01076105, 01076106

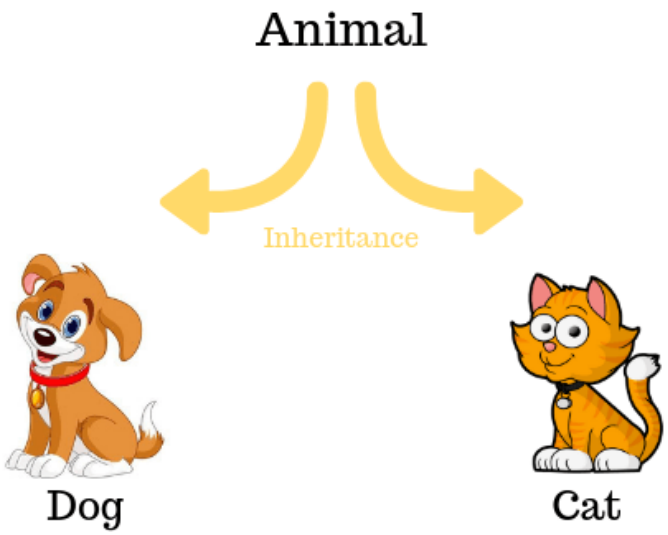
Object Oriented Programming

Object Oriented Programming Project

Inheritance

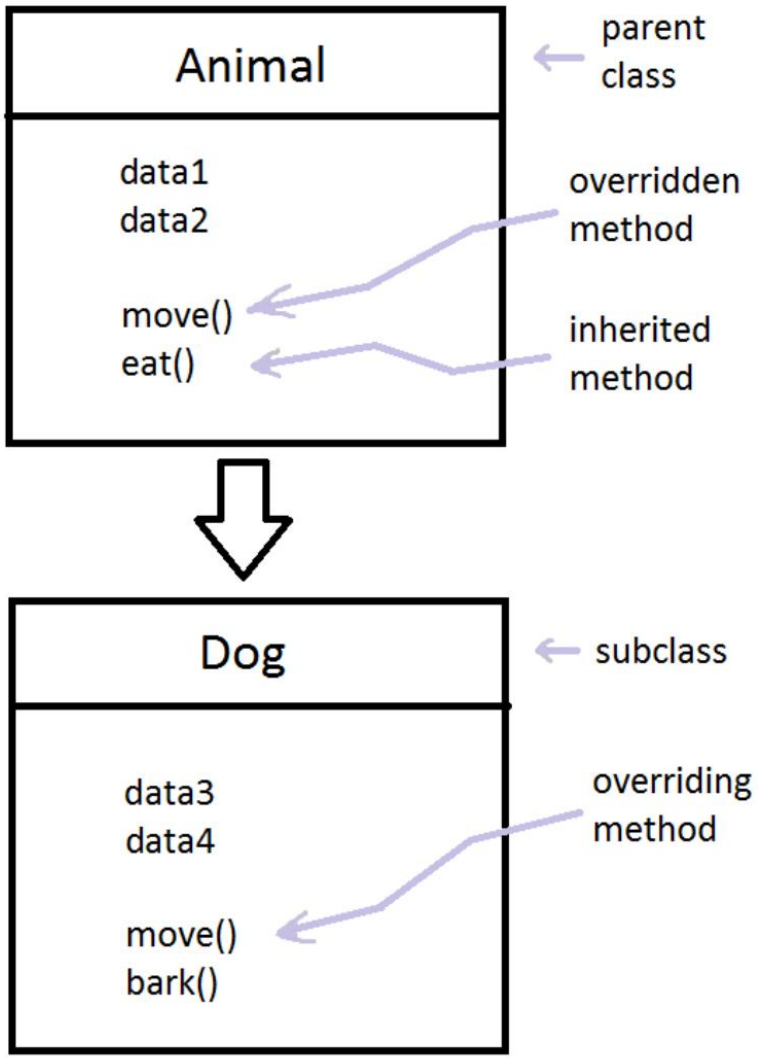
Inheritance

- 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 ไม่ใช่ จึงเป็น 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

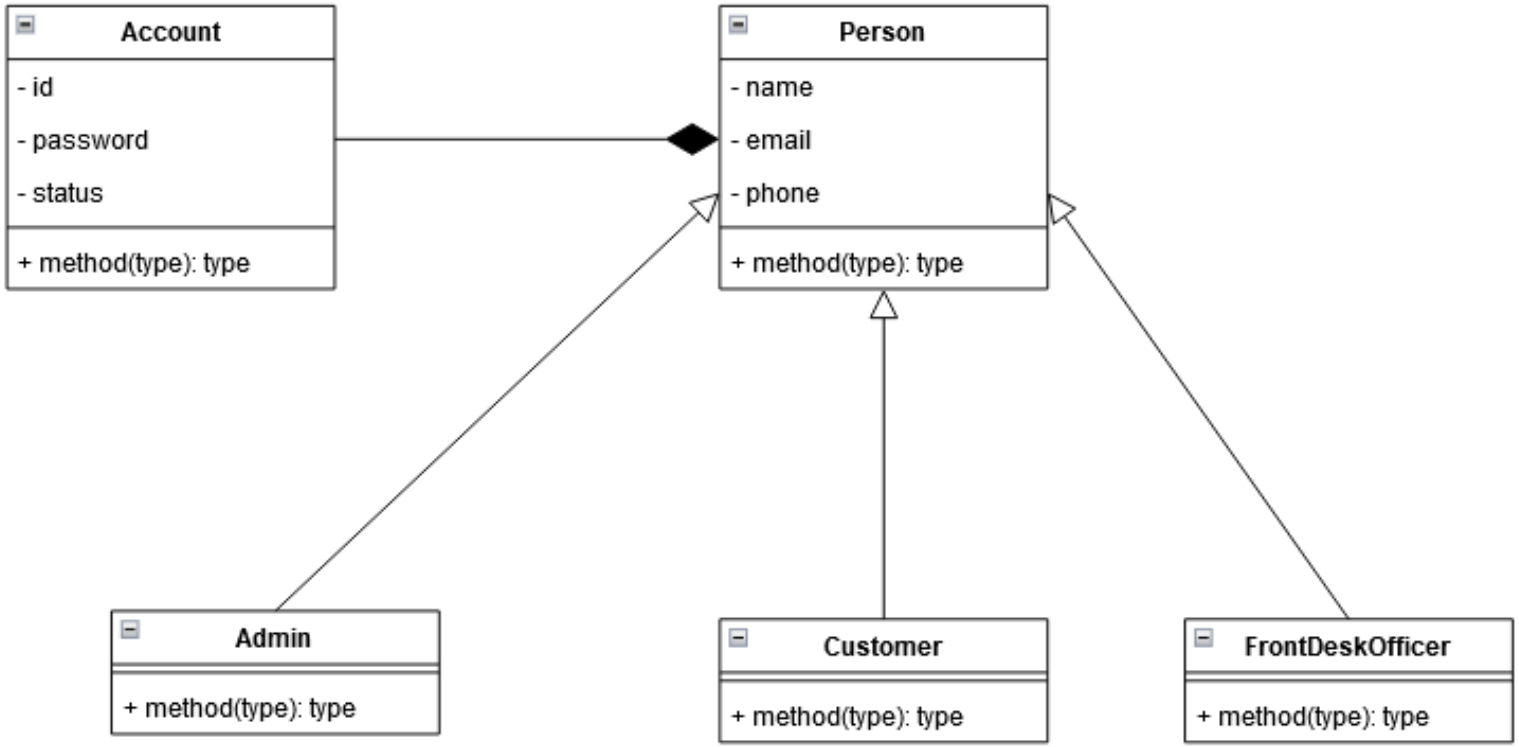
```
class Ractangle(Polygon):
    pass
```

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

- **ตัวอย่าง** สมมติว่ามี Class Person ซึ่งทุกคนจะมี Account เนื่องจากหน้าที่ของ Person มีหลายประเภท จึงใช้ Inheritance คือ Admin, Customer, Front Desk Officer ซึ่ง Inherit มาจาก Person



Inheritance

- การแยกคลาส Person ออกมาจะทำให้ความซ้ำซ้อนของข้อมูลลดลง โดยข้อมูลที่เหมือนกันจะอยู่ในคลาส Person และข้อมูลที่ต่างกันจะอยู่ใน Subclass ของแต่ละประเภทย่อย
- การทำเช่นนี้ มีข้อดี ที่ทำให้การปรับเปลี่ยนในอนาคตสามารถทำได้โดยมีการแก้ไข Code เดิมน้อยลง หากมีการเปลี่ยนแปลงโดยรวมก็แก้ไขเพียงคลาส Person คลาสเดียว หรือ หากมีการเปลี่ยนแปลงย่อย ก็เพียงแต่สร้าง Subclass ใหม่ขึ้นมา
- ขอยกตัวอย่าง หากในอนาคตมีการเพิ่มผู้ใช้ประเภทใหม่ขึ้นมา หากใช้วิธี Inherit จะทำให้ไม่ต้องไปแก้ไข Code เดิมในคลาส Person โดย Code สำหรับกลุ่มผู้ใช้ที่เพิ่มเข้ามาใหม่ ก็จะอยู่ในคลาสที่สร้างเพิ่มเติมขึ้นมาใหม่

Inheritance

- คลาสที่ Inherit มากจากคลาสอื่น และใน Subclass ไม่มี Constructor จะใช้ Constructor ของ Superclass แทน

```
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)
```

```
3
Blue
```

Inheritance

- แต่หาก Subclass มี Constructor ของตนเอง ก็จะไม่ใช่ Constructor ของ Superclass

```
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

my_triangle = Triangle(3, "Blue")

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

Inheritance

- แต่หากจะให้ subclass ไปเรียกใช้ Constructor ของ Superclass จากนั้นจึงเรียกใช้ Constructor ของคลาสตัวเองจะเขียนดังนี้

```
class Triangle(Polygon):

    NUM_SIDES = 3

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

my_triangle = Triangle(5, 4, "blue")

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

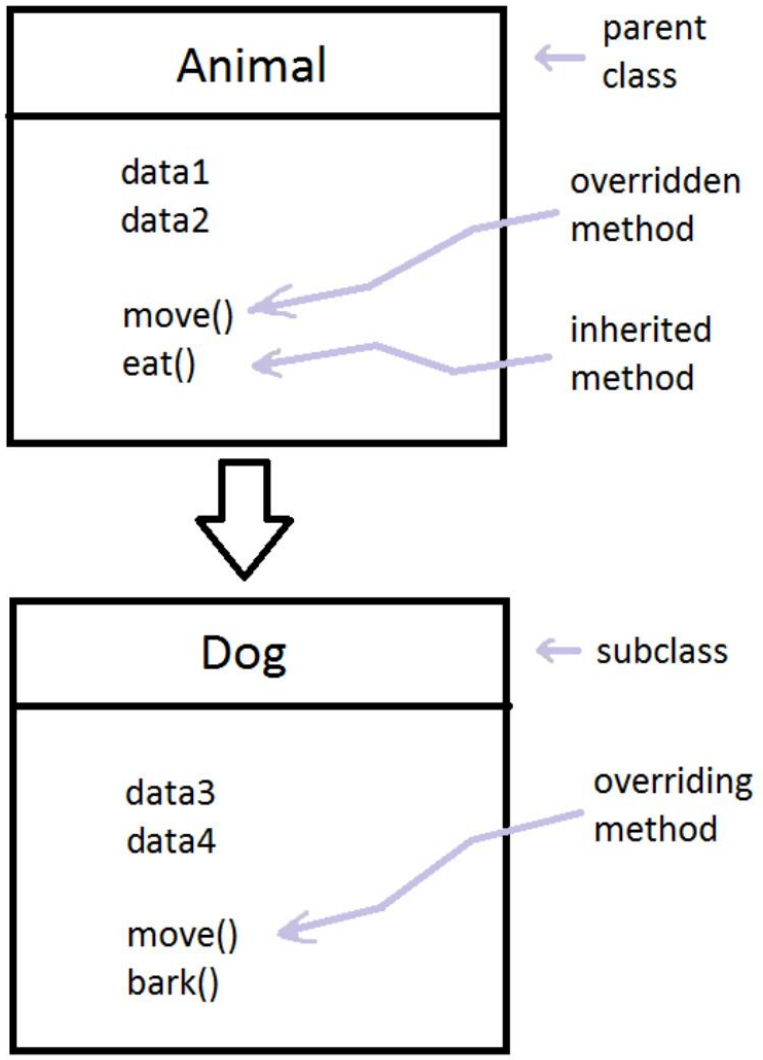
Inheritance

- ให้ subclass ไปเรียกใช้ Constructor ของ Superclass อีกวิธี (มี self)

```
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


my_triangle = Triangle(5, 4, "blue")

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

Method Overriding

- คือการกำหนด การทำงานของ method ของ superclass ใหม่โดย subclass
- จากตัวอย่าง ใน class Animal มี method Move() อยู่ก่อนแล้ว
- สมมติว่าใน class Dog มีการเคลื่อนที่ ซึ่งจากไปจาก class Animal ก็สามารถกำหนดการเคลื่อนที่ของ Dog เสียใหม่ โดยการสร้าง method Move() ทับซ้ำ
- จะเรียกการทำงานแบบนี้ว่า Method overriding



Method Overriding

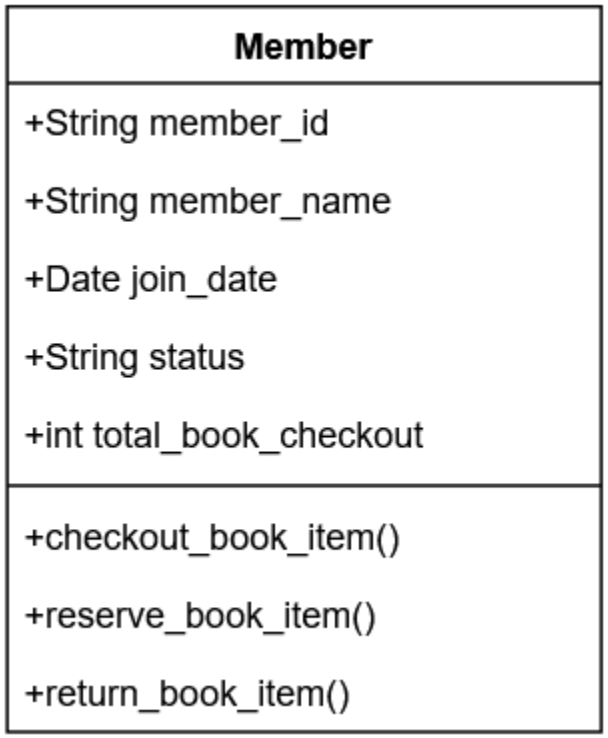
- จาก code ตัวอย่างมี 2 คลาส คือ class Parent และ class Child
- ใน class Parent มีการกำหนด method show( ) เอาไว้ โดยเป็นการแสดงคำว่า “Inside Parent”
- แต่ใน class Child มีการกำหนด method show( ) ทับไป เมื่อมีการเรียก show ของ child ก็จะมีการทำงานต่างออกไป

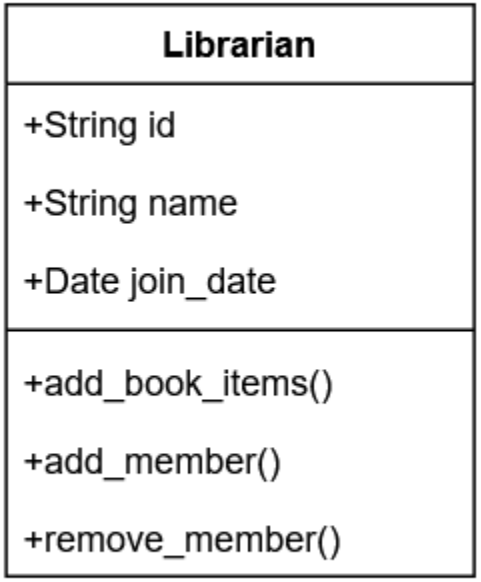
```
Inside Parent
Inside Child
```

```
class Parent():
    def __init__(self):
        self.value = "Inside Parent"

    # Parent's show method
    def show(self):
        print(self.value)

class Child(Parent):
    def __init__(self):
        self.value = "Inside Child"

    # Child's show method
    def show(self):
        print(self.value)

# Driver's code
obj1 = Parent()
obj2 = Child()

obj1.show()
obj2.show()
```

ตัวอย่าง Inheritance

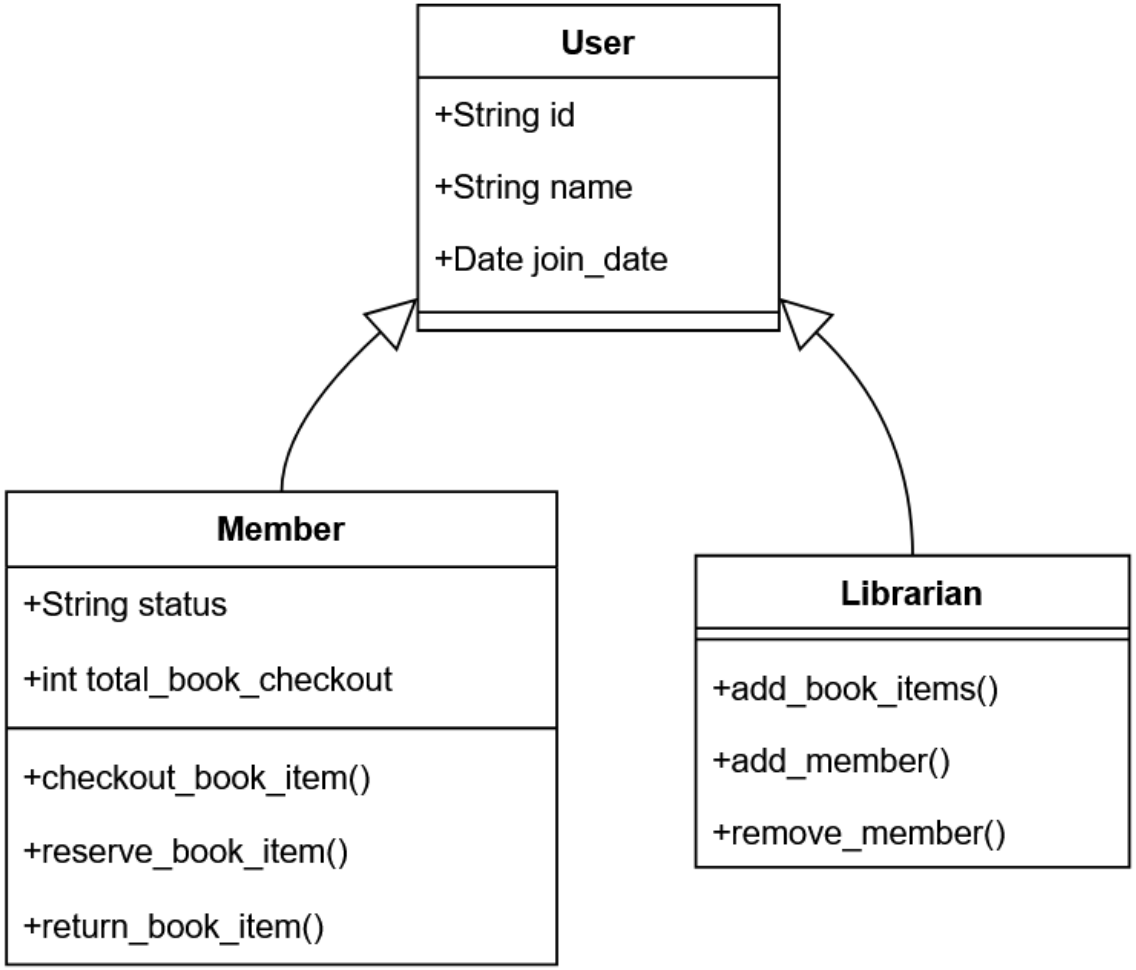
- จากตัวอย่างห้องสมุด เดิม class Member และ Library มีดังนี้
- จะเห็นว่ามีข้อมูลที่ซ้ำกันอยู่ คือ id, name และ join_date ซึ่งควรสร้าง class user ขึ้นมาและใช้เป็น superclass





ตัวอย่าง Inheritance

- จะได้โครงสร้าง class ดังนี้



ตัวอย่าง Inheritance

- มาดูอีกตัวอย่างหนึ่ง จะยกตัวอย่างเป็นเกม RPG
- ในเกม RPG เราจะมีตัวละครหลายประเภท แต่ทุกตัวละครจะมีคุณสมบัติพื้นฐานที่เหมือนกัน เช่น:
  - มีชื่อ (name)
  - มีเลเวล (level)
  - มีพลังชีวิต (hp)
  - มีพลังเวทย์ (mp)
  - มีค่าประสบการณ์ (exp)
  - มีตำแหน่งในเกม (position)

ตัวอย่าง Inheritance

- และมีการกระทำพื้นฐานที่เหมือนกัน เช่น:
  - โจมตี (attack)
  - ใช้สกิล (useSkill)
  - เคลื่อนที่ (move)
  - เลเวลอัพ (levelUp)
  - รับค่าประสบการณ์ (gainExp)
  - เช็คว่าตายหรือยัง (isDead)

ตัวอย่าง Inheritance

- แต่ละตัวละครจะมีวิธีการทำงานที่แตกต่างกัน เช่น
- Warrior (นักรบ)
  - attack():
    - โจมตีระยะประชิด
    - ความแรงขึ้นกับ strength
    - มีโอกาสติด critical hit
    - ใช้ rage ในการโจมตีพิเศษ
  - useSkill():
    - Charge: วิ่งเข้าใส่ศัตรู (ใช้ rage 20)
    - Shield Block: ป้องกันการโจมตี (ใช้ rage 30)
    - Whirlwind: หมุนโจมตีรอบทิศทาง (ใช้ rage 50)
  - move():
    - เคลื่อนที่ช้า (speed = base_speed * 0.8)
    - แต่ทนทานต่อการถูกชะลอ

ตัวอย่าง Inheritance

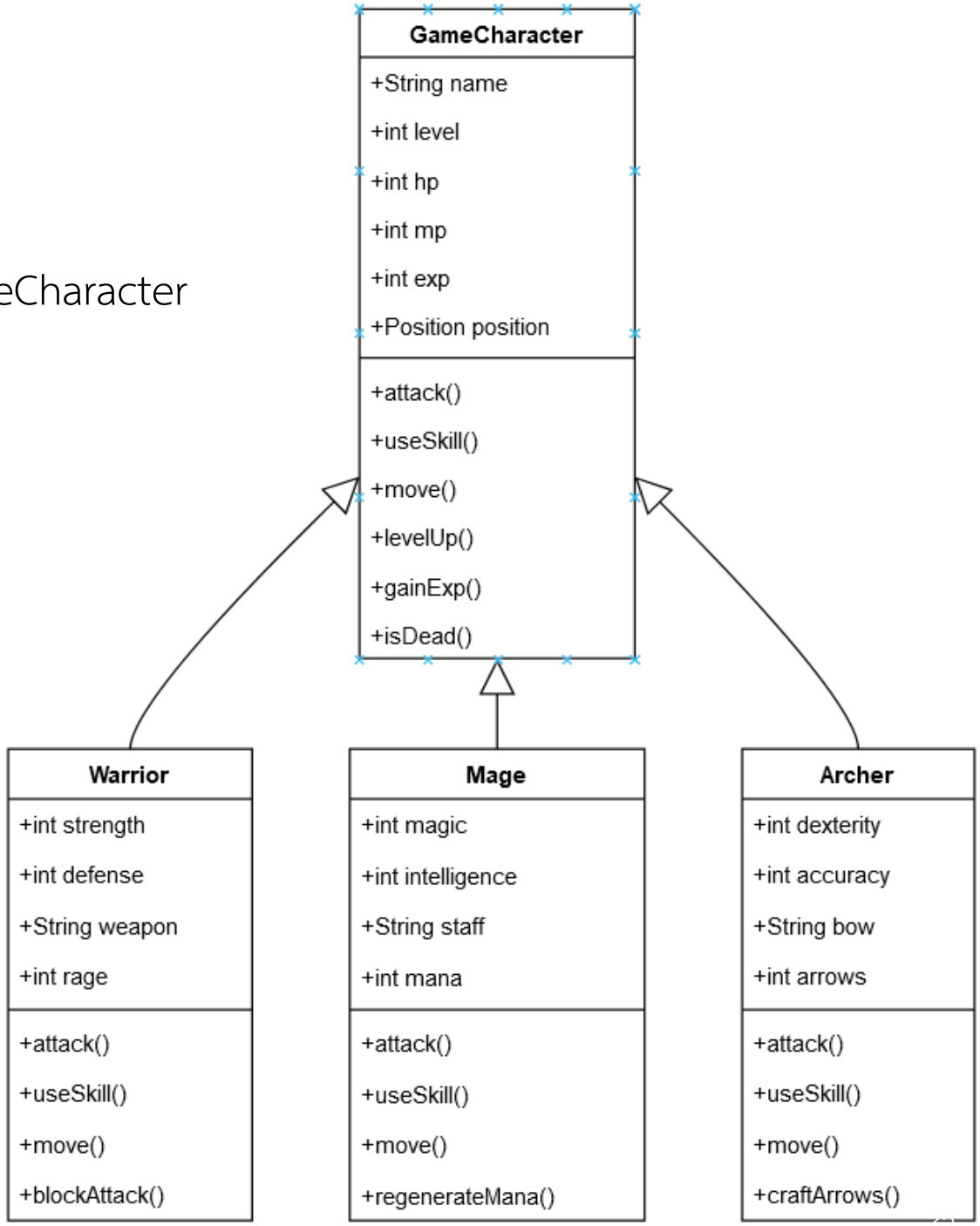
- Archer (นักธนู)
  - attack():
    - โจมตีระยะไกลด้วยธนู
    - ความแรงขึ้นกับ dexterity + accuracy
    - ใช้ arrows ในการโจมตี
    - มีโอกาสยิงทะลุเป้าหมาย
  - useSkill():
    - Multi Shot: ยิงหลายลูกศร (ใช้ arrows 3)
    - Trap: วางกับดัก (ใช้ arrows 1)
    - Aimed Shot: ยิงแม่นยำ (ใช้ arrows 1)
  - move():
    - เคลื่อนที่เร็ว (speed = base_speed * 1.2)
    - สามารถยิงขณะเคลื่อนที่

ตัวอย่าง Inheritance

- Mage (นักเวทย์)
  - attack():
    - โจมตีระยะไกลด้วยเวทย์
    - ความแรงขึ้นกับ magic + intelligence
    - ใช้ mana ในการโจมตี
    - มีโอกาสติด element effect
  - useSkill():
    - Fireball: ลูกบอลไฟ AOE (ใช้ mana 50)
    - Teleport: เคลื่อนย้ายทันที (ใช้ mana 30)
    - Ice Shield: กำแพงน้ำแข็ง (ใช้ mana 40)
  - move():
    - เคลื่อนที่ปกติ (speed = base_speed)
    - มีสก teleport ช่วยหนี

ตัวอย่าง Inheritance

- เราจะสร้าง superclass ชื่อ GameCharacter
- จากนั้นสร้าง Subclass ได้แก่
  - Warrior
  - Mage
  - Archer
- แต่ละ subclass จะมี attribute เพิ่มจาก superclass
- แต่ละ subclass อาจมี method แต่เหมือนกับ superclass แต่วิธีการต่างไป (overriding)
- และอาจมี method ที่ต่างออกไป



ตัวอย่าง Inheritance

- ประโยชน์ของ method overriding คือ ในการเขียนโปรแกรมนั้น เราจะต้องวน loop เพื่อรับการควบคุม (กรณีไม่ใช่เป็น event) ดังนั้นเราจะต้องวนลูปดังนี้
  - รับการควบคุม
  - ตรวจสอบการควบคุม เช่น ตรวจสอบว่าเป็นการสั่งให้ attack ก็ให้เรียก method attack ของ object นั้น
- การที่เราใช้ชื่อ method ว่า attack() เหมือนกัน ทำให้เราสามารถสั่งให้ นักรบ นักธนู หรือ นักเวทย์ โจมตี โดยใช้ loop เดียวกัน แล้วเรียก char.attack() เหมือนๆ กันได้ แม้ว่า code ที่ทำงานจริงของตัวละคนแต่ละตัวจะเป็นคนละ method กันก็ตาม
- ทำให้การเขียนโปรแกรมทำได้ง่ายขึ้น code จะดูสะอาดมากขึ้น ลดการใช้ if ลง

ตัวอย่าง Inheritance

- ถ้าจะเพิ่มตัวละครใหม่ประเภท "Priest" ต้องทำอย่างไรบ้าง?
- หากต้องการเพิ่มระบบ "Buff" ควรเพิ่มที่ class ไหน?

ตัวอย่าง Inheritance

- ระบบ Buff คือการเพิ่มความสามารถชั่วคราวให้กับตัวละคร (ผลทางบวก) โดยทีมเมท
  - Warrior:
    - Battle Cry: เพิ่มพลังโจมตีให้ทีม
    - Defensive Stance: เพิ่มการป้องกันให้ทีม
  - Mage:
    - Arcane Intellect: เพิ่มพลังเวทย์
    - Magic Shield: ป้องกันการโจมตีด้วยเวทย์
  - Archer:
    - Eagle Eye: เพิ่มความแม่นยำให้ทีม
    - Camouflage: ลดโอกาสถูกโจมตี

ตัวอย่าง Inheritance

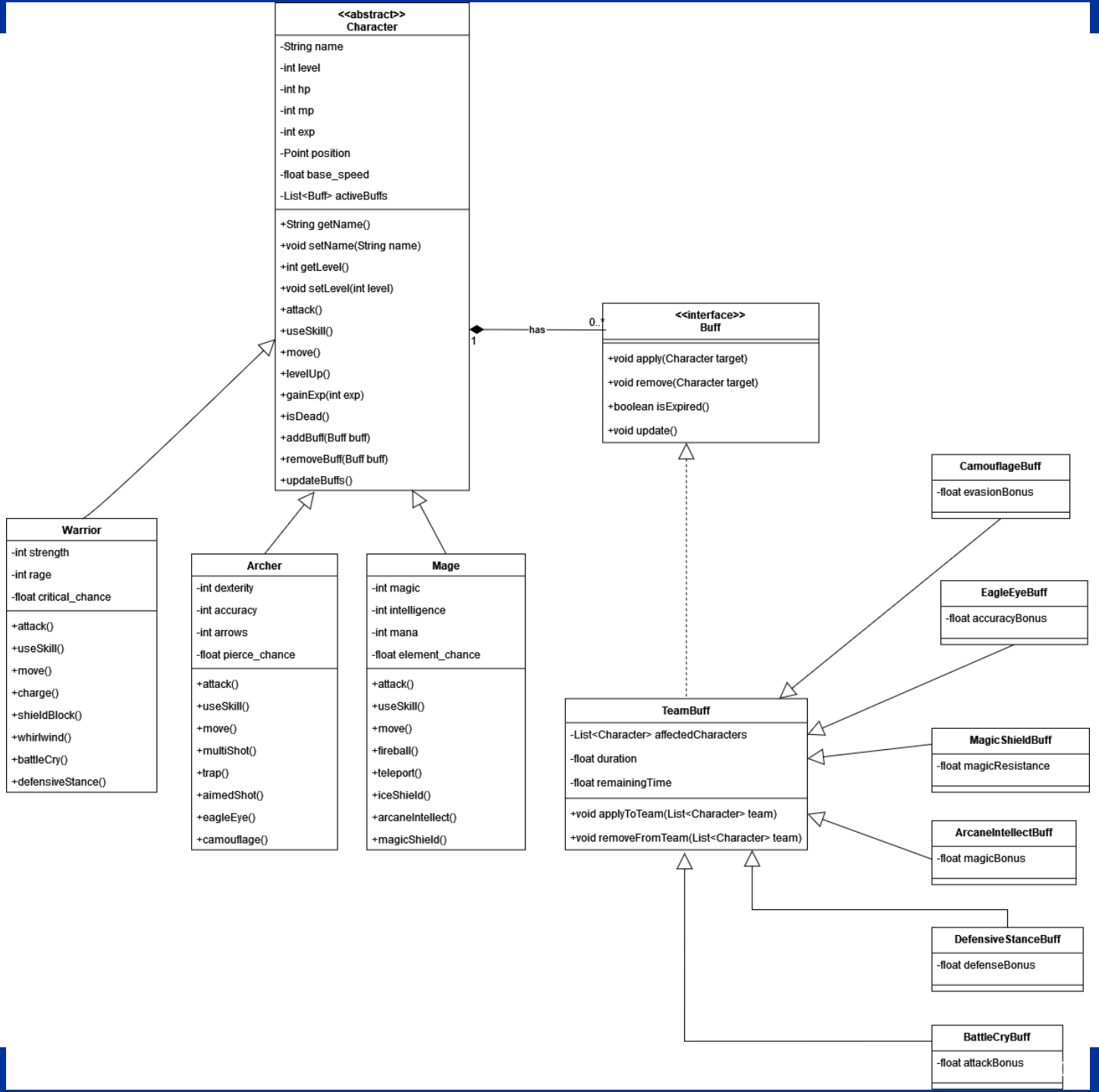
- กรณีของ Buff แม้จะคล้ายกับว่า Buff แต่ละตัวไม่เกี่ยวข้องกัน และ ดูเหมือนว่าเป็น Action ที่ตัวละครตัวหนึ่งกระทำกับตัวละครอีกตัวหนึ่ง แต่เมื่อทดลองเขียน code เช่น

```
def apply_battle_cry(self, target: 'Character') -> None:
    target.stats.attack_power += self.attack_bonus
    self.affected_characters.append(target)

def apply_magic_shield(self, target: 'Character') -> None:
    target.stats.magic_resistance += self.magic_resistance
    self.affected_characters.append(target)
```

- จะเห็นว่า battle cry ซึ่ง warrior ใช้ และ magic shield ที่ mage ใช้ มีรูปแบบเหมือนกัน
- นอกจากนั้นในทุก Buff ยังมี action เดียวกันด้วย คือ apply, remove, is_expire และ update ดังนั้นการทำ buff เป็น class จะทำให้ code มีระเบียบมากกว่า

ตัวอย่าง

Inheritance



ตัวอย่าง Inheritance

- จะเห็นว่าได้สร้าง class Buff ขึ้นมา แล้วค่อย Inherit เป็น TeamBuff โดยใน class Buff จะมี method เป็น
  - apply คือ ใช้ buff นี้กับตัวละครใด
  - remove คือ ยกเลิกการใช้ buff กับตัวละครใด
  - isExpires และ update ใช้สำหรับตรวจสอบการหมดเวลาของ Buff และ update เวลาของ buff
- จะเห็นว่า class Buff จะมีความสัมพันธ์แบบ composition ของ GameCharacter เนื่องจาก เนื่องจาก Buff เป็นส่วนประกอบของ character ไม่สามารถอยู่เดี่ยวๆ ได้
- สำหรับเหตุผลที่ inherit TeamBuff มาอีกที แทนที่จะใช้เลย เนื่องจากอาจมี buff แบบอื่นๆ อีก เช่น Buff จาก item, Buff จากสภาพแวดล้อม ฯลฯ

ตัวอย่าง Inheritance

- ในการเขียนโปรแกรมจริง จะต้องมีการแก้ไข class ตัวละครด้วย
- เพิ่มเมธอดจัดการ Buff (addBuff, removeBuff, updateBuffs)
- เพิ่มเมธอดสำหรับการใช้ Buff ของแต่ละตัวละคร
  - Warrior: battleCry(), defensiveStance()
  - Mage: arcaneIntellect(), magicShield()
  - Archer: eagleEye(), camouflage()

ตัวอย่าง Inheritance

- ตัวอย่างที่ยกมาเพียงให้เห็นแนวคิดและประโยชน์ของการใช้ Inheritance
- ในการเขียนโปรแกรมจริง จะมีรายละเอียดเพิ่มเติมอีก เช่น
  - การซ้อนทับ (Stacking) ของ Buff เนื่องจากบาง Buff ซ้อนกันได้ (Stack) เช่น Healing +10 สองครั้ง = +20
  - บาง Buff ใช้ค่าสูงสุด (Max Value) เช่น Defense Up 20% และ 30% = ใช้ 30%
  - บาง Buff ต่อเวลาได้ เช่น Speed Buff 30 วิ + 30 วิ = 60 วิ
- ทั้งหมดนี้หากใช้ OOP และ Inheritance จะทำให้การเขียนโปรแกรมทำได้ clean และเป็นระเบียบมากขึ้น

Exercise Inheritance

- ร้านอาหารแห่งหนึ่ง รับสั่งทาง online โดยมีรายละเอียดดังนี้
  - ประเภทอาหาร (Food Types):
    - อาหารทั่วไป (Regular Food)
    - อาหารจานด่วน (Fast Food)
    - เครื่องดื่ม (Beverage)
    - ของหวาน (Dessert)
  - รูปแบบการสั่ง (Order Types):
    - สั่งทันที (Instant Order)
    - สั่งล่วงหน้า (Pre-Order)
    - สั่งแบบ Catering

Exercise Inheritance

- ร้านอาหารแห่งหนึ่ง รับสั่งทาง online โดยมีรายละเอียดดังนี้
  - วิธีการชำระเงิน (Payment Methods):
    - เงินสด (Cash)
    - บัตรเครดิต (Credit Card)
    - E-Wallet
    - คูปอง (Voucher)
  - ประเภทการจัดส่ง (Delivery Types):
    - จัดส่งด่วน (Express)
    - จัดส่งปกติ (Normal)
    - รับที่ร้าน (Pick-up)

Exercise Inheritance

- ร้านอาหารแห่งหนึ่ง รับสั่งทาง online โดยมีรายละเอียดดังนี้
- ความต้องการของระบบ:
  - ระบบจัดการเมนูอาหาร:
    - สามารถเพิ่ม/ลบ/แก้ไขเมนูได้
    - แต่ละเมนูมีราคาและวิธีการคำนวณราคาที่ต่างกัน
    - ~~ตรวจสอบความพร้อมในการขาย~~
    - ~~คำนวณเวลาในการเตรียมอาหาร~~
  - ระบบการสั่งอาหาร:
    - รองรับการสั่งหลายรูปแบบ
    - คำนวณราคารวม
    - จัดการคิวการสั่งอาหาร
    - ยกเลิกและแก้ไขการสั่ง

Exercise Inheritance

- ระบบการชำระเงิน:
  - รองรับหลายวิธีการชำระ
  - ตรวจสอบการชำระเงิน
  - คืนเงินกรณียกเลิก
  - คำนวณส่วนลด
- ระบบการจัดส่ง:
  - คำนวณค่าจัดส่ง
  - กำหนดเวลาจัดส่ง
  - ติดตามสถานะการจัดส่ง
  - จัดการคนขับ

Exercise Inheritance

- วิธีการคำนวณราคา
  - อาหารทั่วไป (RegularFood)
    - ราคา = ราคาพื้นฐาน + ค่าวัตถุดิบพิเศษ + ค่าความต้องการพิเศษ
    - ตัวอย่าง: ผัดกะเพราหมู
      - ราคาพื้นฐาน = 50 บาท
      - ขอไข่ดาวเพิ่ม +10 บาท
      - พิเศษเพิ่ม +10 บาท
      - ราคารวม = 75 บาท

Exercise Inheritance

- อาหารจานด่วน (FastFood)
  - ราคา = ราคาพื้นฐาน + ค่าขนาด + ค่าท็อปปิ้ง - ส่วนลดเซ็ต
  - ตัวอย่าง: แฮมเบอร์เกอร์
    - ราคาพื้นฐาน = 79 บาท
    - ขนาดใหญ่ +20 บาท
    - เพิ่มชีส +15 บาท
    - สั่งเป็นชุด (ลด 10%) -11.40 บาท
    - ราคารวม = 102.60 บาท

Exercise Inheritance

- เครื่องดื่ม (Beverage)
  - ราคา = ราคาพื้นฐาน + ค่าขนาด + ค่าเพิ่มเติม
  - ตัวอย่าง: ชานมไข่มุก
    - ราคาพื้นฐาน = 45 บาท
    - ขนาด L +10 บาท
    - เพิ่มไข่มุก +5 บาท
    - เพิ่มวิปครีม +10 บาท
    - ราคารวม = 70 บาท

Exercise Inheritance

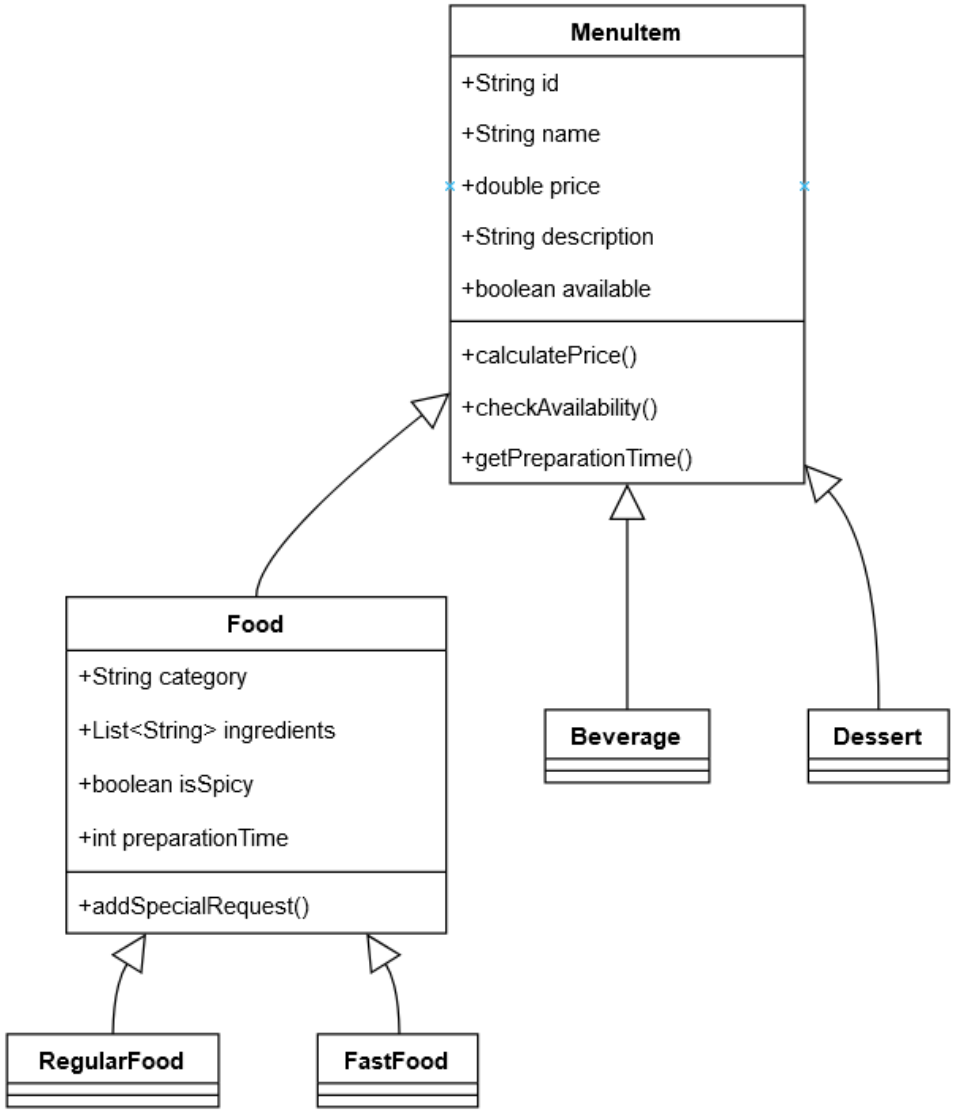
- ของหวาน (Dessert)
  - ราคา = ราคาพื้นฐาน + ค่าปรับแต่ง + ค่าท็อปปิ้ง
  - ตัวอย่าง: บิงซู
    - - ราคาพื้นฐาน = 89 บาท
    - - เพิ่มผลไม้รวม +30 บาท
    - - เพิ่มไอศกรีม +25 บาท
    - - ขนาดใหญ่พิเศษ +40 บาท
    - ราคารวม = 184 บาท

Exercise Inheritance

- ให้ทดลองออกแบบ Class Diagram

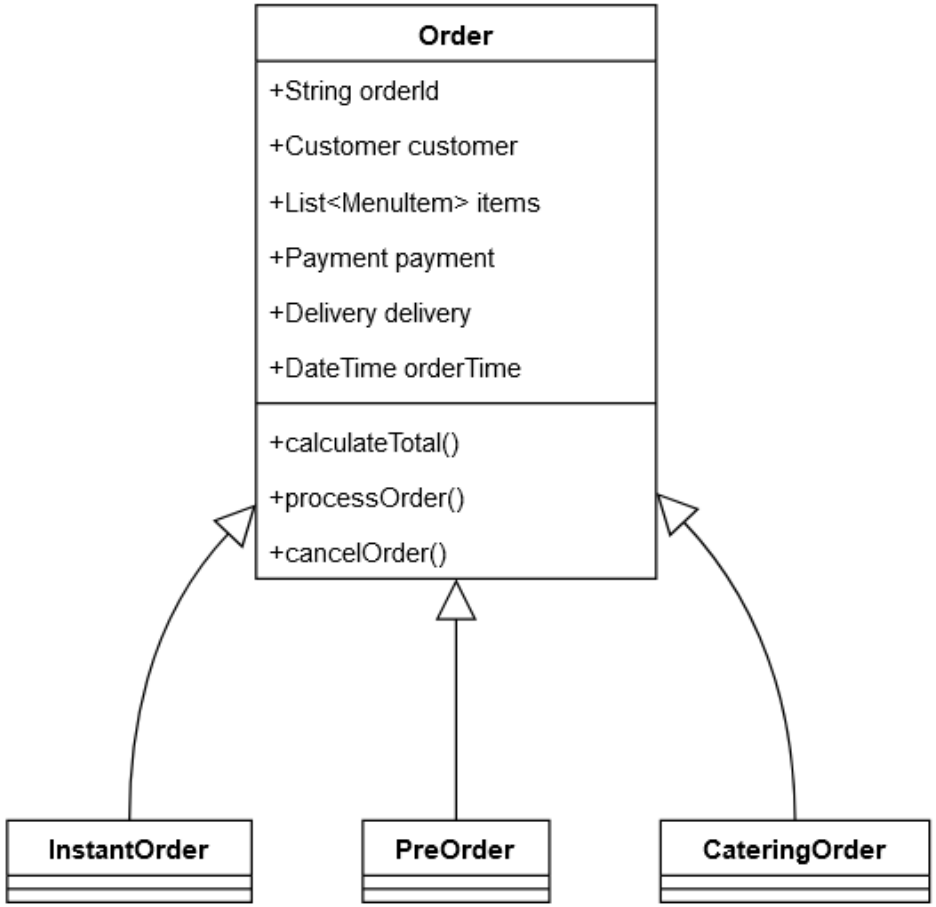
เฉลย

- วิธีการออกแบบที่ดี ควรเริ่มจากการออกแบบเบื้องต้นก่อน คือ ออกแบบโครงสร้างของคลาสแล้วค่อยปรับปรุงรายละเอียด เริ่มจากระบบเมนูอาหาร
- จะสร้าง Class MenuItem
- Inherit มาเป็น Food, Beverage, Dessert
- Food ค่อย Inherit มาเป็น RegularFood และ FastFood (เนื่องจากมีวิธีคำนวณราคาไม่เหมือนกัน)



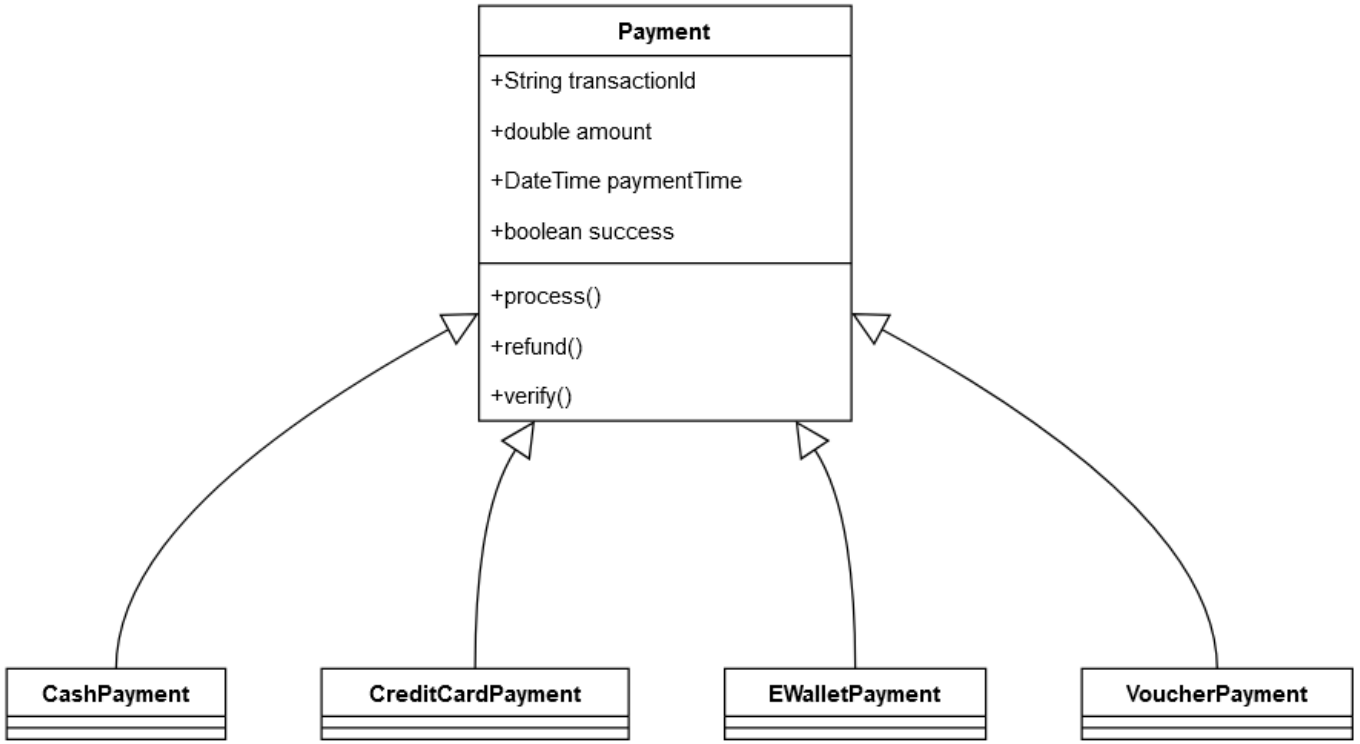
เฉลย

- ระบบสั่งอาหาร
  - ใน 1 order มีได้หลายรายการ
  - จะเห็นว่ามีข้อมูลการชำระเงิน และการจัดส่งอยู่ใน order
  - ใน Order จะมี 3 แบบย่อย
    - สั่งทันที
    - สั่งล่วงหน้า
    - Catering



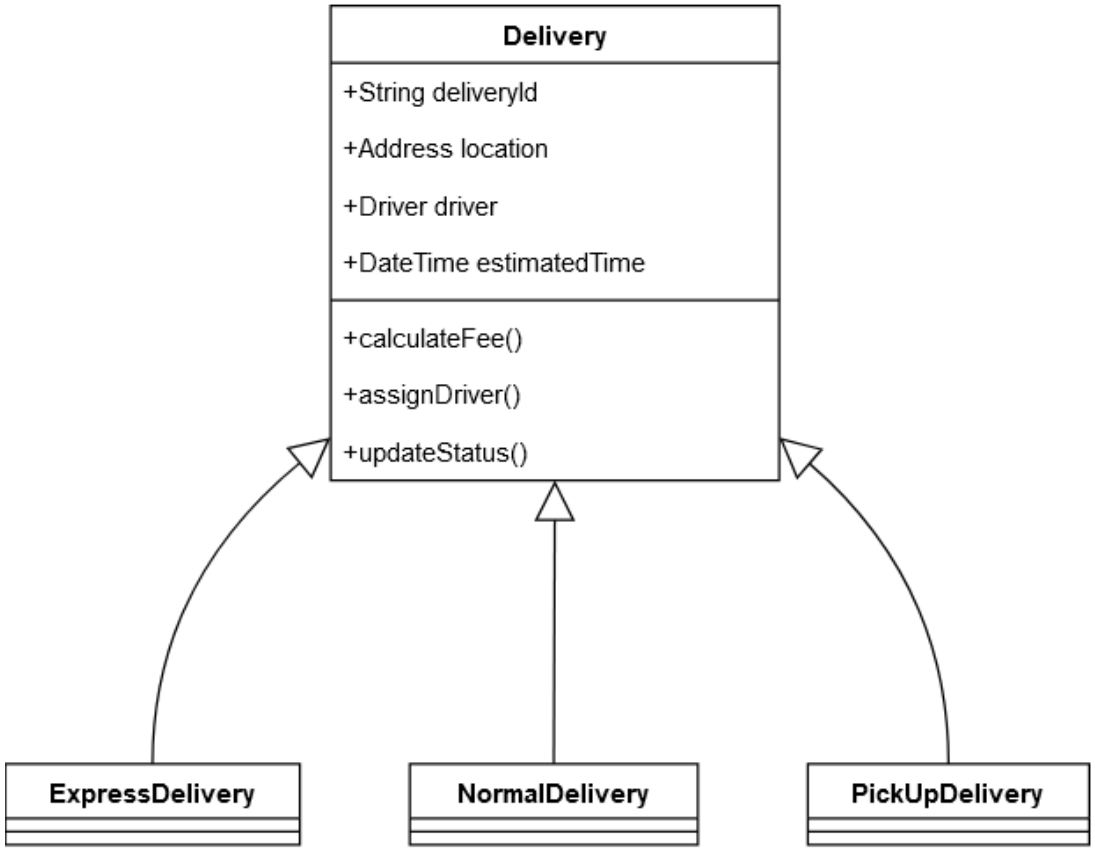
เฉลย

- ระบบชำระเงิน มี 4 subclass ได้แก่ ชำระแบบเงินสด ชำระแบบบัตรเครดิต ชำระแบบ E-waller และแบบใช้คูปอง payment จะถูกใช้โดย Order



เฉลย

- ระบบจัดส่งอาหารจะมี 3 แบบ คือ แบบด่วน แบบปกติ และแบบรับเองที่ร้าน



เฉลย

- จะเห็นว่าการออกแบบโดยใช้ Inheritance ช่วยให้การทำระบบย่อยง่ายมากขึ้น
- ระบบคำนวณราคาที่ต่างกัน เมื่อมี Subclass ก็สามารถจะเขียน overriding method ที่ต่างกันสำหรับแต่ละเมนูอาหารได้
- ในอนาคตอาจมีระบบระบบตรวจสอบความพร้อมในการขาย เช่น อาหารประเภทเครื่องดื่มก่อนรับ order จะตรวจสอบว่าพร้อมขายหรือไม่
  - วัตถุดิบพื้นฐาน - น้ำ - นม - น้ำแข็ง
  - ส่วนประกอบเพิ่มเติม - ไข่มุก - วิปครีม - ไซรัป
  - อุปกรณ์ - เครื่องชงกาแฟ - เครื่องปั่น - แก้ว
  - ตัวอย่าง: ชานมไข่มุก - น้ำชาพร้อม ✓ - นมพร้อม ✓ - ไข่มุกหมด ✗ ⟶ ไม่พร้อมขาย (แนะนำเมนูอื่น)

เฉลย

- หรือหากมีระบบการคำนวณเวลาในการเตรียมอาหาร เช่น เครื่องดื่ม
  - เวลาเตรียม = เวลาชง/ปั่น + เวลาเพิ่มท็อปปิ้ง + เวลาแต่ง
  - ตัวอย่าง: ชานมไข่มุก 1. ชง/ปั่น (3 นาที) - ชงชา - ผสมนม - เขย่า/ปั่น 2. เพิ่มท็อปปิ้ง (2 นาที) - ต้มไข่มุก - เพิ่มไข่มุก 3. แต่ง (1 นาที) - ใส่วิปครีม - โรยท็อปปิ้ง รวม = 6 นาที
- และอาจมีปัจจัยอื่นที่เปลี่ยนแปลงเวลาเตรียมอาหาร
  - ความยุ่งของร้าน
  - ความซับซ้อนของออเดอร์
  - ฯลฯ

For your attention