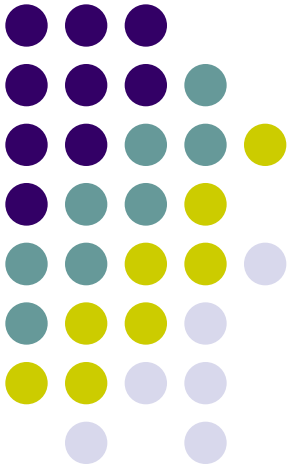
第十一章
抽象類別與介面

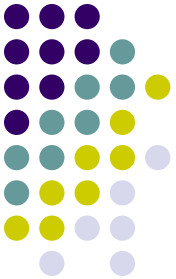
認識抽象類別

學習介面的使用

認識多重繼承與介面的延伸

使用instanceof運算子





抽象類別

- 目的是讓使用者依據它的格式建立新的類別：

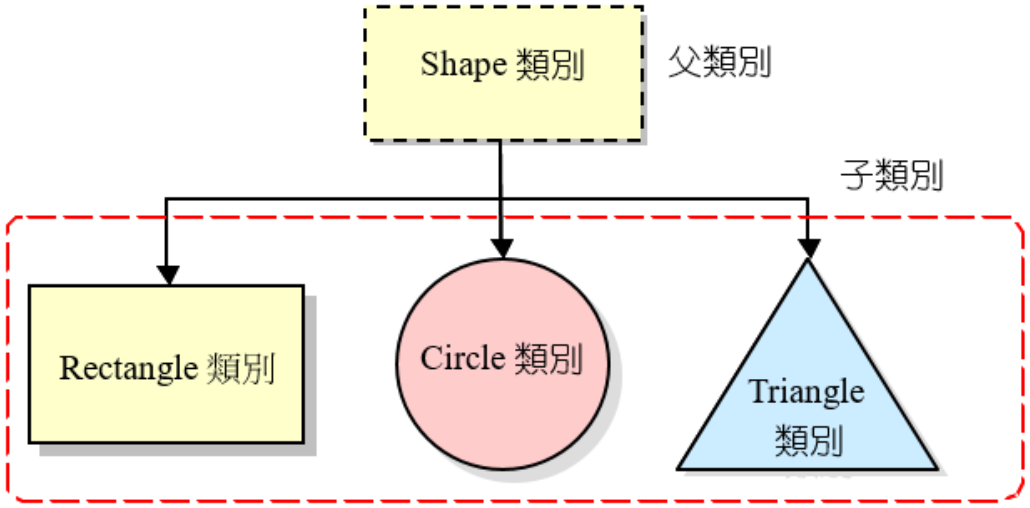
定義抽象類別的語法

```
abstract class 類別名稱 {   // 定義抽象類別
    宣告資料成員;

    傳回值的資料型態 函數名稱(引數...){
        ...
    }                                          } 定義一般函數
    修飾子 abstract 傳回值資料型態 函數名稱(引數...);  — 定義抽象函數。注意於抽象函數裡，沒有定義處理的方式
}
```

抽象類別的實作 (1/6)

- 設計一個父類別CShape，並依據此類別衍生出
  - 圓形類別
  - 長方形類別
  - 三角形類別



抽象類別的實作 (2/6)

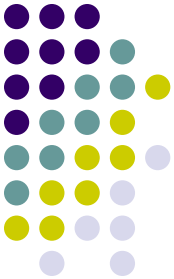
- 抽象類別CShape的內容：

抽象類別必須以abstract宣告

```
01  // 定義抽象類別 Shape
02  abstract class Shape{          // 定義抽象類別 Shape
03     protected String color;     // 資料成員
04     public void setColor(String str){   // 一般函數，用來設定何形狀的顏色
05        color=str;
06     }
07     public abstract void show();         // 抽象函數，在此沒有定義處理方式
08  }
```

抽象函數以abstract宣告

show() 並沒有定義處理的方式



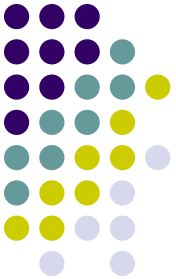
抽象類別的實作 (3/6)

- 下面的程式碼是以子類別Circle為例來撰寫的：

```
01  // 定義由抽象類別 Shape 而衍生出的子類別 Circle
02  class Circle extends Shape{      // 定義子類別 Circle
03     protected double radius;      // 資料成員
04     public Circle(double r){      // 建構元
05        radius=r;
06     }
07     public void show(){
08        System.out.print("color="+color+",  ");
09        System.out.println("area="+3.14*radius*radius);
10     }
11  }
```

在此處明確定義 show() 的處理方式

在父類別裡宣告成abstract的函數，在此明確定義



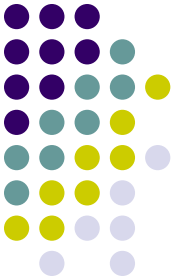
抽象類別的實作 (4/6)

- Ch11_1是抽象類別實作的完整範例：

```
01  // Ch11_1, 抽象類別的實例
02  abstract class Shape{                          // 定義抽象類別 Shape
03     protected String color;                     // 資料成員
04     public void setColor(String str){           // 一般的函數
05        color=str;
06     }
07     public abstract void show(); // 抽象函數，只有定義名稱，沒有定義處理方式
08  }
09  class Rectangle extends Shape{    // 定義子類別 Rectangle
10     protected int width,height;
11     public Rectangle(int w,int h){
12        width=w;
13        height=h;
14     }
15     public void show(){       // 明確定義繼承自抽象類別的 show()
16        System.out.print("color="+color+",  ");
17        System.out.println("area="+width*height);
18     }
19  }
```

執行結果：

```
color=Yellow,  area=50
color=Green,  area=12.56
```



抽象類別的實作 (5/6)

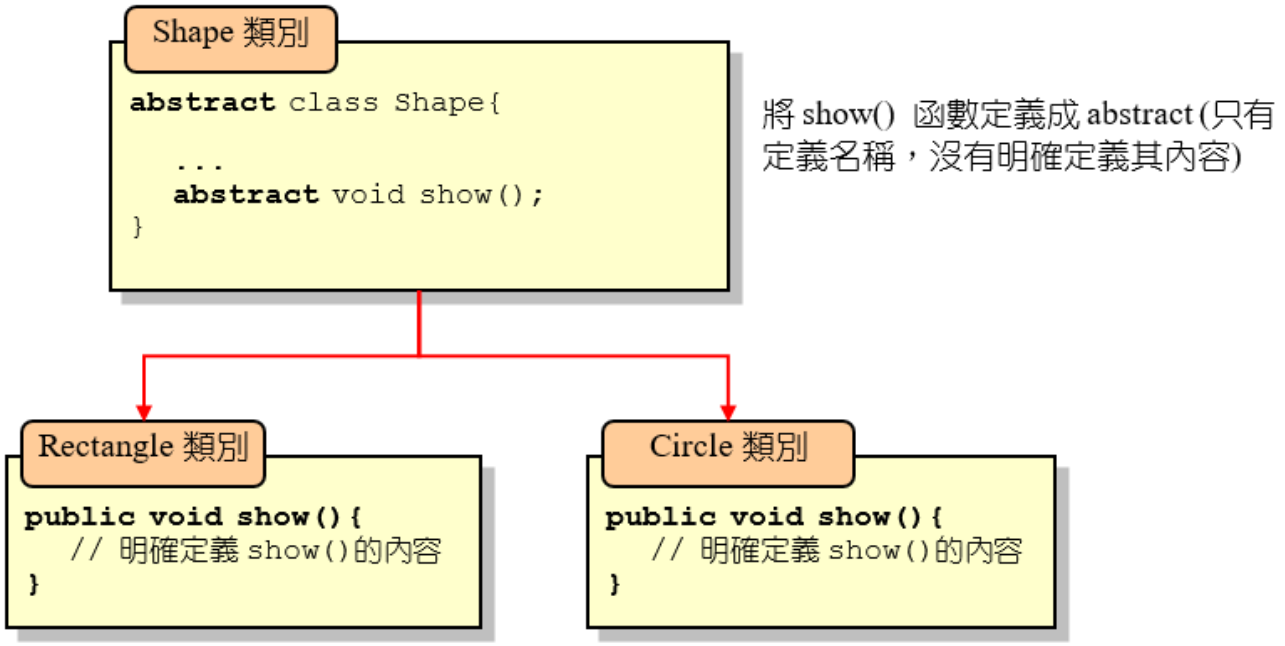
```
20  class Circle extends Shape{      // 定義子類別 Circle
21     protected double radius;
22     public Circle(double r){
23        radius=r;
24     }
25     public void show(){     // 明確定義繼承自抽象類別的 show()
26        System.out.print("color="+color+",  ");
27        System.out.println("area="+3.14*radius*radius);
28     }
29  }
30  public class Ch11_1{
31     public static void main(String args[]){
32        Rectangle r1=new Rectangle(5,10);
33        r1.setColor("Yellow");// 呼叫父類別裡的 setColor()
34        r1.show();             // 呼叫 Rectangle 類別裡的 show()
35
36        Circle c1=new Circle(2.0);
37        c1.setColor("Green");       // 呼叫父類別裡的 setColor()
38        c1.show();                  // 呼叫 Circle 類別裡的 show()
39     }
40  }
```

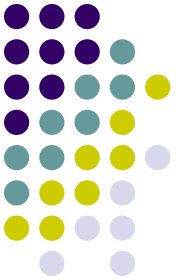
執行結果：

```
color=Yellow,  area=50
color=Green,  area=12.56
```

抽象類別的實作 (6/6)

- Shape、Circle與Rectangle的show() 之間的關係：





用父類別變數存取子類別的成員

- Ch11_2是以抽象類別型態的變數建立物件的範例：

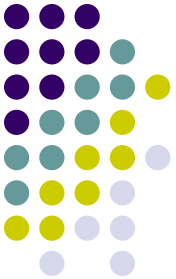
```
01  // Ch11_2, 用抽象類別型態的變數來建立物件
02  // 將 Ch11_1 的 Shape 類別的定義放在這兒
03  // 將 Ch11_1 的 Rectangle 類別的定義放在這兒
04  // 將 Ch11_1 的 Circle 類別的定義放在這兒
05  public class Ch11_2{
06     public static void main(String[] args){
07        Shape s1=new Rectangle(5,10);
08        s1.setColor("Yellow");
09        s1.show();
10
11        Shape s2=new Circle(2.0);
12        s2.setColor("Green");
13        s2.show();
14     }
15  }
```

以類別 Shape 建立物件 s1，並以它來存取子類別 Rectangle 的成員

以類別 Shape 建立物件 s2，並以它來存取子類別 Circle 的成員

執行結果：

```
color=Yellow,  area=50
color=Green,  area=12.56
```



用陣列變數存取子類別物件

- Ch11_3改寫Ch11_2：

```
01  // Ch11_3, 利用父類別的陣列變數來存取子類別的內容
02  // 將 Ch11_1 的 Shape 類別的定義放在這兒
03  // 將 Ch11_1 的 Rectangle 類別的定義放在這兒
04  // 將 Ch11_1 的 Circle 類別的定義放在這兒
05  public class Ch11_3{
06     public static void main(String[] args){
07        Shape s[];          // 宣告 Shape 型態的陣列變數
08        s=new Shape[2];     // 產生兩個 Shape 抽象類別型態的變數
09
10        s[0]=new Circle(2.0);
11        s[0].setColor("Yellow");
12        s[0].show();
13
14        s[1]=new Circle(2.0);
15        s[1].setColor("Green");
16        s[1].show();
17     }
18  }
```

利用陣列變數 s[0]建立物件，並存取子類別的成員

利用陣列變數 s[1]建立物件，並存取子類別的成員

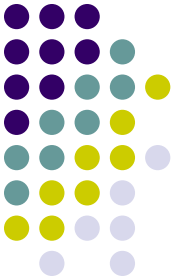
執行結果：

```
color=Yellow,  area=50
color=Green,  area=12.56
```

建立的物件變多時，較好的做法是：

(1) 先建立父類別的陣列變數

(2) 利用陣列元素建立子類別的物件，並以它存取子類別的內容

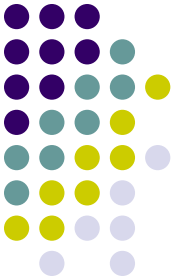


使用抽象類別的注意事項

- 抽象類別不能直接產生物件，因此下面的敘述是錯的：

```
public static void main(String args[]){
    ....
    Shape s;
    s=new Shape();      // 錯誤，不能用抽象類別直接產生物件
}
```

- 抽象類別內可以定義建構元，以供子類別的建構元呼叫
- 抽象類別裡的抽象函數，在子類別裡一定要被「改寫」
- 如果子類別裡沒有「改寫」抽象函數，則子類別也要宣告成abstract



介面

- 介面與抽象類別有下列兩點不同：

  (1) 介面的資料成員必須初始化

  (2) 介面裡的函數必須全部都定義成abstract

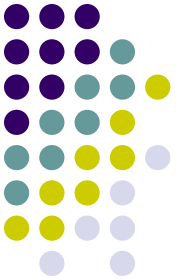
定義介面的語法

```
interface 介面名稱{                          // 定義介面
   final 資料型態 成員名稱=常數;             // 資料成員必須設定初值

   修飾子 abstract 傳回值資料型態 函數名稱(引數...);
}
```

定義抽象函數。注意於抽象函數裡，沒有定義處理的方式

只能宣告為public，或是不做宣告



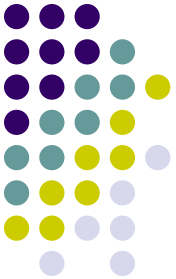
使用介面

- 下面的範例定義一介面iShape2D：

```
01  // 定義 iShape2D 介面
02  interface iShape2D{
03     final double PI=3.14;          // 資料成員一定要初始化
04     abstract void area();          // 抽象函數，不需要定義處理方式
05  }
```

- 上面的程式碼可省略final與abstract：

```
01  // 定義 iShape2D 介面，省略 final 與 abstract 關鍵字
02  interface iShape2D{
03     double PI=3.14;                // 省略 final 關鍵字
04     void area();                   // 省略 abstract 關鍵字
05  }
```

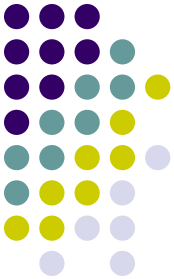


介面的實作

- 利用介面A打造新的類別B的過程，稱為以類別B實作介面A，或簡稱介面的實作（implementation）
- 介面實作的語法如下：

類別實作介面的語法

```
class 類別名稱 implements 介面名稱{     // 介面的實作
   ... ...
}
```

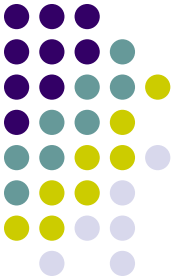


以類別實作介面的範例 (1/3)

- 下面是以Circle類別實作iShape2D介面的範例：

以類別 Circle 來實作介面 iShape2D

```
01  // 介面的實作
02  class Circle implements iShape2D{  // 以 Circle 類別實作 iShape2D 介面
03     double radius;
04     public Circle(double r){        // 建構元
05        radius=r;
06     }
07     public void area(){             // 定義 area()的處理方式
08        System.out.println("area="+PI*radius*radius);
09     }
10  }
```



以類別實作介面的範例 (2/3)

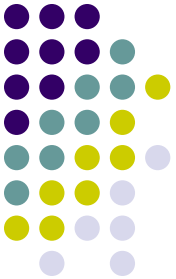
- Ch11_4是以類別實作介面的完整範例：

```
01  // Ch11_4, 介面的實作範例
02  interface iShape2D{                    // 定義介面
03     final double PI=3.14;
04     abstract void area();
05  }
06
07  class Rectangle implements iShape2D{ // 以 Rectangle 類別實作 iShape2D 介面
08     int width,height;
09     public Rectangle(int w,int h){
10        width=w;
11        height=h;
12     }
13     public void area(){       // 定義 area()的處理方式
14        System.out.println("area="+width*height);
15     }
16  }
17
```

執行結果：

```
area=50
area=12.56
```


接續下一頁

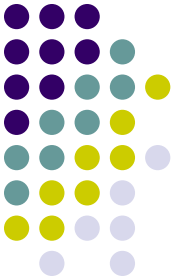


以類別實作介面的範例 (3/3)

```
18  class Circle implements iShape2D{    // 以Circle類別實作 iShape2D 介面
19     double radius;
20     public Circle(double r){
21        radius=r;
22     }
23     public void area(){       // 定義 area()的處理方式
24        System.out.println("area="+PI*radius*radius);
25     }
26  }
27
28  public class Ch11_4{
29     public static void main(String[] args){
30        Rectangle r1=new Rectangle(5,10);
31        r1.area();         // 呼叫 Rectangle 類別裡的 area()
32
33        Circle c1=new Circle(2.0);
34        c1.area();         // 呼叫 Circle 類別裡的 area()
35     }
36  }
```

執行結果：

```
area=50
area=12.56
```



以介面型態的變數存取物件

- Ch11_5是利用介面型態的變數存取物件的範例：

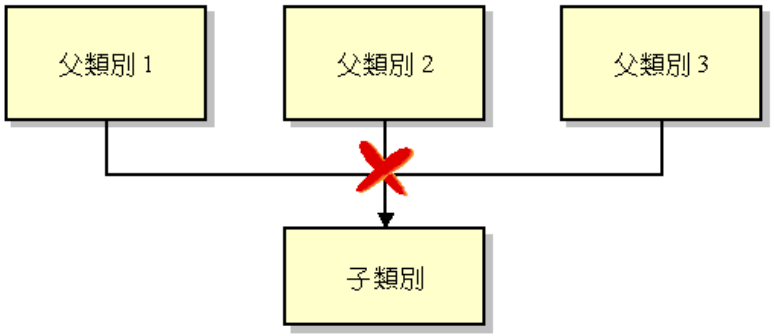
```
01  // Ch11_5,透過介面型態的變數來存取物件
02  // 將 Ch11_4 的 iShape 介面的定義放在這兒
03  // 將 Ch11_4 的 Rectangle 類別的定義放在這兒
04  // 將 Ch11_4 的 Circle 類別的定義放在這兒
05  public class Ch11_5{
06     public static void main(String[] args){
07        iShape2D v1,v2;           // 宣告介面型態的變數
08        v1=new Rectangle(5,10);   // 將介面型態的變數 v1 指向新建的物件
09        v1.area();                // 透過介面 v1 呼叫 show() 函數
10
11        v2=new Circle(2.0);       // 將介面型態的變數 v2 指向新建的物件
12        v2.area();                // 透過介面 v2 呼叫 show() 函數
13     }
14  }
```

執行結果：

```
area=50
area=12.56
```

關於多重繼承

- Java並不允許多個父類別的繼承：

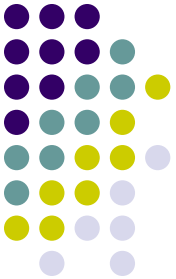


- 但類別可實作兩個以上的介面
- 將類別和兩個以上的介面實作在一起的語法如下：

實作二個以上的介面

```
Class 類別名稱 implements 介面 1, 介面 2, ...{        // 定義介面

   ... ...
}
```



實作兩個以上的介面 (1/2)

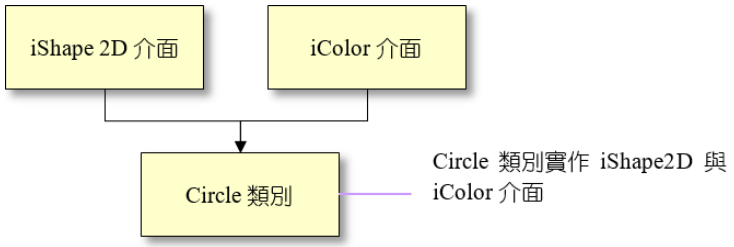
- Ch11_6是以類別實作兩個的介面的範例：

```
01  // Ch11_6, 用 Circle 類別實作兩個以上的介面
02  interface iShape2D{          // 定義 iShape2D 介面
03     final double PI=3.14;
04     abstract void area();
05  }
06
07  interface iColor{            // 定義 iColor 介面
08     abstract void setColor(String str);
09  }
10
11  class Circle implements iShape2D,iColor{    // 實作 iShape2D 與 iColor 介面
12     double radius;
13     String color;
14     public Circle(double r){
15        radius=r;
16     }
```


接續下一頁

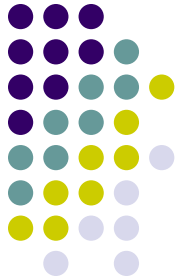
實作兩個以上的介面 (2/2)

```
17      public void setColor(String str){      // 定義 iColor 介面裡的 setColor()
18         color=str;
19         System.out.println("color="+color);
20      }
21      public void area(){           // 定義 iShape2D 介面裡的 area() 函數
22         System.out.println("area="+PI*radius*radius);
23      }
24   }
25   public class Ch11_6{
26      public static void main(String args[]){
27         Circle c1;
28         c1=new Circle(2.0);
29         c1.setColor("Blue");          // 呼叫 setColor()
30         c1.area();                    // 呼叫 show()
31      }
32   }
```



執行結果：

```
color=Blue
area=12.56
```

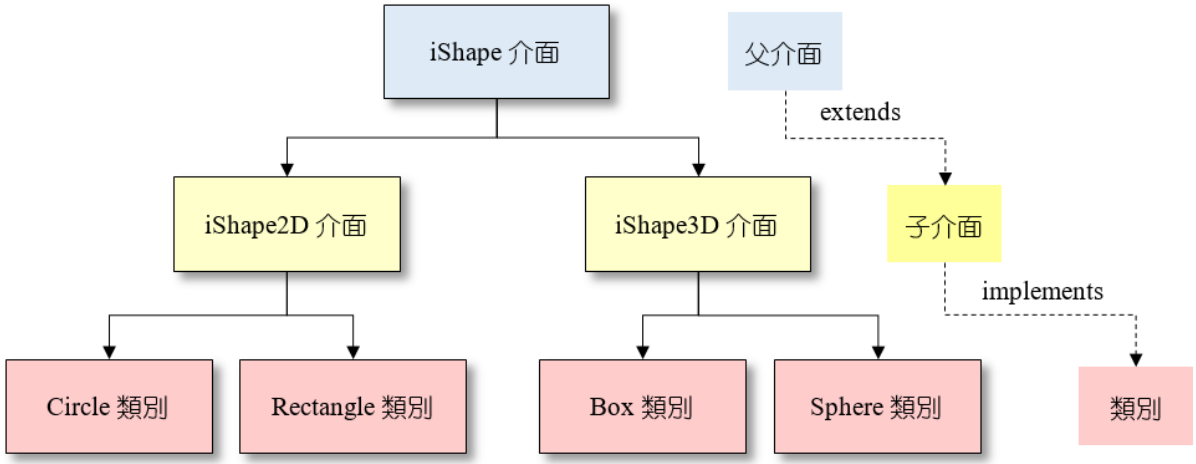


介面延伸的認識

- 介面可透過繼承的技術來衍生出新的介面
  - 原來的介面稱為基底介面（base interface）或父介面（super interface）
  - 衍生出的介面稱為衍生介面（derived interface）或子介面（sub interface）
  - 一個介面可以繼承自多個介面

介面延伸的示意圖

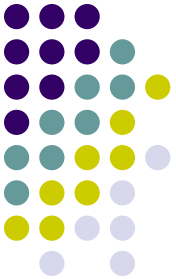
- 下圖中，iShape是父介面，透過關鍵字extends衍生出iShape2D與iShape3D子介面：



- 介面延伸的語法：

介面延伸的語法

```
interface 子介面名稱 extends 父介面名稱 1, 父介面名稱 2,...{
   ... ...
}
```



介面延伸的範例 (1/2)

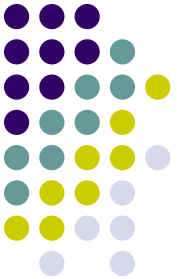
- 下面是介面延伸的範例：

```
01  // Ch11_7, 介面的延伸
02  interface iShape{                          // 定義 iShape 介面
03     final double PI=3.14;
04     abstract void setColor(String str);
05  }
06
07  interface iShape2D extends iShape{     // 定義 iShape2D 介面, 繼承自 iShape
08     abstract void area();
09  }
10
11  class Circle implements iShape2D{      // 實作 iShape2D 介面
12     double radius;
13     String color;
14
15     public Circle(double r){
16        radius=r;
17     }
```

執行結果：

```
color=Blue
area=12.56
```

接續下一頁



介面延伸的範例 (2/2)

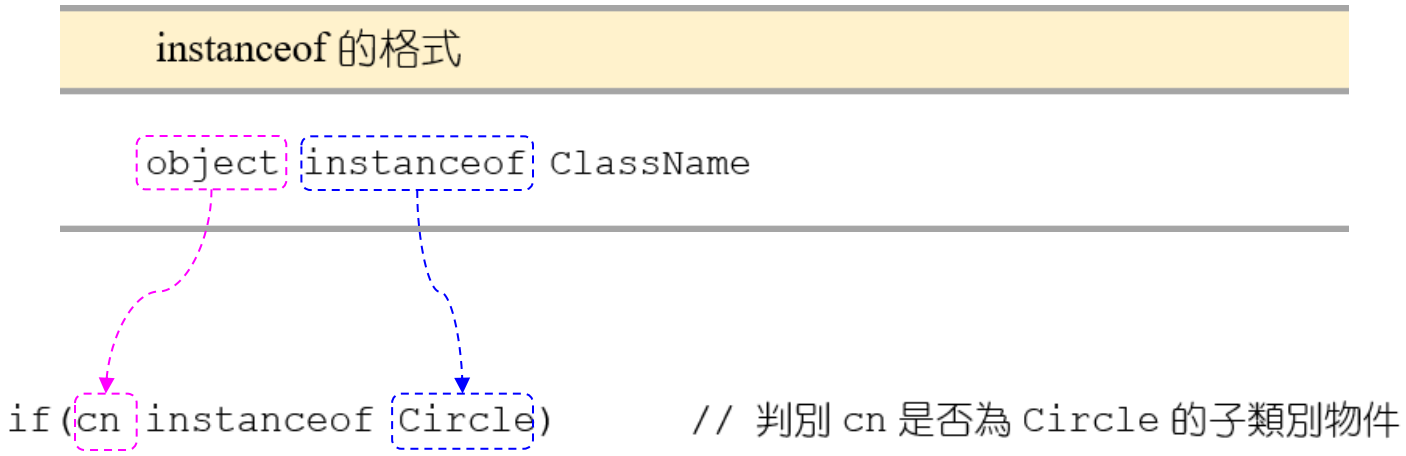
```
18      public void setColor(String str) {   // 定義 iShape 介面的 setColor()
19         color=str;
20         System.out.println("color="+color);
21      }
22      public void area(){                  // 定義 iShape2D 介面裡的 area()
23         System.out.println("area="+PI*radius*radius);
24      }
25   }
26   public class Ch11_7{
27      public static void main(String args[]){
28         Circle c1;
29         c1=new Circle(2.0);
30         c1.setColor("Blue");              // 呼叫 setColor()
31         c1.area();                        // 呼叫 area()
32      }
33   }
```

執行結果：

```
color=Blue
area=12.56
```

instanceof的格式

- instanceof運算子可測試物件與某個類別（class）或介面（interface）是否有繼承關係
  - 傳回值為布林值



instanceof運算子的使用(1/3)

```
01  // Ch11_8, instanceof 運算子的使用
02  class Circle { }
03  class Coin extends Circle { }          // Coin 繼承 Circle 類別
04  public class Ch11_8 extends Coin{      // Ch11_8 繼承 Coin 類別
05     public static void main(String args[]){
06        boolean status;
07        Coin cn=new Coin();
08        Circle c1=new Circle();
09        Ch11_8 myobj=new Ch11_8();
10        Coin carr[]=new Coin[5];
11
12        // 判別 c1 是否為 Coin 類別或其子類別物件
13        status=(c1 instanceof Coin);
14        System.out.println("c1 instanceof CCoin? " + status);
15
16        // 判別 myobj 是否為 Circle 類別或其子類別物件
17        status=(myobj instanceof Circle);
18        System.out.println("myobj instanceof Circle? " + status);
19
```

執行結果：

```
c1 instanceof CCoin? false
myobj instanceof Circle? true
cn instanceof Ch11_8? false
cn instanceof Circle? true
cn instanceof Coin? true
carr instanceof Object? true
```

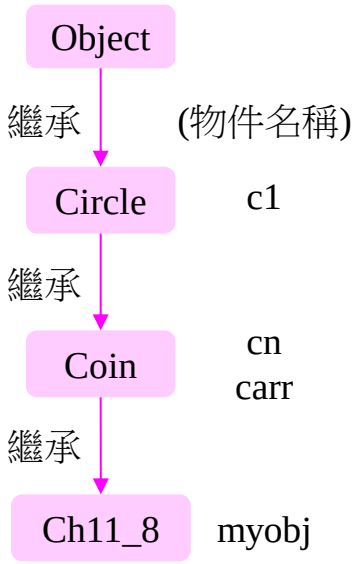

接續下一頁

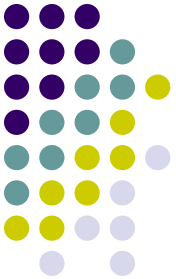
instanceof運算子的使用(2/3)

```
20        // 判別 cn 是否為 Ch11_8 類別或其子類別物件
21        status=(cn instanceof Ch11_8);
22        System.out.println("cn instanceof Ch11_8? "+ status);
23
24        // 判別 cn 是否為 Circle 類別或其子類別物件
25        status=(cn instanceof Circle);
26        System.out.println("cn instanceof Circle? " + status);
27
28        // 判別 cn 是否為 Coin 類別或其子類別物件
29        status=(cn instanceof Coin);
30        System.out.println("cn instanceof Coin? " + status);
31
32        // 判別陣列是否為 Object 類別或其子類別物件
33        status=(carr instanceof Object);
34        System.out.println("carr instanceof Object? " + status);
35
36        // 判別 c1 是否為 String 類別或其子類別物件
37        // status=(c1 instanceof String);
38        // System.out.println("c1 instanceof String? "+ status);
39    }
40  }
```

執行結果：

```
c1 instanceof CCoin? false
myobj instanceof Circle? true
cn instanceof Ch11_8? false
cn instanceof Circle? true
cn instanceof Coin? true
carr instanceof Object? true
```





instanceof運算子的使用(3/3)

- 將第37~38行的註解拿掉並加以編譯，會出現下面的錯誤訊息：

```
"Incompatible conditional operand types Circle and String",
```

- 這是說c1與String類別的型態不合，無法用instanceof做比較，在編譯時即會出現錯誤訊息


-The End-