01076105, 01076106

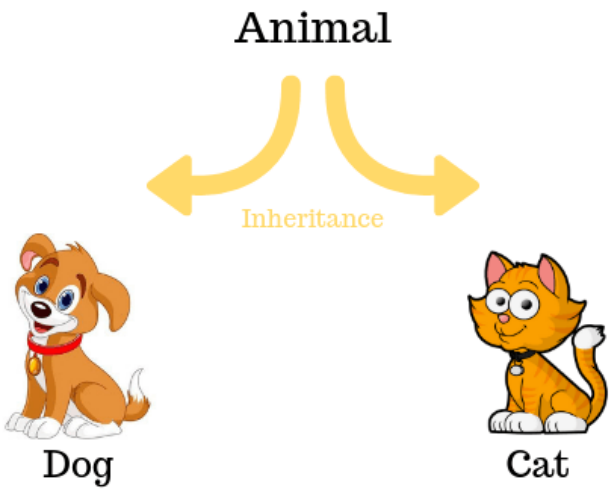
Object Oriented Programming

Object Oriented Programming Project

Inheritance, Use Case Diagram

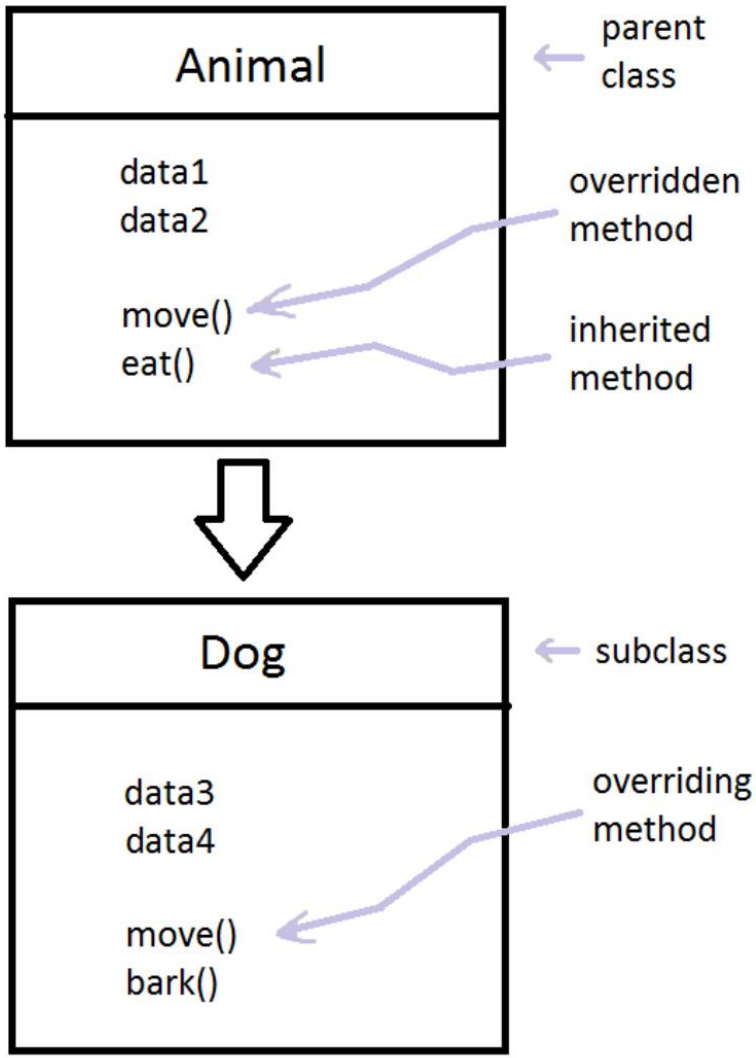
Inheritance

- Inheritance เป็น 1 ใน 4 คุณสมบัติหลักของ Object Oriented Programming
- Inheritance คือ ความสามารถในการสืบทอดคุณสมบัติจาก Class อื่น (เรียก Class ที่สืบทอดว่า Superclass และเรียกตัวเองว่า Subclass (บางครั้งเรียก Parent/Child))
- จากรูป Animal คือ Superclass และ Dog กับ Cat เป็น Subclass



Inheritance

- ประโยชน์ของ Inheritance
  - ลดความซ้ำซ้อนของ Code (หลักการเขียนโปรแกรม คือ เมื่อมี code ที่ซ้ำกันหรือคล้ายกัน ให้หาทางลด)
  - Reuse Code
  - ทำให้ Code อ่านได้ง่ายขึ้น
- จากรูป ถ้าเพิ่มสัตว์ชนิดอื่นๆ ก็จะทำได้ง่าย และกำหนดเฉพาะคุณลักษณะที่เพิ่มเติมเข้ามา



Inheritance

- Class ที่จะ Inherit จาก Class อื่น มีหลักดังนี้
  - ต้องเป็น (“is”) subset ของ Super Class เช่น ถ้า Super Class คือ Car แล้ว Subclass สามารถเป็น Trunk ได้ เพราะรถบรรทุก “เป็น” รถยนต์ประเภทหนึ่ง แต่มอเตอร์ไซค์ เป็น Subclass ไม่ได้
  - Subclass จะต้องมีการกำหนดลักษณะเฉพาะเพิ่มเติม เช่น รถบรรทุก อาจมี นน. บรรทุก พูดโดยรวม คือ Super Class จะมีลักษณะ “ทั่วไป” แต่ Subclass มีลักษณะ “เฉพาะ” เพิ่ม
- Class หนึ่ง อาจ Inherit จากหลาย Class ได้ เรียกว่า Multiple Inheritance (บางภาษาไม่มีคุณลักษณะนี้) และ Class ก็ถูก Inherit จากหลาย Class ได้เช่นกัน

Inheritance

- จากคลาสด้านล่าง จะเห็นว่ามีข้อมูลหลายข้อมูลที่ซ้ำ และเป็นข้อมูลพนักงานเช่นกัน

```
class Programmer:
    salary = 100000
    monthly_bonus = 500

    def __init__(self, name, age, address, phone, programming_languages):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone
        self.programming_languages = programming_languages

class Assistant:
    salary = 100000
    monthly_bonus = 500

    def __init__(self, name, age, address, phone, bilingual):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone
        self.bilingual = bilingual
```

Inheritance

- จะเห็นว่าเมื่อใช้ Inheritance จะทำให้ซ้ำซ้อนน้อยลง และ โครงสร้างดีขึ้น

```
# Superclass
class Employee:
    salary = 100000
    monthly_bonus = 500

    def __init__(self, name, age, address, phone):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone

class Programmer(Employee):
    def __init__(self, name, age, address, phone, programming_languages):
        Employee.__init__(self, name, age, address, phone)
        self.programming_languages = programming_languages

class Assistant(Employee):
    def __init__(self, name, age, address, phone, bilingual):
        Employee.__init__(self, name, age, address, phone)
        self.bilingual = bilingual
```

Inheritance

- รูปแบบการใช้งาน Inheritance

```
class Superclass:
    pass

class Subclass(Superclass)
    pass
```

- เมื่อ Inherit มาจากคลาสใด ให้ใส่วงเล็บต่อท้ายเอาไว้
- เนื่องจากทุกคลาสใน python จะ Inherit มาจากคลาส Object ดังนั้นใน Python เวอร์ชั่นเก่า จะวงเล็บ Object ต่อท้ายหมดทุกคลาสแต่ในเวอร์ชั่นหลังๆ ได้ตัดออกเพื่อให้ดูง่าย

Inheritance

- การ Inheritance มีข้อดีที่สามารถจะเพิ่ม Subclass ที่คล้ายกัน ได้โดย เช่น สมมติว่ามีคลาส Polygon และ Inherit โดยคลาส Triangle หากจะมีการเพิ่มคลาสอื่นๆ เช่น Square ก็ไม่ต้องไปแก้ไข Code ในส่วนคลาส Polygon และ Triangle
- ตัวอย่าง

```
class Polygon:
    pass

class Triangle(Polygon):
    pass
```

- เพิ่ม Class

```
class Ractangle(Polygon):
    pass
```

Inheritance

- มีหลักการออกแบบคลาสข้อหนึ่งมีชื่อว่า Open-Closed Principle
- หลักการข้อนี้มีอยู่ว่า ส่วนประกอบของ Software ควรจะ Close สำหรับการแก้ไข แต่ Open สำหรับการเพิ่มเติม
- หมายความว่าหลังจากที่ Software เขียนเสร็จแล้ว ไม่ควรมีการแก้ไขใดๆ อีก กรณีของ Class คือ ไม่ไปแตะต้องคลาสนั้นอีก กรณีที่มีการเพิ่มเติม ก็ควรใช้วิธีการ Inheritance มากกว่าจะไปแก้ไขที่ Class เดิม
- หลักการข้อนี้ เป็นความพยายามในการหลีกเลี่ยงการแก้ไข Code เดิม โดยหากมีการแก้ไขใดๆ ก็ให้สืบทอดจากคลาส และเพิ่มเติมแทนการแก้ไขคลาสเดิม ทั้งนี้เพื่อให้การดูแลรักษาซอฟต์แวร์สามารถทำได้ง่ายขึ้น

Inheritance

- การ Inheritance มีข้อดีที่สามารถจะเพิ่ม Subclass ที่คล้ายกัน ได้โดย เช่น สมมติว่ามีคลาส Polygon และ Inherit โดยคลาส Triangle หากจะมีการเพิ่มคลาสอื่นๆ เช่น Square ก็ไม่ต้องไปแก้ไข Code ในส่วนคลาส Polygon และ Triangle
- ตัวอย่าง

```
class Polygon:
    pass

class Triangle(Polygon):
    pass
```

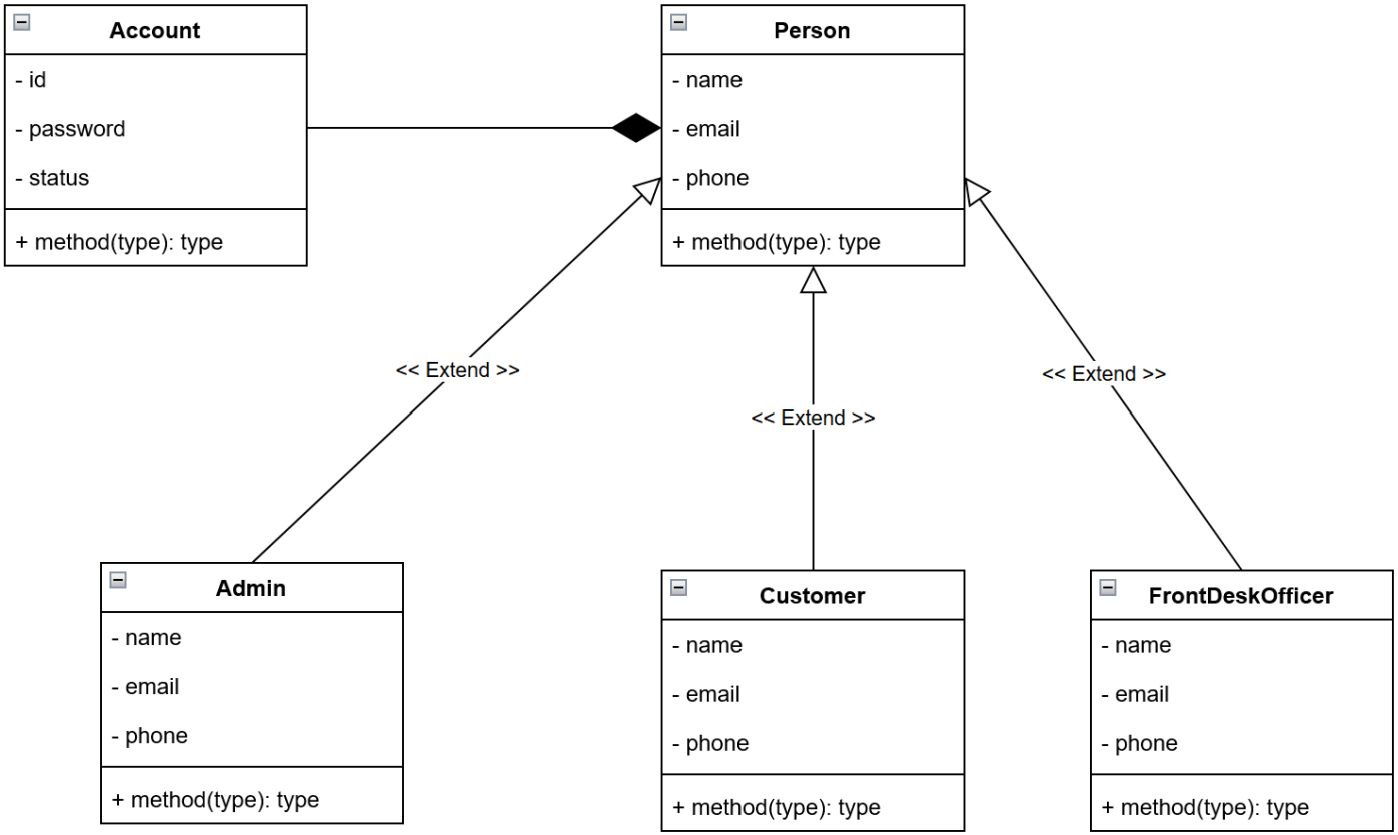
- เพิ่ม Class

```
class Ractangle(Polygon):
    pass
```

- เราสามารถตรวจสอบว่า Class เป็น Subclass ของ Class ใดหรือไม่
- ใช้ฟังก์ชัน issubclass เช่น issubclass(Triangle, Polygon)

Inheritance

- **ตัวอย่าง** ในระบบโรงภาพยนตร์ ส่วนที่สามารถใช้ Inheritance คือ Admin, Customer, Front Desk Officer ซึ่งสามารถ Inherit มาจาก Person



Inheritance

- การแยกคลาส Person ออกมาจะทำให้ความซ้ำซ้อนของข้อมูลลดลง โดยข้อมูลที่เหมือนกันจะอยู่ในคลาส Person และข้อมูลที่ต่างกันจะอยู่ในคลาสเฉพาะของแต่ละประเภทย่อย
- การทำเช่นนี้ มีข้อดี ที่ทำให้การปรับเปลี่ยนในอนาคตสามารถทำได้โดยมีการแก้ไข Code เดิมน้อยลง
- ขอยกตัวอย่าง หากในอนาคตมีการเพิ่มผู้ใช้ประเภทใหม่ขึ้นมา เช่น โรงภาพยนตร์อาจกำหนดผู้ใช้แบบรายเดือน โดยใน 1 เดือนสามารถดูภาพยนตร์ได้ 10 เรื่อง หากใช้วิธี Inherit จะทำให้ไม่ต้องไปแก้ไข Code เดิม โดย Code ที่มีการเพิ่มเติม ก็จะอยู่ในคลาสที่สร้างเพิ่มเติมขึ้นมาใหม่

Inheritance

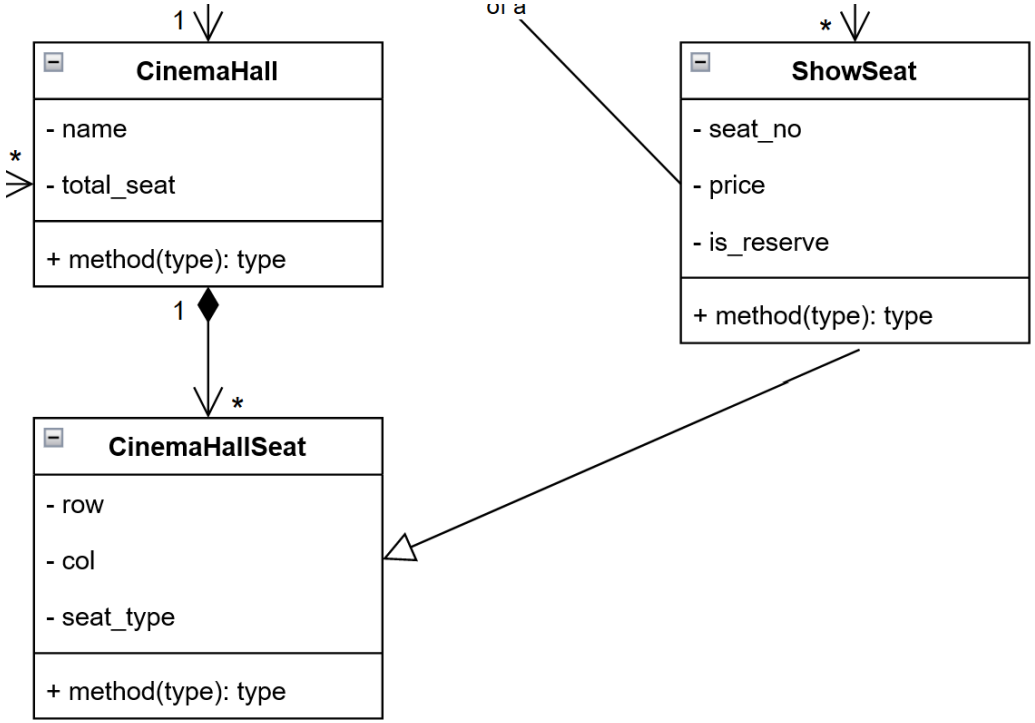
- ข้อมูลอีกส่วนในระบบโรงภาพยนตร์ที่ใช้กับ Inheritance ได้ คือ คลาส CinemaHallSeat กับ ShowSeat ซึ่งเป็นคลาสที่นั่งเหมือนกัน แต่คลาสหนึ่งเป็นที่นั่งของโรง แต่อีกคลาสเป็นที่นั่งที่มีการจอง
- ซึ่งไม่ควรจะมี 2 คลาสที่มีข้อมูลเดียวกัน

```
class CinemaHallSeat:
    def __init__(self, seat_row, seat_col, seat_type):
        self.seat_row = seat_row
        self.seat_col = seat_col
        self.seat_type = seat_type
```

```
class ShowSeat:
    def __init__(self, seat_no, price):
        self.seat_no = seat_no
        self.price = price
        self.is_reserve = None
```

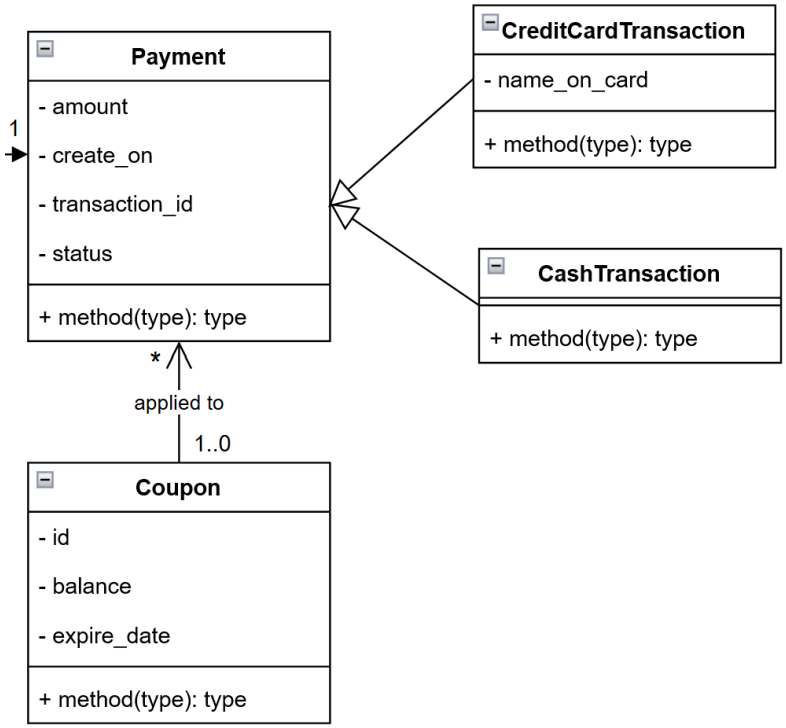
Inheritance

- แต่หากจะรวมเป็นข้อมูลคลาสเดียวกัน จะขัดแย้งในเรื่องของ cohesion เพราะข้อมูลการจอง ไม่ได้มีความเกี่ยวข้องกับที่นั่ง
- ดังนั้นในกรณีนี้ การใช้ Inheritance จะมีความเหมาะสมมากกว่า ดังนี้



Inheritance

- ข้อมูลอีกส่วนหนึ่งที่ควรใช้ Inheritance คือ ข้อมูลการชำระเงิน ซึ่งมีวิธีชำระเงินหลายวิธี แต่ละวิธีก็มีรายละเอียดการทำงานในแบบของตัวเอง
- หากมีช่องทางการชำระเงินวิธีอื่นๆ เพิ่มในอนาคตก็เพียงแต่เพิ่ม Inherit Class เข้าไป



Inheritance

- คลาสที่ Inherit มากจากคลาสอื่น สามารถจะใช้ Attribute

```
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):
    pass


my_triangle = Triangle(3, "Blue")

print(my_triangle.num_sides)
print(my_triangle.color)
```

Inheritance

- แต่ในกรณีที่ Subclass มี __init__ ของตนเอง จะไม่ Inherit จาก Super Class โดยอัตโนมัติ

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):

    def __init__(self, base, height):
        self.base = base
        self.height = height
```

```python
my_triangle = Triangle(3, "Blue")

print(my_triangle.num_sides)    # Error
print(my_triangle.color)        # Error
```

Inheritance

- วิธีการเขียนกรณีมี __init__ ของตนเอง

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):

    NUM_SIDES = 3

    def __init__(self, base, height, color):
        Polygon.__init__(self, Triangle.NUM_SIDES, color)
        self.base = base
        self.height = height
```

Inheritance : Quiz

- ให้ list Instance Attribute ทั้งหมด ของ Enemy Class

```
1  class Sprite:
2
3      def __init__(self, x, y, speed, direction):
4          self.x = x
5          self.y= y
6          self.speed = speed
7          self.direction = direction
8
9
10 class Enemy(Sprite):
11
12     def __init__(self, x, y, speed, direction, num_lives):
13         Sprite.__init__(self, x, y, speed, direction)
14           self.num_lives = lives
```

Inheritance : Quiz

- ใน Instance ของ Puppy Class จะมี Attribute อะไรบ้าง

```
1  class Dog(object):
2
3      def __init__(self, name, age, breed):
4          self.name = name
5          self.age = age
6          self.breed = breed
7
8  class Puppy(Dog):
9
10     def __init__(self, is_vaccinated):
11         self.is_vaccinated = is_vaccinated
```

Inheritance

- เราสามารถใช้ super() ในการแทน Super Class ที่ขึ้นไป 1 ชั้น (ไม่มี self)

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):

    NUM_SIDES = 3

    def __init__(self, base, height, color):
        super().__init__(Triangle.NUM_SIDES, color)
        self.base = base
        self.height = height
```

Inheritance

- ตัวอย่าง

```python
class Employee:

    def __init__(self, full_name, salary):
        self.full_name = full_name
        self.salary = salary


class Programmer(Employee):

    def __init__(self, full_name, salary, programming_language):
        super().__init__(full_name, salary)
        self.programming_language = programming_language
```

Inheritance : exercise

- จาก Class Mammal ที่กำหนดให้ ให้สร้าง Class Panda ให้เพิ่มข้อมูลดังนี้
  - Class Attribute is_dangered = True
  - Instance Attribute code
- ทดลองสร้าง Instance my_panda แล้วทดสอบ

```
class Mammal:
    def __init__(self, name, age, health, num_offspring, years_in_captivity):
        self.name = name
        self.age = age
        self.health = health
        self.num_offspring = num_offspring
        self.years_in_captivity = years_in_captivity
```

Use Case Diagram

- ในการพัฒนาซอฟต์แวร์ งานสำคัญอย่างหนึ่ง คือ การหาความต้องการของซอฟต์แวร์ ความต้องการของซอฟต์แวร์ คือ สิ่งที่แสดงให้เห็นว่าซอฟต์แวร์ที่จะพัฒนาขึ้น จะต้องทำอะไรได้บ้าง หรือ มีความสามารถใดบ้าง
- ความต้องการของระบบ จะแบ่งออกเป็น 2 อย่าง คือ
  - Functional Requirement เป็นสิ่งที่ระบุว่าซอฟต์แวร์ต้องมี หรือ ต้องทำได้ เช่น ซอฟต์แวร์โรงภาพยนตร์ จะต้องค้นหารอบฉายภาพยนตร์ และ จองตั๋วได้
  - Non-Functional Requirement เป็นสิ่งที่ระบุว่าซอฟต์แวร์ควรจะมี แต่สิ่งนั้นไม่ได้เป็น Feature ของโปรแกรมโดยตรง เช่น ความเร็วของการโหลด การจองได้ภายใน 3 คลิ๊ก หรือ ต้องรันใน Browser อะไรได้บ้างเป็นต้น

Use Case Diagram

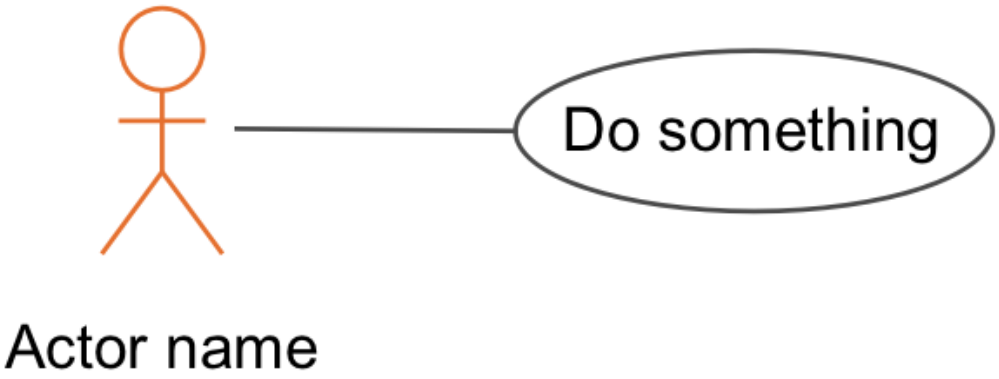
- **ตัวอย่าง Requirement แบบตัวอักษร (บางส่วน)**
  - แต่ละโรงภาพยนตร์จะมีหลายโรงย่อย (Hall) โดยโรงย่อย จะมีที่นั่งได้หลายแบบ โดยที่นั่งจะวางเป็น row และ column
  - แต่ละโรงจะฉายภาพยนตร์ได้ช่วงเวลาละ 1 เรื่องในเวลาหนึ่ง แต่ภาพยนตร์ 1 เรื่อง อาจฉายในหลายโรงได้
  - ภาพยนตร์จะนำเข้าโดยเจ้าหน้าที่ (Admin) โดยจะแสดงที่ catalog ของเว็บ โดยบอกวันที่เริ่มฉาย และหากถูกถอดจากการฉายจะไม่แสดงที่ catalog
  - ลูกค้าสามารถค้นหาภาพยนต์ได้จาก ชื่อเรื่อง ชื่อโรงภาพยนตร์
  - เมื่อลูกค้าเลือกภาพยนตร์ ระบบจะแสดงโรงภาพยนตร์ที่ฉายเรื่องนั้น และ รอบฉายที่มี

Use Case Diagram

- **ตัวอย่าง Requirement แบบตัวอักษร (บางส่วน)**
  - ลูกค้าสามารถจะเลือกรอบฉายในโรงที่ต้องการ และ จองตั๋วได้
  - ระบบจะแสดงการจัดที่นั่งให้กับลูกค้า ลูกค้าสามารถเลือกที่นั่งที่ต้องการ โดยอาจเลือกหลายที่นั่งก็ได้
  - ระบบจะต้องแสดงให้ลูกค้าเห็นว่าที่นั่งใดที่จองแล้ว และ ที่นั่งใดยังว่างอยู่
  - ลูกค้าสามารถชำระเงินได้ผ่านบัตรเครดิตและ shopeepay
  - ระบบจะต้องป้องกันไม่ให้ลูกค้า 2 รายจองที่นั่งเดียวกัน
  - ลูกค้าสามารถเลือก Promotion สำหรับการชำระเงิน เพิ่มเติมได้

Use Case Diagram

- นอกเหนือจากการเขียน Requirement แล้วยังสามารถแสดง Requirement ได้โดยใช้ Use Case Diagram
- สำหรับ Use Case Diagram มักจะมีองค์ประกอบเบื้องต้น 2 ส่วน คือ
  - Actor ซึ่งหมายถึง ผู้ใช้แต่ละกลุ่ม
  - Use Case หมายถึง การทำงานที่ผู้ใช้สามารถทำได้ มักใช้เป็นคำกริยา



Use Case Diagram

- Use Case Diagram เป็น Diagram สำหรับบอกว่าระบบทำอะไรได้บ้าง หรือ ความต้องการของผู้ใช้มีอะไรบ้าง
- และยังบอกว่าผู้ใช้ของระบบแบ่งออกเป็นกี่กลุ่ม ผู้ใช้แต่ละกลุ่ม **สามารถ** ทำอะไรได้บ้าง
- นอกจากนั้นยังบอกความสัมพันธ์ของการทำงาน ว่าในแต่ละการทำงาน มีการขึ้นต่อกันอย่างไร เช่น การกำหนดที่นั่งจะต้องอยู่ในกระบวนการจองตั๋ว เพื่อให้ Developer สามารถเข้าใจรูปแบบการทำงาน
- Use Case Diagram ยังเป็นเครื่องมือที่ดี สำหรับสื่อสารกับผู้ใช้ เพราะแสดงเป็นรูปภาพ ทำให้เข้าใจได้ง่ายกว่า

Use Case Diagram

- ขั้นตอนแรกของการทำ Use Case Diagram คือการค้นหา Actor ซึ่งคือ บุคคลที่มีความเกี่ยวข้องกับระบบ โดยระบบโรงภาพยนต์มี Actor ดังต่อไปนี้
  - Admin: รับผิดชอบในการเพิ่มภาพยนต์และรอบฉาย การยกเลิกรอบฉาย การจัดการกับผู้ใช้ เช่น การล็อก User ที่มีปัญหาและการปลดล็อก
  - FrontDeskOfficer: เจ้าหน้าที่ขายตั๋ว ทำการจองตั๋ว ยกเลิกตั๋ว
  - Customer: สามารถดูรอบฉาย จองตั๋ว หรือยกเลิกการจองได้
  - Guest: สามารถค้นหาภาพยนตร์ รอบหนัง แต่หากจะจองตั๋วต้องสมัครสมาชิก
  - System: ส่งการเตือนต่างๆ ไปยังสมาชิก

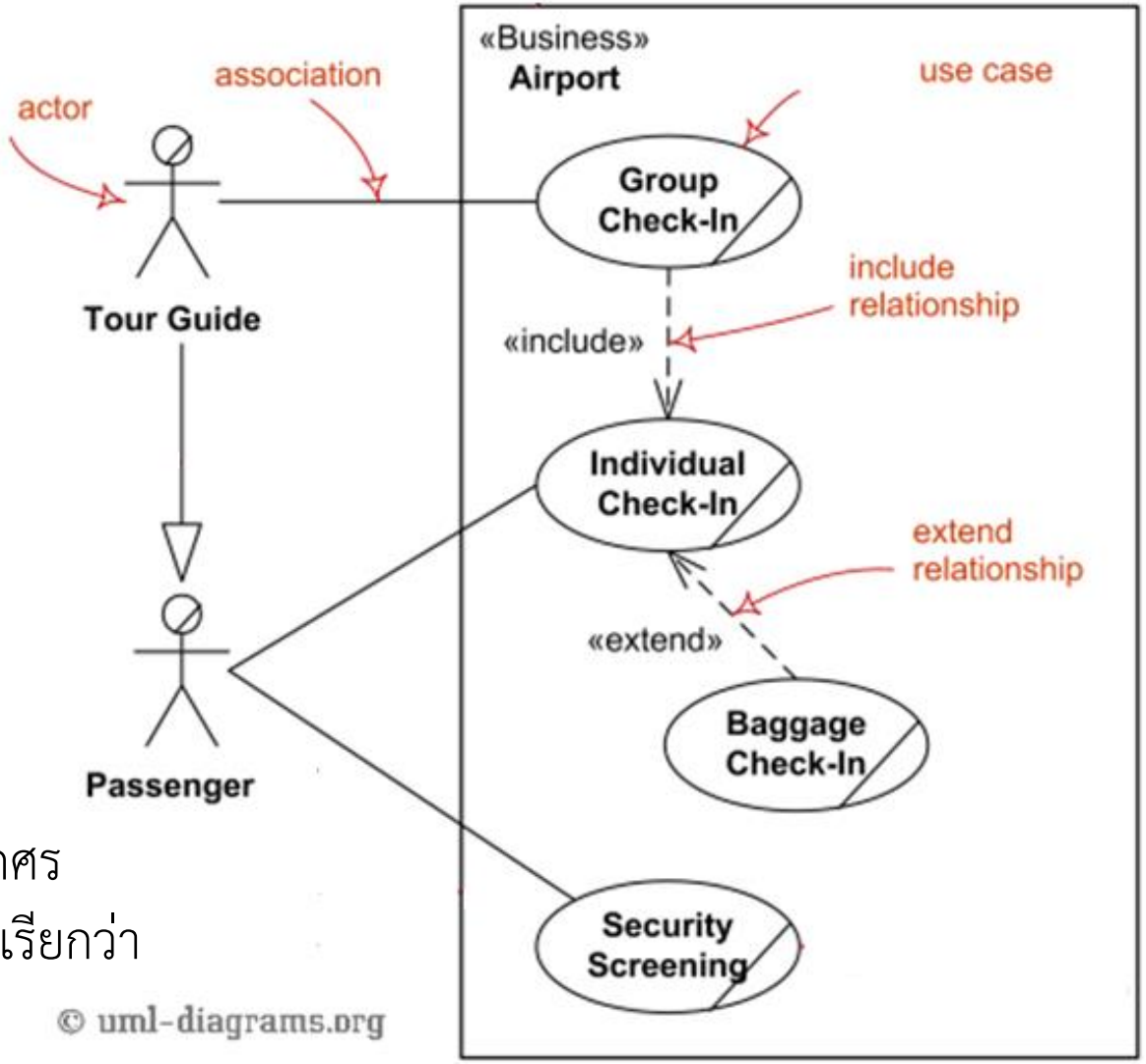
Use Case Diagram

สัญลักษณ์

- Use case
- Actor
- Connection

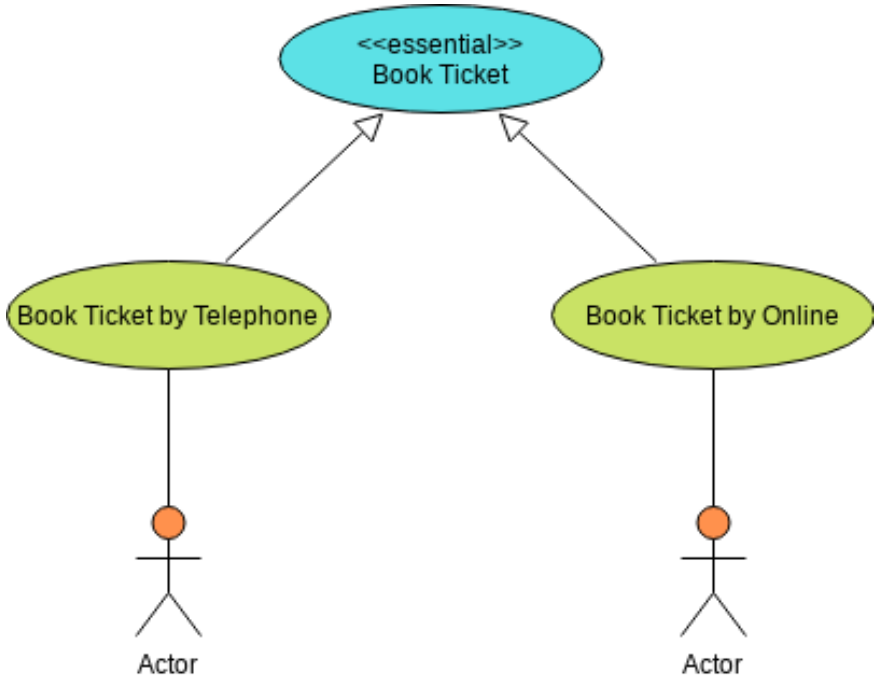
ความสัมพันธ์

- จะลากเส้นตรงระหว่าง Actor และ Use Case
- กรณีที่ Actor มีลักษณะเป็น subset จะใช้เครื่องหมายลูกศร โดยลูกศรวิ่งเข้า Superset (เรียกว่า Generalize)



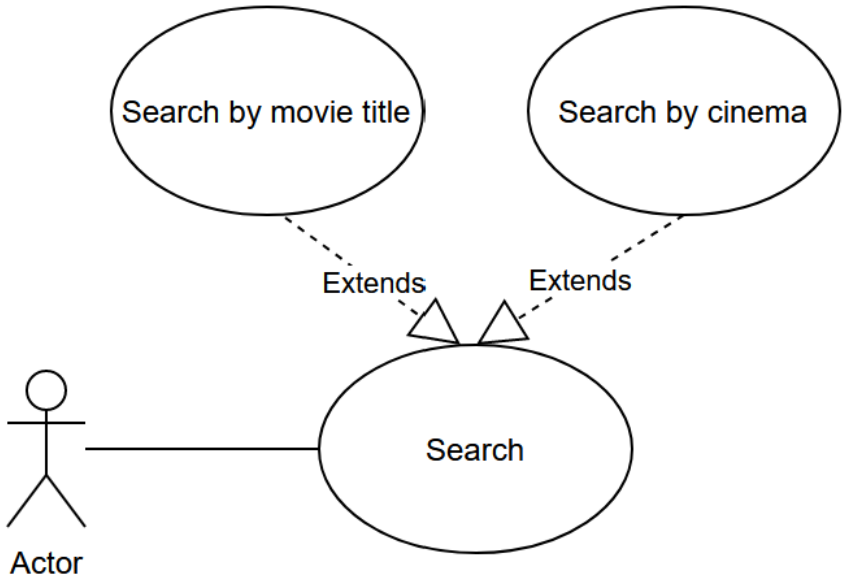
Use Case Diagram

- ความสัมพันธ์แบบ Generalization ระหว่าง Use Case ใช้ในการอธิบายว่าในการทำงานใน Use Case หนึ่งสามารถทำได้มากกว่า 1 วิธี
- จากรูปแสดงให้เห็นว่า การซื้อตั๋ว สามารถทำได้ 2 วิธี



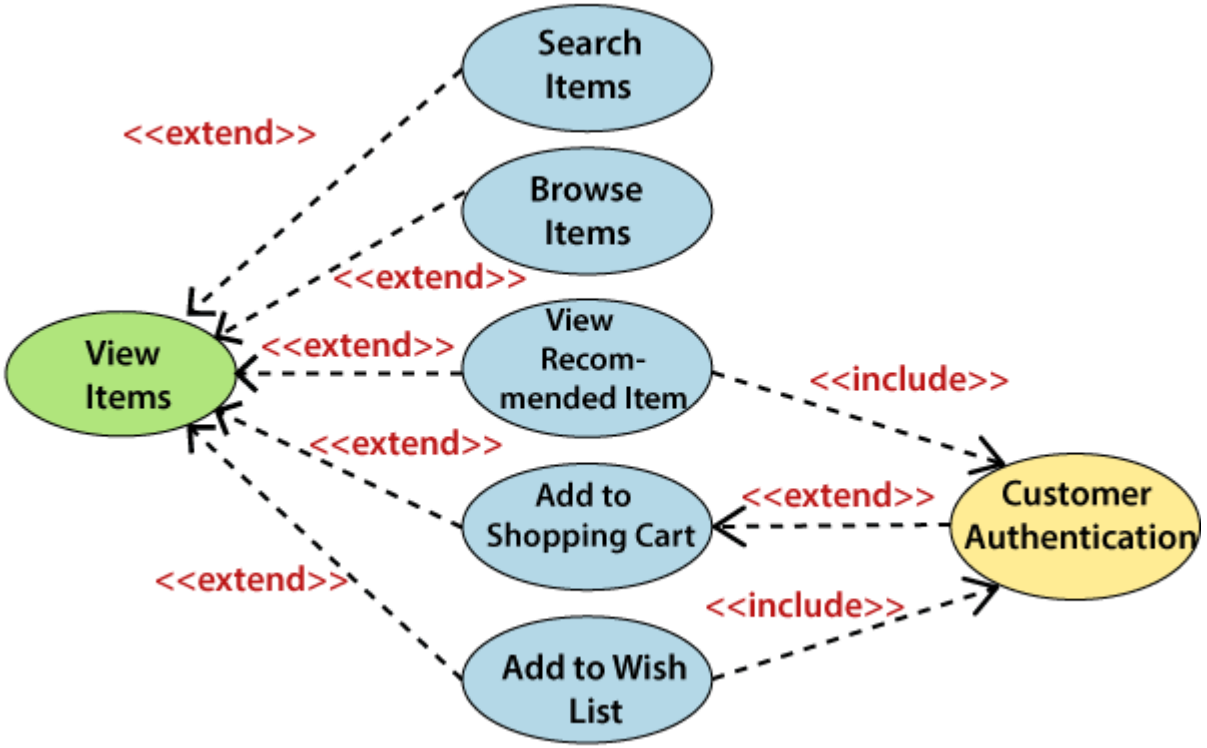
Use Case Diagram

- ความสัมพันธ์แบบ *<<extends>>* ระหว่าง Use Case
  - <<extends>> จะใช้กับกรณีที่การทำงานบางอย่าง เป็นส่วนขยายของอีกงานหนึ่ง
  - เช่น ในระบบโรงภาพยนต์ การค้นหา จะสามารถหาได้ทั้งชื่อภาพยนตร์และโรงภาพยนตร์ การเขียนจะเขียนดังรูป
  - ลูกศรจะชี้ไปยัง use case ตัวที่มีการ extend ออกไป



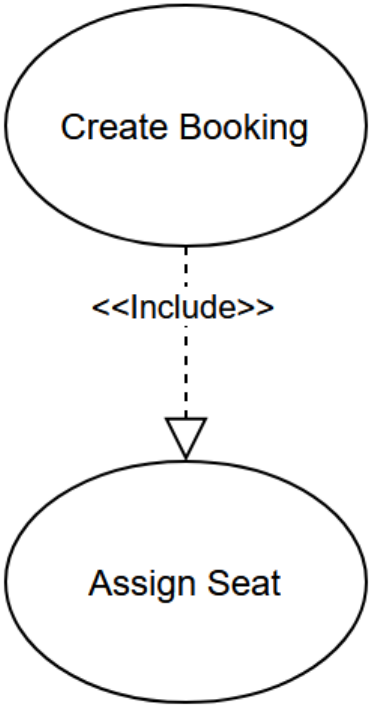
Use Case Diagram

- ตัวอย่างอื่นๆ ของ Extend
- จะเห็นว่าจากหน้า view สินค้า สามารถจะทำอย่างอื่นได้อีก



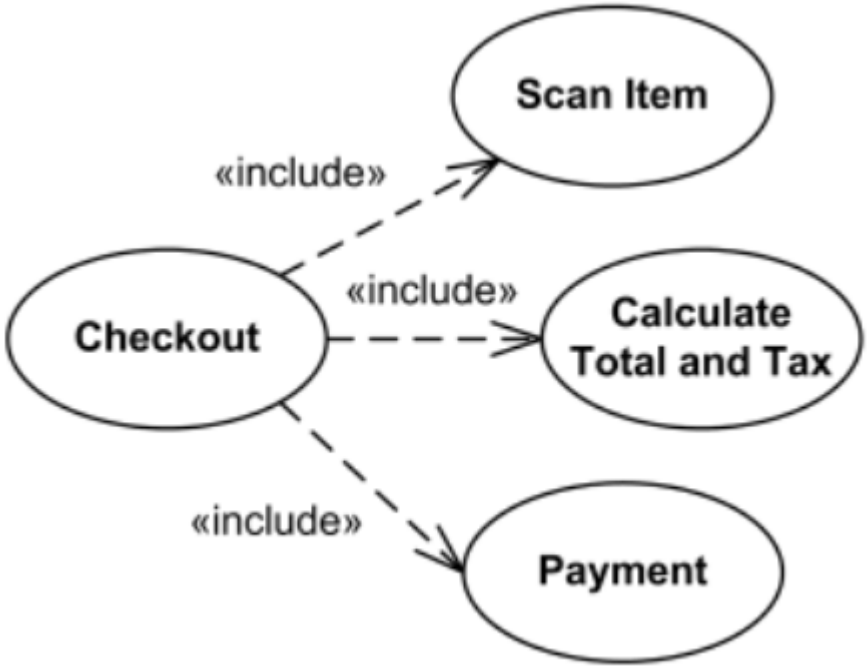
Use Case Diagram

- ความสัมพันธ์ระหว่าง Use Case แบบ *<<include>>*
  - <<include>> จะใช้แทนกรณีที่จะทำงานหนึ่ง จะต้องทำอีกการทำงานหนึ่งเป็นส่วนหนึ่ง
  - เช่น ในระบบโรงภาพยนต์ ก่อนที่จะจองตั๋วได้สำเร็จ จะต้องมีการเลือกที่นั่งก่อน
  - ลูกศรจะชี้ไปยัง use case ตัวที่ถูกใช้งาน



Use Case Diagram

- ตัวอย่างอื่นๆ ของ Include
- จากรูปจะเห็นว่าการ Checkout จะรวมถึง การ Scan ของ คำนวณราคา และ จ่ายเงิน

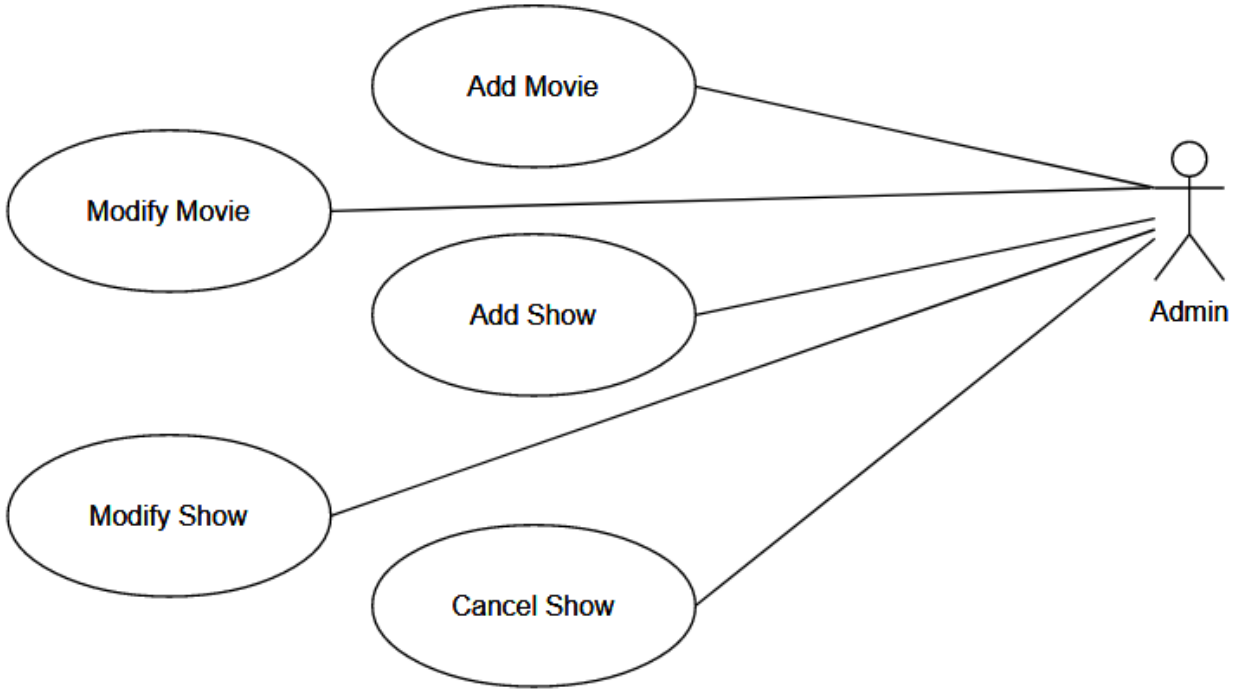


Use Case Diagram

- **ตัวอย่าง** การเขียน Use Case
- พิจารณาการทำงานของระบบ ให้ระบุกิจกรรมที่เกิดขึ้น โดยกิจกรรมควรมีลักษณะเบ็ดเสร็จในตัวเอง
  - ให้แน่ใจว่าทุกส่วนของระบบ จะต้องมีกิจกรรมแสดงใน Use Case Diagram
  - จากนั้นให้พิจารณาว่าแต่ละ Use Case ต้องผ่านการทำงานใน Use Case อื่นมาก่อนหรือไม่ ถ้ามีให้ใส่ความสัมพันธ์แบบ Include
  - จากนั้นให้พิจารณาว่าในแต่ละ Use Case มีอันใดที่เป็นงานขยาย เพิ่มเติมจาก Use Case อื่นหรือไม่ ถ้ามีให้ใส่ความสัมพันธ์แบบ Extend

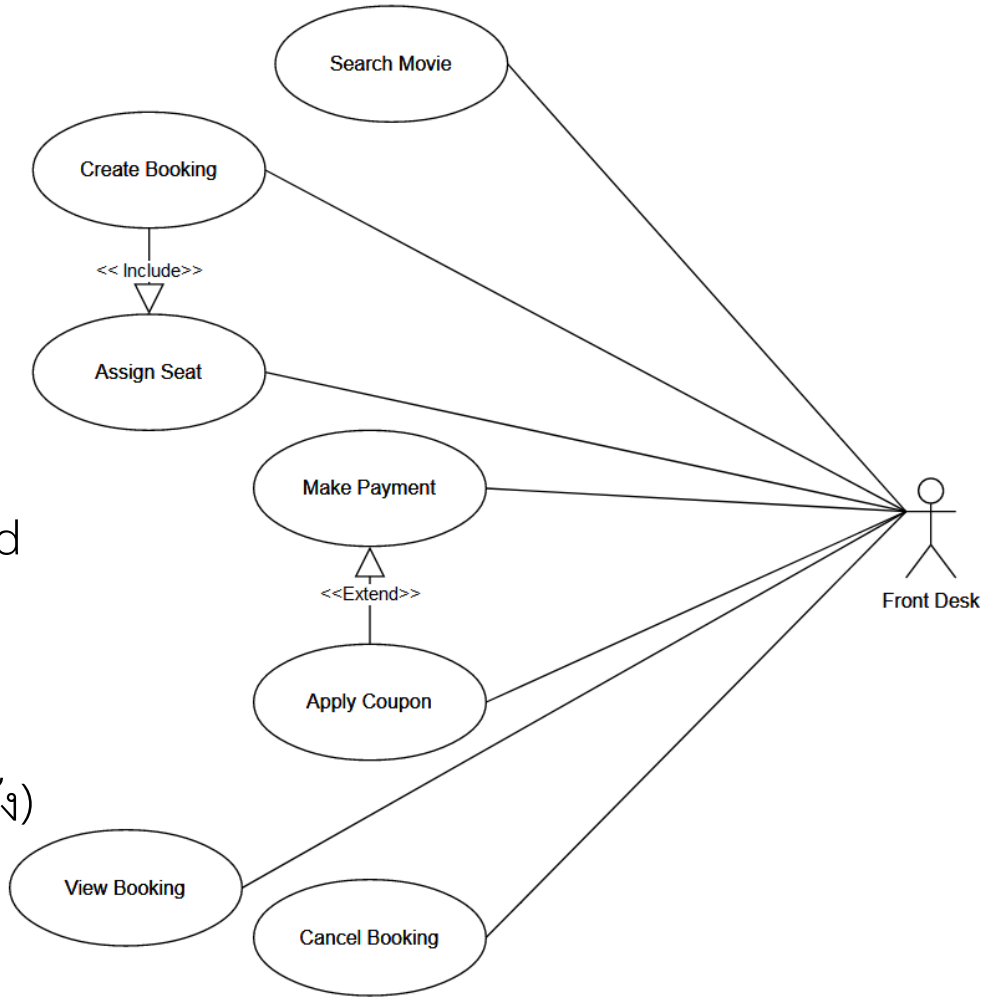
Use Case Diagram

- **ตัวอย่าง** จะเริ่มจากการจัดการกับข้อมูลพื้นฐาน ซึ่งทำโดย Admin ได้แก่ ภาพยนตร์ และ รอบฉาย โดยภาพยนตร์ จะมีการเพิ่ม การแก้ไข สำหรับรอบฉาย นอกจากการเพิ่ม การแก้ไขแล้ว จะมีการยกเลิก กรณีที่มีกรณีพิเศษ หรือ เหตุสุดวิสัย



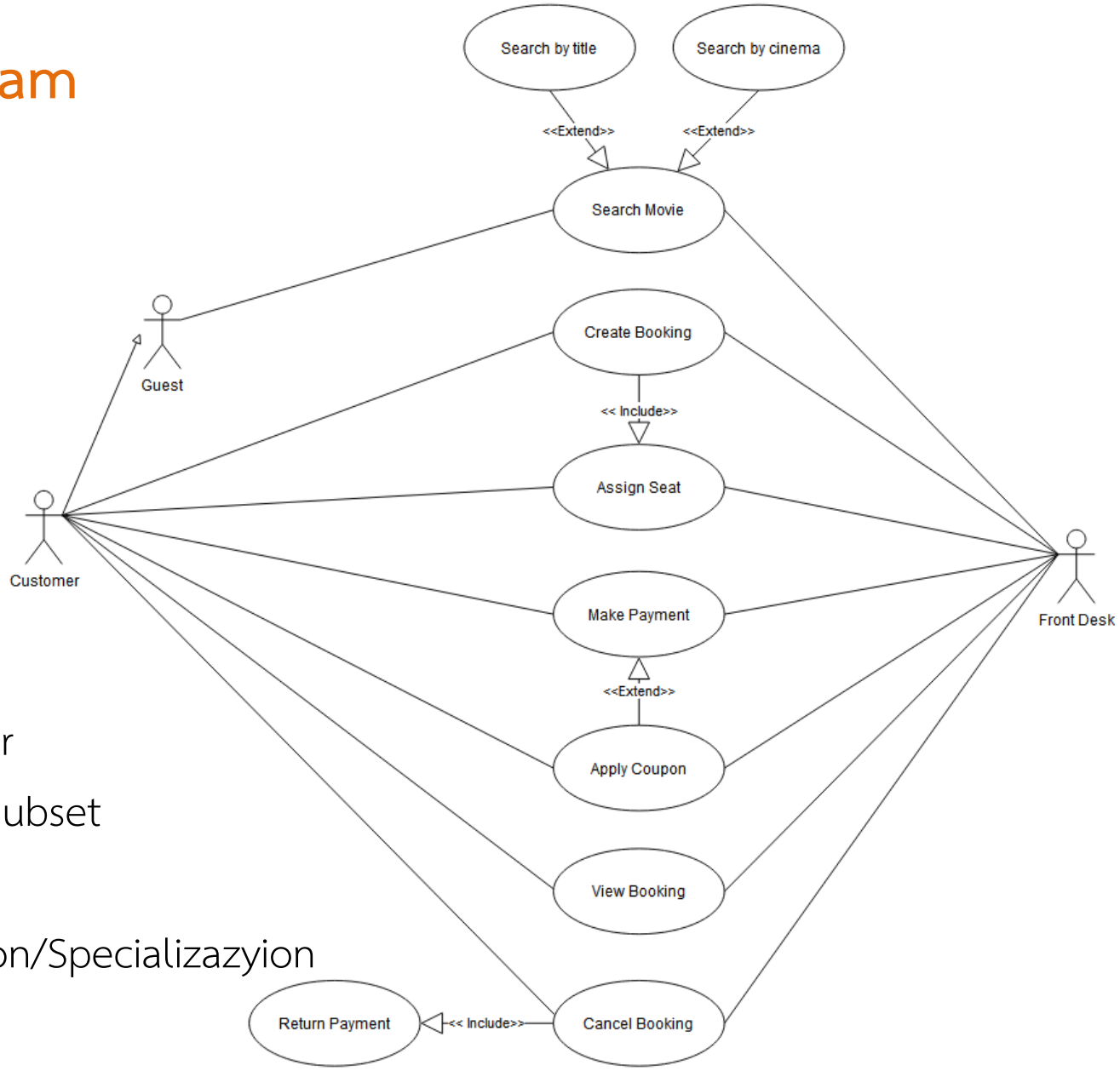
Use Case Diagram

- ส่วนต่อมา คือ ส่วนที่กระทำโดย Front Desk Officer ประกอบด้วย
  - การค้นหาภาพยนตร์
  - การจองตั๋ว ซึ่งจะ Include การกำหนดที่นั่ง
  - การรับชำระเงิน ซึ่งอาจจะ Extend การใช้คูปองลดราคา
  - เจ้าหน้าที่สามารถเรียกดูการจอง (เพราะลูกค้าสามารถชำระภายหลัง)
  - สามารถยกเลิกการจองได้ (กรณีที่มีปัญหา)



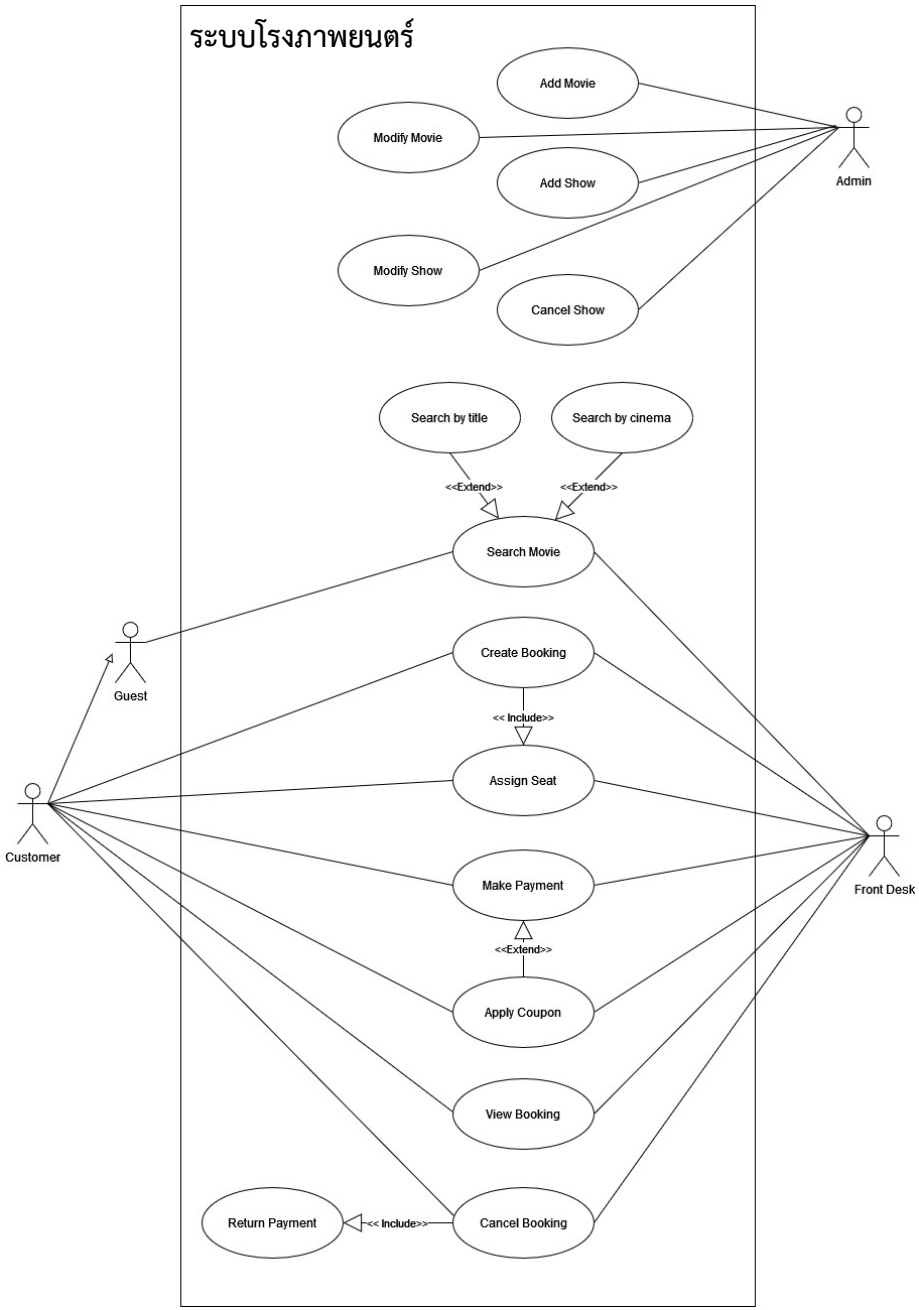
Use Case Diagram

- Use Case Diagram ของ Actor Customer และ Front Desk Officer
- จากรูปจะเห็นความสัมพันธ์ระหว่าง Guest และ Customer โดย Customer เป็น Subset ของ Guest เรียกว่า Generalization/Specializazyion



Use Case Diagram

- รูปแสดง Use Case Diagram ของระบบโรงภาพยนตร์ โดยกรอบสี่เหลี่ยมจะหมายถึงขอบเขตของระบบ (System) ที่จะพัฒนาขึ้น
- การตั้งชื่อ Use Case จะตั้งเป็นคำกริยา เพื่อแสดงว่า Use Case นั้นมีหน้าที่ใด
- สัญลักษณ์ Use Case อาจต่างไปจาก Slide แต่จะคล้ายกัน



Use Case Diagram

- จะเห็นได้ว่าการเขียน Use Case Diagram ที่ดี จำเป็นจะต้องมีขั้นตอนการทำงานที่ชัดเจนเสียก่อน จึงจะเขียน Use Case Diagram ออกมาได้ครบถ้วน
- เมื่อมี Use Case Diagram ที่แสดงความสัมพันธ์ที่ครบถ้วนก็จะช่วยให้การพัฒนาโปรแกรม สามารถทำได้ตรงตามความต้องการได้

Use Case Description

- Use Case Description คือ คำอธิบายรายละเอียดการทำงานของ Use Case แต่ละ Use Case อย่างไรก็ตามอาจเขียน Use Case Description เฉพาะ Use Case หลัก

Use Case Name	จองตั๋วภาพยนตร์ (Create Booking)
Actor	Customer, Front Desk Officer
Description	กระบวนการจองตั๋วภาพยนตร์ เมื่อต้องการเข้ามาชมภาพยนตร์
Normal Course	1. เมื่อผู้ใช้เลือกภาพยนตร์ที่ต้องการชม และ เลือกรอบที่ต้องการชม จะเข้าสู่การจองตั๋ว 2. ระบบจะแสดงรายละเอียดของภาพยนตร์ที่เลือก วันที่ รอบฉาย ภาษาที่ฉาย ชื่อโรงภาพยนตร์ และ โรงย่อย 3. ระบบจะแสดง Layout ของเก้าอี้นั่ง รูปแบบของเก้าอี้นั่ง พร้อมทั้งราคาของที่นั่งแต่ละประเภท 4. หากผู้ใช้เลือกที่นั่ง ระบบจะแสดงที่นั่งที่ผู้ใช้เลือก พร้อมทั้งแสดงราคารวมของการจองทั้งหมด 5. หากผู้ใช้เลือก ส่วนลด หรือ โปรโมชัน ระบบจะเรียกส่วนงานของส่วนลดมาทำงาน
Alternate Course	1. กรณีที่นั่งเต็ม 2. กรณีผู้ใช้ 2 คนเลือกที่นั่งเดียวกัน

For your attention