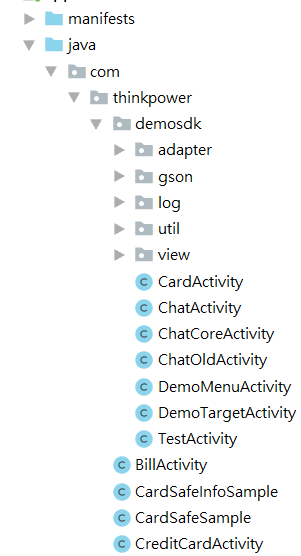
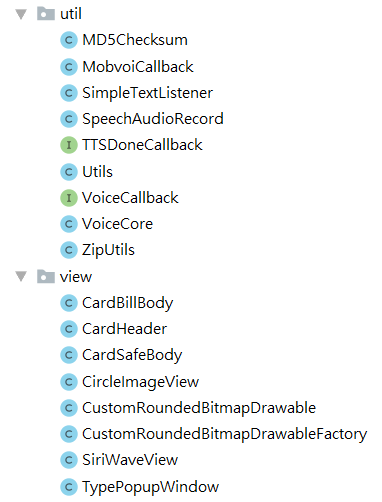
第一章：進階類別設計

* 類別在建立的時候可以善用存取修飾詞將層級分出來

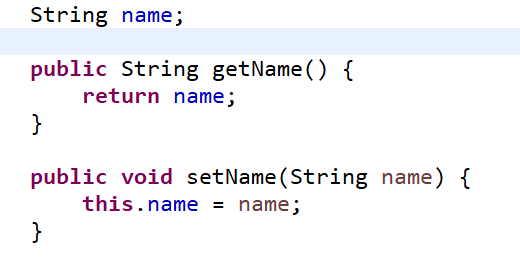


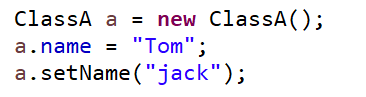
* 將同樣功能的java檔分類處理，達到分工的效果，以android專案為例

* 建立類別時養成封裝的習慣，屬性通常設成private，提供public的方法存取，使資料在傳遞時皆透過設定好的動作傳輸

封裝前：

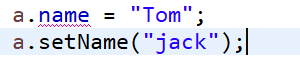




>>>物件的屬性存取太過容易

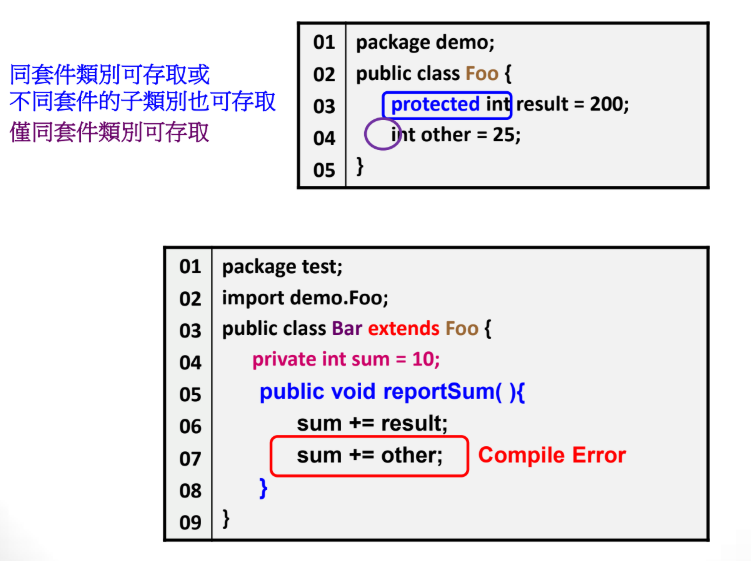
封裝後





>>僅能透過公開方法存取

* Proteced與default差別

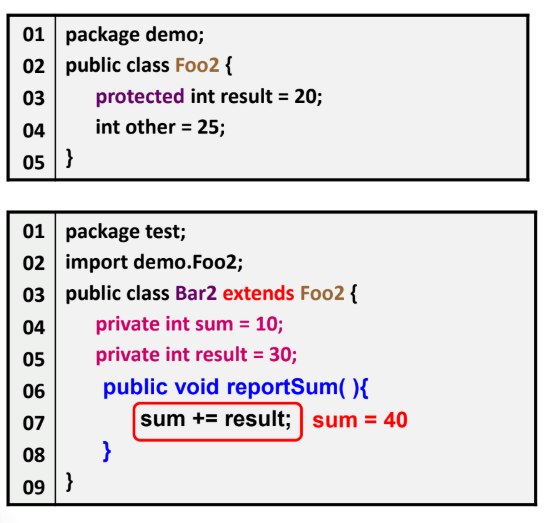


→result為protected，不同package下有繼承關係就可存取

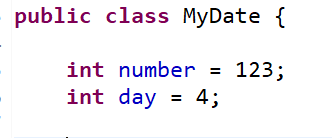
→other為default，只有在同一個package下才可存取

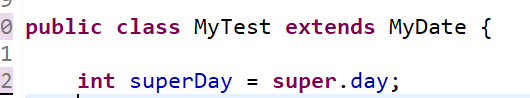
* 屬性遮蔽現象

→同一名稱的屬性，子類別會將父類別屬性遮蔽的現象



→如要存取父類別相同名稱的變數，可以使用super：



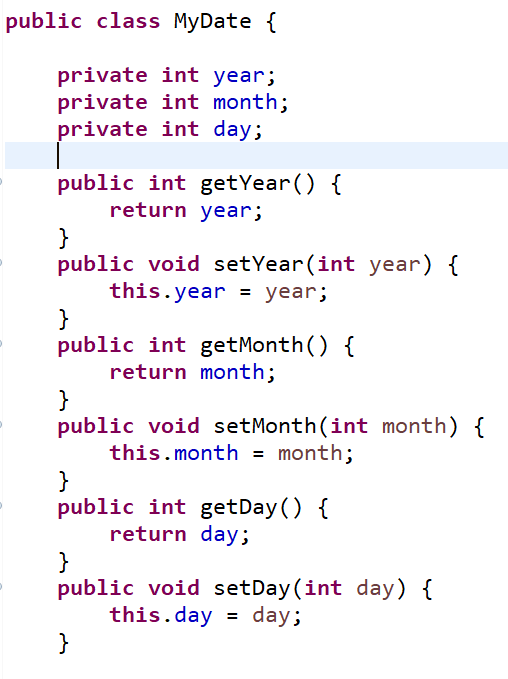


→可存取到父類別**原本的**屬性，但使用上較為不彈性。建議還是改為物件呼叫 ex：new MyDate().day; 或getxxx

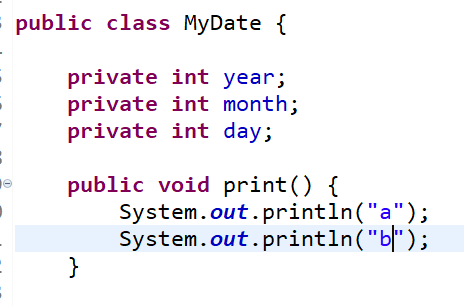
* 屬性封裝

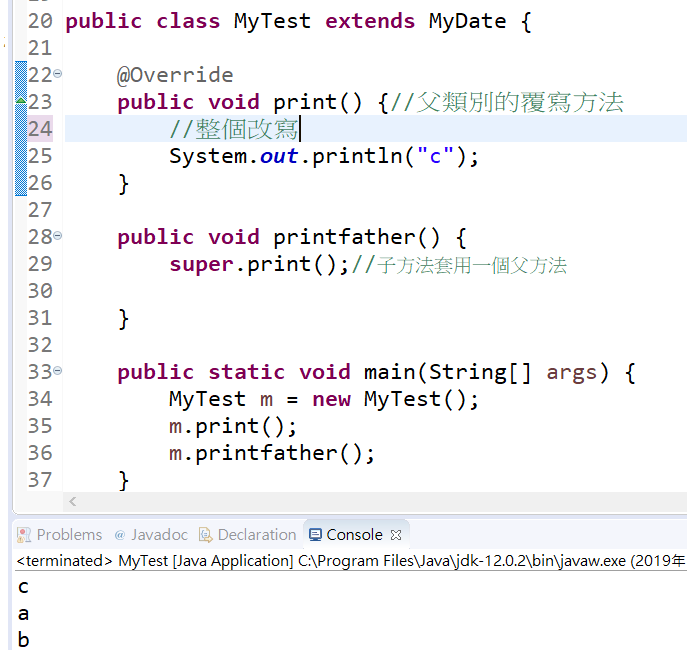
→類別在設計時盡量使用封裝設計，避免直接將屬性曝露在外，使繼承時或建立時僅能使用公開方法互相存取，將可存取方式單一化

→類別封裝的經典設計：

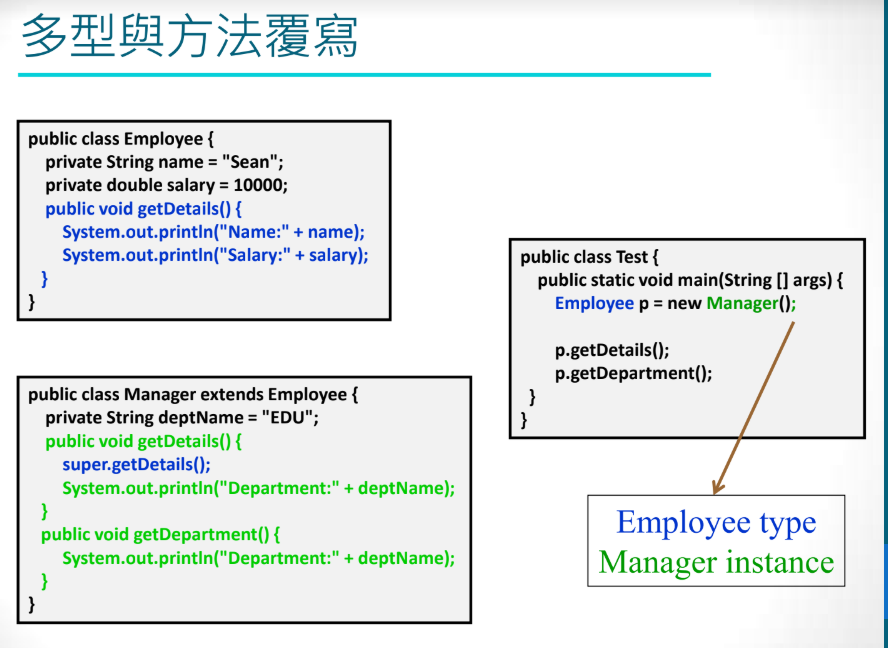


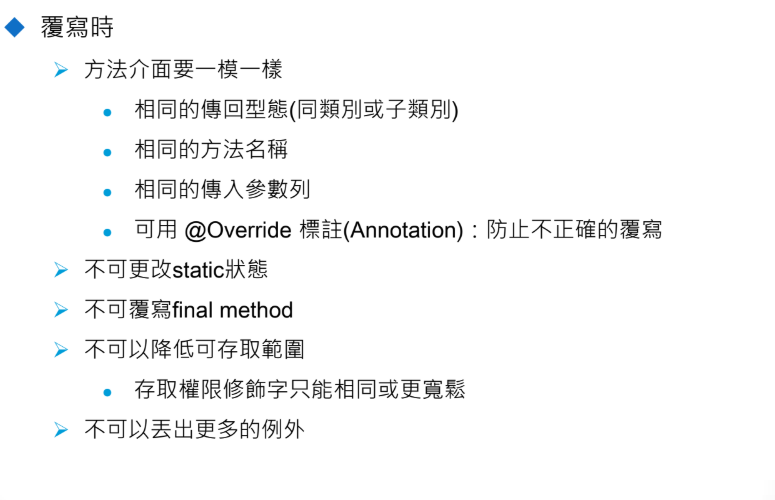
* 明確區分父方法與覆寫方法

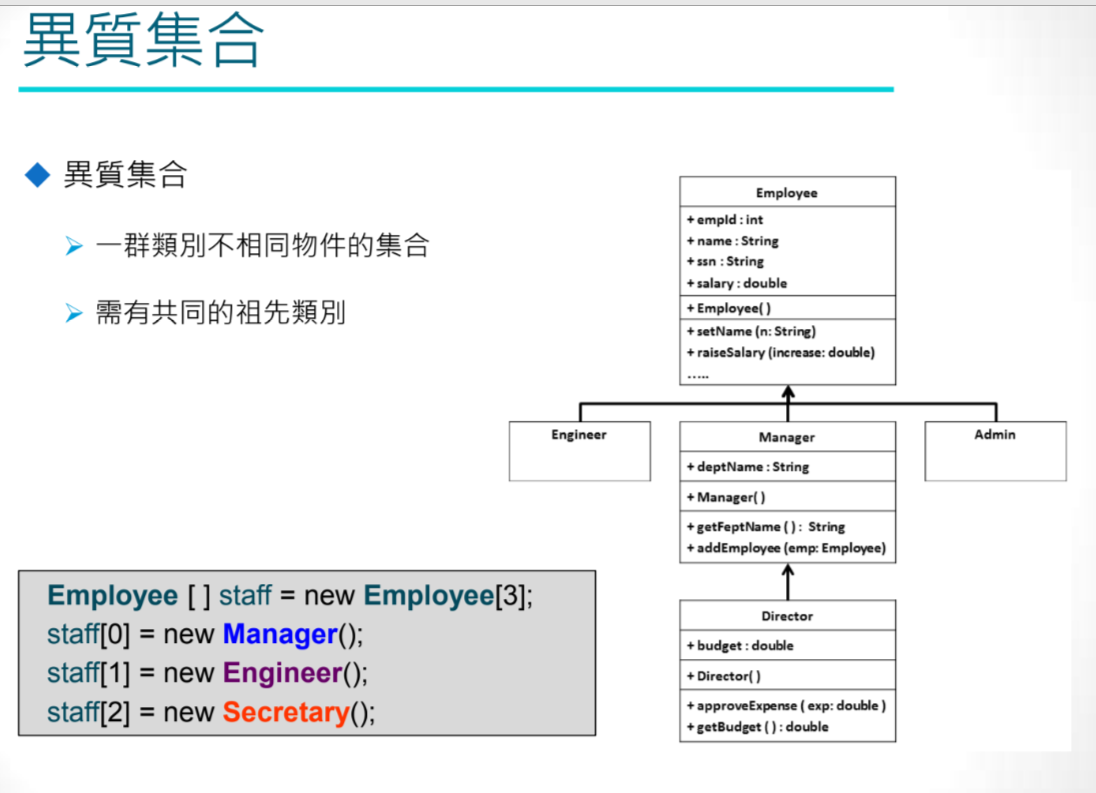




* 針對不同情況適當地使用多型



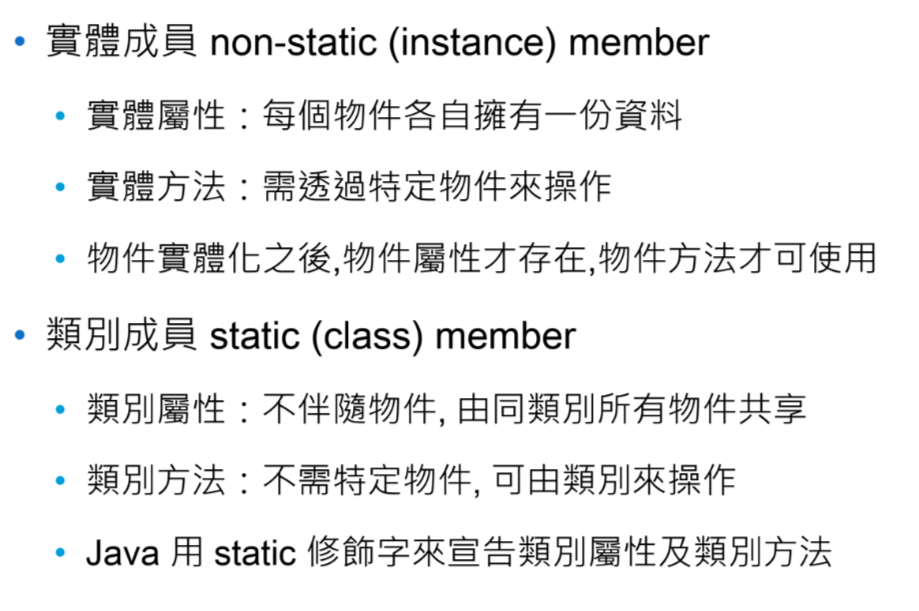


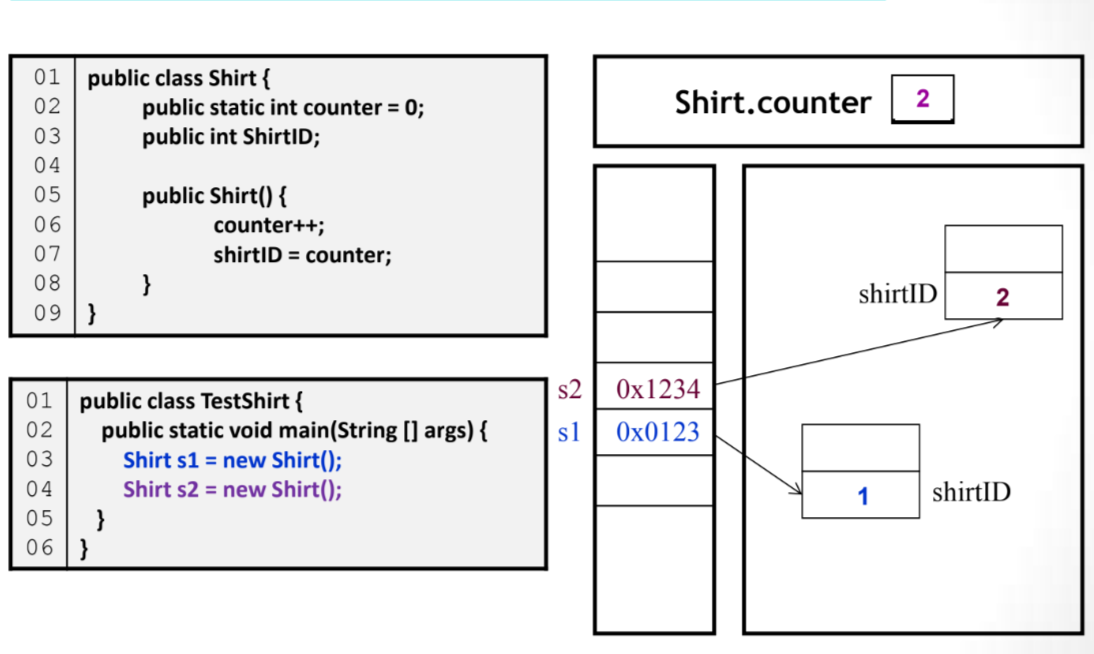


* Static關鍵字

→公設的觀念，存取範圍具有全域性，可在類別與類別間任意遊走

* Static VS Non-Static

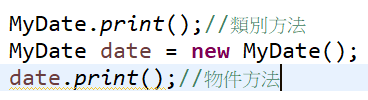


* Static屬性在記憶體示意圖
* Static方法

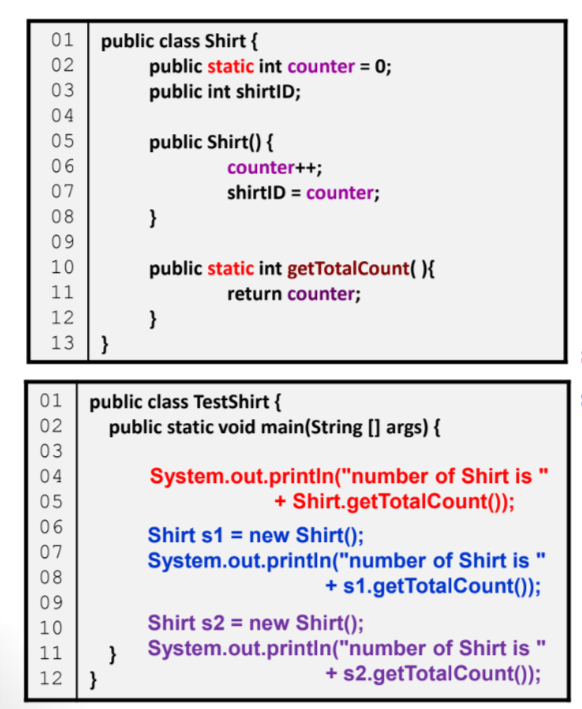
→可不用建立類別就能呼叫的方法

→呼叫方式可有兩種：

1. 類別名稱(class name)+方法名稱
2. 物件名稱(new XXX())+方法名稱



* Static變數的變化



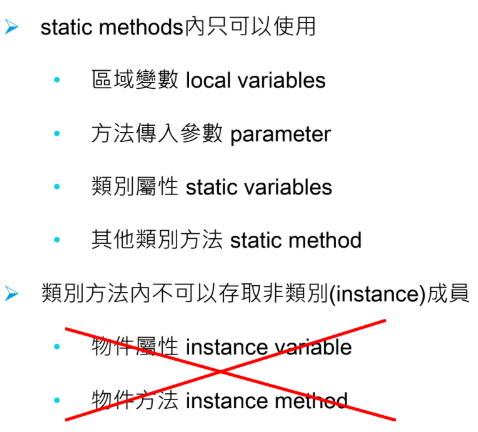
輸出結果：

Number of Shirt is 0

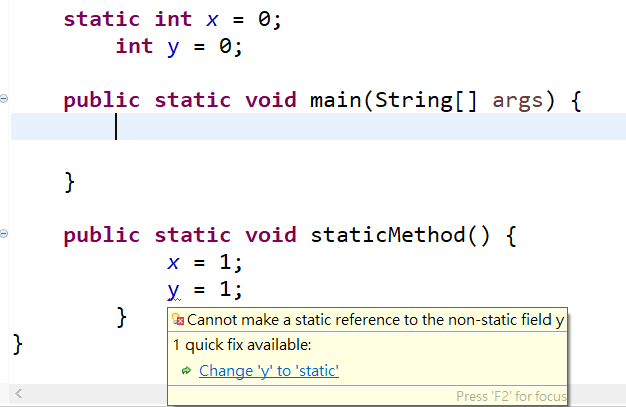
Number of Shirt is 1

Number of Shirt is 2

* 類別方法內的存取規則



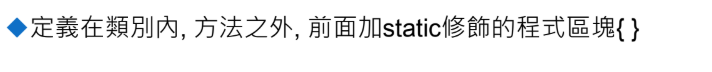
* 類別方法內的錯誤示範

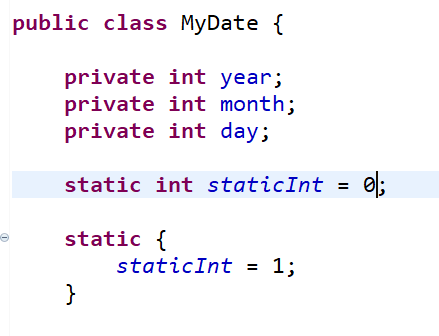


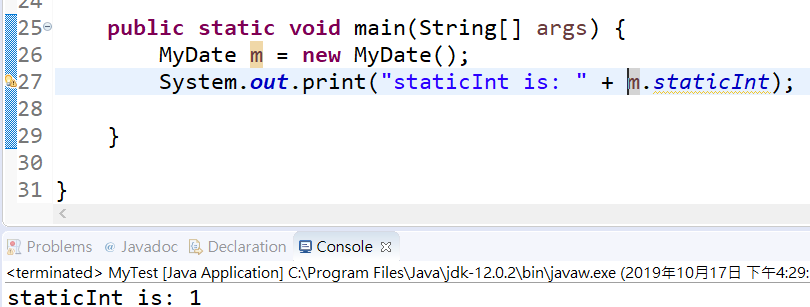
→ 類別方法內不可存取非類別(non-static)成員

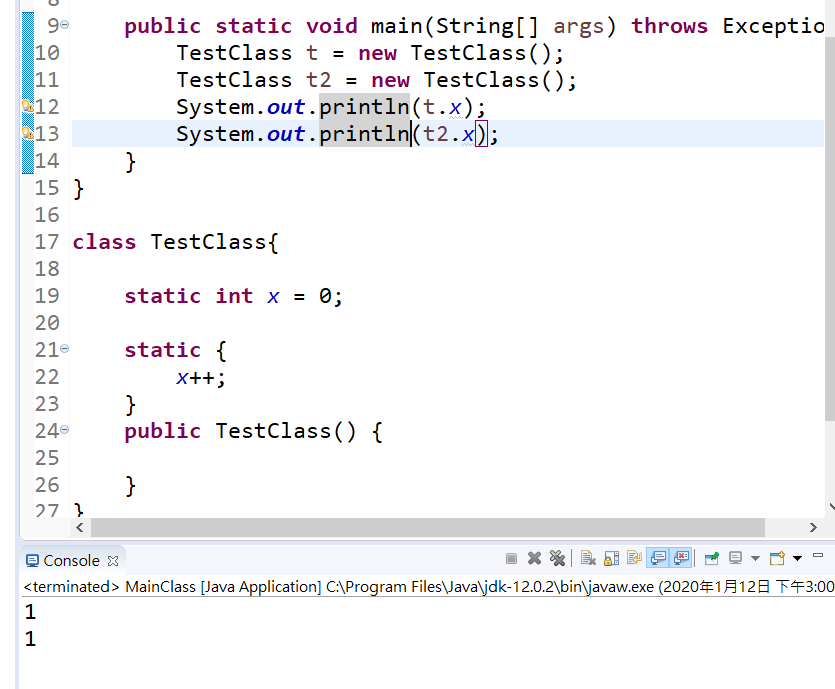
* 類別內的靜態區塊

→類別內一個專門的static的區塊，只會在建立實體時執行一次，與建構式相似，但不用寫在建構式內



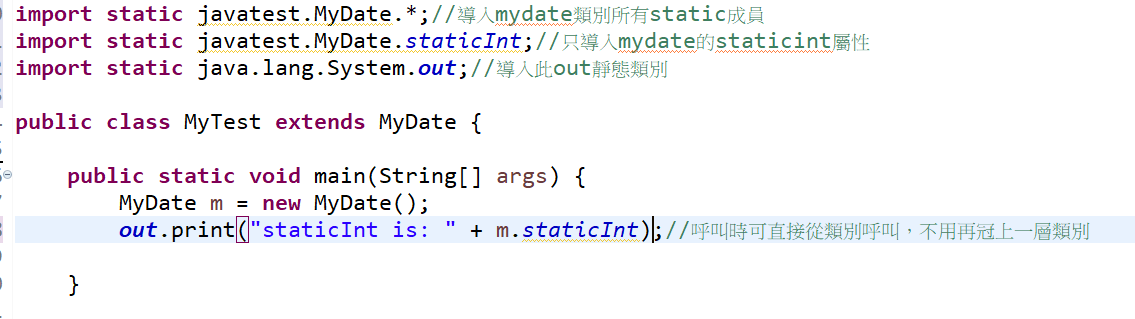




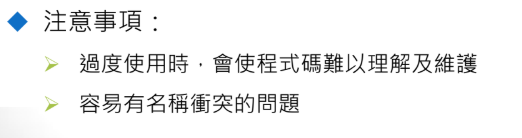


* 導入static成員

→使用import導入靜態成員，可縮短程式碼，不用再依序用類別名稱呼叫



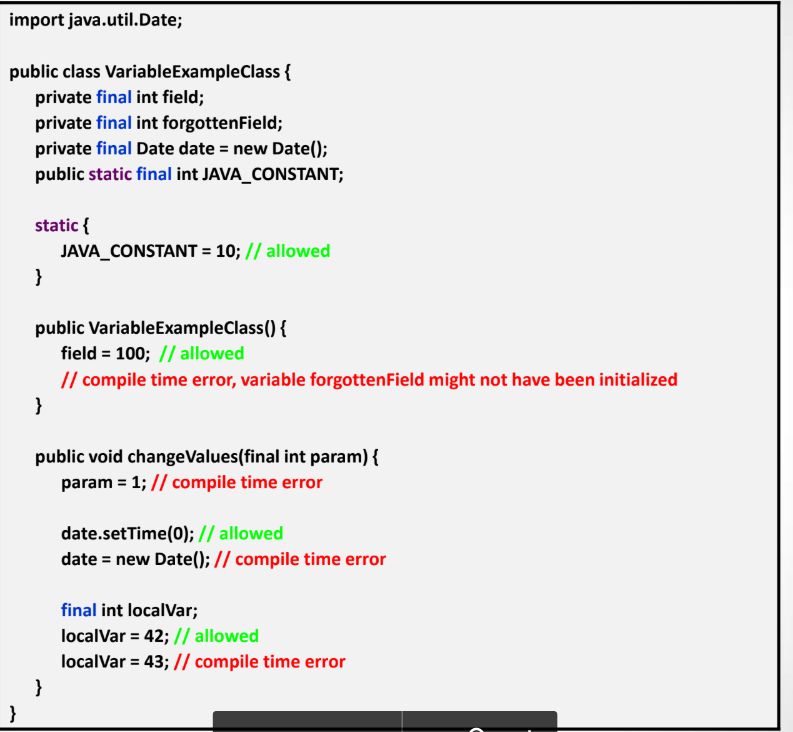
→該類別或該屬性為static要導入時，一定要加static



→對於static所詮譯的程式碼，具有高度曝光率與易於存取的特性，實作上勿將敏感性資料冠上static

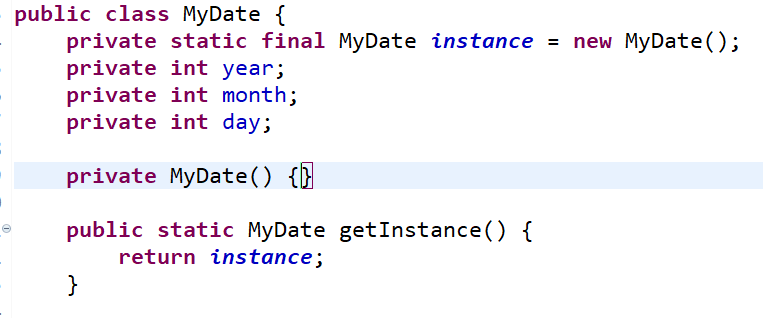
* Final關鍵字

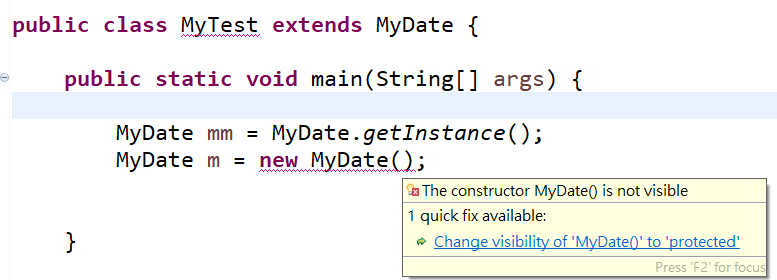
1. 冠上final的屬性只能初始化一次！
2. Final不可被繼承！
3. Final不可被覆寫！



* Singleton設計

→顧名思義為只有一個，用於控制在同一類別建立實體時只能透過getInstance方法建立一個唯一的物件

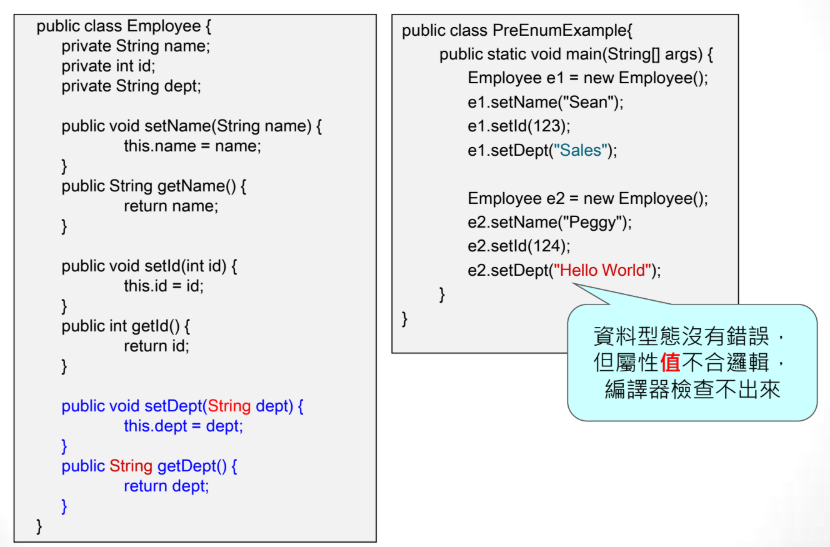


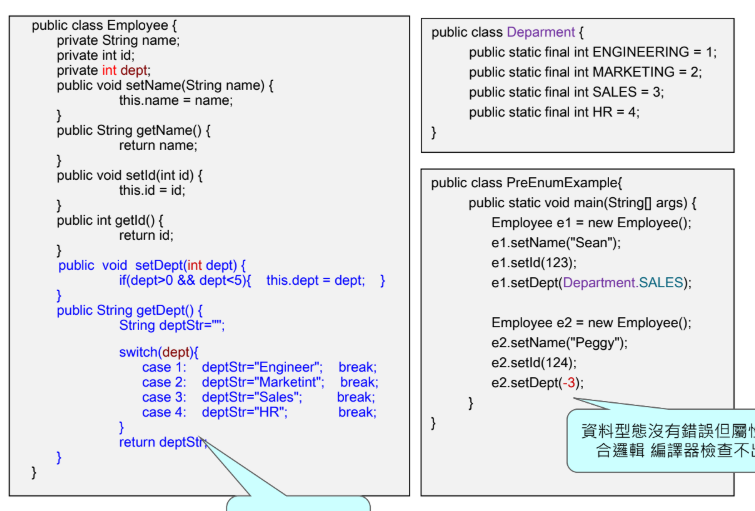


<https://skyyen999.gitbooks.io/-study-design-pattern-in-java/content/singleton.html>

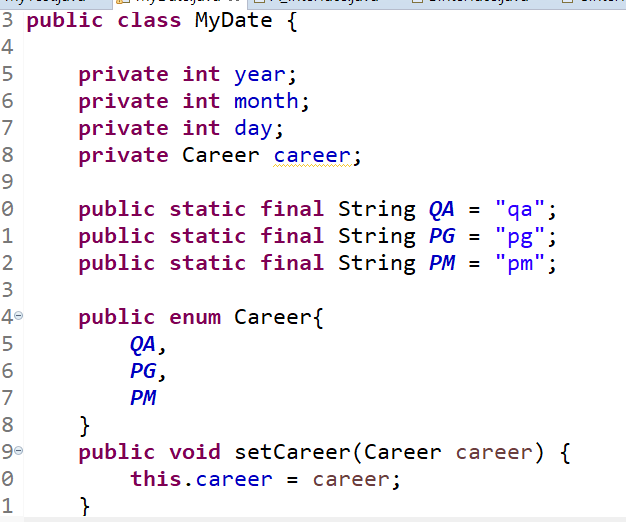
透過singleton設計的類別物件，在同一class只能建立一次，以符合單一物件原則，通會會在getInstance時將物件初始化及其它設定一併完成

* 列舉enum
* 列舉意思是指有著限定範圍的資料集合，例如在設計一個公司部門類別時，如果新增的名稱出現不合邏輯的數值(“電風扇”)，編譯器檢查不出來，程式也可以執行，但出來的資料會很怪(沒有一個部門名稱叫電風扇)，此時就可以使用列舉來規範邏輯

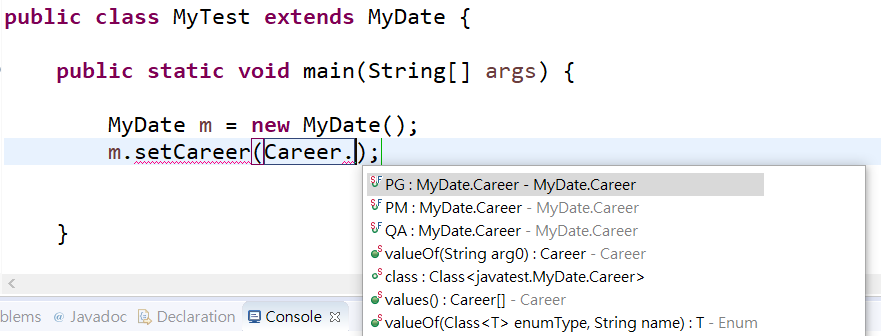


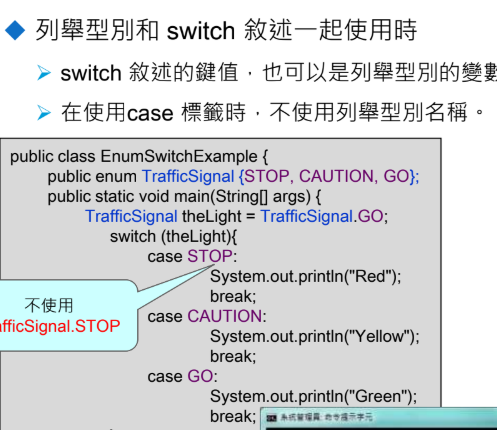


→沒有列舉之前，只要符合型態的程式碼都會成為資料的一部份，但在邏輯上說不過去，若要篩選也會增加許多程式碼，增加寫錯或是catch不到的狀態，不是一個好的設計

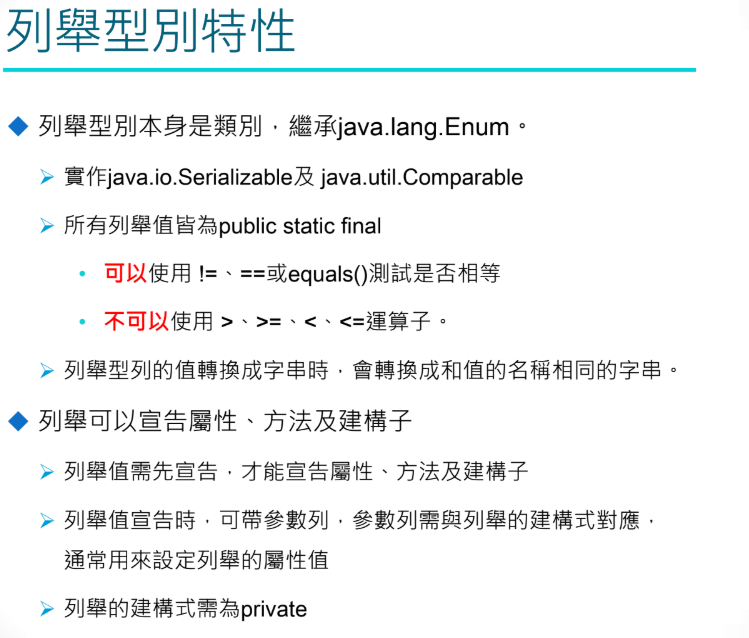


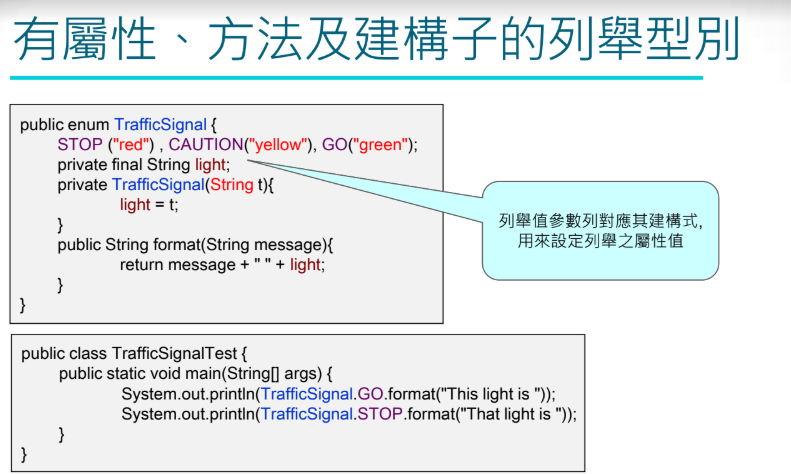
* 使用時



* 列舉可適用在switch中









只能寫在一般類別上。

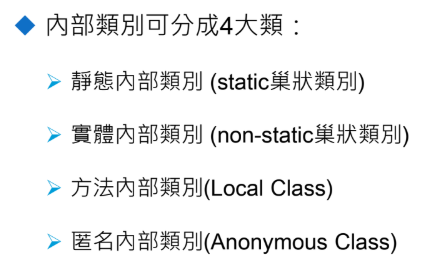
* 匿名類別

→針對使用次數有限的類別，不使用變數建立實體，即不取名稱直接實體化來用的方式，可有效減少變數太多，用量不大卻佔據記憶體的問題，是一種用完即丟的概念



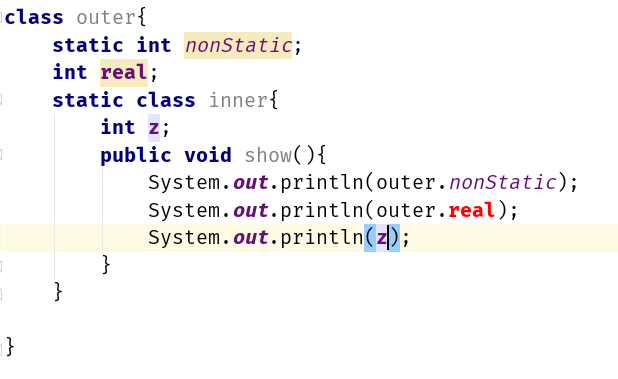
→雖有效降低空間使用，但不建議太頻繁使用，以免需要存取的資料找不到地方放置。

* 巢狀類別
* 類別內還有一類別的用法
* 外部類別稱為outer Class, 內部類別稱作inner Class
* 內部類別可以直接用外類別的屬性，彼此間的依符關係更緊密



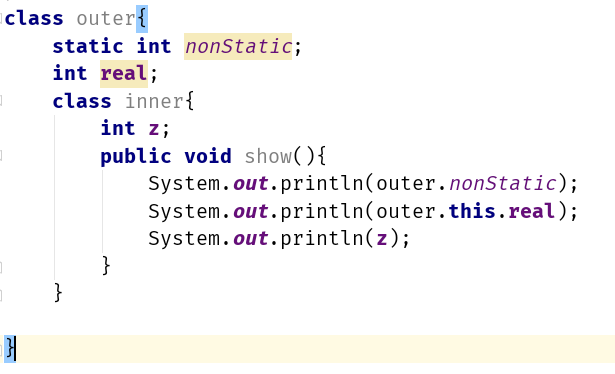
1. 靜態內部類別

→內部類別設為靜態時，只能存取外類別的static成員，實體成員不行

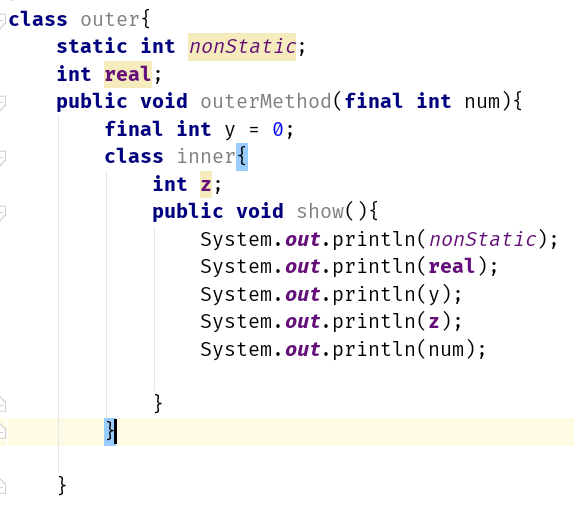


1. 實體類部成員

→與一般類別相同，外類別的實體成員可加上this來呼叫



1. 方法內部類別

→外類別方法內有一個內部類別，可以存取外類成員，

→方法內部類別的方法如果要存取方法外面的屬性，需要加final才可以叫，參數也是

1. 匿名類別進階寫法

→通常實作在介面上，主要是針對在改寫的方法上

