

# 第十五章 GUI設計 (AWT/Swing)與內部類別

## 課前指引

圖形化程式設計就是設計視窗型態的應用程式，而Java則將圖形化使用者介面(Graphic User Interface；GUI)交由AWT與Swing類別庫來負責。

在本章及下一章中，我們將介紹如何撰寫Java的視窗程式，由於AWT與Swing類別庫都非常龐大，因此，我們只會擇要介紹，並且將重點放在Java視窗程式的原理介紹。至於未示範的視窗元件，則只需要依樣畫葫蘆即可妥善應用於視窗程式中。而由於視窗程式牽涉到與使用者互動的事件處理，各種視窗程式語言對於事件處理的語法格式略有不同，在Java中，最常以內部類別來實作事件處理，因此，我們也將在本章的最後來說明何謂「內部類別」，以便讀者能夠順利學習下一章的內容。

## 章節大綱

15.1 Java的視窗元件類別

15.4 收納器內的收納器

15.2 Java視窗應用程式的基礎架構

15.5 內部類別(巢狀類別)

15.3 版面編排

15.6 本章回顧

## 15.1 Java的視窗元件類別

### ● Java的視窗元件類別庫有AWT與Swing兩種

- 其中AWT為早期的設計，而Swing則是基於AWT基礎所設計的新視窗元件。
- 在JDK 1.1時，Java就已經提供了AWT(Abstract Window Toolkit)類別庫，全名是java.awt package。
- 而在JDK1.2(Java2)之後，Java則新增了Swing類別庫，全名是javax.swing package。

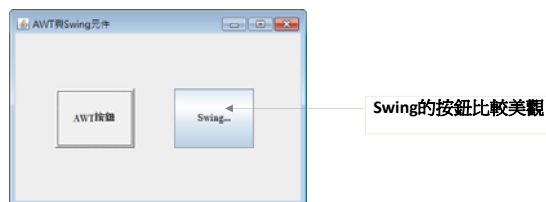


圖15-1 AWT與Swing元件的外觀

## 15.1 Java的視窗元件類別

- 這兩個系列的某些元件具有相同功能，但Swing的元件種類較多。
- 區分的方式非常簡單，Swing的元件一般都以「J」為開頭，例如標籤元件在AWT中，以Label類別來負責，而Swing則是JLabel類別。
- AWT的元件是利用作業系統現有的元件，故與作業系統有關，並且AWT所提供的元件較單調，也缺乏樹狀元件、目錄元件等功能較複雜的元件。
- 以 AWT 開發的物件：Heavyweight
- 以 Swing 開發的物件：Lightweight

## 15.1 Java的視窗元件類別

- Swing是針對AWT的缺失所發展的類別庫，因此擁有樹狀元件等較複雜的元件，並且Swing強調是完全由Java程式碼仿照視窗元件所開發而成，故與作業系統無關，但也因此顯示速度較慢。

## 延伸學習：其他的Java圖形化元件

除了Oracle(Sun)提供的AWT與Swing圖形化元件之外，IBM也曾經開發過一套Java圖形化元件SWT(Standard Widget Toolkit)，不過由於在文件說明方面不易讓開發人員了解，故一直未形成主流。

## 15.1 Java的視窗元件類別

## ● 收納器(Container)

- AWT的元件分為一般元件與收納器(Container)兩類，其中收納器元件可以用來容納其他視窗元件
  - 例如視窗本身就是一個收納器元件，所有呈現在視窗中的元件都被收納在視窗元件之中，這樣的物件稱之為收納器物件
  - 以 java.awt.Container 類別為基礎的，都是收納器物件



## 老師的叮嚀

「收納器」就像是盒子一般，它可以放入元件，也可以放入另一個收納器，您可以將外部的收納器看作是較大的盒子，而內部的收納器看作是放在大盒子中的較小盒子，兩者都可以收納元件。

## 15.1 Java的視窗元件類別

- 除了選單(Menu)以外，AWT所有的類別基礎皆為 `java.awt.Component` 類別
- 而Swing大部分的類別基礎為 `javax.swing.JComponent`，且 `javax.swing.JComponent` 由於繼承自 `java.awt.Container`，故所有的Swing元件幾乎都具有容納其他視窗元件的功能
- 這也使得Swing元件看起來比較美觀。
- 圖15-2是AWT與Swing的部分類別繼承圖（底色部分為Swing類別庫）
- 完整的繼承圖請參閱JDK說明文件。

## 15.1 Java的視窗元件類別

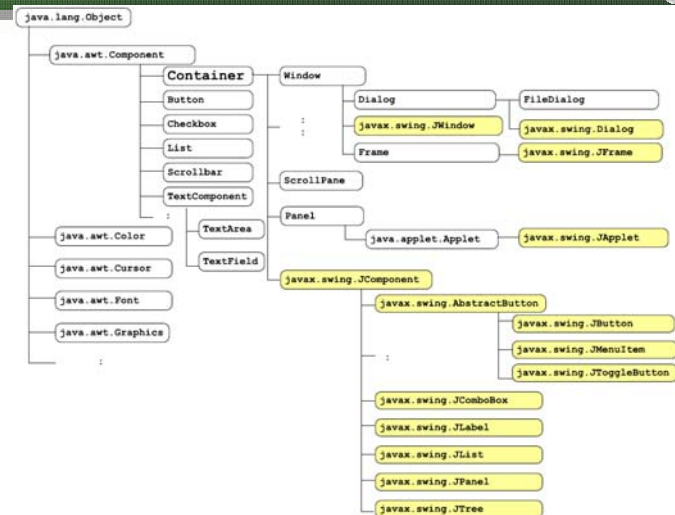


圖15-2 AWT與Swing的部分類別繼承圖  
【省略前方類別庫者皆為java.awt】

## 15.2 Java視窗應用程式的基礎架構



●一個視窗基本上是靠Frame或JFrame物件所構成，它是視窗程式的基礎，由於它是一個收納器，故可將其他視窗元件納入其中，首先我們先來看幾個陽春的Java AWT視窗程式。

●【觀念範例15-1】：使用AWT的Frame類別設計一個陽春的視窗程式。

●範例15-1：ch15\_01.java (隨書光碟 myJava\ch15\ch15\_01.java)

9

## 15.2 Java視窗應用程式的基礎架構



```
1  /* 檔名:ch15_01.java      功能:AWT Frame-視窗程式 (主類別繼承方式) */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*;      //載入AWT類別庫
6
7  public class ch15_01 extends Frame
8  {
9      public ch15_01()
10     {
11         this.setTitle("這是AWT視窗"); //設定視窗的標題列
12         this.setSize(300,200);        //設定視窗的寬與高
13         this.setVisible(true);        //設定視窗是否可被看見
14     }
15
16     public static void main(String args[])
17     {
18         new ch15_01();
19     }
20 }
```

繼承Frame類別就是一個AWT視窗類別

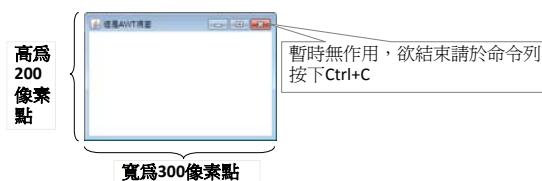
產生視窗類別的實體，代表產生一個視窗

10

## 15.2 Java視窗應用程式的基礎架構



●執行結果




●範例說明：

- (1)在第7行，我們宣告了主類別繼承Frame類別，因此，主類別本身就是視窗類別。
- (2)在第18行，產生主類別的物件，也就是產生一個視窗物件，此時會執行第9~14行的建構子，在當中，我們設定了視窗物件的屬性，包括設定視窗的標題列、視窗的寬與高、視窗是否可被看見等等。要設定這些屬性，都是透過Frame類別定義的特殊方法來完成。
- (3)本範例執行後，在螢幕左上角會出現視窗，這是因為我們並未設定視窗的位置，故預設出現在左上角。

11

## 15.2 Java視窗應用程式的基礎架構



- (4)本範例執行後，無法按下  關閉鈕來關閉視窗，這是因為我們還沒有設定按下關閉鈕的事件處理方式。若要關閉視窗，可以於執行ch15\_01類別的Dos視窗中，按下【Ctrl】+【C】鍵結束執行程式即可。

●事實上，一個Java程式可以產生多個視窗，每一個視窗以Frame物件來表示即可，請見下一個範例。

●【觀念範例15-2】：使用AWT的Frame類別設計一個產生兩個視窗的程式。

●範例15-2：ch15\_02.java (隨書光碟 myJava\ch15\ch15\_02.java)

12



## 15.2 Java視窗應用程式的基礎架構



```
1  /* 檔名:ch15_02.java 功能:AWT Frame-視窗程式 (類別成員與函式變數方式) */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*; //載入AWT類別庫
6
7  public class ch15_02 extends Frame
8  {
9      static ch15_02 frm1 = new ch15_02(); //類別成員
10
11      public static void main(String args[])
12      {
13          Frame frm2 = new Frame(); //函式變數
14          frm1.setTitle("這是第一個AWT視窗"); //設定視窗的標題列
15          frm1.setSize(300,200); //設定視窗的寬與高
16          frm1.setVisible(true); //設定視窗是否可被看見
17
18          frm2.setTitle("這是另一個AWT視窗"); //設定視窗的標題列
19          frm2.setSize(200,300); //設定視窗的寬與高
20          frm2.setLocation(300,300); //設定視窗的位置
21          frm2.setVisible(true); //設定視窗是否可被看見
22      }
23  }
```

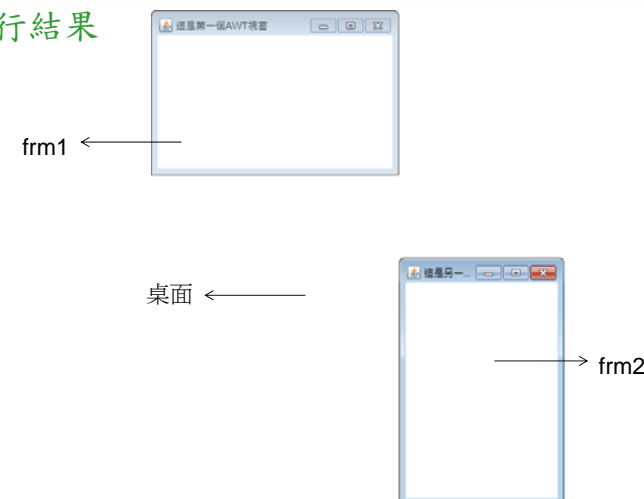
類別內也可以用類別本身當作型態來宣告物件

13

## 15.2 Java視窗應用程式的基礎架構



### ● 執行結果



14

## 15.2 Java視窗應用程式的基礎架構



### ● 範例說明：

- (1) 在第9行，我們使用類別成員的方式宣告了一個成員為ch15\_02類別的實體frm1，由於ch15\_02類別繼承了Frame類別，故也是一個視窗。在第13行，我們使用物件變數的方式宣告了一個函式的物件變數（區域變數）frm2，型態為Frame類別，故也是視窗。
- (2) frm1與frm2都是視窗，所以執行結果有兩個視窗。
- (3) frm2.setLocation(300,300);設定了視窗位置，所以第二個視窗不是位於螢幕左上角。
- (4) 第9行的static是因為該實體要在static main中使用，故宣告為static，若不是在static方法中使用，則成員不需要宣告為static。

15

### 15.2.1 Frame視窗應用程式的基礎架構



- 上面兩個程式都可以產生視窗元件，並且由上一個範例可知，主類別也不一定就是視窗類別，我們其實可以另外宣告一個視窗類別，只要該類別繼承Frame即可。

- 由於視窗元件是收納器的一種，因此，我們若採用繼承格式宣告視窗類別，可以將其他元件加入到該視窗之中，所以整體使用Frame的視窗程式碼架構可以整理如下：

16

## 15.2.1 Frame視窗應用程式的基礎架構



```
import java.lang.*;
import java.awt.*; //使用AWT元件時需載入
import javax.swing.*; //使用Swing元件時需載入

[封裝等級] class 視窗類別名稱 extends Frame //類別代表視窗
{
    //使用類別成員方式宣告視窗內的元件，例如Button、Label等等
    public 視窗類別名稱() //建構子
    {
        //決定版面編排
        //建構各元件的實體
        //設定各元件的屬性，例如外觀，大小等等
        //設定各元件對應的事件傾聽機制，可搭配內部類別(Inner Class)完成
        //透過Frame的add()方法，將各元件加入本視窗中
        //設定視窗的屬性，例如位置，大小等等
        //設定視窗對應的事件傾聽機制，可搭配內部類別(Inner Class)完成
        //顯示視窗，例如使用setVisible()
    }
}

[封裝等級] [修飾字] 回傳值資料型態 方法名稱 (參數串宣告)
{
    //此處可以出現如建構子的建立元件與設定視窗的程式碼
}

//事件傾聽機制，採用內部類別(Inner Class)完成
{
    :
}
```

內部類別於本章最後介紹

17

## 15.2.1 Frame視窗應用程式的基礎架構



### ● 架構說明：

- (1)由於類別即視窗本身，故在類別內操作視窗時，可使用this作為表示。
- (2)事件傾聽機制是Java的事件處理方式，例如鍵盤或滑鼠的動作發生之時就是一個事件，我們將在下一章中說明。

● 【觀念範例15-3】：依照上述格式，建立Frame視窗類別，並使用多種方式為Frame視窗加入元件。

● 範例15-3：ch15\_03.java (隨書光碟 myJava\ch15\ch15\_03.java)

18

## 15.2.1 Frame視窗應用程式的基礎架構



```
1  /* 檔名:ch15_03.java      功能:Frame視窗內加入元件 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*; //載入AWT類別庫
6
7  public class ch15_03
8  {
9      public static void main(String args[])
10     {
11         CMyWindow frm1 = new CMyWindow(); //視窗實體
12
13         frm1.addNewLabel("新增標籤");
14     }
15 }
16
17 class CMyWindow extends Frame //類別繼承Frame類別
18 {
19     //使用類別成員方式宣告視窗內的元件
20     Button btn1; //類別成員
21     Button btn2; //類別成員
```

19

## 15.2.1 Frame視窗應用程式的基礎架構



```
22 public CMyWindow() //建構子
23 {
24     this.setLayout(null); //設定版面編排,手動指定各元件位置
25
26     btn1 = new Button(); //建構各元件的實體
27     btn2 = new Button(); //建構各元件的實體
28
29     //設定各元件的屬性，例如外觀，大小等等
30     btn1.setLabel("按鈕1");
31     btn1.setBounds(75,100,100,75); //設定按鈕位置與大小等等
32     btn2.setBounds(25,50,50,25); //設定按鈕位置與大小等等
33
34     //設定各元件對應的事件傾聽機制，暫時省略
35
36     //透過Frame的add()方法，將各元件加入本視窗中
37     this.add(btn1);
38     this.add(btn2);
39
40     this.setTitle("加入元件的Frame視窗"); //設定視窗的屬性
41     this.setSize(300,300); //設定視窗的屬性
42
43     //設定視窗對應的事件傾聽機制，暫時省略
44
45     this.setVisible(true); //顯示視窗
46 }
```

在視窗產生的建構子中就加入按鈕元件

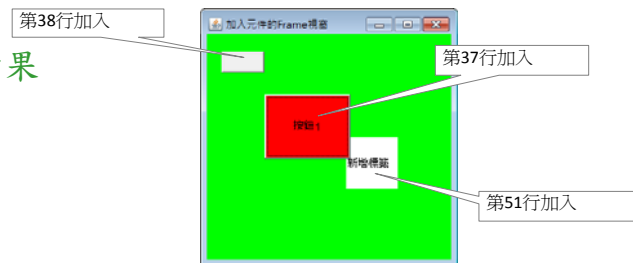
20

```

47 public void addNewLabel(String str1)
48 {
49     Label lab = new Label(str1);    //產生標籤實體
50     lab.setBounds(170,150,60,60);  //設定標籤位置與大小
51     this.add(lab);                  視窗產生後才加入
52                                     標籤元件
53     this.setBackground(Color.green);
54     btn1.setBackground(Color.red);
55 }
56
57 // 事件傾聽機制，採用內部類別(Inner Class)完成，暫時省略
58 }

```

## ● 執行結果



21

## ● 範例說明：

- (1) 我們仿照之前介紹的格式宣告了一個視窗類別 CMyWindow，並在其中宣告了兩個成員 btn1 與 btn2，這兩個成員是按鈕元件。
- (2) 當主程式執行到第11行時，會產生視窗實體 frm1，並且執行第22~46行的建構子。在建構子中，我們首先產生按鈕的實體（第26、27行），並設定按鈕的屬性（第30~32行）。然後透過 add() 方法將按鈕加入到視窗之中（第37、38行），此時的 this 也就是 frm1 視窗。然後設定視窗的標題列與大小（第40、41行），最後將視窗顯示出來（第45行）。

22

- (3) 當第11行執行完畢時，事實上，已經產生了一個視窗，視窗內包含兩個按鈕。當程式執行到第13行時，又執行了視窗物件的一個方法 addNewLabel，這是我們自行撰寫的方法，可以改變視窗的內容並且加入一個標籤元件。
- (4) 第13行執行時會執行第47~55行的程式，同樣地，我們宣告了一個標籤元件變數 lab，並透過 add() 方法加入到視窗中。請注意，由於我們是在函式內宣告該變數，故此標籤無法被其他的函式或外部程式更動其屬性。
- (5) 第53行將視窗的背景設定為綠色，第54行則將 btn1 按鈕的背景設定為紅色。Color 是一個 awt 的類別，其內使用了眾多欄位替各種顏色取一些代號（如果您查閱了 Color 類別的說明文件，會發現全部使用大寫也可以）。

23

- (6) 當第13行執行完畢後，視窗與 btn1 按鈕的背景顏色被改變了，同時視窗內也增加了一個標籤。由於執行速度過快，因此，您可能無法發現第11行與第13行執行完畢，視窗的改變。我們會在後面章節中，採用事件方法來撰寫程式，就可以明顯看出視窗的改變。
- (7) 值得注意的是，視窗內標籤元件與按鈕元件重疊了，這是因為我們在第24行採用了手動方式配置視窗內的元件。而第31、32、50行的 setBounds 就是用來設定元件所在位置，前兩個引數是設定元件對於收納器的左邊界與上邊界的距離，後兩個引數則是元件的寬與高（即元件的大小設定）。由於元件是透過視窗的 add() 方法加入，故此處的收納器就是視窗本身。

24



## ●使用JFrame的視窗程式碼架構與Frame視窗類似

- 不過由於JFrame的視窗內含許多編排層級，通常我們是將元件加入到ContentPane這個層級的收納器中，故一開始應該取得JFrame的ContentPane。

## ●整體使用JFrame的視窗程式碼架構可以整理如下：

```
import java.lang.*;
import javax.swing.*; //使用Swing元件時需載入
import java.awt.*;    //使用AWT元件時需載入

[封裝等級] class 視窗類別名稱 extends JFrame //類別代表視窗
{
    //使用類別成員方式宣告視窗內的元件，例如JButton、JLabel等等
    public 視窗類別名稱() //建構子
    {
        //透過getContentPane()取得JFrame的ContentPane
        //決定ContentPane的版面編排方式
        //建構各元件的實體
        //設定各元件的屬性，例如外觀，大小等等
        //設定各元件對應的事件傾聽機制，可搭配內部類別(Inner Class)完成
        //透過ContentPane的add()方法，將各元件加入ContentPane中
        //設定ContentPane的屬性，例如背景顏色等等
        //設定JFrame視窗對應的事件傾聽機制，可搭配內部類別(Inner Class)完成
        //顯示JFrame視窗，例如使用JFrame的setVisible()
    }
    [封裝等級] [修飾字] 回傳值資料型態 方法名稱 (參數串宣告)
    {
        //此處可以出現如建構子的建立元件與設定視窗的程式碼
    }
    :
    // 事件傾聽機制，採用內部類別(Inner Class)完成
    :
}
```

### ●架構說明：

- (1)ContentPane本身是一個收納器，我們將元件加入到此收納器中，而非直接加入到JFrame視窗中。
- (2)要改變視窗的外觀，有些屬性是在ContentPane中設定，有些屬性則是在JFrame中設定，您可以把ContentPane視為視窗主選單列以下的部分。
- 【觀念範例15-4】：依照上述格式，建立JFrame視窗類別，並使用多種方式為視窗加入元件。
- 範例15-4：ch15\_04.java (隨書光碟 myJava\ch15\ch15\_04.java)

### 延伸學習：視窗元件的屬性與方法

**屬性**是一個「類別／物件」的成員變數（在Java中稱之為欄位），方法則是一個「類別／物件」的成員方法。而視窗元件是java.awt package某類別的物件，或javax.swing package某類別的物件。在前面章節中，我們曾經說過，每一個按鈕都是一個物件，屬於按鈕(Button)類別之下所建造出來的物件實體，但是由於按鈕標題(label)的不同，因此視為不同且獨立的物件。

而在本章中，我們發現到，java並不讓我們直接設定視窗元件的屬性，例如想要設定Button的標題文字，應該透過setLabel()方法來完成。但事實上，用來代表Button物件的標題文字仍舊是Button物件的屬性（欄位），只不過它屬於package等級，故我們無法直接在其他package中使用它，而必須透過public method來設定它。如果您觀察JDK的原始碼，可以在Button.java原始碼檔中發現，Button類別的文字使用的是label欄位來存放，該欄位是一個package等級的String型別變數。

但對於其他支援視窗程式的程式語言來說，有些語言（例如VB）允許我們直接設定某個元件的屬性來設定元件的外觀，而不需要透過函式來間接設定。有些語言甚至將之區分為屬性與欄位兩種（例如VB.NET）不同層次的功能，欄位屬於變數，而屬性具有函式的效果，只不過該函式是特別用來設定特定欄位值之用。這樣做的好處在於，可以於設定欄位值之前，進行一些檢查，甚至可以設定成唯讀或唯寫的屬性。

## 15.2.2 JFrame視窗應用程式的基礎架構



```
1  /* 檔名:ch15_04.java      功能:JFrame視窗內加入元件 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import javax.swing.*;      //載入Swing類別庫
6  import java.awt.*;          //載入AWT類別庫,例如Color類別為AWT類別庫
7
8  public class ch15_04
9  {
10     public static void main(String args[])
11     {
12         CMyJWindow jfrm1 = new CMyJWindow();    //視窗實體
13         jfrm1.addNewLabel("新增標籤");
14     }
15 }
16
17
18 class CMyJWindow extends JFrame    //類別繼承JFrame類別
19 {
20     //使用類別成員方式宣告視窗內的元件
21     JButton btn1; //類別成員
22     JButton btn2; //類別成員
```

29

## 15.2.2 JFrame視窗應用程式的基礎架構



```
23 public CMyJWindow()    //建構子
24 {
25     Container cp = this.getContentPane();    cp代表視窗的ContentPane
26     cp.setLayout(null);    //設定版面編排,手動指定各元件位置
27
28     btn1 = new JButton("按鈕1");    //建構各元件的實體
29     btn2 = new JButton();
30
31     //設定各元件的屬性,例如外觀,大小等等
32
33     btn1.setBounds(75,100,100,75);    //設定按鈕位置與大小等等
34     btn2.setBounds(25,50,50,25);    //設定按鈕位置與大小等等
35     btn1.setFont(new Font("Serif",Font.BOLD,12));    //設定字型
36
37     //設定各元件對應的事件傾聽機制,暫時省略
38
39     //透過Frame的add()方法,將各元件加入本視窗中
40     cp.add(btn1);    加入按鈕到cp中
41     cp.add(btn2);
42
43     cp.setBackground(Color.yellow);    //設定ContentPane的背景顏色
44
45     this.setTitle("加入元件的JFrame視窗");    //設定視窗的屬性
46     this.setSize(300,300);    //設定視窗的屬性
47
48     //設定視窗對應的事件傾聽機制,暫時省略
49
50     this.setVisible(true);    //顯示視窗
51 }
```

30

## 15.2.2 JFrame視窗應用程式的基礎架構



cp1代表視窗的ContentPane

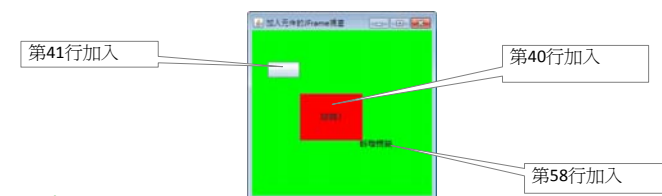
```
52 public void addNewLabel(String str1)
53 {
54     JLabel lab = new JLabel(str1);    //產生標籤實體
55     lab.setBounds(170,150,60,60);    //設定標籤位置與大小
56     lab.setFont(new Font("Serif",Font.BOLD,12));    //設定字型
57     Container cp1 = this.getContentPane();
58     cp1.add(lab);    加入標籤到cp1中
59
60     cp1.setBackground(Color.green);
61     btn1.setBackground(Color.red);
62 }
63
64 // 事件傾聽機制,採用內部類別(Inner Class)完成,暫時省略
65 }
```

31

## 15.2.2 JFrame視窗應用程式的基礎架構



### ● 執行結果



### ● 範例說明：

- (1) 本範例的流程與前一個範例差不多,不過使用JFrame時,我們是將元件加入到cp中,它是JFrame的ContentPane,使用getContentPane()即可取得,由於它會回傳一個Container類別的物件,故我們在第25行使用Container類別來宣告並接收它。並且由此處可知,它是一個收納器。

32



## 15.2.2 JFrame視窗應用程式的基礎架構



- (2) 第35行，我們透過`setFont()`來設定字型，由於它需要傳入一個Font型態的引數，故我們產生一個Font實體傳送給它，事實上，Font類別也是AWT的類別之一。
- (3) 本範例看似改用Swing元件完成與上一個範例相同功能的視窗程式，但事實上，兩者還是有一些差異，由於使用Frame是將元件直接加入到視窗中，因此視窗是元件的收納器，`setBounds()`設定的邊界是以收納器為準，故`btn1`與視窗邊界距離為(75, 100)。而JFrame是將元件加入到ContentPane中，因此ContentPane是元件的收納器，`setBounds()`設定的邊界是以收納器為準，故`btn1`與ContentPane邊界距離為(75, 100)。下圖為兩者的差異。

33

## 15.2.2 JFrame視窗應用程式的基礎架構

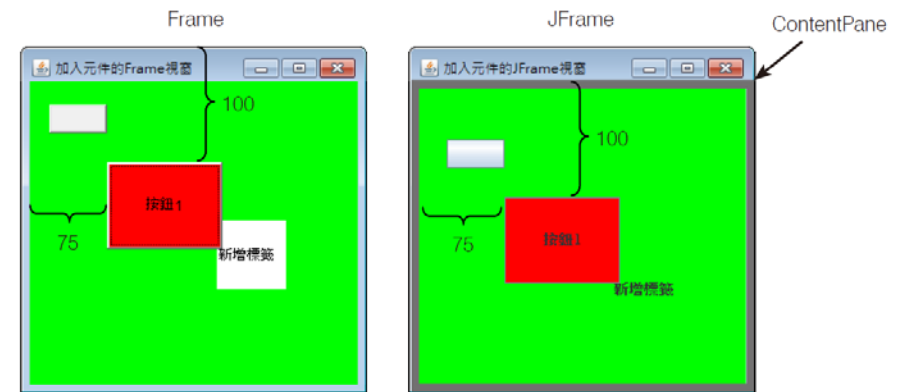


圖15-3 Frame與JFrame的差異

34

## 15.2.3 視窗的層次



### ● 視窗層次

- 在前面兩個範例中，我們已經知道Frame與JFrame的視窗有些許不同，事實上，JFrame比Frame複雜許多，首先我們先由Java視窗的結構來進行說明
- Java的視窗包含「標題列」、「主選單列(Menu Bar)」、「內容面板」等三大區塊，如下圖

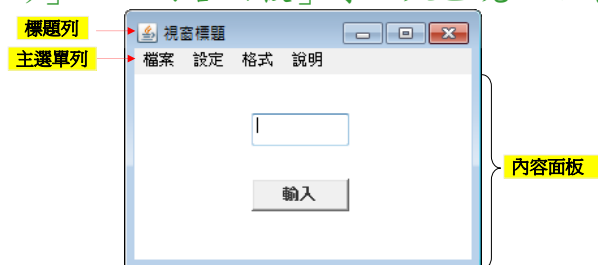


圖15-4 Java視窗結構

35

## 15.2.3 視窗的層次



- 「標題列」：
  - 包含放大、縮小、關閉鈕的那一列區域，它還包括視窗標題文字，它的高度約有20個像素。
- 「主選單列」：
  - 包含功能表選單的一列部分，也可以包含子選單，不展開子選單時，它的高度約有20個像素。
- 「內容面板」：
  - 放置元件的區域。
- 對於Frame視窗而言，Frame本身是一個收納器，並且包含上述三大區域
  - 由於我們將元件直接加入在Frame之中，所以至少應該距離上邊界20像素，否則會被覆蓋
  - 若視窗設計了主選單列，則應該距離上邊界40像素才不會被覆蓋。

36

## 15.2.3 視窗的層次



- 對於JFrame來說，情況變得非常複雜。
- 之前提過Swing大部分的元件都是繼承自JComponent類別，但有四個例外，分別是JFrame、JDialog、JWindow、與JApplet。
- 這些元件由於必須在作業系統顯示視窗，故需要使用作業系統的資源，而無法完全由Java程式碼設計。事實上，JFrame是繼承AWT的Frame類別，而不是繼承JComponent類別。
- 而這四種元件在Swing中，稱之為最上層(Top-Level)元件，所有的Swing元件都必須依附在其中之一才能顯示出來。
- 而這四種元件都實作了RootPaneContainer介面，該介面定義了各種收納器的設定與取得的方法。這些收納器包括RootPane、Glass Pane、Layered Pane和Content Pane等四種。

37

## 15.2.3 視窗的層次



- 在Java Swing的說明網頁中，可以發現下列一張圖，它說明了JFrame包含了上述四種Pane。

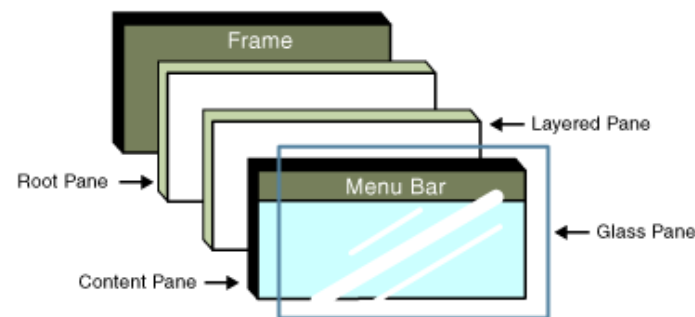


圖15-5 JFrame層次圖【節自Java Tutorial文件】

【參考

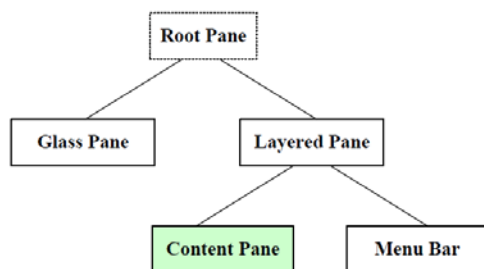
<http://download.oracle.com/javase/tutorial/uiswing/components/toplevel.html>】

38

## 15.2.3 視窗的層次



- 上圖中的各種類別可透過下圖的架構來說明
- 其中，RootPane並非真實的收納器（可視為虛擬的收納器），它是由Glass Pane與Layered Pane所組成
- 而Layered Pane則包含了Content Pane（內容面板）與Menu Bar（功能表列）



39

## 15.2.3 視窗的層次



- 事實上，Layered Pane分為許多層，Content Pane與Menu Bar只是其中一層，該層為Frame-Content Layer。
- 由於超出本書範圍，若有興趣者，可參閱筆者合著的「精通Java Swing程式設計」一書的第三章與第五章。
- 在本書中，我們只要了解，使用JFrame視窗時，元件是加入在Content Pane即可。

40

## 15.3版面編排



- 如果手動控制各元件在收納器版面中的位置，為了防止元件產生重疊狀況，因此要仔細計算各元件的絕對位置與大小。
- 而Java提供了版面管理員（Layout Manager），可以讓元件有秩序的擺放在適當位置，當視窗大小更動時，也會自動更新版面以配合視窗的大小，而不會因此出現紊亂的畫面。
- 在AWT中，共提供了五種版面管理員如下：

41

## 15.3版面編排



1. BorderLayout
  2. FlowLayout
  3. GridLayout
  4. CardLayout
  5. GridBagLayout
- 上述五項版面管理員，都是AWT提供的版面管理員，並且在Swing中仍可使用。
  - 事實上，除了上述五種之外，Swing還提供了SpringLayout與BoxLayout，以及較不需管理的ScrollPaneLayout、ViewportLayout與OverlayLayout等等。
  - 由於篇幅所限，在本節中，我們將只示範BorderLayout、FlowLayout與GridLayout等版面管理員，若讀者有興趣了解其他版面管理員，可參閱「精通Java Swing程式設計」一書。

42

### 15.3.1 BorderLayout



- BorderLayout是以邊界(border)作為版面配置依據
- 它將版面分割為東(EAST)、西(WEST)、南(SOUTH)、北(NORTH)、中(CENTER)五個區域，這些區域內的元件會填滿區域
  - 當收納器的大小改變時，也會自動調整收納器內的元件的尺寸，調整目標主要為中(CENTER)區域的大小
  - 一旦中區域大小確定後，東西區域的高度會與中央的邊界(border)對齊，而南北的寬度會與收納器的邊界對齊，以此為原則自動調整各區域的元件大小。

43

### 15.3.1 BorderLayout



- 收納器要使用BorderLayout作為版面管理員
  - 只要在setLayout()中傳入一個BorderLayout類別的物件引數即可。
  - 而Frame與JFrame的ContentPane預設都使用BorderLayout作為版面管理員，因此省略此步驟也可以。
- 【觀念範例15-5】：使用BorderLayout作為版面管理員，並觀察當視窗大小變動時，元件的變化。
- 範例15-5：ch15\_05.java（隨書光碟myJava\ch15\ch15\_05.java）

44



## 15.3.1 BorderLayout



```
1  /* 檔名:ch15_05.java      功能:BorderLayout版面管理員 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*;      //載入AWT類別庫
6
7  public class ch15_05
8  {
9      public static void main(String args[])
10     {
11         Frame frml=new Frame();    //視窗實體
12         frml.setLayout(new BorderLayout());
13         frml.setTitle("使用BorderLayout版面管理員");
14         frml.add(new Button("東按鈕"),BorderLayout.EAST);
15         frml.add(new Button("西按鈕"),BorderLayout.WEST);
16         frml.add(new Button("南按鈕"),BorderLayout.SOUTH);
17         frml.add(new Button("北按鈕"),BorderLayout.NORTH);
18         frml.add(new Label("中央標籤",Label.CENTER),
19                     BorderLayout.CENTER);
20
21         frml.setSize(300,300);
22         frml.setVisible(true);
23     }
24 }
```

可省略，因為預設也是使用  
BorderLayout來配置版面

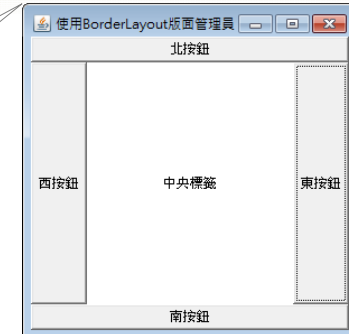
45

## 15.3.1 BorderLayout



### ● 執行結果

拉動視窗大小，各區域  
也會隨之縮放，並且區  
域內的元件也會跟著縮  
放



### ● 範例說明：

- (1)第12行使用setLayout()方法設定要使用BorderLayout作為版面管理員。第14~17行分別加入四個按鈕，加入時指定加入到哪一個區域。例如加入到西區域，則將區域引數設定為BorderLayout.WEST。

46

## 15.3.1 BorderLayout



- (2)第18行，加入標籤到中央區域，事實上，省略BorderLayout.CENTER結果也相同，因為區域預設參數值為BorderLayout.CENTER。本行中的Label.CENTER是設定標籤文字置於標籤中央（即置中對齊效果）。
- (3)請讀者自行將視窗拉大縮小，將會發現所有的元件都會跟著自動調整大小。

- 【觀念範例15-6】：使用BorderLayout作為版面管理員，並在某些區域不加入元件，觀察其變化。

- 範例15-6：ch15\_06.java（隨書光碟myJava\ch15\ch15\_06.java）

47

## 15.3.1 BorderLayout



```
1  /* 檔名:ch15_06.java      功能:BorderLayout版面管理員 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*;      //載入AWT類別庫
6
7  public class ch15_06
8  {
9      public static void main(String args[])
10     {
11         Frame frml=new Frame();    //視窗實體
12         frml.setLayout(new BorderLayout());
13         frml.setTitle("使用BorderLayout版面管理員");
14         Button btn1 = new Button("東按鈕");
15         btn1.setBounds(75,100,100,75);
16         frml.add(btn1,BorderLayout.EAST);
17         frml.add(new Button("南按鈕"),BorderLayout.SOUTH);
18         frml.add(new Button("北按鈕"),BorderLayout.NORTH);
19         frml.add(new Label("中央標籤",Label.CENTER),
20                     BorderLayout.CENTER);
21
22         frml.setSize(300,300);
23         frml.setVisible(true);
24     }
25 }
```

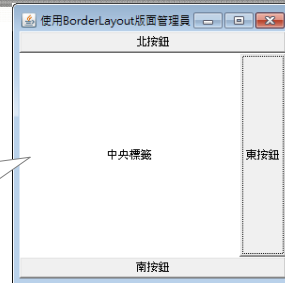
使用BorderLayout配置  
版面時，設定按鈕元件的位  
置與大小不會產生作用

48

## 15.3.1 BorderLayout

### ● 執行結果

西區域是空的，所以中央區域會自動往西放大延伸



### ● 範例說明：

- (1) 我們未在西區域加入元件，此時，中央區域會自動補上西區域欠缺的地方，並將中央區域內的元件自動往西放大。
- (2) 第15行，btn1透過setBounds()方法並不能改變該按鈕的配置位置與大小，因為該方法只在手動配置版面時有效，要手動配置版面應該使用setLayout(null)來設定Frame。

49

## 15.3.2 FlowLayout

### ● FlowLayout採用的是是一個一個排下去的流程編排方式

- 由上而下，由左而右，只有在上方已無法加入元件時，才會往下一列排放。

- 【觀念範例15-7】：使用FlowLayout作為版面管理員，並設定元件間的距離為50，並且置中排放。

- 範例15-7：ch15\_07.java (隨書光碟 myJava\ch15\ch15\_07.java)

50

## 15.3.2 FlowLayout

```
1  /* 檔名:ch15_07.java      功能:FlowLayout版面管理員 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*;      //載入AWT類別庫
6
7  public class ch15_07
8  {
9      public static void main(String args[])
10     {
11         Frame frml=new Frame();    //視窗實體
12         frml.setLayout(new FlowLayout(FlowLayout.CENTER,50,50));
13
14         frml.setTitle("使用FlowLayout版面管理員");
15         frml.add(new Button("按鈕1"));
16         frml.add(new Button("按鈕2"));
17         frml.add(new Button("按鈕3"));
18         frml.add(new Button("按鈕4"));
19         frml.add(new Button("按鈕5"));
20
21         frml.setSize(360,240);
22         frml.setVisible(true);
23     }
24 }
```

對齊方式，水平距離，垂直距離

51

## 15.3.2 FlowLayout

### ● 執行結果



### ● 範例說明：

- (1) 第11行透過setLayout()方法設定要使用FlowLayout作為版面管理員。我們在產生FlowLayout實體時，傳入了三個引數，若您查閱FlowLayout類別的建構子，就會發現第一個參數為對齊方式，後兩個為元件的水平距離與垂直距離。
- (2) 這個範例加入了5個按鈕，執行時第一列最多放得下三個按鈕，故剩餘兩個按鈕被安排到下一列，並且由執行結果可以發現它們都是置中排列的。當您將視窗的寬度拉大時，由於第一列可以放下更多按鈕了，故按鈕4也被移到第一列的末端，使得第二列只剩下一個按鈕，並且仍舊是置中排列的。

52

## 15.3.3 GridLayout



### ●GridLayout需要指定欄與列，如此形成一個「表格般」的配置方式。

- 若產生GridLayout實體時，不輸入引數，則內定為一列一欄。
- 同樣地，GridLayout與FlowLayout類似，也是必須在第一列的所有格子填滿後，才會填入第二列的第一個格子內。
- 而且GridLayout會將所填入的元件大小設為相同大小，因為我們無法設定不同高度或寬度的格子。

53

## 15.3.3 GridLayout



- 若我們不確定之後加入的元件有幾個，而因此無法在程式撰寫時設定正確的列數以容納所有的元件，此時可以將列數設定為0
  - 如此將形同「不限列數」的格子（會自動計算列數以填滿收納器）
  - 當每一列的格子被填滿時，會自動產生新的一列以存放新增的元件（其他列會縮小高度）。
- 而若我們指定了列數與欄數，但加入了超過「列數x欄數」的元件
  - 此時會自動增加欄數以容納所有的元件，這樣就可能與預期的有所不同
  - 並且根據測試，有時元件不足以填入最後一列時，也可能會縮減欄數
  - 因此，使用GridLayout編排版面時，最好將列數設定為0或仔細計算元件數量來設計欄數與列數。
  - 【觀念範例15-8】：使用GridLayout作為版面管理員，並且設定為三欄，不設定列數，元件間無距離。
  - 範例15-8：ch15\_08.java（隨書光碟 myJava\ch15\ch15\_08.java）

54

## 15.3.3 GridLayout



```
1  /* 檔名:ch15_08.java      功能:GridLayout版面管理員 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*;      //載入AWT類別庫
6
7  public class ch15_08
8  {
9      public static void main(String args[])
10     {
11         Frame frml=new Frame();    //視窗實體
12         frml.setLayout(new GridLayout(0,3));  //欄數為3，不限定列數
13
14         frml.setTitle("使用GridLayout版面管理員");
15         frml.add(new Button("按鈕1"));
16         frml.add(new Button("按鈕2"));
17         frml.add(new Button("按鈕3"));
18         frml.add(new Button("按鈕4"));
19         frml.add(new Button("按鈕5"));
20
21         frml.setSize(300,300);
22         frml.setVisible(true);
23     }
24 }
```

55

## 15.3.3 GridLayout



### ●執行結果



### ●範例說明：

- 第12行透過setLayout()方法設定使用GridLayout作為版面管理員，並且為3欄不限定列數。由於我們有五個按鈕，因此會自動產生第二列。。

### 小試身手15-1

請將範例15-8的按鈕減少兩個（留下三個），然後重新編譯與執行，看看會有什麼結果？

56



## 15.4 收納器內的收納器



- 如果您在上一節的版面管理員中多加幾個元件測試的話，就會發現
  - BorderLayout的一個區域只能加入一個元件，如果加入第二個元件，則第一個元件會不見。
  - 而GridLayout也無法在一格內加入兩個元件，因此，每個元件的大小都是一樣的，這對於我們來說並不是很方便
- 不過，如果您在置放元件處，放入一個內部收納器（收納器也是元件的一種），則我們又可以在內部收納器中進行版面配置，如此就可以產生更多的變化，以符合我們的需求。請看下面這個範例。

57

## 15.4 收納器內的收納器



- 【觀念範例15-9】：透過收納器中的收納器技巧，使得GridLayout每一列的按鈕數不同。
- 範例15-9：ch15\_09.java（隨書光碟myJava\ch15\ch15\_09.java）

```
1  /* 檔名:ch15_09.java      功能:收納器中的收納器 */
2
3  package myJava.ch15;
4  import java.lang.*;
5  import java.awt.*;      //載入AWT類別庫
6
7  public class ch15_09
8  {
9      public static void main(String args[])
10     {
11         Frame frml = new Frame();    //視窗實體
12         frml.setLayout(new GridLayout(0,3)); //欄數為3，不限定列數
13
14         Panel pnl = new Panel();      pnl當作內部收納器
15         pnl.setLayout(new GridLayout(1,2)); //1列2欄
16         pnl.add(new Button("按鈕a"));
17         pnl.add(new Button("按鈕b"));
18     }
```

58

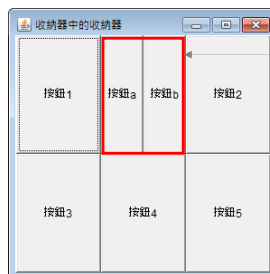
## 15.4 收納器內的收納器



```
19  frml.setTitle("收納器中的收納器");
20  frml.add(new Button("按鈕1"));
21  frml.add(pnl);
22  frml.add(new Button("按鈕2"));
23  frml.add(new Button("按鈕3"));
24  frml.add(new Button("按鈕4"));
25  frml.add(new Button("按鈕5"));
26
27  frml.setSize(300,300);
28  frml.setVisible(true);
29  }
30  }
```

把pnl內部收納器加入外部收納器frml之中

### ● 執行結果



這裡有一個Panel

59

## 15.4 收納器內的收納器



- 範例說明：
  - (1)第14行，我們宣告了一個Panel物件pnl，它是一個收納器，因此，我們可以在第15行設定它的版面管理員為GridLayout(1,2)，代表1列2欄。
  - (2)在第16~17行中，為pnl加入了兩個按鈕。
  - (3)對於視窗frml而言，我們採用的是GridLayout(0,3)的版面管理（第12行），並且在第20~25行中陸續加入元件，其中，第21行加入的是Panel元件，也就是一個內部的收納器。
  - (4)在執行結果中，您可以發現，第一列中間的欄位出現兩個按鈕，這是因為這兩個按鈕都屬於pnl收納器所有，故出現在pnl元件應該出現的格子中。

60

## 15.4 收納器內的收納器



- 從上述範例來看，在Java的視窗程式設計中，時常以多層的收納器作為設計基礎，這使得視窗元件的配置可以有更多樣的變化。
- 但事實上，要一次就完成所希望的配置並非易事，因此，通常在設計Java的視窗介面時，我們可以利用各種開發Java的IDE來為我們安排元件的配置，如此，我們就可以專注於開發程式的邏輯以及事件處理的程式碼。
- 在下一章中，我們將介紹Java的事件處理方式，如此本章所介紹的各個按鈕按下後，就可以執行某些程式，完成更多的需求。

61

## 15.5 內部類別（巢狀類別）



- 在說明下一章的內容之前，我們先來說明一項關於類別宣告的技術：巢狀類別，也就是內部類別。
- 內部類別由於較為複雜，故而在第七章中並未提及，但內部類別卻是Java視窗程式常見的技術，因此本書將之移到此處來做說明。

62

## 15.5 內部類別（巢狀類別）



- 首先我們回顧過往所學習到的技術，我們可以在類別內宣告一些變數，這些變數除了是原始資料型態之外，也可以是類別型態（代表該變數是物件變數）。例如下列範例：

```
class CMyClass1
{ ..... }
class CMyClass2
{
    CMyClass 1 var1,var2;
}
class CMyClass3
{ ..... }
```

- 上述範例中的var1、var2是CMyClass2的成員變數（欄位），並且CMyClass1也可以被其他的類別用來當作一種資料型態，例如CMyClass3。

63

## 15.5 內部類別（巢狀類別）



- 而如果我們有一個類別，它只想要讓某一個類別使用，則我們可以利用「巢狀類別」來完成。
- 例如，假設CMyClass1只想要被CMyClass2使用，則可以將CMyClass1類別的定義移到CMyClass2的定義之內，如此一來，CMyClass3就不能使用CMyClass1類別了。如下範例：

```
class CMyClass2
{
    private class CMyClass1
    { ..... }
    CMyClass 1 var1,var2;
}
class CMyClass3
{ ...不能使用CMyClass1類別... }
```

CMyClass1定義於CMyClass2之內，只有CMyClass2可以取用，而CMyClass3不能取用它

64

## 15.5 內部類別（巢狀類別）



● 所謂巢狀類別(Nested Class)，代表類別定義內還有其他類別的定義，當然這樣的層次不僅止於兩層，還可以定義更多層的巢狀類別。不過，為了簡化問題，我們暫時只討論兩層的巢狀類別。

- 兩層的巢狀類別之外層類別可以稱為外部類別(outer class)，由於它將內部的類別包圍，因此也稱為外圍類別(Enclosing Class)。
- 而內層的類別則分為三種
  - 一種是有名字的成員內部類別(member inner class)
  - 第二種是有名字的區域內部類別(local inner class)
  - 最後一種則是沒有名字的匿名類別(anonymous class)。

65

## 15.5 內部類別（巢狀類別）



- 外部類別如同一般定義類別的方式，並且「外部類別可以使用內部類別當作資料型態來宣告物件變數」，而在本節中，我們將分別介紹內部類別與匿名類別的規範。

### 延伸學習：靜態巢狀類別

事實上巢狀類別除了上述的三種類型之外，還有一種靜態巢狀類別(Static Nested Class)，不過為了簡化討論，我們暫時不介紹靜態巢狀類別。

66

### 15.5.1 成員內部類別 (member inner class)



● 在介紹Java的視窗程式設計時，我們可能會實作事件傾聽者的adapter類別（例如範例16-3），而它使用的就是內部類別。

- 在此，我們先介紹如何定義成員內部類別。定義內部類別必須在外部類別之內，並將之定義為其中一個成員，一般我們將成員內部類別(member inner class)簡稱為內部類別(inner class)，語法如下：
- 定義內部類別語法：

67

### 15.5.1 成員內部類別



```
[封裝等級] [修飾字] class 外部類別名稱
{
    [封裝等級] [修飾字] 外部類別成員;
    :
    [封裝等級] [修飾字] class 內部類別名稱
    {
        [封裝等級] [修飾字] 內部類別成員;
    }
}
```

- 【語法說明】：
  - (1) 成員內部類別也可以宣告封裝等級，不過我們暫時省略此欄位（在本節的最後，我們會討論一個被宣告為private的內部類別）。

68



## 15.5.1 成員內部類別



- (2)內部類別也可以宣告修飾字，但僅限於final與abstract。
- (3)內部類別的名稱可以和外部類別或其他一般類別相同，因為完整的類別名稱是「Package名稱.外部類別名稱.內部類別名稱」。
- (4)內部類別經由定義之後，可以在宣告外部類別成員時，將內部類別當作一種資料型態來宣告。但若僅宣告物件變數的話，則只是宣告了一個內部類別的物件參考而已。當外部類別產生物件實體時，並不會產生內部類別的物件實體。如果想要產生內部類別的物件實體來做應用的話，應該在外部類別的函式中，使用new來產生。
- (5)內部類別可以取用外部類別的成員，但請勿使用this來讀取外部類別的成員，因為外部類別與內部類別在產生物件實體時，各自擁有屬於自己的this。

69

## 15.5.1 成員內部類別



- (6)內部類別的private等級成員也可以在外部類別中使用。
- (7)內部類別編譯後，也會產生類別檔。同時檔名會出現完整的巢狀檔名，也就是外部類別\$內部類別.class。
- (8)若內部類別非宣告為static類別，則不可以宣告或定義static類別成員。
- 當我們定義了內部類別之後，就可以透過完整的類別名稱來宣告物件變數，語法如下：
- 一般類別宣告內部類別之物件變數語法：

```
外部類別名稱.內部類別名稱 物件變數;
```

70

## 15.5.1 成員內部類別



### ●【語法說明】：

- (1)上述語法是在外部類別以外的其他類別中使用內部類別的宣告語法，所以必須記載完整的類別名稱（包含外部類別以及巢狀層次關係）。
- (2)如果是在外部類別中宣告變數，則可以減少前面一層的「外部類別名稱.」。
- (3)如果內部類別被宣告為private，則其他的一般類別就無法使用它來宣告物件變數。
- (4)上述語法只產生了一個內部類別物件的參考。要產生實體，則必須使用下列語法：

71

## 15.5.1 成員內部類別



- 一般類別宣告內部類別之物件變數並產生物件實體語法：

```
外部類別名稱.內部類別名稱 物件變數  
= (new外部類別名稱(引數串)).new內部類別名稱(引數串);
```

### ●【語法說明】：

- 上述語法是將之前的語法合併，不過這樣做的結果，我們將只有一個內部類別的物件變數名稱，而沒有外部類別的物件變數名稱。而雖然我們沒有外部類別的物件變數名稱，但記憶體中仍將產生外部類別的物件實體。
- 【觀念範例15-10】：外部程式存取內部類別及內部類別存取外部類別。
- 範例15-10：ch15\_10.java（隨書光碟myJava\ch15\ch15\_10.java）

72

## 15.5.1 成員內部類別



```
1  /* 檔名:ch15_10.java      功能:外部程式存取內部類別 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_10      //主類別
7  {
8      public static void main(String args[])
9      {
10         //宣告並產生外部類別物件outY
11         CMyOutClass.CMyInnerClass innerX =
12             (new CMyOutClass(10)).new CMyInnerClass(20);
13         CMyOutClass outY = new CMyOutClass(100);
14         CMyOutClass.CMyInnerClass innerY =
15             outY.new CMyInnerClass(200);
16         //這樣也可以產生內部類別的物件
17         innerX.innerShow("第17行的呼叫");
18         innerY.innerShow("第18行的呼叫");
19     }
20 }
21
```

73

## 15.5.1 成員內部類別



```
22 class CMyOutClass
23 {
24     private int outVar;
25     public CMyOutClass(){}
26     public CMyOutClass(int i){outVar=i;}
27     class CMyInnerClass //內部類別的定義
28     {
29         private int innerVar;
30         public CMyInnerClass(){}           //內部類別建構子
31         public CMyInnerClass(int i){innerVar=i;} //內部類別建構子
32         public void innerShow(String str)
33         {
34             //內部類別可讀取外部類別的成員
35             System.out.println(str +
36                 ",run內部類別函式 outVar=" + outVar);
37             System.out.println(str +
38                 ",run內部類別函式 innerVar=" + innerVar);
39         }
40     }
41 }
```

74

## 15.5.1 成員內部類別



### ● 執行結果：

```
第17行的呼叫,run內部類別函式 outVar=10
第17行的呼叫,run內部類別函式 innerVar=20
第18行的呼叫,run內部類別函式 outVar=100
第18行的呼叫,run內部類別函式 innerVar=200
```

### ● 範例說明：

- (1)第27~38行是內部類別CMyInnerClass的定義，它被包含在外部類別CMyOutClass定義之內。

75

## 15.5.1 成員內部類別



- (2)第35行，可以在內部類別直接存取外部類別的成員outVar，因為您可以將內部類別看作是外部類別的成員之一，而成員之間原本就能夠互相存取，並且不受private等級的限制。
- (3)第11行是宣告內部類別物件的方式之一。這樣的方式，並沒有外部類別物件名稱可以作其他的應用。
- (4)第13、15行是宣告內部類別物件的方式之二。它是先產生外部類別物件outY，然後再由外部類別物件outY產生內部類別物件。這樣的方式，不但有內部類別物件名稱可以使用，也有外部類別物件名稱outY可以作其他的應用。

76

## 15.5.1 成員內部類別



- 【觀念範例15-11】：外部類別存取內部類別成員。
- 範例15-11：ch15\_11.java (隨書光碟 myJava\ch15\ch15\_11.java)

```
1  /* 檔名:ch15_11.java      功能:外部類別存取內部類別成員 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_11      //主類別
7  {
8      public static void main(String args[])
9      {
10         CMyOutClass outY = new CMyOutClass(); //產生外部類別物件
11
12         outY.outShow("第12行的呼叫");
13         outY.changeInnerVar("第13行的呼叫");
14     }
15 }
16
```

77

## 15.5.1 成員內部類別



```
17 class CMyOutClass
18 {
19     class CMyInnerClass //內部類別的定義
20     {
21         public int innerVar; //內部類別的成員
22         public CMyInnerClass(){innerVar=20;} //內部類別建構子
23         public void innerShow() //內部類別的成員
24         {
25             System.out.println("run內部類別函式執行中 innerVar="
26                                 + innerVar);
27         }
28     }
29     public int outVar;
30     public CMyOutClass(){outVar=10;} //外部類別建構子
31     public void outShow(String str)
32     {
33         System.out.println(str + ",run外部類別函式 outVar=" + outVar);
34         //innerVar=0; //錯誤的敘述,因為沒有內部類別的物件實體
35     }
36 }
```

78

## 15.5.1 成員內部類別



```
36 public CMyInnerClass objInner;
37 public void changeInnerVar(String str)
38 {
39     objInner = new CMyInnerClass();
40     System.out.print(str);
41     System.out.println(",run外部類別函式,
42                     準備修改內部類別物件實體的資料");
43     objInner.innerVar=50;
44     objInner.innerShow();
45 }
46
```

### ●執行結果：

第12行的呼叫,run外部類別函式 outVar=10  
第13行的呼叫,run外部類別函式,準備修改內部類別物件實體的資料  
run內部類別函式執行中 innerVar=50

79

## 15.5.1 成員內部類別



### ●範例說明：

- (1)第19~27行是內部類別CMyInnerClass的定義。
- (2)第36行，宣告外部類別的一個成員objInner，它的資料型態是一個物件參考，未來將指向內部類別CMyInnerClass物件。不過初始值為null。
- (3)第12行呼叫時，在第34行不能存取內部類別的成員，因為當時並未產生任何的內部類別物件實體（objInner當時為null，尚未指向任何一個物件實體）。
- (4)第13行呼叫時，在changeInnerVar函式內產生了內部類別的實體，故可以透過objInner存取內部類別物件的成員（第43、44行）。

### ●在上面的兩個範例中，我們建立了三個觀念如下：

- (1)內部類別可以視為外部類別的成員，所以存取權限並不會被private所限制，且可以相互存取。

80



## 15.5.1 成員內部類別



- (2) 外部類別要存取內部類別的物件成員，必須已經產生了內部類別的物件實體。
- (3) 如果內部類別未被宣告為private等級，則一般類別的外部程式也可以將它視為類別來使用，只不過在產生內部類別物件實體時，必須包含產生外部類別物件實體與產生內部類別物件實體等兩個步驟。
- 針對第(2)點，我們可以將產生內部類別的物件實體放在外部類別的建構子中執行，如此可以確保每一個外部類別的函式都可以取用內部類別物件的成員。

81

## 15.5.1 成員內部類別



- 針對第(3)點，我們思考如何利用該特點，加強內部類別的封裝性。(內部)類別可以被宣告為private等級，如果我們將類別宣告為private等級，則一般類別不能取用該類別。但雖然內部類別被宣告為private等級，由於它仍是外部類別的成員之一，故外部類別仍可以取用它。
- 如此，對於一般類別而言，想要取用內部類別，就如同要存取其他private等級的成員一般，一定要透過外部類別的其他成員來完成，而無法直接取用它，這樣子一來，就加強了內部類別的封裝性。請見下一個範例。

82

## 15.5.1 成員內部類別



- 【觀念範例15-12】：將內部類別多加一層封裝。
- 範例15-12：ch15\_12.java (隨書光碟 myJava\ch15\ch15\_12.java)

```
1  /* 檔名:ch15_12.java      功能:將內部類別多加一層封裝 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_12      //主類別
7  {
8      public static void main(String args[])
9      {
10         CMyOutClass outX = new CMyOutClass();
11         //CMyOutClass.CMyInnerClass innerX = outX.new CMyInnerClass();
12
13         outX.runInnerMethod(); //將產生執行結果第一行的輸出
14         outX.changeInnerVar(); //將改變內部類別物件成員
15         outX.printInnerVar(); //將產生執行結果第二行的輸出
16         outX.runInnerMethod(); //將產生執行結果第三行的輸出
17     }
18 }
19
```

產生外部類別實體時，會執行建構子，在建構子中以敘述產生內部類別的物件實體

83

```
20 class CMyOutClass
21 {
22     private class CMyInnerClass
23     {
24         public int innerVar;
25         public CMyInnerClass(){innerVar=20;}
26         public void innerShow()
27         {
28             System.out.println("內部類別函式執行中 innerVar=" +
29                                     innerVar);
30         }
31     }
32     public CMyInnerClass objInner; //以內部類別作為資料型態宣告物件變數
33     public CMyOutClass()
34     {
35         objInner = new CMyInnerClass();
36     }
37     public void printInnerVar()
38     {
39         System.out.println("外部類別函式執行中 innerVar=" +
40                                 objInner.innerVar);
41     }
42     public void changeInnerVar()
43     {
44         //因為有內部類別的物件objInner的實體，所以可讀取內部類別的成員
45         objInner.innerVar=50;
46     }
47 }
```

private等級的內部類別定義

在外部類別的建構子中，先產生內部類別的物件實體，則只要產生外部類別的物件實體，就一定包含一個內部類別的物件實體

84

## 15.5.1 成員內部類別



```
47 public void runInnerMethod()
48 {
49     //因為有內部類別的物件objInner的實體,所以可執行內部類別的成員
50     objInner.innerShow();//正確的敘述
51 }
52 }
```

### ●執行結果：

內部類別函式執行中 innerVar=20  
外部類別函式執行中 innerVar=50  
內部類別函式執行中 innerVar=50

85

## 15.5.1 成員內部類別



### ●範例說明：

- (1)第22~30行是內部類別CMyInnerClass的定義。它被宣告為private等級，由於仍是外部類別的一個成員，故外部類別可正確存取它。但一般類別則無法存取它，例如第11行是錯誤的敘述。雖然內部類別的成員全部都被宣告為public等級，但由於一般類別無法產生內部類別的物件，故仍無法讀取這些成員。
- (2)第32~51行，外部類別的其他成員，由於都宣告為public，所以一般類別可以存取。
- (3)第33~36行，為了要讓外部類別的成員函式可以正確存取內部類別物件的成員，故我們在外部的建構子中，先產生內部類別的物件實體。由於建構子一定會先被執行，故所有的成員函式都有內部類別的物件實體可以取用。

86

## 15.5.1 成員內部類別



- (4)第13~16行，由於內部類別被封裝起來了，故我們要存取內部類別，只能夠過外部類別的public函式來完成，如此就等於進一步對於內部類別進行封裝了。下圖是本範例封裝示意圖。

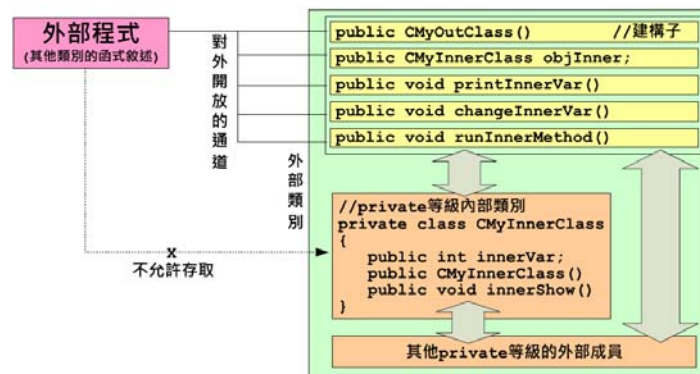


圖15-6 private內部類別封裝示意圖

87

## 15.5.1 成員內部類別



### ●static外部類別成員函式存取內部類別的解決方案

- method若被宣告為static，就無法存取非static的成員，而如果一個method被宣告為static（例如main函式），要如何存取內部類別的成員呢？
- 方法有兩個，第一是將內部類別宣告為static（加上修飾字），但由於我們並未介紹何謂static class，故採用第二個方法
- 也就是在外部類別的建構子內建立內部類別的物件（因為建構子並非static method，故不受影響），然後在static method中，產生一個外部類別物件（產生時會自動執行外部類別的建構子），並透過「外部類別物件.內部類別物件.成員」來存取即可，可請見下列範例。

88

## 15.5.1 成員內部類別



- 【觀念及實用範例15-13】：透過外部類別的建構子存取內部類別的成員以解決static的問題。
- 範例15-13：ch15\_13.java（隨書光碟 myJava\ch15\ch15\_13.java）

```
1  /* 檔名:ch15_13.java 功能:透過外部類別的建構子存取內部類別的成員 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_13      //主類別
7  {
8      public static void main(String args[])
9      {
10         ch15_13 obj = new ch15_13();
11         obj.objInner.innerVar=50;    //存取內部類別物件實體成員
12         obj.objInner.innerShow();    //呼叫內部類別物件實體成員
13     }
14     public ch15_13()
15     {
16         objInner = new MyInnerClass();
17     }
```

建立主類別的物件實體，會自動呼叫建構子

在建構子內產生內部類別的物件實體，而建構子不是static

89

## 15.5.1 成員內部類別



```
18  public MyInnerClass objInner; //objInner外部類別的一個欄位
19  class MyInnerClass
20  {
21      public int innerVar;
22      public MyInnerClass(){innerVar=20;}
23      public void innerShow()
24      {
25          System.out.println("內部類別函式執行中 innerVar=" +
26                               innerVar);
27      }
28  }
```

內部類別不必被宣告為static

- 執行結果：

內部類別函式執行中 innerVar=50

90

## 15.5.1 成員內部類別



- 範例說明：

- 在static main()內建立主類別的物件實體，此時會呼叫主類別的建構子，而在建構子內建立內部類別的物件實體。所以當外部類別物件實體obj被產生後，內部類別物件實體objInner也已被產生，此時，obj物件會有一個欄位稱為objInner，該欄位也是一個物件，所以也可以透過傳統語法存取物件成員。故只要透過兩層的「.」即可完成我們的需求（如第11、12行）。
- 在許多物件中包含更小物件的設計中（小物件為大物件的欄位），也時常看到這樣的語法。

91

## 15.5.2 區域內部類別 (local inner class)



- 區域內部類別

- 相對於成員內部類別定義於外部類別之內，並將之視為外部類別的一個成員；「區域內部類別」則是定義於某一個方法(method)之內，並將之視為方法內的一個區域類別，故生命週期與可見度僅限於該方法內。
- 區域內部類別的實際應用極少（後面要介紹的匿名類別比較多），以下我們僅透過一個範例來解說。



- 【觀念範例15-14】：區域內部類別的練習。

- 範例15-14：ch15\_14.java（隨書光碟 myJava\ch15\ch15\_14.java）

92



## 15.5.2 區域內部類別



```
1  /* 檔名:ch15_14.java      功能:區域內部類別 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_14      //主類別
7  {
8      public static void main(String args[])
9      {
10         CMYOutClass outX = new CMYOutClass(); //宣告並產生外部類別物件
11         outX.runOutClassMehtod1();
12         // CMYOutClass.CMYInnerClass innerX
13         //          = (new CMYOutClass()).new CMYInnerClass();
14     }
15 }
```

93

## 15.5.2 區域內部類別



```
16 class CMYOutClass
17 {
18     public void runOutClassMehtod1()
19     {
20         class CMYInnerClass
21         {
22             public int innerVar;
23             public CMYInnerClass(){innerVar=20;}
24             public void innerShow()
25             {
26                 System.out.println("區域內部類別函式執行中 innerVar="
27                                     + innerVar);
28             }
29         }
30         CMYInnerClass objInner = new CMYInnerClass();
31         System.out.println("外部類別函式執行中 innerVar=" +
32                             objInner.innerVar);
33
34         objInner.innerVar=50;
35         objInner.innerShow();
36         System.out.println("外部類別函式執行中 innerVar=" +
37                             objInner.innerVar);
38     }
39
40     //CMYInnerClass objInner2;
41     public void runOutClassMehtod2()
42     {
43         //CMYInnerClass objInner2;
44     }
45 }
```

定義區域內部類別

錯誤，不能宣告區域內部類別的物件變數

錯誤，不能宣告區域內部類別的物件變數

94

## 15.5.2 區域內部類別



- 執行結果：  
外部類別函式執行中 innerVar=20  
區域內部類別函式執行中 innerVar=50  
外部類別函式執行中 innerVar=50

- 範例說明：
  - (1)第20~28行的內部類別定義於runOutClassMehtod1函式內，所以是一個區域內部類別。
  - (2)第29~33行，在runOutClassMehtod1函式內可以存取區域內部類別。
  - (3)對於非同類別的程式而言，不能存取區域內部類別。例如第12行是不合法的語法。
  - (4)對於非同函式的程式而言，即使與該函式位於同一個類別，仍舊不能存取區域內部類別。例如第35、38行是不合法的語法。因為區域內部類別的可見度(scope)僅限於該函式內。

95

## 15.5.3 匿名類別(anonymous class)



- 「匿名類別」
- 完整名稱為匿名內部類別(anonymous inner class)，它代表的是一個沒有名字的內部類別。嚴格來說，它應該屬於一種「區域內部類別」，只不過它沒有名字。因此，它也必須定義於某個方法之內，如下語法：

```
Method名稱(參數串)
{
    類別名稱 物件名稱 = new 類別名稱(引數串)
    {
        ...匿名類別的實作區...
    };
    // .....其他Method的程式.....
}
```

定義(區域)匿名類別

其實是父類別的名稱

記得加上「;」

96

## 15.5.3 匿名類別



### 【語法說明】：

- 通常使用匿名類別都是為了要將某個類別補足相關的函式，或要實作某個介面的相關函式，甚至是改寫某個函式，但卻又不想讓這些新增與改寫的函式影響範圍超過該物件，換句話說，您可以把它看作是一種繼承或實作，只不過影響力只及於該物件而已。
- 事實上，上述的類別名稱，其實是匿名類別的父類別，如果沒有父類別，就使用Object當作父類別。以下我們透過範例來示範如何定義匿名類別。

### ● 【觀念範例15-15】：定義匿名類別。

### ● 範例15-15：ch15\_15.java（隨書光碟 myJava\ch15\ch15\_15.java）

97

## 15.5.3 匿名類別



```
1  /* 檔名:ch15_15.java      功能:匿名類別 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_15      //主類別
7  {
8      public static void main(String args[])
9      {
10         CB objX = new CB();
11         objX.runMehtod1();
12     }
13 }
14
15 class CA //類別定義
16 {
17     public int Var;
18     public CA(){Var=20;}
19     public void show1()
20     {
21         System.out.println("類別CA的show1執行中,Var=" + Var);
22     }
23     public void show2()
24     {
25         System.out.println("類別CA的show2執行中,Var=" + Var);
26     }
27 }
```

98

## 15.5.3 匿名類別



```
28
29 class CB
30 {
31     public void runMehtod1()
32     {
33         //匿名類別開始
34         CA obj1 = new CA()
35         {
36             public void show1()
37             {
38                 System.out.print("這是由匿名類別改寫的show1函式");
39                 System.out.println(",Var=" + Var);
40             }
41             public void show3()
42             {
43                 System.out.print("這是由匿名類別新增的show3函式");
44                 System.out.println(",Var=" + Var);
45             }
46         }; //匿名類別結束
47         obj1.Var=50;
48         obj1.show1();
49         obj1.show2();
50         //obj1.show3();
51     }
52 }
```

匿名類別的影響力僅及於obj1，而非CA類別的所有物件

在匿名類別內改寫show1()

在匿名類別內新增show3()

執行第36行

無法執行新增的show3函式，下一個範例解決此問題

99

## 15.5.3 匿名類別



```
53     CA obj2 = new CA();
54     System.out.println("-----");
55     obj2.Var=30;
56     obj2.show1();
57     obj2.show2();
58     //obj2.show3();
59 }
60 }
```

執行第19行

無法執行新增的show3函式

### ● 執行結果：

這是由匿名類別改寫的show1函式,Var=50  
類別CA的show2執行中,Var=50  
-----  
類別CA的show1執行中,Var=30  
類別CA的show2執行中,Var=30

第49、38、39行的輸出

第50、25行的輸出

第49、21行的輸出

100

## 15.5.3 匿名類別



### ● 範例說明：

- (1)第15~27行，這是一個類別CA的定義，它包含了一個成員變數Var，以及兩個成員函式show1, show2。
- (2)第29~60行，這是一個類別CB的定義，其中包含一個runMethod1函式，我們在函式內定義了區域匿名類別（第34~46行）。
- (3)第34~46行的匿名類別，只會對物件obj1產生作用。在匿名類別中，則定義了兩個成員函式，show1與show3。事實上，CA是匿名類別的父類別。
- (4)obj1、obj2都是CA類別的物件。不過由於匿名類別改寫了show1函式，故第49行所執行的函式，是改寫後的show1函式。而匿名類別對於obj2並不產生作用，故第56行執行的仍是原本的show1函式。
- (5)第58行的obj2無法執行show3函式是很正常的，因為匿名類別不對obj2產生作用。但第51行的obj1也無法執行新增的show3函式。我們會在下一個範例解決這個問題。

101

## 15.5.3 匿名類別



### ● 執行匿名類別新增的函式

- 在上一個範例中，我們得知，如果原本類別中不存在該函式，則透過匿名類別新增的函式，也無法被執行。但如果修改一下語法，則可以執行匿名類別新增的函式，語法如下：

```
Method名稱(參數串)
{
    (
        定義(區域)匿名類別
        new 類別名稱 (引數串)
        {
            ...匿名類別的實作區...
            ...新增的函式宣告...
        }
    )
    .新增的函式名稱;
    .....其他Method的程式.....
}
```

new會回傳一個物件參考，外面用()包裝起來，就可以用.來執行新增的函式，正如同(obj).method;

記得加上「;」

102

## 15.5.3 匿名類別



### ● 【語法說明】：

- 我們可以取消原語法中接收的物件名稱，此時，整個匿名類別由於new運算子仍會回傳一個可視為隱藏的物件參考，故透過該物件參考就可以執行新增的函式。不過要記得將全部的匿名類別定義使用()包裝起來，再執行新增的函式，並且原本的「;」也應該移動到敘述的最後。
- 若我們將上述語法中的不必要空白省略，則可以形成下列語法格式：

```
Method名稱(參數串)
{
    (new 類別名稱 (引數串)) //定義(區域)匿名類別
    {
        ...匿名類別的實作區...
        ...新增的函式宣告...
    }).新增的函式名稱;

    .....其他Method的程式.....
}
```

103

## 15.5.3 匿名類別



### ● 【語法說明】：

- 這個語法格式與上一個具有相同效力，差別在於我們將new往上移到(之後，也把)移到}之後，最後再把「.新增的函式名稱;」移到)之後。下列範例我們將採用如上的格式，如果讀者覺得怪怪的，可以自行修改成第一種格式語法。

- 【觀念範例15-16】：執行匿名類別新增的函式。

- 範例15-16：ch15\_16.java（隨書光碟myJava\ch15\ch15\_16.java）

104



## 15.5.3 匿名類別



```
1  /* 檔名:ch15_16.java      功能:執行匿名類別新增的函式 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_16      //主類別
7  {
8      public static void main(String args[])
9      {
10         CB objX = new CB();
11         objX.runMehtod1();
12     }
13 }
14
15 class CA //類別定義
16 {
17     public int Var;
18     public CA(){Var=20;}
19 }
20
```

105

## 15.5.3 匿名類別



```
20
21 class CB
22 {
23     public void runMehtod1()
24     {
25         ( new CA() //匿名類別開始
26         {
27             public void show()
28             {
29                 System.out.print("這是由匿名類別新增的show函式");
30                 System.out.println(",Var=" + Var);
31             }
32         }).show(); //執行新增的show函式
33     }
34 }
```

● 執行結果： 這是由匿名類別新增的show函式,Var=20

● 範例說明：

- 第25~32行是匿名類別的定義，並且執行新增的show函式，請注意在匿名類別定義的前後必須使用( )來包裝。

106

## 15.5.3 匿名類別



### ● 匿名類別的限制

- 匿名類別將會在視窗程式設計中，使用作為關閉視窗事件的處理，例如範例16-4。
- 而使用匿名類別還有許多限制，整理如下：
  - (1)在匿名類別內，可以讀取外部類別的成員。
  - (2)匿名類別內不能宣告static的成員。
  - (3)匿名類別只能讀取在原隸屬函式中被宣告為final的區域變數，無法讀取非final的區域變數。

107

## 15.5.3 匿名類別



● 【觀念範例15-17】：匿名類別的限制。

● 範例15-17：ch15\_17.java (隨書光碟 myJava\ch15\ch15\_17.java)

```
1  /* 檔名:ch15_17.java      功能:匿名類別的限制 */
2
3  package myJava.ch15;
4  import java.lang.*;
5
6  public class ch15_17      //主類別
7  {
8      public static void main(String args[])
9      {
10         CB objX = new CB();
11         objX.runMehtod1();
12     }
13 }
14
15 class CA //類別定義
16 {
17     public int Var;
18     public CA(){Var=20;}
19 }
20
```

108

## 15.5.3 匿名類別

```
21 class CB
22 {
23     public int Var1=5;
24
25     public void runMethod1()
26     {
27         int localVar1=100;        //非final變數
28         final int localVar2=100;  //final變數
29
30         ( new CA() //匿名類別開始
31         {
32             public int a=10;
33             //public static int b=10; 不能宣告static變數
34             public void show()
35             {
36                 System.out.println("這是由匿名類別新增的show函式");
37                 System.out.println("Var1=" + Var1); 不能讀取runMethod1函式
38                 System.out.println("a=" + a);        中被宣告為final的區域變
39                 //System.out.println(localVar1);      數
40                 System.out.println("localVar2=" + localVar2);
41             }
42         } ).show();    //執行新增的show函式
43     }
44 }
```

109

## 15.5.3 匿名類別

### ● 執行結果：

```
這是由匿名類別新增的show函式
Var1=5
a=10
localVar2=100
```

### ● 範例說明：

- (1) 第37行可讀取外部類別的成員Var1。
- (2) 第40行可讀取函式的final區域變數。第39行不可讀取函式的非final區域變數。



### Coding 偷撇步

由於巢狀類別的語法很複雜，因此能夠不使用就盡量不要使用巢狀類別。本書在此介紹是為了方便下一章要介紹的視窗事件處理。我們可能會使用內部類別實作一個事件傾聽者，如此一來，傾聽者可以直接存取視窗元件，而不必透過引數的傳遞，因為內部類別可以直接取用外部類別的成員。

110

## 15.6 本章回顧

- 在本章中，我們介紹了Java視窗程式的架構，它主要是依靠Frame與JFrame類別來完成。
  - 其中Frame為AWT所提供，JFrame則是Swing所提供。
- 設計視窗程式，可以設計一個視窗類別，只要繼承Frame或JFrame即可。
  - 如果不想額外設計類別，也可以直接透過new產生Frame或JFrame的實體來操作。

111

## 15.6 本章回顧

- 因為每一個元件都是一個物件，因此視窗程式的設計，是透過執行特定的方法設定屬性來完成。
  - 另一部分則為下一章要介紹的事件處理。
- 收納器是可以容納其他元件的一種特殊元件，以java.awt.Container類別為基礎的元件就是收納器，並且視窗本身就是一個收納器，因此可以容納其他元件。
  - 在Frame視窗中，我們將元件直接透過Frame實體的add()方法加入到Frame視窗中
  - 而在JFrame視窗中，我們則是將元件加入到JFrame的ContentPane中
    - 而取得JFrame的ContentPane可以透過getContentPane()方法來達成。

112

## 15.6 本章回顧



- 關於版面的管理，我們應該執行收納器元件的`setLayout()`方法來設定版面的管理
  - 如果想要手動設定各元件的位置與大小，則可以設定為`setLayout(null)`。
  - 如果要使用Java提供的版面管理員，則可產生版面管理員類別的實體，並將其參考作為引數傳入`setLayout()`方法
    - 在本章中，我們示範了三種簡單的版面管理員，包括`BorderLayout`、`FlowLayout`與`GridLayout`。

113

## 15.6 本章回顧



- 收納器中也可以容納其他的收納器，這些收納器可以視為內部的收納器，而內部的收納器也可以容納元件，因此，可以造就更多種類的版面編排方式，但通常更方便的方法是透過IDE直接來配置元件。
- 由於我們並不打算詳細介紹每一個視窗元件，只有在使用到某個視窗元件時才進行說明。欲了解Java的視窗元件外觀與對應的類別，可以連到下列網址，該網址中會顯示各種Swing元件的外觀。

● <http://download.oracle.com/javase/tutorial/uiswing/components/>

114

## 15.6 本章回顧



- 在本章的最後，我們補充說明了巢狀類別，也就是內部類別。內層的類別分為三種
  - 第一種是有名字的成員內部類別(member inner class)
  - 第二種是有名字的區域內部類別(local inner class)
  - 最後一種則是沒有名字的匿名類別(anonymous class)。
    - 其中，匿名類別的主要作用是作為補充某個類別未實作的部分，並且新增或改寫的函式內容之影響範圍只限制在某個物件，這種方式將會在下一章的事件處理中常常出現
    - 所以讀者務必理解範例15-16，才有助於繼續下一章的學習

115

## 本章結束



# Q&A討論時間

116