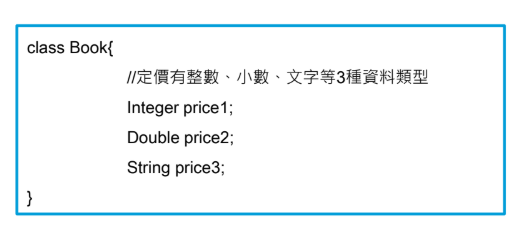
第三章：泛型(Generic)

1. 基本介紹

* 泛型是java6.0後新增的一個語法功能，在程式撰寫時很常用，目的是在規範類型，以便在傳輸資料或製作數據時符合所需格式之外，也能限定型別的範圍，降低錯誤的頻率！
* 「泛型」是「參數化型態」，可以先用方法參數來想，我們在放入方法參數後，調用該方法時會用該參數來代替外來傳進的資料來做傳遞，參數化型態就是將型態參數化，使得在使用時能傳入具體的型態
* 泛型可以用在類別(泛型類別Generic Class)、介面(Generic Interface)和方法(Generic Method)中

<https://ethan-imagination.blogspot.com/2018/11/javase-gettingstarted-017.html>

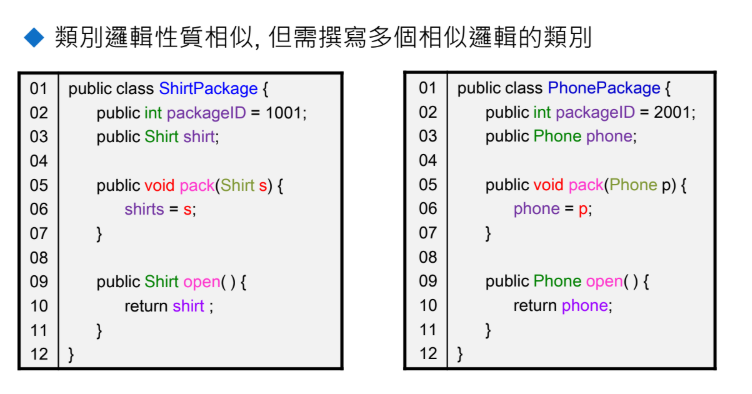
* 為解決同一屬性可能有多種型別的時候，取值就需規定型態存取(泛型)

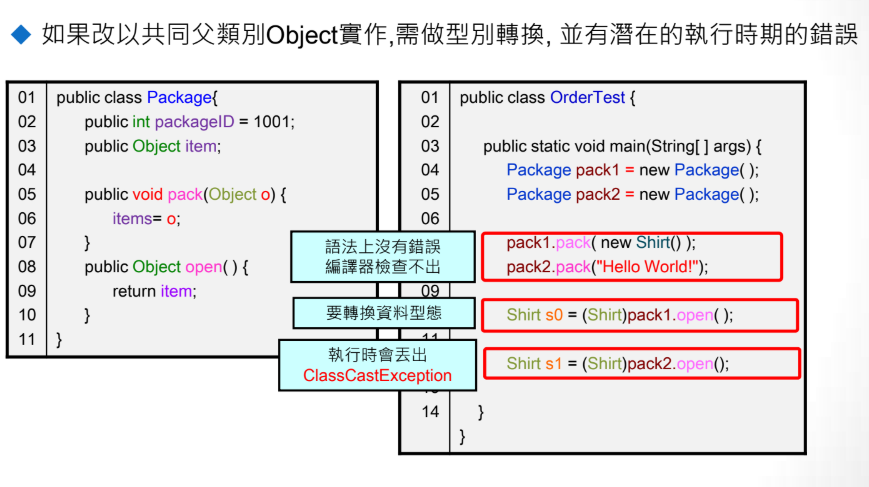


* 泛型的優點：

1. 避免頻繁轉型時潛藏的例外錯誤
2. 避免將不適當的格式放入集合中，編譯器會幫你檢查
3. 泛型過後取出的資料不用再轉型

* 還沒使用泛型前





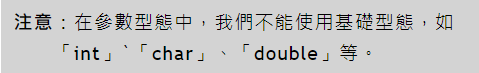
* 型別轉換基本上就有一定程度的風險，如果轉不成功會產生例外，寫判斷式嚴格來說也無法解決如果要改成這種寫法會產生的狀況
* 沒有使用泛型之前，建立的集合或陣列常常會因為型態或格式不符合對方要求，而得到無回應或是無法得到想要的結果，萬一出錯時要檢查程式碼的資料結構也會佔去很大時間(因為類別種類太多，程式碼相似的地方太多)

1. 泛型宣告

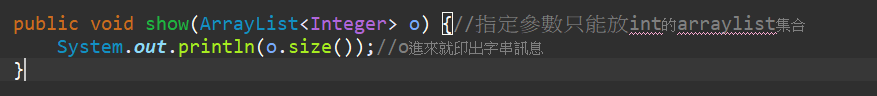
* 類別宣告後加上 「 <>」 來指定該物件集合的型態

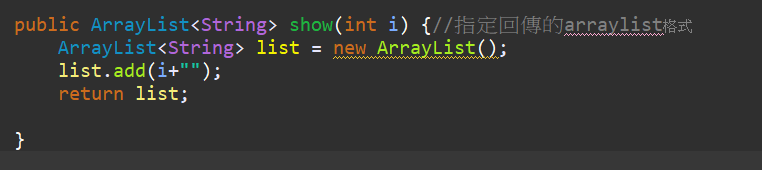


* 加入泛型後的list規定只能放string的資料

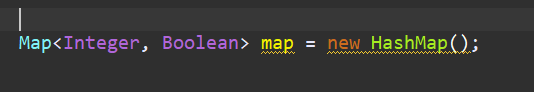


\*泛型規則內只能放物件類型

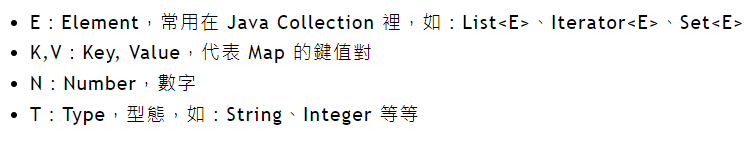




Map的泛型宣告要兩個



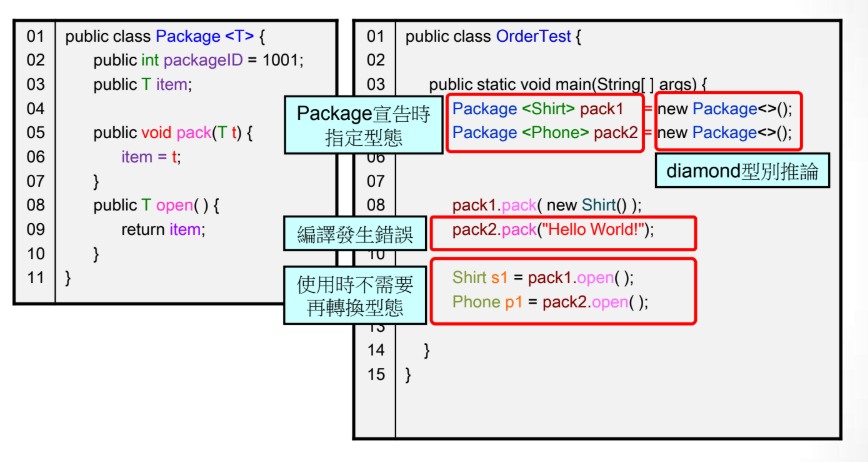
* 常用的泛型格式字母規範



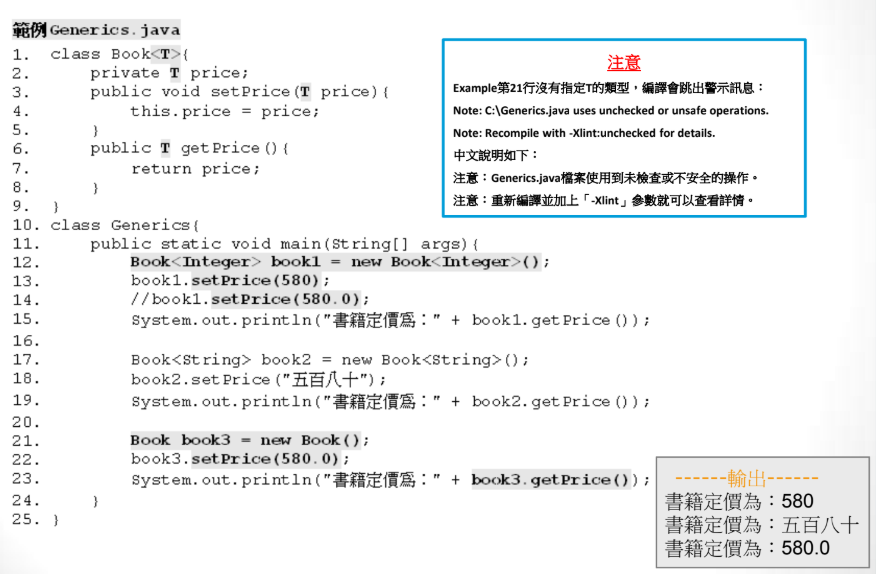
→可以前後都寫，也可以只寫前面



* 泛型後的檢查機制



* 泛指<T>的範例



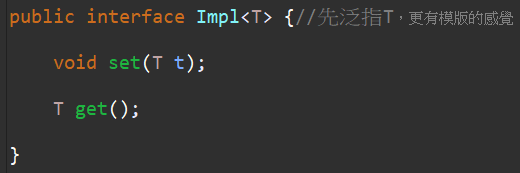
建立物件時需指定一基本型態

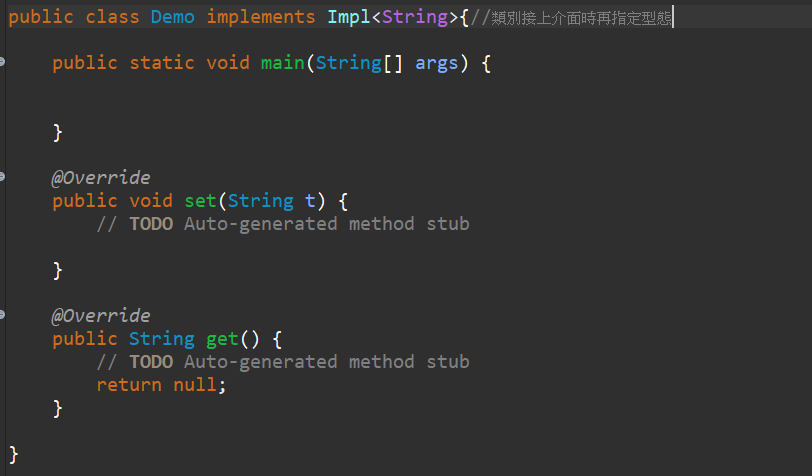
這邊沒有給型態，編譯就不會過

* 泛型給T時，代表型態未知，但宣告或建立時就要指定，但指定的型態不受限，藉由這樣來先把規則寫好，實現時再說



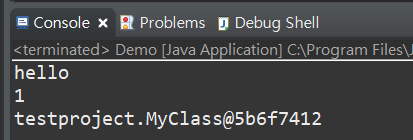
* 泛型介面





* 泛型方法

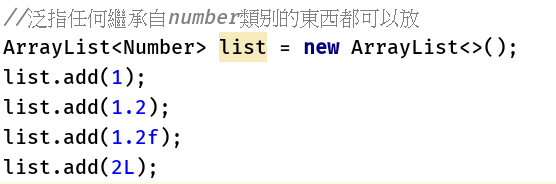


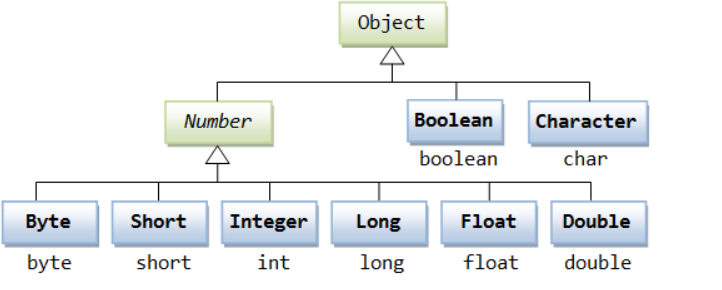


* 泛型「 ? 」 的使用

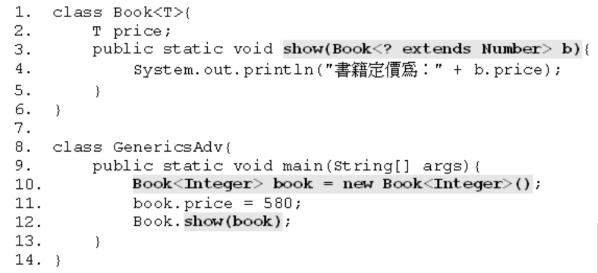


* 可泛型指定某一個大類別，之後任何繼承自它的都可以視為是他的泛型。



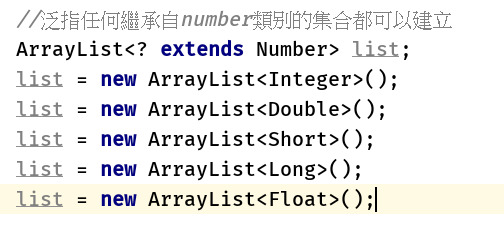


* 泛指某一型態的範例<? Extends Number>

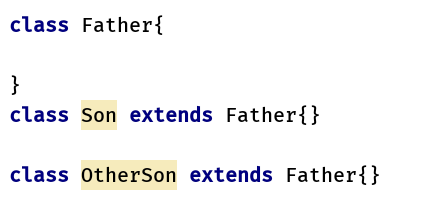


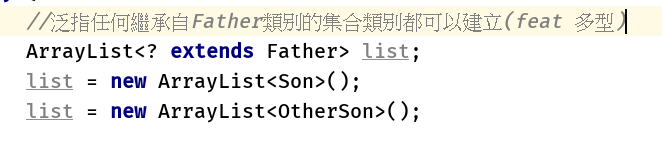
→泛指任何符合Number型態所建立的book物件

Ex：

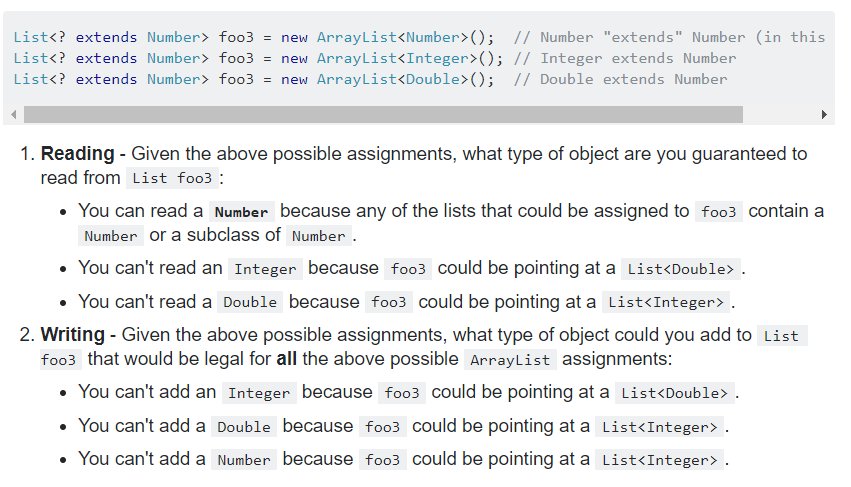


Ex:

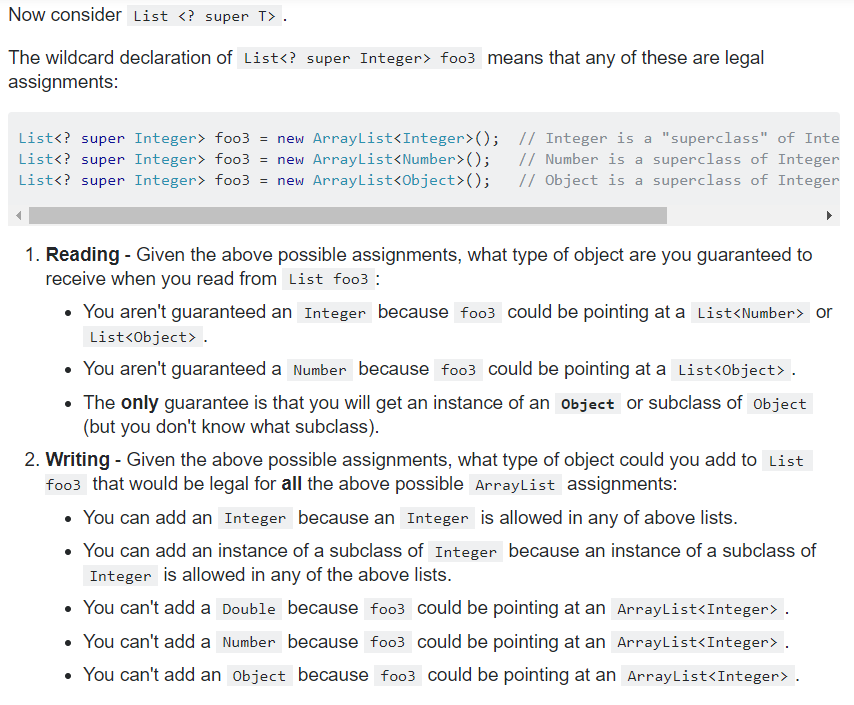




* 補充說明：<? extends T> & <? super T>的差別與限制



由以上說明可得知，使用? Extends的方式在新增元素時可能會導致無法確保新增至該泛型的資料是否可接受(泛型範圍太大)而被編譯器先行擋住。

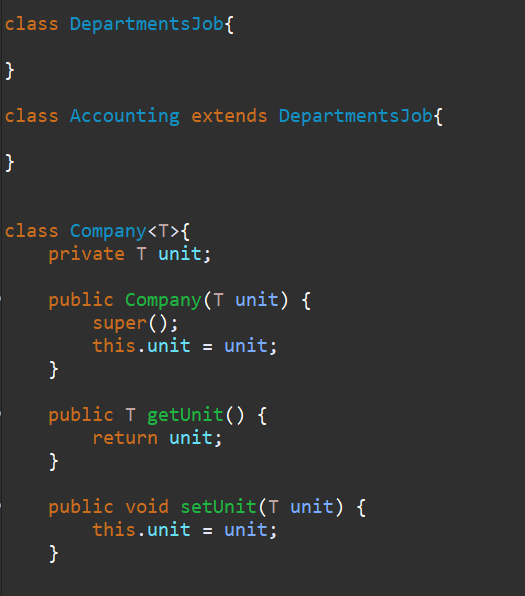


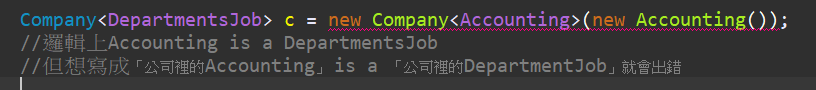
可放任何之於Integer類的父類別

* 由以上說明可以得知，當你指定泛型super至Integer(? Super Integer)時，比Integer高的父類別可以被指定，也可以被讀取，但新增時只能新增最小範圍的Integer(泛型已限定在Integer類別)，否則編譯不會過

<https://www.zhihu.com/question/20400700>

* <? extends T>(上層通配符) & <? super T>(下層通配符)是泛型中的通配符(Wildcards)和邊界(Bounds)的概念
* 上層通配符使用的場景：<? extends T>

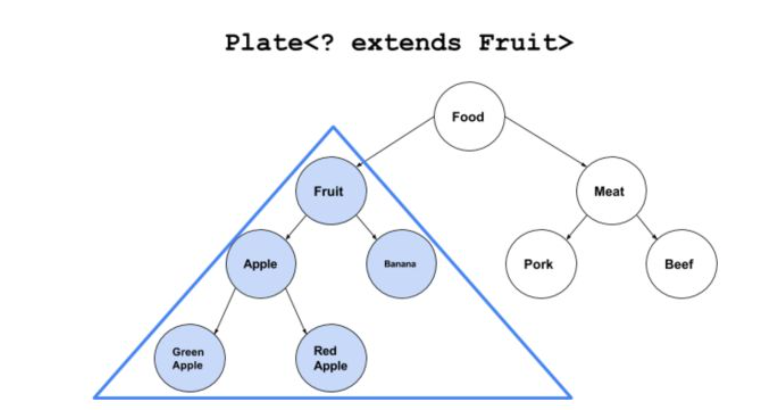




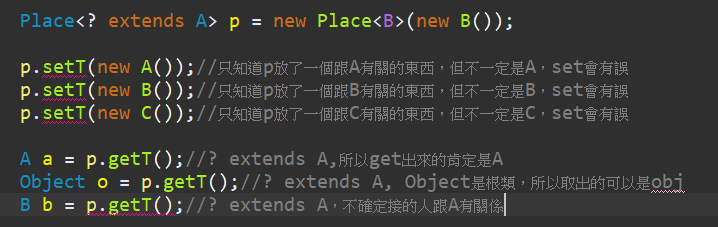
* 實際在使用的時候會發現一個問題，那就是容器內的物件(DepartmentsJob & Accounting)就算彼此有繼承關係，但泛型時會認為是獨立的個體，所以無法將Company<DepartmentsJob> 的物件傳給Company<Accounting>，兩個被視為沒有關係的物件，但在商業邏輯上說不通
* 這時如果使用<? extends T>(上層通配符)就能說得通了！它的意思是：「什麼部門都可以放到這個公司裡講」



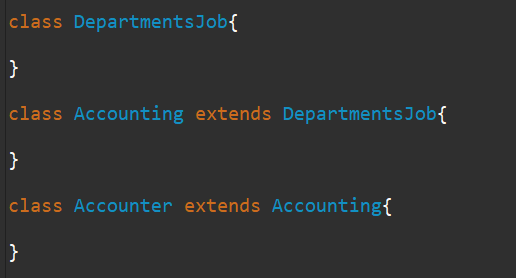
* <? extends T>這樣的寫法就是把「?」視為T的子類，所以就變成任何繼承自T的都可以放，而T又可以是任何類，使用上就可彈性
* 上述這樣改寫後，Accounting類別可以和DepartmentsJob產生關係

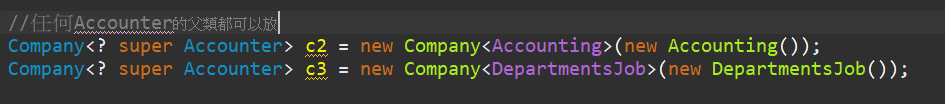


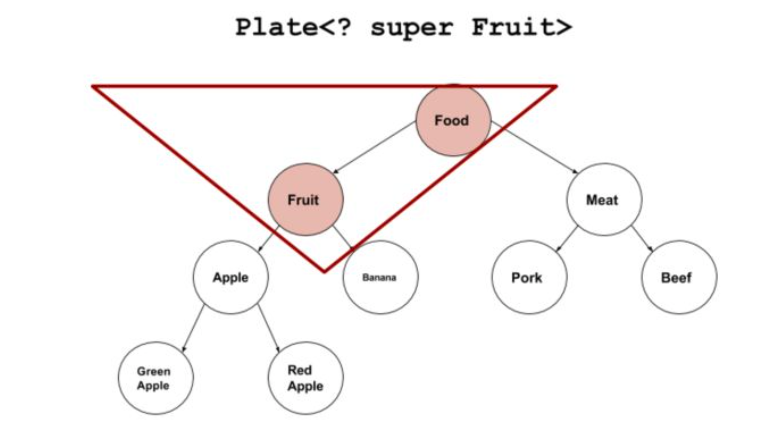
存取的限制：



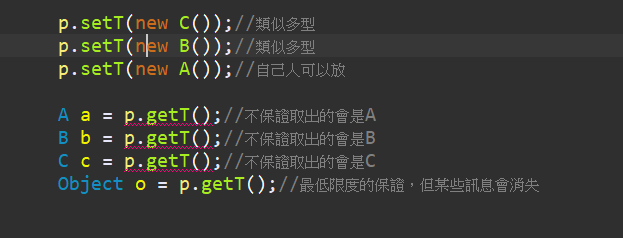
* 下層通配符使用的場景：<? super T>
* 意思就是「?」是屬於T的父類，所以任何T的父類都可以放







取值的限制



1. 包覆類別Wrapper Class

Java的Wrapper就是類別型態，為了提升效率而將資料包成物件，因為物件可以挾帶更多資訊

* 基本定義

1. 每一個基本型態會對應到一個包覆類別，即將基本型態視為物件的形式



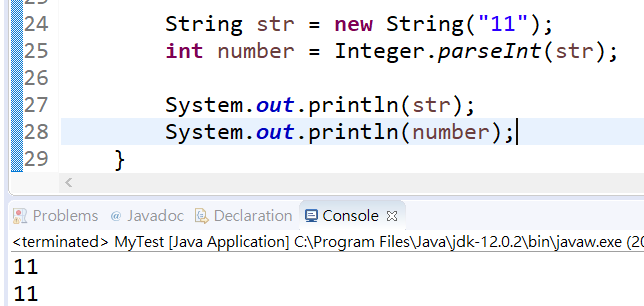
1. 由於泛型在指定型態的時候需要以類別為單位，所以要使用包覆類別來指定
2. 包覆類別所裝的型態在建立後就不可改變，可增加穩定性
3. 提供下列方法將包覆類別的字串轉成基本型態



* 使用parse方法之前，一定要先確定此字串轉型時符合該型別的值ex: parseInt時，字串須為數字：”12”，”a”這種不是數字的轉型會掛掉

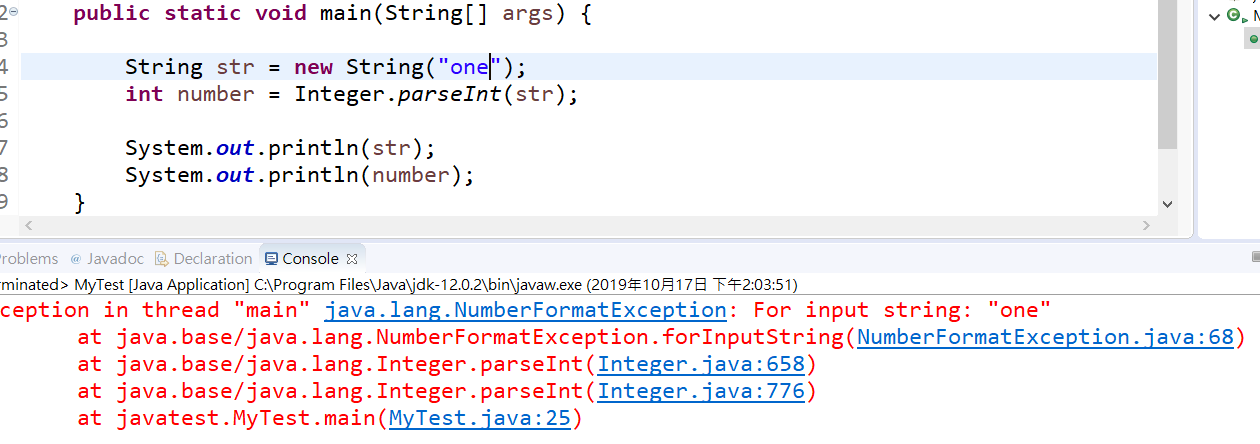
Ps：以上為靜態方法呼叫

Ex：

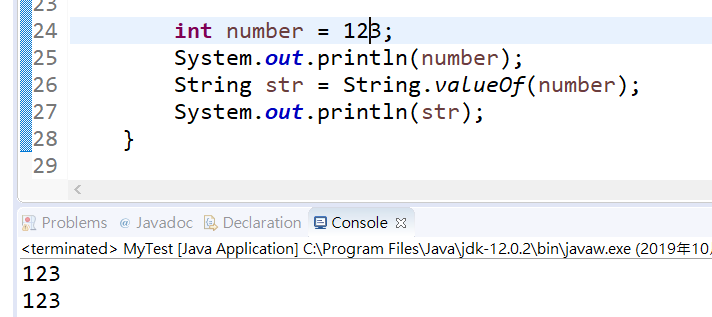


→第一個11為字串，第二個為int

* 注意：轉換前要確定字串內的值為對應的數值，否則會出錯：



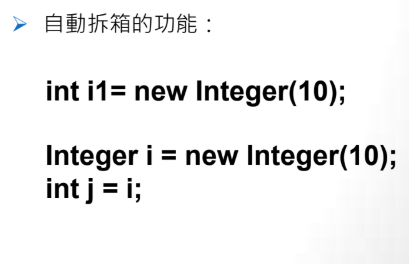
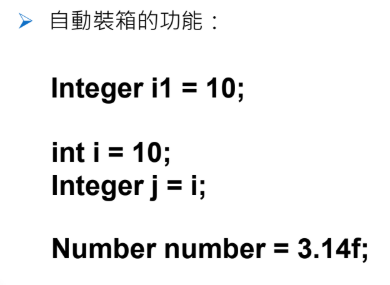
* 數值轉成字串



* 第一個123為int, 第二個123為字串格式
* 自動裝箱、自動拆箱的概念(Autoboxing & unboxing)

<https://github.com/JustinSDK/JavaSE6Tutorial/blob/master/docs/CH04.md>

→基本型態對應到的包覆類別會有自動轉型的功能



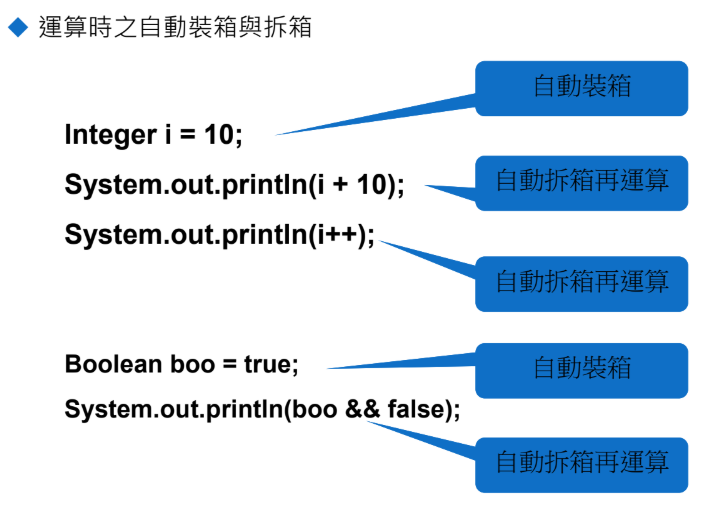
補充：java的Number類別

Number類別為java中管理所有數值相關的類，其子類別分別為：Integer, Long, Byte, Double, Float, Short。也就是常用的型別宣告類型，以大寫的型式呈現，以此宣告的時候，即為一個物件

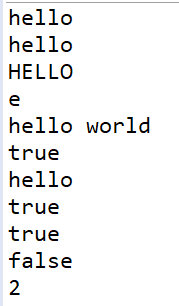


利用自動裝箱、拆箱的概念，可以將此物件當做一般數值使用



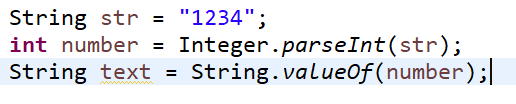


* 包覆類別的好處
* 類別即物件的藍圖，有了類別定義好的方法，我們便可以利用建立出來的物做出更多事情，舉例來說

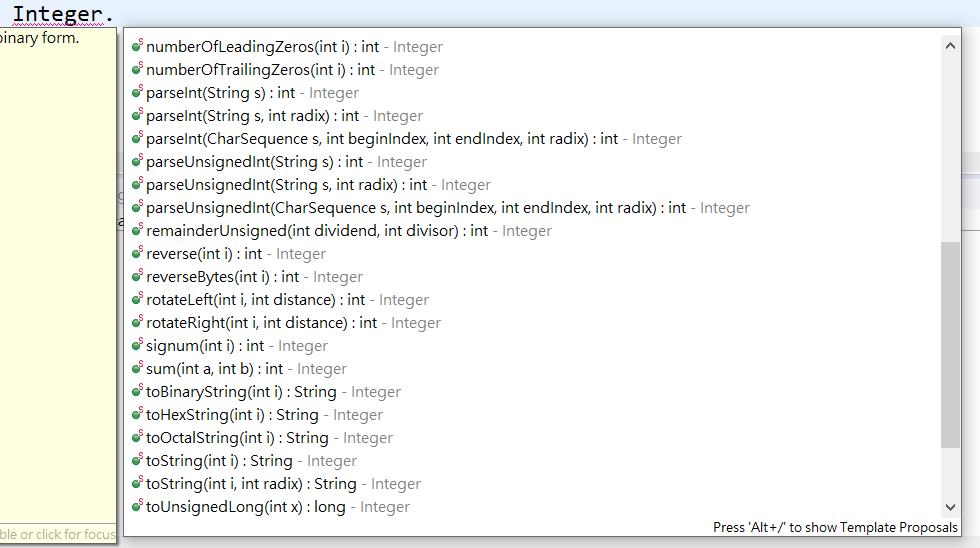


* 可使用方法自由轉換型態

→字串與數值間的轉換



實際上各包覆類別可做到的事情超多！

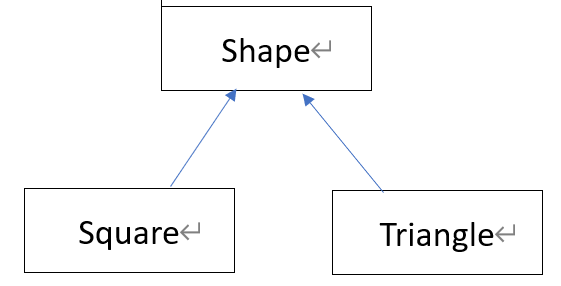


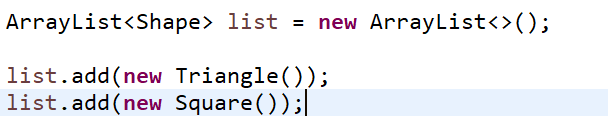
* 泛型也可用使用物件為指定型態



→指定放入list的內容只能是Shape物件

→可搭配多型使用





→可放入任何繼承自shape類別的物件

\*補充：

．電腦儲存單位

