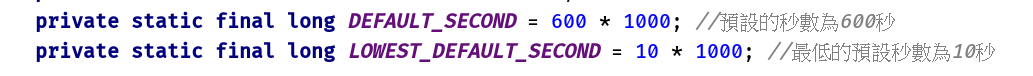
第一章：進階類別設計

1. 修飾詞與變數

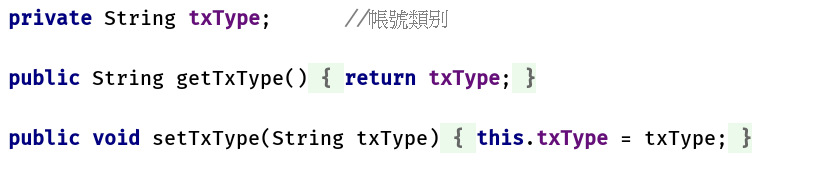
* 修飾詞層級



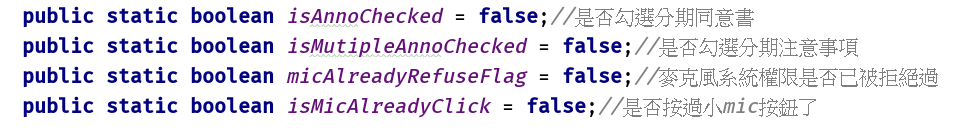
* 冠上public static final的變數通常是為了格式或專案的基本參數設定，不常更動的，命名為全大寫



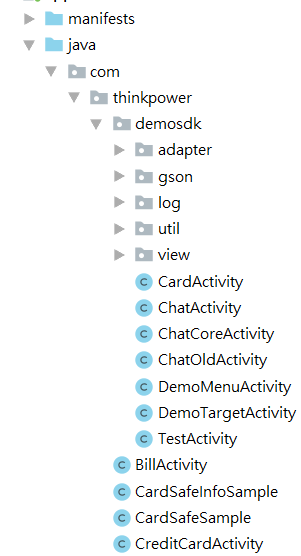
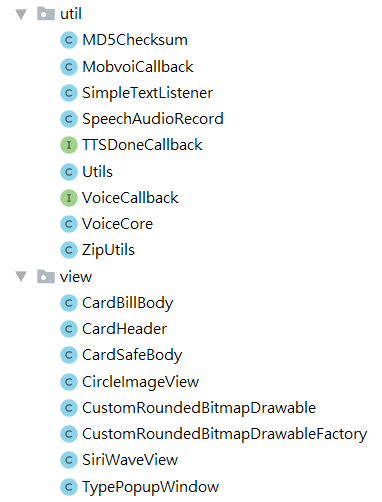
* 若是建立物件模組時，變數基本上都需要封裝處理



* 方法想經由繼承讓別的類別使用的話，可以將方法冠上protected或public
* Static 變數為全域變數，可以用來記錄狀態，例如整個專案在進行時要進行累加的或是狀態會頻繁改變，不屬於任一類別的



* 類別中較不是處理重要資訊，只是要做資料傳遞等基本功能的話，變數可以都以default顯示
* 檔案分類處理
* 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



其它

主活動頁

客製畫面

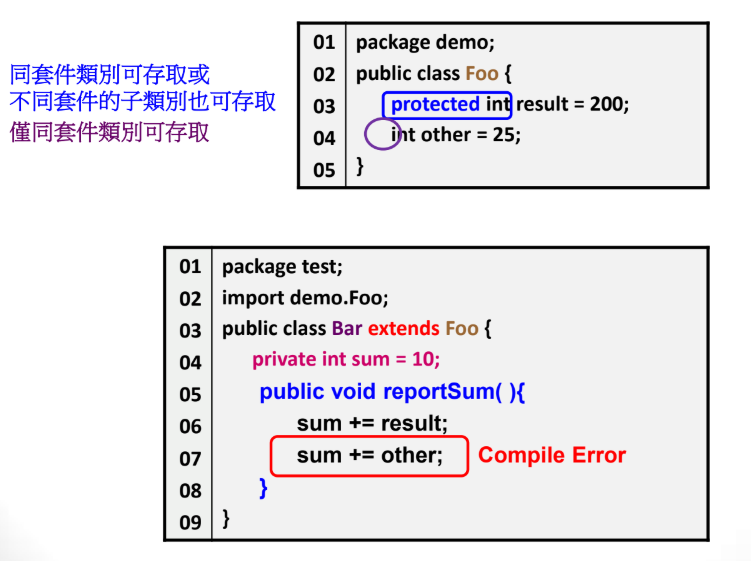
工具包

記錄

資料model

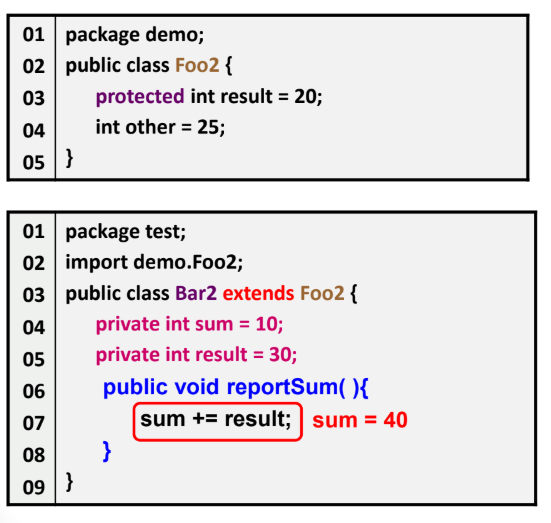
分頁資料

* Proteced與default差別



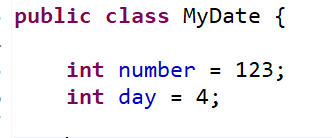
* result為protected，不同package下有繼承關係就可存取
* other為default，只有在同一個package下才可存取
* 繼承狀態下的屬性遮蔽現象

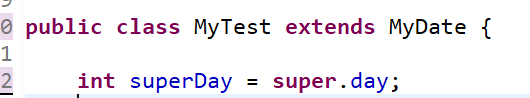
→同一名稱的屬性，子類別會將父類別屬性遮蔽的現象



**X**

* 如要存取父類別相同名稱的變數，可以使用super：

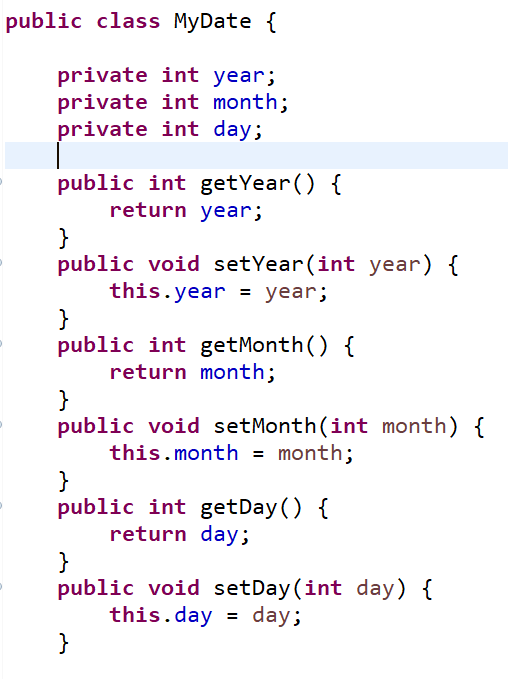




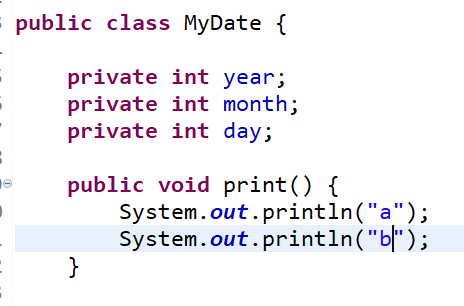
* 可存取到父類別**原本的**屬性，但使用上較為不彈性。建議還是改為物件呼叫 ex：new MyDate().day; 或getxxx
* 遮蔽以字面上意思來說是隱藏，所以被遮蔽的屬性事實上並沒有消失，而失隱藏在後面，用特定的方式一樣可以呼叫得到
* 屬性封裝

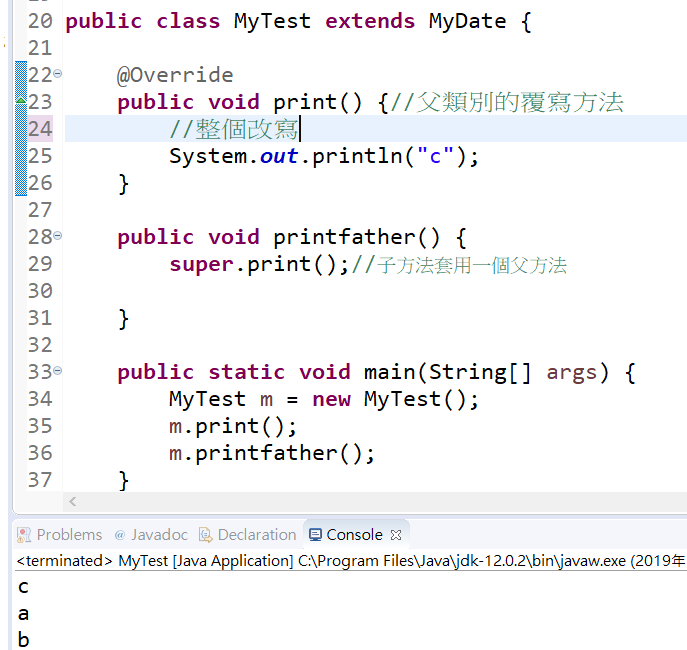
→類別在設計時盡量使用封裝設計，避免直接將屬性曝露在外，使繼承時或建立時僅能使用公開方法互相存取，將可存取方式單一化

→類別封裝的經典設計：

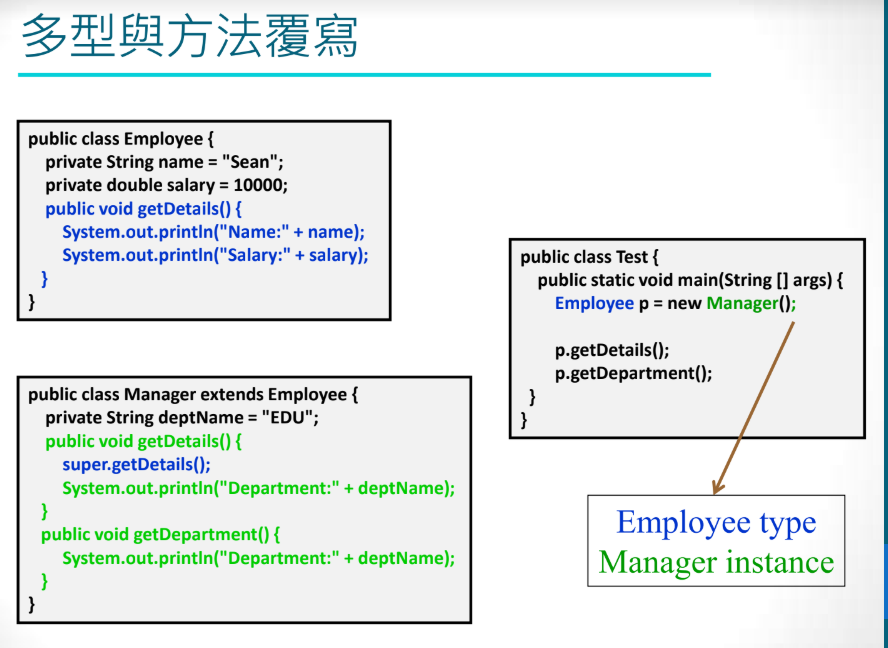


* 明確區分父方法與覆寫方法

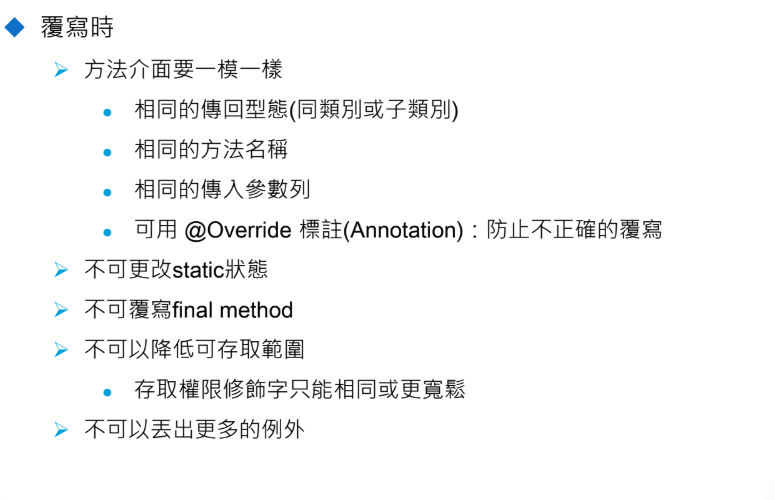


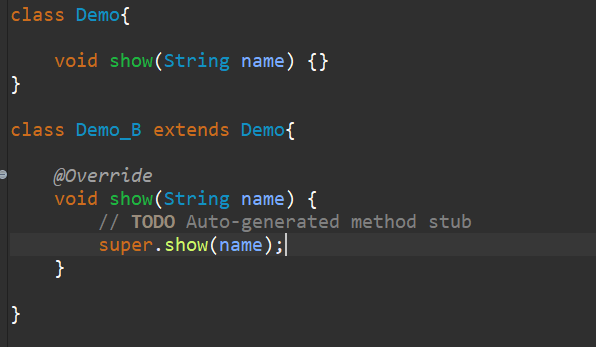


* 針對不同情況適當地使用多型

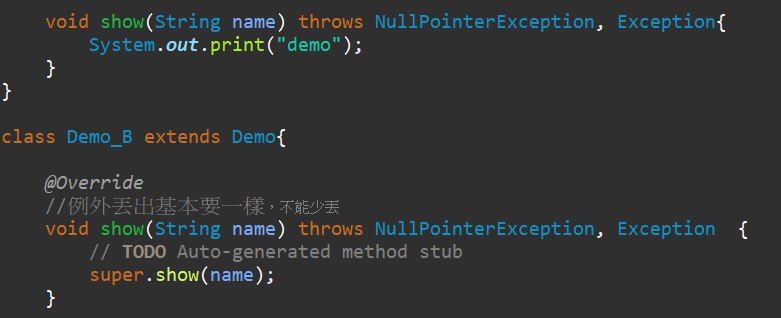


* JAVA覆寫規範



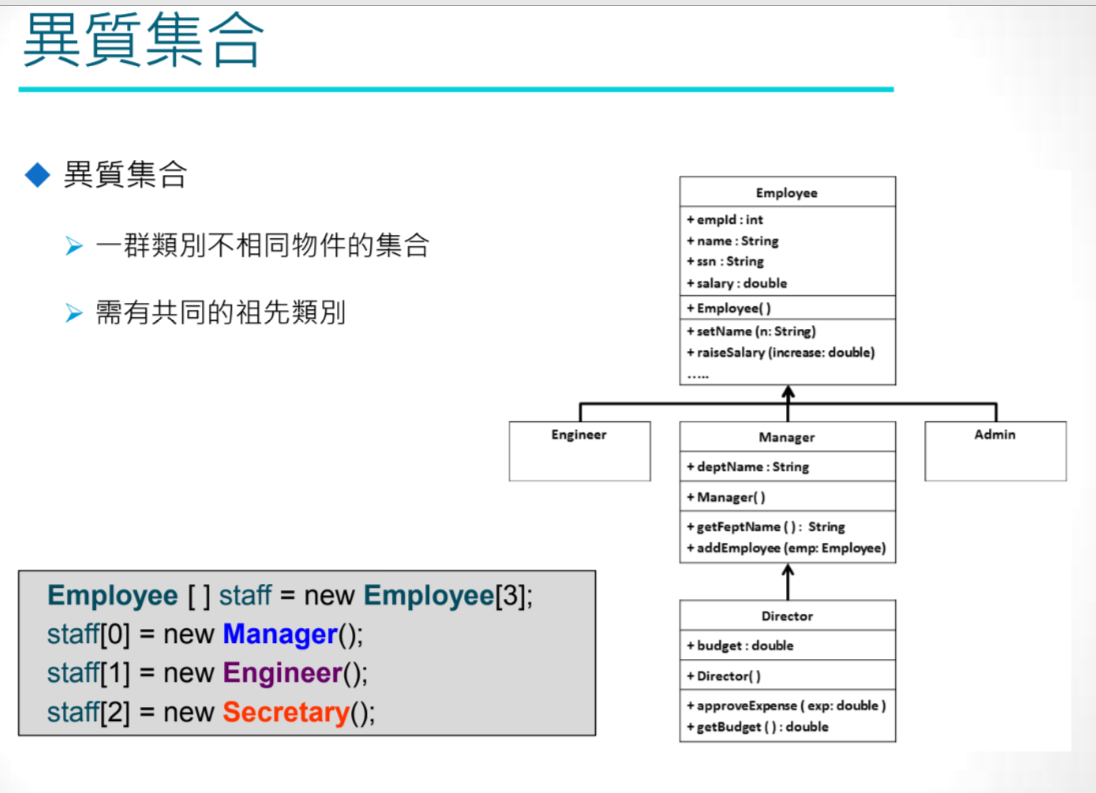


\*注意修飾詞範圍 & 覆寫方法樣式



有例外丟出時的覆寫方法

* 丟出的例外基本上不能少
* 覆寫方法不能丟出更多的例外
* 異質集合操作



Admin();

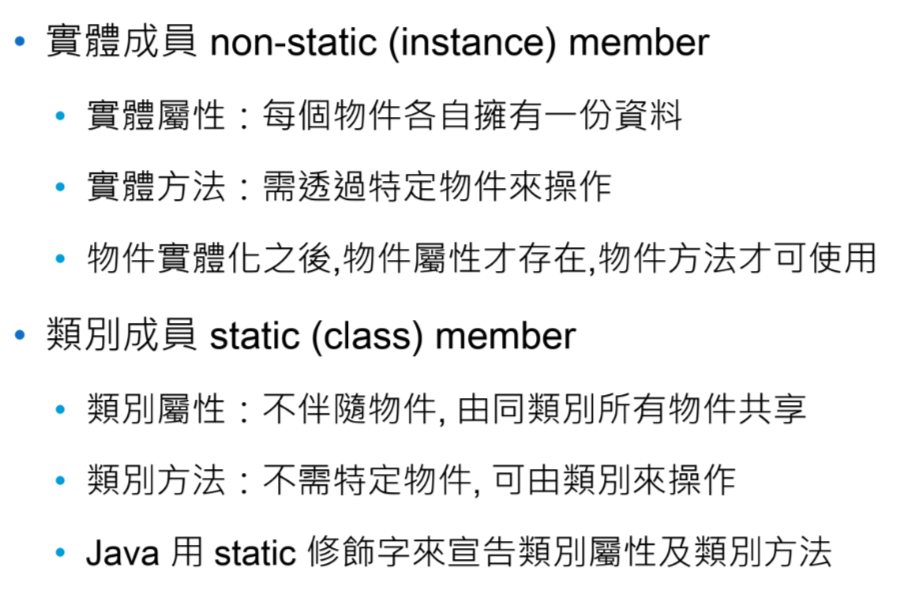
裡面可放繼承的子類別，符合多型概念

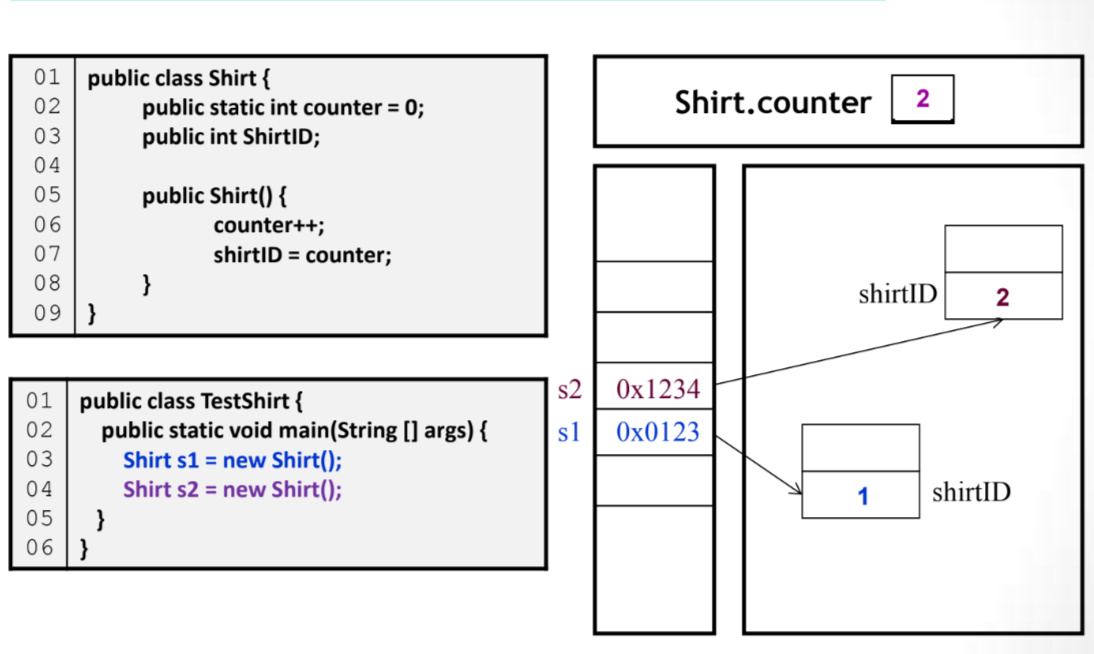
父類別集合

\*也可以視為是一種多型的集合

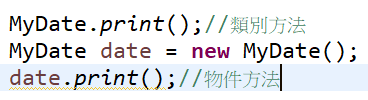
1. Static關鍵字

* 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

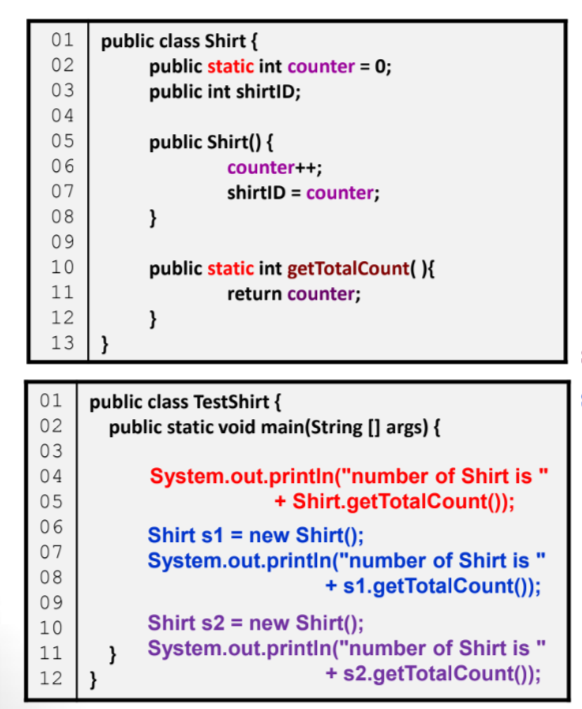


* Static屬性在記憶體示意圖
* Static方法
* 可不用建立類別就能呼叫的方法
* 呼叫方式可有兩種：

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



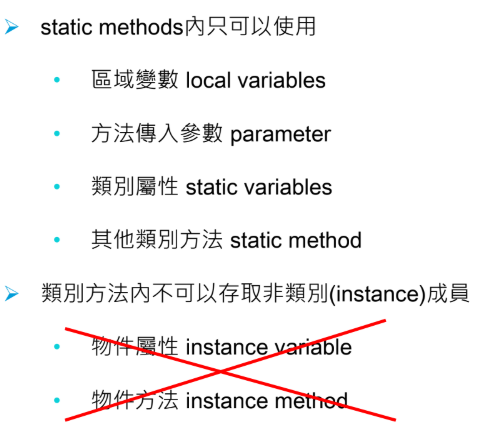
輸出結果：

Number of Shirt is 0

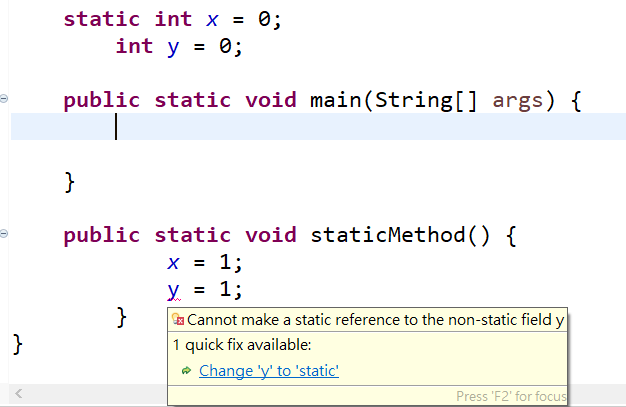
Number of Shirt is 1

Number of Shirt is 2

* 類別方法內的存取規則



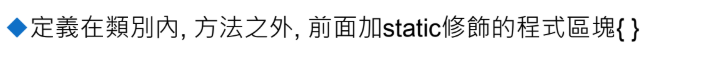
* 類別方法內的錯誤示範

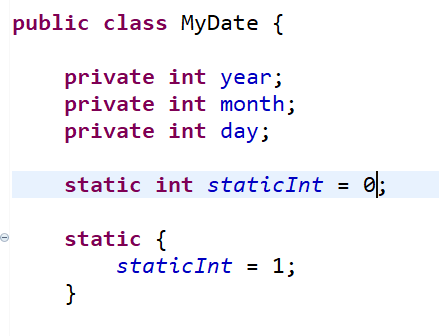


→ 類別方法內不可存取非類別(non-static)成員

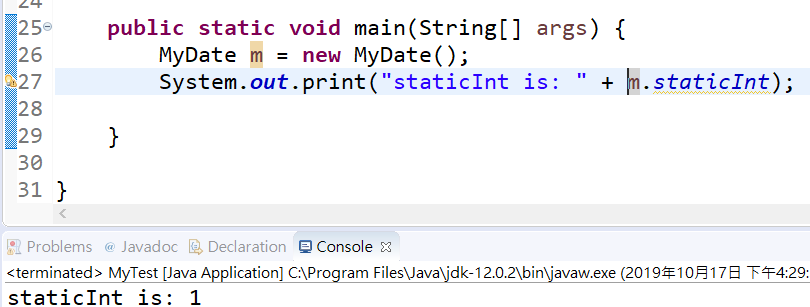
* 類別內的靜態區塊

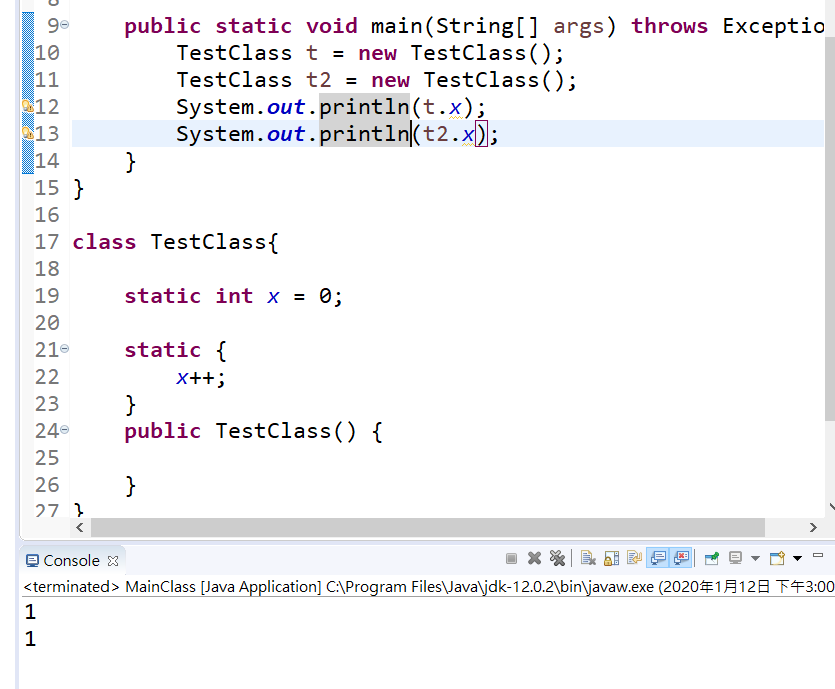
→類別內一個專門的static的區塊，只會在建立實體時執行一次，與建構式相似，但不用寫在建構式內





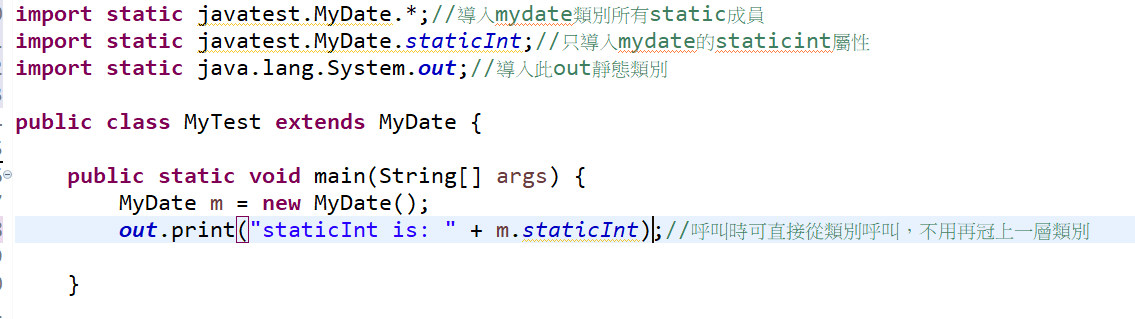
MyDate建立時會呼叫一次改變staticInt變數，只會執行一次



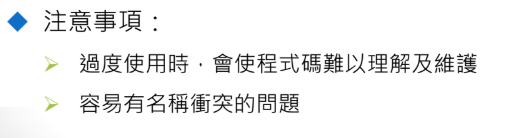


* 導入static成員

→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

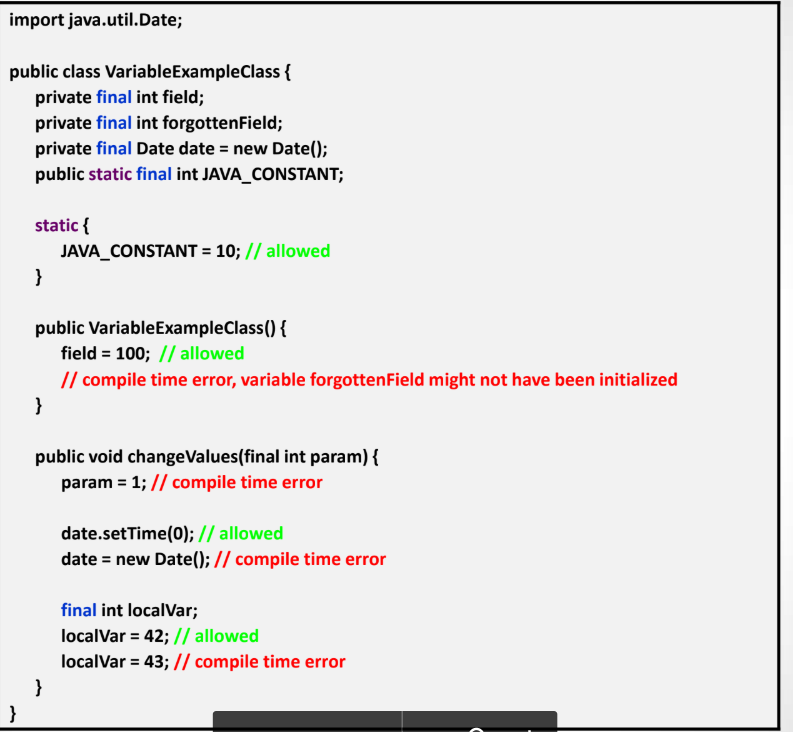


* 該類別或該屬性為static要導入時，一定要加static



* 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

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



取在類別內的final在static方法可以初始化一次

建構式可將尚未初始化的final初始化一次

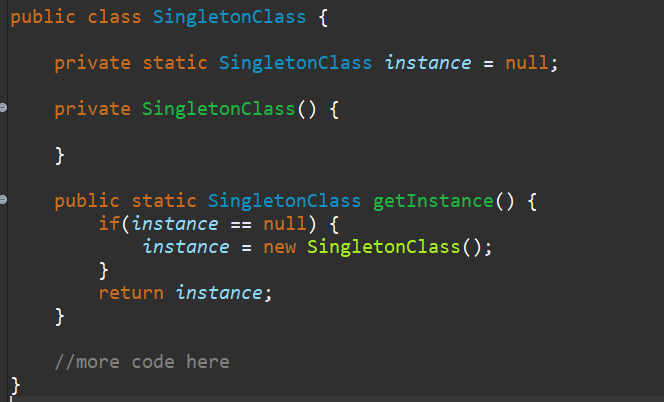
方法的final參數不可以再給值

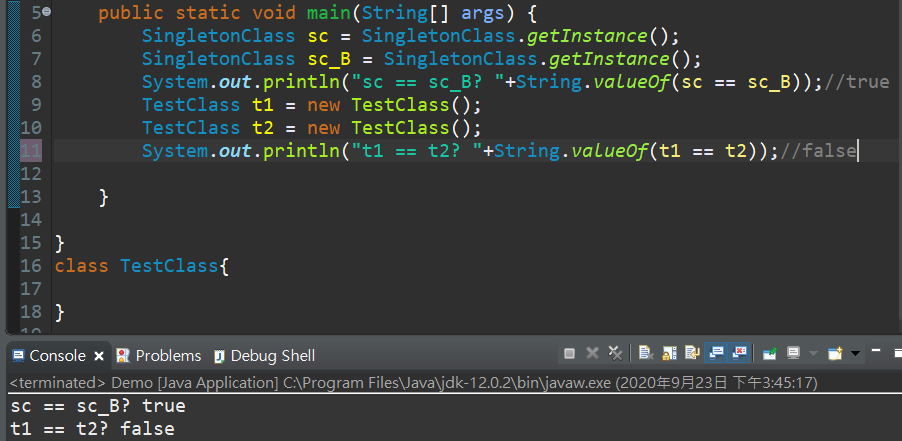
已實體化一次的final不可以再重新new

方法內的final只能初始化一次

1. Singleton設計

* 保證一個類別只能產生一個物件
* 使用統一的方法存取該物件





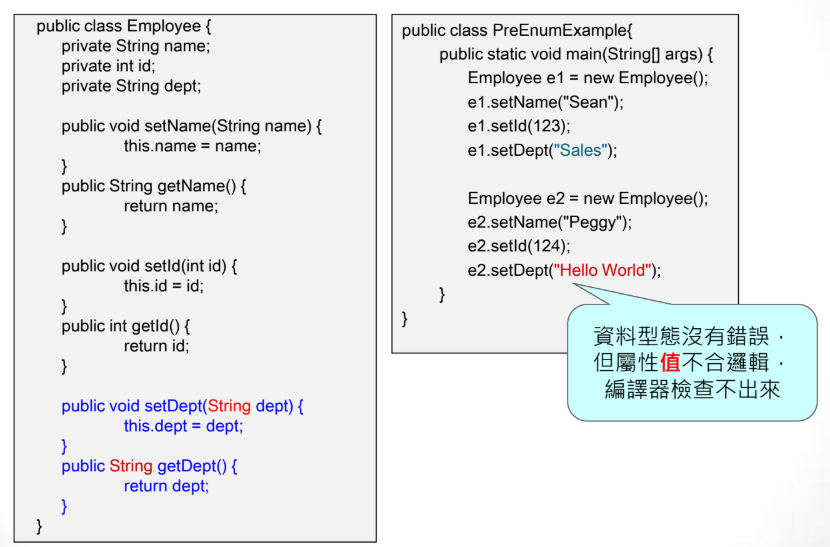
<https://skyyen999.gitbooks.io/-study-design-pattern-in-java/content/singleton.html>

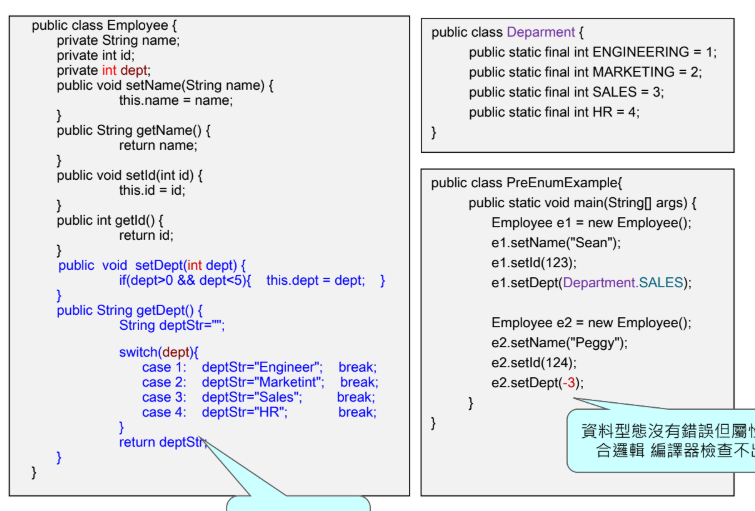
https://matthung0807.blogspot.com/2018/02/java-singleton.html

* 使用singleton建立物件時，不是直接new一個新的物件，是透過靜態方法取得
* 藉由統一的方法getInstance取得物件時，會先判斷是否已建立過了，若存在就直接用原本的實例，而不是每次都建一個新的
* Singleton可以保證該物件保有同一個狀態，如果要記錄的是公共的內容，可保證其值不會受到干擾
* 由於限制了實例個數的建立，更有利於GC或提升效能

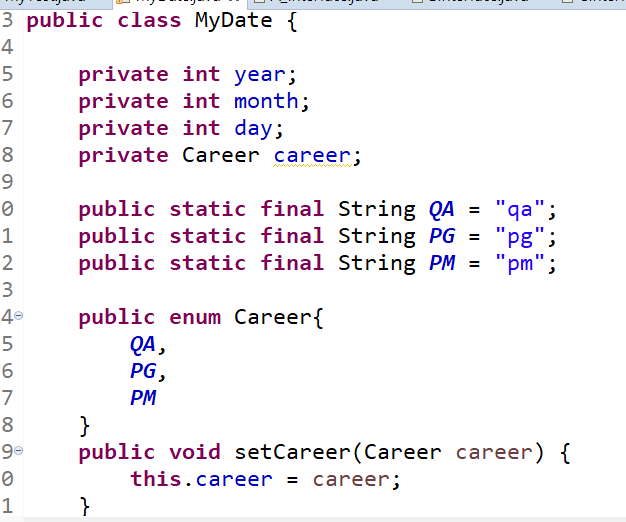
1. 列舉enum

* 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
* 主要用來規範商業邏輯或限定篩選範圍
* 基本觀念



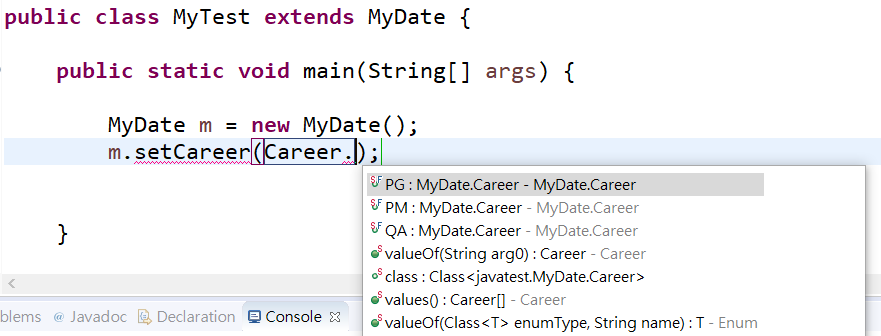


* 沒有列舉之前，只要符合型態的程式碼都會成為資料的一部份，但在邏輯上說不過去，若要篩選也會增加許多程式碼，增加寫錯或是catch不到的狀態，不是一個好的設計

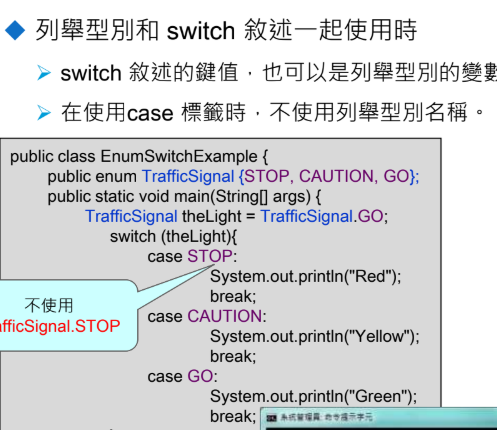


使用enum指定setCareer時只能給什麼值，限縮參數的範圍

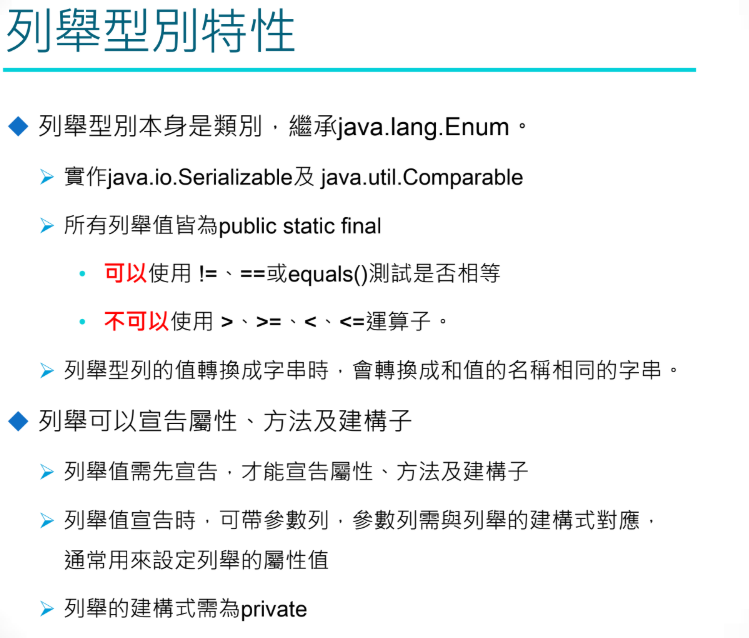
使用時

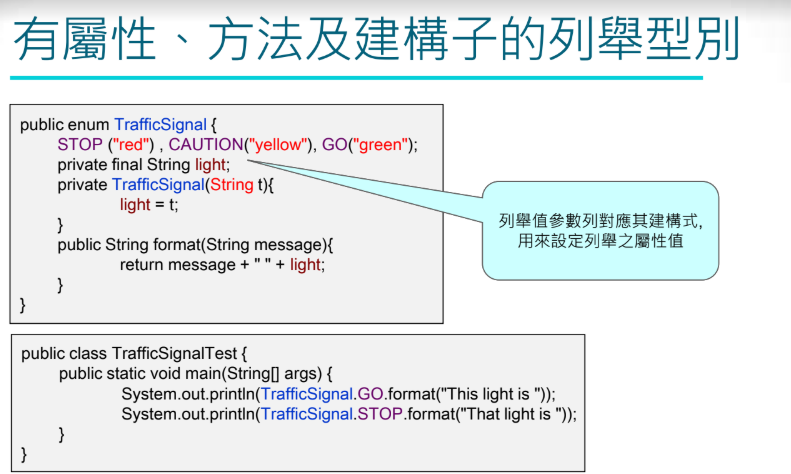


參數一開始時指定時就只能放列舉內的項目，往後要更新項目只要更新列舉內容即可

列舉可適用在switch中

Case內的值就值接用列舉值





呼叫列舉屬性時，等同呼叫其建構式，並將對應的值指定給light變數



* 列舉只能寫在一般類別上。

1. 匿名類別

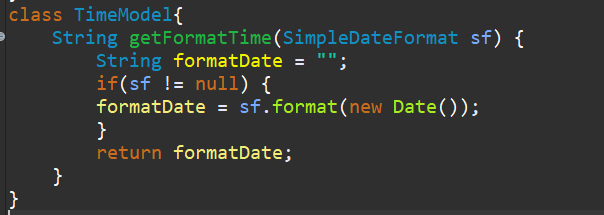
* 基本觀念

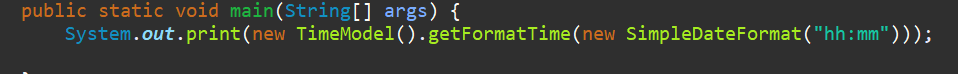
→針對使用次數有限的類別，不使用變數建立實體，即不取名稱直接實體化來用的方式，可有效減少變數太多，用量不大卻佔據記憶體的問題，是一種用完即丟的概念



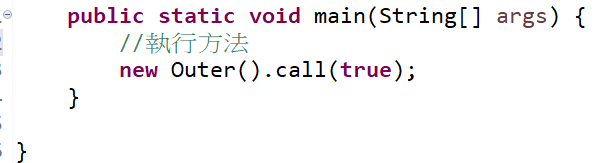
* 目的只是為了使用該類別的某個屬性或方法，並不是要用該物件進行多項操作
* 雖有效降低空間使用，但不建議太頻繁使用，以免需要存取的資料找不到地方放置。
* 匿名類別範例

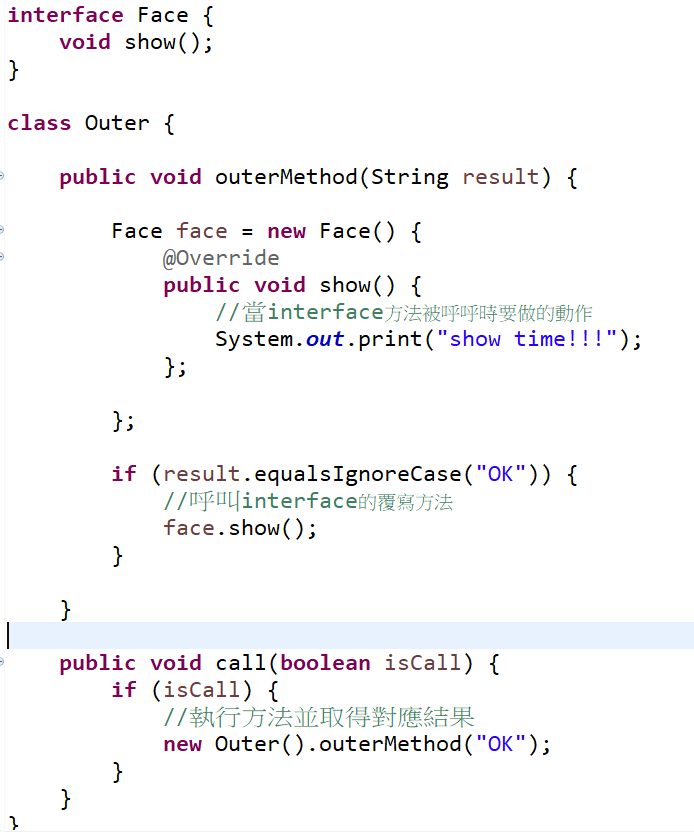
Ex1:





Ex2:





一介面有一抽象方法，當作收到正確結果時的通知動作

當程式走到這句時，自動執行show的改寫方法，通常在串接api得到回應時要做的動作，接給下一位時使用

實作該介面時要覆寫該方法的內容

甲方

A類別傳送資料

收到回應後通知B要做show

B類別得到A的呼叫後，做出得到回應時的反應

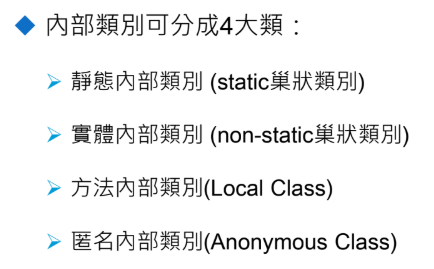
回傳結果

傳送資料

伺服器

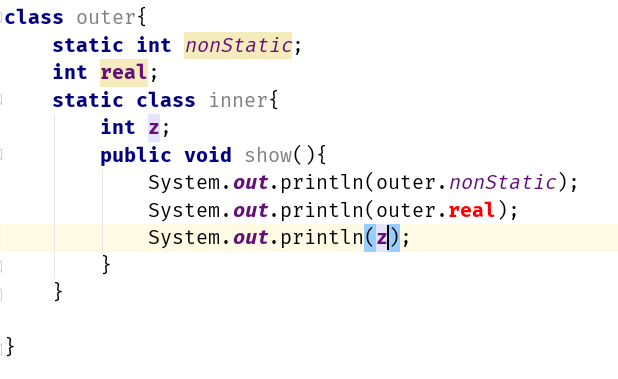
系統端

1. 巢狀類別
2. 類別內還有一類別的用法
3. 外部類別稱為outer Class, 內部類別稱作inner Class
4. 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



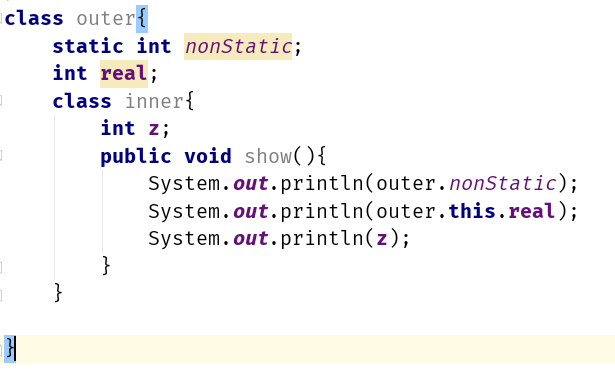
1. 靜態內部類別

→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

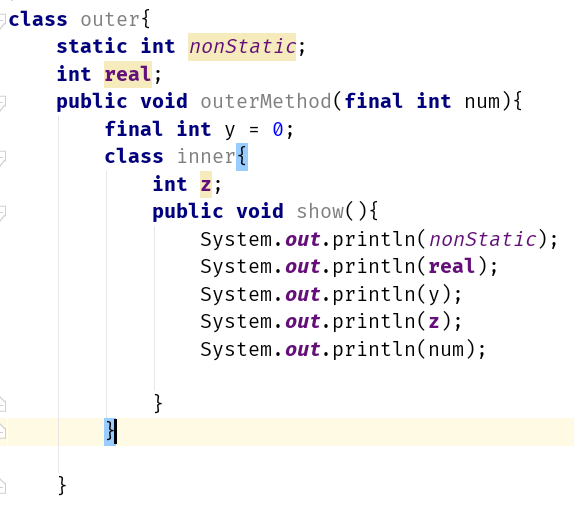


1. 實體類部成員

→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