# 第九章
# 類別的進階認識

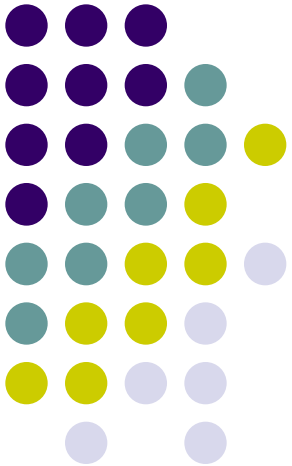
認識建構子與建構子的多載

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



# 建構子的基本認識

- 建構子（constructor）是幫助新建立的物件設定初值
- 建構子的名稱必須與其所屬之類別的類別名稱相同
- 建構子可視為一種特殊的函數，其語法如下：

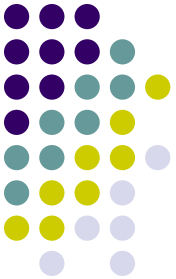
定義建構子的語法

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

可以是 public 或 private

建構子的名稱必須和類別名稱相同

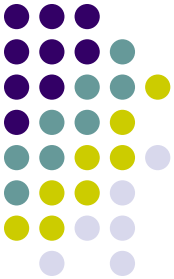
建構子沒有傳回值



# 建構子的呼叫時機

- 一般的函數
  - 在需要用到時才呼叫
- 建構子
  - 在建立物件時，便會自動呼叫，並執行建構子的內容
  - 建構子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

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

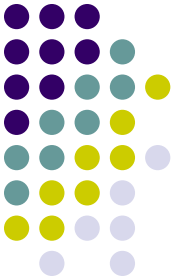


# 建構子的使用範例

```
01  // Ch9_1, 建構子的使用
02  class Circle{                       // 定義類別 Circle
03     private double pi=3.14;
04     private double radius;
05
06     public Circle(double r){         // 定義建構子 Circle()
07        radius=r;
08     }
09     public void show(){
10        System.out.printf("radius=%5.2f, area=%6.2f",radius,pi*radius*radius);
11     }
12  }
13  public class Ch9_1{
14     public static void main(String[] args){
15        Circle c1=new Circle(4.0);      // 建立物件並呼叫 Circle()建構子
16        c1.show();
17     }
18  }
```

執行結果：

```
radius= 4.00, area= 50.24
```



# 建構子的多載 (1/2)

- 建構子也可以多載，如下面的範例：

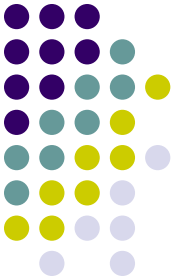
```
01  // Ch9_2,建構子的多載
02  class Circle{                  // 定義類別 Circle
03     private String color;
04     private double pi=3.14;
05     private double radius;
06
07     public Circle(){                          // 沒有引數的建構子
08        System.out.println("Constructor Circle() called");
09        color="Green";
10        radius=1.0;
11     }
12     public Circle(String str, double r){    // 有兩個引數的建構子
13        System.out.println("Constructor Circle(String,double) called");
14        color=str;
15        radius=r;
16     }
17     public void show(){
18        System.out.printf("color=%s, Radius=%5.2f\n",color,radius);
19        System.out.printf("area=%6.2f\n",pi*radius*radius);
20     }
21  }
```

執行結果：

```
Constructor Circle() called
color=Green, Radius= 1.00
area=  3.14
Constructor Circle(String,double) called
color=Blue, Radius= 4.00
area= 50.24
```



接續下一頁



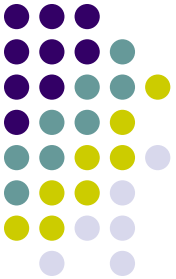
# 建構子的多載 (2/2)

```
22  public class Ch9_2{
23     public static void main(String[] args){
24        Circle c1=new Circle();              // 呼叫沒有引數的建構子
25        c1.show();
26
27        Circle c2=new Circle("Blue",4.0);    // 呼叫有引數的建構子
28        c2.show();
29     }
30  }
```

呼叫不同的建構子

執行結果：

```
Constructor Circle() called
color=Green, Radius= 1.00
area=  3.14
Constructor Circle(String,double) called
color=Blue, Radius= 4.00
area= 50.24
```



# 建構子之間的呼叫 (1/3)

- 從某建構子呼叫另一建構子，是透過this() 來呼叫 ：

```
01  // Ch9_3, 從某一建構子呼叫另一建構子
02  class Circle{                       // 定義類別 Circle
03     private String color;
04     private double pi=3.14;
05     private double radius;
06
07     public Circle(){                 // 沒有引數的建構子
08        this("Green",1.0);            // 此行會呼叫第 11 行的建構子
09        System.out.println("Constructor Circle() called");
10     }
11     public Circle(String str, double r){    // 有引數的建構子
12        System.out.println("Constructor Circle(String,double) called");
13        color=str;
14        radius=r;
15     }
16     public void show(){
17       System.out.printf("color=%s, Radius=%5.2f\n",color,radius);
18       System.out.printf("area=%6.2f\n",pi*radius*radius);
19     }
20  }
```

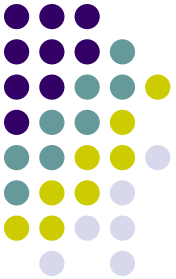
把color設為 "Green"，radius設為1.0，必須以this()呼叫

執行結果：

```
Constructor Circle(String,double) called
Constructor Circle() called
color=Green, Radius= 1.00
area=  3.14
```



接續下一頁

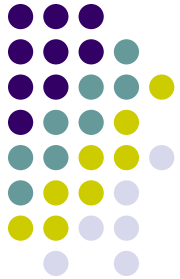


# 建構子之間的呼叫(2/3)

```
21  public class Ch9_3{
22     public static void main(String[] args){
23        Circle c1=new Circle();
24        c1.show();
25     }
26  }
```

執行結果：

```
Constructor Circle(String,double) called
Constructor Circle() called
color=Green, Radius= 1.00
area=  3.14
```



# 建構子之間的呼叫(3/3)

- 於某建構子呼叫另一建構子時，必須以this() 來呼叫
  - 例如，若把第8行改寫為：

```
Circle("Green",1.0);          // 錯誤的建構子呼叫
```

編譯時會出現如下的錯誤訊息：

"The method Circle(String, double) is undefined for the type Circle"

- this() 必須寫在建構子內第一行的位置

# 建構子的公有與私有 (1/3)

- public建構子可以在程式的任何地方被呼叫
- private建構子無法在該建構子所在的類別以外的地方被呼叫
- 看看下面的範例：

```
01  // Ch9_4, 公有與私有建構子的比較
02  class Circle{                    // 定義類別 Circle
03     private String color;
04     private double pi=3.14;
05     private double radius;
06
07     private Circle(){                        // 私有建構子
08        System.out.println("Private constructor called");
09     }
```

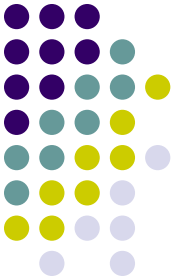
只能在Circle類別內被呼叫

執行結果：

```
Private constructor called
color=Blue, Radius= 1.00
area=  3.14
```



接續下一頁



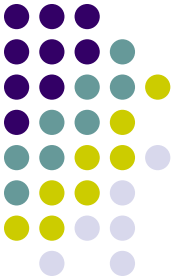
# 建構子的公有與私有 (2/3)

```
10    public Circle(String str, double r){   // 公有建構子
11       this();
12       color=str;
13       radius=r;
14    }
15    public void show(){
16      System.out.printf("color=%s, Radius=%5.2f\n",color,radius);
17      System.out.printf("area=%6.2f\n",pi*radius*radius);
18    }
19 }
20 public class Ch9_4{
21    public static void main(String[] args){
22       Circle c1=new Circle("Blue",1.0);
23       c1.show();
24    }
25 }
```

可在Circle類別的內部或外部呼叫

執行結果：

```
Private constructor called
color=Blue, Radius= 1.00
area=  3.14
```



# 建構子的公有與私有 (3/3)

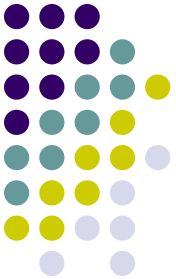
- 如果把第22行的敘述改為：

```
Circle c1=new Circle();      // 呼叫 private 的建構子 Circle()
```

  - 將會得到下列的錯誤訊息：

    "The constructor Circle() is not visible"

  - 這是因為private的建構子無法在類別外部被呼叫
  - private建構子可對建構子的存取設限



# 建構子的省略

- 如果省略建構子
  - Java會呼叫預設的建構子（default constructor）
  - 預設的建構子是沒有任何引數的建構子，格式如下：

預設的建構子格式

```
public Circle(){    // 預設的建構子
}
```

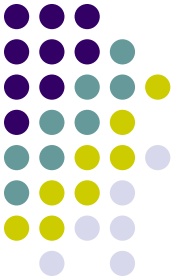
建構子的三個重要特點

(1) 建構子的名稱和類別名稱相同
(2) 建構子裡沒有引數
(3) 不做任何事情，也就是建構子內沒有任何的敘述

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

# 實例變數與實例函數





```
01  // Ch9_5, 簡單的範例:實例變數與實例函數
02  class Circle{
03     private double pi=3.14;
04     private double radius;
05
06     public Circle(double r){    // Circle()建構子
07        radius=r;
08     }
09     public void show(){
10        System.out.printf("area=%6.2f\n",pi*radius*radius);
11     }
12  }
13  public class Ch9_5{
14     public static void main(String[] args){
15        Circle c1=new Circle(1.0);
16        c1.show();              // show()必須透過物件來呼叫
17        Circle c2=new Circle(2.0);
18        c2.show();              // show()必須透過物件來呼叫
19     }
20  }
```

實例變數

實例函數

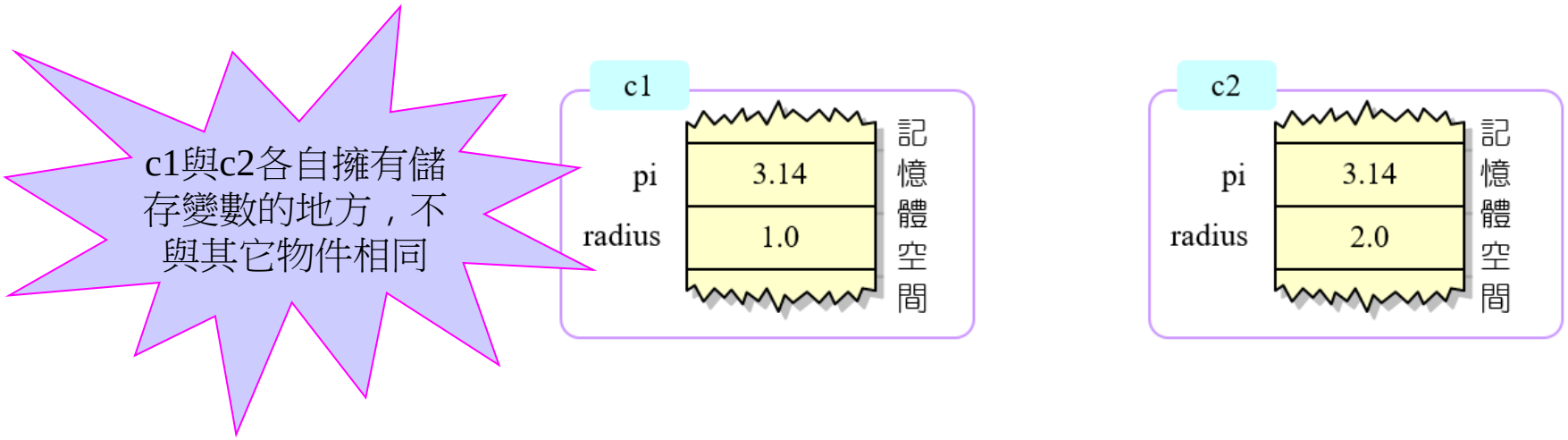
認識實例變數
與實例函數

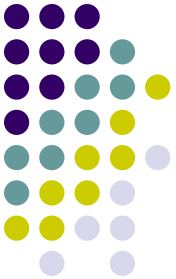
執行結果：

```
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)
  - 變數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函數

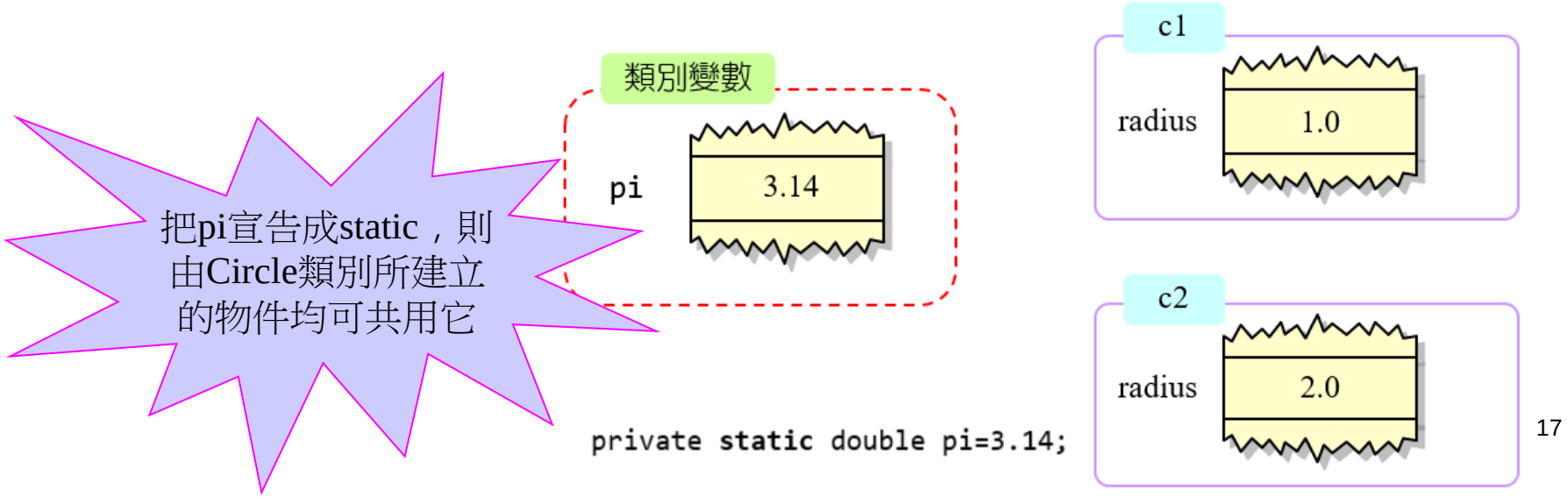
```
Circle c1=new Circle(1.0);     // 建立物件 c1
c1.show();                     // 由物件 c1 呼叫 show() 函數
Circle c2=new Circle(2.0);     // 建立物件 c2
c2.show();                     // 由物件 c2 呼叫 show() 函數
```

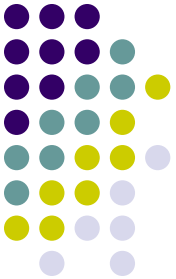
show() 必須透過物件c1或c2呼叫

# 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
- 類別變數必須以static修飾子宣告：

```
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：

```
01  // Ch9_6,「類別變數」的使用
02  class Circle{
03     private static int count=0;          // 宣告 count 為類別變數
04     private static double pi=3.14;       // 宣告 pi 為類別變數
05     private double radius;
06
07     public Circle(){                     // 沒有引數的 Circle()建構子
08        this(1.0);                        // 呼叫第 10 行的建構子，並傳入 1.0
09     }
10     public Circle(double r){             // 有一個引數的 Circle()建構子
11        radius=r;
12        count ++;                         // 當此建構子被呼叫時，count 便加 1
13     }
14     public void show(){
15        System.out.printf("area=%6.2f",pi*radius*radius);
16     }
17     public void show_count(){            // show_count(),顯示目前物件建立的個數
18        System.out.println(count+" object(s) created");
19     }
20  }
```

透過this() 呼叫定義在12~16行的建構子

用來計算物件的數目，count宣告為static，它由所有的物件所共用

執行結果：

```
1 object(s) created
3 object(s) created
3 object(s) created
3 object(s) created
```

接續下一頁

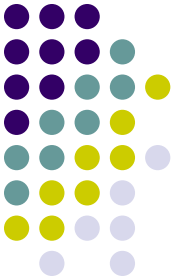
# 類別變數的範例 (2/2)

```
21  public class Ch9_6{
22     public static void main(String[] args){
23        Circle c1=new Circle();          // 呼叫第 7 行的建構子
24        c1.show_count();     // 用 c1 物件呼叫 show_count() 函數
25        Circle c2=new Circle(2.0); // 呼叫第 10 行的建構子
26        Circle c3=new Circle(4.3); // 呼叫第 10 行的建構子
27        c1.show_count();     // 用 c1 物件呼叫 show_count() 函數
28        c2.show_count();     // 改用 c2 物件呼叫 show_count() 函數
29        c3.show_count();     // 改用 c3 物件呼叫 show_count() 函數
30     }
31  }
```

執行結果：

```
1 object(s) created
3 object(s) created
3 object(s) created
3 object(s) created
```

均是透過物件來呼叫函數



# 類別函數

- 若將函數定義成類別函數，則可以直接由類別呼叫

在函數之前加上static即可定義成類別函數

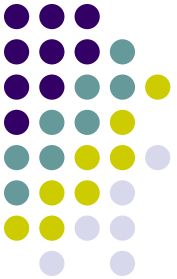
```
public static void show_count(){     // 將 show_conut()定義成類別函數
   System.out.println(count+" object(s) created");
}
```

- 使用時直接用類別呼叫：

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

# 類別函數的使用 (1/2)





```
01  // Ch9_7,「類別函數」的使用
02  class Circle{
03     private static int count=0;          // 宣告 count 為類別變數
04     private static double pi=3.14;       // 宣告 pi 為類別變數
05     private double radius;
06
07     public Circle(){                     // 沒有引數的 Circle()建構子
08        this(1.0);                        // 呼叫第 10 行的建構子，並傳入 1.0
09     }
10     public Circle(double r){             // 有一個引數的 Circle()建構子
11        radius=r;
12        count++;                          // 當此建構子被呼叫時，count 便加 1
13     }
14     public void show(){
15        System.out.printf("area=%6.2f\n",pi*radius*radius);
16     }
17     public static void show_count(){  // 顯示目前物件建立的個數
18        System.out.println(count+" object(s) created");
19     }
20  }
```

宣告show_count() 為類別函數

執行結果：

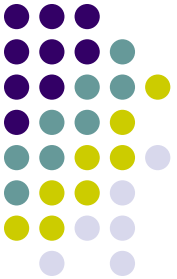
```
0 object(s) created
1 object(s) created
3 object(s) created
```



接續下一頁

# 類別函數的使用 (2/2)





```
21  public class Ch9_7{
22     public static void main(String[] args){
23        Circle.show_count();        // 用 Circle 類別呼叫 show_count()
24        Circle c1=new Circle();     // 呼叫第 7 行的建構子
25        Circle.show_count();        // 用 Circle 類別呼叫 show_count()
26        Circle c2=new Circle(2.0); // 呼叫第 10 行的建構子
27        Circle c3=new Circle(4.3); // 呼叫第 10 行的建構子
28        c3.show_count();            // 用 c3 物件呼叫 show_count()
29     }
30  }
```

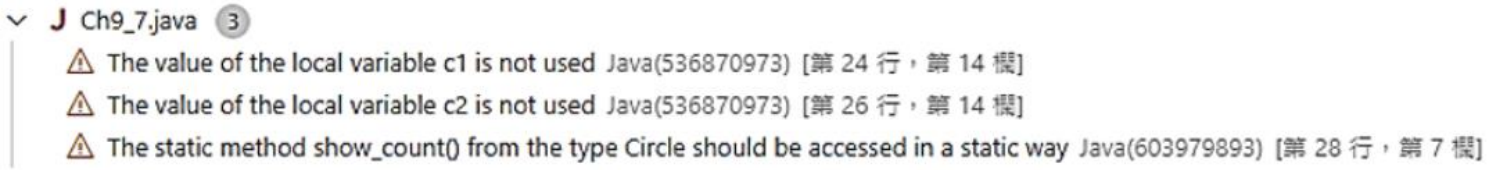
類別函數也可以透過物件呼叫

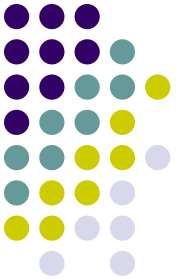
類別函數可直接利用類別來呼叫

執行結果：

```
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() 函數與static修飾子

- main() 函數 也有一個static修飾子：

```
21  public class Ch9_7{
22      public static void main(String[] args){
            ...
29      }
30  }
```

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

- 在執行時main() 直接由類別Ch9_7呼叫，因此main()必須宣告成static

# 「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Ch9_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
}
```

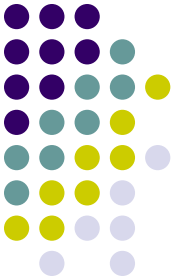
編譯時將產生如下的錯誤：

radius不是類別變數，無法由「類別函數」的內部呼叫

"Cannot make a static reference to the non-static field radius"

"Cannot make a static reference to the non-static method show() from the type Circle"

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：

"Cannot use this in a static context"

在「類別函數」內部不能使用this關鍵字

# 設值給類別型態的變數



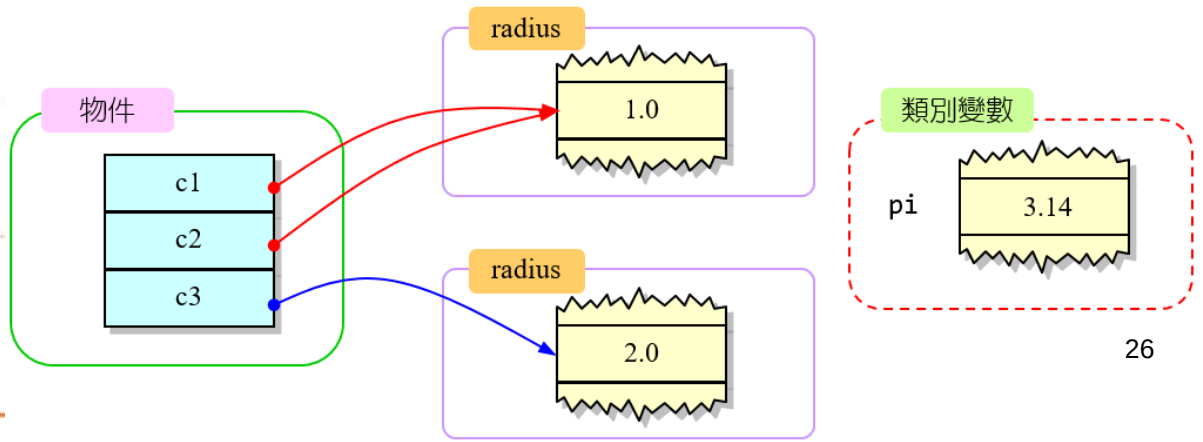
```
01  // Ch9_8, 設值給物件變數
02  class Circle{           // 定義類別 Circle
03     private static double pi=3.14;
04     private double radius;
05
06     public Circle(double r){
07         radius=r;
08     }
09     public void show(){
10         System.out.printf("area=%6.2f\n",pi*radius*radius);
11     }
12  }
13  public class Ch9_8{
14     public static void main(String[] args){
15        Circle c1,c2;            // 宣告 c1,c2 為物件變數
16        c1=new Circle(1.0);      // 建立新的物件，並將 c1 指向它
17        c1.show();
18
19        c2=c1;     // 將 c1 設給 c2，此時這兩個變數所指向的內容均相等
20        c2.show();
21
22        Circle c3=new Circle(2.0);
23        c3.show();
24     }
25  }
```

類別型態的變數

執行結果：

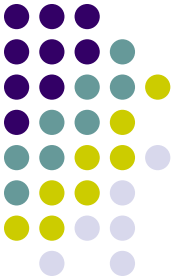
```
area=  3.14
area=  3.14
area= 12.56
```

設定c2=c1可將兩個類別型態的變數指向同一個物件



# 類別型態的變數另一例





```
01  // Ch9_9, 物件變數使用的注意事項
02  class Circle{
03     private static double pi=3.14;
04     private double radius;
05
06     public Circle(double r){      // Circle 建構子
07        radius=r;
08     }
09     public void setRadius(double r){
10        radius=r;                  // 設定 radius 成員的值
11     }
12     public void show(){
13        System.out.printf("area=%6.2f\n",pi*radius*radius);
14     }
15  }
16  public class Ch9_9{
17     public static void main(String[] args){
18        Circle c1,c2;
19        c1=new Circle(1.0);
20        c1.show();
21        c2=c1;                     // 將 c1 設給 c2，此時這兩個變數指向相同的物件
22        c2.setRadius(2.0);         // 將 c2 物件的半徑設為 2.0
23        c1.show();
24     }
25  }
```

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

執行結果：

```
area=  3.14
area= 12.56
```

# 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() 函數須以下面的格式來撰寫：

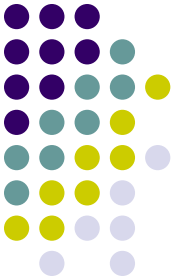
傳遞物件變數到函數的語法

```
傳回值型別 compare( Circle obj){
  ...
}
```

引數型別為 Circle

# 比較二個物件是否相等





```
01  // Ch9_10, 傳遞物件變數
02  class Circle{
03     private static double pi=3.14;
04     private double radius;
05
06     public Circle(double r){        // Circle()建構子
07        radius=r;
08     }
09     public void compare(Circle cir){  // compare() 函數
10        if(this.radius==cir.radius)    // 判別物件的 radius 成員是否相等
11           System.out.println("Radius are equal");
12        else
13           System.out.println("Radius are not equal");
14     }
15  }
16  public class Ch9_10{
17     public static void main(String[] args){
18        Circle c1=new Circle(1.0);
19        Circle c2=new Circle(2.0);
20        c1.compare(c2);        // 比較 c1 與 c2 的 radius 是否相等
21     }
22  }
```

不能寫成 `if(this==cir)`

執行結果：

```
Radius are not equal
```

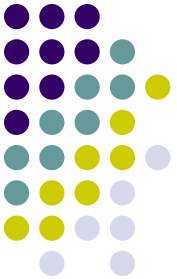
# 由函數傳回類別型態的變數

- 由compare() 函數傳回Circle類別型態的變數：

從函數傳回物件變數的語法

傳回 Circle 型別的變數

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



# 傳回類別型態的變數範例(1/2)

- 下面的範例會比較物件半徑的大小，並傳回半徑較大的物件：

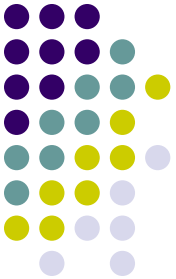
```
01  // Ch9_11, 由函數傳回物件變數
02  class Circle{
03     private static double pi=3.14;
04     private double radius;
05
06     public Circle(double r) {     // Circle 建構子
07        radius=r;
08     }
09     public Circle compare(Circle cir) { // Compare() 函數
10        if(this.radius>cir.radius)
11           return this;             // 傳回呼叫 compare() 函數的物件
12        else
13           return cir;              // 傳回傳入 compare() 函數的物件
14     }
15  }
```

執行結果：

```
radius of c2 is larger
```



接續下一頁



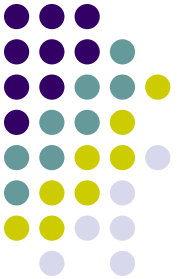
# 傳回類別型態的變數範例(2/2)

```
16  public class Ch9_11{
17     public static void main(String[] args){
18        Circle c1=new Circle(1.0);
19        Circle c2=new Circle(2.0);
20        Circle obj;
21
22        obj=c1.compare(c2);      // 呼叫 compare() 函數
23        if(c1==obj)
24           System.out.println("radius of c1 is larger");
25        else
26           System.out.println("radius of c2 is larger");
27     }
28  }
```

由obj接收傳回的變數(物件)

執行結果：

```
radius of c2 is larger
```



# 類別型態的陣列

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。

合併成一行 `Circle cir[]=new Circle[3];`

```
Circle[] cir;          // 宣告 Circle 型別的陣列 cir
cir=new Circle[3];     // 配置記憶體空間給 cir
```

- 建立好陣列之後，便可把陣列元素指向物件：

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

將每一個陣列元素 cir 指向由 Circle 類別所建立的物件

- 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 Circle();
}
```

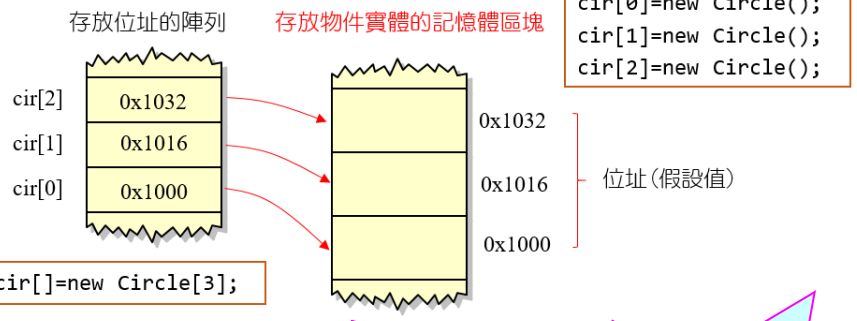
# 建立物件陣列的範例



```
01  // Ch9_12, 建立物件陣列
02  class Circle{
03     private static double pi=3.14;
04     private double radius;
05
06     public Circle(double r){
07        radius=r;
08     }
09     public void show(){
10        System.out.printf("area=%6.2f\n",pi*radius*radius);
11     }
12  }
13  public class Ch9_12{
14     public static void main(String[] args){
15        Circle[] cir;
16        cir=new Circle[3];
17        cir[0]=new Circle(1.0);
18        cir[1]=new Circle(4.0);
19        cir[2]=new Circle(2.0);
20
21        cir[0].show();   // 利用物件 cir[0]呼叫 show() 函數
22        cir[1].show();   // 利用物件 cir[1]呼叫 show() 函數
23        cir[2].show();   // 利用物件 cir[2]呼叫 show() 函數
24     }
25  }
```

宣告 Circle 型別的陣列，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



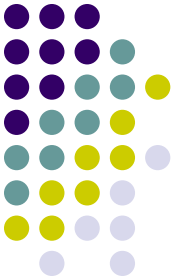

類別型態陣列與物件陣列的記憶空間配置情形

執行結果：

```
area=  3.14
area= 50.24
area= 12.56
```

# 傳遞物件陣列到函數





```
01  // Ch9_13, 傳遞物件陣列到函數
02  class Circle{
03     private static double pi=3.14;
04     private double radius;
05
06     public Circle(double r){
07        radius=r;
08     }
09     public static double compare(Circle[] c){  // compare() 函數
10        double max=0.0;
11        for(int i=0;i<c.length;i++)
12           if(c[i].radius>max)
13              max=c[i].radius;
14        return max;
15     }
16  }
17  public class Ch9_13{
18     public static void main(String[] args){
19        Circle[] cir;
20        cir=new Circle[3];
21
22        cir[0]=new Circle(1.0);
23        cir[1]=new Circle(4.0);
24        cir[2]=new Circle(2.0);
25        System.out.println("Largest radius = "+Circle.compare(cir));
26     }
27  }
```

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

靜態函數

Circle 型別的陣列

傳回型別為 double

陣列名稱

```
Circle.compare(cir)
```

傳遞陣列時，括號內填上陣列名稱即可

執行結果：

```
Largest radius = 4.0
```

# 內部類別的認識

- 巢狀類別（Nested Classes）
  - 在類別A的內部定義一個類別B

外部類別（outer class）

內部類別（inner class）

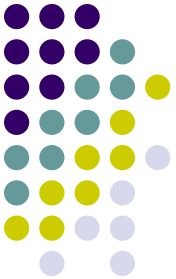
可宣告成public或是private

定義內部類別的語法

```
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



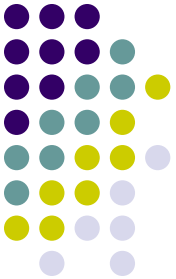
# 內部類別的撰寫 (1/2)

- 複習一下類別的基本格式 ：

```
01  // Ch9_14, 類別的複習
02  class Circle{
03     private double radius;
04     void set_radius(double r){
05        radius=r;
06        System.out.printf("radius=%5.2f",radius);
07     }
08  }
09  public class Ch9_14{
10     public static void main(String[] args){
11        Circle c1=new Circle();
12        c1.set_radius(5.2);
13     }
14  }
```

執行結果：

```
radius= 5.20
```



# 內部類別的撰寫 (2/2)

- 將類別Circle改寫為內部類別

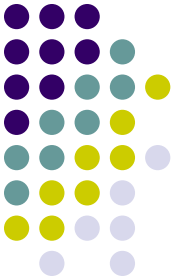
```
01  // Ch9_15, 內部類別的使用
02  public class Ch9_15{
03     public static void main(String[] args){
04        Circle c1=new Circle();
05        c1.set_radius(5.2);
06     }
07     static class Circle{
08        private double radius;
09        void set_radius(double r){
10           radius=r;
11           System.out.printf("radius=%5.2f",radius);
12        }
13     }
14  }
```

外部類別

內部類別

執行結果：

```
radius= 5.20
```



# 外部類別取用內部類別的成員(1/2)

```
01  // Ch9_16, 於外部類別內取用內部類別的成員
02  class Circle{   // 外部類別
03     private double radius;
04     private Color clr;
05
06     public Circle(double r, String c){
07        radius=r;
08        clr=new Color(c);
09        System.out.println("Circle() 建構子被呼叫了");
10     }
11     public void show(){
12        System.out.printf("radius=%5.2f, color= %s\n",radius,clr.color);
13     }
14     private class Color{     // Circle 的內部類別
15        private String color;
16        Color(String c){
17           color=c;
18           System.out.println("Color() 建構子被呼叫了");
19        }
20     }
21  }
```

在外部類別的建構子裡建立內部類別的物件之做法：

(1) 在外部類別的建構子裡建立內部類別的物件

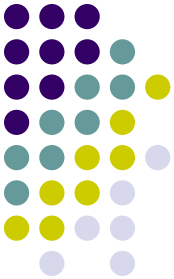
(2) 在main() 裡建立一個外部類別的物件

執行結果：

```
Color() 建構子被呼叫了
Circle() 建構子被呼叫了
radius= 2.00, color= Blue
```



接續下一頁



# 外部類別取用內部類別的成員(2/2)

```
22  public class Ch9_16{
23     public static void main(String[] args){
24        Circle c1=new Circle(2.0,"Blue");
25        c1.show();
26     }
27  }
```

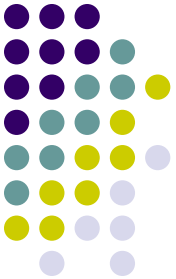
執行結果：

```
Color() 建構子被呼叫了
Circle() 建構子被呼叫了
radius= 2.00, color= Blue
```

在外部類別的建構子裡建立內部類別的物件之做法：

(1) 在外部類別的建構子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



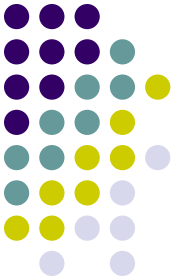
# 回收記憶體 (1/2)

- 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：

```
class Test{
   public static void main(String[] args){
      Circle c1=new Circle();      // 建立物件，並配置記憶體給它
      ...
      c1=null;                     // 將 c1 指向 null,代表 c1 已不再指向任何物件
      ...
   }
}
```



# 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：

```
class Test{
   public static void main(String[] args){
   Circle c1=new Circle();
   Circle c2;
   c2=c1;        // 設定 c2 與 c1 均指向同一個物件
   ....
   c1=null;      // 將 c1 指向 null,但 c2 仍指向該物件，因此不會被回收
   }
}
```


-The End-