程式設計概論
Programming 101
—物件導向(object oriented)

授課老師：邱淑怡

Date:5/31/2021

Outline

- 認識物件導向(object oriented)
- 名詞定義
- 物件導向程式(object oriented programming, OOP)特色

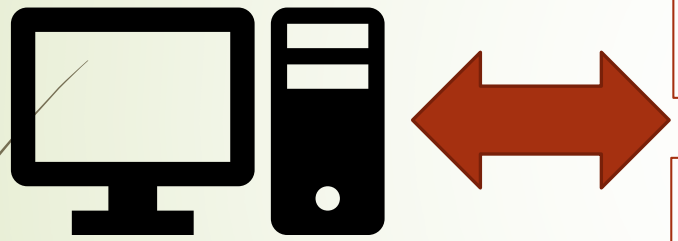
常見名詞定義

- 物件(object)或實體(instance)就像在生活中所看到的各種物體，如：房子、電腦、手機...
  - 這些物件可能又是由許多子物件所組成，如**電腦**又是由硬碟、CPU、主機板、記憶體...等所組成
  - Windows作業系統中的**視窗**是一個物件，它是由標題列、功能表列、工具列等組成
  - 在Python中，物件是資料與程式碼的組合，它可以是整個應用程式或應用程式的一部分

常見名詞定義(cont.)

- 屬性(attribute)或成員變數(member variable)用來描述物件的特質
  - **電腦**的CPU等級、硬碟空間、製造廠商等用來描述電腦的特質就是這物件的屬性
  - **視窗**的大小、位置等用來描述視窗的特質就是這個物件的屬性
- 方法(method)或成員函式(member function)是用來定義物件的動作
  - 電腦的開機、關機、執行應用程式等動作就是這物件的方法

常見名詞定義(cont.)



Attribute (屬性)
- CPU: Intel Core i7
- Memory: 16GB
- Hard Disk: 1TB

Method(方法)
- Boot
- Shutdown
- Execute

常見名詞定義(cont.)

- 類別(class)是物件的分類，屬於相同類別的物件具有相同的屬性和方法，但屬性的值則不一定相同
  - car是一個類別，它有廠牌、顏色、型號等屬性；開門、關門、發動等是方法

物件導向程式(object oriented programming, OOP)特點

- 封裝(encapsulation): 將處理與用來處理資料的函式放在一起成為一個類別，稱為封裝，著重物件與物件間的操作
  - 類別內部的資料與函式可以設定存取層級，如: 設定私有屬性或私有方法，限制只有類別內部的敘述可以存取
- 繼承(inheritance): 指從既有的類別(parent class)定義出新的類別(child class)，提高軟體的重複使用性
- 多型(polymorphism): 指當不同的物件收到相同的訊息時，會以各自的方法來做處理

圖例說明

父類別:
起飛、降落

子類別:
利用燃料產生熱氣來起飛或降落

子類別:
利用螺旋槳來起飛或降落

子類別:
利用噴射引擎來起飛或降落

舉例說明: 建立可以計算圓面積的類別

- 建立一個類別是Circle物件，令半徑統一初始值為1，之後再將半徑設定為想要的數值。
- 類別的第一個參數是self，之所以要有self參數，是要讓方法內的敘述透過這個參數存取物件的屬性。

宣告類別

```
class Circle:
    PI = 3.14
    radius = 1

    def getArea(self):
        return self.PI * self.radius * self.radius
```

function v.s. class

```
def check_leap(year):
        if(year%400==0) or (year%100!=0 and year%4==0):
                print("leap year")
        else:
                print("common year")
```

舉例說明: 建立可以計算圓面積的類別(cont.)

- 建立Circle物件，令半徑統一初始值為1，之後再將半徑設定為想要的數值。

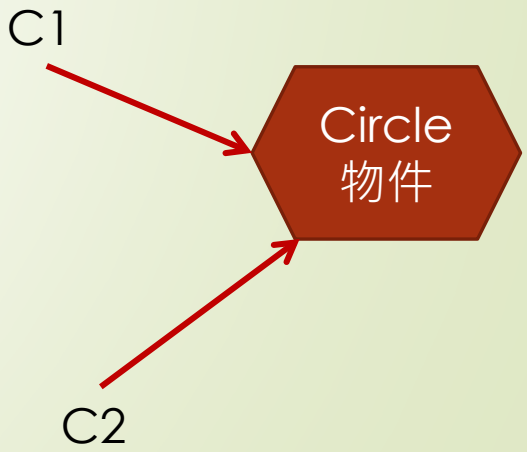
宣告類別

```
class Circle:
    PI = 3.14
    radius = 1

    def getArea(self):
        return self.PI * self.radius * self.radius
```

使用類別

```
C1 = Circle()
print(C1.radius)
print("半徑為", C1.radius, "的圓面積為", C1.getArea())

C2 = C1
C2.radius = 10
print("半徑為", C1.radius, "的圓面積為", C1.getArea())
```

變數C2參照變數C1所參照的Circle物件
兩者參照相同的物件



__init()__方法

- Python允許類別提供一個名稱為__init__()的特殊方法，在建立物件時，會自動呼叫這個方法將物件初始
- 常見的初始化動作
  - 資料、變數的初始值
  - 開啟檔案
  - 建立資料庫連接
  - 建立網路連線
  - …

__init()__方法 (cont.)

- 第一個參數必須是self，參照剛被建立的物件本身
- 前面的例子，用__init__()將半徑統一用初始值1，之後再將半徑設定為想要的數值

宣告類別

```
class Circle:
    PI = 3.14

    def __init__(self, r = 1):
        self.radius = r

    def getArea(self):
        return self.PI * self.radius * self.radius
```

使用類別

```
C1 = Circle()
print("半徑為", C1.radius, "的圓面積為", C1.getArea())

C2 = Circle(10)
print("半徑為", C2.radius, "的圓面積為", C2.getArea())
```

私有成員(私有屬性和私有方法)

- 前面例子，類別外部的敘述都能直接存取類別內部的資料，在實務上這種設計並不好，因有些資料需要被保護，如:成績或薪資，且不應該允許任何敘述都能直接存取，故需要將這些資料設定為私有屬性(private attribute)，限制只有類別內部的敘述才能夠存取
- For python programming,私有屬性的名稱前面有加上兩個底線，但名稱後面不能有底線，例如: __radius為私有屬性

私有屬性實例說明

- __radius為私有屬性，需透過getRadius()取得半徑的值

宣告類別

```
class Circle:
    PI = 3.14

    def __init__(self, r = 1):
        self.__radius = r

    def getRadius(self):
        return self.__radius

    def getArea(self):
        return self.PI * self.__radius * self.__radius
```

私有屬性實例說明(cont.)

宣告類別

```
class Circle:
    PI = 3.14

    def __init__(self, r = 1):
        self.__radius = r

    def getRadius(self):
        return self.__radius

    def getArea(self):
        return self.PI * self.__radius * self.__radius
```

使用類別

```
C1 = Circle(10)
print("C1的半徑為", C1.getRadius())
print("C1的圓面積為", C1.getArea())
```

練習題

- 有個Employee類別如下，用來表示員工的姓名及薪資，為保護姓名與薪資不受到隨意更改，將這兩者設為私有變數，如何取得這兩位員工的姓名及薪資？

```
class Employee: #父類別
    def __init__(self, name):
        self.__name = name

    def getName(self):
        return self.__name

    def setSalary(self, basic, bonus = 0):
        self.__salary = basic + bonus

    def getSalary(self):
        return self.__salary
```

```
E1 = Employee("John")
E2 = Employee("Alan")
E1.setSalary(22000)
E2.setSalary(42000, 6000)
# print
```

練習題_參考程式

- 有個Employee類別如下，用來表示員工的姓名及薪資，為保護姓名與薪資不受到隨意更改，將這兩者設為私有變數，如何取得這兩位員工的姓名及薪資？

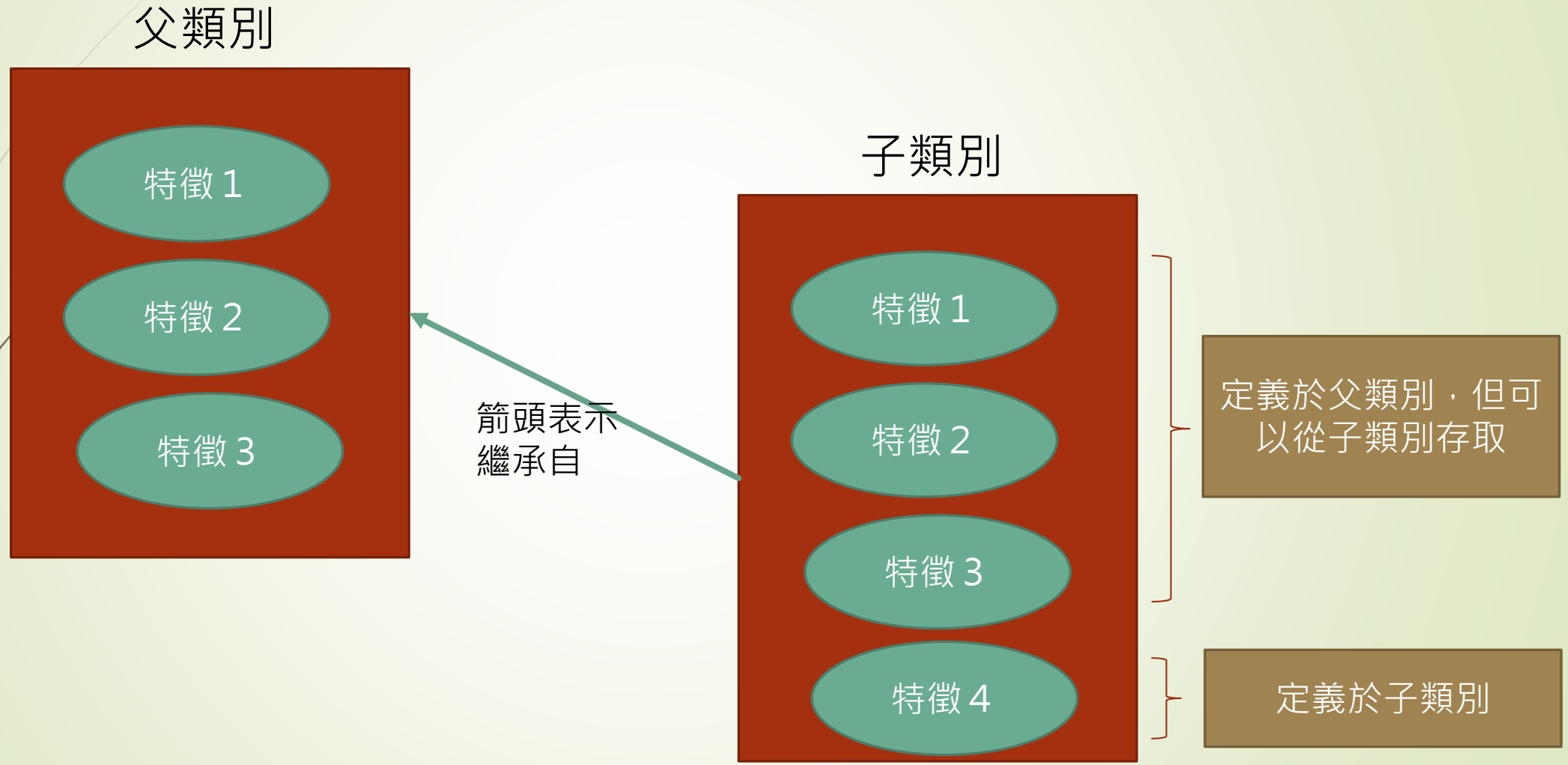
```
class Employee:    #父類別
    def __init__(self, name):
        self.__name = name

    def getName(self):
        return self.__name

    def setSalary(self, basic, bonus = 0):
        self.__salary = basic + bonus

    def getSalary(self):
        return self.__salary
```

```
E1 = Employee("John")
E2 = Employee("Alan")
E1.setSalary(22000)
E2.setSalary(42000, 6000)
print("員工", E1.getName(), "的薪水", E1.getSalary())
print("員工", E2.getName(), "的薪水", E2.getSalary())
```

繼承(inheritance)

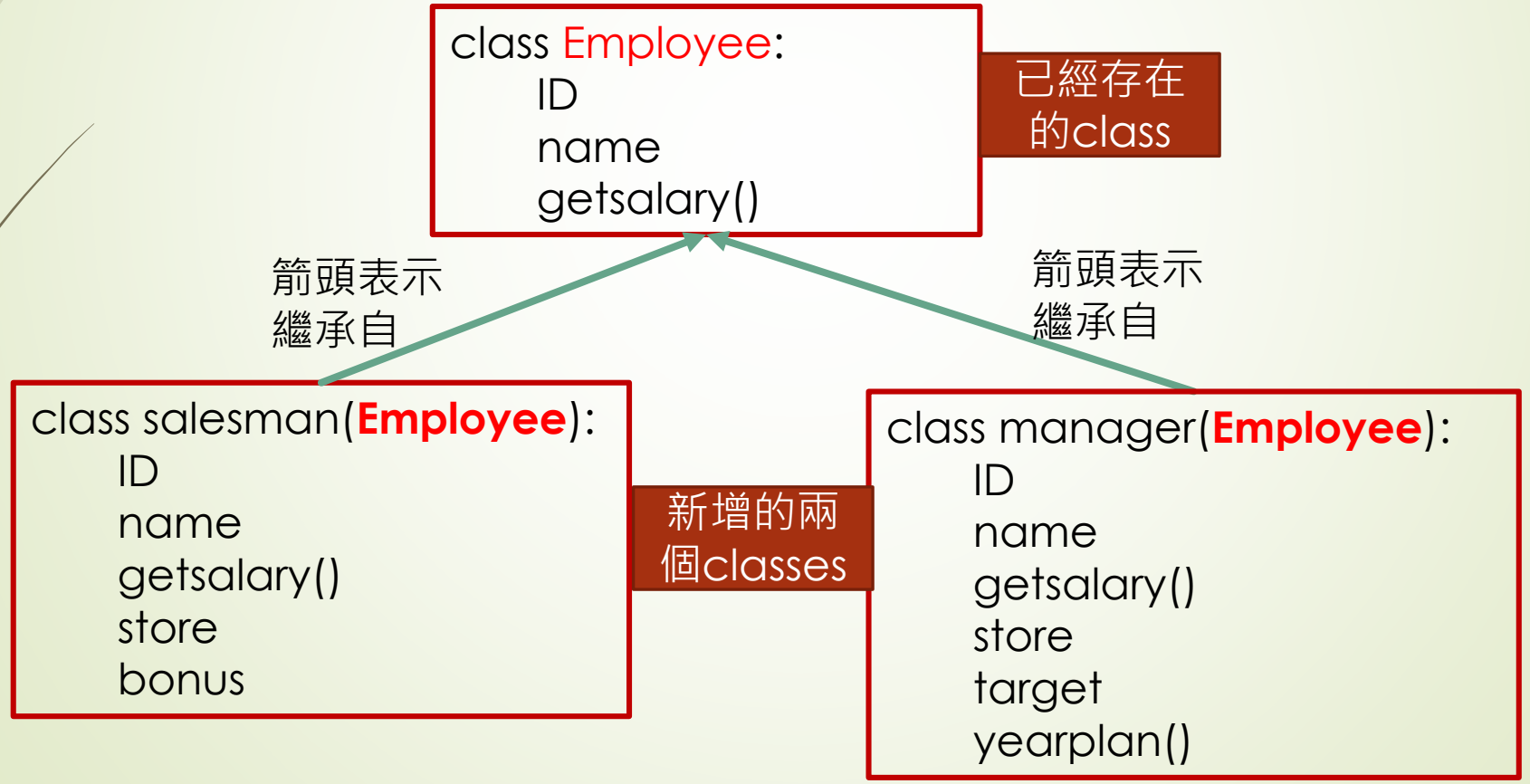
- 指從既有的類別(parent class)定義出新的類別(child class)，提高軟體的重複使用性
  - 既有的類別叫父類別，由於是用來作為基礎的類別，又稱基底類別(base class)或超類別(super class)
  - 新的類別則叫子類別，由於繼承自基底類別，又稱衍生類別或擴充類別
  - 子類別繼承了父類別的**非私有成員**，同時可以加入新的成員(包含attribute and method)或覆蓋繼承自父類別的方法，也就是將繼承父類別的方法重新定義，並且不會影響到父類別的方法
  - 上述的繼承關係即所謂的類別階層(class hierarchy)

繼承(cont.)


父類別
特徵 1
特徵 2
特徵 3
箭頭表示
繼承自
子類別
特徵 1
特徵 2
特徵 3
特徵 4
定義於父類別，但可
以從子類別存取
定義於子類別

繼承的實例說明

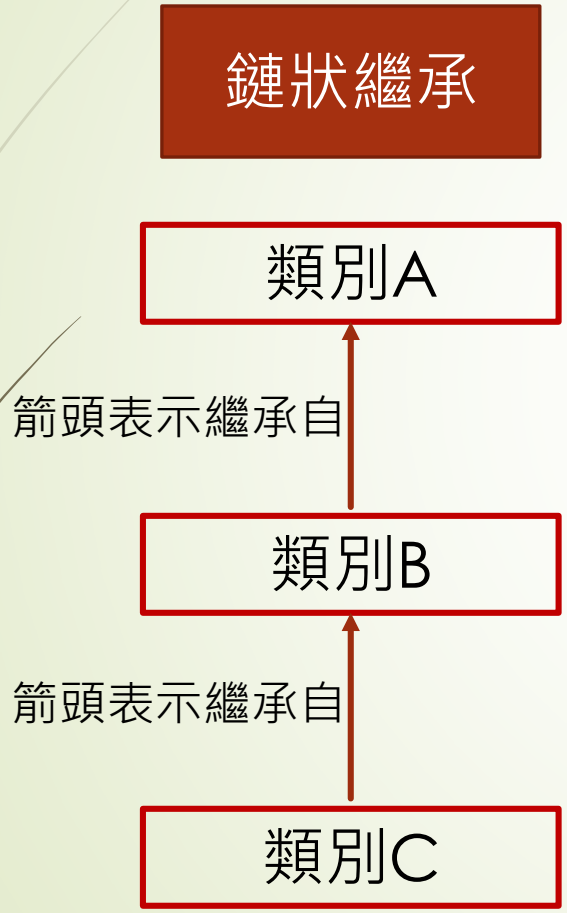
- 父類別是Employee
- 子類別是salesman, manager:分別繼承父類別的非私有成員，同時可加入新的成員或覆蓋繼承自父類別的方法

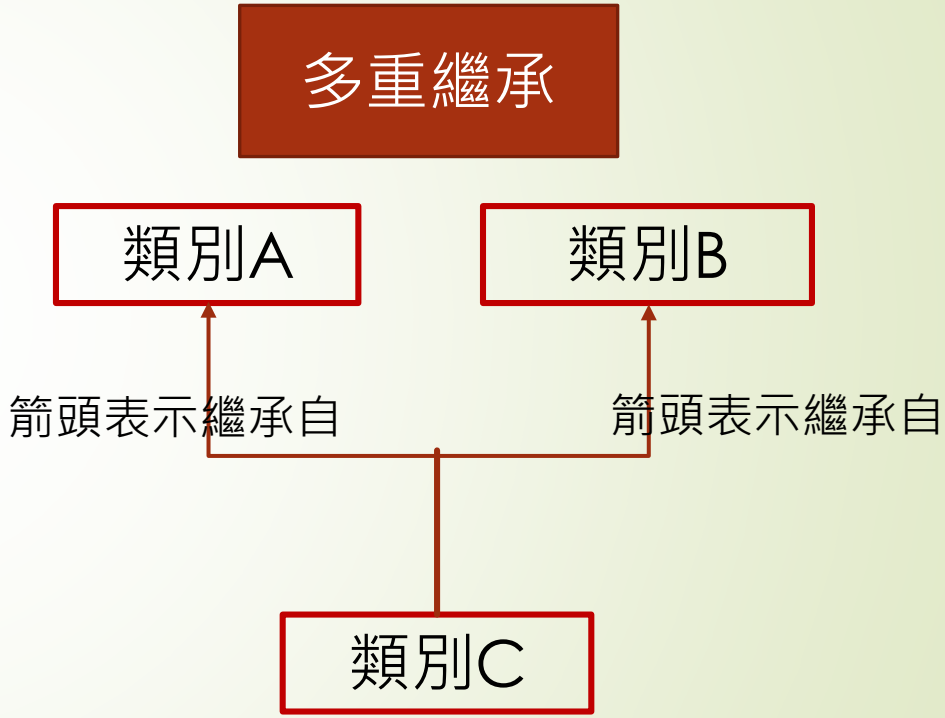


定義子類別: 類別B(child class)繼承於類別A (parent class)

```
class A:        # parent class
    __x = “我是屬性__x”   #定義私有屬性，無法被子類別繼承
    y = “我是屬性y”        #定義非私有屬性，能夠被子類別繼承

    def __M1(self):          #定義私有方法，無法被子類別繼承
        print("我是方法M1()")

    def M2(self):          #定義非私有方法，能夠被子類別繼承
        print("我是方法M2()")


class B(A):          #child class, 定義類別B繼承自類別A
    z = "我是屬性z"

    def M3(self):
        print("我是方法M3()")
```

不同繼承類型


鏈狀繼承
類別A
箭頭表示繼承自
類別B
箭頭表示繼承自
類別C


多重繼承
類別A
類別B
箭頭表示繼承自
箭頭表示繼承自
類別C

覆蓋繼承自父類別的方法: 實例說明

```
# 宣告父類別及子類別
class Employee:    #父類別
    def __init__(self, name):
        #初始化方法是設定員工姓名
        self.__name = name

    # 此方法傳回員工姓名
    def getName(self):
        return self.__name

    # 此方法傳回員工薪資
    def getSalary(self, hours, payrate):
        return hours * payrate

class salesman(Employee):  #子類別
        #傳回銷售人員的本月薪水(含業績獎金)
    def getSalary(self, hours, payrate, bonus):
        return hours * payrate + bonus
```

```
# 執行程式
E1 = Employee("Bob")
E2 = salesman("John")
print("員工", E1.getName(), "薪水", E1.getSalary(120, 150))
print("銷售人員", E2.getName(), "薪水", E2.getSalary(120, 150, 3000))
```

Python提供兩個與繼承相關的函式

- 語法如下:

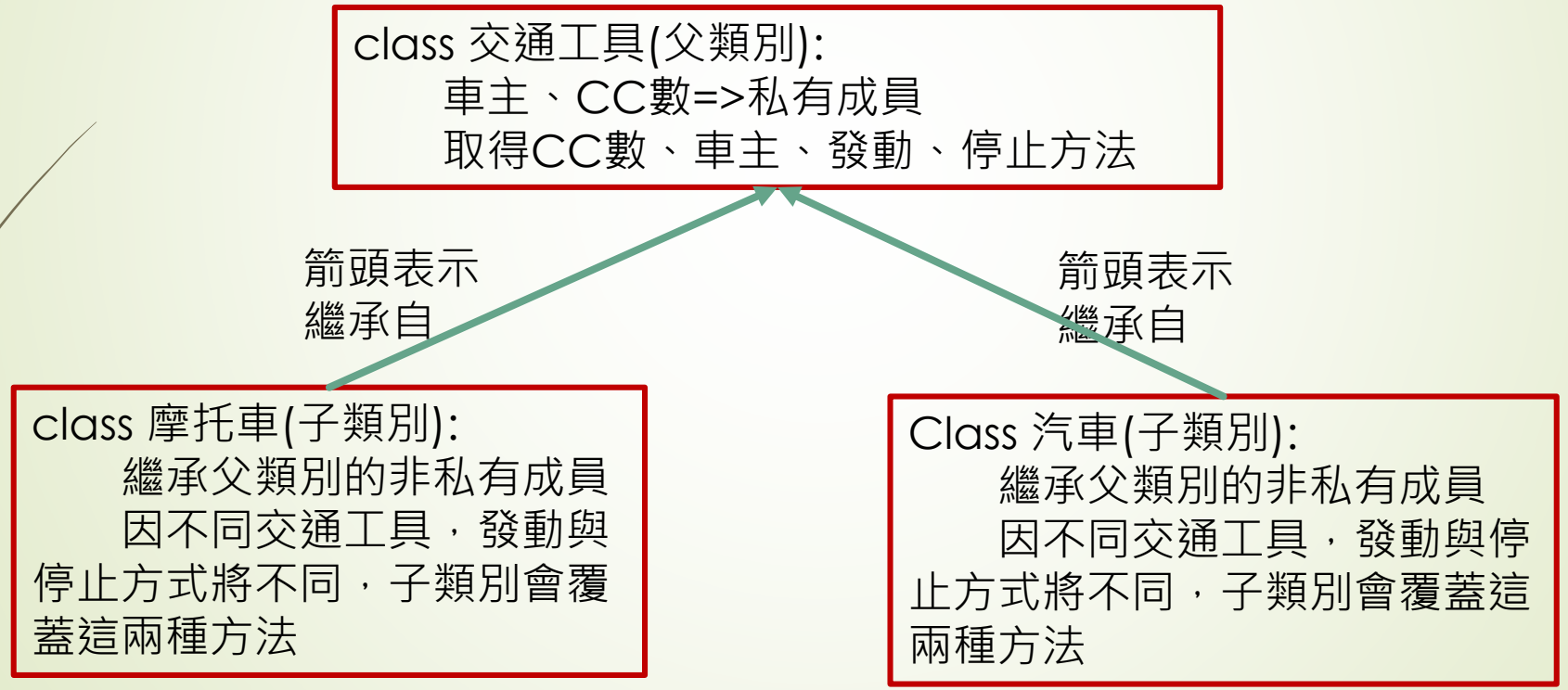
1. isinstance(obj, classinfo): 若參數obj是參數classinfo所指定之類別或其子類別的物件就傳回True
2. issubclass(class,classinfo):若參數class是參數classinfo所指定之類別或其子類別的物件就傳回True

```
isinstance(100,int)
isinstance(True,int)
```

```
class A:
    x=1
class B(A):
    y=2

Obj1=A()
Obj2=B()
print(isinstance(Obj1,A))
print(isinstance(Obj2,A))
print(issubclass(B,A))
```

多型

- 當不同的物件收到相同訊息時，會以各自的方法來做處理



Online Quiz