01076105, 01076106

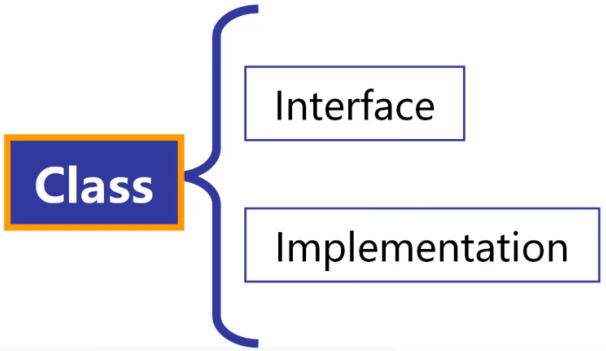
Object Oriented Programming

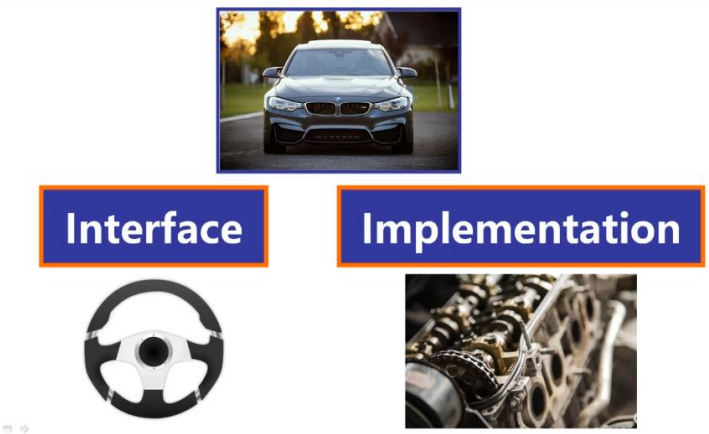
Object Oriented Programming Project

Polymorphism

Abstraction

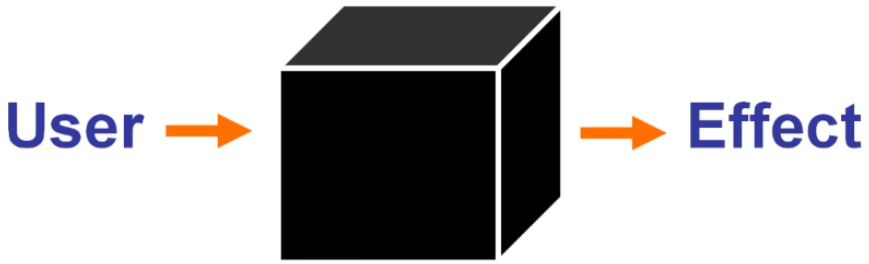
- คำว่า Abstraction มาจากรากศัพท์ Latin 2 คำคือ abs หมายถึง away from และ trahere หมายถึง draw ซึ่งเมื่อแปลรวมกันจะได้ เป็นกรรมวิธีที่กำจัดหรือเอาลักษณะเฉพาะบางประการออกไป เพื่อทำให้เหลือเฉพาะลักษณะเฉพาะเท่าที่จำเป็นเท่านั้น
- Abstraction มีนัย 2 ความหมาย ความหมายแรก คือ การ Encapsulation ทำให้ภายนอกมองเห็นเพียงการใช้งานคลาส คือ เห็น Interface แต่ไม่เห็น Implementation

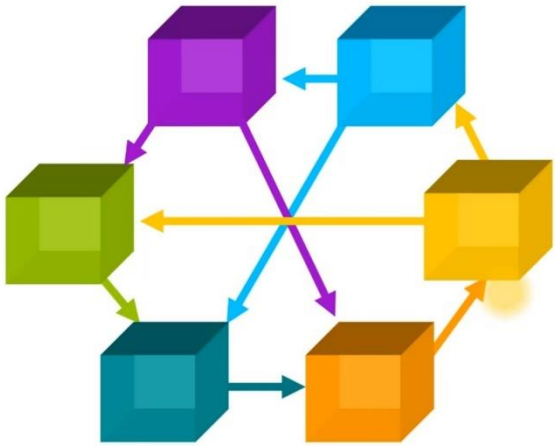




Abstraction

- เราอาจมอง Class หรือ Object เป็น กล่องดำ ที่เห็นเฉพาะผลการทำงาน เมื่อใส่ Input หรือ message เข้าไป
- การมองเช่นนี้ เป็นการแยกโปรแกรมออกเป็นส่วนๆ โดยแต่ละส่วนจะมีการทำงานเบ็ดเสร็จของตนเอง เมื่อเรียกใช้แต่ละส่วน (Module) ก็ทำผ่าน Interface โดยไม่จำเป็นต้องทราบวิธีการทำงานภายใน ดังนั้นการทำงานอาจมองได้ว่าเป็น Message ที่ส่งระหว่าง Object



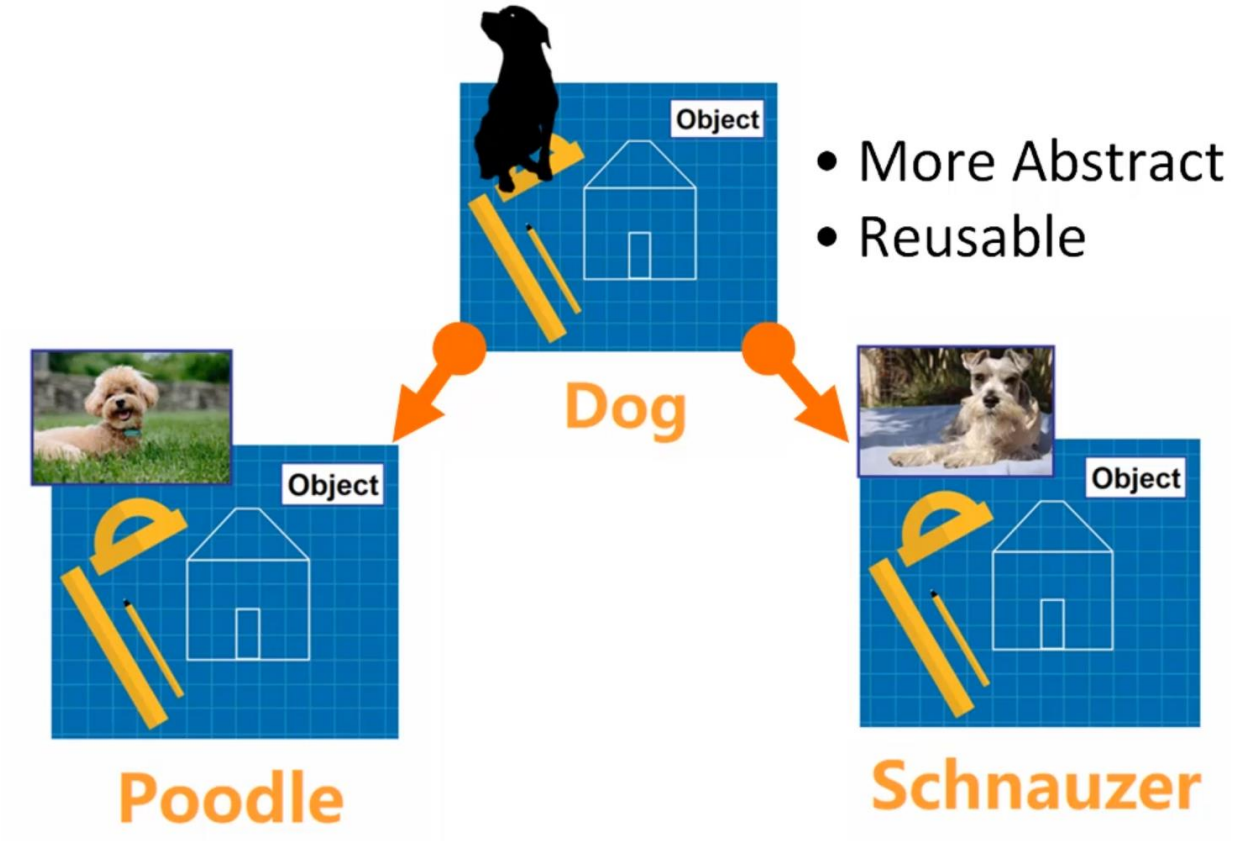


Abstraction

- ทำให้สามารถเปลี่ยนแปลงการทำงานภายใน ของแต่ส่วนได้โดยไม่กระทบกับโปรแกรมส่วนอื่น ตราบใดที่ Interface ยังคงเหมือนเดิม

Abstraction

- นัยของคำว่า Abstraction ที่ 2 หมายถึง การนำสิ่งที่คล้ายกันมารวมกัน เพื่อลดความซ้ำซ้อน ซึ่งเป็นแนวคิดที่ทำให้เกิด Inheritance



Polymorphism

- คุณสมบัติข้อที่ 4 ของ OOP คือ polymorphism
- คำนี้ มาจากคำว่า “poly” แปลว่าหลาย กับ “morph” แปลว่าเปลี่ยน รวมแล้วแปลว่า เปลี่ยนได้หลายแบบ
- คำนี้หมายถึง การที่ operation หรือ method ใดๆ สามารถใช้งานกับ object ที่หลากหลายได้ เช่น จากรูป จะเห็นได้ว่า operator * สามารถใช้ได้กับ ตัวเลข string list โดยมีพฤติกรรมที่แตกต่างกันไปในแต่ละประเภท

```
l = [2, "1", 3, 4, [1]]

for shape in l:
    print(shape*2)
```

Polymorphism

- อีกตัวอย่าง คือ การเล่นไฟล์เพลง audio_file.play()

```
class AudioFile:
    def __init__(self, filename):
        if not filename.endswith(self.ext):
            raise Exception("Invalid file format")
        self.filename = filename

class MP3File(AudioFile):
    ext = "mp3"
    def play(self):
        print("playing {} as mp3".format(self.filename))

class WavFile(AudioFile):
    ext = "wav"
    def play(self):
        print("playing {} as wav".format(self.filename))

mp3 = MP3File("myfile.mp3")
wav = WavFile("myfile.wav")
mp3.play()
wav.play()
```

Polymorphism

- polymorphism จะเกิดขึ้นเมื่อ **Interface** ของคลาส **เหมือนกัน**
- คลาส AudioFile เป็น base class มี constructor แต่คลาส MP3File และ WavFile ที่ Inherit มา ไม่มี constructor จึงใช้ constructor ของ base class
- Constructor มีหน้าที่ตรวจสอบชนิดของไฟล์ และ กำหนดชื่อไฟล์
- จะเห็นว่า method play() ในแต่ละคลาสจะทำงานต่างกัน ในไฟล์แต่ละประเภท
- ดังนั้นไฟล์แต่ละประเภท จึงถูกจัดการด้วยวิธีการที่แตกต่างกัน เรียกว่า polymorphism
- ความสามารถ polymorphism จะเกิดขึ้นได้ ต้องมี Inheritance มาก่อน
- เราอาจกล่าวได้ว่า polymorphism คือการใช้ Interface ร่วมกันของข้อมูลที่แตกต่างกัน

Polymorphism : magic method

- เราอาจออกแบบให้ Class ของเรา ตอบสนองกับ เครื่องหมาย +-*/ หรือ in
- หรือตอบสนองกับ subscript หรือ slice หรือ loop (คล้ายกับ list) ได้
- ฟังก์ชัน __add__ อยู่ในกลุ่มที่เรียกว่า magic method หรือ dunder (double under)

```
>>> var1 = 'hello '
>>> var2 = 'world'
>>> var1 + var2
'hello world'
>>> var1.__add__(var2)
'hello world'
```

Polymorphism : magic method

- สร้างคลาสที่เมื่อกำหนดความยาวด้วยหน่วยหนึ่ง สามารถแสดงในหน่วยอื่นได้

```
class Length:
    __metric = {"mm" : 0.001, "cm" : 0.01, "m" : 1, "km" : 1000,
                "in" : 0.0254, "ft" : 0.3048, "yd" : 0.9144,
                "mi" : 1609.344 }
    def __init__(self, value, unit = "m" ):
        self.value = value
        self.unit = unit
    def Converse2Metres(self):
        return self.value * Length.__metric[self.unit]
    def __add__(self, other):
        l = self.Converse2Metres() + other.Converse2Metres()
        return Length(l / Length.__metric[self.unit], self.unit)
    def __str__(self):
        return str(self.Converse2Metres())
    def __repr__(self):
        return "Length(" + str(self.value) + ", '" + self.unit + "')"
```

Polymorphism : magic method

- เรียกใช้งาน

```
x = Length(4)
print(x)
z = Length(4.5, "yd") + Length(1)
print(repr(z))
print(z)
```

- ผลการทำงาน

```
4
Length(5.593613298337708, 'yd')
5.1148
>
```

Polymorphism : magic method

- ใน constructor จะรับมูลความยาวและหน่วย หากไม่กำหนดจะให้หน่วยเป็น m
- ฟังก์ชัน Converse2Metres ทำหน้าที่แปลงจากหน่วยอื่นๆ เป็นเมตร
- ฟังก์ชัน __add__ เป็น magic method ซึ่งจะเรียกใช้งานเมื่อกระทำ operator “+” ระหว่าง object ในคลาสเดียวกันและส่งกลับเป็น object
- ฟังก์ชัน __str__ เป็น magic method ซึ่งจะเรียกใช้งานเมื่อใช้กับคำสั่ง print หรือ str ทำหน้าที่แสดงข้อมูลที่เก็บที่แปลงเป็นเมตรแล้ว
- ฟังก์ชัน __repr__ เป็น magic method ซึ่งจะถูกเรียกใช้งานเมื่อใช้กับคำสั่ง repr หรือเมื่อเรียก method ใน interpreter ทำหน้าที่แสดงความยาวในหน่วยที่กำหนด

Polymorphism : magic method

- จากโปรแกรม เนื่องจาก method __add__ จะรับข้อมูลเป็น object จึงไม่สามารถไปบวกกับข้อมูล type อื่นได้ เช่น เขียนว่า

```
x = Length(1) +1
```

- ก็จะเกิด Error ดังนั้นควรแก้ไข __add__ ให้เป็นดังนี้ ก็จะทำงานได้ การทำงานคือ ตรวจสอบชนิดข้อมูลก่อน หากเป็นชนิดอื่นก็แปลงก่อนแล้วค่อยบวก

```
def __add__(self, other):
    if type(other) == int or type(other) == float:
        l = self.Converse2Metres() + other
    else:
        l = self.Converse2Metres() + other.Converse2Metres()
    return Length(l / Length.__metric[self.unit], self.unit)
```

Polymorphism : magic method

- แต่ก็จะยังไม่สามารถทำงานในกรณีนี้ได้ เนื่องจาก ใช้กับ += ไม่ได้

```
x += Length(1)
```

- กรณีนี้ต้องใช้ __radd__


```
def __radd__(self, other):
    if type(other) == int or type(other) == float:
        l = self.Converse2Metres() + other
    else:
        l = self.Converse2Metres() + other.Converse2Metres()
    return Length(l / Length.__metric[self.unit], self.unit)
```

- แม้ในโปรแกรมจะเขียนเหมือนกัน แต่ผลการทำงานจะต่างกัน

Polymorphism : magic method

Binary Operators

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__truediv__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)
^	object.__xor__(self, other)
|	object.__or__(self, other)

Extended Assignments

Operator	Method
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other)
**=	object.__ipow__(self, other[, modulo])
<<=	object.__ilshift__(self, other)
>>=	object.__irshift__(self, other)
&=	object.__iand__(self, other)
^=	object.__ixor__(self, other)
|=	object.__ior__(self, other)

Polymorphism : magic method

Unary Operators

Operator	Method
-	object.__neg__(self)
+	object.__pos__(self)
abs()	object.__abs__(self)
~	object.__invert__(self)
complex()	object.__complex__(self)
int()	object.__int__(self)
long()	object.__long__(self)
float()	object.__float__(self)
oct()	object.__oct__(self)
hex()	object.__hex__(self

Comparison Operators

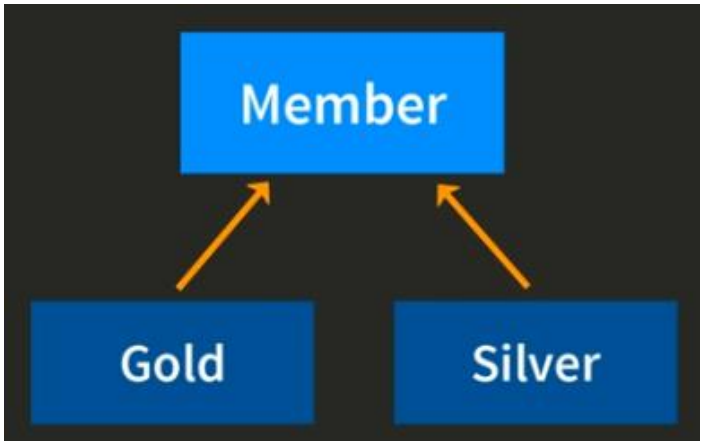
Operator	Method
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)

Polymorphism : magic method

- __getitem__ ใช้สำหรับดึงข้อมูลที่กำหนด
- __getslice__ ใช้สำหรับดึงข้อมูลในช่วงที่กำหนด
- __contains__ ใช้สำหรับ operator “in”
- https://rszalski.github.io/magicmethods/
- https://www.analyticsvidhya.com/blog/2021/08/explore-the-magic-methods-in-python/

Abstract Base Class

- Abstract base class อาจเรียกว่า **คลาสต้นแบบ** เพราะเป็นคลาสที่ไม่ใช้สำหรับสร้าง Instance โดยตรง แต่ทำหน้าที่เป็นโครงสำหรับคลาสที่สืบทอดไป เพื่อให้คลาสที่สืบทอดไป ต้อง implement ตามที่กำหนด
- ยกตัวอย่างระบบสมาชิก แบ่งเป็นสมาชิก 2 ประเภท คือ Gold กับ Silver โดยกำหนดว่าในคลาสที่สืบทอดไปจะต้อง implement เรื่องส่วนลด



Abstract Base Class

```python
from abc import ABC, abstractmethod

class Member(ABC):
    def __init__(self, m_id, fname, lname):
        self.m_id = m_id
        self.fname = fname
        self.lname = lname

    @abstractmethod
    def discount(self):
        pass

    def full_name(self):
        return "{} {}".format(self.fname, self.lname)

class Gold(Member):
    def discount(self):
        return .10

class Silver(Member):
    def discount(self):
        return .05
```

Abstract Base Class

- ในการใช้ abstract base class จะต้อง import จาก library ชื่อ abc
- คลาสใดที่ inherit จาก ABC จะไม่สามารถสร้าง instance ของคลาสนั้นได้ หากพยายามสร้าง instance จะเกิด Type Error
- Decorator @abstracmethod หากใช้กับ method ใด จะเป็นการบังคับว่าคลาสที่ inherit จากคลาสนั้นไป จะต้อง implement method นั้นใหม่เสมอ
- จะเห็นได้ว่าคลาส Gold และ Silver ที่ inherit มาจากคลาส Member ก็จะถูกบังคับให้ implement method ชื่อ discount
- ทั้งนี้เพื่อให้เกิด interface ที่เหมือนกันหมดสำหรับทุกคลาสที่ inherit มาจากคลาสที่เป็น abstract base class

Abstract Base Class

- นอกเหนือจาก abstract method แล้วยังมี abstract property อีกด้วย
- Class MediaLoader จะทำหน้าที่เป็น base class โดยบังคับให้ต้องกำหนด attribute ext และ method play

```
from abc import ABC, abstractmethod, abstractproperty

class MediaLoader(ABC):
    @abstractmethod
    def play(self):
        pass

    @abstractproperty
    def ext(self):
        pass
```

Abstract Base Class

- จะเห็นว่าเมื่อคลาส Ogg ทำการ Inherit มาจากคลาส MediaLoader
- จะต้องมีการกำหนด attribute ext และ method play มิฉะนั้นจะ Error

```
class Ogg(MediaLoader):
    ext = '.ogg'

    def play(self):
        pass

o = Ogg()
```

For your attention