง Use Case Diagram
  - ค้นหา Actor
  - ค้นหา Use Case (การทำงาน) ที่มีปฏิสัมพันธ์กับ Actor นั้นโดยตรง
  - ค้นหาและสร้างความสัมพันธ์ระหว่าง Use Case หรือ Actor
- เขียนคำอธิบายแต่ละ Use Case จนครบถ้วน



Use Case Diagram

Name: `Purchase ticket`

Participating actor: `Passenger`

Entry condition:

- `Passenger` standing in front of ticket distributor.
- `Passenger` has sufficient money to purchase ticket.

Exit condition:

- `Passenger` has ticket.

Event flow:

1. `Passenger` selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. `Passenger` inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Anything missing?

Exceptional cases!

Use Case Diagram

สัญลักษณ์

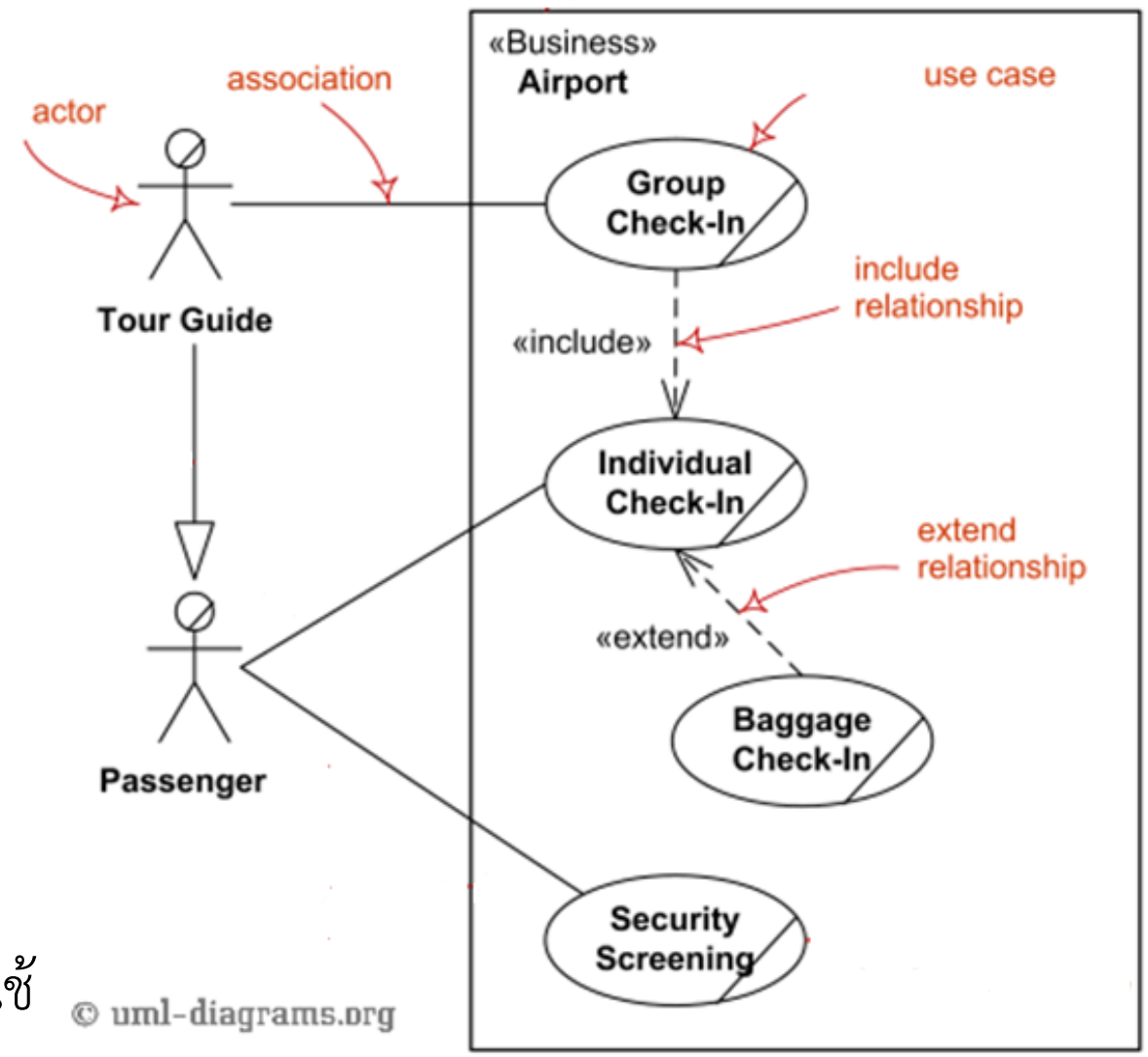
- Use case
- Actor
- Connection

ความสัมพันธ์

- Extend

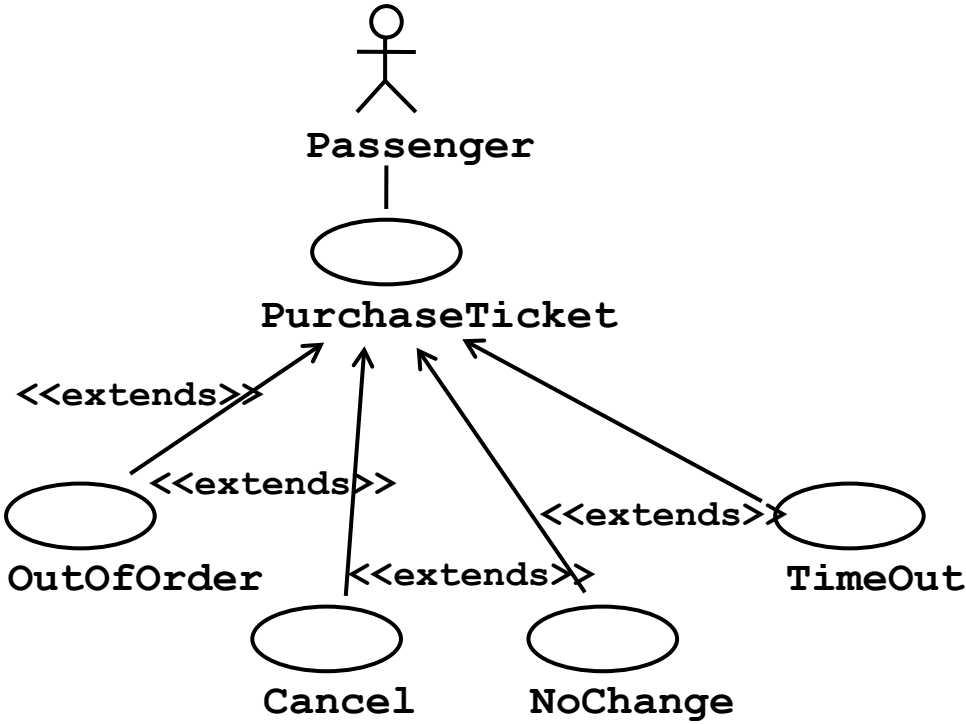
  ความสัมพันธ์แบบขยาย
- Include

  ความสัมพันธ์แบบเรียกใช้



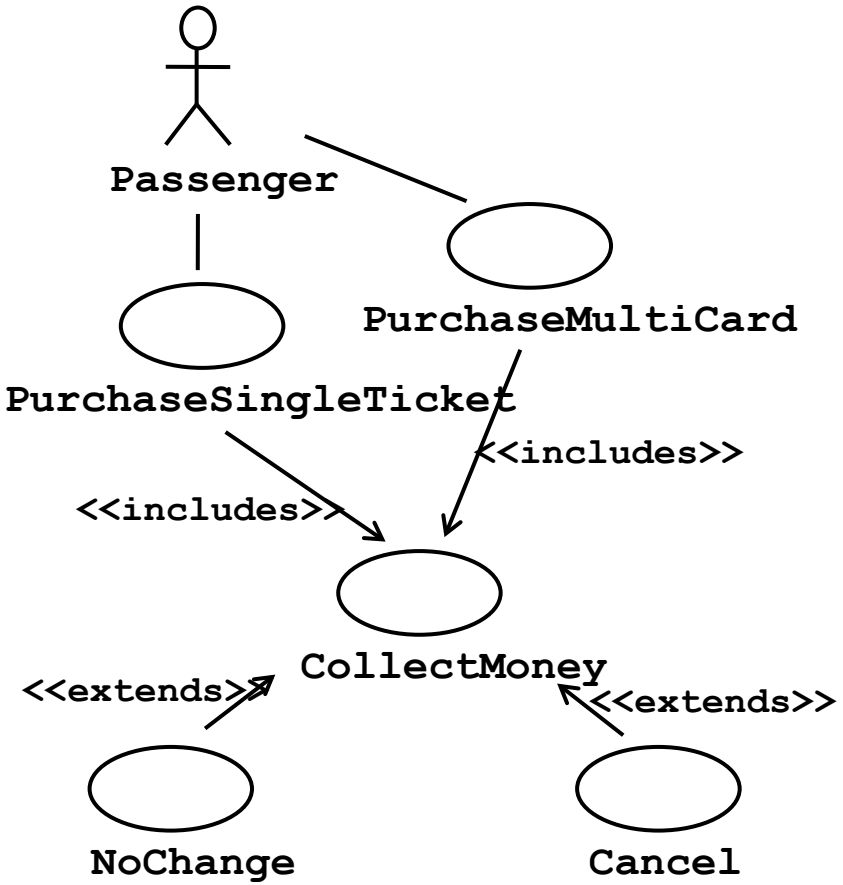
Use Case Diagram

- ความสัมพันธ์ แบบ *<<extends>>*
  - <<extends>> จะใช้แทนการทำงานบางอย่าง บางกรณีที่อาจเกิดร่วม หรือ เฉพาะอย่าง
  - เพื่อให้เห็นการทำงานในกรณีต่างๆ ให้ชัดเจนมากขึ้น เช่น ตั๋วหมด ยกเลิก หมดเวลา
  - ลูกศรจะชี้ไปยัง use case ตัวที่มีการ extend



Use Case Diagram

- ความสัมพันธ์ แบบ *<<include>>*
  - <<include>> จะใช้แทนกรณีที่เกี่ยวเนื่องกับ use case
  - เพื่อการ reuse เช่น ซื้อตั๋ว 1 คนกับหลายคน ต้องมีการเก็บเงินเหมือนกัน
  - ลูกศรจะชี้ไปยัง use case ตัวที่มีการเรียกใช้



Case Study : ห้องสมุด

- สมาชิกห้องสมุดต้องสามารถค้นหาหนังสือได้ โดยใช้ Title, Author, Subject Category และ Publication Date
- หนังสือแต่ละเล่มจะมี รหัสประจำเล่ม (Identification No.) ที่ไม่ซ้ำกัน แม้จะมีชื่อเดียวกัน โดยระบุหนังสือแต่ละเล่มโดยใช้ Item No.
- นอกจากนั้นหนังสือจะมีรายละเอียดอื่นๆ เช่น Rack No. ซึ่งจะเป็นการระบุชั้นหนังสือที่เก็บอยู่
- สมาชิก สามารถ จองหนังสือ ยืมหนังสือ และ คืนหนังสือ เล่มใดก็ได้
- ระบบ สามารถค้นหาได้ว่าใครยืมหนังสือเล่มใดไป หรือ ค้นหาว่าสมาชิกคนนั้นยืมหนังสือเล่มใดไปบ้าง

Case Study : ห้องสมุด

- สมาชิกแต่ละคน สามารถยืมหนังสือได้ 5 เล่ม
- สมาชิกแต่ละคน สามารถยืมหนังสือได้เป็นระยะเวลา 10 วัน
- สมาชิก สามารถจองคิว หนังสือที่ถูกยืมไปแล้วได้
- ระบบ สามารถแจ้งเตือนว่าหนังสือที่ยืมไป มีการคืนมาแล้ว หรือ ไม่มาคืนภายในกำหนด
- หมายเลขสมาชิก และ เลข Item No. ของหนังสือจะไม่ซ้ำ

Case Study : ห้องสมุด

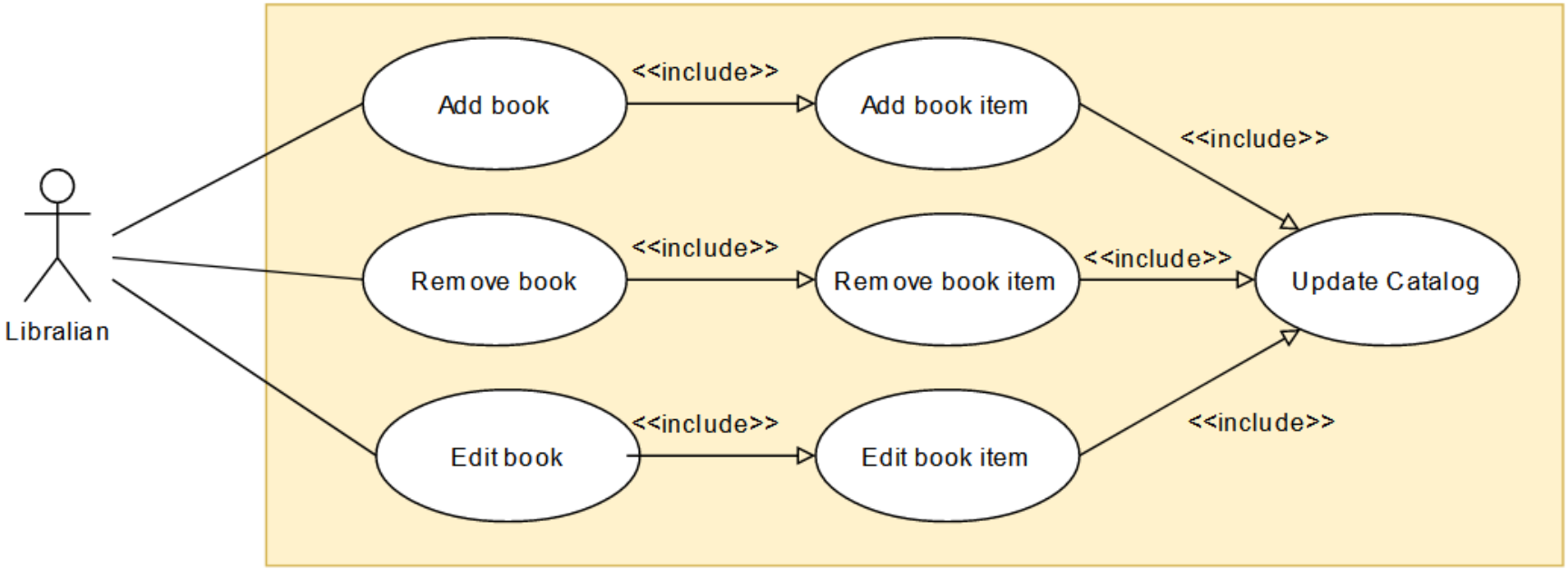
- ในระบบจะมี actor จำนวน 3 actor คือ
- Librarian: รับผิดชอบในการเพิ่ม แก้ไข books, book items และ users. โดยบรรณารักษ์จะให้ยืม issue, ทำจอง reserve, และรับคืน book items.
- Member: สมาชิกแต่ละคนสามารถค้นหาหนังสือใน catalog และขอยืม check-out, ขอจอง reserve, ต่ออายุ renew, และส่งคืนหนังสือ
- System: รับผิดชอบในการส่งการแจ้งเตือน สำหรับหนังสือที่เกิดกำหนดเวลาคืน overdue books, การยกเลิกการจอง canceled reservations, และอื่นๆ

Case Study : ห้องสมุด

- Use cases ของระบบห้องสมุด
  - Add/Remove/Edit book: เพิ่ม, ลบ หรือ แก้ไข book หรือ book item.
  - Search catalog: ค้นหาหนังสือ โดยใช้ title, author หรือ subject
  - Register new account/cancel membership: เพิ่มสมาชิกใหม่ หรือ ยกเลิกสมาชิกเดิม
  - Check-out book: ให้ยืมหนังสือจากห้องสมุด
  - Reserve book: จองหนังสือที่มีผู้อื่นยืมไป
  - Renew a book: ยืมต่อสำหรับหนังสือที่เรายืมอยู่ และไม่มีคนจอง
  - Return a book: คืนหนังสือให้กับห้องสมุด

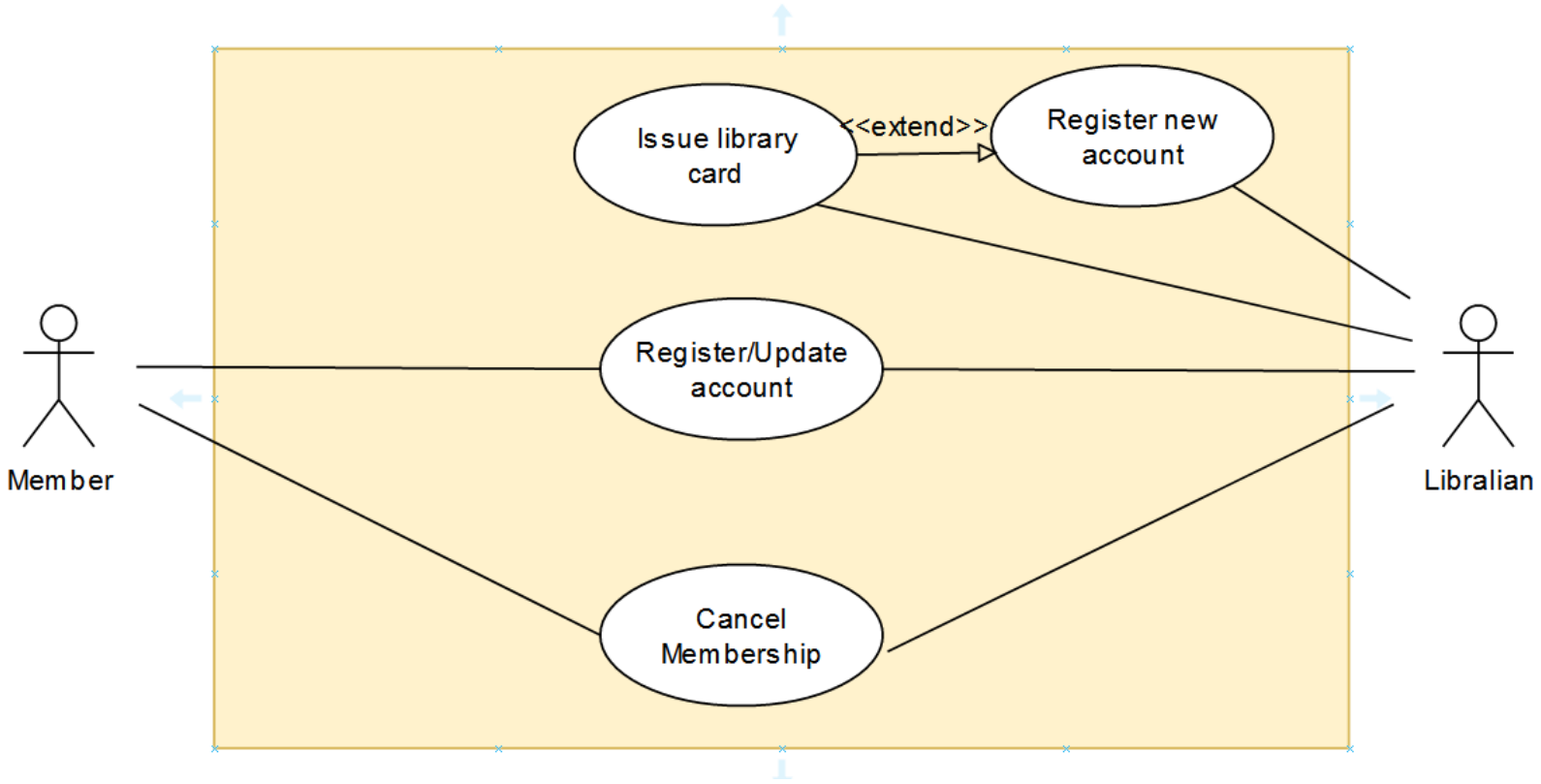
Case Study : ห้องสมุด

- หนังสือ : ในการนำเข้าระบบ จะต้องสามารถเพิ่ม ลบ และ แก้ไขได้
- ในการเพิ่มหนังสือจะมี 2 ขั้นตอน คือ เพิ่มชื่อหนังสือ เช่น “Python Book” และเพิ่มเล่มหนังสือ ซึ่งระบุโดย Item_no เพราะหนังสือ 1 ชื่อ จะมี 1 เล่มขึ้นไป
- สุดท้าย คือ เพิ่มหนังสือเข้าไปใน catalog



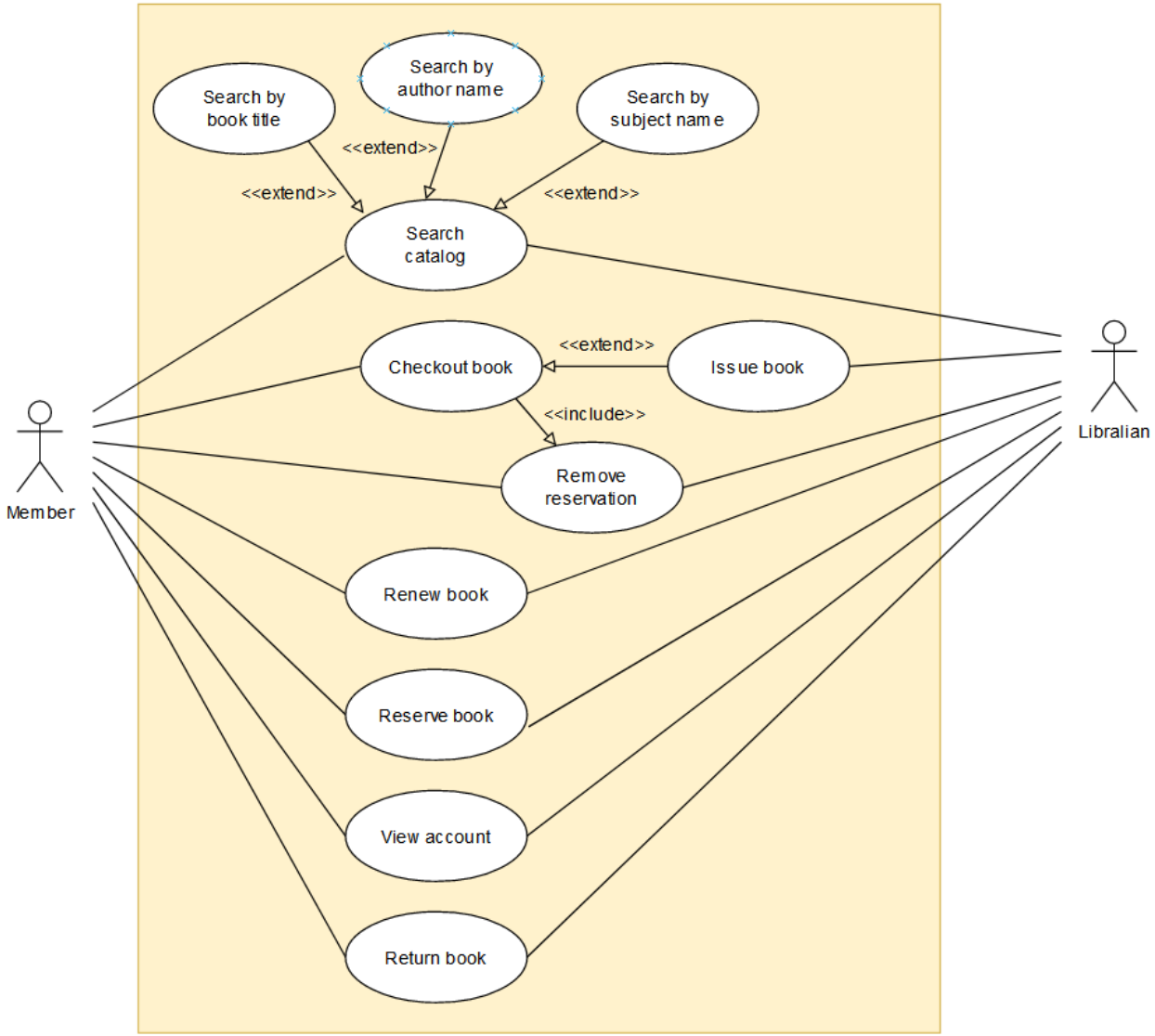
Case Study : ห้องสมุด

- สมาชิก : สามารถสมัครและยกเลิกสมาชิกได้ หรือ แก้ไขข้อมูลได้ (ต้องมี user 2 ฝ่าย)
- การลงทะเบียนผู้ใช้ใหม่ จะกระทำโดยบรรณารักษ์ โดยมีขั้นตอนการออกบัตร (extend)

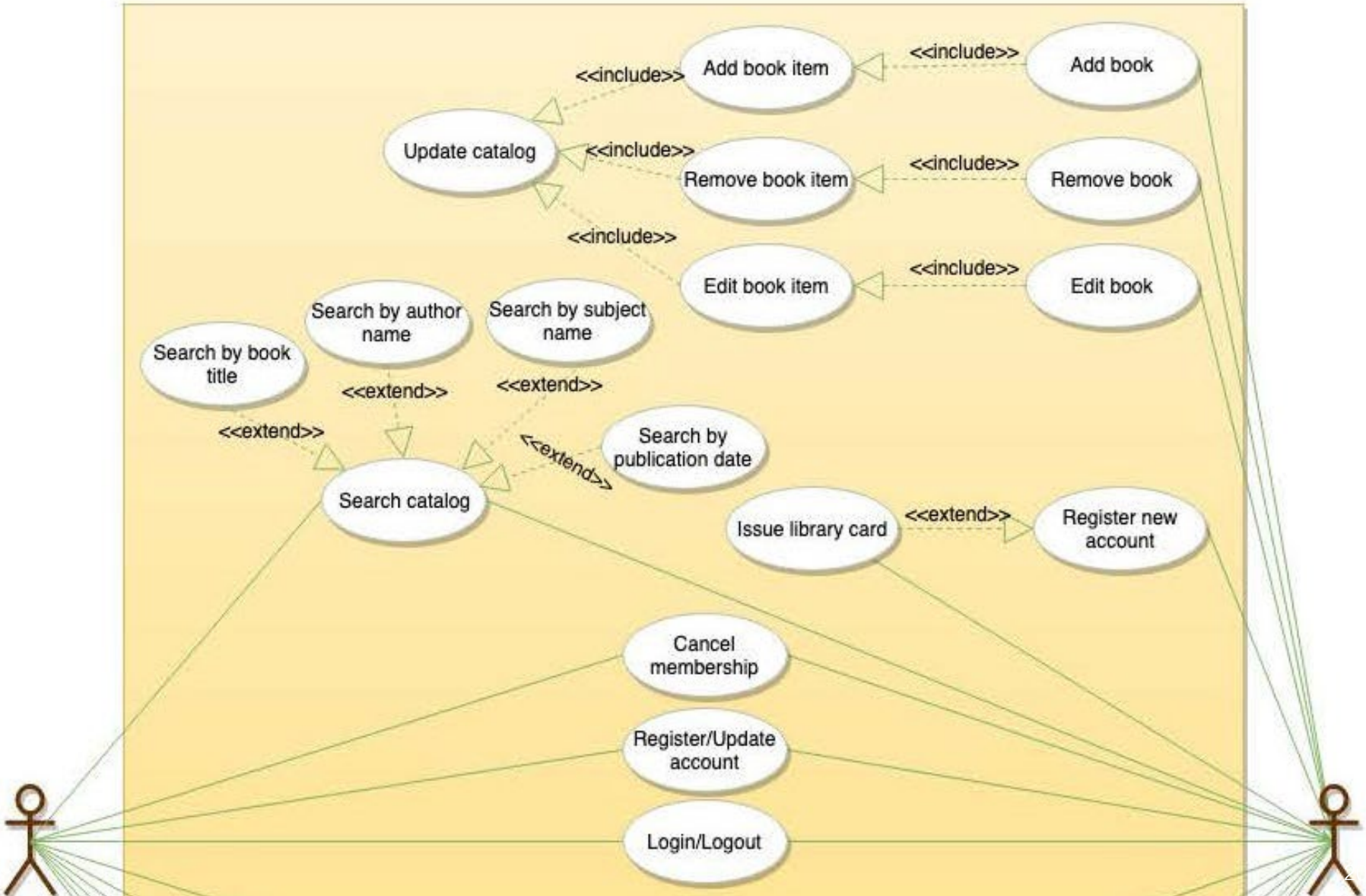


Case Study : ห้องสมุด

- การค้นหา
- การยืมหนังสือ
- การจอง/ยกเลิกจอง
- การต่ออายุ
- การคืนหนังสือ

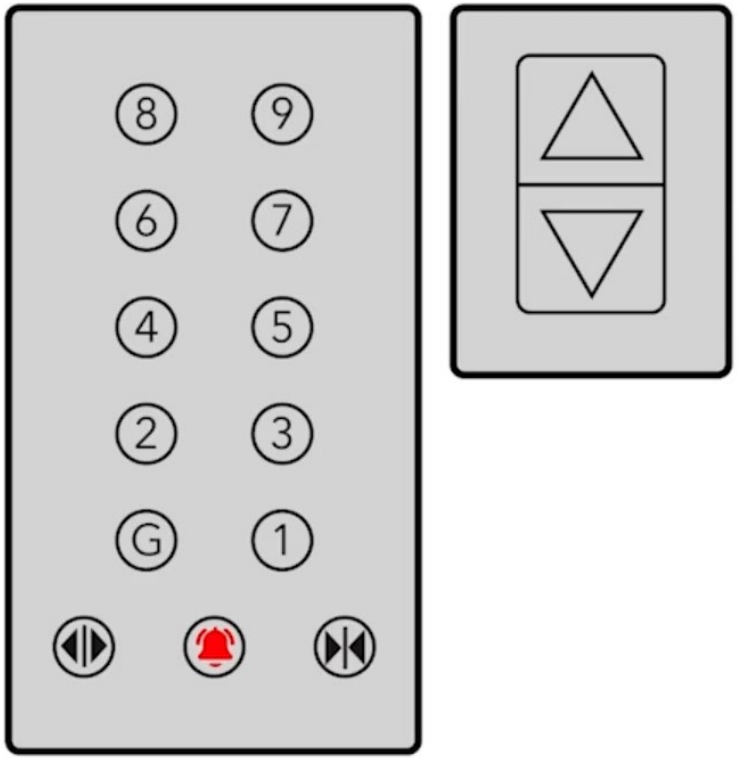


Case Study : ห้องสมุด



Exercise

- ให้นึกถึงลิฟต์ แล้วคิดว่าจะมี Use Case อะไรบ้าง
- ลิฟต์ต้องทำงาน 24/7
- ต้องมีการซ่อมและดูแลรักษา



Exercise

- เรียกลิฟต์ (Call elevator)
- เลือกชั้น (Select the floor)
- เข้าลิฟท์ (Ride elevator)
- ควบคุมประตูลิฟท์ (Operate door)
- ปุ่มฉุกเฉิน (Trigger emergency alarm)
- ตรวจสอบลิฟท์ (Inspect elevator)
- ดูแลรักษา (Service elevator)
- ซ่อมบำรุง (Repair elevator)

Class Diagram

- ส่วนประกอบของ Class Diagram
  - คลาสอาจจะเป็นตัวแทนของ คน สถานที่ เหตุการณ์ หรือสิ่งต่าง ๆ ซึ่งเป็นส่วนประกอบของระบบที่เรากำลังวิเคราะห์และออกแบบอยู่

แอตทริบิวต์ (Attribute)

แอตทริบิวต์คือข้อมูลที่เป็นคุณสมบัติของคลาส คือข้อมูลที่เราสนใจจะจัดเก็บและนำมาใช้ในระบบ

เมธอด (Method)

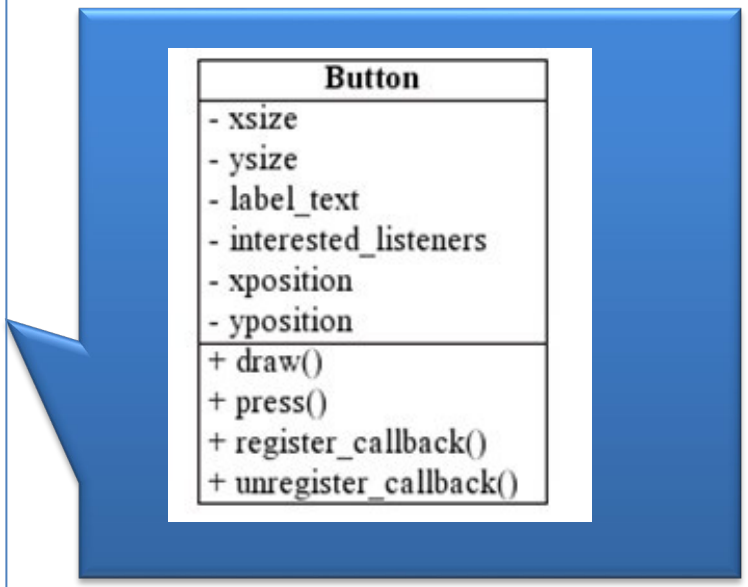
เมธอด คือการทำงานที่คลาสสามารถทำงานได้

ระดับของการเข้าถึงข้อมูล

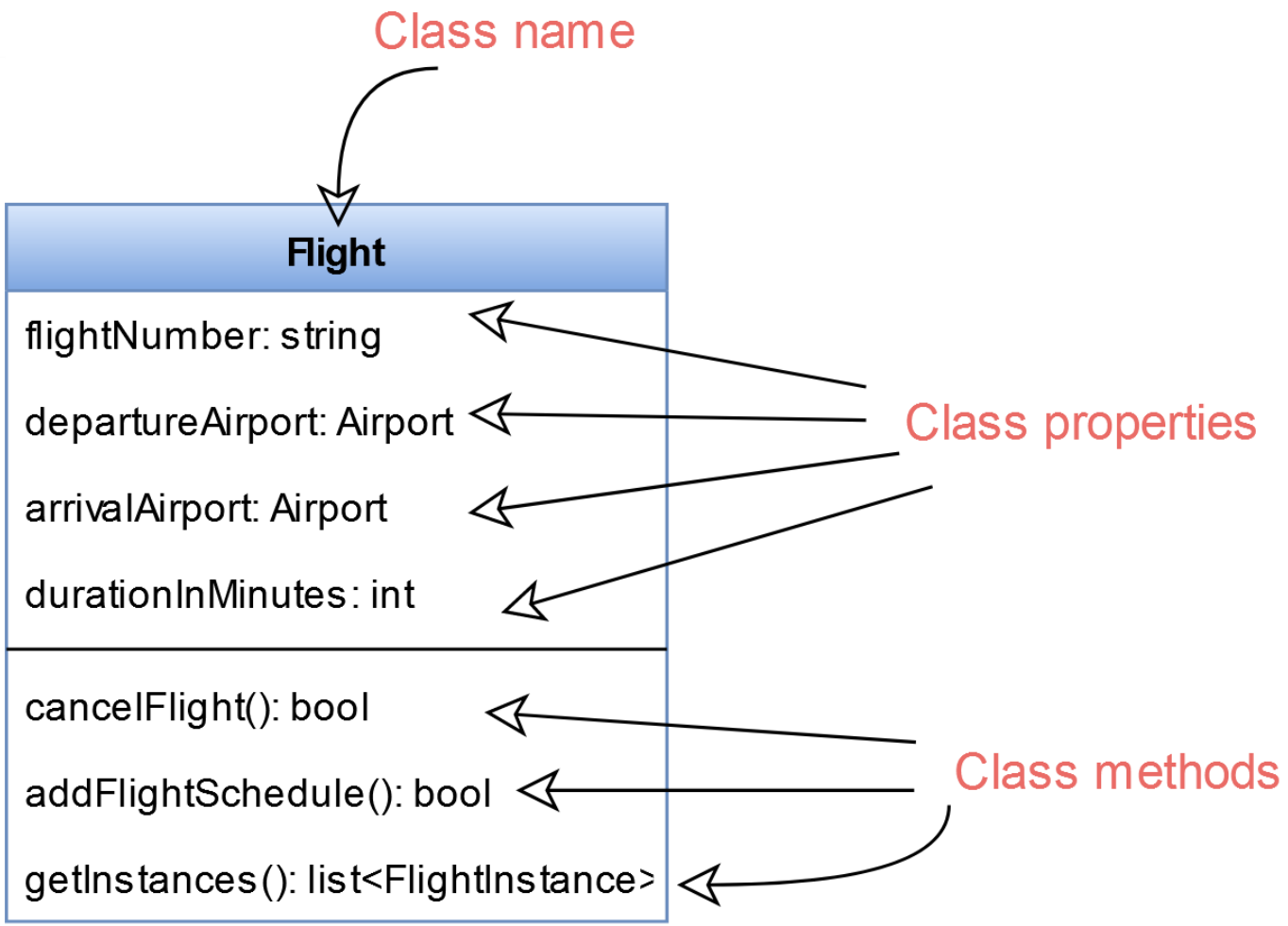
(+) public ให้คลาสอื่น ๆ ใช้งานข้อมูลนี้ได้อิสระ

(#) protected ให้เฉพาะคลาสที่สืบทอดใช้งานได้

(-) private ไม่อนุญาติให้คลาสอื่นใช้งานได้

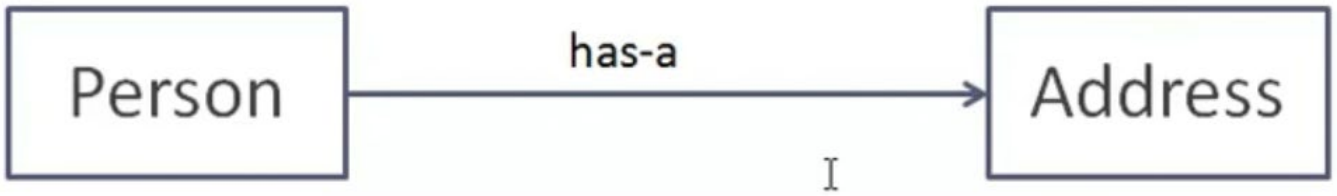


Class Diagram



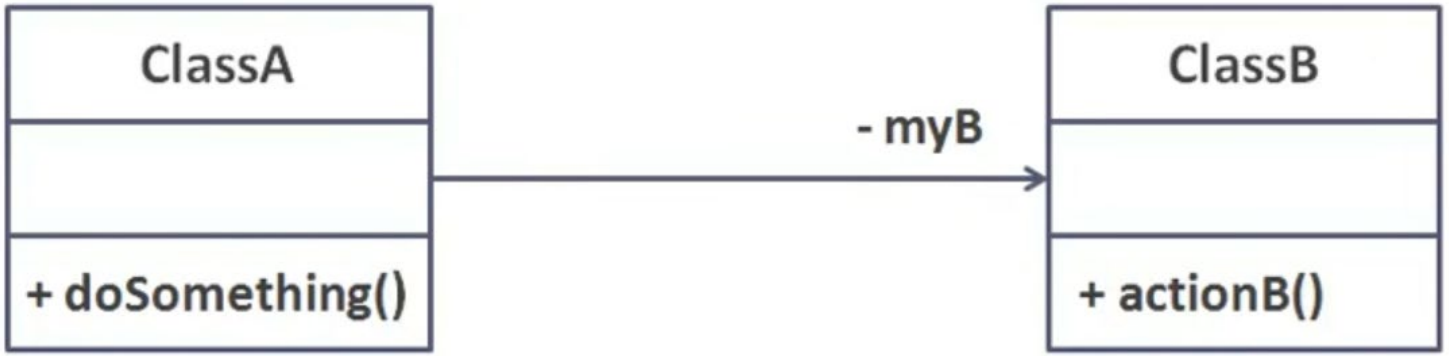
Class Diagram : ความสัมพันธ์

- **Unary Association (ความเกี่ยวข้องทิศทางเดียว) :** มีการอ้างถึง Object แต่มีการอ้างถึงฝั่งเดียว
- หัวลูกศรจะชี้ไปยังคลาสที่ถูกเรียกใช้ (Person รู้จัก Address ที่ตัวเองอยู่ แต่ Address ไม่จำเป็นต้องรู้ว่ามีใครอยู่บ้าง)



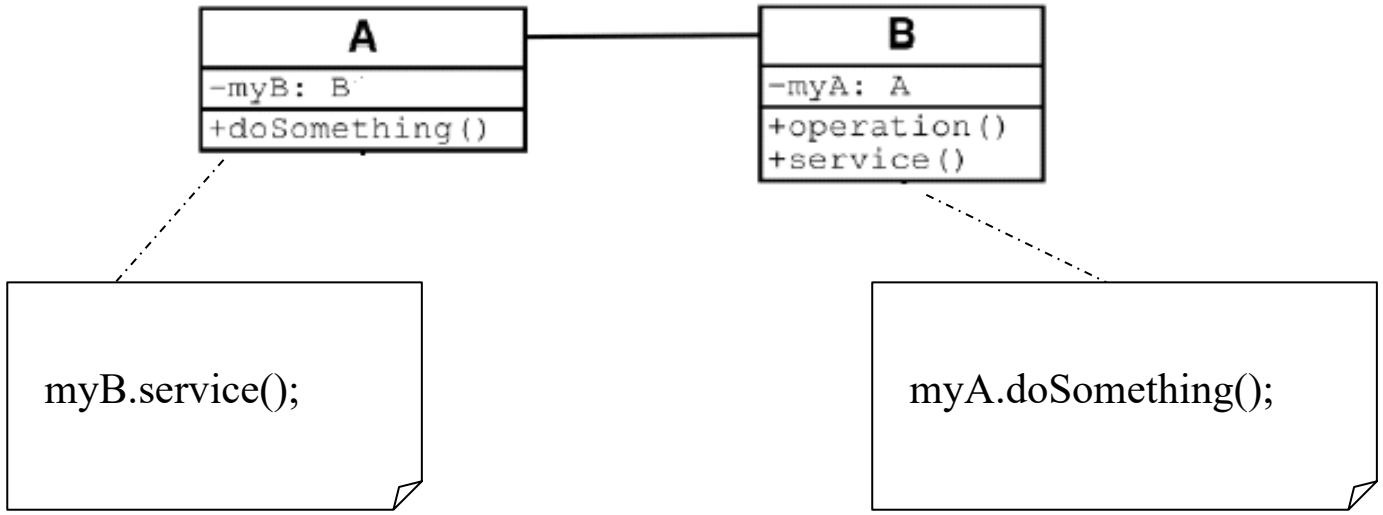
Class Diagram : ความสัมพันธ์

- **Unary Association** อาจเขียนรายละเอียดเพิ่มเติม เช่น จากตัวอย่าง
- ความหมายของ –myB คือ เมื่อมีการเขียนโปรแกรมจะต้องมี attribute ชื่อ myB เป็นส่วนหนึ่งของ ClassA (อาจเขียน myB เป็น attribute เลยก็ได้ แต่ไม่ต้องเขียนที่เส้น
- จึงเป็นความหมายว่า A รู้จัก A แต่ B ไม่รู้จัก A



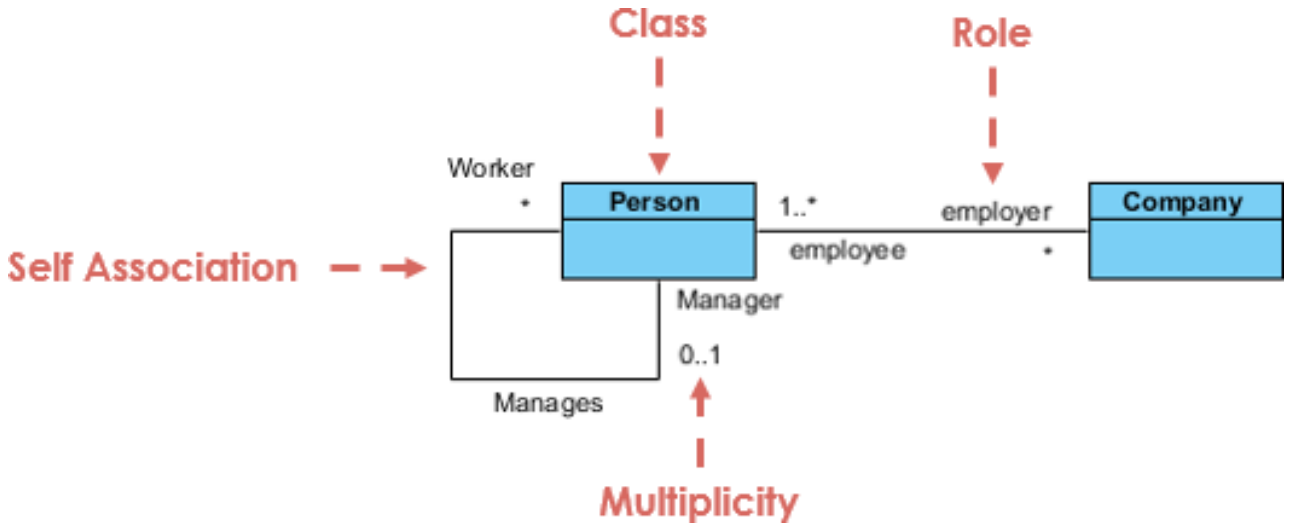
Class Diagram : ความสัมพันธ์

- **Binary Association (ความเกี่ยวข้อง 2 ทิศทาง)** : ก็คล้ายกับ Unary แต่ทั้ง 2 คลาส จะมีการอ้างถึงซึ่งกันและกัน คือ ต่างคนต่างรู้จักกัน จะเห็นว่า myA เป็น attribute ของ B และ myB เป็น attribute ของ A



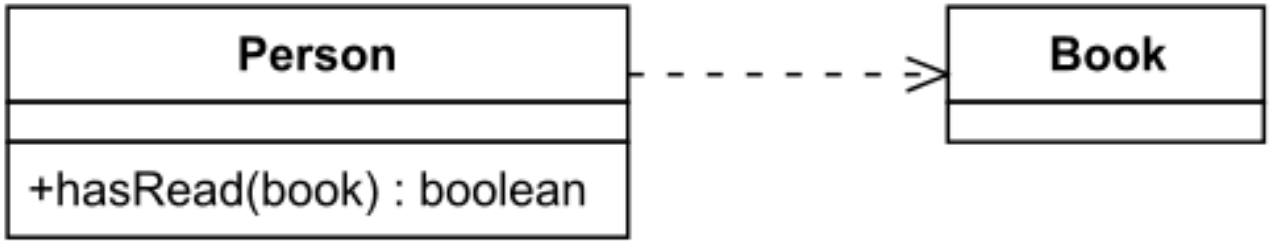
Class Diagram : ความสัมพันธ์

- **Self Association** เป็นความสัมพันธ์ในลักษณะที่ object ใน class มีความสัมพันธ์กับ object ใน class เดียวกัน เช่น หัวหน้ากับลูกน้อง



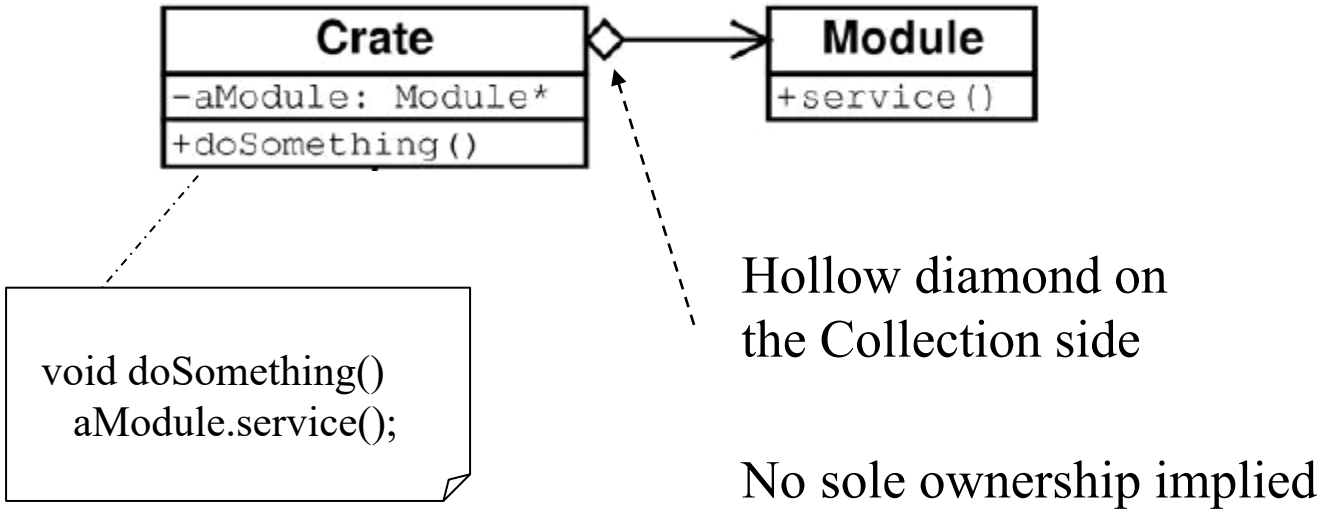
Class Diagram : ความสัมพันธ์

- **Dependency** : เป็นความสัมพันธ์ที่คลาสหนึ่ง ใช้อีกคลาสเป็นพารามิเตอร์ของ method เท่านั้น (เป็นความสัมพันธ์ที่อ่อนกว่า)
- จะแสดงโดยใช้เส้นประ
- จากรูปจะเห็นว่า Book จะเป็นเพียงพารามิเตอร์ของฟังก์ชันเท่านั้น



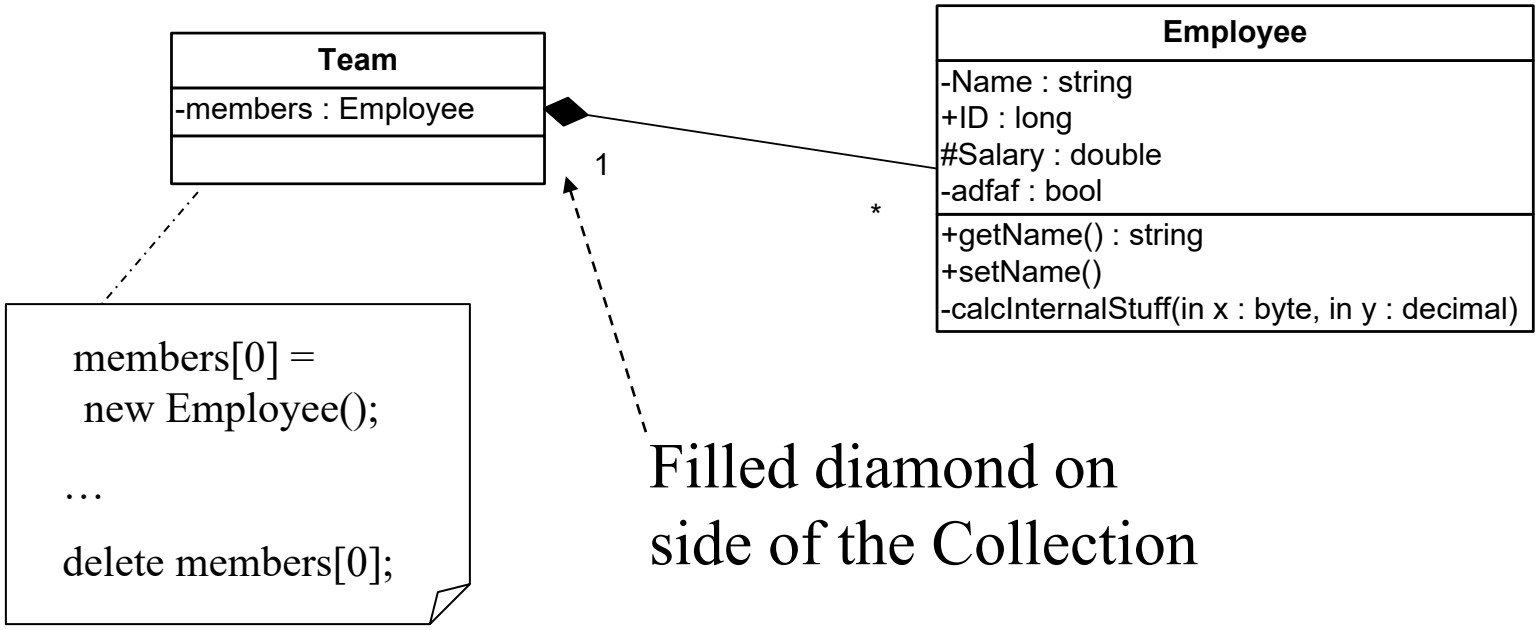
Class Diagram : ความสัมพันธ์

- **Aggregation (การเป็นส่วนหนึ่ง) (has-a)** : เป็นความสัมพันธ์ที่แข็งมากขึ้น ลักษณะสำคัญ โดย Child Object หนึ่งเป็นส่วนหนึ่งของ Parent Object (Whole/Part Relationship) แต่ Object ลูกจะแยกจาก Object แม่ เช่น ความสัมพันธ์ Class กับ Student ที่ Student จะคงอยู่ แม้จะลบ Class ไป
- จะใช้สี่เหลี่ยมขนมเปียกปูนแบบโปร่ง แสดงความสัมพันธ์



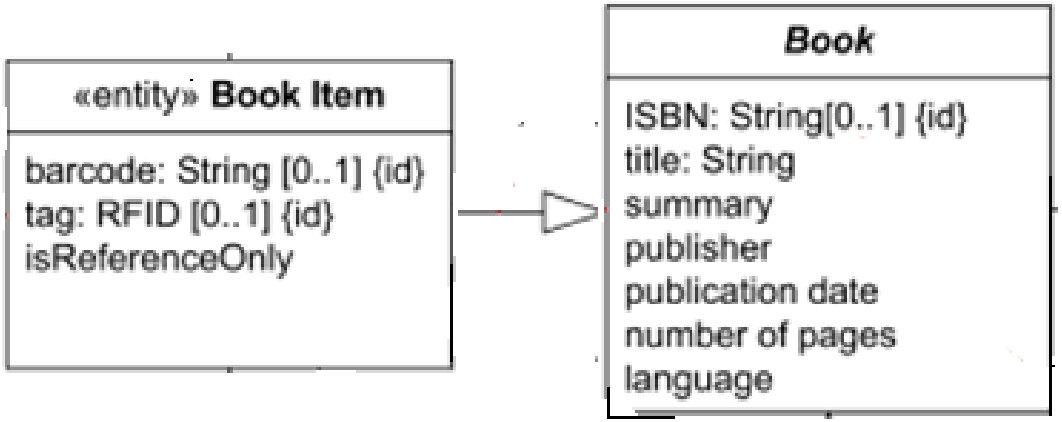
Class Diagram : ความสัมพันธ์

- **Composition (เป็นองค์ประกอบ)** : คล้ายกับ Aggregation แต่ Child จะเป็นส่วนหนึ่งของ Parent ไม่สามารถอยู่เดี่ยวได้ จะอยู่ใน object เดียว เช่น ห้องจะอยู่โดยไม่มีบ้านไม่ได้ ความสัมพันธ์แบบนี้จะมีการสร้าง instance ในอีก object
- จะใช้สี่เหลี่ยมขนมเปียกปูนแบบทึบ แสดงความสัมพันธ์



Class Diagram : ความสัมพันธ์

- **การสืบทอดคุณสมบัติ (Generalization)**
  - โดยคลาสที่เป็นผู้รับการสืบทอดจะมีคุณสมบัติเช่นเดียวกับคลาสที่เป็นผู้ให้การสืบทอด (เหมือนกับ Inheritance)

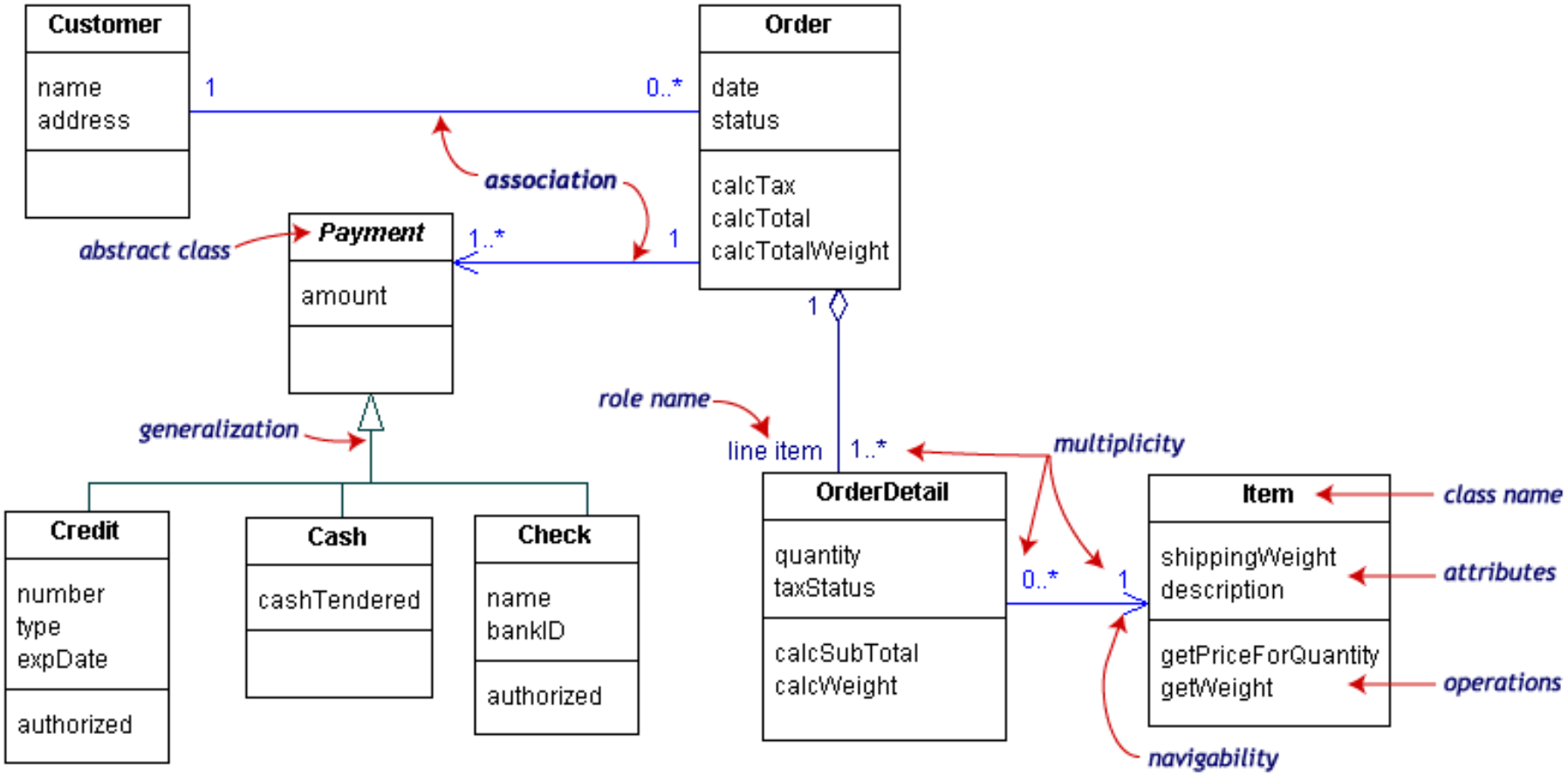


Class Diagram : Multiplicities

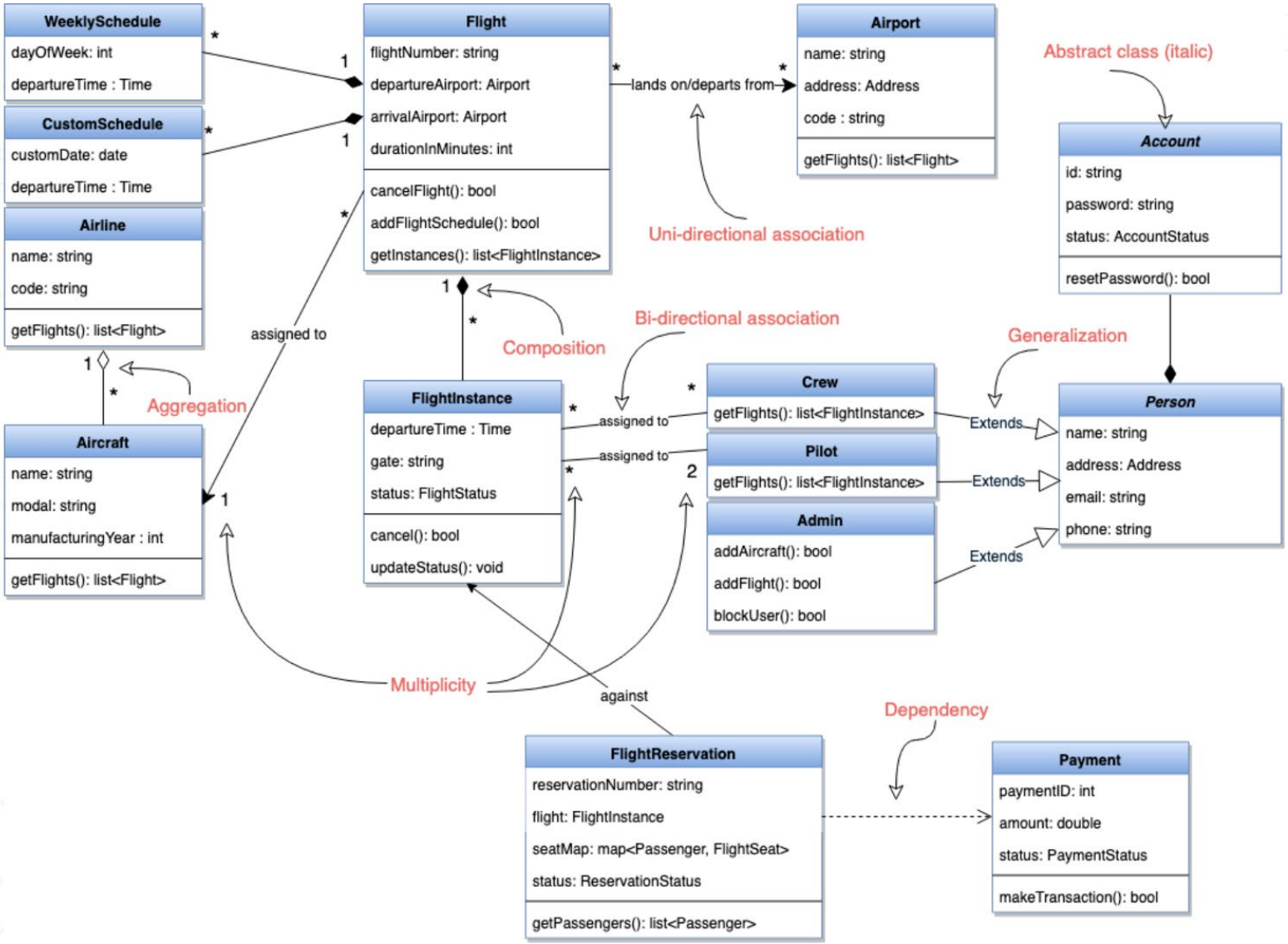
- ใช้สำหรับระบุรายละเอียดของความสัมพันธ์ ในแง่ของจำนวน

Multiplicities	**Meaning**
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

UML Class Example

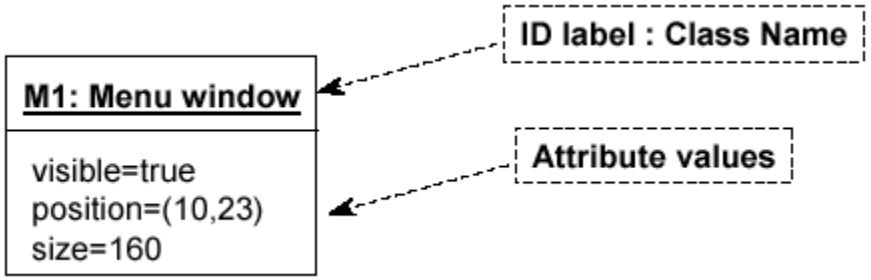


UML Class Example

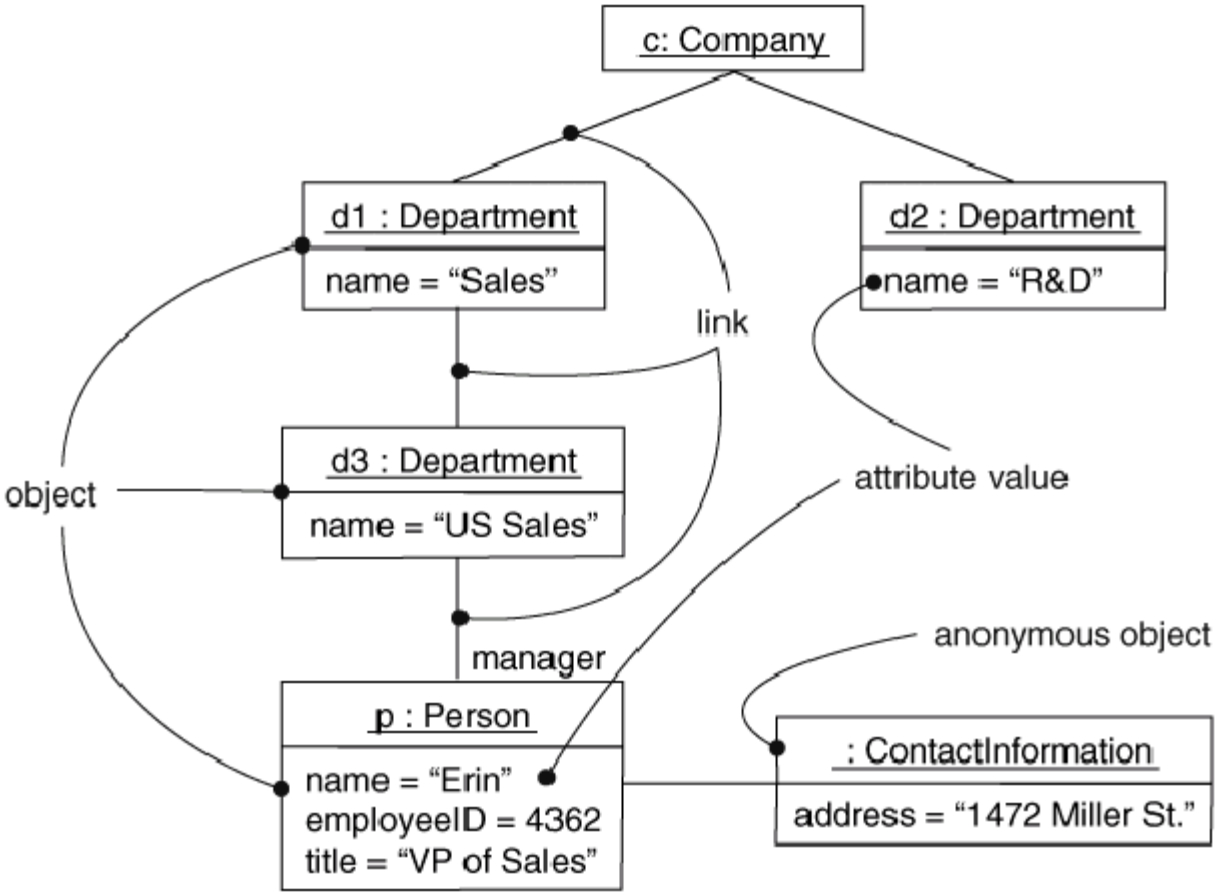


Object Diagrams

- คล้ายกับ Class Diagrams แต่แสดงในรูปแบบของ Instance และความสัมพันธ์ระหว่าง Instance
- Object diagram จะคล้ายกับ snapshot (ภาพถ่าย) แสดงสถานะของ Object ณ เวลาหนึ่ง จุดประสงค์ เพื่อดู
  - การโต้ตอบกัน ที่เป็นข้อมูลจริงๆ ซึ่งจะทำให้เข้าใจมากขึ้น
  - เห็น message ที่รับส่งกัน

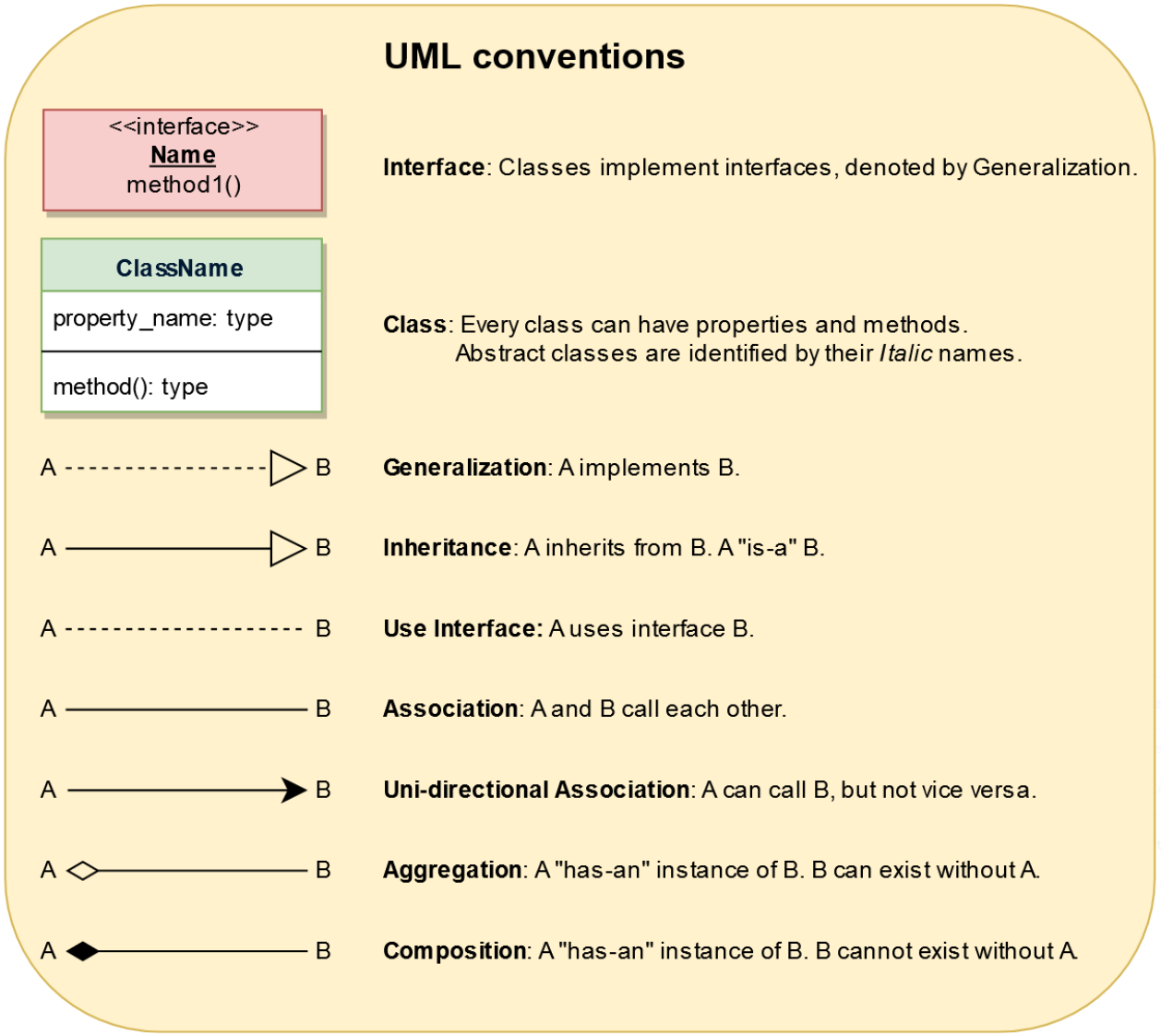


Object Diagrams



Can add association type and also message type

Class Diagram



Case Study : ห้องสมุด

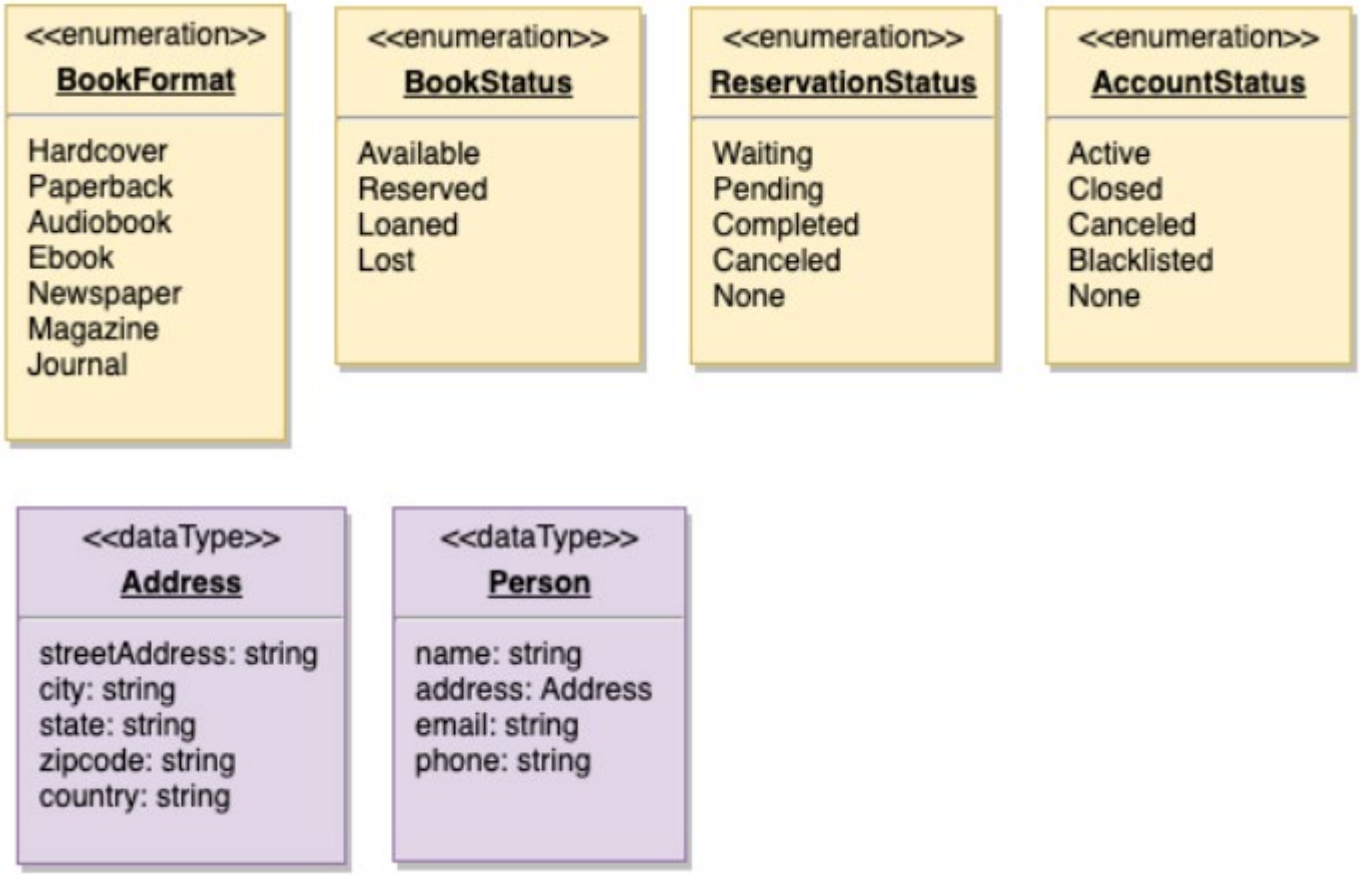
- ระบบห้องสมุด ประกอบด้วยคลาสดังนี้
  - Library: เป็นตัวแทนของห้องสมุด เก็บข้อมูล คือ ชื่อ และ ที่อยู่
  - Book: หมายถึงหนังสือแต่ละรายการ ประกอบด้วย ISBN, Title, Subject, Publishers, etc.
  - BookItem: หนังสือแต่ละรายการอาจมีหลายเล่ม ดังนั้นจึงต้องกำหนดหมายเลขเพื่อให้รู้ว่าเล่มไหน
  - Account: มี 2 ประเภท คือ สมาชิก และบรรณารักษ์
  - LibraryCard: บัตรห้องสมุด ใช้ในการระบุถึงสมาชิกแต่ละคน และใช้ในการยืมคืนหนังสือ
  - BookReservation: รับผิดชอบในการจองหนังสือแต่ละเล่ม

Case Study : ห้องสมุด

- ระบบห้องสมุด ประกอบด้วยคลาสดังนี้
  - BookLending: การให้ยืมหนังสือ
  - Catalog: เป็นรายการของหนังสือที่อยู่ในห้องสมุด ซึ่งสามารถค้นหาได้ Title, Author, Subject
  - Fine: รับผิดชอบในการคำนวณและจัดเก็บค่าปรับ
  - Author: ชื่อผู้เขียน
  - Rack: หนังสือจะจัดเก็บบนชั้นหนังสือ แต่ละ rack จะระบุโดย rack No
  - Notification: รับผิดชอบเรื่องการแจ้งเตือนต่างๆ ในระบบ

Case Study : ห้องสมุด

- Class ประเภท enumerate และ Data Type ประกอบด้วย



Case Study : ห้องสมุด

- โปรแกรมในส่วน Enumerate ใช้ในการแทน status ต่างๆ

```
class BookFormat(Enum):
    HARDCOVER, PAPERBACK, AUDIO_BOOK, EBOOK, NEWSPAPER, MAGAZINE, JOURNAL = 1, 2, 3, 4, 5, 6, 7

class BookStatus(Enum):
    AVAILABLE, RESERVED, LOANED, LOST = 1, 2, 3, 4

class ReservationStatus(Enum):
    WAITING, PENDING, CANCELED, NONE = 1, 2, 3, 4

class AccountStatus(Enum):
    ACTIVE, CLOSED, CANCELED, BLACKLISTED, NONE = 1, 2, 3, 4, 5
```

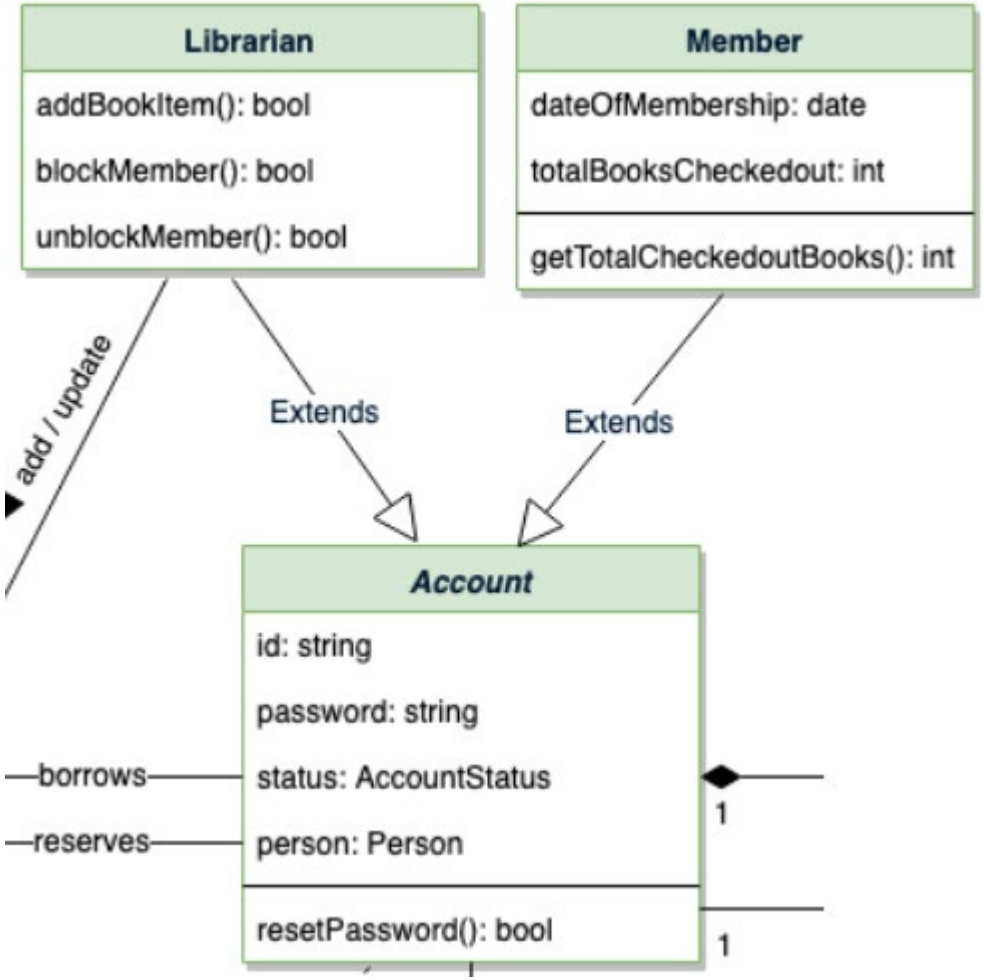
Case Study : ห้องสมุด

- โปรแกรมในส่วน Data Type เพื่อทำหน้าที่เก็บข้อมูลกลุ่มเดียวกันให้เป็นก้อนเดียว

```
class Address:
    def __init__(self, street, city, state, zip_code, country):
        self.__street_address = street
        self.__city = city
        self.__state = state
        self.__zip_code = zip_code
        self.__country = country

class Person(ABC):
    def __init__(self, name, address, email, phone):
        self.__name = name
        self.__address = address
        self.__email = email
        self.__phone = phone
```

Case Study : ห้องสมุด

- Class ประเภท Actor



Case Study : ห้องสมุด

- คลาส Account เป็น Abstract Base Class คือเป็นต้นแบบให้ Inherit ไปยัง Member และ Librarian
- จะเห็นว่า attribute เป็น Private ทั้งหมด, มี method reset_password

```
class Account(ABC):
    def __init__(self, id, password, person, status=AccountStatus.ACTIVE):
        self.__id = id
        self.__password = password
        self.__status = status
        self.__person = person

    def reset_password(self):
        pass
```

Case Study : ห้องสมุด

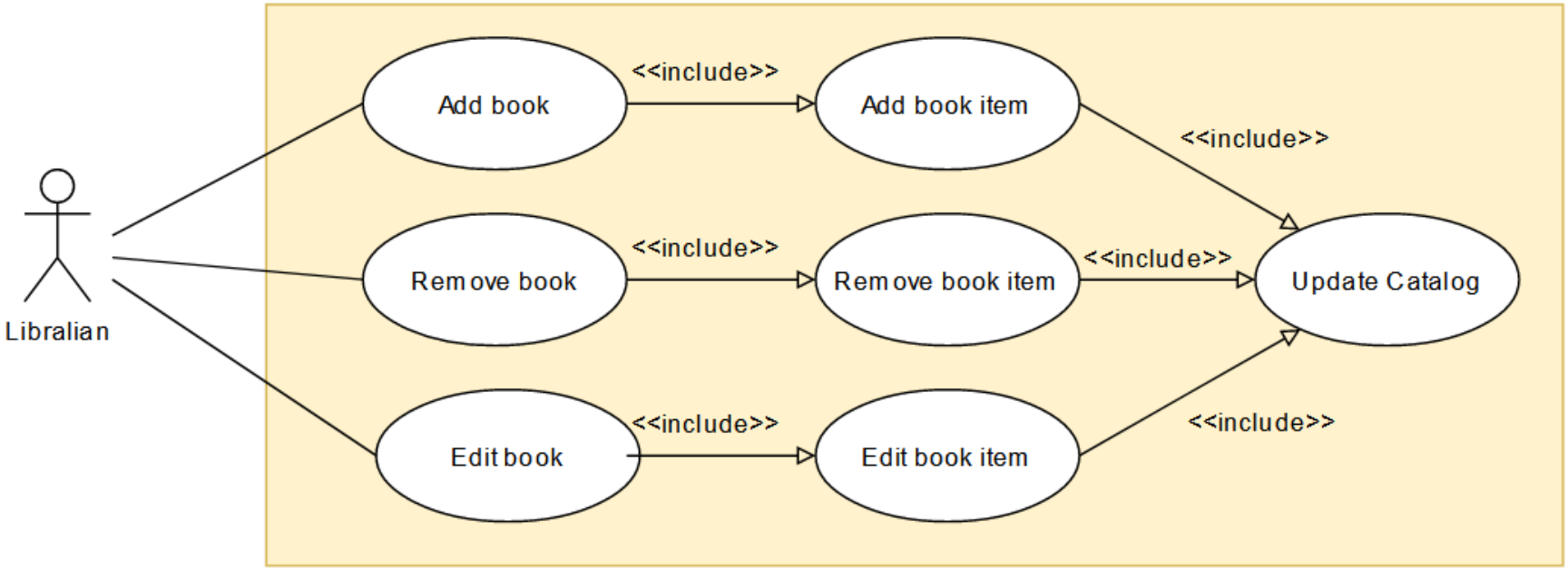
- คลาสบรรณารักษ์ จะสืบทอดมาจากคลาส Account

```
class Librarian(Account):
  def __init__(self, id, password, person, status=AccountStatus.ACTIVE):
    super().__init__(id, password, person, status)

  def add_book_item(self, book_item):
    pass

  def block_member(self, member):
    pass

  def un_block_member(self, member):
    pass
```

Case Study : ห้องสมุด

- ข้อสังเกตของคลาส Librarian คือ ยังขาดฟังก์ชันตาม Use Cased Diagram ไปหลายฟังก์ชัน หากเขียนโปรแกรมจริงต้องทำให้ครบ

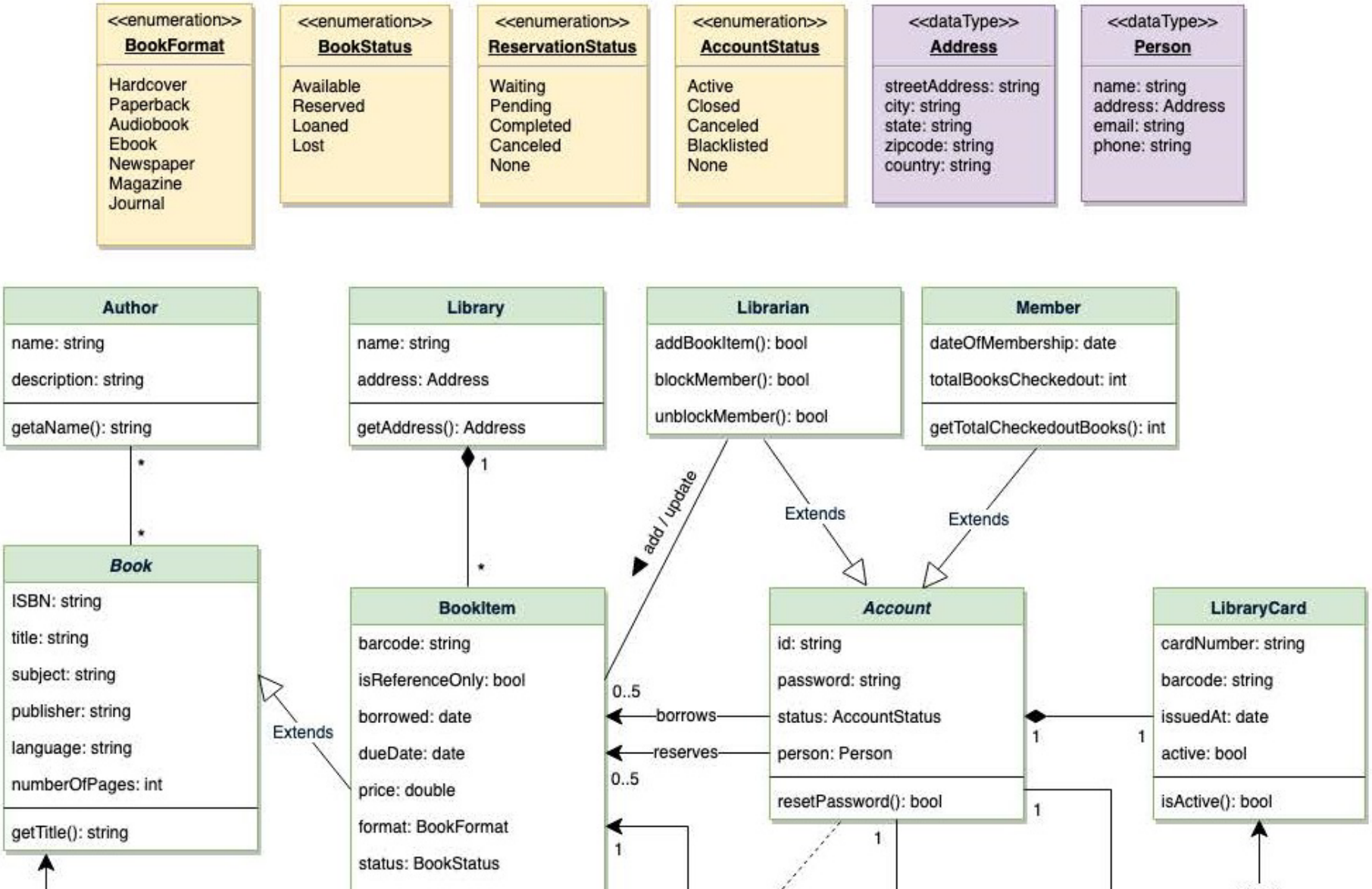
Librarian
addBookItem(): bool
blockMember(): bool
unblockMember(): bool

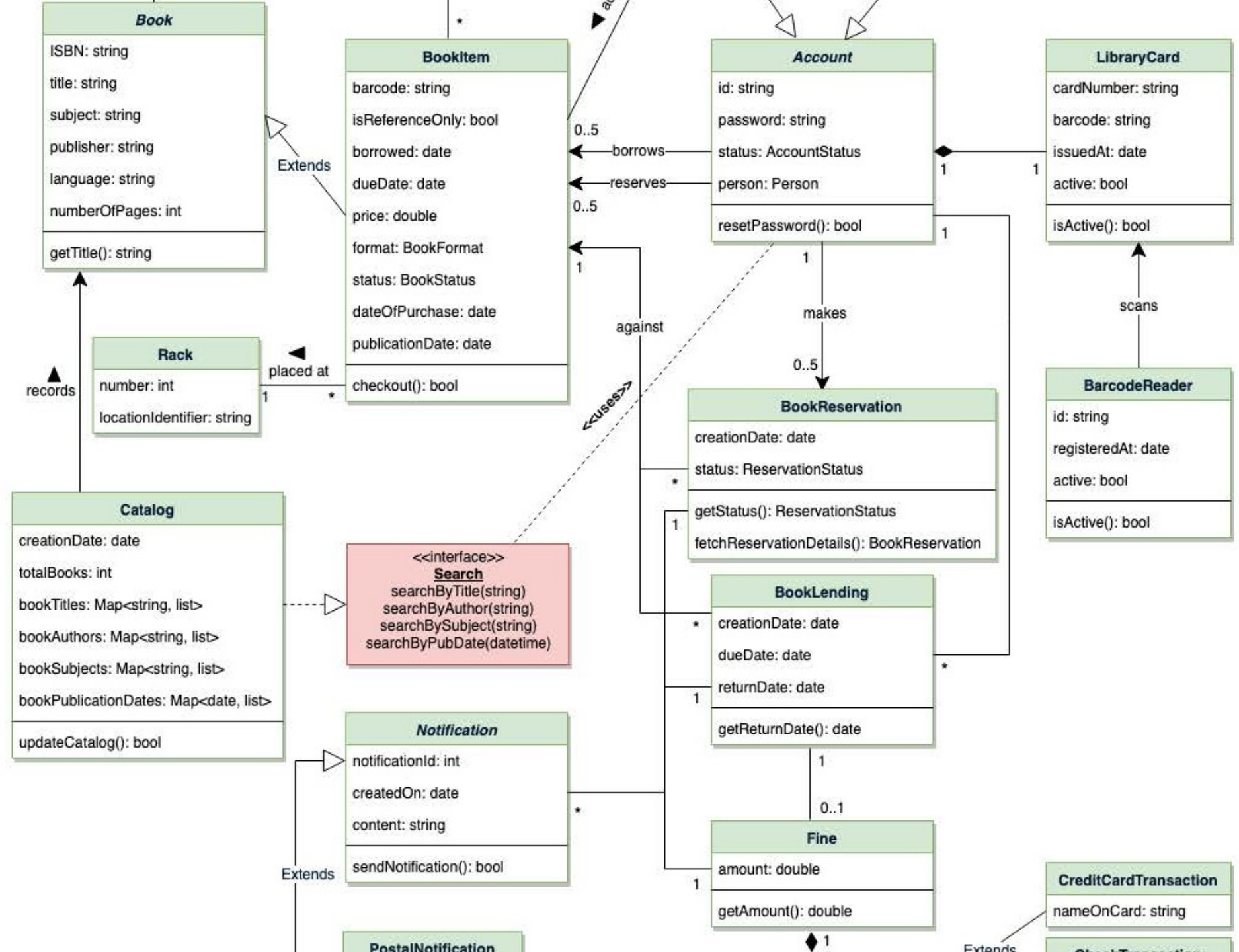


Case Study : ห้องสมุด

- Exercise :
  - ให้เขียน Class Librarian ที่ครบถ้วนให้ครบตาม Use Case Diagram
  - ให้เขียน Class Member ที่ครบถ้วนให้ครบตาม Use Case Diagram ว่ามี method อะไรบ้าง

Case Study : ห้องสมุด




Book
ISBN: string
title: string
subject: string
publisher: string
language: string
numberOfPages: int
getTitle(): string
BookItem
barcode: string
isReferenceOnly: bool
borrowed: date
dueDate: date
price: double
format: BookFormat
status: BookStatus
dateOfPurchase: date
publicationDate: date
checkout(): bool
Account
id: string
password: string
status: AccountStatus
person: Person
resetPassword(): bool
LibraryCard
cardNumber: string
barcode: string
issuedAt: date
active: bool
isActive(): bool
Extends
borrows
reserves
0..5
0..5
1
1
1
1
1
1
makes
0..5
against
scans
Rack
number: int
locationIdentifier: string
placed at
records
1
*
<<uses>>
BookReservation
creationDate: date
status: ReservationStatus
getStatus(): ReservationStatus
fetchReservationDetails(): BookReservation
BarcodeReader
id: string
registeredAt: date
active: bool
isActive(): bool
Catalog
creationDate: date
totalBooks: int
bookTitles: Map<string, list>
bookAuthors: Map<string, list>
bookSubjects: Map<string, list>
bookPublicationDates: Map<date, list>
updateCatalog(): bool
<<interface>>
Search
searchByTitle(string)
searchByAuthor(string)
searchBySubject(string)
searchByPubDate(datetime)
BookLending
creationDate: date
dueDate: date
returnDate: date
getReturnDate(): date
Notification
notificationId: int
createdOn: date
content: string
sendNotification(): bool
Extends
Fine
amount: double
getAmount(): double
CreditCardTransaction
nameOnCard: string
Extends
PostalNotification

Case Study : ห้องสมุด

- คลาส Member
  - total_book_checkedout คือ จำนวนหนังสือที่ยืม

```
class Member(Account):
  def __init__(self, id, password, person, status=AccountStatus.ACTIVE):
    super().__init__(id, password, person, status)
    self.__date_of_membership = datetime.date.today()
    self.__total_books_checkedout = 0

  def get_total_books_checkedout(self):
    return self.__total_books_checkedout

  def reserve_book_item(self, book_item):
    pass
```

Case Study : ห้องสมุด

```
def checkout_book_item(self, book_item):
  if self.get_total_books_checked_out() >= Constants.MAX_BOOKS_ISSUED_TO_A_USER:
    print("The user has already checked-out maximum number of books")
    return False
  book_reservation = BookReservation.fetch_reservation_details(
    book_item.get_barcode())
  if book_reservation != None and book_reservation.get_member_id() != self.get_id():
    # book item has a pending reservation from another user
    print("self book is reserved by another member")
    return False
  elif book_reservation != None:
    # book item has a pending reservation from the give member, update it
    book_reservation.update_status(ReservationStatus.COMPLETED)

  if not book_item.checkout(self.get_id()):
    return False

  self.increment_total_books_checkedout()
  return True
```

Case Study : ห้องสมุด

```
def check_for_fine(self, book_item_barcode):
  book_lending = BookLending.fetch_lending_details(book_item_barcode)
  due_date = book_lending.get_due_date()
  today = datetime.date.today()
  # check if the book has been returned within the due date
  if today > due_date:
    diff = today - due_date
    diff_days = diff.days
    Fine.collect_fine(self.get_member_id(), diff_days)

def return_book_item(self, book_item):
  self.check_for_fine(book_item.get_barcode())
  book_reservation = BookReservation.fetch_reservation_details(
    book_item.get_barcode())
  if book_reservation != None:
    # book item has a pending reservation
    book_item.update_book_item_status(BookStatus.RESERVED)
    book_reservation.send_book_available_notification()
  book_item.update_book_item_status(BookStatus.AVAILABLE)
```

Case Study : ห้องสมุด

```
def renew_book_item(self, book_item):
    self.check_for_fine(book_item.get_barcode())
    book_reservation = BookReservation.fetch_reservation_details(
        book_item.get_barcode())
    # check if self book item has a pending reservation from another member
    if book_reservation != None and book_reservation.get_member_id() != self.get_member_id():
        print("self book is reserved by another member")
        self.decrement_total_books_checkedout()
        book_item.update_book_item_state(BookStatus.RESERVED)
        book_reservation.send_book_available_notification()
        return False
    elif book_reservation != None:
        # book item has a pending reservation from self member
        book_reservation.update_status(ReservationStatus.COMPLETED)
    BookLending.lend_book(book_item.get_bar_code(), self.get_member_id())
    book_item.update_due_date(
        datetime.datetime.now().AddDays(Constants.MAX_LENDING_DAYS))
    return True
```

Case Study : ห้องสมุด

- คลาส Book
  - ใช้เป็น Abstract Class ของ Book Item

```
class Book(ABC):
  def __init__(self, ISBN, title, subject, publisher, language, number_of_pages):
    self.__ISBN = ISBN
    self.__title = title
    self.__subject = subject
    self.__publisher = publisher
    self.__language = language
    self.__number_of_pages = number_of_pages
    self.__authors = []
```

Case Study : ห้องสมุด

- Class BookItem ยังไม่สมบูรณ์ เพราะไม่ได้เรียน __init__ ของ superclass โดยละไว้ เพื่อไม่ให้ยาวเกินไป

```
class BookItem(Book):
    def __init__(self, barcode, is_reference_only, borrowed, due_date, price,
                 book_format, status, date_of_purchase, publication_date, placed_at):
        self.__barcode = barcode
        self.__is_reference_only = is_reference_only
        self.__borrowed = borrowed
        self.__due_date = due_date
        self.__price = price
        self.__format = book_format
        self.__status = status
        self.__date_of_purchase = date_of_purchase
        self.__publication_date = publication_date
        self.__placed_at = placed_at
```

Lab 5

- ให้เลือกไฟล์โจทย์กลุ่มละ 1 โจทย์ (กลุ่มใหญ่) โดยแต่ละโจทย์จะมีเพียง 2 กลุ่มเท่านั้น
- ให้แปลเป็นภาษาไทยให้สละสลวย (จะนำไปใช้กับรุ่นน้องปีหน้า
- จาก Use Case Diagram ให้เขียน Use Case Diagram : Detail Description ตามตัวอย่างใน Slide 9
- ให้เขียน Class Diagram ที่สมบูรณ์ ตาม Use Case และให้บอกว่าเพิ่มตรงไหน หรือ ตัดตรงไหนออก เพราะอะไร
- ให้เขียนโครงของ Class ที่แทน Class ทั้งหมดในระบบ

For your attention