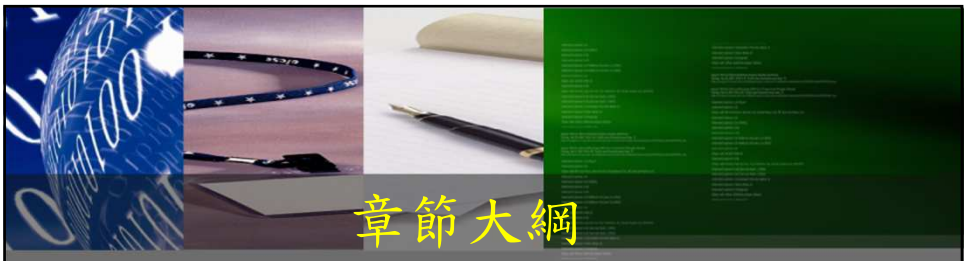
第七章 物件導向設計：類別與物件

課前指引

在本章中，我們將正式進入物件導向程式設計的領域，雖然我們在前面章節，已經使用過某些Java 類別庫的類別或物件（例如：Math 類別、String 物件），但卻未曾學習如何建立一個物件（事實上建立物件必須先宣告類別）

在本章中，我們將從頭教您如何使用Java 並以物件觀點設計程式，逐漸體驗物件導向程式設計帶來的好處，尤其是在發展中大型專案時，物件導向程式設計更是目前最佳的解決之道。



章節大綱

7.1 物件導向程式設計

- Java是一個**物件導向程式語言(OOPL；Object-Oriented Program Language)**
  - 所謂物件導向設計理念，就是利用軟體模擬現實生活中實體所擁有的特性與行為，這些實體就是物件，而每一個物件都可以擁有各自的屬性及方法
  - **物件導向程式設計**(OOP ；Object-Oriented Programming)是一種以物件觀念來設計程式的程式設計技巧，它透過物件的方法產生互動以完成程式要求。

3

7.1 物件導向程式設計

- 在純物件導向的程式中，每一個運作實體都可視為某種類別衍生出來的物件
  - 而類別則較具抽象的概念，也就是某些具有共同特性物件的集合
  - 換句話說，物件就是類別的實體。
  - 物件導向具有封裝、繼承、多型等特性，原則上每個物件相互獨立且無關聯性
  - 而物件導向程式設計就是依照物件的方法產生互動以完成要求。

4

7.1 物件導向程式設計

- 傳統的結構化程式設計將問題切割成許多小問題（模組），這些模組可能會存取同一個資料結構，這將導致模組之間的獨立性質降低
  - 舉例來說，某兩個模組可能都會修改陣列資料結構中的元素，因此當我們希望將某一個模組的操控對象從「陣列」改為其他的資料結構時，另外一個模組也必須跟著改變。
  - 因此，結構化程式設計雖然對於某些問題可以快速尋得解決方案，但對於日後的維護則顯得不足

5

7.1 物件導向程式設計

- 故後來又發展了以物件為基礎的程式語言。並且又可以分為**物件基礎程式語言**與**物件導向程式語言**兩類。
  - 這兩類程式語言都是以**物件**為出發點，藉由物件與物件之間的互動完成問題的解答，比較符合真實世界環境。
  - 物件基礎程式語言最著名者為JavaScript
    - 它允許設計者使用物件來設計程式，但無法提供繼承等機制來擴充程式，不利於大型程式的開發，因此常用於客戶端網頁等程式設計。
  - Java是物件導向程式語言
    - 提供完整的繼承、封裝、多型等特性，適合開發大型程式。

6

7.1 物件導向程式設計

- 物件之間的互動是透過訊息(Message)的傳遞來達成
  - 發送訊息的物件稱之為發送者(sender)
  - 接受訊息的物件稱之為接收者(receiver)
  - 當物件需要其他物件服務時，會發送訊息給接收者，接收者在收到訊息後，會執行符合要求的方法完成任務以提供服務。
  - 由於每一個物件都是一個獨立的個體，因此改良某一物件的內容時，其他不同類別的物件並不需要跟隨著變動。
    - 如此更有助於大型或超大型方案的日後維護與修改。

7

7.1.1物件(object)

- 真實世界中的所有具體或抽象的事物，都可以將之視為一個『物件』。
  - 例如：您可以把一架飛機想像成是一個**物件(Object)**；而飛機的零件（例如：座椅、引擎、操縱桿）則是較小的物件
    - 明顯地，這些物件仍舊可以再細分為更小的物件（例如：螺絲釘）。
- Java的物件實際上是一些程式碼和資料的組合，物件必須能夠單獨成為一個完整單元，也可以組合成更大的物件。
  - 例如：一個按鈕或一個表單都是一個物件。而一個應用程式最大的物件就是應用程式的本身。

8

7.1.1物件(object)

- **屬性 (Property)**
  - 物件擁有許多特性(attribute)，這些特性代表了一個物件的外觀或某些性質
    - 例如：民航機物件的最高速度、民航機物件的重量、載重量…等等都可以用來代表飛機的某些特性
    - 這些特性在物件導向程式設計中稱之為**屬性(Property)**
    - 事實上，有的時候在取得物件的某些屬性時，所得到的也可能是另一個（子）物件（若該屬性的資料型態是一個類別）。
      - 例如：民航機的引擎也是一個物件，它可以由更多更小的零件來組成，同時也存在自己的**方法(Method)**，例如：引擎點火。
  - 在程式設計或執行階段，我們可以藉由改變屬性值來改變整個物件的某些特性，完成我們想要的物件表示形式。
    - 例如：將民航機物件的機殼漆成藍色，將按鈕背景顏色設為黃色

9

7.1.1物件(object)

- 在Java語言中，**屬性(Property)**被稱為**Field**（有人翻譯為**欄位**；本書有時也翻譯為**成員資料變數**），並且被區分為不同的等級
  - 某些等級的成員資料變數不允許外部程式修改
    - 必須透過該物件的方法，才能夠修改這些成員資料變數
  - 具備保護資料的功能。

10

7.1.1物件(object)

- **方法（Method）**
  - 每個物件都擁有不同數量的行為(method)，這些行為在物件導向程式設計中稱之為**方法(Method)**
  - 所謂**方法**，也就是為了完成該物件某些目標的處理方式。
    - 例如：飛機類別的物件有許多方法使得飛機變得有些用途，這些方法如起飛、降落、逃生等等。
  - 每個方法都有許多的細節
    - 例如起飛，可能包含『發動引擎、、、、直到拉動操縱桿』，這些就是起飛方法的細節。

11

7.1.1物件(object)

- 在Java中，**方法**也稱之為**method**（有些書籍翻譯為方法；本書翻譯為**方法**或**函式**，也就是**成員函式**）。
  - 同時Java的method也和Field一樣，被區分為多種等級
    - 某些等級的成員函式同樣不允許外部程式呼叫執行（只能做為內部成員函式呼叫執行的對象）
  - 達到封裝物件的功能。
- 由他人完成的物件必須提供許多關於物件的方法，如此一來，我們就不需要了解這些方法的細節，而能夠快速運用物件來完成工作。
  - 例如飛機物件提供了起飛方法，我們只要指定執行起飛方法，而不需要了解起飛過程的種種細節，飛機就會起飛。

12

7.1.1物件(object)

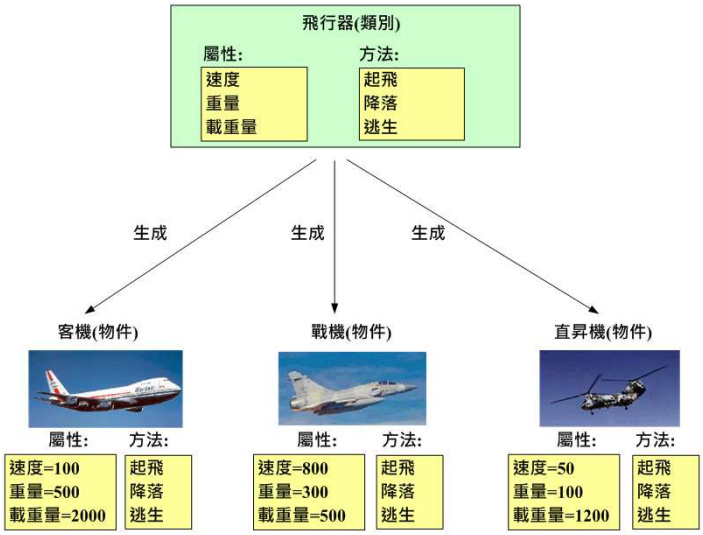


圖7-1 類別與物件關係圖

13

7.1.2類別(class)

- **類別**
  - 物件雖然是獨立的個體，不同的物件可以擁有相同的屬性及方法，例如：「一架民航機」和「一架戰鬥機」各是一個物件，但兩者的速度、爬升力、重量、載重量雖不相同，但卻同樣擁有這些屬性以便表達完整物件的各個特性。
  - 所以，不同的物件若擁有共同的屬性，但因為屬性內容的不同，因此可以創造出同類性質但卻獨立的不同物件。而同類型的物件，則構成了**類別(Class)**。
    - 例如每一個按鈕都是一個物件，屬於按鈕類別之下所建造出來的物件實體，但是由於按鈕的名稱不同，因此視為不同且獨立的物件。

14

7.1.2類別(class)

- 更明確地說，**類別才是物件導向程式設計最基本的單元**
  - 以上例來說，「民航機」和「戰鬥機」都是物件，而『飛行器』或『飛機』才是類別。
  - 不同的只是在建立該類別的實體物件時，給予不同的屬性而已。
- 我們必須先定義類別，然後才能夠透過類別宣告各個屬於該類別下的物件，接著再設定物件的屬性來代表該物件某方面的特性，並使用物件的方法來操作物件。

15

7.1.2類別(class)

- 類別是許多同類物件的集合
  - 同類物件代表擁有相同屬性（資料變數）及方法（函式）的物件
    - 例如：每一台『飛行器』類別下的物件，都包含「速度」、「爬升力」、「重量」、「載重量」等變數，也包含「起飛」、「降落」、「逃生」等方法，以便完成工作。
    - 如同「民航機」和「戰鬥機」各是一個物件，而『飛行器』則是類別。
- 在純物件導向的程式中
  - 程式碼是以類別為基礎
  - 物件只是類別的實體
    - 就如同汽車是一個類別，它是一個抽象的名詞
    - 而車牌號碼為010、015等的車輛是汽車類別的兩個物件
  - 實際運作時「通常」依靠的是物件。我們將於下一小節中，更深入介紹物件導向程式語言的特性。

16

7.1.2類別(class)

老師的叮嚀

有些人在說明類別與物件的關係時，會將類別視為一種板模，而物件則是依靠板模建立出來的實體，就如同雞蛋糕板模是類別，而每一個實際做出來可以吃的雞蛋糕是物件。
這種比喻方式在某種程度上說得通，不過仍舊不能完全說明Java類別與物件的實際應用，因為Java也允許在未產生物件實體時，存取變數或執行方法，也就是靜態變數與靜態方法。

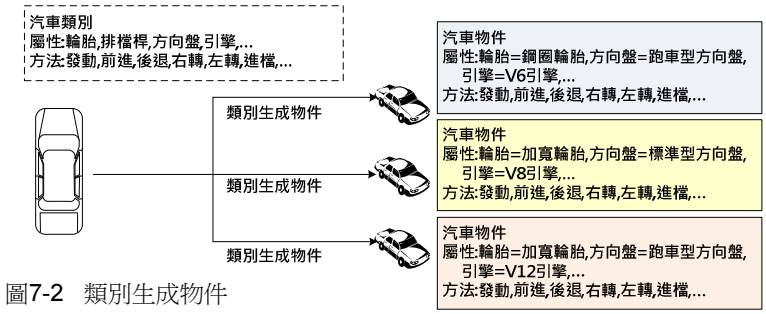


圖7-2　類別生成物件

17

7.2物件導向的三大特性

- 物件導向程式設計具有某些特點如下，使得它更適合用以開發、管理大型程式。
  - 封裝性、繼承性、多型性
- **封裝性(encapsulation)：**
  - 將物件區分為可被外界使用的特性及受保護的內部特性；
  - 除非是允許外部程式存取的資料，否則外部程式無法改變物件內的資料。如此就能夠達到封裝保護資料的特性。
  - 物件導向程式設計將物件的資料與方法至少區分為三種等級：（Java除了下列等級之外，尚有Package等級）
    - public　　：公用等級。
    - private　　：私用等級。
    - protected　　：保護等級。

18

7.2物件導向的三大特性

- 其中私用(private)等級的資料與方法，只允許相同類別內的程式碼取用
- 而保護(protected)等級的資料與方法，只允許相同類別及衍生類別（何謂衍生類別後述）的程式碼取用
- 至於公用(public)等級的資料與方法，則開放給任何程式碼取用。

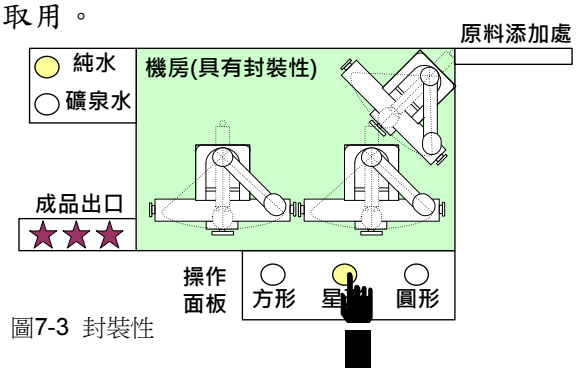


圖7-3 封裝性

19

7.2物件導向的三大特性

機房物件

公開屬性：原料、成品

私有屬性：水……

公開方法：使用純水、使用礦泉水、方形餅乾製作、星形餅乾製作、圓形餅乾製作

私有方法：機器A啟動、機器B啟動、機器A快轉……

- 以圖7-3為例，食品餅乾工廠內部機房就具有封裝特性
  - 對外，它只開放原料與成品兩個可存取的屬性
  - 並且提供製作方形餅乾、製作星形餅乾、製作圓形餅乾等操作方法的按鈕。
    - 原料屬性對外是可存取的，例如您可能放入的是三號麵粉，但也可以改為二號麵粉。
    - 成品也是可以存取的，例如您可以把餅乾吃掉、裝箱或捏碎報廢。
  - 被封裝的屬性則是不可存取的，除非透過外部方法間接存取
    - 製作過程可能需要水，而我們無法直接變動水的種類，必須透過對外的外部方法（也就是純水或礦泉水按鈕）來完成
    - 我們無法隨意將水變更為海洋深層水，因為無法進入到機房內部。

20

7.2物件導向的三大特性

- 同理，由於我們無法進入機房內部直接操作機器，所以只能透過對外操作面板上的三個按鈕選擇要執行的製作程序。
- 當我們按下某一個按鈕時，相當於呼叫機房物件的一個**公開**方法
  - 此方法的細節可以動用到機房內的所有資源
  - 機房的設計一但完成後，這些方法的細節也就固定了
- 對於使用機房物件的使用者而言
  - 他只能選擇執行某一個按鈕方法
  - 無法更動按鈕方法的執行內容細節。

21

7.2物件導向的三大特性

- 假設我們是設計機房物件的程式設計師
  - 我們可以進入機房，也可以設計各種餅乾的製作細節
  - 但即使進入了機房，當我們面對每一台機器時，由於每一台機器也都是一個物件（同樣也具有封裝性），故我們只能操作機器對外的按鈕，而無法進入機器的內部。
- 因此，封裝性使得問題變為分層負責的狀態
  - 設計物件者有責任將物件設計的完整且不會出錯，也需要留下一些對外可操作的方法，必要時也可留下一些外部可存取的屬性。
  - 而使用物件者，則只能夠透過物件對外的屬性與方法來利用物件達到自己的需求。

小試身手7-1
「電子鬧鐘」物件具有封裝性，試以電子鬧鐘為例，說明電子鬧鐘的哪一些屬性為公開屬性？哪一些為私有屬性？哪一些為公開方法？哪一些為私有方法？

22

7.2物件導向的三大特性

- **繼承性(inheritance)：**
  - 繼承性是為了達成重覆使用目的所採取的一種策略
    - 例如：一個滑鼠類別只要加上滾輪裝置，就變成了滾輪滑鼠
    - 但滾輪滑鼠也同樣可以上下左右移動改變指標位置，也可以按兩下執行程式，只不過現在又多了一個滾輪使得瀏覽網頁時更加方便
    - 因此，這個滾輪滑鼠類別可繼承滑鼠類別再加以擴充。
  - 在物件導向程式設計中，允許使用者定義**基底類別**(base class)與**衍生類別**(derived class)
    - 衍生類別**繼承**基底類別的屬性及方法，並「加入新的屬性及方法」或者改寫(override)某些繼承的方法，改成適用於本身的方法。
    - 在開發大型程式時，我們可以延續已完成的技術，再加以擴充。

23

7.2物件導向的三大特性

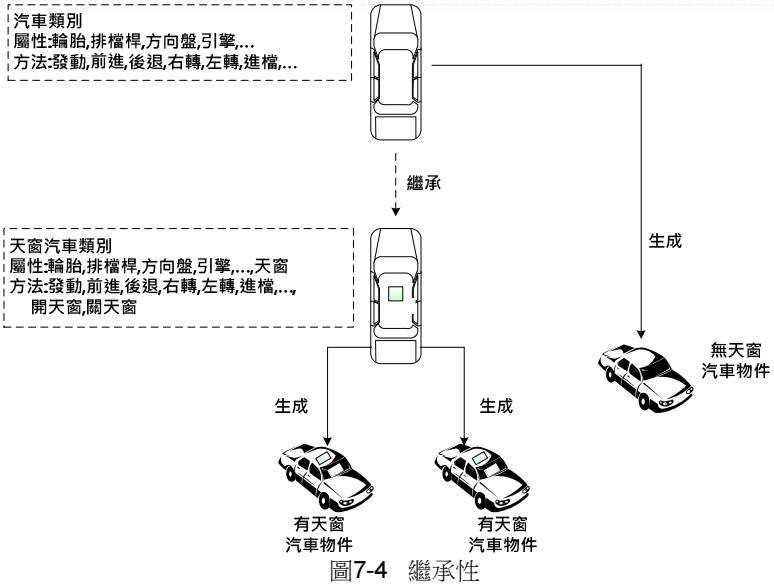


圖7-4 繼承性

24

7.2物件導向的三大特性

- 多型性(polymorphism) ：
  - 就多型性而言，其實它是一個非常抽象的名詞，代表著一種彈性。
  - 瓦斯爐、電磁爐、熱水瓶都可以煮開水，所以『煮開水』其實是一個相當抽象的名詞
  - 使用瓦斯爐、電磁爐、熱水瓶的煮開水方式都不一樣，但『煮開水』一詞卻涵蓋了這些差異性的行為，而此種特質則稱為「多型性」。
- 多型的實作方法有很多種
  - 其中一種是改寫(override；或稱覆寫)，它與繼承有關
    - 瓦斯爐、電磁爐、熱水瓶衍生類別都繼承自加熱器基底類別，該基底類別中定義了加熱的方法，我們必須在衍生類別內，改寫這個方法，使得加熱的細節有所不同
    - 如此，不同衍生類別所產生的物件，執行方法時的細節就會不同

25

7.2物件導向的三大特性

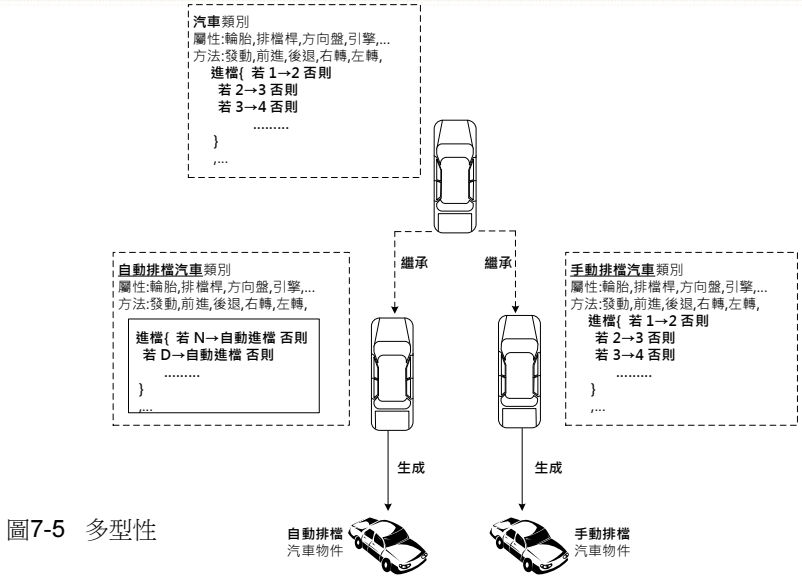


圖7-5 多型性

26

7.2物件導向的三大特性

- 另一種屬於多型的技術稱之為多載(overload)
  - 允許程式中出現相同名稱但不同署名的函式，在上一章已經介紹過。
- 多型是一種呼叫相同名稱函式，但卻可執行不同函式內容的機制，因此，又有人將『多型』稱之為『同名異式』。
  - 編譯器實作多型，依靠的是動態繫結來達成。
  - 傳統上（在非物件導向語言中），我們在呼叫函式時，必定會註明要呼叫哪一個函式，而編譯器在編譯階段就可以知道各個函式在記憶體中的位址（當然這是個相對位址），因此只要我們註明了要呼叫哪一個函式，編譯器在編譯階段就可以將該位址記錄在機器碼中，所以執行到該機器碼指令時，就會去呼叫那一個函式，不可能執行別的函式。

27

7.2物件導向的三大特性

- 而動態繫結則是
  - 程式碼中可能有好幾個同名的函式，我們雖然指明了要執行哪一個名稱的函式，但並未完全指定要執行同名中的哪一個函式。
  - 因此，編譯器在處理這種函式呼叫時，會編譯為由傳入引數來決定要執行哪一個函式，而這個引數將會是函式的起始位址，所以只有等到程式真的執行到該機器碼並傳入引數時，才會知道要執行的是哪一個函式。

28

7.2物件導向的三大特性

- 物件導向設計的三大特性，Java全部都支援
  - 在本章中將學習到Java的**封裝性**。
  - 並且在本篇的後面章節將介紹Java的**繼承性**及**多型性**。
- 再說明一次關於Java的物件與類別。
  - 物件是由類別所衍生，例如透過String類別，我們可以宣告一個String物件。
  - 而物件名稱是一個參考。透過new我們可以產生物件的實體，並將某一個物件名稱指向它，以便操作該物件。
    - 類別有fields與methods，而該類別衍生的物件也有相同的fields與methods。

29

7.3 Java的類別

- 陣列是將相同資料型態的變數集合起來形成一個結構，而類別則可以將不同資料型態的變數集合起來形成一個結構
  - 這些變數將成為類別的成員
  - 不僅如此，我們還可以為類別定義屬於該結構的函式。
- 類別是物件導向程式最基本的單元，是產生同一類物件的基礎
  - 類別可以由使用者自行定義，以下是在Java中定義類別的語法：

30

7.3 Java的類別

- 定義類別語法：

```
[封裝等級] [修飾字] class 類別名稱 [extends 父類別] [implements介面]
{
   [封裝等級] [修飾字]  資料型態  field名稱1;
   [封裝等級] [修飾字]  資料型態  field名稱2;
                :                                  宣告fields
                :
   [封裝等級] [修飾字]  回傳值資料型態  method名稱1(參數串宣告)
   {                                                定義method
         method的內容
   }
   [封裝等級] [修飾字]  回傳值資料型態  method名稱2(參數串宣告)
   {                                                定義method
         method的內容
   }
                :
                :
                :
}
```

31

7.3 Java的類別

- 語法說明：
  - (1)類別定義的第一行，我們可以視為類別的宣告。當中所有的[]都可以省略，因此我們暫不討論（後面會陸續介紹）
    - 而其中類別的[封裝等級]，只有public或不出現兩種，當「出現public」時，代表該類別為「主類別」，而主類別必須與檔名取相同的名稱。
    - 同時，只有被宣告為public的類別才能被import進來。在本章中，我們宣告類別時，都不會出現該欄位（除主類別外）。
  - (2)在類別中定義的資料與函式，分別稱為fields與methods。關於fileds及methods的翻譯與物件導向設計的對照請見下表。

32

7.3 Java的類別

Java原始說明文件	物件導向設計	C++	本書翻譯	其他
field	屬性(Property)	成員變數	資料欄位、成員變數、成員資料、屬性	有時不翻譯
method	方法(Method)	成員函式	函式、成員函式、方法	有時不翻譯

表7-1 filed與method翻譯對照表

- (3)類別中可以對fileds及methods定義多種『封裝等級』如下：
  - public：公用等級資料型態
  - private：私用等級資料型態
  - protected：保護性等級資料型態
  - 空白：package等級

33

7.3 Java的類別

- (4)在類別中，『不同的存取等級』代表『不同的資料保護方式』，資料之所以需要保護，主要是為了讓各類別的資料獨立開來，不易被其他不相干的函式修改，達到物件導向程式設計中，「資料封裝」及「資料隱藏」的功能。
- (5)根據上述定義類別的語法，可利用下圖來示意

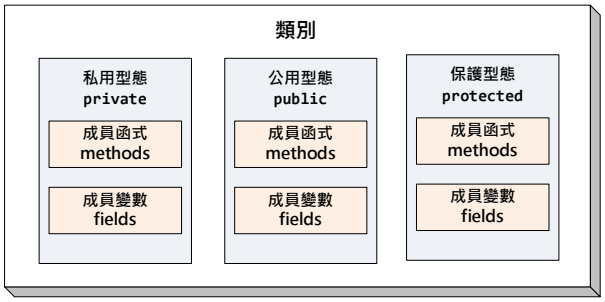


圖7-6 資料等級
（省略package等級）

34

7.3 Java的類別

- (6)私用等級資料只能被同一個類別的成員函式存取；而公用等級的資料則除了類別內的成員可以存取之外，其他外部的函式也都可以存取宣告為公用等級的資料。
- (7)宣告fields與定義methods的順序可以交錯。
- (8)field可以設定初值且該初值會在物件實體產生時，自動被設定。
- (9)類別的命名在第二章曾經提過，本書所採用的習慣為第一個字母大寫（通常命為C，代表class），其後由一個以上的單字所組成，每個單字的開頭字母為大寫，其餘為小寫
  - 例如：CPerson, CStack, CCircularQueue。

35

7.3 Java的類別

- 【範例1】
  - 定義一個CStudent類別
    - 將其成績、地址、電話宣告為『私用等級資料型態』；
    - 學號、姓名、科系宣告為『公用等級資料型態』。

```
class CStudent
{
  private float  score;
  private String address;
  private int    phone;
  public int     id;
  public String  name;
  public String  major;
}
```

```
class CStudent
{
  private float  score;
  public int     id;
  private int    phone;
  public String  name;
  private String address;
  public String  major;
}
```

相同封裝等級的不一定要擠在一起

36

7.3 Java的類別

- 【範例2】
  - 延續範例1，在CStudent類別中
    - 宣告兩個『公用等級』的（成員）函式，分別為showId( )和showMajor( )
    - 函式用途分別為『顯示學號』及『顯示科系』。

```
class CStudent
{
  private float  score;
  private String address;
  private int    phone;
  public  int    id;
  public  String name;
  public  String major;

  public int showId( ){...}
  public void showMajor( ){...}

}
```

顯示並回傳學號，回傳型態為int

顯示科系，回傳型態為void

37

7.3 Java的類別

或

```
class CStudent
{
  private float  score;
  private String address;
  private int    phone;

  public int showId( ){...}
  public void showMajor( ){...}

  public  int    id;
  public  String name;
  public  String major;

}
```

顯示並回傳學號，回傳型態為int

顯示科系，回傳型態為void

- 範例說明：
  - 上述範例，當物件被實際建立之後，id, name, major可以被外部函式所取用，showId()與showMajor()也可以被外部程式所執行，但score、address與phone只能被同一類別的成員函式存取（例如showId成員函式）。

38

7.3.1 類別的定義

延伸學習：**組合與繼承**

物件導向是模擬真實世界來設計程式，而真實世界中有一些物件之間具有某些組合或繼承的關係，舉例來說，禿鷹和鳥的關係應該是「禿鷹」是「鳥」的一種，也就是「禿鷹」is a kind of「鳥」的概念，這個時候「禿鷹」類別就應該繼承「鳥」類別，在下一章中，我們將說明繼承機制。

而從上述的範例來看，score,address,phone 都應該是學生資料的一部分，也就是「住址」is a part of「學生資料」的概念，所以address 應該是CStudent 的一個欄位，這種概念稱之為組合（Composition）。有時候程式設計師會在某些狀況下誤用或濫用繼承機制，而那些狀況可能更適合使用組合來完成設計。

39

7.3.2 UML的類別圖

- UML(Unified Modeling Language)是一種用來作為物件導向系統分析的圖形化語言，包含許多種圖形
  - 由於在學完基礎的程式設計之後，不可避免地將會學習UML，因此本書在此先預告一些簡單的UML。
- 首先介紹的是UML當中的類別圖
  - UML類別圖很簡單
    - 就是將類別名稱放在第一欄，
      屬性（對應為Java 的欄位）放在第二欄，
      操作（對應為Java 的方法） 放在第三欄
    - 至於封裝等級則用以
      「+」對應public、
      「-」對應private、
      「#」對應protected、
      「~」對應package。
    - 至於靜態屬性則會加上底線來表示。

40

7.3.2 UML的類別圖

- 所以上述的範例2可用下列的UML圖形來表示
  - （其實在實務上，應該是先畫出UML 圖形，然後再依照圖形來設計出程式碼）

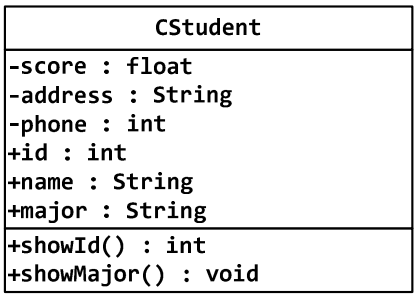


圖7-7 UML 的類別圖範例

41

7.4 Java的物件

- 類別是許多資料變數及函式合成的抽象資料型態，和陣列很相似。
- 當我們想要使用類別時，必須透過宣告變數並產生實體後才能夠使用（除了僅使用被宣告為static的變數與方法外）
- 而由類別宣告的變數，稱之為**物件參考**或**物件變數**。而產生的實體則稱為**物件實體(instance)**。

42

7.4 Java的物件

- 對照圖7-1，我們可以將Java的類別與物件重新描述如下圖：

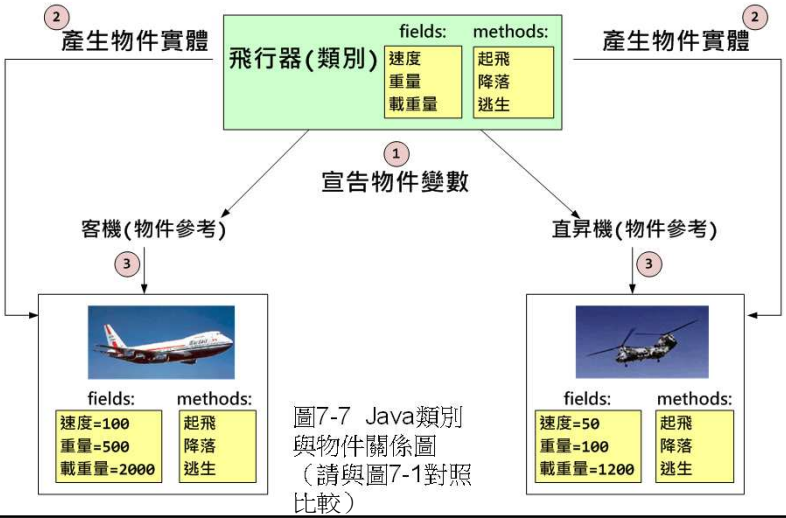


圖7-7 Java類別與物件關係圖（請與圖7-1對照比較）

43

7.4.1 宣告物件變數與產生物件實體

- 宣告物件變數
  - 宣告物件變數與宣告一般變數差不多
  - 類別宣告物件變數語法

```
類別名稱  物件變數名稱;
```

  - 語法說明：
    - 上述語法可以出現在任何類別內，也可以出現在任何method內。
    - 當宣告物件變數後，由於物件變數是一個參考，故開始時並未指向任何實體，此時的內容會被初始化為null。

44

7.4.1 宣告物件變數與產生物件實體

- 【範例】：宣告一個CStudent類別的物件變數，名稱為John。

```
class CStudent
{
  ……類別定義……
}

CStudent John;  //宣告物件變數John，本敘述應出現在類別或方法內
```

- 【範例】：宣告一個String類別的物件變數，物件名稱為str1。

```
import java.lang.String;
String str1;  //宣告String物件變數str1，本敘述應出現在類別或方法內
```

- 【說明】String類別已經被定義於java.lang Package，故載入該類別即可。

45

7.4.1 宣告物件變數與產生物件實體

- 產生物件實體
  - 產生物件實體使用的是new運算子，語法如下：
  - 【產生物件實體語法】 物件變數 = new 類別名稱(引數1,引數2...);
  - 語法說明：
    - 由於不屬於變數宣告，故只能出現在method內。
    - 除了可以產生物件實體，並將實體的參考設定給物件變數之外，還可以透過引數串列呼叫特定的建構子。
    - 建構子(Constructor)是一個特殊的函式，可以在物件實體產生時，進行一些初始化動作。
    - 如果不需要傳遞引數，則可以省略引數串列。
  - 【範例】：為CStudent類別的物件變數John產生一個實體。
    - John = new CStudent();

46

7.4.1 宣告物件變數與產生物件實體

- 【範例】：為String物件變數str1產生一個實體並設定其初始值為"Hello"。
  - str1 = new String("Hello");
- 【說明】
  - String類別不是我們所開發的類別，其建構子(Constructor)已經由JDK完成
  - 在Java說明文件中，可以查詢Constructor Summary文件段，會發現許多與類別同名的函式，這些函式就是建構子。

47

7.4.1 宣告物件變數與產生物件實體

- 宣告物件變數並產生實體
  - 可以於宣告物件變數時，同時產生物件實體，語法：

    類別名稱 物件變數名稱 = new 類別名稱(引數1,引數2...);
  - 語法說明：
    - 將之前的兩個語法合併。
    - 上述可以出現在任何類別內，也可以出現在任何method內。
  - 【範例】：宣告CStudent類別的物件變數John，並產生一個實體，使John指向該實體。
    - CStudent John = new CStudent();
  - 【範例】：宣告String物件變數str1，並產生一個實體，實體內容初始化為"Hello"，並使str1指向該實體。
    - String str1 = new String("Hello");

48

7.4.2 存取成員變數與執行成員函式

- 當物件實體被產生以後，我們就可以合法存取**公用等級**的fields以及執行**公用等級**methods（如果敘述處於同一類別則不限封裝等級皆可存取）
  - 存取方法很簡單，只要在物件與欲取用資料之間加上『.』運算子即可，如下語法：
  - 【存取成員資料與執行成員函式語法】

```
物件.成員資料
物件.成員函式(引數1,引數2,...)
```

49

7.4.2 存取成員變數與執行成員函式

- 語法說明：
  - (1)私用資料與函式，只能被同一類別內的成員函式存取與呼叫。（至於保護資料與函式則留待繼承一章再作說明）
  - (2)上述語法是外部程式的存取與呼叫語法。如果是同一類別內的存取與呼叫，則不需要指定物件，因為當它被執行時，它已經是物件本身。
- 【範例1】將John物件的id變數設定為9923807，設定John物件的name變數為"John"。
  - John.id = 9923807;
  - John.name="John";
- 【範例2】透過showMajor()成員函式顯示John的科系名稱。
  - John.showMajor();

50

7.4.2 存取成員變數與執行成員函式

- 【錯誤範例】私用資料與函式，只能被類別內的成員函式存取，所以下列是錯誤的敘述：

```
public class 主類別名稱
{
 static CStudent John;
 public static void main(String args[])
 {
    John = new CStudent();
    John.score = 85.5;                    //錯誤，因為score是私用資料
    John.address="台北市大安區";  //錯誤，因為address是私用資料
 }
}
```

- 【觀念範例7-1】：定義類別、宣告物件並存取物件的公用成員資料。
  - **範例7-1：**ch7_01.java（隨書光碟 myJava\ch07\ch7_01.java）

51

7.4.2 存取成員變數與執行成員函式

```
1   /*   檔名:ch7_01.java       功能:定義汽車類別與宣告3個物件    */
6   public class ch7_01          //主類別
7   {
8       public static void main(String args[])
9       {
10          CCar bus = new CCar();
11          CCar truck = new CCar();
12          CCar taxi = new CCar();
13
14          bus.name = "公車";
15          bus.wheel = 6;
16          bus.person = 40;
17
18          truck.name = "卡車";
19          truck.wheel = 8;
20          truck.person = 3;
21
22          taxi.name = "計程車";
23          taxi.wheel = 4;
24          taxi.person = 5;
25          //taxi.engine = "V16";
```

宣告CCar類別下的3個物件變數並產生實體，物件名稱分別為bus,truck,taxi。

可存取公用等級的變數

不可存取私用等級的變數

52

7.4.2 存取成員變數與執行成員函式

```
26
27          System.out.print(bus.name + "有" + bus.wheel + "個輪子");
28          System.out.println(",可載" + bus.person + "人");
29          System.out.print(truck.name + "有" + truck.wheel + "個輪子");
30          System.out.println(",可載" + truck.person + "人");
31          System.out.print(taxi.name + "有" + taxi.wheel + "個輪子");
32          System.out.println(",可載" + taxi.person + "人");
33      }
34  }
35
36  class CCar
37  {
38      public int wheel;
39      public int person;
40      public String name;
41      private String engine;
42  }
```

定義CCar類別，包含3個公用等級變數wheel、person、name，以及一個私用等級變數engine。

- 執行結果：

```
公車有6個輪子,可載40人
卡車有8個輪子,可載3人
計程車有4個輪子,可載5人
```


7.4.2 存取成員變數與執行成員函式

- 範例說明：
  - (1)當成功編譯此範例後，您會發現目錄底下除了ch7_01.class之外，還會出現CCar.class檔，這是因為我們除了主類別外，另外又定義了一個類別CCar。
  - (2)
    - 第14~16行，設定bus物件的公用等級變數。
    - 第18~20行，設定truck物件的公用等級變數。
    - 第22~24行，設定taxi物件的公用等級變數。
  - (3)第25行是不合法的敘述，因為engine是私用變數，所以**不允許外部程式（非同類別的函式敘述）存取該變數**
    - （但允許相同類別下的各級成員函式存取，詳見下一節）。

7.4.3 UML的物件圖

- UML的物件圖只有兩欄
  - 第一欄記錄的是物件名稱以及所屬類別，並且會加上底線來表示。
  - 第二欄則是物件的屬性（即欄位），以及屬性值。
    - 由於所有同類別的物件擁有的操作是一樣的，所以物件圖沒有操作欄。
  - 以範例7-1為例，其物件圖如下：

bus : CCar
wheel : int = 6 person : int = 40 name : String = 公車 engine : String = null

truck : CCar
wheel : int = 8 person : int = 3 name : String = 卡車 engine : String = null

taxi : CCar
wheel : int = 4 person : int = 5 name : String = 計程車 engine : String = null

圖7-9　UML 的物件圖範例（對應於範例7-1）

55

7.5 方法

- **方法(method)**又稱為**成員函式**，它與成員變數一樣，除空白（未宣告封裝等級）的package等級之外，封裝等級還可宣告為public、private、protected等三種等級。
  - 並且只有public等級的成員函式可以被外部程式（非同類別的函式敘述）呼叫執行
  - private等級的成員函式只能被其他同類別的成員函式呼叫執行但不可以被外部程式呼叫執行
  - protected等級留待繼承一章再作說明。

56

7.5 方法

- 目前為止，我們了解到private等級的成員變數或函式無法被外部程式所取用與執行，因此能夠達到物件導向「封裝」的目的。
- 而public等級的成員變數或函式可以被外部程式所取用與執行，是物件導向封裝時對外唯一的管道，同時private等級的成員變數或函式也可以被public等級的成員函式呼叫
- 外部程式如果想要取用private等級的成員變數，可以先呼叫public等級的成員函式，再由public等級的成員函式修改private等級的成員變數
  - 如此不但達到封裝的目的，也可以在類別中增加了設計程式的彈性。

57

7.5 方法

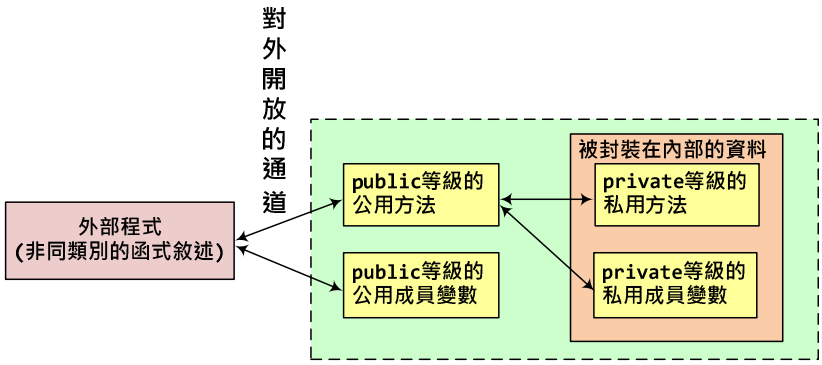


圖7-10 外部程式存取private等級的資料必須透過public等級的方法來完成

58

7.5.1定義方法

- 【在類別內定義成員函式語法】

```
[封裝等級] [修飾字]   class   類別名稱   [extends 父類別] [implements介面]
{
   [封裝等級] [修飾字]   回傳值資料型態   method名稱     //定義method
   {
        method的內容
   }
}
```

- 語法說明：
  - 關於[修飾字]部分，我們暫時省略，而封裝等級則為public、private、protected或不填。

59

7.5.1定義方法

- 【範例】請宣告並定義CMyClass類別中的成員函式如下：
  - 公用等級的成員函式funcA，不接受任何引數，也無回傳值。
  - 公用等級的成員函式funcB，接受兩個整數引數，並回傳一個整數回傳值。
  - 私用等級的成員函式funcC，不接受任何引數，也無回傳值。

```
class CMyClass
{
 public void funA()          //公用成員函式
 {
            .....成員函式定義.....
 }
 public int funcB(int m,int n)  //公用成員函式
 {
            .....成員函式定義.....
 }
 private void funcC()         //私用成員函式
 {
            .....成員函式定義.....
 }
}
```

60

7.5.2透過成員函式存取私用性資料變數

- 私用性(private)資料變數無法被外部程式存取，但可以透過同一類別的成員函式（等級無限制）來存取它，彈性應用私用的資料變數。

- 【觀念範例7-2】：透過公用成員函式存取私用資料與成員函式。
  - **範例7-2：**ch7_02.java（隨書光碟myJava\ch07\ch7_02.java）

61

7.5.2透過成員函式存取私用性資料變數

```
1   /*   檔名:ch7_02.java      功能:透過公用成員函式存取私用資料與成員函式   */
6   public class ch7_02          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          CMyClass Y = new CMyClass();
12
13          X.initVar();        //在X物件上,執行initVar成員函式
14          Y.initVar();        //在Y物件上,執行initVar成員函式
15
16          X.addVar(10);       //在X物件上,執行addVar成員函式
17          System.out.print("物件X\t");
18          X.showVar();
19
20          System.out.print("物件Y\t");
21          Y.addVar(5);        //在Y物件上,執行addVar成員函式
22          Y.showVar();
23
24          System.out.print("物件Y\t");
25          Y.addVar(3);        //在Y物件上,執行addVar成員函式
26          Y.showVar();
27      }
28  }
29
```

執行結果

```
物件X  Var=11
物件Y  Var=6
物件Y  Var=9
```

62

7.5.2透過成員函式存取私用性資料變數

```
30  class CMyClass
31  {
32      public void initVar()
33      {
34          Var = 1;
35      }
36      public void addVar(int b)
37      {
38          Var = Var + b;
39      }
40      public void showVar()
41      {
42          realShow();  //執行private等級的函式
43      }
44      private int Var;
45      private void realShow()
46      {
47          System.out.println("Var=" + Var);
48      }
49  }
```

不可被外部程式存取與執行，但可被同類別的函式存取與執行。

63

7.5.2透過成員函式存取私用性資料變數

- 範例說明：
  - (1)第32~43行，定義了3個公用的成員函式，可以被外部程式執行。
  - 第44~48行，宣告了1個私用的變數及定義了一個私用成員函式，不可以被外部程式存取與執行。
  - (2)變數Var與成員函式realShow雖然不能夠被外部程式存取與執行，但可以被同一類別的成員函式存取與執行，例如第34、38、42行。
  - (3)程式執行時，記憶體內部呈現下列變化。
    - Step1：第10~11行執行完畢，記憶體中將同時存在兩個物件變數X,Y以及它的實體。由於我們並未定義建構子，因此Java會執行預設的建構子（不做任何事），並且所有的成員都會被自動初始化。由於Var是int型態，故為0

64

7.5.2透過成員函式存取私用性資料變數

- （下圖是記憶體的變化，由於尚未介紹static method，故省略main在記憶體資料區的狀況）

記憶體程式區

```
class CMyClass
{
   public void initVar()
   {
      Var = 1;
   }
   public void addVar(int b)
   {
      Var = Var + b;
   }
   public void showVar()
   {
       realShow();
   }
   private int Var;
   private void realShow()
   {
      System.out.println("Var=" + Var);
   }
}
```

記憶體程式區

```
public class ch7_02        //主類別
{
   public static void main(String args[])
   {
      CMyClass X = new CMyClass();
      CMyClass Y = new CMyClass();

      X.initVar();
      Y.initVar();

      X.addVar(10);
      System.out.print("物件X\t");
      X.showVar();

      System.out.print("物件Y\t");
      Y.addVar(5);
      Y.showVar();

      System.out.print("物件Y\t");
      Y.addVar(3);
      Y.showVar();
   }
}
```

記憶體資料區

	:	
	3100	X
	3600	Y
	:	
3100	Var=0	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	
3600	Var=0	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	

65

7.5.2透過成員函式存取私用性資料變數

- Step2：執行第13行，執行完畢時，X的成員變數Var被設定為1。（下圖中，我們省略了部分的記憶體程式區）

記憶體程式區

```
public class ch7_02        //主類別
{
   public static void main(String args[])
   {
      CMyClass X = new CMyClass();
      CMyClass Y = new CMyClass();

      X.initVar();
      Y.initVar();

      X.addVar(10);
      System.out.print("物件X\t");
      X.showVar();

      System.out.print("物件Y\t");
      Y.addVar(5);
      Y.showVar();

      System.out.print("物件Y\t");
      Y.addVar(3);
      Y.showVar();
   }
}
```

記憶體資料區

	:	
	3100	X
	3600	Y
	:	
3100	Var=1	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	
3600	Var=0	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	

66

7.5.2透過成員函式存取私用性資料變數

- Step3：執行第14行，執行完畢時，Y的成員變數Var被設定為1。

記憶體程式區

```
public class ch7_02        //主類別
{
   public static void main(String args[])
   {
      CMyClass X = new CMyClass();
      CMyClass Y = new CMyClass();

      X.initVar();
      Y.initVar();

      X.addVar(10);
      System.out.print("物件X\t");
      X.showVar();

      System.out.print("物件Y\t");
      Y.addVar(5);
      Y.showVar();

      System.out.print("物件Y\t");
      Y.addVar(3);
      Y.showVar();
   }
}
```

記憶體資料區

	:	
	3100	X
	3600	Y
	:	
3100	Var=1	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	
3600	Var=1	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	

67

7.5.2透過成員函式存取私用性資料變數

- Step4：執行第16行，執行完畢時，X的成員變數Var被設定為11。

記憶體程式區

```
public class ch7_02        //主類別
{
   public static void main(String args[])
   {
      CMyClass X = new CMyClass();
      CMyClass Y = new CMyClass();

      X.initVar();
      Y.initVar();

      X.addVar(10);
      System.out.print("物件X\t");
      X.showVar();

      System.out.print("物件Y\t");
      Y.addVar(5);
      Y.showVar();

      System.out.print("物件Y\t");
      Y.addVar(3);
      Y.showVar();
   }
}
```

記憶體資料區

	:	
	3100	X
	3600	Y
	:	
3100	Var=11	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	
3600	Var=1	
	initVar()	
	addVar(int)	
	showVar()	
	realShow()	
	:	

68

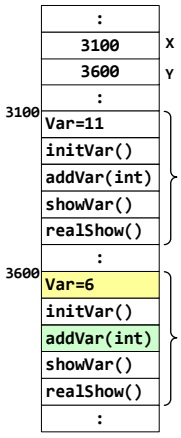
7.5.2透過成員函式存取私用性資料變數

- Step5：執行第17~18行，執行完畢時，印出X的成員變數Var為11。
- Step6：執行第21行，執行完畢時，Y的成員變數Var被設定為6。

記憶體程式區

```
public class ch7_02        //主類別
{
   public static void main(String args[])
   {
      CMyClass X = new CMyClass();
      CMyClass Y = new CMyClass();

      X.initVar();
      Y.initVar();

      X.addVar(10);
      System.out.print("物件X\t");
      X.showVar();

      System.out.print("物件Y\t");
      Y.addVar(5);
      Y.showVar();

      System.out.print("物件Y\t");
      Y.addVar(3);
      Y.showVar();
   }
}
```

記憶體資料區



69

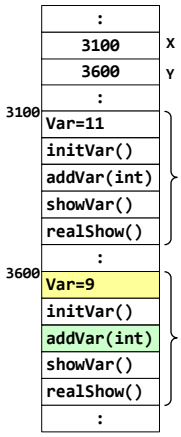
7.5.2透過成員函式存取私用性資料變數

- Step7：執行第22行，執行完畢時，印出Y的成員變數Var為6。
- Step8：執行第25行，執行完畢時，Y的成員變數Var被設定為9。

記憶體程式區

```
public class ch7_02        //主類別
{
   public static void main(String args[])
   {
      CMyClass X = new CMyClass();
      CMyClass Y = new CMyClass();

      X.initVar();
      Y.initVar();

      X.addVar(10);
      System.out.print("物件X\t");
      X.showVar();

      System.out.print("物件Y\t");
      Y.addVar(5);
      Y.showVar();

      System.out.print("物件Y\t");
      Y.addVar(3);
      Y.showVar();
   }
}
```

記憶體資料區



- Step9：執行第26行，執行完畢時，印出Y的成員變數Var為9。

70

7.5.2透過成員函式存取私用性資料變數

- 在範例7-2中，我們必須先呼叫initVar函式將成員變數的初始值設定為1。是否有「更簡便」的方式可以進行這樣的工作呢？
  - 答案是有的
  - 第一種方式是將成員變數於宣告時設定初始值（如範例7-3）
  - 另一種方式則是透過建構子來設定（如範例7-4）。
- 【觀念範例7-3】：改寫範例7-2，將成員變數設定初始值，完成相同的初值設定效果。
  - **範例7-3：**ch7_03.java（隨書光碟myJava\ch07\ch7_03.java）

71

7.5.2透過成員函式存取私用性資料變數

```
1   /*  檔名:ch7_03.java          功能:成員變數設定初始值  */
6   public class ch7_03           //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          CMyClass Y = new CMyClass();
12
13          X.addVar(10);      //在X物件上,執行addVar成員函式
14          System.out.print("物件X\t");
15          X.showVar();
16
17          System.out.print("物件Y\t");
18          Y.addVar(5);       //在Y物件上,執行addVar成員函式
19          Y.showVar();
20
21          System.out.print("物件Y\t");
22          Y.addVar(3);       //在Y物件上,執行addVar成員函式
23          Y.showVar();
24      }
25  }
26
```

72

7.5.2透過成員函式存取私用性資料變數

```
27  class CMyClass
28  {
29      public void addVar(int b)
30      {
31          Var=Var+b;
32      }
33      public void showVar()
34      {
35          realShow();
36      }
37      private int Var=1;          //設定初始值
38      private void realShow()
39      {
40          System.out.println("Var=" + Var);
41      }
42  }
```

當物件實體產生時，物件的Var成員就被設定為1。

- 執行結果：（同範例7-2）
- 範例說明：
  - 第37行，Var成員變數被設定了初始值為1，因此不需要initVar()函式，因為當物件實體產生時，物件的Var成員就被設定為1。

73

7.6 建構子

- 設定成員變數初始值
  - 在範例7-2中，我們必須先呼叫initVar函式將成員變數的初始值設定為1。這個工作可以透過設定成員變數初始值來完成（如範例7-3）
  - ，當我們希望初始值與物件實體產生時有關（而非一個單純的常數）時，就無法單單透過設定成員變數初始值來完成了。

74

7.6 建構子

- 如果在產生物件實體時，能夠自動幫我們呼叫某個特定函式，進行某些初始化動作，不是使得設計程式方便多了嗎？
  - Java可以在建構子中來完成這些事情。
- **建構子(Constructor)又稱建構函式**，它是生成物件實體時會自動執行的函式。由於它很特別，因此定義語法與一般成員函式有些不同，語法如下：

```
[封裝等級] 建構函式名稱(參數串)        //建構函式名稱就是類別名稱
{
        ……建構函式內容……
}
```

75

7.6 建構子

- 語法說明：
  - (1)建構函式名稱一定要和類別名稱相同。
  - (2)不可加上回傳值型態，連void也不行。如果加上回傳值型態（含void）則不是建構函式，如此就不會在物件實體產生時自動執行。
  - (3)沒有任何修飾字。
  - (4)建構函式只能被new運算子於產生物件實體時自動呼叫，無法讓物件的一般函式自由呼叫。
  - (5)封裝等級一般宣告為public等級。如果宣告為private等級，則只有其他建構函式可以呼叫它。無法被new自動呼叫。
- 【觀念範例7-4】：使用建構函式改寫範例7-2，使其成為建構函式以便自動進行成員變數的初始化。
  - 範例7-4：ch7_04.java（隨書光碟 myJava\ch07\ch7_04.java）

76

7.6 建構子

```
1   /*   檔名:ch7_04.java          功能:建構函式    */
6   public class ch7_04           //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass(1);     //自動呼叫建構函式
11          CMyClass Y = new CMyClass(2);     //自動呼叫建構函式
12
13          X.addVar(10);      //在X物件上,執行addVar成員函式
14          System.out.print("物件X\t");
15          X.showVar();
16
17          System.out.print("物件Y\t");
18          Y.addVar(5);       //在Y物件上,執行addVar成員函式
19          Y.showVar();
20
21          System.out.print("物件Y\t");
22          Y.addVar(3);       //在Y物件上,執行addVar成員函式
23          Y.showVar();
24      }
25  }
```

77

7.6 建構子

```
26
27  class CMyClass
28  {
29      public CMyClass(int i)
30      {
31          Var=i;
32      }
:       ...同範例7-2的第36~48行...
46  }
```

建構子名稱需與類別名稱相同。

建構子

- 執行結果：

```
物件X   Var=11
物件Y   Var=7
物件Y   Var=10
```

- **範例說明：**
  - 第10行，宣告物件變數X並產生實體，當實體產生時，就會自動立刻執行該類別的建構函式，也就是執行第29~32行程式。第11行亦同理。如果將第11行傳入的引數改為『1』，則執行結果同範例7-2。

78

7.6.1建構子的多載

- 上一章所提到的多載功能不但可以應用於一般成員函式，也可以應用於建構函式。
  - 程式可藉由定義多個不同參數（或不同資料型態）的建構函式，提供物件實體初始化時更大的彈性。
  - 【觀念範例7-5】：多載建構函式，使得在物件實體產生時，可以依照不同情況執行不同的建構函式。
    - **範例7-5：**ch7_05.java（隨書光碟 myJava\ch07\ch7_05.java）

79

7.6.1建構子的多載

```
1   /*   檔名:ch7_05.java          功能:建構函式的多載   */
6   public class ch7_05           //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          CMyClass Y = new CMyClass(5,40);
12          CMyClass Z = new CMyClass(20.2,30.6);
13          X.showVar();
14          Y.showVar();
15          Z.showVar();
16      }
17  }
18
19  class CMyClass
20  {
21      public double VarA;
22      private double VarB;
23
```

呼叫第24行的建構子

呼叫第29行的建構子

呼叫第34行的建構子

80

7.6.1建構子的多載

```
24      public CMyClass()                   //定義無參數的建構函式
25      {
26          VarA = 10;
27          VarB = 10;
28      }
29      public CMyClass(int a,int b)     //定義兩個整數參數的建構函式
30      {
31          VarA = a;
32          VarB = a + b;
33      }
34      public CMyClass(double a,double b)  //定義兩個浮點數參數的建構函式
35      {
36          VarA = a;
37          VarB = a * b;
38      }
39      public void showVar()
40      {
41          System.out.println("VarA=" + VarA);
42          System.out.println("VarB=" + VarB);
43      }
44  }
```

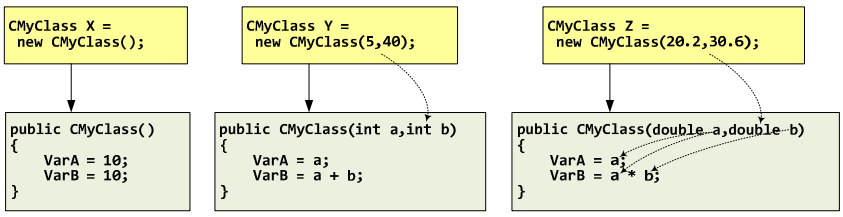
- 執行結果：

```
VarA=10.0
VarB=10.0
VarA=5.0
VarB=45.0
VarA=20.2
VarB=618.12
```

81

7.6.1建構子的多載

- 範例說明：
  - (1)第24~38行是3個建構函式的定義，其署名不相同，符合多載的規定。並且其內的建構內容也可以不同。
  - (2)第10行產成X物件實體，由於沒有引數，所以會執行第24~28行的建構函式。
  - 第11行生成Y物件實體，由於有2個int整數引數，所以會執行第29~33行的建構函式。
  - 第12行生成Z物件實體，由於有2個double浮點數引數，所以會執行第34~38行的建構函式。



82

7.6.2 預設的建構子

- 如果您未定義建構函式，則Java編譯器會自動產生一個內建的預設建構函式(Default Constructor)到程式中，函式內容為空，也沒有參數，格式如下：

```
public 類別名稱()       //預設的建構函式
{
}
```

  - 正因為有此預設的建構函式，所以在之前的範例中，有時我們並未定義建構函式，但編譯仍不會出錯。
  - 但是如果您已經定義了一個以上的建構函式時，Java編譯器便會認定您已經準備好所有的建構函式，此時將不會再產生預設建構函式
  - 而若您沒有建立一個無參數的建構函式，卻又於產生物件實體時不傳入引數，則會發生錯誤，請見以下的範例。

83

7.6.2 預設的建構子

- 【觀念範例7-6】：取消預設建構函式的注意事項
  - **範例7-6：**ch7_06.java（隨書光碟myJava\ch07\ch7_06.java）

```
1   /*   檔名:ch7_06.java       功能:預設的建構子  */
2
3   package myJava.ch07;
4   import java.lang.*;
5
6   public class ch7_06          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass(5,40);    //實體產生有兩個整數引數
11          //CMyClass Y = new CMyClass();
12          X.showVar();
13          //Y.showVar();
14      }
15  }
16
```

執行結果

```
VarA=5
VarB=45
```

無法找到對應的建構子

84

7.6.2 預設的建構子

```
17  class CMyClass
18  {
19      public int VarA;
20      private int VarB;
21
22      public CMyClass(int a,int b) //定義兩個整數參數的建構函式
23      {
24          VarA = a;
25          VarB = a + b;
26      }
27
28      public void showVar()
29      {
30          System.out.println("VarA=" + VarA);
31          System.out.println("VarB=" + VarB);
32      }
33  }
```

範例說明：

- 第11行的註解如果取消，則會發生錯誤，因為預設的建構函式並不會產生，故找不到對應的建構函式可執行。所以在定義建構函式時，應該先定義一個無參數的建構函式（函式內容可以暫時為空），以避免這樣的狀況。

小試身手7-1
依照範例7-6的說明，將第11、13 的註解取消，並加入無參數的建構函式（函式內容為空）， 使得可編譯並執行成功。

85

7.6.3回收物件實體

- 自動回收物件實體
  - 在Java中，想要釋放物件實體，只要將所有參考到該物件實體的物件變數都設為null或指向其他實體，則Java的回收機制就會在一段時間後自動回收該物件實體的記憶體空間。
  - 而其他程式語言（例如C++）則提供了delete與解構函式，需要使用者手動關心物件釋放的後續動作。

86

7.7物件引數的傳參考值呼叫

- 在呼叫函式時，引數也可以是物件
  - 在前面的範例中，我們曾經示範過傳遞字串物件。當時我們曾經提及，傳遞字串物件，事實上採用的是傳「參考值」呼叫
    - 因為物件變數本身是一個參考而非實體。
  - 當我們要傳遞一個自行定義的類別所宣告的物件變數時，同樣採用的是傳「參考值」呼叫，因此，呼叫端與被呼叫端可以共用一個物件實體。
  - 在下面這個範例中，我們將傳遞物件，並且將使用成員函式多載進行不同資料型態的加法。

87

7.7物件引數的傳參考值呼叫

- 【觀念範例7-7】：使用多載設計成員函式，並且接受引數為其他類別的物件。在本範例中，我們將把加法函式擴充到二維向量的加法，範例如右：

$$i = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \quad j = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

$$\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = k = i + j = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

  - **範例7-7：**ch7_07.java（隨書光碟 myJava\ch07\ch7_07.java）

88

7.7物件引數的傳參考值呼叫

```
1   /*  檔名:ch7_07.java       功能:傳遞物件與接收物件  */
2
3   package myJava.ch07;
4   import java.lang.*;
5
6   public class ch7_07          //主類別
7   {
8       public static void main(String args[])
9       {
10          CVector2 i = new CVector2();
11          i.set(20,40);
12          CVector2 j = new CVector2();
13          j.set(15,45);
14          CVector2 k;      //k是CVector2物件變數,在第16行用來接收回傳值
15          CMyClass X = new CMyClass();
16          k = X.sum(i,j);
17          System.out.println("Vector i=(" + i.x + "," + i.y + ")");
18          System.out.println("Vector j=(" + j.x + "," + j.y + ")");
19          System.out.println("Vector k=i+j=(" + k.x + "," + k.y + ")");
20      }
21  }
22
```

執行結果

```
Vector i=(20,40)
Vector j=(15,45)
Vector k=i+j=(35,85)
```

89

```
23  class CMyClass
24  {
25      public int sum(int a,int b)
26      {
27          return a+b;
28      }
29      public double sum(double a,double b)
30      {
31          return a+b;
32      }
33
34      public CVector2 sum(CVector2 a,CVector2 b)//二維向量的加法成員函式
35      {
36          CVector2 tempVector = new CVector2();
37          tempVector.x = a.x + b.x;
38          tempVector.y = a.y + b.y;
39          return tempVector;
40      }
41  }
42
43  class CVector2                    //二維向量類別
44  {
45    public int x,y;                 //二維向量的兩項元素資料
46    public void set(int m,int n)    //用於設定二維向量的兩項元素資料
47    {
48          x = m;
49          y = n;
50    }
51  }
```

90

7.7物件引數的傳參考值呼叫

- 範例說明：
  - (1)我們在第43~51行，宣告了另一個類別CVector2，用來代表二維向量。並提供一個set成員函式在第46~50行。
  - (2)除了整數與浮點數的加法，在第34~40行，我們又增加了CMyClass的另一個成員函式sum，它接受的引數是兩個CVector2類別的物件，並且回傳一個CVector2類別的物件，它會做二維向量的加法，並回傳二維向量。
  - (3)第10~11行，宣告一個CVector2類別的物件i並產生實體，且設定成員變數為(20,40)。
  - (4)第12~13行，宣告一個CVector2類別的物件j並產生實體，且設定成員變數為(15,45)。
  - (5)第14行，宣告一個CVector2類別的物件變數k。
  - (6)第15行，宣告一個CMyClass類別的物件X並產生實體。

91

7.7物件引數的傳參考值呼叫

- (7)第16行，透過X物件的sum成員函式，幫我們做二維向量的加法，所以k向量=(35,85)。
- (8)在第16行的函式呼叫，i物件的參考傳送給對應的a物件參考，故i與a實際上指向同一個物件實體。j與b亦同理。
- (9)在第14行時，k是一個物件參考，初始值為null，而第16行時，由於第39行會回傳一個物件實體位址，所以k儲存該位址後，可以指向物件的實體。

- 在上一個範例中，雖然i與a指向同一個物件，j與b指向同一個物件，但依據的是傳「參考值」呼叫。
  - 對於Java的傳「參考值」呼叫，每一年都會引起論壇的討論，爭辯傳參考呼叫與傳參考值呼叫的差別。

92

7.7物件引數的傳參考值呼叫

- 傳參考呼叫(Pass by reference)一般而言，指的是程式語言功能中的傳址呼叫(Call by address)
  - 其特色是被呼叫端的參數會影響呼叫端的引數，如C++與C#都提供此一功能。
- 傳值呼叫(Pass by value)則是程式語言功能中的傳值呼叫(Call by value)
  - 其特色是被呼叫端的參數不會影響呼叫端的引數，大多數的程式語言都提供了此一功能，如Java。
- 由於傳址呼叫所能達到的效果，在被呼叫端需要影響呼叫端兩個以上的變數時非常有用（因為函式回傳值只能傳遞一個值），因此程式語言必須提供傳址呼叫
  - 若未提供傳址呼叫，則必須提供其他機制來達到同樣的效果
  - 而所謂其他機制，則不可避免地仍需要傳遞一個記憶體位址才可能達到同樣的效果
    - 例如C語言就是透過傳指標呼叫(Pass by pointer)來達成此一效果。

93

7.7物件引數的傳參考值呼叫

- Java的傳「參考值」呼叫則與C語言的傳指標呼叫(Pass by pointer)類似
  - 它雖然也是透過引數傳遞一個位址給被呼叫端的參數，但即便參數所保存的位址值（參考值）改變了，也不會影響引數所保存的位址值（參考值）。
  - 換句話說，傳「參考值」呼叫仍屬於傳值呼叫(Call by value)的一種，只不過這個值是一個參考罷了。
  - 所以傳參考呼叫與傳參考值呼叫是不同的兩種機制。
    - 因此，在眾多的Java原文技術文件中，都會提及，Java只支援傳值呼叫，這是因為傳「參考值」呼叫仍屬於傳值呼叫(Call by value)，而不是傳址呼叫(Call by address)。
- 以上，在上一章介紹Java傳遞陣列時，我們已經說明過了，下面這個範例，將會證明Java在傳遞物件或陣列時，採用的只是傳「參考值」呼叫，而非傳參考呼叫。

94

7.7物件引數的傳參考值呼叫

- 假設Java提供了傳參考呼叫（傳址呼叫），我們可以很容易的製作一個swap(i, j)函式，用以將i, j互相對調。
  - 以下，我們嘗試製作一個swap函式將兩個引數(物件參考i,物件參考j)對調看看。
  - 【觀念範例7-8】：嘗試設計一個swap函式互換兩個物件參考。
    - **範例7-8：**ch7_08. java（隨書光碟myJava\ch07\ch7_08. java）

95

7.7物件引數的傳參考值呼叫

```
1   /*   檔名:ch7_08.java      功能:物件的傳參考值呼叫    */
2
3   package myJava.ch07;
4   import java.lang.*;
5
6   public class ch7_08          //主類別
7   {
8       public static void main(String args[])
9       {
10          CVector2 i = new CVector2();
11          i.set(20,40);
12          CVector2 j = new CVector2();
13          j.set(15,45);
14          System.out.println("-----原始------");
15          System.out.println("Vector i=(" + i.x + "," + i.y + ")");
16          System.out.println("Vector j=(" + j.x + "," + j.y + ")");
17          CMyClass X = new CMyClass();
18          X.swap(i,j);
19          System.out.println("-----swap後-------");
20          System.out.println("Vector i=(" + i.x + "," + i.y + ")");
21          System.out.println("Vector j=(" + j.x + "," + j.y + ")");
22      }
23  }
24
```

執行結果

```
-----原始------
Vector i=(20,40)
Vector j=(15,45)
-----swap後-------
Vector i=(20,40)
Vector j=(15,45)
```

96

7.7物件引數的傳參考值呼叫

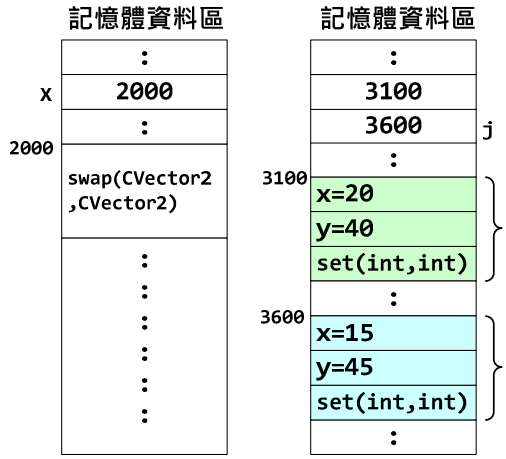
```
25  class CMyClass
26  {
27      public void swap(CVector2 a,CVector2 b)  //互換函式
28      {
29          CVector2 temp;
30          temp = a;
31          a = b;
32          b = temp;
33      }
34  }
35
36  class CVector2                          //二維向量類別
37  {
38    public int x,y;                  //二維向量的兩項元素資料
39    public void set(int m,int n)  //用於設定二維向量的兩項元素資料
40    {
41          x = m;
42          y = n;
43    }
44  }
```

- 範例說明：
  - (1)從執行結果中，我們發現對調失敗了，為什麼會這樣呢？我們以記憶體的變化來說明。

97

7.7物件引數的傳參考值呼叫

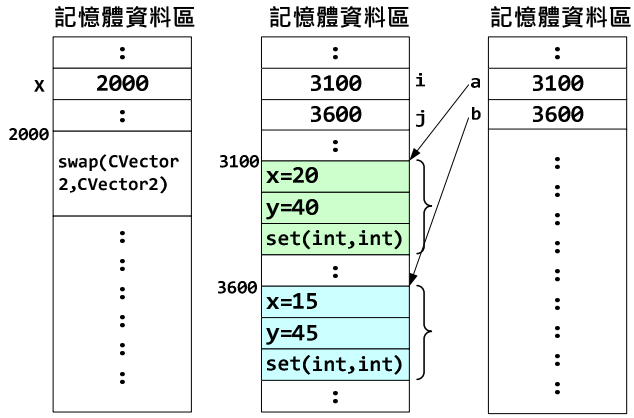
- (2)當第10~17行執行完畢時，記憶體內容呈現下圖狀況，i指向3100的物件實體，j指向3600的物件實體。



98

7.7物件引數的傳參考值呼叫

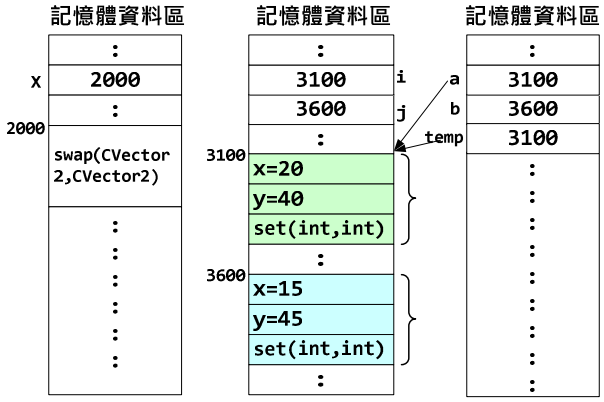
- (3)第18行的X.swap(i, j);會把i與j的參考傳送給a與b，如下圖。然後開始執行第28~33行的程式。



99

7.7物件引數的傳參考值呼叫

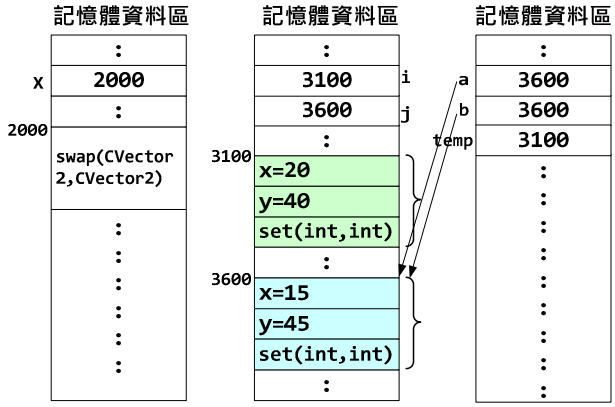
- (4)第29行會配置一個可存放參考的記憶體給temp，第30行則會讓a的參考設定給temp，使得兩者都指向同一個物件。第30行執行完畢時，結果如下圖：



100

7.7物件引數的傳參考值呼叫

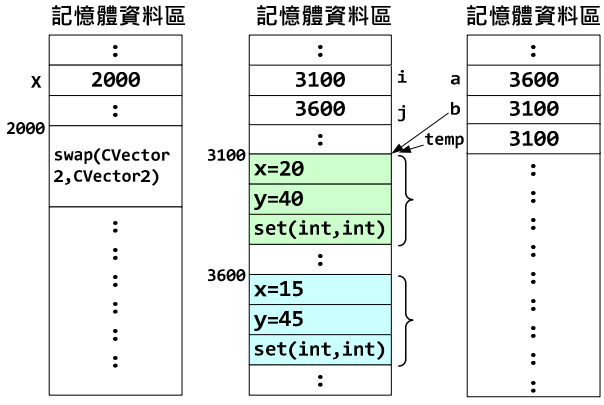
- (5)第31行則會讓b的參考設定給a，使得兩者都指向同一個物件。第31行執行完畢時，結果如下圖：



101

7.7物件引數的傳參考值呼叫

- (6)第32行則會讓temp的參考設定給b，使得兩者都指向同一個物件。第32行執行完畢時，結果如下圖：（a與b的參考已經互換完成）



102

7.7物件引數的傳參考值呼叫

- (7)第33行執行完畢時會返回函式呼叫處，此時，由於參數a,b及函式內的temp都屬於區域變數，故會被釋放。故a與b的互換根本沒有意義，因為i與j並沒有互換，記憶體內容和執行函式呼叫前是完全相同的。所以印出的結果自然是沒有互換成功的結果。

- (8)雖然我們互換失敗，但確實在swap中，若透過a或b或temp修改物件實體的變數內容的話，則呼叫返回後，i或j的物件實體內容也會被更動。但這並非傳統所謂的傳參考呼叫(pass by reference)，頂多只能稱的上是傳參考值呼叫(pass by value of reference)。

- (9)如果您熟悉C++語言的話，就會知道，C++語言支援傳參考呼叫，一個類似功能的程式碼如下，請注意其結果，i與j的內容是被更動的。

103

7.7物件引數的傳參考值呼叫

```
1   /*   檔名:ch7_08.cpp   功能:傳參考呼叫   */
2
3   #include <iostream>
4   #include <stdlib.h>
5   using namespace std;
6
7   class CVector2        //二維向量類別
8   {
9     public: int x,y;    //二維向量的兩項元素資料
10    public: void set(int m,int n)  //用於設定二
11    {
12          x = m;
13          y = n;
14    }
15  };
16
17  class CMyClass
18  {
19      public: void swap(CVector2 &a,CVector2 &b)  //互換函式,傳參考呼叫
20      {
21          CVector2 temp;
22          temp = a;
23          a = b;
24          b = temp;
25      }
26  };
```

執行結果

```
-----原始-----
Vector i=(20,40)
Vector j=(15,45)
-----swap後-------
Vector i=(15,45)
Vector j=(20,40)
```

104

7.7物件引數的傳參考值呼叫

```
27
28  int main(char args[])
29  {
30     CVector2 i;
31     i.set(20,40);
32     CVector2 j;
33     j.set(15,45);
34     cout << "-----原始------" << endl;       //列印
35     cout << "Vector i=(" << i.x << "," << i.y << ")" << endl;
36     cout << "Vector j=(" << j.x << "," << j.y << ")" << endl;
37     CMyClass X;
38     X.swap(i,j);   //互換i,j
39     cout << "-----swap後-------" << endl;   //列印
40     cout << "Vector i=(" << i.x << "," << i.y << ")" << endl;
41     cout << "Vector j=(" << j.x << "," << j.y << ")" << endl;
42     system("pause");
43     return 0;
44  }
```

- (10)不只C++支援傳參考呼叫，C#同樣也支援傳參考呼叫，上述範例改寫為C#版本的程式碼如下
  - 請注意其結果，i與j的內容是被更動的。

105

7.7物件引數的傳參考值呼叫

```
1   //檔名:ch7_08.cs    功能:傳參考呼叫
2
3   using System;
4   using System.Collections.Generic;
5   using System.Linq;
6   using System.Text;
7
8   namespace ConsoleApplication1
9   {
10      class Program
11      {
12          static void Main(string[] args)
13          {
14           CVector2 i = new CVector2();
15           i.set(20,40);
16           CVector2 j = new CVector2();
17           j.set(15,45);
18           Console.WriteLine("-----原始------");
19           Console.WriteLine("Vector i=(" + i.x + "," + i.y + ")");
20           Console.WriteLine("Vector j=(" + j.x + "," + j.y + ")");
21           CMyClass X = new CMyClass();
22           X.swap(ref i, ref j);   //傳參考呼叫
23           Console.WriteLine("-----swap後-------");
24           Console.WriteLine("Vector i=(" + i.x + "," + i.y + ")");
25           Console.WriteLine("Vector j=(" + j.x + "," + j.y + ")");
26           Console.Read();
27          }
28      }
```

執行結果

```
-----原始------
Vector i=(20,40)
Vector j=(15,45)
-----swap後-------
Vector i=(15,45)
Vector j=(20,40)
```

106

7.7物件引數的傳參考值呼叫

```
29      class CMyClass
30      {
31          public void swap(ref CVector2 a, ref CVector2 b) //傳參考呼叫
32          {
33              CVector2 temp;
34              temp = a;
35              a = b;
36              b = temp;
37          }
38      }
39
40      class CVector2                          //二維向量類別
41      {
42          public int x, y;                    //二維向量的兩項元素資料
43          public void set(int m, int n)  //用於設定二維向量的兩項元素資料
44          {
45              x = m;
46              y = n;
47          }
48      }
49  }
```

107

7.8 this關鍵字

- this是一個特殊的關鍵字，在撰寫程式時代表類別的本身，在實際執行時，則代表物件的本身。
- 不過我們先思考另外一個問題，即『當成員變數與區域變數同名』的問題。

108

7.8.1 區域變數與成員變數同名

- 如果是類別內的敘述，則使用類別內的成員時，不用加上物件名稱（例如範例7-8的第41、42行）。
- 如果在函式中定義了與成員變數同名的區域變數，此時，取用該名稱的變數會是區域變數還是成員變數呢？
  - 由於區域變數的宣告較靠近存取敘述，因此取用的將是區域變數，請見下列範例。
-
  - 【觀念範例7-9】：區域變數與成員變數同名的探討
    - **範例7-9：**ch7_09.java（隨書光碟 myJava\ch07\ch7_09.java）

7.8.1 區域變數與成員變數同名

```
1   /*   檔名:ch7_09.java          功能:區域變數與成員變數同名   */
6   public class ch7_09          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          X.show();
12      }
13  }
14
15  class CMyClass
16  {
17      public int var=10;     //成員變數
18      public void show()
19      {
20          int var=20;                          //區域變數
21          System.out.println("var=" + var);
22      }
23  }
```

執行結果

```
var=20
```

將會取用到區域變數var

7.8.1 區域變數與成員變數同名

- **範例說明：**
  - 從執行結果中，我們發現取用到的是區域變數var，而非成員變數var
  - 那麼我們如果在show()函式內要取用成員變數var該怎麼辦呢？此時必須搭配this關鍵字來取用。

111

7.8.2 this關鍵字的簡易使用法

- this關鍵字是一個非常特別的關鍵字
  - 對於任何類別而言，this就是代表著該類別本身，當類別產生物件實體後，則物件的this代表物件實體本身
    - 因此，如果要使得範例7-9能夠取用到成員變數，可以透過this關鍵字來完成。
  - 【觀念範例7-10】：改寫範例7-9，使得區域變數與成員變數都能夠被印出。
    - **範例7-10：**ch7_10.java（隨書光碟myJava\ch07\ch7_10.java）

112

7.8.2 this關鍵字的簡易使用法

```
1   /*  檔名:ch7_10.java              功能:this關鍵字的簡易使用法  */
6   public class ch7_10           //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          X.show();
12      }
13  }
14
15  class CMyClass
16  {
17      public int var=10;     //成員變數
18      public void show()
19      {
20          int var=20;                        //區域變數
21          System.out.println("區域變數var=" + var);
22          System.out.println("成員變數var=" + this.var);
23      }
24  }
```

執行結果

```
區域變數var=20
成員變數var=10
```

將會取用物件的var成員

7.8.2 this關鍵字的簡易使用法

- 範例說明：
  - 在這個範例中，我們發現如果區域變數與成員變數不同名的話，使用與不使用this關鍵字並沒有差別。
  - 那麼this關鍵字的功用是否僅止於此呢？事實上並不是的，不過我們應該先深入探討this的真正涵意以及實作上的變化，然後再來討論this的用途。

7.8.3 深入探討this關鍵字

- this可於實作時期（程式執行時），當作物件本身來使用。
  - 因此上述範例的this，代表的就是物件X。如果在其他類別中（例如主類別main函式中），我們要取用物件X的var變數值，可透過「X.var」來達成，而在X的方法中，則可以透過「this.var」來達成，因為this就是X。
  - 而this是X的什麼呢？
    - 從「X.var」中的X可以得知，X在Java中扮演的是一個物件的參考，故this實際上是代表物件本身的參考。
    - 假設X的實體位址為3100，則this也是3100，所以對於JVM而言，不論遇到「X.var」或「this.var」，它看到的都是存取位於位址3100之物件實體中的成員變數var。

115

7.8.3 深入探討this關鍵字

- 不論我們在程式中是否出現this關鍵字，Java編譯器在編譯類別時，都會自動加上隱含的參考變數this，並且this並不算是類別的成員，而且只能在類別中使用。
- this變數值設定的時機，是在遇到「new 類別名稱()」時，由於new代表產生類別的物件實體，故會有一個實體的開始位址，在Java中，我們將該位址稱為物件的參考(reference)，而當new指令執行完畢時，this變數就被設定為該物件的參考。
  - 而在範例7-10中，第10行執行完畢時，不僅X被設定為物件的參考，this也被設定為物件的參考。

116

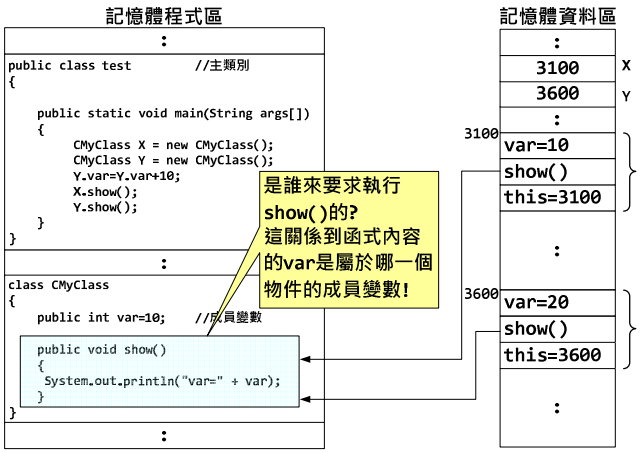
7.8.3 深入探討this關鍵字

- 事實上，**this()代表的是呼叫該類別的建構函式，不過這樣的語法只能出現在建構函式的第一行**
  - 因此，當建構函式想要呼叫另一個建構函式以進行初始化的分工時，就可以藉由this()來完成。
- this既然是編譯時期會加入的隱含變數，故在產生物件實體時，它也是物件實體的一部分。
  - 在之前的圖示中，我們有時候會看到物件實體內包含了成員函式，但它未包含完整的成員函式內容，事實上，它的作用只是用來呼叫程式段的成員函式實作。

117

7.8.3 深入探討this關鍵字

- 由於每一個物件都可能去執行程式段的成員函式，故對於程式段的成員函式而言，它必須知道是哪一個物件要求執行它，以便於當它要取用成員變數時，知道要取用哪一個物件的成員變數。如下圖示意：



118

7.8.3 深入探討this關鍵字

- 程式段的成員函式需要知道是哪一個物件呼叫它，故在編譯時期，編譯器會對所有的成員函式都加上一個隱含的final參數this，參數的型別是類別本身
- 因此，this參數是一個參考，而該this參數將會在函式呼叫時，被設定為引數物件的參考值，故而呼叫時，會變成是將物件本身當作引數傳遞過去，使得在呼叫成員函式時，函式得以判斷是哪一個物件呼叫它，並且取用到正確的成員變數。
- 由於this是被宣告為final參數，故只有在呼叫時能被設定，之後就不能再被設定了。
- 因此上圖應該修改為下圖才是完整的圖示。

119

7.8.3 深入探討this關鍵字

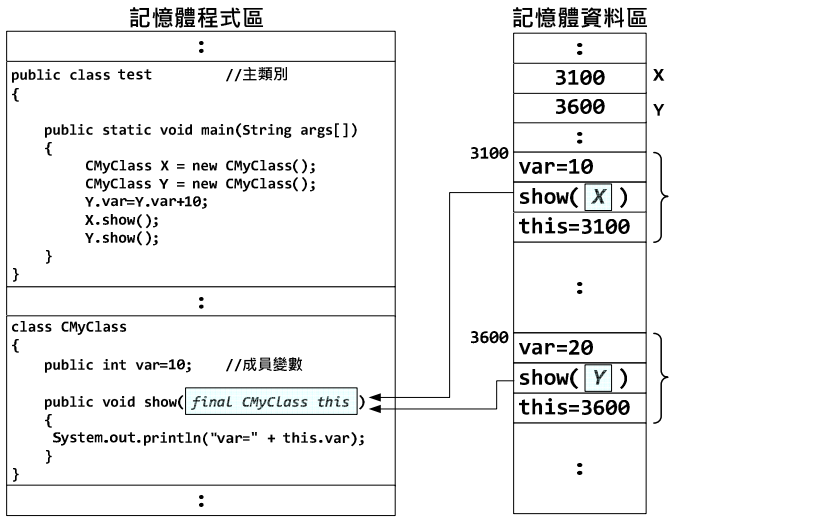


圖7-11 this的作用

120

7.8.3 深入探討this關鍵字

- 在上圖中，如果是X的show函式要求執行程式段的show函式，則會傳入X的參考，也就是3100，因此，程式段的show函式在取用this.var變數時，知道要取用處於位址3100之物件實體的成員變數var
- 我們透過範例來觀察，this的內容究竟為何？
- 【觀念範例7-11】：觀察this變數內容，以求得物件實體的所在位址。
  - **範例7-11：**ch7_11.java（隨書光碟myJava\ch07\ch7_11.java）

121

7.8.3 深入探討this關鍵字

```
1   /*   檔名:ch7_11.java        功能:this的內容  */
6   public class ch7_11         //主類別
7   {
8
9       public static void main(String args[])
10      {
11          CMyClass X = new CMyClass();
12          CMyClass Y = new CMyClass();
13          System.out.print("X物件實體所在位址=");
14          X.show_this();
15          System.out.print("Y物件實體所在位址=");
16          Y.show_this();
17      }
18  }
19
20  class CMyClass
21  {
22      public int var=10;    //成員變數
23      public void show_this()
24      {
25          System.out.println(this);
26      }
27  }
```

執行時，this代表物件的實體位址

122

7.8.3 深入探討this關鍵字

- 執行結果：

```
X物件實體所在位址=myJava.ch07.CMyClass@5fa6fb3e
Y物件實體所在位址=myJava.ch07.CMyClass@4bb8d481
```

- 範例說明：
  - 從執行結果中，我們可以得知
    - X物件的實體所在位址為「5fa6fb3e」
    - Y物件的實體所在位址為「4bb8d481」。

123

7.8.4 this關鍵字的使用時機

- 總結上述的介紹，我們可以將this關鍵字的使用時機整理如下：
  - 1. 取得物件實體的參考：如同範例7-11。
  - 2. 區別成員變數與區域變數：如同範例7-10。
  - 3. 用以作為物件本身，例如進行物件的比較：請見範例7-13。
  - 4. 在建構函式內，藉以呼叫其他的建構函式：使用的是this(引數串)格式，語法如下，請見範例7-12。

124

7.8.4 this關鍵字的使用時機

- 使用this(引數串)呼叫建構函式語法：

```
[封裝等級] 建構函式名稱(參數串)       //建構函式名稱就是類別名稱
{
    this(引數串);    //必須是建構函式的第一個敘述
        ……建構函式的其餘內容……
}
```

125

7.8.4 this關鍵字的使用時機

- 【觀念及實用範例7-12】：藉由this(引數串)呼叫其他的建構函式。
  - **範例7-12：**ch7_12.java（隨書光碟 myJava\ch07\ch7_12.java）

```
1   /*  檔名:ch7_12.java      功能:在建構函式中使用this(引數列)    */
2
3   package myJava.ch07;
4   import java.lang.*;
5
6   public class ch7_12          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass(3);    //實體產生時呼叫第22行的建構子
11          CMyClass Y = new CMyClass(5,40);//實體產生時呼叫第27行的建構子
12          X.showVar();
13          Y.showVar();
14      }
15  }
16
17  class CMyClass
18  {
19      public int VarA;
20      private int VarB;
```

執行結果

```
VarA=1
VarB=3
VarA=200
VarB=40
```

126

7.8.4 this關鍵字的使用時機

```
21      public CMyClass(){}          //良好的習慣是補上一個沒有參數的建構函式
22      public CMyClass(int i)               //定義一個整數參數的建構函式
23      {
24          VarA = 1;
25          VarB = i;
26      }
27      public CMyClass(int a,int b)     //定義兩個整數參數的建構函式
28      {
29          this(b);              //呼叫第22行的建構函式,藉以設定VarB
30          VarA = a * b;
31      }
32      public void showVar()
33      {
34          System.out.println("VarA=" + VarA);
35          System.out.println("VarB=" + VarB);
36      }
37  }
```

- **範例說明：**
  - 在第29行，將設定VarB的工作交給第22行的建構函式完成，雖然它也會設定VarA，但第30行會重新計算VarA的值。
  - **使用this(引數串)呼叫建構函式，必須在建構函式中才能使用，並且必須是第一個敘述。**

127

7.8.4 this關鍵字的使用時機

- 【觀念及實用範例7-13】：兩物件的比較。
  - **範例7-13：**ch7_13.java（隨書光碟myJava\ch07\ch7_13.java）

```
1   /*   檔名:ch7_13.java         功能:物件的比較   */
2
3   package myJava.ch07;
4   import java.lang.*;
5
6   public class ch7_13          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass(5);
11          CMyClass Y = new CMyClass(5);
12          CMyClass Z = X;
13          System.out.print("物件X與物件Y  ");
14          X.compare2Obj(Y);
15          System.out.print("物件X與物件Z  ");
16          X.compare2Obj(Z);
17      }
18  }
19
```

128

7.8.4 this關鍵字的使用時機

```
20  class CMyClass
21  {
22      private int Var;
23      public CMyClass(){}        //良好的習慣是補上一個沒有參數的建構函式
24      public CMyClass(int i)     //一個整數參數的建構函式
25      {
26          Var=i;
27      }
28
29      public void compare2Obj(CMyClass Obj)
30      {
31          if(this==Obj)
32             System.out.println("兩物件相等");
33          else
34             System.out.println("兩物件不相等");
35      }
36  }
```

- 執行結果：

```
物件X與物件Y  兩物件不相等
物件X與物件Z  兩物件相等
```

129

7.8.4 this關鍵字的使用時機

- 範例說明：
  - (1)在第31行進行兩物件的比較，其中this將會是物件本身的參考，例如第14、16行，由於是由X執行該函式，故this代表X。
  - 而Obj則是接收傳入的引數，例如第14行的呼叫，則Obj代表Y，第16行的呼叫，則Obj代表Z。
  - (2)從執行結果中，我們可以發現物件的比較事實上是參考的比較，而非實體內容的比較。
  - 因為不論是物件名稱或this，代表的都是一個參考，而相等比較『==』代表的是比較兩個參考值是否相同。所以如果兩個參考值不同（指向不同的實體），即使所指向的實體「內容」完全相同，也不會被判斷為相等。

130

7.9 物件陣列

- 物件也可以存放於陣列中，只不過存放的將會是物件的參考
  - 要使用物件陣列，至少需要使用兩次以上的new運算子。語法如下：

```
類別名稱 陣列名稱[] = new 類別名稱[物件數量];
for(int i=0;i<陣列名稱.length;i++)
{
  陣列名稱[i]=new 類別名稱(引數串列);  //為每一個陣列元素產生物件實體
}
```

  - 【語法說明】
    - (1)第一行是宣告一維物件陣列，並產生陣列實體。同樣的，它可以分解為兩行如下

```
類別名稱 陣列名稱[];                          //宣告陣列名稱
陣列名稱 = new 類別名稱[物件數量];    //產生陣列實體
```

131

7.9 物件陣列

- (2)由於每一個陣列元素代表每一個物件的參考，所以我們可以透過迴圈逐次產生各個物件實體。如果您想要呼叫的建構函式不同，則可以將迴圈展開，手動逐個產生各個物件實體，如下範例語法：

```
陣列名稱[0] = new 類別名稱(引數串列);
陣列名稱[1] = new 類別名稱(引數串列);
陣列名稱[2] = new 類別名稱(引數串列);
      :
      :
      :
```

132

7.9 物件陣列

- 【範例1】宣告一個包含10個字串物件的陣列strArr，初始內容為空。

```
String strArr[] = new String[10];
for(int i=0;i<strArr.length;i++)
{
  strArr[i] = new String();    //為每一個陣列元素產生物件實體
}
```

- 【說明】strArr包含10個陣列元素，每一個陣列元素都是字串物件的參考，在迴圈中，我們則為這些物件參考產生物件實體。

133

7.9 物件陣列

- 【範例2】宣告一個包含7個字串物件的陣列strArr，初始內容為各星期的英文單字。

```
String strArr[] = new String[7];
strArr[0] = new String("Sunday");       //為陣列元素strArr[0]產生物件實體
strArr[1] = new String("Monday");       //為陣列元素strArr[1]產生物件實體
strArr[2] = new String("Tuesday");      //為陣列元素strArr[2]產生物件實體
strArr[3] = new String("Wendesday");    //為陣列元素strArr[3]產生物件實體
strArr[4] = new String("Thursday");     //為陣列元素strArr[4]產生物件實體
strArr[5] = new String("Friday");       //為陣列元素strArr[5]產生物件實體
strArr[6] = new String("Saturday");     //為陣列元素strArr[6]產生物件實體
```

- 【說明】由於要藉由建構函式建置不同的字串內容，故將迴圈展開手動來設定引數。

134

7.9 物件陣列

- 【觀念範例7-14】：建立一個物件陣列，並透過引數設定，觀察建構函式的執行狀況。
  - **範例7-14：**ch7_14.java（隨書光碟 myJava\ch07\ch7_14.java）

```
1   /*   檔名:ch7_14.java          功能:物件陣列與建構函式   */
6   public class ch7_14          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X[] = new CMyClass[3];
11          System.out.println("------------------");
12          for(int i=0;i<X.length;i++)
13              X[i] = new CMyClass();
14
15          CMyClass Y[] = new CMyClass[3];
16          System.out.println("------------------");
17          for(int i=0;i<Y.length;i++)
18              Y[i] = new CMyClass(i+10);
19      }
20  }
```


7.9 物件陣列

```
21
22  class CMyClass
23  {
24      public int VarA;
25      private int VarB;
26      public CMyClass()       //無參數的建構函式
27      {
28          System.out.println("無參數的建構函式執行中");
29      }
30      public CMyClass(int i)
31      {
32          System.out.println("有參數的建構函式執行中,參數為" + i);
33      }
34  }
```

- 執行結果：

```
------------------
無參數的建構函式執行中
無參數的建構函式執行中
無參數的建構函式執行中
------------------
有參數的建構函式執行中,參數為10
有參數的建構函式執行中,參數為11
有參數的建構函式執行中,參數為12
```

7.9 物件陣列

- 範例說明：
  - 利用第11、16行的列印，我們可以得知，物件的建構函式會等到實際產生物件實體時才進行，而不會在產生物件陣列實體時進行，因為物件陣列內的元素只是物件的參考而已。

- 傳遞物件陣列
  - 我們時常會有需要將物件陣列傳遞給某個函式，進行更進一步的應用。
  - 傳遞物件陣列和傳遞一般陣列的語法差不多，只不過是將原始資料型態改為類別名稱而已，請見下列範例。

137

7.9 物件陣列

- 【觀念範例7-15】：觀察物件陣列中，每個元素的參考值。藉以了解物件實體在記憶體的配置情形。
  - **範例7-15：**ch7_15.java（隨書光碟myJava\ch07\ch7_15.java）

```
1   /*   檔名:ch7_15.java          功能:傳遞物件陣列與物件配置情形    */
6   public class ch7_15          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X[] = new CMyClass[5];
11          for(int i=0;i<X.length;i++)
12              X[i] = new CMyClass();
13          COtherClass Obj = new COtherClass();
14          Obj.show(X);
15      }
16  }
17
```

138

7.9 物件陣列

```
18  class COtherClass
19  {
20      public void show(CMyClass X[])
21      {
22          for(int i=0;i<X.length;i++)
23              System.out.println("物件X[" + i + "]實體的位址在" + X[i]);
24      }
25  }
26
27  class CMyClass
28  {
29      public int VarA;
30      private int VarB;
31  }
```

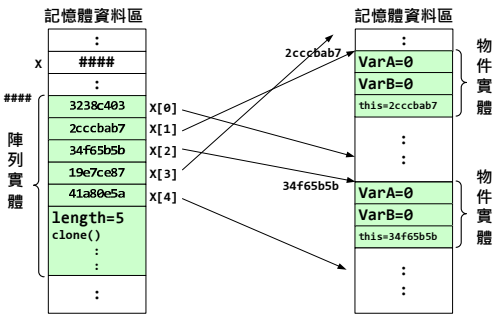
- 執行結果：

```
物件X[0]實體的位址在myJava.ch07.CMyClass@3238c403
物件X[1]實體的位址在myJava.ch07.CMyClass@2cccbab7
物件X[2]實體的位址在myJava.ch07.CMyClass@34f65b5b
物件X[3]實體的位址在myJava.ch07.CMyClass@19e7ce87
物件X[4]實體的位址在myJava.ch07.CMyClass@41a80e5a
```

139

7.9 物件陣列

- 範例說明：
  - (1)第14行Obj物件呼叫show函式時，將物件陣列當作引數傳入函式中。
  - (2)在show函式中，我們列印X[i]的內容，會列印出每個物件的參考位址，您可以發現，雖然陣列元素X[0]~X[5]會位於連續的記憶體位址，但實際的物件實體並不一定會位於連續的記憶體位址。參考圖示如下：



140

7.10 類別變數與類別方法

- 回顧定義類別的語法，我們可以發現，在宣告成員變數或成員函式時，可以指定其[修飾字]。
  - 其中，對於成員變數而言，我們可以將修飾字指定為static, final, transient, volatile等等，或其組合，不過final與volatile不能合用。
  - 對於成員函式而言，我們可以將修飾字指定為static, final, abstract, native, synchronized等等，或其組合，但abstract則不能與其他修飾字合用。
- 在本節中，我們要介紹的只有static修飾字
  - 使用static修飾字定義的成員變數稱為**類別變數(class variable)**
  - 使用static修飾字定義的成員函式稱為**類別方法(class method)**。因此，我們可以將語法修改如下：

141

7.10 類別變數與類別方法

- **定義「類別變數」與「類別方法」語法**

```
[封裝等級] [修飾字]  class  類別名稱  [extends 父類別] [implements介面]
{
   [封裝等級]  static  [其他修飾字]  資料型態  field名稱;        //宣告類別變數
                  :
                  :
   [封裝等級]  static  [其他修飾字]  回傳值型態  method名稱(參數串)  //定義類別方法
   {
         method的內容
   }
                  :
                  :
}
```

  - 【說明】：
    - (1)由於static仍可以和其他的修飾字合用，故我們保留了[其他修飾字]語法欄位。不過，如果使用在成員函式宣告，則static不可與abstract合用。
    - (2)在類別方法內，不可以存取實體變數。

142

7.10 類別變數與類別方法

- 變數的種類
  - 本節要深入探討的是實體變數與類別變數的區別
    - (1)實體變數(Instance Variables)：見7.10.1節。
    - (2)類別變數(Class Variables)：見7.10.2節。
    - (3)區域變數(Local Variables)：在函式內宣告的變數，生命週期侷限於函式執行時，視野也侷限於函式內。
    - (4)參數(Parameters)：又稱為參數變數，函式定義第一行所宣告的變數。用於呼叫函式時，接收呼叫者傳遞之資料。

143

7.10.1 實體變數與實體方法

- 我們在類別內宣告成員變數與成員函式時，如果沒有加上static修飾字，則可以將這些成員變數與成員函式稱為**實體變數(Instance Variables)**與**實體方法(Instance Methods)**。
  - 實體變數與實體方法代表一定要有實體，這些變數與方法才可使用。
  - 當我們宣告一個物件變數並透過new產生實體後（或透過設定運算子指向某一個實體後），就可以在外部程式透過「物件名稱.成員變數」或「物件名稱.成員函式()」來存取呼叫public等級的成員變數及成員函式。

144

7.10.1 實體變數與實體方法

- 「必須透過物件名稱才能夠完成存取呼叫」是實體變數與實體方法的特色。
  - 當每一個物件實體使用new產生時，在記憶體中，針對每一個物件實體，會配置實體變數所需要的記憶體空間
  - 實體方法也會配置一個記憶體空間，記載著程式段的函式名稱與引數，並且this也會自動變成參數之一。
  - 換句話說，如果沒有使用new產生物件實體的話，則在記憶體中，並不存在實體變數與實體方法。
  - 例如下列範例發生錯誤的原因，是因為X並沒有物件實體，所以無法存取實體變數，也無法執行實體方法。如果將CMyClass X;修改為CMyClass X=new CMyClass();則語法就沒有錯誤。

145

7.10.1 實體變數與實體方法

- 並非每一個變數與方法都必須藉由物件才能執行。
  - 例如在表3-7所介紹的toString()，就不需要透過物件即可執行，因為它屬於類別方法。

```
public class test       //主類別
{
   public static void main(String args[])
   {
      CMyClass X;
      X.Var = 3;       //錯誤，因為X的參考為null
      X.show();        //錯誤，X並未參考到實體
   }
}

class CMyClass
{
   public int Var;
   public CMyClass() //定義無參數的建構函式
   {
   }
   public void show()
   {
      System.out.println("Hello");
   }
}
```

146

7.10.2類別變數與類別方法

- 雖然Java是一個物件導向語言，但實質上，它是以類別為基礎(class-based)來撰寫程式
  - 在Java中，所有的變數與方法都必須隸屬於某一個類別，而無法像C++定義出不屬於任何類別的變數與函式。
  - 至於Java的物件，只是類別生成出的一個實體，它仍規範在類別的基礎之上。
- Java允許我們在不產生物件的狀況下，使用變數與方法，但僅侷限於類別變數與類別方法。
  - **類別變數(Class Variables)**與**類別方法(Class Methods)**合稱類別成員(Class Members)
  - 在宣告類別成員時，必須將修飾字宣告為**static**，如同前面的語法格式，而使用時，由於沒有物件名稱，故外部程式必須指名類別名稱才能夠執行，如下語法：

147

7.10.2類別變數與類別方法

- **類別變數(Class Variables)**與**類別方法(Class Methods)**合稱類別成員(Class Members)
- 在宣告類別成員時，必須將修飾字宣告為**static**，如同前面的語法格式，而使用時，由於沒有物件名稱，故外部程式必須指名類別名稱才能夠執行，如下語法：

- **外部程式存取「類別變數」與執行「類別方法」語法：**

```
類別名稱.類別變數;
類別名稱.類別方法(引數串);
```

148

7.10.2類別變數與類別方法

- 【說明】
  - (1)在類別內部的程式，不需要指名類別名稱即可取用這些類別變數或執行類別方法。
  - (2)即使透過static宣告為「類別變數」與「類別方法」，當該類別生成物件時，仍可以當作物件的「變數」與「方法」來存取，存取語法與實體變數或方法完全相同，只不過，這些由該類別生成的物件，「必須共用類別變數」，同時，呼叫類別方法也是由同一個記憶體內容指出被呼叫端位於的程式段。

149

7.10.2類別變數與類別方法

- 當編譯器完成編譯動作並開始執行時
  - 於載入類別的同時，如果發現類別中包含類別變數與類別方法，就會立刻配置這些類別成員（不必等到物件實體產生時）
  - 而對於實體變數與實體方法，則必須等到執行new敘述而配置物件實體時才會配置。
- 以下，我們透過一個範例來觀察類別變數與類別方法的使用方式與記憶體配置。
  - 【觀念範例7-16】：存取類別變數與呼叫類別方法。
    - **範例7-16：**ch7_16.java（隨書光碟myJava\ch07\ch7_16.java）

150

7.10.2類別變數與類別方法

```
1  /*   檔名:ch7_16.java          功能:類別變數與類別方法   */
2
3  package myJava.ch07;
4  import java.lang.*;
5
6  public class ch7_16          //主類別
7  {
8      public int Var;
9      public static int sVar;
10
11     public void show(String str)
12     {
13         System.out.println(str + "'s Var=" + Var);
14         System.out.println("ch7_16 class sVar=" + sVar);
15     }
16     public static void add()          //類別方法
17     {
18         //Var++;
19         sVar++;
20     }
21
```

錯誤,因為類別方法內不可以存取實體變數

執行結果

```
X's Var=3
ch7_16 class sVar=7
Y's Var=10
ch7_16 class sVar=7
```


7.10.2類別變數與類別方法

```
22     public static void main(String[] args)
23     {
24         ch7_16 X = new ch7_16();
25         ch7_16 Y = new ch7_16();
26         X.Var = 3;
27         Y.Var = 10;
28         X.sVar = 5;          //使用物件存取類別變數
29         X.add();             //使用物件執行類別方法
30         Y.add();             //使用物件執行類別方法
31         X.show("X");
32         Y.show("Y");
33     }
34 }
```

- 範例說明：
  - (1)這個範例只有一個類別ch7_16。ch7_16類別內包含一個實體變數Var，一個實體方法show。
  - (2)ch7_16類別內包含一個類別變數sVar，兩個類別方法main, add。在add中，不可以存取Var，因為它是類別方法，不可以直接存取實體變數。

7.10.2類別變數與類別方法

- (3)X,Y是ch7_16類別的兩個物件實體的參考。
- (4)在第26~32行，我們透過X,Y（物件名稱）存取實體變數及類別變數，呼叫實體方法及類別方法，語法都相同。
- (5)由執行結果中，可以發現，不論是由X物件或Y物件呼叫show函式，印出的sVar都相同，因為sVar是「共用的變數」。
  - 在第28行首先透過X將sVar設定為5
  - 然後在第29行將它加1。
  - 第30行則是透過Y，將sVar又加1。因此最後印出的為7
- (6)本範例執行完畢時的記憶體圖示如下，其中，main,add,sVar是所有該類別之生成物件所共用的，故它只有一份拷貝，並且該份拷貝在類別被載入記憶體時，就被配置了（並非等到new敘述執行時）。

153

7.10.2類別變數與類別方法

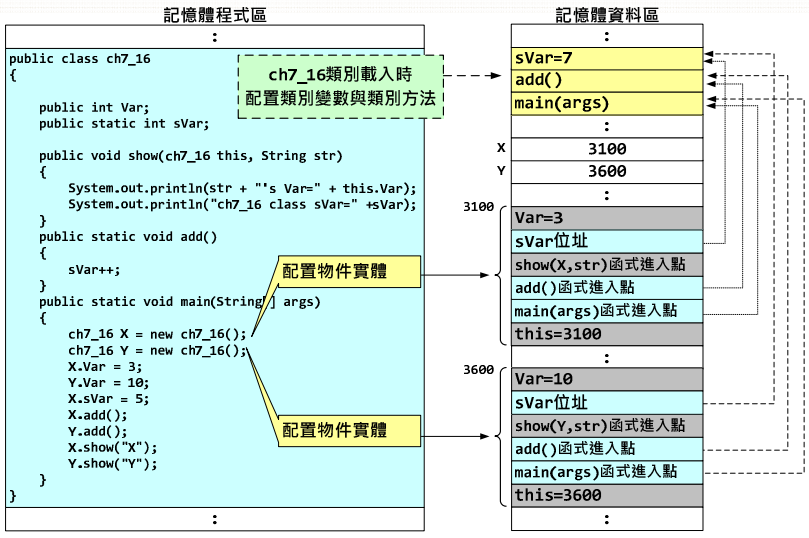


圖7-12 實體變數（及方法）與類別變數（及方法）

154

7.10.2類別變數與類別方法

- 【觀念範例7-17】：透過類別名稱存取類別變數與呼叫類別方法。
  - 範例7-17：ch7_17.java（隨書光碟 myJava\ch07\ch7_17.java）

```
1  /*  檔名:ch7_17.java      功能:類別名稱存取類別成員  */
2
3  package myJava.ch07;
4  import java.lang.*;
5
6  public class ch7_17          //主類別
7  {
8      public static void main(String args[])
9      {
10         CMyClass.show("修改前");     //執行類別方法
11         CMyClass.Var = 20;           //存取類別變數
12         CMyClass.show("修改後");     //執行類別方法
13         CMyClass X = new CMyClass();
14         X.show("使用物件呼叫時");
15     }
16 }
17
```

執行結果

```
修改前Var=10
修改後Var=20
使用物件呼叫時Var=20
```


7.10.2類別變數與類別方法

```
18 class CMyClass
19 {
20     public static int Var=10;
21
22     public static void show(String str)
23     {
24         System.out.println(str+ "Var=" + Var);
25     }
26 }
```

- 範例說明：
  - (1)這個範例有兩個類別。其中CMyClass只有一個類別變數與一個類別方法。
  - (2)在ch7_17類別中的main函式如果要存取CMyClass的類別變數，並不需要產生物件，只要註明類別名稱即可，如第11行。呼叫類別方法，也不需要產生物件，只要註明類別名稱即可，如第10、12行。
  - (3)即使透過物件呼叫類別方法，您也可以由執行結果看出物件X存取到的Var，仍是在類別一開始就被配置的Var變數，也就是類別變數仍被共用，而非專屬於某一個物件。

7.10.3 類別方法的限制

- 類別方法由於被宣告為static，所以只能存取被宣告為static的類別變數，同時，它也只能直接呼叫被宣告為static的類別方法。
  - 由於main函式已經被宣告為static，所以也只能呼叫類別方法。
- 除了上述限制之外，**在類別方法內也不能使用this關鍵字**
  - 因為this在執行時代表的是一個物件的參考，而由於類別方法可以直接由類別名稱來呼叫執行，因此呼叫當時並沒有所謂的物件實體，當然也就不會有物件實體的參考，故而在類別方法使用this關鍵字是不合法的。

157

7.10.4 類別成員的初始化

- 建構子在物件實體產生時，自動被new敘述呼叫，進行物件實體的初始化設定
  - 可是類別成員在類別被載入時就會被配置記憶體，如果想要將類別變數初始化，除了設定初值之外，就必須透過static區段來設計初始化動作，如下語法：

```
static{
  ……敘述群（可進行類別變數初始化）……
}
```

  - 【語法說明】：
    - 上面這個static區段，必須放在類別內，它不屬於任何方法，並且會在類別變數配置之後自動開始執行。

158

7.10.4 類別成員的初始化

- 【觀念範例7-18】：改寫範例7-17，透過static區段初始化類別變數。
  - 範例7-18：ch7_18.java（隨書光碟 myJava\ch07\ch7_18.java）

```
1   /*   檔名:ch7_18.java      功能:static區段   */
6   public class ch7_18        //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass.show("修改前");    //執行類別方法
11          CMyClass.Var = 20;            //存取類別變數
12          CMyClass.show("修改後");    //執行類別方法
13          CMyClass X = new CMyClass();
14          X.show("使用物件呼叫時");
15      }
16  }
17
```

159

7.10.4 類別成員的初始化

```
18  class CMyClass
19  {
20      public static int Var;
21
22      public static void show(String str)
23      {
24          System.out.println(str+ "Var=" + Var);
25      }
26      static{
27          Var = 100;
28          System.out.println("類別變數Var初始化完畢" + Var);
29          show("初始化之後");
30      }
31  }
```

- 執行結果：

```
類別變數Var初始化完畢100
初始化之後Var=100
修改前Var=100
修改後Var=20
使用物件呼叫時Var=20
```

160

7.10.4 類別成員的初始化

- 範例說明：
  - 從執行結果中，可以發現，static是最早被執行的程式區塊（比main函式還要早）
  - 因為JVM會先載入ch7_18與CMyClass類別後才執行主類別的main函式，而當載入CMyClass類別後，就會自動執行static區段的敘述。

161

7.10.5類別內的無名區段

- 在上一小節中，我們發現以static為名的區段會在類別載入後自動開始執行。
  - Java允許我們在類別內定義一個無名的區段，而該區段會在物件實體產生時，自動被執行，並且執行時機比建構子還要早。
  - 利用建構子對物件進行初始化是比較正統的做法，因此建議盡量少用這樣的語法。
  - 【觀念範例7-19】：類別內無名區段的執行時機
  - **範例7-19：**ch7_19.java（隨書光碟myJava\ch07\ch7_19.java）

162

7.10.5類別內的無名區段

```
1   /*   檔名:ch7_19.java       功能:類別內的無名區段   */
6   public class ch7_19         //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          X.show("使用物件呼叫時");
12      }
13  }
14
15  class CMyClass
16  {
17      public int Var;
18      public CMyClass()
19      {
20          Var = 10;
21          System.out.println("建構函式執行中");
22          show("建構函式執行完畢時");
23      }
```

163

7.10.5類別內的無名區段

```
24  public void show(String str)
25      {
26          System.out.println(str+ "Var=" + Var);
27      }
28      {
29          Var = 100;
30          System.out.println("無名區塊執行中");
31          show("無名區塊執行完畢時");
32      }
33  }
```

無名區段

- 執行結果：

```
無名區塊執行中
無名區塊執行完畢時Var=100
建構函式執行中
建構函式執行完畢時Var=10
使用物件呼叫時Var=10
```

164

7.10.5類別內的無名區段

- 範例說明：
  - 由執行結果中可以發現，無名區段會在建構函式開始前就先被執行。
  - 而如果您將第10行的new敘述去除，例如改寫為CMyClass X;（當然第11行必須刪除）則無名區段也不會被執行，因為物件實體尚未產生。

Coding 注意事項

不論是無名區段或static區段，都可以有多個，其執行順序為由上而下。

165

本章結束

Q&A討論時間

延伸學習：**巢狀類別**

巢狀類別指的是類別內又定義了另一個類別，由於巢狀類別的語法較為複雜，因此在本章先不介紹，但是由於視窗程式常會使用內部類別（也屬於巢狀類別）實作一個事件傾聽者，因此，我們會在視窗程式設計單元介紹巢狀類別。

166