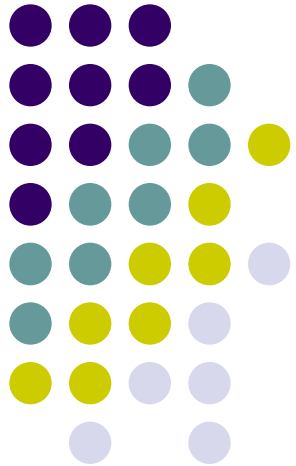
第十七章
虛擬函數與抽象類別

認識虛擬函數與指向基底類別的指標

認識抽象類別與泛虛擬函數的關係

熟悉抽象類別應用於多層繼承的方式

使用虛擬解構元



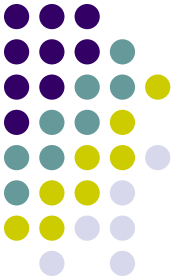
虛擬函數 (1/2)

- 當函數的執行內容現在無法定義，或現在已定義，將來卻可能會被改變，就可以將函數以虛擬函數的方式撰寫
- 錯誤的範例 - 由早期連結所引起

```
01  // prog17_1, 錯誤的範例，未使用虛擬函數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CWin                                  // 定義 CWin 類
06  {
07     protected:
08       char id;
09       int width, height;
10     public:
11       CWin(char i='D',int w=10, int h=10)     // 父類別的建構元
12       {
13          id=i;
14          width=w;
15          height=h;
16       }
```

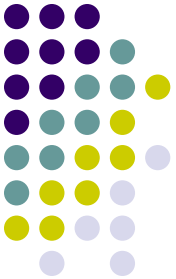
```
/* prog17_1 OUTPUT------

Window A, area = 5600
Window B, area = 3000
-----------------------*/
```

虛擬函數 (2/2)



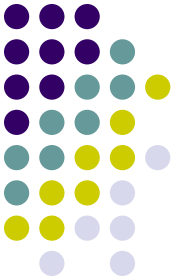
```
17        void show area()                  // 父類別的 show area() 函數
18        {
19           cout << "Window " << id << ", area = " << area() << endl;
20        }
21        int area()                        // 父類別的 area() 函數
22        {
23           return width*height;
24        }
25    };
26
27    class CMiniWin : public CWin          // 定義子類別 CMiniWin
28    {
29       public:
30         CMiniWin(char i,int w,int h):CWin(i,w,h){}    // 子類別的建構元
31
32         int area()                       // 子類別的 area() 函數
33         {
34            return (int)(0.8*width*height);
35         }
36    };
37
38    int main(void)
39    {
40       CWin win('A',70,80);              // 建立父類別物件 win
41       CMiniWin m win('B',50,60);        // 建立子類別物件 m win
42
43       win.show area();          // 以父類別物件 win 呼叫 show area() 函數
44       m win.show area();        // 以子類別物件 m win 呼叫 show area() 函數
45
46       system("pause");
47       return 0;
48    }
```

```
/* prog17_1 OUTPUT------

Window A, area = 5600
Window B, area = 3000

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



認識虛擬函數

- 編譯時便把父類別裡的show_area() 和area() 函數連結在一起編譯，這種函數連結的方式稱為早期連結（early binding）
- 虛擬函數可以與呼叫它的函數進行晚期連結（late binding），也就是於程式執行時才由當時的情況來決定是哪一個函數被呼叫，而非在編譯時就把函數配對

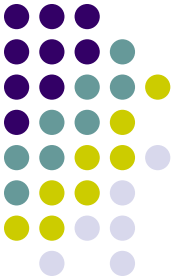


修正使用虛擬函數的錯誤 (1/2)

- 下面的程式碼是使用虛擬函數來修正錯誤

```
01  // prog17_2, 使用虛擬函數來修正錯誤
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CWin                              // 定義 CWin 類別，在此為父類別
06  {
07     protected:
08       char id;
09       int width, height;
10     public:
11       CWin(char i='D',int w=10, int h=10)
12       {
13          id=i;
14          width=w;
15          height=h;
16       }
17        void show area()                  // 父類別的 show area()函數
18        {
19           cout << "Window " << id << ", area = " << area() << endl;
20        }
```

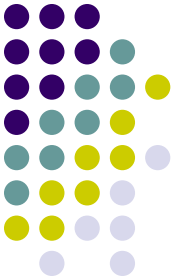
```
/* prog17_2 OUTPUT-----

Window A, area = 5600
Window B, area = 2400

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



修正使用虛擬函數的錯誤 (2/2)

```
21        virtual int area()                  // 父類別的 area()函數
22        {
23           return width*height;
24        }
25  };
26
27  class CMiniWin : public CWin              // 定義子類別 CMiniWin
28  {
29     public:
30       CMiniWin(char i,int w,int h):CWin(i,w,h){}  // 子類別的建構元
31
32        virtual int area()                  // 子類別的 area()函數
33        {
34           return (int)(0.8*width*height);
35        }
36  };
37
38  // 將 prog17_1 的主函數 main()放在這兒
```

```
/* prog17_2 OUTPUT-----

Window A, area = 5600
Window B, area = 2400

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

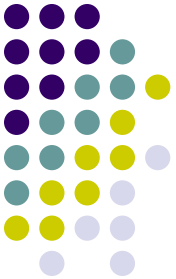


指向基底類別物件的指標 (1/2)

- 指向基底類別的指標，也可指向衍生類別所建立的物件
- 下面的範例是指向基底類別物件的指標之應用

```
01  // prog17_3, 簡單的應用–指向基底類別物件的指標
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  // 將 prog17_2 的 CWin 類別放在這兒
06  // 將 prog17_2 的 CMiniWin 類別放在這兒
07
08  int main(void)
09  {
10     CWin win('A',70,80);
11     CMiniWin m win('B',50,60); // 建立子類別的物件
12
```

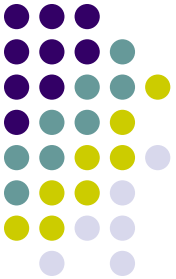
```
/* prog17_3 OUTPUT-----
Window A, area = 5600
Window B, area = 2400
----------------------*/
```



指向基底類別物件的指標 (2/2)

```
13      CWin *ptr=NULL;                 // 宣告指向基底類別(父類別)的指標
14
15      ptr=&win;                       // 將ptr 指向父類別的物件 win
16      ptr->show area();               // 以ptr 呼叫 show area()函數
17
18      ptr=&m win;                     // 將ptr 指向子類別的物件 m win
19      ptr->show area();               // 以ptr 呼叫 show area()函數
20
21      system("pause");
22      return 0;
23  }
```

```
/* prog17_3 OUTPUT-----

Window A, area = 5600
Window B, area = 2400

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



錯誤的示範 (1/4)

- 錯誤示範 - 指向由動態記憶體配置之物件的指標

```
01  // prog17_4, 錯誤示範，指向由動態記憶體配置之物件的指標
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  // 將 prog17_3 的 CWin 類別放在這兒
06  // 將 prog17_3 的 CMiniWin 類別放在這兒
07
08  int main(void)
09  {
10     CWin *ptr=new CWin('A',70,80);  // 設定 ptr 指向 CWin 類別的物件
11     CMiniWin m_win('B',50,60);
12
13     ptr->show_area();               // 以 ptr 呼叫 show_area()函數
14
15     ptr=&m_win;                     // 將 ptr 指向子類別的物件 m_win
16     ptr->show_area();               // 以 ptr 呼叫 show_area()函數
17
18     delete ptr;                     // 清除 ptr 所指向的記憶空間
19
20     system("pause");
21     return 0;
22  }
```

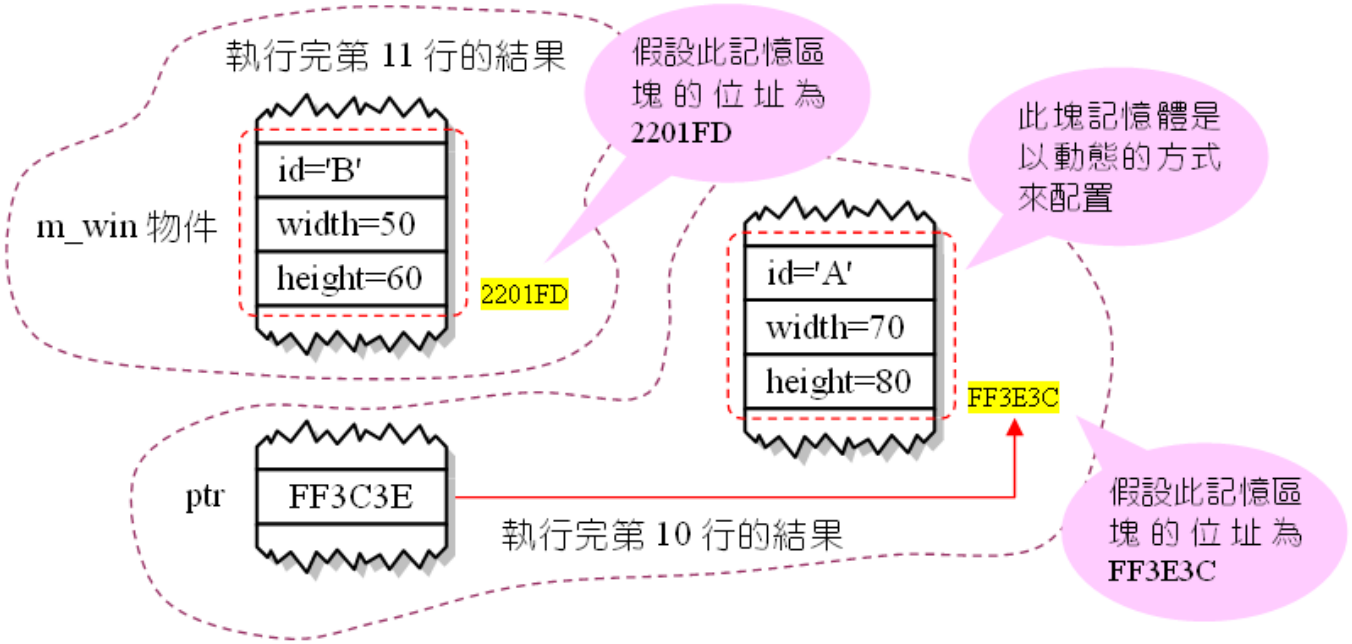
```
/* prog17_4 OUTPUT----

Window A, area = 5600
Window B, area = 2400

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

錯誤的示範 (2/4)

- 執行結果的圖說

```
10      CWin *ptr=new CWin('A',70,80);  // 設定 ptr 指向 CWin 類別的物件
11      CMiniWin m win('B',50,60);
```

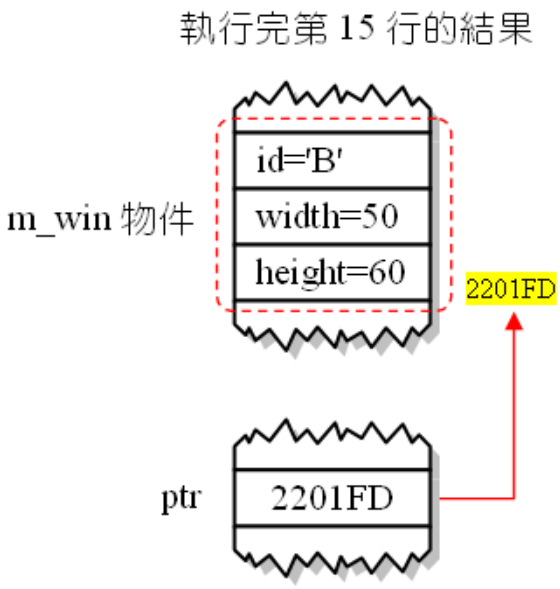
執行完 10~11 行之後的結果

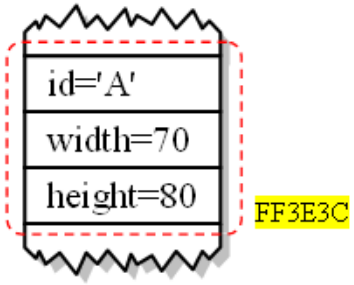


錯誤的示範 (3/4)

```
15      ptr=&m_win;    // 將 ptr 指向子類別的物件 m_win
```

執行完第 15 行之後的結果



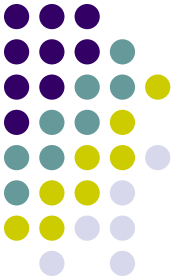


執行完第 15 行後，
沒有任何指標指向
此記憶區塊

- 讀者可試著在第19行加上下面的敘述

```
ptr->show_area();
```

但ptr所指向的物件並未被銷毀

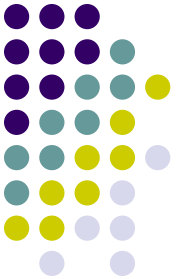


錯誤的示範 (4/4)

- 下面的範例修正prog17_4的錯誤

```
01  // prog17_5, 修正 prog17_4 的錯誤
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  // 將 prog17_3 的 CWin 類別放在這兒
06  // 將 prog17_3 的 CMiniWin 類別放在這兒
07
08  int main(void)
09  {
10     CWin *ptr=new CWin('A',70,80);  // 設定 ptr 指向 CWin 類別的物件
11     CMiniWin m_win('B',50,60);
12
13     ptr->show_area();               // 以 ptr 呼叫 show_area()函數
14     delete ptr;                     // 先釋放 ptr 所指向的記憶空間
15
16     ptr=&m_win;                     // 再將 ptr 指向子類別的物件 m_win
17     ptr->show_area();               // 以 ptr 呼叫 show_area()函數
18
19     system("pause");
20     return 0;
21  }
```

```
/* prog17_5 OUTPUT----

Window A, area = 5600
Window B, area = 2400

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



泛虛擬函數與抽象類別

- 「泛虛擬函數」（pure virtual function）

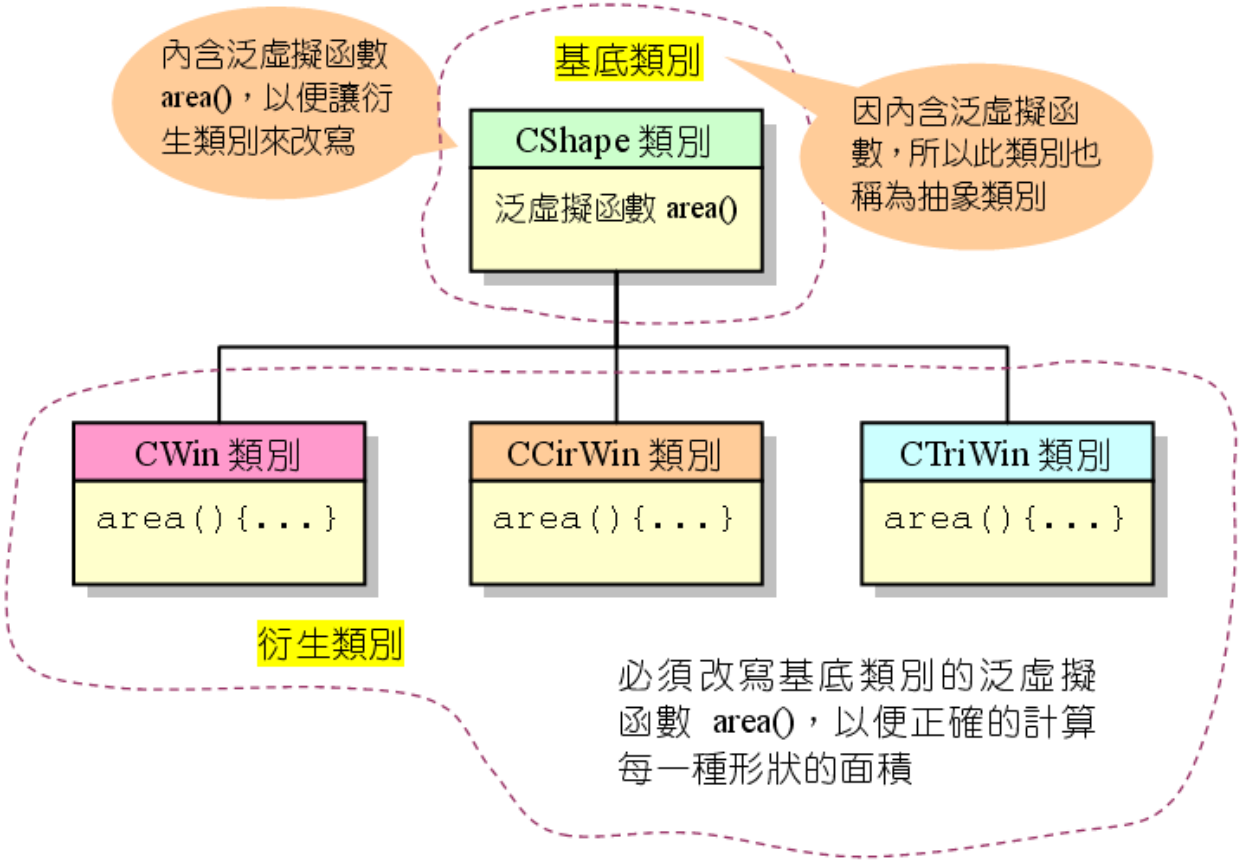
  在基底類別裡撰寫虛擬函數，使得子類別必須藉由改寫的技術重新定義虛擬函數，具有這個特性的虛擬函數稱為泛虛擬函數

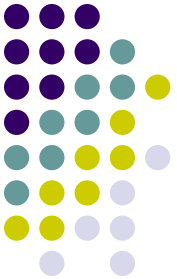
- 「抽象類別」（abstract class）

  包含有泛虛擬函數的類別稱為抽象類別

定義泛虛擬函數 (1/2)

- 下圖是由抽象類別CShape衍生出子類別的示意圖



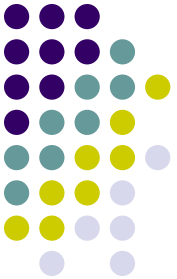


定義泛虛擬函數 (2/2)

- 下面為CShape基底抽象類別程式碼

```
01  class CShape                    // 定義抽象類別 CShape
02  {
03     public:
04        virtual int area()=0;     // 定義 area()，並令設之為 0 代表它是泛虛擬函數
05
06        void show area()          // 定義成員函數 show area()
07        {
08           cout << "area = " << area() << endl;
09        }
10  };
```

- 抽象類別有點類似「範本」的作用，其目的是要讓您依據它的格式來修改並建立新的類別

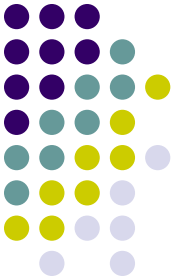


抽象類別的實作

- 下面的程式碼是以子類別CWin為例來撰寫的

```
01  class CWin : public CShape  // 定義由 CShape 類別所衍生出的子類別 CWin
02  {
03     protected:
04       int width, height;
05
06     public:
07       CWin(int w=10, int h=10)        // CWin()建構元
08       {
09          width=w;
10          height=h;
11       }
12       virtual int area()
13       {
14          return width*height;
15       }
16  };
```

在此處明確定義 area()的處理方式

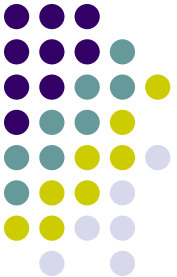


抽象類別的完整實例 (1/4)

- prog17_6是抽象類別實作的完整實例

```
01  // prog17_6, 抽象類別的實作
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CShape                          // 定義抽象類別 CShape
06  {
07    public:
08      virtual int area()=0;  // 定義 area()，並令之為 0 來代表它是泛虛擬函數
09
10      void show area()           // 定義成員函數 show area()
11      {
12         cout << "area = " << area() << endl;
13      }
14  };
15
16  class CWin : public CShape     // 定義由 CShape 所衍生出的子類別 CWin
17  {
18    protected:
19      int width, height;
20
```

```
/* prog17_6 OUTPUT-------

area = 3000
CCirWin 物件的面積 = 31400
------------------------*/
```

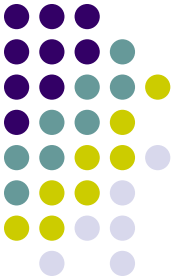


抽象類別的完整實例 (2/4)

```
21      public:
22        CWin(int w=10, int h=10)  // CWin()建構元
23        {
24           width=w;
25           height=h;
26        }
27        virtual int area()
28        {
29           return width*height;
30        }
31    };
32
33    class CCirWin : public CShape    // 定義由CShape所衍生出的子類別CCirWin
34    {
35       protected:
36         int radius;
37
38       public:
39         CCirWin(int r=10)           // CCirWin()建構元
40         {
41            radius=r;
42         }
```

在此處明確定義 area()的處理方式

```
/* prog17_6 OUTPUT-------
area = 3000
CCirWin 物件的面積 = 31400
-----------------------*/
```



抽象類別的完整實例 (3/4)

```
43        virtual int area()
44        {
45           return (int) (3.14*radius*radius);
46        }
47        void show area()
48        {
49           cout << "CCirWin 物件的面積 = " << area() <<endl;
50        }
51    };
52
53    int main(void)
54    {
55       CWin win1(50,60);            // 建立 CWin 類別的物件 win1
56       CCirWin win2(100);           // 建立 CCinWin 類別的物件 win2
57
58       win1.show area();            // 用 win1 呼叫 show area();
59       win2.show area();            // 用 win2 呼叫 show area();
60
61       system("pause");
62       return 0;
63    }
```

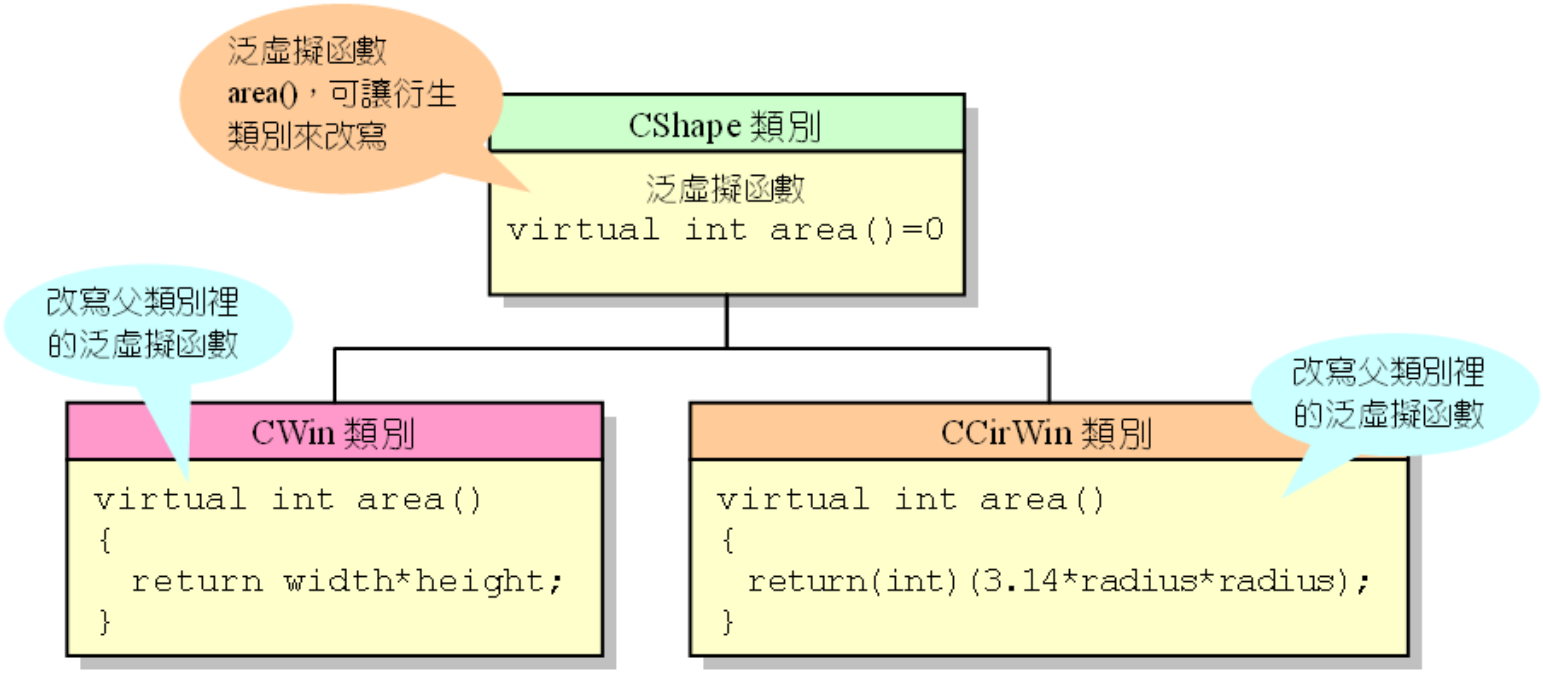
在此處明確定義 area()的處理方式

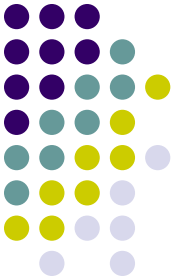
改寫父類別的 show_area()函數

```
/* prog17_6 OUTPUT-------

area = 3000
CCirWin 物件的面積 = 31400
-----------------------*/
```

抽象類別的完整實例 (4/4)

- 下圖是抽象類別CShape內，泛虛擬函數的實作示意圖





使用抽象類別的注意事項

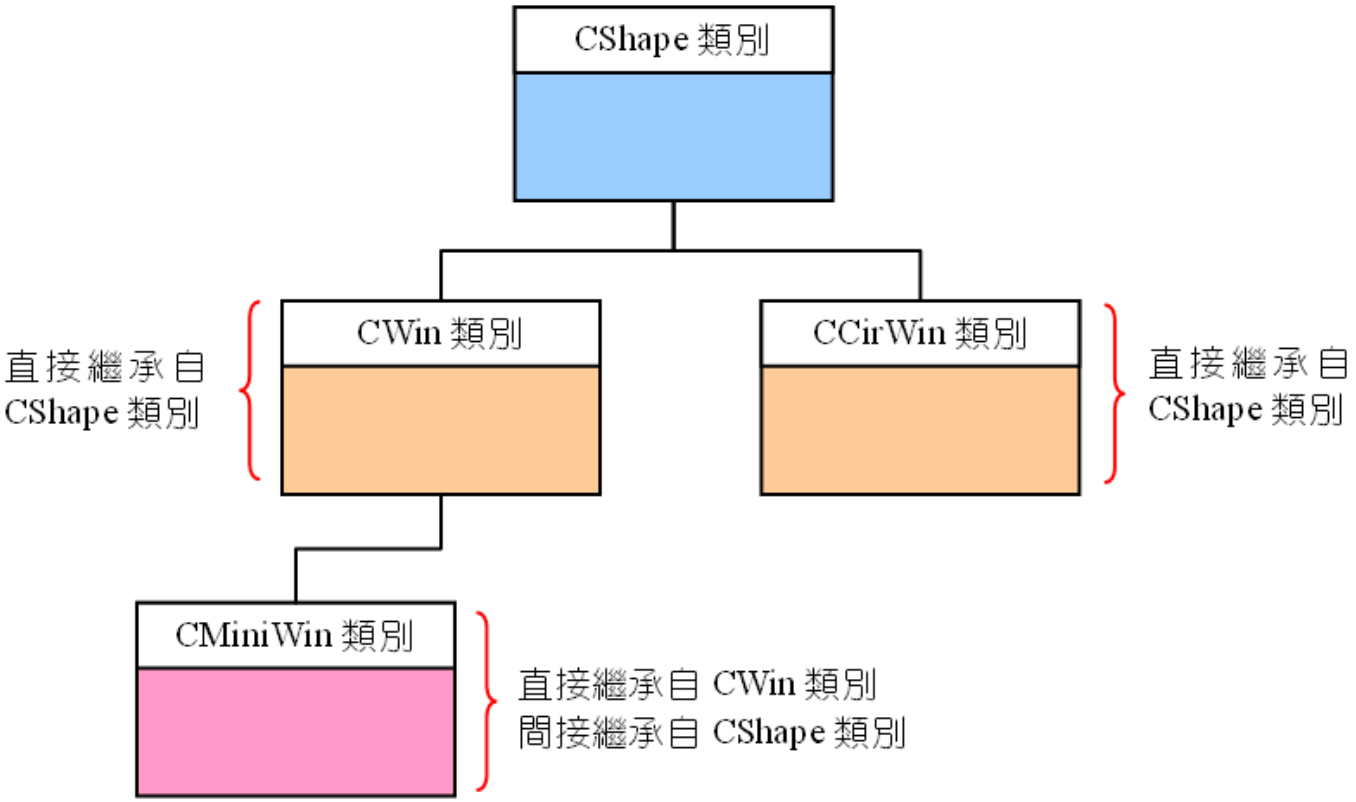
- 抽象類別不能用來直接產生物件

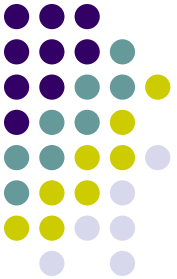
  您不能撰寫如下的程式碼

```
int main(void)
{
   CShape shape;    // 錯誤，不能用抽象類別來產生物件 shape
   ...
}
```

繼承的關係圖

- 下圖為直接繼承與間接繼承的關係圖



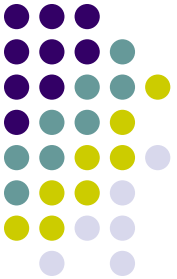


抽象類別與多層繼承 (1/4)

- 抽象類別也可以應用於多層繼承的架構

```
01  // prog17_7, 抽象類別於多層繼承的應用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CShape                        // 定義抽象類別 CShape
06  {
07    public:
08       virtual int area()=0;          // 定義 area() 為泛虛擬函數
09
10       void show area()               // 定義成員函數 show area()
11       {
12          cout << "area = " << area() << endl;
13       }
14  };
15
16  class CWin : public CShape          // 定義由 CShape 所衍生出的子類別 CWin
17  {
18    protected:
19      int width, height;
20
```

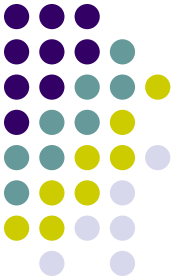
```
/* prog17_7 OUTPUT------
CWin 物件的面積 = 3000
CCirWin 物件的面積 = 31400
CMiniWin 物件的面積 = 1500
----------------------*/
```



抽象類別與多層繼承 (2/4)

```
21      public:
22        CWin(int w=10, int h=10)  // CWin()建構元
23        {
24           width=w;
25           height=h;
26        }
27        virtual int area()
28        {
29           return width*height;
30        }
31        void show area()
32        {
33           cout << "CWin 物件的面積 = " << area() << endl;
34        }
35    };
36
37    class CCirWin : public CShape  // 定義由CShape所衍生出的子類別CCirWin
38    {
39      protected:
40        int radius;
41
42      public:
43        CCirWin(int r=10)          // CCirWin()建構元
44        {
45           radius=r;
46        }
```

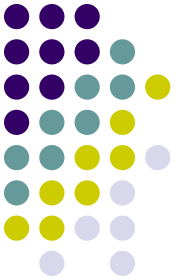
```
/* prog17_7 OUTPUT------
CWin 物件的面積 = 3000
CCirWin 物件的面積 = 31400
CMiniWin 物件的面積 = 1500
-----------------------*/
```



抽象類別與多層繼承 (3/4)

```
47        virtual int area()
48        {
49           return (int)(3.14*radius*radius);
50        }
51        void show area()
52        {
53           cout << "CCirWin 物件的面積 = " << area() << endl;
54        }
55    };
56
57    class CMiniWin : public CWin  // 定義由 CWin 所衍生出的子類別 CMiniWin
58    {
59       public:
60         CMiniWin(int w,int h):CWin(w,h){}          // 子類別的建構元
61
62         virtual int area()
63         {
64            return (int) (0.5*width*height);
65         }
66         void show area()
67         {
68            cout << "CMiniWin 物件的面積 = " << area() << endl;
69         }
70    };
71
```

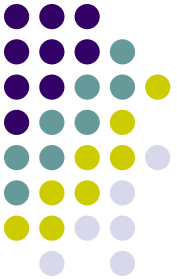
```
/* prog17_7 OUTPUT------
CWin 物件的面積 = 3000
CCirWin 物件的面積 = 31400
CMiniWin 物件的面積 = 1500
-----------------------*/
```



抽象類別與多層繼承 (4/4)

```
72  int main(void)
73  {
74     CWin win1(50,60);
75     CCirWin win2(100);
76     CMiniWin win3(50,60);
77
78     win1.show area();
79     win2.show area();
80     win3.show area();
81
82     system("pause");
83     return 0;
84  }
```

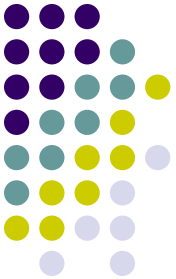
```
/* prog17_7 OUTPUT------
CWin 物件的面積 = 3000
CCirWin 物件的面積 = 31400
CMiniWin 物件的面積 = 1500
-----------------------*/
```



錯誤的範例 (1/4)

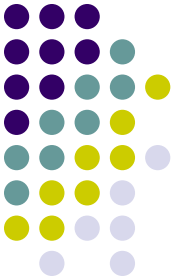
- prog17_8是使用解構元的錯誤範例

```
01  // prog17_8, 錯誤的範例,虛擬函數與解構元
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CShape                    // 定義抽象類別 CShape
06  {
07     public:
08        virtual int area()=0;     // 定義 area()為泛虛擬函數
09        void show area()
10        {
11           cout << "area = " << area() << endl;
12        }
13        ~CShape()                 // ~CShape() 解構元
14        {
15           cout << "~CShape()解構元被呼叫了..." << endl;
16           system("pause");
17        }
18  };
19
20  class CWin : public CShape     // 定義由 CShape 所衍生出的子類別 CWin
21  {
22     protected:
23       int width, height;
```



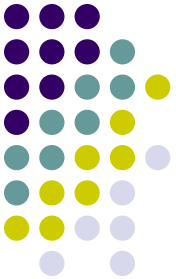
錯誤的範例 (2/4)

```
24
25      public:
26        CWin(int w=10, int h=10):width(w),height(h){} // CWin()建構元
27
28        virtual int area() {return width*height; }
29
30        void show area() {
31           cout << "CWin 物件的面積 = " << area() << endl;
32        }
33        ~CWin()                          // ~CWin() 解構元
34        {
35           cout << "~CWin()解構元被呼叫了..." << endl;
36           system("pause");
37        }
38  };
39
40  class CMiniWin : public CWin  // 定義由 CWin 所衍生出的子類別 CMiniWin
41  {
42     public:
43        CMiniWin(int w,int h):CWin(w,h){}       // CMiniWin()建構元
44
45        virtual int area() {
46           return (int) (0.5*width*height);
47        }
```



錯誤的範例 (3/4)

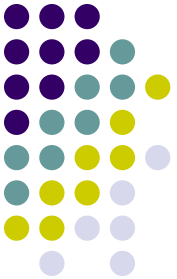
```
48        void show area(){
49           cout << "CMiniWin 物件的面積 = " << area() << endl;
50        }
51        ~CMiniWin()                    // ~CMiniWin() 解構元
52        {
53           cout << "~CMiniWin()解構元被呼叫了..." << endl;
54           system("pause");
55        }
56    };
57
58    int main(void)
59    {
60       CShape *ptr=new CWin(50,60);
61       ptr->show area();
62       cout << "銷毀 CWin 物件..." << endl;
63       delete ptr;
64       cout << endl;
65
66       ptr=new CMiniWin(50,50);
67       ptr->show area();
68       cout << "銷毀 CMiniWin 物件..." << endl;
69       delete ptr;
70       cout << endl;
71
```



錯誤的範例 (4/4)

```
72      CMiniWin m win(100,100);
73      m win.show area();
74
75      system("pause");
76      return 0;
77  }
```

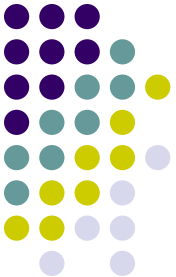
```
/* prog17_8 OUTPUT---------
area = 3000                          ——— 抽象類別 CShape 的 show_area()函數被呼叫了
銷毀 CWin 物件...
~CShape()解構元被呼叫了...           ——— 63 行的執行結果
請按任意鍵繼續 . . .

area = 1250                          ——— 抽象類別 CShape 的 show_area()函數被呼叫了
銷毀 CMiniWin 物件...
~CShape()解構元被呼叫了...           ——— 69 行的執行結果
請按任意鍵繼續 . . .

CMiniWin 物件的面積 = 5000
請按任意鍵繼續 . . .
~CMiniWin()解構元被呼叫了...
請按任意鍵繼續 . . .                 } 自動處理物件的銷毀，此時會先執行
~CWin()解構元被呼叫了...               自己的解構元再執行父類別的解構
請按任意鍵繼續 . . .                   元，最後再執行基底類別的解構元
~CShape()解構元被呼叫了...
請按任意鍵繼續 . . .
-------------------------*/
```



使用虛擬解構元 (1/2)

- prog17_9是使用虛擬解構元的範例

```
01  // prog17_9, 使用虛擬解構元
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CShape                          // 定義抽象類別 CShape
06  {
07     public:
08        virtual int area()=0;           // 定義 area()為泛虛擬函數
09        virtual void show area()        // 定義 show area()為虛擬函數
10        {
11           cout << "area = " << area() <<endl;
12        }
13        virtual ~CShape()               // 定義 ~CShape() 為虛擬解構元
14        {
15           cout << "~CShape()解構元被呼叫了..." << endl;
16           system("pause");
17        }
18  };
19
20  // 將prog17_8的CWin類別放在這兒
21  // 將prog17_8的CMiniWin類別放在這兒
22  // 將prog17_8的main()主程式放在這兒
```



使用虛擬解構元 (2/2)

```
/* prog17_9 OUTPUT---------
CWin 物件的面積 = 3000
銷毀 CWin 物件...
~CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
~CShape() 解構元被呼叫了...
請按任意鍵繼續 . . .

CMiniWin 物件的面積 = 1250
銷毀 CMiniWin 物件...
~CMiniWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
~CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
~CShape() 解構元被呼叫了...
請按任意鍵繼續 . . .

CMiniWin 物件的面積 = 5000
請按任意鍵繼續 . . .
~CMiniWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
~CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
~CShape() 解構元被呼叫了...
請按任意鍵繼續 . . .
-------------------------*/
```

銷毀 CWin 物件的，此時會先執行 ~CWin()解構元，再執行基底類別的解構元~CShape()

銷毀 CMiniWin 物件，此時會先執行自己的解構元再執行父類別的解構元，最後再執行基底類別的解構元

自動處理物件的銷毀，此時會先執行自己的解構元再執行父類別的解構元，最後再執行基底類別的解構元

- 如果程式碼裡有使用抽象類別，可把基底類別的解構元設為 virtual，如此可以確保解構元會正確地被呼叫以及釋放記憶空間


-The End-