第 7 章
類別的進階認識

本章學習目標

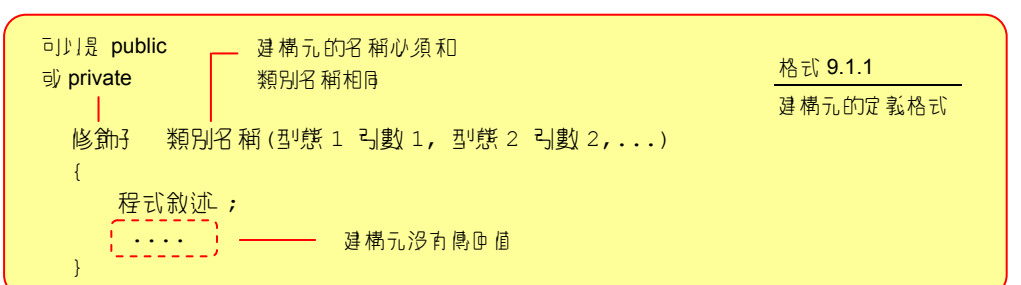
- 認識建構元與建構元的多載
- 認識「類別變數」與「類別函數」
- 認識類別型態的變數
- 學習利用 陣列來儲存物件
- 認識內部類別

9.1 建構元

在建立物件的同時，一併設定資料成員，其方法是利用「建構元」（constructor）。

9.1.1 建構元的基本認識

建構元可視為一種特殊的 method，其語法如下：



格式 9.1.1
建構元的定義格式

建構元的名稱必須與其所屬之類別的類別名稱相同。

以 CCircle 類別為例，如果想利用建構元來設定資料成員 radius 的值，可把 CCircle 類別的建構元撰寫如下：

```
01      public CCircle(double r)    // 定義建構元 CCircle()
02      {
03         radius=r;                // 設定資料成員 radius 的值
04      }
```

9.1.2 建構元的呼叫時機

在建立物件時，便會自動呼叫建構元，並執行建構元的內容，因此可利用它來做初始化（initialization）的設定。

下面的例子說明了建構元的使用：

```
01  // app9_1, 建構元的使用
02  class CCircle   // 定義類別 CCircle
03  {
04     private double pi=3.14;
05     private double radius;
06
07     public CCircle(double r)    // 定義建構元 CCircle()
08     {
09        radius=r;
10     }
11     public void show()
12     {
13       System.out.println("radius="+radius+", area="+pi*radius*radius);
14     }
15  }
16  public class app9_1
17  {
18     public static void main(String args[])
19     {
20        CCircle cir1=new CCircle(4.0);// 建立物件並呼叫 CCircle()建構元
21        cir1.show();
22     }
23  }
```

```
/* app9_1 OUTPUT---

radius=4.0, area=50.24
----------------------*/
```

9.1.3 建構元的多載

建構元也可以多載，如下面的範例：

```
01  // app9_2,建構元的多載
02  class CCircle   // 定義類別 CCircle
03  {
04     private String color;
05     private double pi=3.14;
06     private double radius;
07
08     public CCircle()        // 沒有引數的建構元
09     {
10        System.out.println("constructor CCircle() called");
11        radius=1.0;
12        color="Green";
13     }
14     public CCircle(String str, double r)   // 有兩個引數的建構元
15     {
16        System.out.println("constructor CCircle(String,double) called");
17        color=str;
18        radius=r;
19     }
```

```
20      public void show()
21      {
22         System.out.println("color="+color+", Radius="+radius);
23         System.out.println("area="+pi*radius*radius);
24      }
25   }
26   public class app9_2
27   {
28      public static void main(String args[])
29      {
30         CCircle cir1=new CCircle();           // 呼叫沒有引數的建構元
31         cir1.show();
32
33         CCircle cir2=new CCircle("Blue",4.0); // 呼叫有引數的建構元
34         cir2.show();
35      }
36   }
```

```
/* app9_2 OUTPUT------------------------
constructor CCircle() called
color=Green, Radius=1.0
area=3.14
constructor CCircle(String,double) called
color=Blue, Radius=4.0
area=50.24
-----------------------------------------*/
```

9.1.4 從某一建構元呼叫另一建構元

下面的例子是在沒有引數的 CCircle() 建構元裡，利用 this() 來呼叫有引數的建構元：

```
01  // app9_3, 從某一建構元呼叫另一建構元
02  class CCircle   // 定義類別 CCircle
03  {
04     private String color;
05     private double pi=3.14;
06     private double radius;
07
08     public CCircle()              // 沒有引數的建構元
09     {
10        this("Green",1.0);         // 此行會呼叫第 13 行的建構元
11        System.out.println("constructor CCircle() called");
12     }
13     public CCircle(String str, double r)    // 有引數的建構元
14     {
15        System.out.println("constructor CCircle(String,double) called");
16        color=str;
17        radius=r;
18     }
```

```
19      public void show()
20      {
21         System.out.println("color="+color+", Radius="+radius);
22         System.out.println("area="+pi*radius*radius);
23      }
24   }
25   public class app9_3
26   {
27      public static void main(String args[])
28      {
29         CCircle cir1=new CCircle();
30         cir1.show();
31      }
32   }
```

```
/* app9_3 OUTPUT------------------------
constructor CCircle(String,double) called
constructor CCircle() called
color=Green, Radius=1.0
area=3.14
----------------------------------------*/
```

- ✓ 於某一建構元呼叫另一建構元時，必須以 this() 來呼叫，而不能以建構元直接呼叫。

- ✓ this() 必須寫在建構元內第一行的位置。

- ✓ 呼叫沒有引數的建構元時，在 this() 的括號裡不必填上任何引數：

```
this();      // 呼叫沒有引數的建構元
```

9.1.5 建構元的公有與私有

若建構元為 public，則可以在程式的任何地方被呼叫。

如果建構元被設成 private，則無法在該建構元所在的類別以外的地方被呼叫。

來看看下面的範例：

```
01  // app9_4, 公有與私有建構元的比較
02  class CCircle   // 定義類別 CCircle
03  {
04     private String color;
05     private double pi=3.14;
06     private double radius;
07
08     private CCircle()     // 私有建構元
09     {
10        System.out.println("private constructor called");
11     }
12     public CCircle(String str, double r) // 公有建構元
13     {
14        this();
15        color=str;
16        radius=r;
17     }
18     public void show()
19     {
20        System.out.println("color="+color+", Radius="+radius);
21        System.out.println("area="+pi*radius*radius);
22     }
```

```
23   }
24   public class app9_4
25   {
26      public static void main(String args[])
27      {
28         CCircle cir1=new CCircle("Blue",1.0);
29         cir1.show();
30      }
31   }
```

```
/* app9_4 OUTPUT---------
private constructor called
color=Blue, Radius=1.0
area=3.14
-------------------------*/
```

9.1.6 建構元的省略

如果省略建構元，Java 會呼叫預設的建構元（default constructor）。

預設建構元的格式如下：

```
public CCircle()    // 預設的建構元
{

}
```

格式 9.1.2

預設的建構元。如果沒有寫出定義好建構元，則 Java 會使用此一版本的建構元

如果自行撰寫了建構元，無論是否有引數，則 Java 會假設已備妥好所有的建構元，不會再提供預設的建構元。

9.2 類別變數與類別函數

9.2.1 實例變數與實例函數

app9_5是一個很簡單的程式，可以認識實例變數與實例函數。

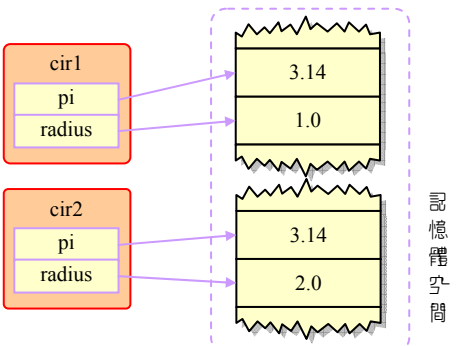
```
01  // app9_5, 簡單的範例,實例變數與實例函數
02  class CCircle   // 定義類別 CCircle
03  {
04     private double pi=3.14;
05     private double radius;
06
07     public CCircle(double r)   // CCircle()建構元
08     {
09        radius=r;
10     }
11     public void show()
12     {
13        System.out.println("area="+pi*radius*radius);
14     }
15  }
16  public class app9_5
17  {
```

```
18      public static void main(String args[])
19      {
20         CCircle cir1=new CCircle(1.0);
21         cir1.show();          // show()必須透過物件來呼叫
22         CCircle cir2=new CCircle(2.0);
23         cir2.show();          // show()必須透過物件來呼叫
24      }
25   }
```

```
/* app9_5 OUTPUT--
area=3.14
area=12.56
--------------------*/
```

實例變數

物件各自擁有儲存資料成員的記憶體空間，不與其它物件共用：



實例函數

必須透過物件來呼叫的函數稱為實例函數：

```
CCircle cir1=new CCircle(1.0);      // 建立物件 cir1
cir1.show();                        // 由物件 cir1 呼叫 show() method
CCircle cir2=new CCircle(2.0);      // 建立物件 cir2
cir2.show();                        // 由物件 cir2 呼叫 show() method
```

9.2.2 類別變數(class variable)

「實例變數」是各別物件所有，彼此之間不能共享。

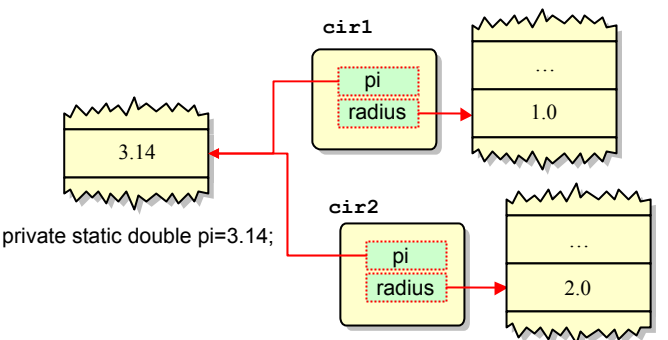
「類別變數」是由所有的物件共享。

如要把變數宣告為「類別變數」，必須在變數之前加上 static 修飾子。

假設 CCircle 類別裡的變數 pi，想要改為「類別變數」，可將它宣告成：

```
private static double pi=3.14;  // 將 pi 宣告為「類別變數」
```

下圖是把 pi 宣告成 static 之後，變數與記憶體之間的配置關係：



下面的程式碼是類別變數的範例：

```
01  // app9_6,「類別變數」的使用
02  class CCircle   // 定義類別 CCircle
03  {
04     private static int num=0;        // 宣告 num 為「類別變數」
05     private static double pi=3.14;  // 宣告 pi 為「類別變數」
06     private double radius;
07
08     public CCircle()              // 沒有引數的 CCircle()建構元
09     {
10        this(1.0);                 // 呼叫第 12 行的建構元，並傳入 1.0
11     }
12     public CCircle(double r)     // 有一個引數的 CCircle()建構元
13     {
14        radius=r;
15        num++;    // 當此建構元被呼叫時，num 便加 1
16     }
17     public void show()
18     {
19        System.out.println("area="+pi*radius*radius);
20     }
21     public void count()     // count()，用來顯示目前物件建立的個數
22     {
23        System.out.println(num+" object(s) created");
```

```
24      }
25   }
26   public class app9_6
27   {
28      public static void main(String args[])
29      {
30         CCircle cir1=new CCircle();       // 呼叫第 8 行的建構元
31         cir1.count();      // 用 cir1 物件呼叫 count() method
32         CCircle cir2=new CCircle(2.0);    // 呼叫第 12 行的建構元
33         CCircle cir3=new CCircle(4.3);    // 呼叫第 12 行的建構元
34         cir1.count();      // 用 cir1 物件呼叫 count() method
35         cir2.count();      // 改用 cir2 物件呼叫 count() method
36         cir3.count();      // 改用 cir3 物件呼叫 count() method
37      }
38   }
```

```
/* app9_6 OUTPUT---

1 object(s) created
3 object(s) created
3 object(s) created
3 object(s) created
---------------------*/
```

9.2.3 類別函數

若將 method 定義成類別函數，則可以直接由類別來呼叫：

```
01  // app9_7,「類別變數」的使用
02  class CCircle   // 定義類別 CCircle
03  {
04     private static int num=0;        // 宣告 num 為「類別變數」
05     private static double pi=3.14;  // 宣告 pi 為「類別變數」
06     private double radius;
07
08     public CCircle()              // 沒有引數的 CCircle()建構元
09     {
10        this(1.0);                 // 呼叫第 12 行的建構元，並傳入 1.0
11     }
12     public CCircle(double r)     // 有一個引數的 CCircle()建構元
13     {
14        radius=r;
15        num++;            // 當此建構元被呼叫時，num 便加 1
16     }
17     public void show()
18     {
19        System.out.println("area="+pi*radius*radius);
20     }
```

```
21      public static void count()  // count()，用來顯示目前物件建立的個數
22      {
23         System.out.println(num+" object(s) created");
24      }
25   }

26   public class app9_7
27   {
28      public static void main(String args[])
29      {
30         CCircle.count();     // 用 CCircle 類別呼叫 count() method
31         CCircle cir1=new CCircle();        // 呼叫第 8 行的建構元
32         CCircle.count();     // 用 CCircle 類別呼叫 count() method
33         CCircle cir2=new CCircle(2.0);     // 呼叫第 12 行的建構元
34         CCircle cir3=new CCircle(4.3);     // 呼叫第 12 行的建構元
35         cir3.count();        // 用 cir3 物件呼叫 count() method
36      }
37   }

/* app9_7 OUTPUT----

0 object(s) created
1 object(s) created
3 object(s) created
----------------------*/
```

9.2.4 「類別函數」使用的限制

「類別函數」無法存取「實例變數」與「實例函數」

如果在 app9_7 中撰寫如下的程式碼：

```
public static void count()
{
   System.out.println(num+" object(s) created");
   System.out.println("radius="+radius);  // 錯誤
   show();          // 錯誤
}
```

編譯時將發生錯誤。

「類別函數」內部不能使用 this 關鍵字

下面的程式碼是錯誤的：

```
public static void count()
{
    // 錯誤，不可使用 this
    System.out.println(this.num+" object(s) created");
    System.out.println("radius="+radius);
}
```

9.3 類別型態的變數

由類別宣告而得的變數，稱之為「類別型態的變數」，它是屬於「非基本型態的變數」的一種。

下列的程式碼則是宣告了 cir1 為 CCircle 類別型態的變數：

```
CCircle cir1;
cir1=new CCircle();
```

9.3.1 設值給類別型態的變數

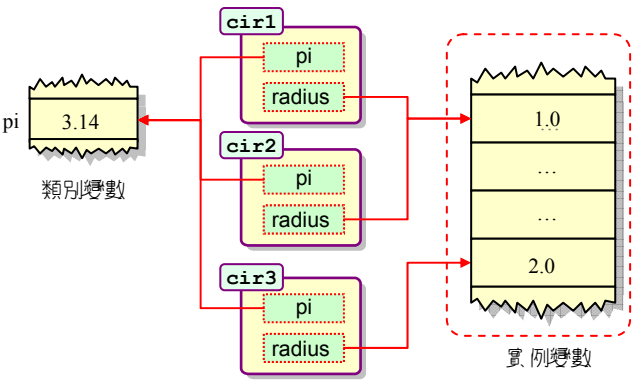
即使沒有用 new 產生新的物件，依然可對類別型態的變數設值：

```
01  // app9_8, 設值給類別型態的變數
02  class CCircle   // 定義類別 CCircle
03  {
04     private static double pi=3.14;
05     private double radius;
06
07     public CCircle(double r)
08     {
09        radius=r;
10     }
11     public void show()
12     {
13        System.out.println("area="+pi*radius*radius);
14     }
15  }
16  public class app9_8
17  {
18     public static void main(String args[])
19     {
20        CCircle cir1,cir2;       // 宣告 cir1,cir2 為類別型態的變數
```

```
21          cir1=new CCircle(1.0);    // 建立新的物件，並將 cir1 指向它
22          cir1.show();
23
24          cir2=cir1;  // 將 cir1 設給 cir2，此時這兩個變數所指向的內容均相等
25          cir2.show();
26
27          CCircle cir3=new CCircle(2.0);// 建立新的物件，並將 cir3 指向它
28          cir3.show();
29      }
30  }
```

```
/* app9_8 OUTPUT--
area=3.14
area=3.14
area=12.56
--------------------*/
```

透過 24 行的設定，即可將二個不同名稱的變數指向同一個物件：



透過其中一個變數對物件做更動，另一變數所指向之物件內容也會隨著更改，下面的例子說明了其中的道理：

```
01  // app9_9, 類別型態之變數的應用
02  class CCircle   // 定義類別 CCircle
03  {
04     private static double pi=3.14;
05     private double radius;
06
07     public CCircle(double r) // CCircle 建構元
08     {
09        radius=r;
10     }
11     public void setRadius(double r)
12     {
13        radius=r;          // 設定 radius 成員的值
14     }
15     public void show()
16     {
17        System.out.println("area="+pi*radius*radius);
18     }
19  }
20  public class app9_9
```

```
21  {
22     public static void main(String args[])
23     {
24        CCircle cir1,cir2;
25        cir1=new CCircle(1.0);
26        cir1.show();
27
28        cir2=cir1; // 將 cir1 設給 cir2，此時這兩個變數所指向的內容均相等
29        cir2.setRadius(2.0);  // 將 cir2 物件的半徑設為 2.0
30        cir1.show();
31     }
32  }
```

```
/* app9_9 OUTPUT--

area=3.14
area=12.56
--------------------*/
```

9.3.2 以類別型態的變數傳遞引數

想撰寫 compare() method，用來比較呼叫物件 cir1 與 compare() 裡的引數 cir2 的資料成員是否完全相同，可用下面的敘述來完成：

```
cir1.compare(cir2);
```

compare() method 的定義必須以下面的格式來撰寫：

```
傳回值型態 compare(CCircle obj)
{
  ....
}
```

引數型態為 CCircle

格式 9.3.1
傳遞類別型態的變數

下面的範例設計了 compare() method，用來比較二個物件是否相等：

```
01  // app9_10, 傳遞類別型態的變數
02  class CCircle
03  {
04     private static double pi=3.14;
05     private double radius;
06
07     public CCircle(double r) // CCircle()建構元
08     {
09        radius=r;
10     }
11     public void compare(CCircle cir) // compare() method
12     {
13        if(this.radius==cir.radius)   // 判別物件的radius成員是否相等
14           System.out.println("radius are equal");
15        else
16           System.out.println("radius are not equal");
17     }
18  }
19  public class app9_10
20  {
21     public static void main(String args[])
22     {
```

```
23          CCircle cir1=new CCircle(1.0);
24          CCircle cir2=new CCircle(2.0);
25          cir1.compare(cir2);   // 比較 cir1 與 cir2 的 radius 是否相等
26      }
27  }

/* app9_10 OUTPUT----

radius are not equal
-----------------------*/
```

9.3.3 由 method 傳回類別型態的變數

以 compare() method 為例，如要傳回 CCircle 類別型態的變數，可利用下面的語法來撰寫：

傳回型態為 CCircle 類別的變數

```
CCircle compare( CCircle obj)
 {
   ....
 }
```

格式 9.3.2

由 method 傳回類別型態的變數

下面的範例以compare()比較物件半徑的大小，並傳回半徑較大的物件：

```
01  // app9_11, 由 method 傳回類別型態的變數
02  class CCircle   // 定義類別 CCircle
03  {
04     private static double pi=3.14;
05     private double radius;
06
07     public CCircle(double r)    // CCircle 建構元
08     {
09        radius=r;
10     }
11     public CCircle compare(CCircle cir) // Compare() method
12     {
13        if(this.radius>cir.radius)
14           return this;       // 傳回呼叫 compare() method 的物件
15        else
16           return cir;        // 傳回傳入 compare() method 的物件
17     }
18  }
19  public class app9_11
20  {
21     public static void main(String args[])
22     {
```

```
23        CCircle cir1=new CCircle(1.0);
24        CCircle cir2=new CCircle(2.0);
25        CCircle obj;
26
27        obj=cir1.compare(cir2);      // 呼叫 compare() method
28        if(cir1==obj)
29           System.out.println("radius of cir1 is larger");
30        else
31           System.out.println("radius of cir2 is larger");
32     }
33  }
```

```
/* app9_11 OUTPUT-------
radius of cir2 is larger
-------------------------*/
```

9.3.4 回收記憶體

如果此物件不再使用了，可收回被它所佔用的記憶體空間，如下面的程式碼片段：

```
01  class app
02  {
03     public static void main(String args[])
04     CCircle cir1=new cir1();   // 建立物件，並配置記憶體給它
05     ....
06     cir1=null;   // 將 cir1 指向 null,代表 cir1 已不再指向任何物件
07     ....
08  }
```

- 經設定為 null，該變數便不指向任何物件。

於下面的程式碼中，Java 的乾淨殘餘記憶體機制不會回收：

```
01  class app
02  {
03     public static void main(String args[])
04     CCircle cir1=new cir1();
05     CCircle cir2;
06     cir2=cir1;   // 設定 cir2 與 cir1 均指向同一個物件
07     ....
08     cir1=null;   // 將 cir1 指向 null,但 cir2 仍指向該物件,因此不會被回收
09  }
```

9.4 利用陣列來儲存物件

用陣列來存放物件，必須經過下面兩個步驟：

1. 宣告類別型態的陣列變數，並用 new 配置記憶空間給陣列。
2. 用 new 產生新的物件，並配置記憶空間給它。

如下列的語法：

```
01   CCircle cir[];
02   cir=new Circle[3];
```

宣告 CCircel 類別型態的陣列變數，並用 new 配置記憶空間

建立好陣列之後，便可把陣列元素指向由 CCircle 類別所建立的物件：

```
03   cir[0]=new CCircle();
04   cir[1]=new CCircle();
05   cir[2]=new CCircle();
```

用 new 建立新的物件，並配置記憶空間給它

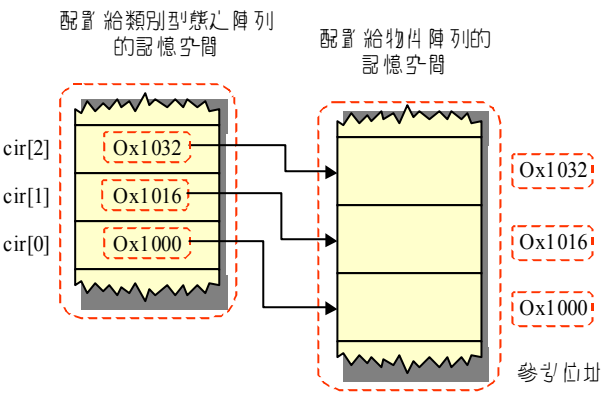
也可以把第 1 行與第 2 行合併成一行：

```
01   CCircle cir[]=new Circle[3];   // 建立物件陣列元素，並配置記憶空間
```

或者利用 for 迴圈來完成指向新建立物件之動作：

```
03    for(int i=0; i<cir.length; i++)
04    {
05        cir[i]=new CCircle();
06    }
```

下圖為類別型態的陣列與物件陣列的記憶空間配置情形：



9.4.1 建立物件陣列的範例

下面的程式碼是物件陣列的使用範例：

```
01  // app9_12, 建立物件陣列
02  class CCircle   // 定義類別 CCircle
03  {
04     private static double pi=3.14;
05     private double radius;
06
07     public CCircle(double r)       // CCircle 建構元
08     {
09        radius=r;
10     }
11     public void show()
12     {
13        System.out.println("area="+pi*radius*radius);
14     }
15  }
```

```
16  public class app9_12
17  {
18     public static void main(String args[])
19     {
20        CCircle cir[];                    宣告類別型態的陣列，並用 new 配
21        cir=new CCircle[3];               置記憶空間
22        cir[0]=new CCircle(1.0);          用 new 產生新的物件，並配置給
23        cir[1]=new CCircle(4.0);          陣列元素
24        cir[2]=new CCircle(2.0);
25
26        cir[1].show();   // 利用物件 cir[1]呼叫 show() method
27        cir[2].show();   // 利用物件 cir[2]呼叫 show() method
28     }
29  }
```

```
/* app9_12 OUTPUT--

area=50.24
area=12.56
--------------------*/
```

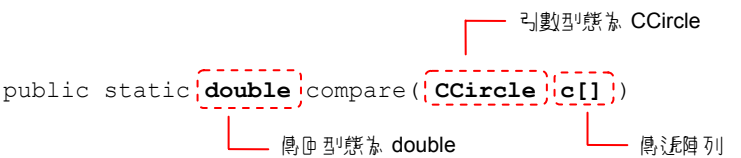
9.4.2 傳遞物件陣列到 method 裡

app9_13 是傳遞物件陣列的練習：

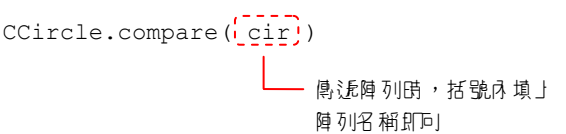
```
01  // app9_13, 傳遞物件陣列到 method
02  class CCircle   // 定義類別 CCircle
03  {
04     private static double pi=3.14;
05     private double radius;
06
07     public CCircle(double r)
08     {
09        radius=r;
10     }
11     public static double compare(CCircle c[])  // compare() method
12     {
13        double max=0.0;
14        for(int i=0;i<c.length;i++)
15           if(c[i].radius>max)
16              max=c[i].radius;
17        return max;
18     }
19  }
20
```

```
21  public class app9_13
22  {
23     public static void main(String args[])
24     {
25        CCircle cir[];
26        cir=new CCircle[3];
27        cir[0]=new CCircle(1.0);
28        cir[1]=new CCircle(4.0);
29        cir[2]=new CCircle(2.0);
30
31        System.out.println("Largest radius = "+CCircle.compare(cir));
32     }
33  }

/* app9_13 OUTPUT----

Largest radius = 4.0
----------------------*/
```

第 11 行的語法解說如下：



傳遞陣列時，compare()的括號內所填上的是陣列的名稱：



9.5 內部類別與匿名類別

類別 A 的內部再定義一個類別 B，此時類別 B 稱為內部類別（inner class），而類別 A 則稱為外部類別（outer class）。

下面列出了內部類別的定義格式：

格式 9.5.1
定義內部類別

```
修飾子 class 外部類別的名稱
{
    // 外部類別的成員
    修飾子 class 內部類別的名稱
    {
        // 內部類別的成員
    }
}
```

內部類別

外部類別

9.5.1 內部類別的撰寫

app9_14 是一個簡單的範例，稍後將以此範例做延伸來介紹內部類別的撰寫方式：

```
01  // app9_14, 類別的複習
02  class Caaa
03  {
04     int num;
05     void set_num(int n)
06     {
07        num=n;
08        System.out.println("num= "+ num);
09     }
10  }
11  public class app9_14
12  {
13     public static void main(String args[])
14     {
15        Caaa aa=new Caaa();
16        aa.set_num(5);
17     }
18  }
```

```
/* app9_14 OUTPUT---
num= 5
--------------------*/
```

將類別 Caaa 改寫為內部類別

下面的程式碼是內部類別的撰寫範例：

```
01  // app9_15, 內部類別的撰寫
02  public class app9_15
03  {
04     public static void main(String args[])
05     {
06        Caaa aa= new Caaa();
07        aa.set_num(5);
08     }
09
10     static class Caaa
11     {
12        int num;
13        void set_num(int n)
14        {
15           num=n;
16           System.out.println("num= "+ num);
17        }
18     }
19  }
```

外部類別

內部類別

```
/* app9_15 OUTPUT---

num= 5
---------------------*/
```

在外部類別的建構元裡建立內部類別的物件

如果不想把內部類別 Caaa 宣告成 static，但卻又要在 main()裡存取到它，利用下列的步驟便可以做到：

(1) 在外部類別的建構元裡建立內部類別的物件

(2) 在 main() 裡建立一個外部類別的物件

如下面的範例：

```
01  // app9_16, 在建構元裡建立內部類別的物件
02  public class app9_16
03  {
04    public app9_16()
05    {
06       Caaa aa= new Caaa();
07       aa.set_num(5);
08    }
09
10     public static void main(String args[])
```

外部類別的建構元

在外部類別的建構元裡建立內部類別的物件

```
11      {
12         app9_16 obj=new app9_16(); // 呼叫建構元 app9_16()建立外部類別的物件
13      }
14
15      class Caaa
16      {
17         int num;
18         void set_num(int n)
19         {
20            num=n;
21            System.out.println("num= "+ num);
22         }
23      }
24   }
```

內部類別

```
/* app9_16 OUTPUT---
num= 5
--------------------*/
```

9.5.2 匿名內部類別

建立匿名內部類別並存取成員的語法如下：

```
(
    new 類別名稱(引數)
    {
        傳回值型態 method名稱(引數1, 引數2,...,引數n)
        {
            method 敘述;
        }
    }
).method名稱(引數1, 引數2, ..., 引數n);
```

注意是小括號

格式 9.5.2

建立匿名內部類別，並執行所定義的 method

匿名內部類別可用來補足內部類別裡沒有定義到的 method：

```
01  // app9_17, 匿名內部類別
02  public class app9_17
03  {
04     public static void main(String args[])
05     {
06        (
07          new Caaa()  // 建立匿名內部類別 Caaa 的物件
08          {
09             void set_num(int n)
10             {
11                num=n;
12                System.out.println("num= "+ num);
13             }
14          }
15        ).set_num(5);    // 執行匿名內部類別裡所定義的method
16     }
17
18     static class Caaa  //內部類別 Caaa
19     {
20         int num;
21     }
22  }
```

建立匿名內部類別 Caaa 的物件

補足內部類別 Caaa 裡沒有定義到的 method

```
/* app9_17 OUTPUT---

num= 5
--------------------*/
```

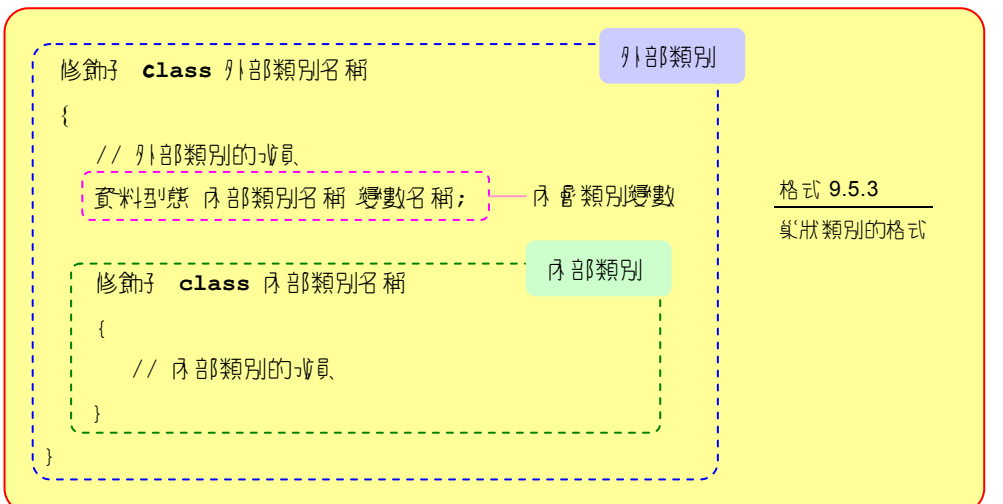
習慣上會把匿名內部類別的程式碼“擠”在短短的幾行，如下面的程式碼所示：

```
01  // app9_18, 匿名內部類別
02  public class app9_18
03  {
04     public static void main(String args[])
05     {
06       (new Caaa(){void set_num(int n){num=n;
07        System.out.println("num= "+ num);}}).set_num(5);
08     }
09     static class Caaa
10     {
11        int num;
12     }
13  }

/* app9_18 OUTPUT---
num= 5
--------------------*/
```

建立匿名內部類別的物件，並呼叫 set_num(5)

9.5.3 巢狀類別

在類別裡含有其它的類別，稱為巢狀類別（Nested Classes）：



格式 9.5.3
巢狀類別的格式

下面是定義巢狀類別的範例：

```
class CBox                  // 外部類別
{
   private int length;
   private int width;
   private int height;
   private CColor cr;

   class CColor             // 內部類別
   {
      String color;
   }
}
```

下面示範了如何利用外部類別的成員，存取、呼叫內部類別的成員：

```
01  // app9_19, 巢狀類別
02  class CBox                  // 外部類別
03  {
04     private int length;      // CBox 類別物件的長
05     private int width;       // CBox 類別物件的寬
06     private int height;      // CBox 類別物件的高
07     private CColor cr;       // CColor 類別的物件變數
08
09     public CBox(int l,int w,int h,String col) // CBox 建構元
10     {
11        length=l;
12        width=w;
13        height=h;
14        cr=new CColor(col); // 用 new 建立 CColor 物件
15     }
16     class CColor             // 內部類別
17     {
18        private String color;
19
20        public CColor(String clr)  // CColor 建構元
21        {
22           color=clr;
```

```
/* app9_19 OUTPUT---
length=2
width=3
height=4
color=Blue
--------------------*/
```

```
23        }
24        public void show_color()   // 顯示顏色
25        {
26           System.out.println("color="+color);
27        }
28     }
29     public void show()      // 外部類別 CBox 的成員函數
30     {
31        System.out.println("length="+length);
32        System.out.println("width="+width);
33        System.out.println("height="+height);
34        cr.show_color();
35     // System.out.println("color="+cr.color);
36     }
37  }
38
39  public class app9_19
40  {
41     public static void main(String args[])
42     {
43        CBox box=new CBox(2,3,4,"Blue");
44        box.show();
45     }
46  }
```

```
/* app9_19 OUTPUT---

length=2
width=3
height=4
color=Blue
--------------------*/
```

巢狀類別在使用上有如下的特點：

(1) 當巢狀類別宣告成public時，其內部類別也擁有public的權限。

(2) 外部類別的成員可以存取、呼叫內部類別裡的成員，不過亦用，不受private的限制。


-The End-