



第十一章 抽象類別與介面

本章學習目標

- ✚ 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





11.1 抽象類別

抽象類別的目的是要依據它的格式來修改並建立新的類別。

11.1.1 定義抽象類別

定義抽象類別的語法如下：

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

格式 11.1.1
抽象類別的定義格式

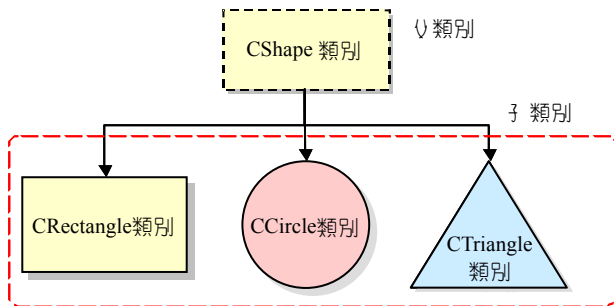
} 定義一般 method

— 定義抽象 method。注意於抽象 method 裡，沒有定義處理的方式



11.1.2 抽象類別的實作

例如，想設計一個抽象類別 **CShape**，依據此類別可用來衍生出圓形、正方形與三角形等幾何形狀的類別：





下面的C類別程式碼，即是抽象類別 CShape 的內容：

```
01 // 定義抽象類別 CShape
02 abstract class CShape // 定義抽象類別 CShape
03 {
04     protected String color; // 資料成員
05     public void setColor(String str) // 一般 method, 用來設定何形狀的顏色
06     {
07         color=str;
08     }
09     public abstract void show(); // 抽象函數, 在此沒有定義處理方式
10 }
```



下面的程式碼是以下類別 `CCircle` 為例來撰寫的：

```
01 // 定義由抽象類別 CShape 而衍生出的子類別 CCircle
02 class CCircle extends CShape // 定義子類別 CCircle
03 {
04     protected double radius; // 資料成員
05     public CCircle(double r) // 建構元
06     {
07         radius=r;
08     }
09     public void show()
10     {
11         System.out.print("color="+color+", ");
12         System.out.println("area="+3.14*radius*radius);
13     }
14 }
```

} 在此處明確定義
show()的處理方式



app11_1 是抽象類別實作的完整範例。

```
01 // app11 1, 抽象類別的實例
02 abstract class CShape // 定義抽象類別 CShape
03 {
04     protected String color; // 資料成員
05     public void setColor(String str) // 一般的函數
06     {
07         color=str;
08     }
09     public abstract void show(); // 抽象函數, 只有宣告, 但沒有定義處理方式
10 }
11 class CRectangle extends CShape // 定義子類別 CRectangle
12 {
13     protected int width,height;
14     public CRectangle(int w,int h)
15     {
16         width=w;
17         height=h;
18     }
19     public void show() // 明確定義繼承自抽象類別的 Show() method
20     {
21         System.out.print("color="+color+", ");
22         System.out.println("area="+width*height);
23     }
24 }
```

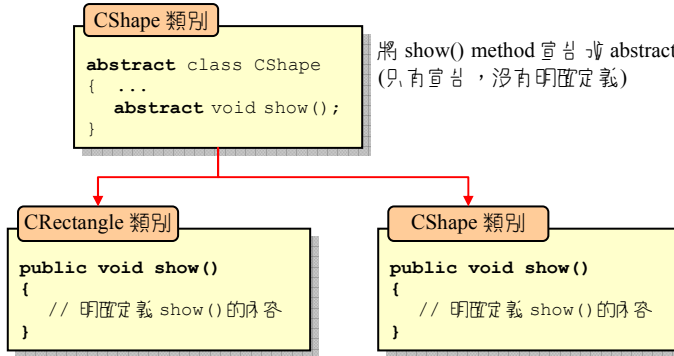


```
25 class CCircle extends CShape // 定義子類別CCircle
26 {
27     protected double radius;
28     public CCircle(double r)
29     {
30         radius=r;
31     }
32     public void show() // 明確定義繼承自抽象類別的Show() method
33     {
34         System.out.print("color="+color+", ");
35         System.out.println("area="+3.14*radius*radius);
36     }
37 }
38 public class app11_1
39 {
40     public static void main(String args[])
41     {
42         CRectangle rect=new CRectangle(5,10);
43         rect.setColor("Yellow");
44         rect.show();
45
46         CCircle cir=new CCircle(2.0);
47         cir.setColor("Green");
48         cir.show();
49     }
50 }
```

```
/* app11_1 OUTPUT---
color=Yellow, area=50
color=Green, area=12.56
-----*/
```



CShape、CCircle 與 CRectangle 的 show() method 之間的關係圖：





11.1.3 用抽象類別型態的變數來建立物件

利用抽象類別的變數來存取抽象類別物件的成員

app11_2 是以抽象類別型態的變數建立物件的範例：

```
01 // app11_2, 用抽象類別型態的變數來建立物件
02 // 將 app11_1 的 CShape 類別的定義放在這兒
03 // 將 app11_1 的 CRectangle 類別的定義放在這兒
04 // 將 app11_1 的 CCircle 類別的定義放在這兒
05 public class app11_2
06 {
07     public static void main(String args[])
08     {
09         CShape shape1=new CRectangle(5,10);
10         shape1.setColor("Yellow");
11         shape1.show();
12
13         CShape shape2=new CCircle(2.0);
14         shape2.setColor("Green");
15         shape2.show();
16     }
17 }
```

```
/* app11_2 OUTPUT-----
color=Yellow, area=50
color=Green, area=12.56
-----*/
```

以類別 CShape 建立物件 shape1，並用它來存取子類別 CRectangle 的成員。

以類別 CShape 建立物件 shape2，並用它來存取子類別 CCircle 的成員。



利用 \cup 類別的陣列變數來存取子類別物件的成員

當所建立的物件較多時，可利用 \cup 類別的陣列變數來存取子類別物件的成員：

- (1) 先建立 \cup 類別的陣列變數
- (2) 利用陣列元素建立子類別的物件，並以它來存取子類別的內容



app11_3 改寫了 app11_2 :

```
01 // app11_3, 利用 C 類別的陣列變數來存取子類別的內容
02 // 將 app11_1 的 CShape 類別的定義放在這兒
03 // 將 app11_1 的 CRectangle 類別的定義放在這兒
04 // 將 app11_1 的 CCircle 類別的定義放在這兒
05 public class app11_3
06 {
07     public static void main(String args[])
08     {
09         CShape shape[];           // 宣告 CShape 型態的陣列變數
10         shape=new CShape[2];      // 產生兩個 CShape 抽象類別型態的變數
11
12         shape[0]=new CRectangle(5,10); } 利用陣列變數 shape[0]
13         shape[0].setColor("Yellow");    } 建立物件,並存取子類別
14         shape[0].show();                } 的成員
15
16         shape[1]=new CCircle(2.0);      } 利用陣列變數 shape[1]
17         shape[1].setColor("Green");     } 建立物件,並存取子類別
18         shape[1].show();                } 的成員
19     }
20 }
```

```
/* app11_3 OUTPUT-----
color=Yellow, area=50
color=Green, area=12.56
-----*/
```



11.1.4 使用抽象類別的注意事項

- ✓ 抽象類別不能用來直接產生物件。
- ✓ 在抽象類別內可以定義建構元，以供子類別的建構元呼叫。
- ✓ 定義在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`。



11.2 介面的使用

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（即設定初值）。
- (2) 介面裡的 **method** 必須全部都宣稱為 **abstract**。



下面的範例定義一個 `iShape2D` :

```
01 // 定義 iShape2D 介面
02 interface iShape2D
03 {
04     final double pi=3.14; // 資料成員 - 定義初值化
05     abstract void area(); // 抽象函數，不需要定義處理方式
06 }
```

程式碼可改寫如下：

```
01 // 定義 iShape2D 介面，省略 final 與 abstract 關鍵字
02 interface iShape2D
03 {
04     double pi=3.14; // 省略 final 關鍵字
05     void area(); // 省略 abstract 關鍵字
06 }
```



利用介面 A 打造新的類別 B 的過程，稱之為以類別 B 實作介面 A，或簡稱介面的實作（implementation）。

介面實作的語法：

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

格式 11.2.2

以類別實作介面的語法



下面是以 `CCircle` 類別實作 `iShape2D` 介面的範例：

```
01 // 介面的實作
02 class CCircle implements iShape2D // 以CCircle類別實作 iShape2D 介面
03 {
04     double radius;
05     public CCircle(double r) // 建構元
06     {
07         radius=r;
08     }
09     public void area() // 定義 area() 的處理方式
10     {
11         System.out.println("area="+pi*radius*radius);
12     }
13 }
```

以類別 `CCircle` 來實作
介面 `iShape2D`



app11_4 是以 iShape2D 介面實作 CCircle 與 CRectangle 二個類別的範例：

```
01 // app11_4, 介面的實作範例
02 interface iShape2D // 定義介面
03 {
04     final double pi=3.14;
05     abstract void area();
06 }
07
08 class CRectangle implements iShape2D// 以CRectangle類別實作 iShape2D 介面
09 {
10     int width,height;
11     public CRectangle(int w,int h)
12     {
13         width=w;
14         height=h;
15     }
16     public void area() // 定義 area() 的處理方式
17     {
18         System.out.println("area="+width*height);
19     }
20 }
21
22 class CCircle implements iShape2D// 以CCircle類別實作 iShape2D 介面
```



```
23  {
24      double radius;
25      public CCircle(double r)
26      {
27          radius=r;
28      }
29      public void area()    // 定義 area() 的處理方式
30      {
31          System.out.println("area="+pi*radius*radius);
32      }
33  }
34
35  public class app11_4
36  {
37      public static void main(String args[])
38      {
39          CRectangle rect=new CRectangle(5,10);
40          rect.area();    // 呼叫 CRectangle 類別裡的 area() method
41
42          CCircle cir=new CCircle(2.0);
43          cir.area();    // 呼叫 CCircle 類別裡的 area() method
44      }
45  }
```

```
/* app11_4 OUTPUT---
```

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

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



app11_5 是利用透過介面型態的變數來存取物件的範例。

```
01 // app11_5,透過介面型態的變數來存取物件
02 // 將 app11_4 的 iShape 介面的定義放在這兒
03 // 將 app11_4 的 CRectangle 類別的定義放在這兒
04 // 將 app11_4 的 CCircle 類別的定義放在這兒
05 public class app11_5
06 {
07     public static void main(String args[])
08     {
09         iShape2D var1,var2;           // 宣告介面型態的變數
10         var1=new CRectangle(5,10);    // 將介面型態的變數 var1 指向新建的物件
11         var1.area();                  // 透過 var1 呼叫 show() method
12
13         var2=new CCircle(2.0);        // 將介面型態的變數 var2 指向新建的物件
14         var2.area();                  // 透過 var2 呼叫 show() method
15     }
16 }
```

/* app11_5 OUTPUT--

area=50

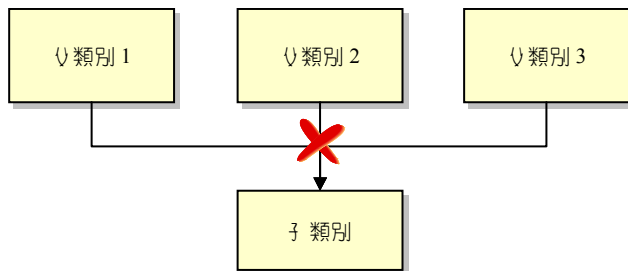
area=12.56

-----*/



11.3 多重繼承

Java 並不允許多個父類別的繼承：



用類別來實作兩個以上的介面，即可實現多重繼承。



將類別和兩個以上的介面實作在一起的語法如下：

```
class 類別名稱 implements 介面 1, 介面 2, ...  
{  
    ... ..  
}
```

格式 11.3.1

實作兩個以上的介面



app11_6 是以兩個介面 `iShape2D` 和 `iColor` 來實作 `CCircle` 類別的範例。

```
01 // app11_6, 用 CCircle 類別實作兩個以上的介面
02 interface iShape2D // 定義 iShape2D 介面
03 {
04     final double pi=3.14;
05     abstract void area();
06 }
07
08 interface iColor // 定義 iColor 介面
09 {
10     abstract void setColor(String str);
11 }
12
13 class CCircle implements iShape2D,iColor // 實作 iShape2D 與 iColor 介面
14 {
15     double radius;
16     String color;
17     public CCircle(double r)
18     {
19         radius=r;
20     }
21     public void setColor(String str) // 定義 iColor 介面裡的 setColor()
```



```
22     {
23         color=str;
24         System.out.println("color="+color);
25     }
26     public void area()    // 定義 iShape2D 介面裡的 area() method
27     {
28         System.out.println("area="+pi*radius*radius);
29     }
30 }
31 public class appl1_6
32 {
33     public static void main(String args[])
34     {
35         CCircle cir;
36         cir=new CCircle(2.0);
37         cir.setColor("Blue");    // 呼叫 setColor() method
38         cir.area();              // 呼叫 show() method
39     }
40 }
```

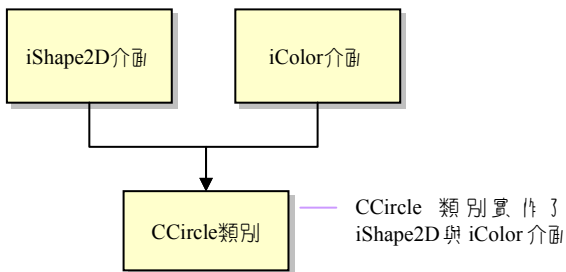
/* app11_6 OUTPUT---

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

-----*/



透過類別與介面的實作，可達到多重繼承的目的，如下圖所示：



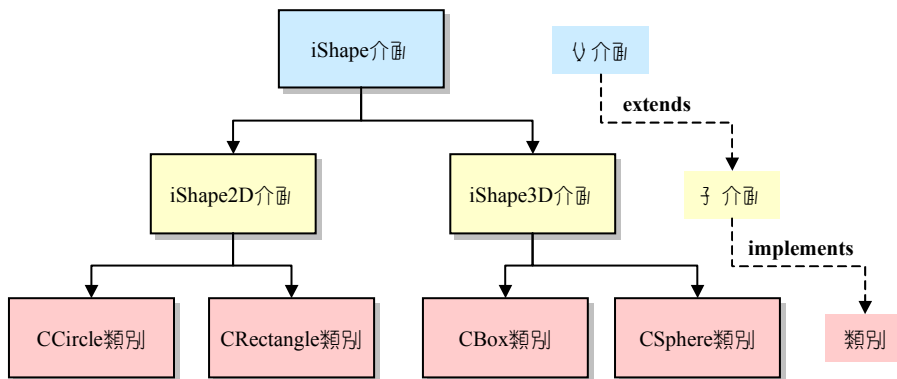


11.4 介面的延伸

- ✓ 介面可透過繼承的技術來衍生出新的介面。
- ✓ 原來的介面稱為基座介面（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）。



下圖中的 iShape 是 U 介面，透過關鍵字 extends 衍生出 iShape2D 與 iShape3D 子介面：





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, ...  
{  
    ... ..  
}
```

格式 11.4.1

介面延伸的語法



下面是介面延伸的範例：

```
01 // app11_7, 介面的延伸
02 interface iShape // 定義 iShape 介面
03 {
04     final double pi=3.14;
05     abstract void setColor(String str);
06 }
07
08 interface iShape2D extends iShape // 定義 iShape2D 介面, 繼承自 iShape
09 {
10     abstract void area();
11 }
12
13 class CCircle implements iShape2D // 實作 iShape2D 介面
14 {
15     double radius;
16     String color;
17
18     public CCircle(double r)
19     {
20         radius=r;
21     }
22     public void setColor(String str) // 定義 iShape 介面的 setColor()
23     {
```



```
24     color=str;
25     System.out.println("color="+color);
26 }
27 public void area()           // 定義 iShape2D 介面裡的 area()
28 {
29     System.out.println("area="+pi*radius*radius);
30 }
31 }
32 public class app11_7
33 {
34     public static void main(String args[])
35     {
36         CCircle cir;
37         cir=new CCircle(2.0);
38         cir.setColor("Blue"); // 呼叫 setColor() method
39         cir.area();           // 呼叫 area() method
40     }
41 }
```

/* app11_7 OUTPUT--

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

-----*/



-The End-