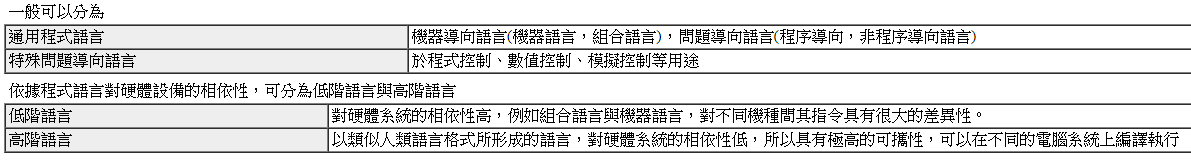
**第六章：物件、屬性、方法**

* 物件導向(Object-oriented programming, 簡稱OOP)在程式設計領域是一項非常重要的概念，也是java技術相當核心的一環及現今程式設計的主流，必須完整並清楚了解其運作原理或使用方式，才能將java技術發揚光大！
* 非物件導向語言一般已被視為是低階語言↓，使用物件導向可使程式語言更加直覺且人性化>高階語言，也是未來程式語言發展的主流
* <https://zh.wikipedia.org/wiki/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1> 維基百科的物件導向說明



1. **物件(Object)**

→物件導向的程式語言，顧名思義就是程式在運行時是將所有對象都視為是一個物件，並且透過物件來**傳遞**或**夾帶**資訊：

物件

送出請求

伺服器、接收方、後台etc.

抬頭、裝置資訊、簽章

資料內容

結果回覆

→類別vs物件，就好比設計圖與成品，架構一樣但成品樣式可以完全不同

(藍色XL)的t桖

設計圖(t桖)

(黑色s)的t桖

→物件有其自己的屬性與方法(動作)，並與其它物件或類別做溝通

方法溝通

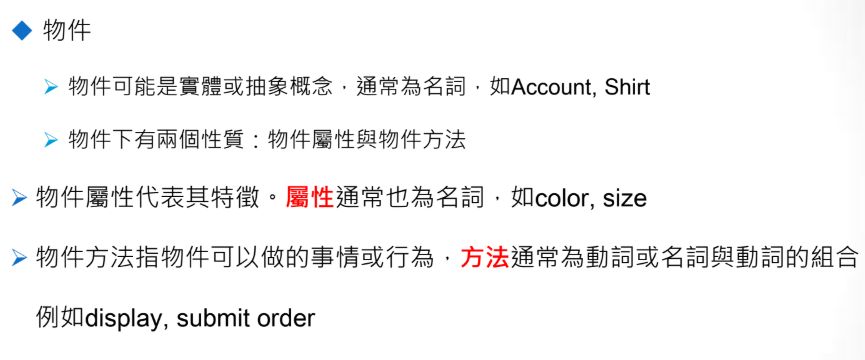
B物件

A物件

→物件的概念也可想像成RPG GAME，創建一個新角色就是建立新物件的意思，並會在初期給與其屬性(ex人物個性、外觀、喜好等等)，並給予特定的動作(ex人物的基本技能、職業技能、特殊能力之類)。不同的物件利用其自身不同的特性來與其它物件溝通互動，這就是程式的基本架構！

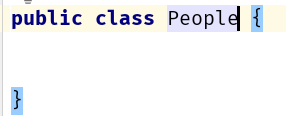
|  |  |  |
| --- | --- | --- |
| **物件(class)** | **屬性(field)** | **方法(method)** |
| 人類(小明) | 力量、敏捷、智力 | 大吼、全力一擊、逃跑 |

* 物件的命名習慣



現在來根據上述表格格式來建立一個新物件看看

* 建立類別



→結構為：修飾詞+class+類別名稱

→至此已經是完成的藍圖了，已經可以使用此類別建立該物件，只是尚未有屬性與方法

* 建立物件的寫法



→認識new關鍵字：在記憶體空間中新增一個存放於該物件的地方

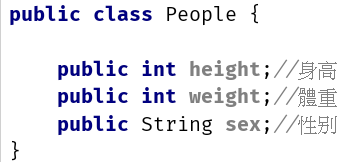
Ps：為新物件分配記憶體是一件非常嚴謹和複雜的任務，JVM的設計者們不僅需要考慮記憶體如何分配、在哪分配等問題，並且由於記憶體分配演算法與記憶體回收演算法密切相關，所以還要考慮GC執行完記憶體回收後是否會在記憶體空間中產生內部碎片， 但java使用者只要會使用就可以了，剩下的事就交給JVM吧

→物件皆需要實體化後才能開始執行動作

→執行動作的基本單位：物件(非常重要！！)

→david為物件名稱，new出來的物件名稱為該類別名稱+小刮號(屬性)→建構式用

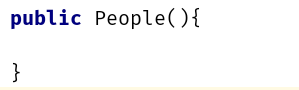
* 建立類別屬性



→屬性初始值可以設定也可以不設定，先設定的話在建立物件時就會同時附上

二 建構式(Constructor)

* 在類別內建立建構式，是為了新增該類別物件時能先在該物件賦與初始狀態，以便在新增物件同時擁有並賦予各項屬性數值的功能
* 建構式範例→修飾詞+類別名稱



→存取權限大多是public，為了提供建立物件時的初始化設定

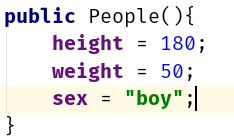
→這邊要注意的是：java在建立類別時就會自帶一個預設建構式default constructor

(body與參數內均無內容)如上圖所示，如果建立class時但沒有寫建構式，預設也會有個預設建構式。

→建構式名稱需要跟類別名稱相同！也是辨別該段程式碼是否為該類別建構式的特徵

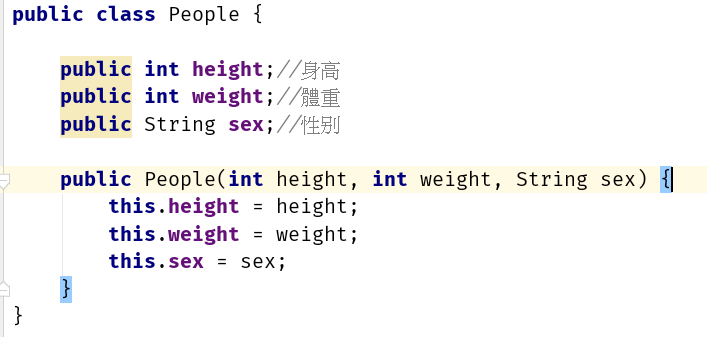
→如果有自行寫建構式的話，**預設建構式就會消失(非常重要!!!)**，除非再自行新增

* 自己建立建構式(參考用)



→物件建立初期就會決定所有屬性的數值，但較為不彈性(上述意思為，一旦建立people物件後，height與weight與sex的值就是固定的)，變成複製人了

* 帶有參數的建構式

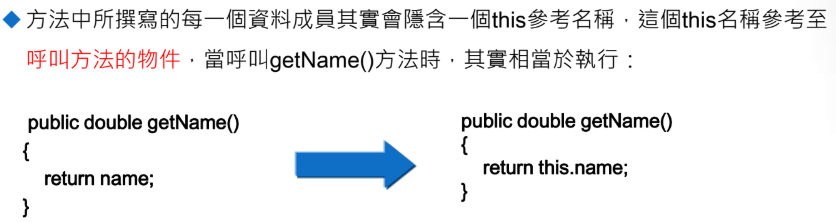


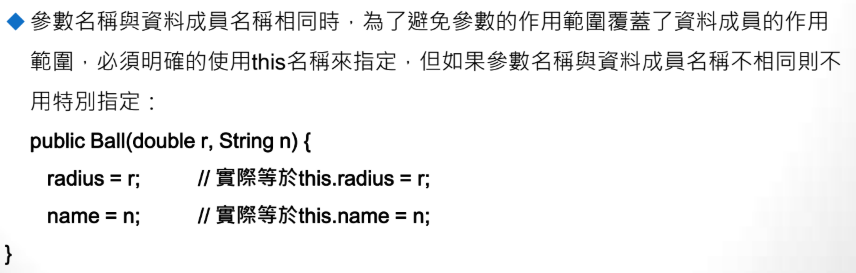
→物件建立時代入對應的內容，用來建立物件的屬性數值

* 補充說明：This的概念

→this即指自己本身，用於強調或是簡稱某些變數或該程式

→body內名稱一樣時用於區隔不同範圍的變數





→小刮號內參數意思為：建立物件時需要放入的數值，並規定型態，也可放物件、集合、陣列等等

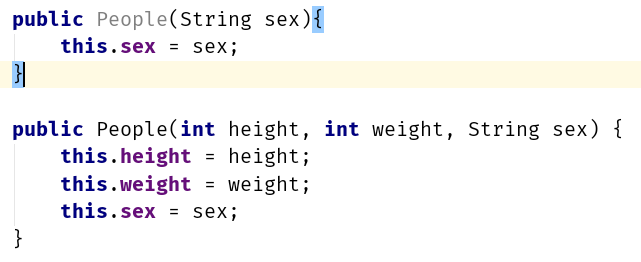
→參數代入後，即可賦予該物件鍵入的數值成為它的屬性

* 重新建立物件



→至此david物件有擁有了三個帶有數值的屬性了！

Ps：一類別的建構式可以有多種情況(多重建構式)



→代表建立該物件時，可選擇帶入多少參數成為何種物件

問題：此時該物件有預設建構式嗎？

答：

* 建立物件後，可隨時更改屬性，使用「物件名稱.屬性名稱」



→David的height最後會變成160！

→小數點為呼叫的意思

\*練習 建立一個車子的藍圖，並製作出兩台不同外觀顏色、幾人座、電動/汽油的車子。

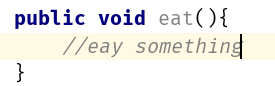
三 建立物件方法

* 回憶一下main方法結構

→層級+靜/動+回傳值+方法名稱(參數){ //方法內容// }

Public static void main(String [] args){ }

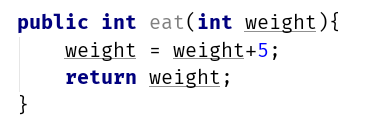
1 無回傳值方法



→方法多為動態的動作，所以取名時可以動詞為主，如eat, make, build, wait等等

→無回傳值時，表示方法執行完即結束

2 有回傳值的方法



int 型態

INTnt 型態

→回傳值要指定一種回傳型態(不一定是基本型態，也可回傳物件、集合、陣列等)

→只能指定回傳一種型態的東西，不能多個

→參數內可放入呼叫該方法時需要代入的值(材料)

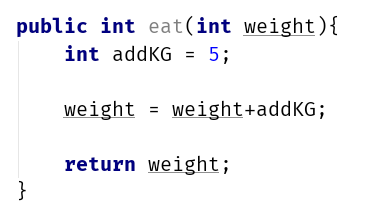
→回傳值的型態要與方法前面指定的型態一樣

→補充：null 與””的概念

**一物件值為null代表無參照值，或是物件尚未實體化，參照值為null的物件是不能使用的！**

**“”代表有參照值，值為””，但不是null，所以可以使用，但某些方法如果欄位不能是空值，可能會出錯**

* 變數的範圍



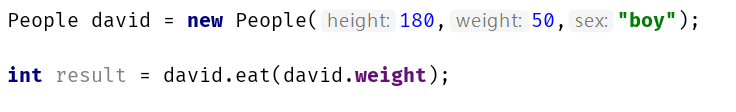
→傳進來的參數不等於外面的變數，即使名稱一樣(名稱只是代號)

→方法內才宣告的變數只有方法內有效，方法外的空間皆看不到此變數，也隨著方法結束而GC

→注意回傳值的內容，呼叫方法取值時，在方法外面或裡面盡量做額外處理(之後章節再說)

四 呼叫方法

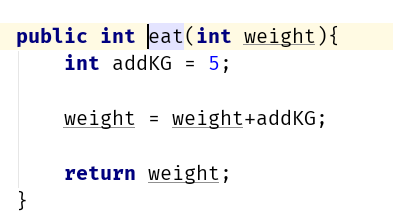
1. 方法中的程式運行方向



試算出輸出結果

* 程式運行方向

呼叫方法

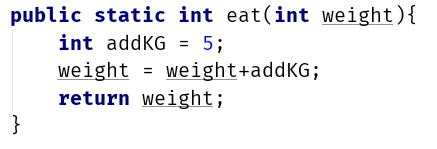


→david物件的weight數值會代入到方法內的weight變數內

→執行方法後的回傳結果代入給result變數做存儲

1. 類別方法

* 冠上static的方法，呼叫時也可不透過物件，經由類別名稱+方法名稱呼叫
* 靜態方法



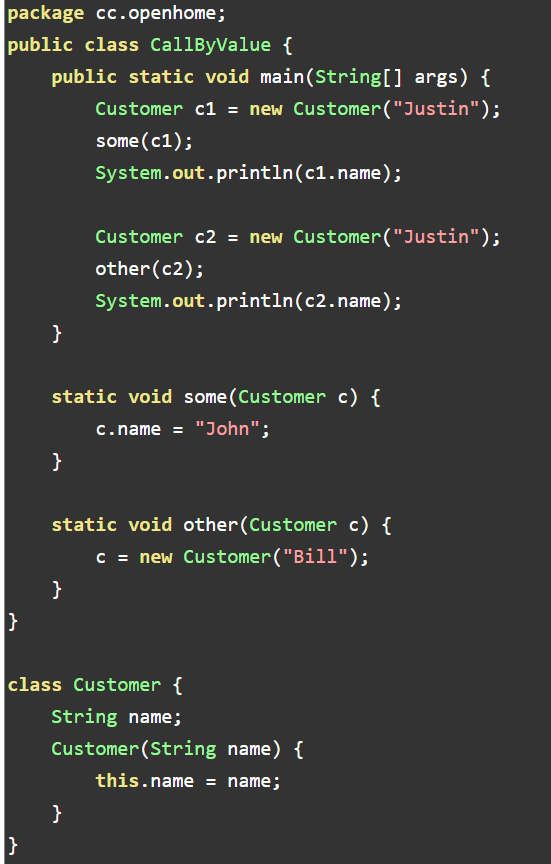
→呼叫方法時：

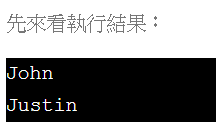


1. java的方法傳值呼叫

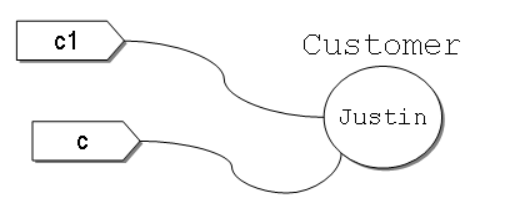
<https://openhome.cc/Gossip/Java/PassByValue.html>

→java本身在方法的傳遞為傳值呼叫(pass by value)，以下面例子來說明：

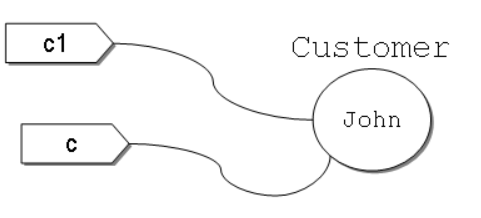




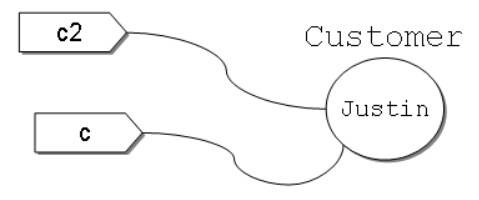
(1 呼叫some方法時，將c1參照的值傳給c，這時c1與c參照到的值是一樣的

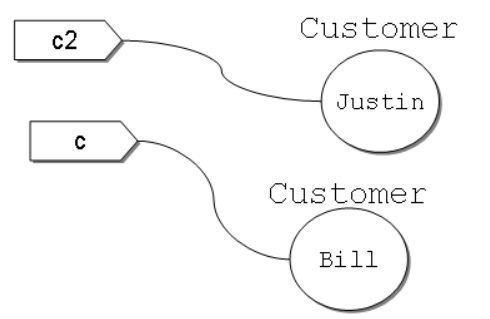


而後由c去改變參照值，方法結束後c就消失了，這時c1因為跟c參照到的值是一樣的，所以c1的參照值也會跟著改變

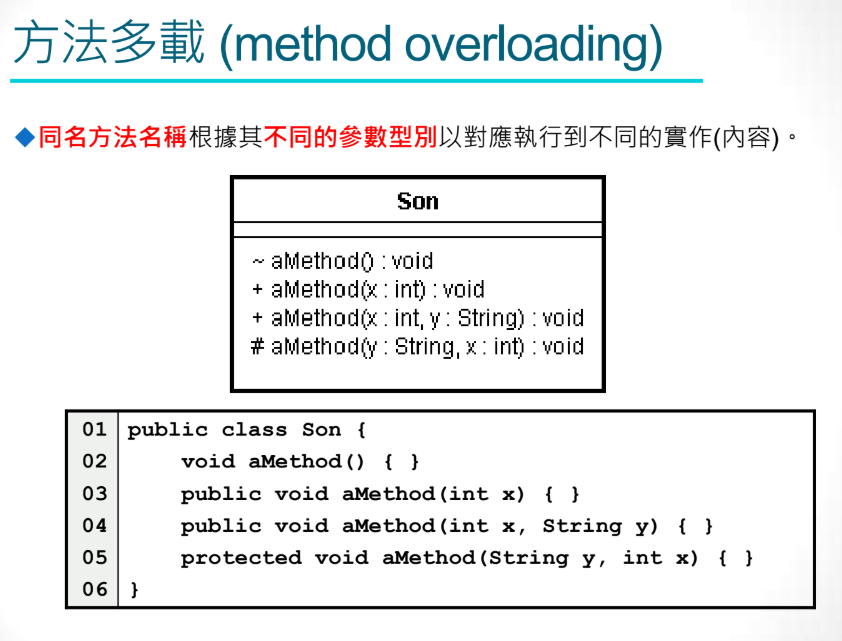


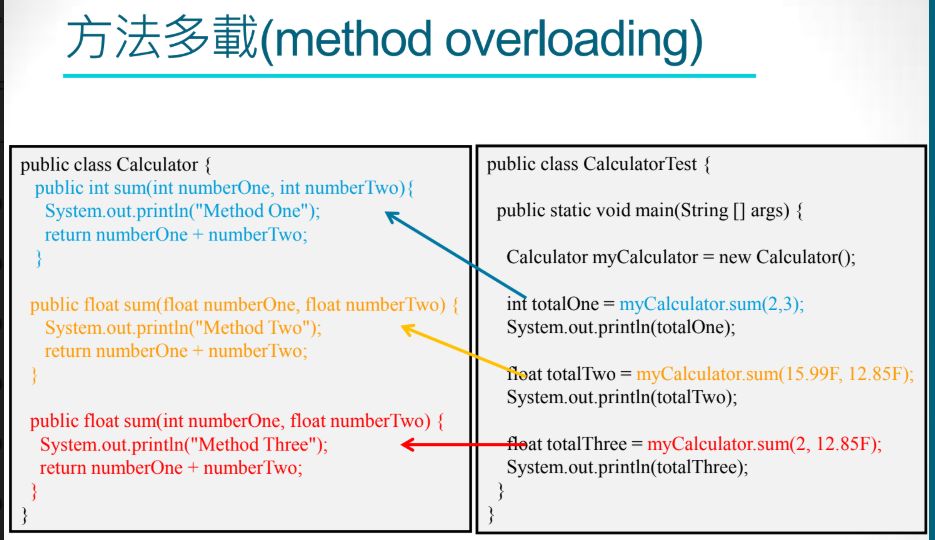
(2 呼叫other方法時，這時代入的c跟c2是相同的，但後來由新建立的customer物件代入c，使c參照到另一個位置的值

before

after

→other方法結束後，c就不存在了，同樣的它所參照到的值也會消失，所以這時再看c2的值，仍然是原本的那個





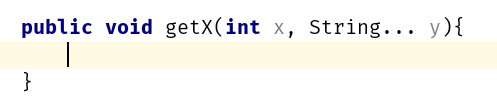
→方法多載的限制：

\* 參數數量可以一樣，但型態不能一樣

\*練習：試建立一簡單的文字RPG模組，有兩位人物，人物有HP、武器兩種屬性，並擁有各自的招式，使用招式讓角色攻擊對方各一次，攻擊造成的傷害是隨機的，並會使對方的HP下降，以此結束一回合，最後使用文字輸出顯示攻擊過程

* 補充知識：傳入不定數量參數時的設定

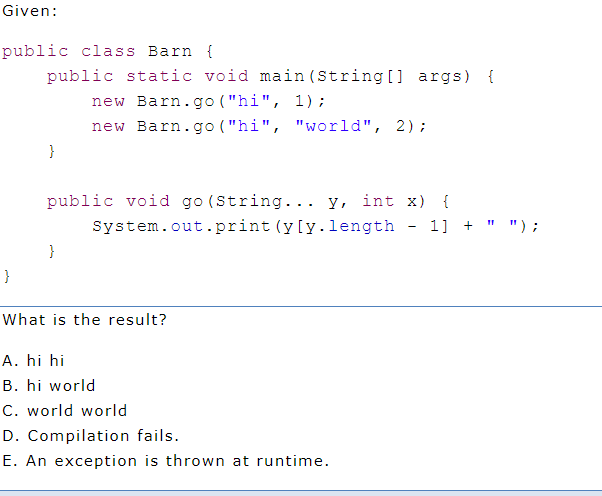
→一般來說，參數設定為1個型態+1個名稱，如有多個不確定數量的參數需加入時，可寫成：



→此方法為傳入不定數量的string參數，可傳一個或多個

→若有此種不確定數量的參數設定，需寫在最後面！

範例：



Ans: E

參考來源：

<https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/100318/>

**練習：看懂下列程式碼並解說流程與輸出結果**

**public** **static** **void** main(String[] args) **throws** Exception {

Member m1 = **new** Member("Tom",1,"male");

Member m2 = **new** Member("Jane",2,"female");

m1.deliverTo(m1);

m2.deliverTo(m2);

}

**class** Member{

String name;

**int** number;

String sex;

**public** Member(String name, **int** number, String sex) {

**super**();

**this**.name = name;

**this**.number = number;

**this**.sex = sex;

}

**public** String getSex() {

**return** **this**.sex;

}

**public** **void** deliverTo(Member member) {

**switch**(member.sex) {

**case** "male":

goToRoomA(member);

**break**;

**case** "female":

goToRoomB(member);

**break**;

}

}

**public** **void** goToRoomA(Member member) {

System.***out***.println(member.name + " should go to Room A!!");

}

**public** **void** goToRoomB(Member member) {

System.***out***.println(member.name + " should go to Room B!!");

}