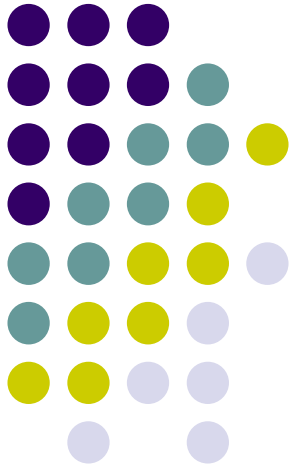
第九章
類別的進階認識



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

建構元的基本認識

- 建構元（constructor）是幫助新建立的物件**設定初值**
- 建構元的名稱必須與其所屬之類別的**類別名稱**相同
- 建構元可視為一種特殊的method，其語法如下：

建構元的定義格式

可以是public或private

建構元的名稱必須和類別名稱相同

```
修飾子   類別名稱(型態1 引數1, 型態2 引數2,...)
{
    程式敘述 ;
    ....
}
```

建構元沒有傳回值

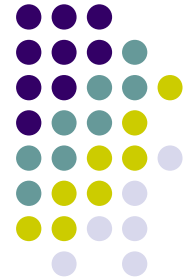
建構元的呼叫時機

- 一般的method
  - 在需要用到時才呼叫
- 建構元
  - 在建立物件時，便會**自動呼叫**，並執行建構元的內容
  - **建構元**可對物件的**資料成員**做**初始化**的設定
  - 初始化（initialization）就是設定物件的初值

建構元的使用範例

```
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
-----------------------*/
```



建構元的多載 (1/2)

- 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        color="Green";
12        radius=1.0;
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     }
```

```
/* 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
----------------------------------------*/
```

建構元的多載 (2/2)

```
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
----------------------------------------*/
```

從建構元呼叫另一建構元 (1/3)

- 從某建構元呼叫另一建構元，是透過this() 來呼叫 ：

```
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     }
```

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

把color設為 "Green"，radius設為1.0，必須以this()呼叫

從建構元呼叫另一建構元 (2/3)

```
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
----------------------------------------*/
```

從建構元呼叫另一建構元 (3/3)

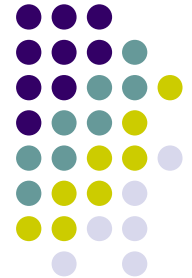
- 於某建構元呼叫另一建構元時，必須以this() 來呼叫
  - 例如，若把第10行改寫為：

```
CCircle("Green",1.0);        // 錯誤的建構元呼叫
```

編譯時會出現如下的錯誤訊息：

```
cannot find symbol
symbol  : method CCircle  (java.lang.String,double)
location: class CCircle
     CCircle("Green",1.0);
```

- this() 必須寫在**建構元**內**第一行**的位置



建構元的公有與私有 (1/3)

- 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
- 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     }
```

```
/* app9_4 OUTPUT----------

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

建構元的公有與私有 (2/3)

```
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
-------------------------*/
```

建構元的公有與私有 (3/3)

- 如果把第28行的敘述改為：

```
CCircle cir1=new CCircle();      // 呼叫 private 的建構元 CCircle()
```

  - 將會得到下列的錯誤訊息：

```
cannot find symbol
symbol  : constructor CCircle()
location: class CCircle
     CCircle cir1=new CCircle();
```

  - 這是因為private的建構元**無法**在**類別外部被呼叫**
  - 用private關鍵字來**保護建構元**，如此可對建構元的**存取設限**

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）
  - 預設的建構元是沒有任何引數的建構元，格式如下：

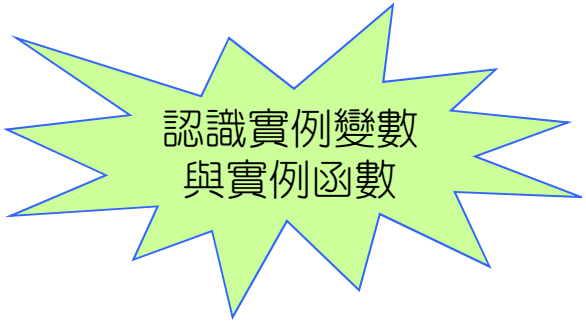
預設建構元的格式

```
public CCircle()
{


}
```

- 如果自行撰寫建構元，無論是否有引數，則Java會假設**已備妥**所有的建構元，**不會**再提供預設的建構元

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  }
```



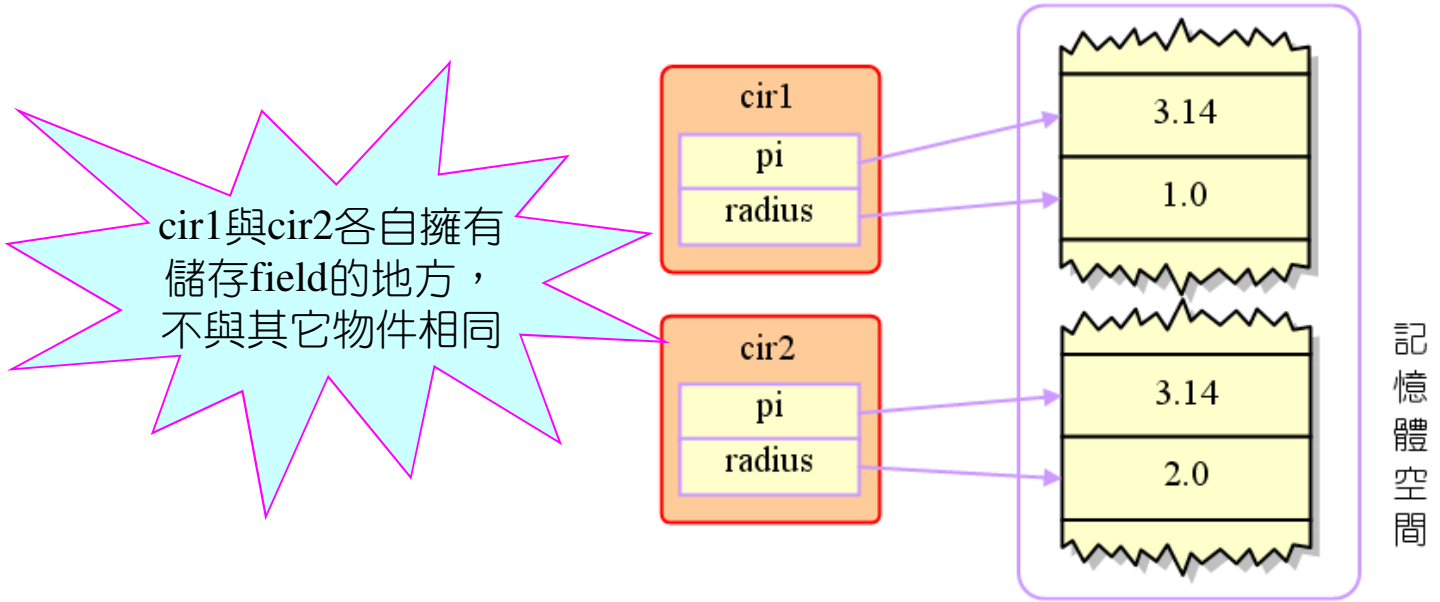

```
/* app9_5 OUTPUT---

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

實例變數

- 各自獨立且存於不同的記憶體之變數，稱為

  「**實例變數**」 (instance variable)

  - 物件擁有自己儲存資料的記憶體空間，不與其它物件共用：



實例函數

- 實例函數 (instance method)：
  - 必須先建立物件，再利用物件來呼叫的method

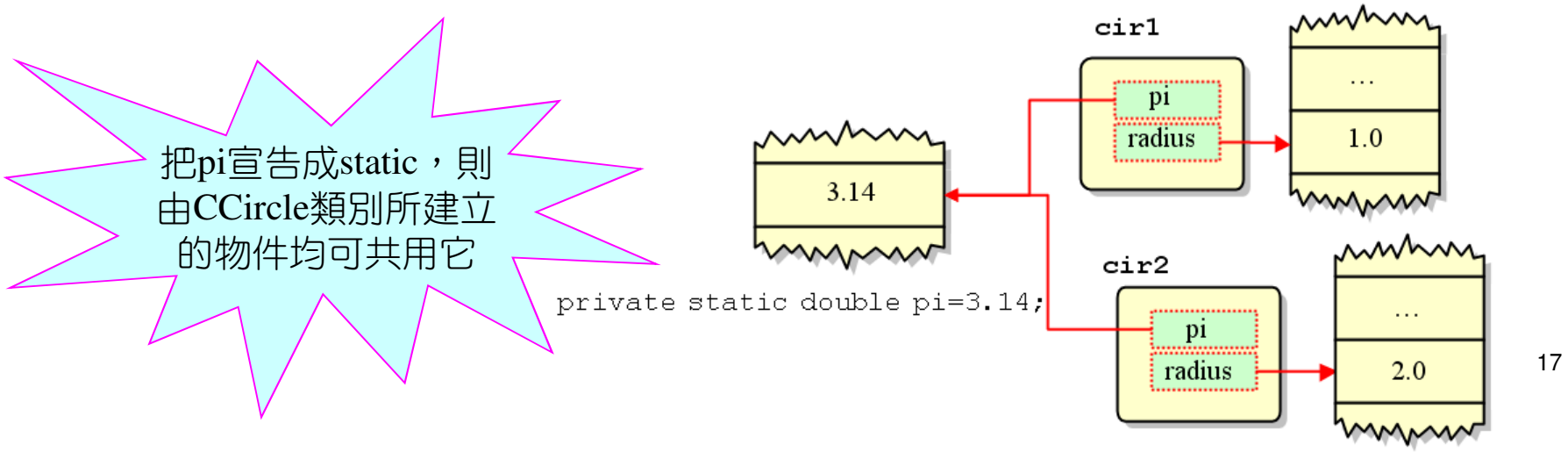
```
CCircle cir1=new CCircle(1.0);     // 建立物件 cir1
cir1.show();                       // 由物件 cir1 呼叫 show() method
CCircle cir2=new CCircle(2.0);     // 建立物件 cir2
cir2.show();                       // 由物件 cir2 呼叫 show() method
```

show() method必須透過物件cir1或cir2呼叫

類別變數

- 「**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**
- 要把變數宣告為「**類別變數**」，必須在變數前加上static修飾子，如下面的範例：

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



類別變數的範例 (1/2)

- 下面的程式碼是類別變數的範例：

```
/* app9_6 OUTPUT-----
1 object(s) created
3 object(s) created
3 object(s) created
3 object(s) created
---------------------*/
```

```
01  // app9_6,「類別變數」的使用
02  class CCircle                          // 定義類別 CCircle
03  {
04     private static int count=0;         // 宣告 count 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        count ++;                        // 當此建構元被呼叫時，count 便加 1
16     }
17     public void show()
18     {
19        System.out.println("area="+pi*radius*radius);
20     }
```

透過this() 呼叫定義在12~16行的建構元

用來計算物件的數目，count宣告為static，它由所有的物件所共用

類別變數的範例 (2/2)

```
21      public void show_count()    // show_count(),顯示目前物件建立的個數
22      {
23         System.out.println(count+" 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.show_count();     // 用 cir1 物件呼叫 show_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.show_count();     // 用 cir1 物件呼叫 show_count() method
35         cir2.show_count();     // 改用 cir2 物件呼叫 show_count() method
36         cir3.show_count();     // 改用 cir3 物件呼叫 show_count() method
37      }
38   }
```

均是透過物件來呼叫method

```
/* app9_6 OUTPUT-----

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

類別函數

- 若將method定義成類別函數，則可以直接由類別呼叫
  - 要定義成類別函數，在method之前加上static修飾子即可：

```
public static void show_count()     // 將 show_conut()宣告成「類別函數」
{
    System.out.println(count+" object(s) created");
}
```

  - 使用時直接用類別呼叫：

```
CCircle.show_count();               // 直接用 CCircle 類別呼叫「類別函數」
```

類別函數的使用 (1/2)

```
01  // app9_7,「類別函數」的使用
02  class CCircle                          // 定義類別 CCircle
03  {
04     private static int count=0;         // 宣告 count 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        count++;                         // 當此建構元被呼叫時，count 便加 1
16     }
17     public void show()
18     {
19        System.out.println("area="+pi*radius*radius);
20     }
21     public static void show_count()  // 顯示目前物件建立的個數
22     {
23        System.out.println(count+" object(s) created");
24     }
25  }
```

```
/* app9_7 OUTPUT------

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

類別函數的使用 (2/2)


```
26  public class app9_7
27  {
28     public static void main(String args[])
29     {
30        CCircle.show_count();       // 用 CCircle 類別呼叫 show_count()
31        CCircle cir1=new CCircle();       // 呼叫第 8 行的建構元
32        CCircle.show_count();       // 用 CCircle 類別呼叫 show_count()
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.show_count();          // 用 cir3 物件呼叫 show_count()
36     }
37  }
```

```
/* app9_7 OUTPUT------
0 object(s) created
1 object(s) created
3 object(s) created
----------------------*/
```

- 類別函數仍**可以**由**物件**呼叫，但**必須先建立物件**
- 類別函數可在**沒有物件**的情況下直接以類別呼叫

main() method與static修飾子

- main() method 也有一個static修飾子：

```
26  public class app9_7
27  {
28     public static void main(String args[])
29     {
..          .. ...
36     }
37  }
```

main() 之前加上static修飾子，使得main() 變成是一個「類別函數」

「類別函數」使用的限制 (1/2)

- **類別函數**無法存取**實例變數**或呼叫**實例函數**

如果在app9_7中寫如下的程式碼：

```
public static void show_count()
{
   System.out.println(count+" object(s) created");
   System.out.println("radius="+radius); // 錯誤，不可存取「實例變數」
   show();                              // 錯誤，不能呼叫「實例函數」
}
```

編譯時將產生如下的錯誤：

```
non-static variable radius cannot be referenced from a static context
     System.out.println("radius="+radius);

non-static method show() cannot be referenced from a static context
     show();
```

radius不是類別變數，無法由「類別函數」的內部呼叫

show() 為「實例函數」，也不能直接在「類別函數」內部呼叫

「類別函數」使用的限制 (2/2)

- 「**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 show_count()
{
   System.out.println(this.count+" object(s) created"); // 錯誤，不可使用 this
}
```

編譯後將得到下列的錯誤訊息：

```
non-static variable this cannot be referenced from a static context
     System.out.println(this.count+" object(s) created");
```

在「類別函數」內部
不能使用this關鍵字

變數的種類

- 變數分為
  - 基本型態的變數 與 非基本型態的變數
- 基本型態的變數
  - 是指由int、double等關鍵字所宣告而得的變數，如：

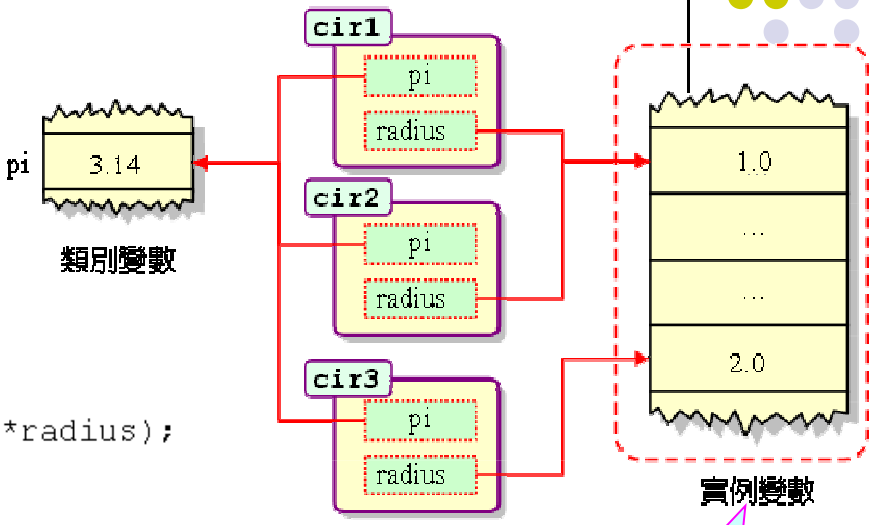
```
private double radius;
```

- 非基本型態的變數
  - 由類別宣告而得的變數是屬於「**類別型態的變數**」，如：

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

設值給類別型態的變數


```
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  }
```



設定cir2=cir1可將兩個類別型態的變數指向同一個物件

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

類別型態的變數另一例

```
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
--------------------*/
```

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

以類別型態的變數傳遞引數

- 下面敘述可用來比較兩個物件的**資料成員**是否相同：

```
cir1.compare(cir2);          // 比較物件 cir1 與 cir2 的資料成員是否相同
```

- compare() method的定義須以下面的格式來撰寫：

傳遞類別型態的變數之格式

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

引數型態為CCircle

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  }
```

不能寫成 if(this==cir)

```
/* app9_10 OUTPUT----
radius are not equal
--------------------*/
```

由method傳回類別型態的變數

- 以compare() method傳回CCircle類別型態的變數為例，由method傳回類別型態的變數之格式：

由method傳回類別型態的變數之格式

傳回型態為CCircle類別的變數

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

傳回類別型態的變數範例

左邊的範例會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
------------------------*/
```

回收記憶體 (1/2)

- Java有一套蒐集殘餘記憶體的機制，稱為

  垃圾回收機制（garbage collection）

  - 作法是把指向該**物件**的**變數值**設為null即可：

```
01   class app
02   {
03      public static void main(String args[])
04      CCircle cir1=new CCircle();   // 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   }
```

回收記憶體 (2/2)

- 若兩個變數指向同一個物件：
  - 如果把其中一個變數設為null，由於另一個變數還是指向它，蒐集殘餘記憶體機制不會回收，如：

```
01  class app
02  {
03     public static void main(String args[])
04     CCircle cir1=new CCircle();
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  }
```

類別型態的陣列 (1/2)

- 用陣列來存放物件的兩個步驟：

  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。

```
CCircle cir[];
cir=new CCircle[3];
```

合併成一行 `CCircle cir[]=new CCircle[3];`

宣告類別型態的陣列變數，並用 new 配置記憶體空間給陣列

- 建立好陣列之後，便可把陣列元素指向物件：

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

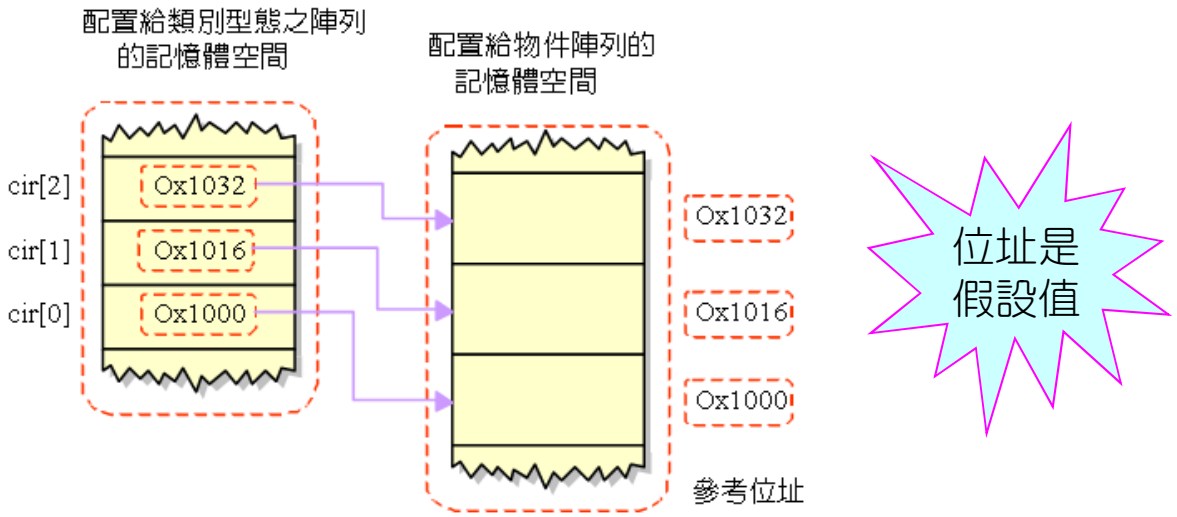
用 new 建立新的物件，並配置記憶體空間給陣列元素

類別型態的陣列 (2/2)

- 利用for迴圈亦可完成指向新建立物件之動作：

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

- 下圖為類別型態陣列與物件陣列的記憶空間配置情形：



建立物件陣列的範例

```
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[];
21        cir=new CCircle[3];
22        cir[0]=new CCircle(1.0);
23        cir[1]=new CCircle(4.0);
24        cir[2]=new CCircle(2.0);
25
26        cir[0].show();   // 利用物件 cir[0]呼叫 show() method
27        cir[1].show();   // 利用物件 cir[1]呼叫 show() method
28        cir[2].show();   // 利用物件 cir[2]呼叫 show() method
29     }
30  }
```

宣告類別型態的陣列，並用 new 配置記憶體空間

用 new 產生新的物件，並配置給陣列元素

```
/* app9_12 OUTPUT---
area=3.14
area=50.24
area=12.56
--------------------*/
```

傳遞物件陣列到method

```
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  }
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  }
```

```
public static double compare( CCircle c[] )
```

- `CCircle`：引數型態為 CCircle
- `double`：傳回型態為 double
- `c[]`：傳遞陣列

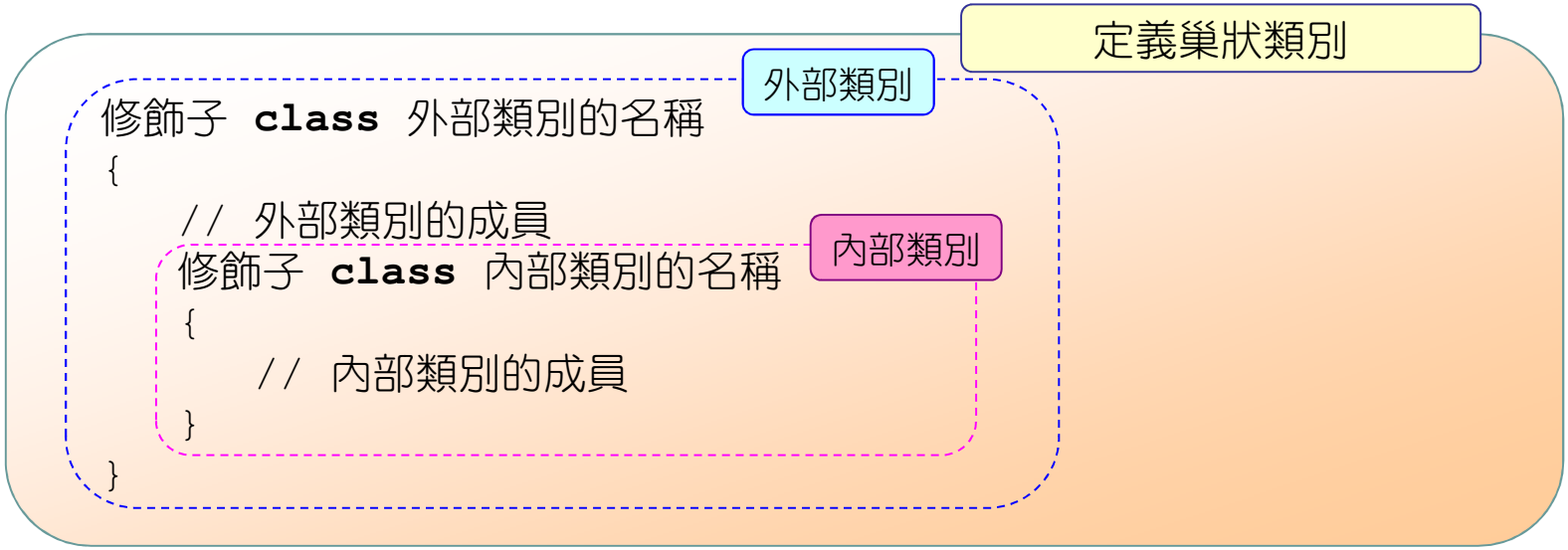
```
CCircle.compare( cir )
```

- `cir`：傳遞陣列時，括號內填上陣列名稱即可

```
/* app9_13 OUTPUT-----
Largest radius = 4.0
---------------------*/
```

巢狀類別的認識

- 巢狀類別（Nested Classes）
  - 在類別A 的內部定義一個沒有宣告成static的類別B
  - 此時的類別B稱為**內部類別**（inner class），類別A稱為**外部類別**（outer class）
  - 內部類別也可宣告成public或private，存取的限制與field或method完全相同



內部類別的撰寫 (1/2)

- 複習一下類別的基本格式 ：

```
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
--------------------*/
```

內部類別的撰寫 (2/2)

- 將類別Caaa改寫為內部類別

```
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
--------------------*/
```

建立內部類別的物件的方法

```
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[])
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  }
```

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

內部類別

在外部類別的建構元裡建立內部類別的物件之做法：

(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

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

建立匿名內部類別

- 「**匿名內部類別**」（anonymous inner class）可利用**內部類別**建立**不具名稱的物件**，並存取類別裡的成員
- 建立匿名內部類別並存取成員的語法如下：

建立匿名內部類別

```
(
    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

匿名內部類別的用法 (1/2)

```
/* app9_17 OUTPUT---
num= 5
--------------------*/
```

- 匿名內部類別
  - 用來補足內部類別裡沒有定義到的method，可化簡程式碼：

```
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

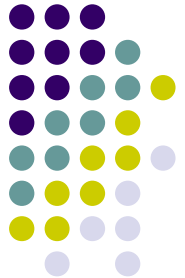
匿名內部類別的用法 (2/2)

- 匿名內部類別也可以"擠" 在短短的幾行：

```
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)

巢狀類別裡的其它成員



- 巢狀類別裡的成員，包括
  - 一般的資料成員
  - 成員函數
  - 成員類別，也就是內部類別

巢狀類別的成員之撰寫格式

外部類別

```
修飾子 class 外部類別的名稱
{
    // 外部類別的成員
    資料型態 內部類別名稱 變數名稱;
    修飾子 class 內部類別的名稱
    {
        // 內部類別的成員
    }
}
```

內部類別的變數

內部類別

巢狀類別的範例 (1/2)

- 利用外部類別的成員，存取、呼叫內部類別的成員：

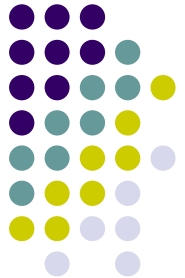
```
01  // app9_19, 巢狀類別
02  class CBox                  // 外部類別
03  {
04     private int length;      // CBox 類別物件的長
05     private int width;       // CBox 類別物件的寬
06     private int height;      // CBox 類別物件的高
07     private CColor cr;       // CColor 類別的物件變數 cr，用來表示顏色
08
09     public CBox(int l,int 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
```

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

巢狀類別的範例 (2/2)

```
20        public CColor(String clr)     // CColor 建構元
21        {
22           color=clr;
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
--------------------*/
```



巢狀類別的特點

- 要特別強調**類別與類別之間的特殊關係**時，就可以使用巢狀類別
- 巢狀類別在使用上有如下的**特點**：
  - 當巢狀類別定義成public時，其內部類別也擁有public的權限
  - 巢狀類別裡的**內部類別**為外部類別的成員之一，因此外部類別與內部類別裡的成員可以互相存取、呼叫，不受private的限制