第八章 繼承

課前指引

在本章中，我們將介紹類別的繼承，而在下一章中我們將介紹一種特殊的介面類別，對於一個Java程式設計師來說，是相當重要且基本的兩種技術，因為它包含了物件導向的三大特性－『封裝』、『繼承』、『多型』。

章節大綱

備註：可依進度點選小節

課前指引

- 就『封裝』特性而言
  - Java透過類別中成員變數及成員函式的等級，達成資料封裝的目的，其中protected保護等級由於牽扯到繼承時的特殊變化，因此，我們留到本章才加以介紹。
- 就『繼承』特性而言
  - Java提供了『類別繼承』。『類別繼承』是基於已經存在的類別，利用一種類似繼承的動作來取得相同類別的機制。這可以避免一些重複的程式碼並可進行有效的擴充；因此有助於管理程式原始碼。
- 就『多型』特性而言
  - 除了之前介紹的『多載』之外，另一項更重要的技術則是**改寫(override)**，這個技術在抽象類別及下一章所介紹的『介面』尤其重要。這兩種方式可以讓程式設計師克服一些物件繼承時的限制，達到統一介面的目的。

3

8.1 類別繼承的目的

- 物件導向是為了在開發大型程式時，能夠更有效率而發展的一套技術。
  - 在物件導向程式設計中，如何讓相同的程式碼被重複利用，是相當重要的。
  - Java為了讓每一行程式都具有高利用價值，而採用了類別繼承的觀念來實現此一精神。
  - 類別繼承能夠讓程式更利於管理和閱讀。

4

8.1 類別繼承的目的

- 假設我們有一個滑鼠類別
  - 當中包含了滑鼠的元件及行為模式（例如移動游標method、按一下method等等）
  - 假設它是傳統的滾球滑鼠
- 則我們可以在設計新型態的滾輪及光學滑鼠時採用繼承技巧
  - 假設命名為光學滾輪滑鼠類別
  - 則該類別可以先繼承滑鼠類別，然後加上新的成員－『光學元件』、『滾輪元件』
  - 修改原本的移動游標method，新增滾輪瀏覽method。
  - 而其他如按一下method則不需要變動，這樣一來，我們就省去了開發按一下method的時間，因為它已經是父類別（滑鼠類別）的既有Method。

5

8.2 一般類別的繼承

- Java的類別繼承可以分為一般類別的繼承與介面類別的繼承
  - 由於尚未介紹介面類別，因此本節先介紹一般類別的繼承。

6

8.2.1 單一繼承

- Java的一般類別繼承可以是『單一繼承』，也可以是擴展出來的『多層繼承』。我們先介紹單一繼承。

延伸學習：**多重繼承**

C++支援多重繼承（一個類別擁有多個父類別稱為多重繼承），但Java不直接支援多重繼承，而類似的機制則必須透過介面來完成。

7

8.2.1 單一繼承

- 衍生類別與基底類別
  - 類別的繼承非常類似類別的複製，但並不完全是複製，所以我們不將之稱為類別的複製，而稱之為類別的**繼承(inheritance)**。
    - 就『繼承』的字面意義而言，一般人很容易聯想到『誰繼承了誰』這個問題，就現實社會而言，一般都是『子繼承父』；在Java程式設計的單一繼承中，也是相同的道理

8

8.2.1 單一繼承

- 衍生類別與基底類別
  - 我們將被繼承者稱為父類別，繼承者稱為子類別。『子』為何會繼承『父』呢？當然是因為『父』生『子』的緣故。而在本書中，我們將『生』這個動作特別稱為『衍生』(derive)。
  - 在Java的單一繼承中，我們通常稱被衍生的類別為**父類別(parent class)**或**基底類別(base class)**。而繼承衍生而來的類別稱之為**子類別(child class)**或**衍生類別(derived class)**，父類別和子類別有其先後和相似的關係。

9

8.2.1 單一繼承

- 在Java中，宣告『單一繼承』的子類別定義語法如下：

```
class  衍生(子)類別名稱  extends  基底(父)類別名稱
{
    子類別實作區
}
```

繼承自；延伸自

- 【語法說明】：
  - (1)父類別可以是使用者自訂的類別，也可以是Java提供的類別，但使用Java提供的類別時，必須先將其import進來。
  - (2)宣告子類別時，必須指定父類別的名稱。並且其中使用了關鍵字extends，extends的原文具延伸之意，故繼承之後，除了父類別的既有成員之外，子類別還可以重新定義或新增更多的成員。

10

8.2.1 單一繼承

- (3)子類別除了繼承父類別，也可以另外再定義一些屬於自己的成員變數和函式（只要記錄於上述語法的子類別實作區中即可），因此父子類別之間的關係如圖8-1所示
  - 請注意，為了解說方便該圖箭頭方向與UML是相反的
- (4)繼承自父類別的成員其封裝等級同父類別原本之封裝等級（含package, public, protected）。
  - 但private等級無法由子類別直接取用，必須透過父類別其他等級的成員取用。

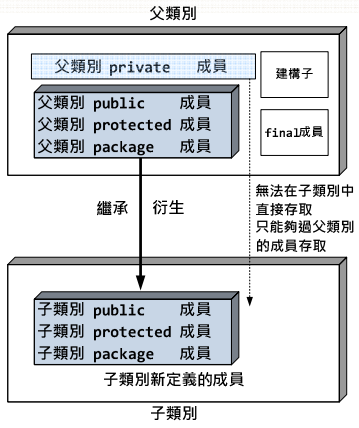


圖8-1 父子類別之間的關係

11

8.2.1 單一繼承

- (5)如果有某些成員不希望公開給外部程式取用，但希望繼承者仍可使用，應該宣告為protected等級，但仍需搭配不同package來限制。
- (6)建構子無法被繼承。

- 透過繼承機制，當我們需要重複使用某個類別裡的一些成員時，我們不需要重複撰寫類似的程式碼，我們只需要衍生出一個新的子類別，取得其基本功能和成員，再針對需求加以新增或修改即可。這也就是重複使用程式碼的精神所在。

12

8.2.1 單一繼承

- 假設有一個類別CX為父類別，我們想要利用這個父類別去衍生出一個子類別CY，則如下片段程式
  - 假設類別CX的定義如下：

```
class CX
{
 //CX類別的成員定義
}
```

  - 類別CY欲繼承類別CX的成員，則類別CY的定義應該如下：

```
class CY extends CX
{
 //CY類別新定義的成員
}
```

- 【觀念範例8-1】：單一繼承語法的基本練習。
  - **範例8-1：**ch8_01.java（隨書光碟myJava\ch08\ch8_01.java）

13

8.2.1 單一繼承

```
1  /*   檔名:ch8_01.java        功能:單一繼承   */
5  import java.util.Scanner;
6
7  public class ch8_01          //主類別
8  {
9      public static void main(String args[])
10     {
11         CA objA =  new CA();
12         CB objB =  new CB();;            //objB是子類別物件
13         System.out.println("輸入類別CA的objA物件資料");
14         objA.inputX();
15         objA.inputY();
16         objA.inputZ();
17         System.out.print("類別CA的objA物件:");
18         objA.show();
19         System.out.println("------------------------");
20         System.out.println("輸入類別CB的objB物件資料");
21         objB.inputX();
22         objB.inputY();
23         objB.inputZ();
24         objB.varX++;
25         //objB.varY++;
26         objB.addOne();
27         System.out.print("類別B的objB物件:");
28         objB.show();
29     }
30 }
31
```

不同package的外部程式無法存取protected等級資料，但ch8_01目前位於同一個package所以可以存取varY

14

```
32  class CA
33  {
34      Scanner keyboardInput = new Scanner(System.in);
35      public int varX;
36      protected int varY;
37      private int varZ;
38      public void inputX()
39      {
40          System.out.print("varX:");
41          varX = Integer.parseInt(keyboardInput.nextLine());
42      }
43
44      public void inputY()
45      {
46          System.out.print("varY:");
47          varY = Integer.parseInt(keyboardInput.nextLine());
48      }
49
50      public void inputZ()
51      {
52          System.out.print("varZ:");
53          varZ = Integer.parseInt(keyboardInput.nextLine());
54      }
55
56      public void show()
57      {
58          System.out.println("varX=" + varX +
                                   " varY=" + varY + " varZ=" + varZ);
59      }
60  }
```

varY被宣告為保護(protected)等級

15

8.2.1 單一繼承

```
61
62  class CB extends CA
63  {
64      public int varW;          //新增的成員變數
65      public void addOne()      //新增的成員函式
66      {
67          varX++;
68          varY++;
69          //varZ++;
70      }
71  }
```

類別CB繼承自類別CA

無法直接存取父類別private等級資料，必須透過同樣是由父類別繼承而來的method進行存取

- 執行結果：

```
輸入類別CA的objA物件資料
varX:10
varY:10
varZ:10
類別CA的objA物件:varX=10 varY=10 varZ=10
------------------------
輸入類別CB的objB物件資料
varX:20
varY:20
varZ:20
類別B的objB物件:varX=22 varY=21 varZ=20
```

16

8.2.1 單一繼承

- 範例說明：
  - (1)第32~60行是父類別的定義。第62~71行是子類別的定義。繼承之後，子類別又新增了兩個成員varW與addOne()。下表是CB子類別的成員及其等級。

等級	CA父類別的成員	CB子類別的成員
package	Console console	Console console
public	int varX	int varX
protected	**int varY**	**int varY**
private	**int varZ**	**無法直接存取**
public	void inputX()	void inputX()
public	void inputY()	void inputY()
public	void inputZ()	void inputZ()
public	無	int varW
public	無	void addOne()
public	CA()預設建構子	無法繼承
public	無	CB()預設建構子

17

8.2.1 單一繼承

- (2)第25行是透過外部程式存取objB的varY成員變數，若外部程式與宣告protected成員的CA類別位於不同的package，則無法存取。不過此時是可以存取的（ch8_01類別目前與CA類別位於同一個package）。
- 您若將第25行註解取消，會發現範例仍舊是可以編譯的，我們將於第11章講解類別庫時，重新探討這個案例。
- (3)varZ在父類別中，被宣告為private等級，所以在被繼承後無法被直接存取，因此第69行無法透過子類別的成員函式存取varZ成員變數（註解不可取消）。
- 但varZ仍舊可以被繼承而來的父類別inputZ、show等method存取。

18

8.2.1 單一繼承

- **父子類別的建構子執行順序**
  - 在建立子類別物件時，**會先執行父類別的建構子，然後才執行子類別的建構子**。當您未建立建構子時，將會執行預設建構子。
  - 通常，我們會將父類別成員的初始化放在父類別建構子中完成，因為建立子類別物件時，會先自動執行父類別建構子。

19

8.2.1 單一繼承

- 【觀念範例8-2】：父子類別的建構子執行順序。
  - **範例8-2：**ch8_02.java（隨書光碟 myJava\ch08\ch8_02.java）

```
1   /*  檔名:ch8_02.java          功能:繼承的建構子執行順序    */
6   public class ch8_02          //主類別
7   {
8       public static void main(String args[])
9       {
10          CB objB =  new CB();            //objB是子類別物件
11          objB.show();
12      }
13  }
14
```

產生子類別物件實體，會先執行父類別的建構子，然後才執行子類別的建構子

20

8.2.1 單一繼承

```
15  class CA
16  {
17      protected int varX;
18      public CA()
19      {
20          System.out.println("父類別建構子執行中....");
21          varX = 10;
22      }
23  }
24
25  class CB extends CA    // 類別CB繼承自類別CA
26  {
27      public CB()
28      {
29          System.out.println("子類別建構子執行中....");
30      }
31      public void show()
32      {
33          System.out.println("varX=" + varX);
34      }
35  }
```

父類別建構子

初始化父類別的成員變數

子類別建構子

8.2.1 單一繼承

- 執行結果：

```
父類別建構子執行中....
子類別建構子執行中....
varX=10
```

先執行的是父類別的建構子

後執行的是子類別的建構子

- 範例說明：
  - 由執行結果中，可以看出在建立子類別物件時，會先執行父類別的建構子，然後才執行子類別的建構子，同時，我們在父類別內對成員變數作了初始化動作，所以在子類別物件中的varX也是10。

8.2.2 super關鍵字

- 子類別在繼承父類別時，子類別的新成員可以和父類別成員同名
  - 如果是method同署名的話，則稱為改寫(override)，我們將會在後面介紹。
  - 如果是變數同名，則不會將父類別的變數覆蓋，而是將繼承而來的父類別之同名成員變數隱藏(hide)，所以是兩者獨立的兩個變數。
    - 要取用子類別的同名變數，就如同一般撰寫程式般取用即可
    - 而要取用父類別物件的同名變數，則可透過super關鍵字來達成。

23

8.2.2 super關鍵字

- 在Java中，this代表的是參考物件本身，super代表的則是參考父類別物件本身
  - 如果要讀取父類別某個成員，可以使用「super.父類別成員」語法來完成，但不可以使用這種方式在子類別中讀取父類別的private成員。
  - 如果想要在子類別的建構子中執行父類別的建構子，只需要使用「super(引數串)」來完成即可，不過它必須放在子類別建構子的第一行才行。
  - 【觀念範例8-3】：子類別變數與父類別變數名稱相同。
    - **範例8-3：**ch8_03.java（隨書光碟 myJava\ch08\ch8_03.java）

24

8.2.2 super關鍵字

```
1   /*   檔名:ch8_03.java          功能:super的練習    */
6   public class ch8_03           //主類別
7   {
8       public static void main(String args[])
9       {
10          CB objB =  new CB(10); //objB是子類別物件
11          objB.show();
12      }
13  }
14
15  class CA
16  {
17      public int varX;
18      public CA()
19      {
20          System.out.println("父類別無參數建構子執行中...");
21      }
22      public CA(int i)
23      {
24          varX = i;
25      }
26  }
```

父類別的成員

執行結果

```
子類別varX=100
父類別varX=11
```


8.2.2 super關鍵字

```
27
28  class CB extends CA   // 子類別CB繼承自類別CA
29  {
30      public int varX;          //新增的成員變數,與父類別成員同名
31      public CB(){}
32      public CB(int i)
33      {
34          super(i);
35          varX = 100;
36      }
37      public void show()        //新增的成員函式
38      {
39          super.varX++;
40          System.out.println("子類別varX=" + varX);
41          System.out.println("父類別varX=" + super.varX);
42      }
43  }
```

呼叫父類別建構子public CA(int i)，必須放在第一行

存取子類別的成員

存取父類別的成員

- 範例說明：
  - (1)CB繼承CA，並且新增一個同名的變數varX。第35、40行取用的是子類別的變數varX，第39、41行取用的是父類別的varX。

8.2.2 super關鍵字

- (2)第34行是呼叫第22~25行的父類別建構子。由執行結果中，可以發現，無參數的父類別建構子並不會被執行，因為編譯器發現您在子類別的建構子中，已經呼叫父類別的建構子。如果將第34行刪除，則第18~21行的父類別建構子會自動被執行。
- (3)如果您同時將第34行刪除，也將第18~21行的父類別建構子刪除，則編譯時會發生錯誤，因為編譯器想要執行一個無參數的建構子，卻發現找不到。還記得上一章曾經提及，一但我們建立了任何一個建構子，則預設的建構子將不會出現嗎！

小試身手8-1
請將範例8-3的第34行、第18~21行都刪除，然後重新編譯，看看會產生什麼錯誤訊息。

27

8.2.3 改寫(Override)

- 在範例8-3中，成員變數在被繼承後仍舊可以另行宣告一個專屬於子類別的同名成員變數
- 那麼如果在子類別中宣告一個與父類別同「署名」的method，情況又會是如何呢？
  - 答案是發生改寫(override)現象。當子類別欲改寫父類別的method時，有以下幾點必須注意：
    - (1)署名雖然不包含方法的封裝等級，但在改寫時，只能擴大封裝等級，而不能縮小封裝等級。例如protected可在改寫時，改為public，但相反則否。
    - (2)改寫的回傳值型態只能是相同或原本資料型態的子類別型態。（在JDK5.0之前，則完全不允許改變回傳值型態）
    - (3)父類別的static method無法被改寫。但我們確實可以在子類別中定義一個新的同名static method，但那並非改寫，而是兩個不相關的method。

28

8.2.3 改寫(Override)

- 改寫(override)與多載(overload)
  - 改寫(override)與之前介紹的多載(overload)是完全不同的兩種機制，兩者之分別如下：
    - (1)多載的methods必須位於同一個類別視野。而改寫的兩個成員則分處於不同的類別視野，一個位於父類別，另一個位於子類別。
    - (2)多載必須是同名但不同署名。而改寫則必須是同署名。例如，如果類別A有一個func()，而類別B繼承自類別A，同時又定義了一個func(int X)，則兩者並沒有多載或改寫現象，但這還是允許的，因為編譯器可以輕鬆地辨別所呼叫的對象。

延伸學習：**abstract method**

abstract method是一個只有宣告沒有定義的method。事實上，改寫最常發生在abstract method，我們將於8.5節說明。

29

8.2.3 改寫(Override)

- 在改寫的狀況下，我們必須告知編譯器，目前要呼叫的是子類別物件的同名method還是父類別物件的同名method
  - 呼叫子類別物件的同名method只要直接呼叫即可
  - 呼叫父類別物件的同名method，則必須透過super來完成。
  - 請看下一個範例的示範。
  - 【觀念範例8-4】：改寫(Override)的練習。
    - **範例8-4：**ch8_04.java（隨書光碟myJava\ch08\ch8_04.java）

30

8.2.3 改寫(Override)

```
1   /*   檔名:ch8_04.java          功能:改寫(override)的練習    */
6   public class ch8_04
7   {
8       public static void main(String args[])
9       {
10          CB objB =  new CB();              //objB是子類別物件
11          objB.show();
12          objB.show(10);
13          objB.show(10.0);
14      }
15  }
16
17  class CA
18  {
19      public void show()
20      {
21          System.out.println("目前執行父類別的show()");
22      }
23  }
24
```

執行結果

```
目前執行子類別的show()
目前執行子類別的show(10)
目前執行子類別的show(double)
目前執行父類別的show()
```

31

8.2.3 改寫(Override)

```
25  class CB extends CA    // 類別CB繼承自類別CA
26  {
27      public void show()
28      {
29          System.out.println("目前執行子類別的show()");
30      }
31      public void show(int x)
32      {
33          System.out.println("目前執行子類別的show(" + x +")");
34      }
35      public void show(double x)
36      {
37          System.out.println("目前執行子類別的show(double)");
38          super.show();
39      }
40  }
```

子類別的show() override父類別的show()

呼叫父類別的method

- 範例說明：
  - (1)第27~30行的show()改寫了第19~22行的show()成員函式。
  - (2)第27行與第19行的methods為改寫(Override)的關係。第27行、第31行、第35行的methods為多載(Overload)的關係。第31行與第19行的methods既非多載也非改寫的關係。

32

8.2.3 改寫(Override)

- (3)第38行為執行父類別物件的show()函式，故由此可以看出改寫並非將原本的函式覆蓋，可以將其視為隱藏的現象。
- (4)您可以將第19行的public改寫為protected，程式仍是合法的，因為改寫時，允許擴大封裝等級。

小試身手8-2
請將範例8-4的第19行改為protected void show()，然後重新編譯，看看是否可通過編譯？

33

8.2.4 @Override註記

- 註記(annotation)也可以算是一種註解，但為何本書不將之納為第二章的註解種類
  - 主要是因為，註記(annotation)不只是給人看的註解，也是給編譯器看的註解。
- 註記是以@為開頭
  - 常見的有@Override, @Deprecated, @SuppressWarnings,@interface, @Document與@Deplicate...等等，甚至還允許使用者自行定義註記，因此許多以Java開發的框架（例如JUnit）都會大量使用註記。
  - 它是Java SE5才開始支援的機制，最主要的目的是幫助程式的開發
    - 例如避免因撰寫程式時的不注意而犯下合乎語法，但不符設計者原意的錯誤。
  - 在本小節中，我們將介紹@Override註記。

34

8.2.4 @Override註記

- 假設我們有A父類別，當中有一個rideBike()方法
- 而另有一個子類別B繼承了A類別並且想要改寫rideBike()方法
- 但是由於A父類別放置在其他檔案中，因此設計師以為該方法的名稱是rideBicycle()方法，故而在B類別中定義了一個rideBicycle()方法想要改寫rideBike()方法
  - 結果是，編譯器不會幫設計者找出這個錯誤
- 對於編譯器而言，類別B將會有兩個不同名稱的方法
  - 一個是繼承而來未改寫的rideBike()方法
  - 另一個是新增的rideBicycle()方法。
  - 除非設計者在B類別中呼叫了super.rideBicycle()方法，否則編譯器不會顯示錯誤，但執行結果可能與設計者想要的結果不同。

35

8.2.4 @Override註記

- 更常見的例子出現在參數數量或型態的不同
  - 假設類別A有一個用來計算面積的calArea(int len, int width)方法
  - 而類別B繼承類別A之後欲改寫計算面積的calArea方法，因此定義了一個calArea(double len, double width)方法
  - 則編譯器還是會將之視為新增的方法而非改寫，這也可能與設計者的原意並不相同。
- 有鑑於編譯器只能以完整的署名來判斷是否為改寫，因此，Java提供了一項機制，協調設計者與編譯器的認知
  - 亦即在撰寫方法前，使用@Override來做註記，代表接下來定義的方法是一個改寫方法，而非子類別的新增方法。
  - 這個註解雖然主要是寫給編譯器看的，但實際上也可以使得日後維護的人員很容易辨認為何會出現這個method。

36

8.2.4 @Override註記

- 因此， @Override註記在許多「使用由他人提供類別庫的場合」中，非常具有實用性
  - 例如開發Android應用程式時，就會常常見到原始碼中出現@Override這個關鍵字。
  - 【觀念範例8-5】：將範例8-4改編，利用@Override來標記要改寫的方法。
    - **範例8-5：**ch8_05.java（隨書光碟myJava\ch08\ch8_05.java）

37

8.2.4 @Override註記

```
1   /*   檔名:ch8_05.java          功能:@Override的練習    */
6   public class ch8_05          //主類別
7   {
8       public static void main(String args[])
9       {
10          CB objB =  new CB();            //objB是子類別物件
11          objB.show();
12          objB.show(10);
13          objB.show(10.0);
14      }
15  }
16
17  class CA
18  {
19      public void show()
20      {
21          System.out.println("目前執行父類別的show()");
22      }
23  }
24
```

38

8.2.4 @Override註記

告知編譯器，這個method是要改寫父類別的show() override父類別的show()

```
25  class CB extends CA
26  {
27      @Override public void show()
28      {
29          System.out.println("目前執行子類別的show()");
30      }
31      public void show(int x)
32      {
33          System.out.println("目前執行子類別的show(" + x +")");
34      }
35      public void show(double x)
36      {
37          System.out.println("目前執行子類別的show(double)");
38          super.show();
39      }
40  }
```

- 執行結果：
  - （同範例8-4執行結果）
- 範例說明：
  - 第27行加入了@Override註記，編譯器就可以知道第27~30行的show()是要改寫父類別的show()。

小試身手8-3

請將範例8-5的第31行改為@Override public void show(int x)，然後重新編譯，看看是否可通過編譯？並解釋為何能通過或為何不能通過編譯。

39

8.2.5 繼承機制之物件實體建構順序

- 由之前所介紹的建構子執行順序，我們可以將單一繼承的物件實體建構順序繪製如下圖：

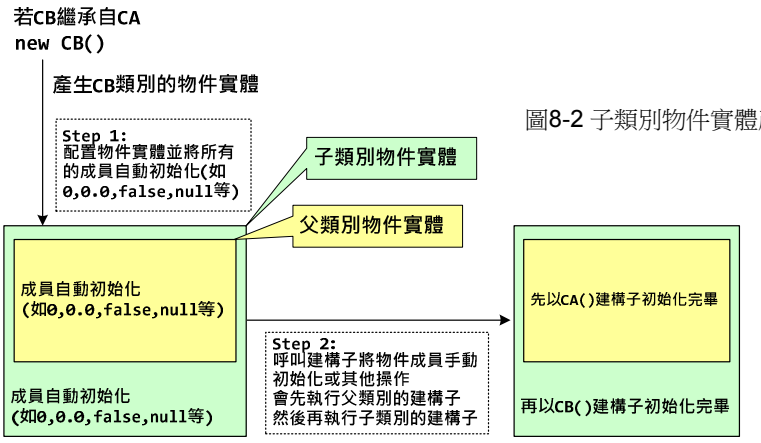


圖8-2 子類別物件實體產生過程

40

8.2.5 繼承機制之物件實體建構順序

- 對於非static的成員變數而言，物件實體產生後，將有一塊記憶體存放專屬於該物件實體的成員變數，而對於非static method而言，則會存放method的程式進入點。
- 如果類別在定義時，採用的是繼承機制，則產生類別的物件實體時，如圖8-2所示，物件實體內會包含父類別的物件實體，這也是為何父類別的成員變數即使與子類別的成員變數同名，也不會被覆蓋的原因
  - 因為兩者在物件實體產生後，會在物件實體內佔據不同的記憶體位置。所以當變數同名時，仍可以藉由super存取到其中的父類別物件實體內的變數。

41

8.2.6多層繼承

- 多層繼承屬於垂直性的層代繼承
  - 例如，子類別繼承父類別，父類別繼承祖父類別。
  - 由於多層繼承的機制，使得Java在開發大型程式時，能夠更有效率並且更容易管理。
  - 前面所介紹的單一繼承，僅討論父類別與子類別的兩代繼承關係，若將其層代擴充到三代以上，就形成了多層類別的繼承

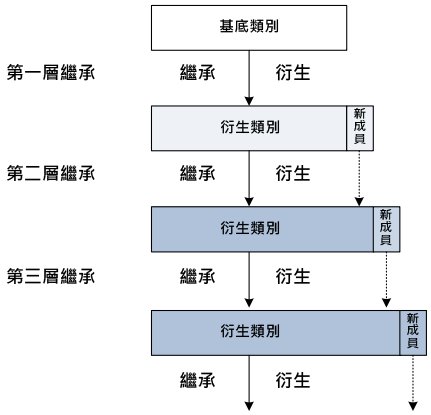


圖8-3 多層繼承示意圖

42

8.2.6多層繼承

- 當進行多層繼承時，越下層的物件實體將會越變越大
  - 因為，下層物件實體內將包含所有上層的物件實體。
- 而建構子的執行順序，也是由最上層的建構子開始執行。
  - 您可以將建構子的執行看作是一種引發的關係
  - 在兩層繼承之時，子類別的建構子會先引發父類別的建構子（引發可分為自動及手動，手動時super(引數串)呼叫必須放在第一行），所以父類別的建構子會先執行，然後才執行子類別的建構子（手動時，則執行子類別建構子第2行以後的敘述）。
  - 在三層繼承時，則子類別的建構子會先引發父類別的建構子，而父類別的建構子則會引發祖父類別的建構子，所以祖父類別的建構子會先被執行，然後是父類別的建構子，最後是子類別的建構子。
  - 我們透過一個範例來說明多層類別的建構子執行順序。

43

8.2.6多層繼承

- 【觀念範例8-6】：多層繼承的建構子執行順序。
  - **範例8-6：**ch8_06.java（隨書光碟 myJava\ch08\ch8_06.java）

```
1   /*   檔名:ch8_06.java          功能:多層繼承之建構子呼叫   */
6   public class ch8_06          //主類別
7   {
8       public static void main(String args[])
9       {
10          CC objC1 =  new CC();
11          CC objC2 =  new CC(10);
12          CC objC3 =  new CC(3,5);
13          objC1.show();
14          objC2.show();
15          objC3.show();
16      }
17  }
18
```

44

8.2.6多層繼承

```
19  class CA
20  {
21      protected int varA;
22      public CA() { varA = 1;}
23      public CA(int i) { varA = i;}
24  }
25
26  class CB extends CA   // 類別CB繼承自類別CA
27  {
28      protected int varB;
29      public CB() { varB = 1;}
30      public CB(int i)
31      {
32          super(i);
33          varB = i;
34      }
35  }
36
37  class CC extends CB   // 類別CC繼承自類別CB
38  {
39      protected int varC;
40      public CC() { varC = 1;}
41      public CC(int i){varC = i;}
42      public CC(int i,int j)
43      {
44          super(i*j);
45          varC = i * j;
46      }
```

45

8.2.6多層繼承

```
47      public void show()
48      {
49          System.out.print("varA=" + varA + "\t");
50          System.out.print("varB=" + varB + "\t");
51          System.out.println("varC=" +varC);
52      }
53  }
```

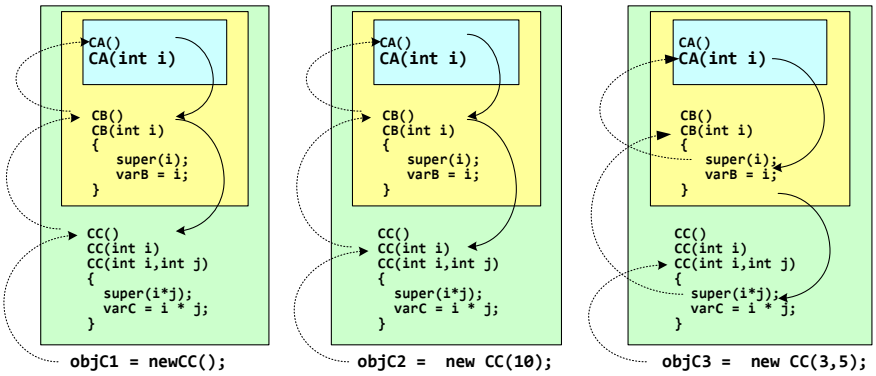
- 執行結果：

```
varA=1   varB=1   varC=1
varA=1   varB=1   varC=10
varA=15  varB=15  varC=15
```

46

8.2.6多層繼承

- 範例說明：
  - CC類別繼承CB類別，CB類別繼承CA類別。本範例一共有三個CC類別的物件，objC1, objC2, objC3，當物件建立時，其建構子的呼叫順序如下圖，如果底層的建構子內，在第一行未註明super執行某個特定的上層建構子，則會自動先執行上層無參數的建構子。



47

8.2.7父類別型態變數參考子類別物件實體

- 類別型態的物件參考可以指向相容類別的物件實體
  - 此處所謂**相容(compatible)**類別代表的是其衍生類別，例如子類別或孫類別…等等。
- 我們可以宣告一個物件變數，宣告型態時採用父類別型態，但卻將它指向子類別物件實體。例如下列範例：

```
class CA{.....}
class CB extends CA{.....}
CA obj = new CB();
```

父類別型態變數參考子類別物件實體

  - 當父類別型態的物件變數指向子類別的物件實體後，我們可以透過該變數執行子類別中改寫(override)的method或其他未改寫的繼承method，**但不能執行子類別的新增method。**

48

8.2.7父類別型態變數參考子類別物件實體

- 【觀念範例8-7】：父類別型態變數參考子類別物件實體
  - **範例8-7：**ch8_07.java（隨書光碟 myJava\ch08\ch8_07.java）

```
1   /*   檔名:ch8_07.java          功能:父類別型態變數參考子類別物件實體    */
6   public class ch8_07           //主類別
7   {
8       public static void main(String args[])
9       {
10          CA obj =  new CB();                  父類別型態變數參考子類別物件實體
11          obj.show1();          //可執行子類別改寫的函式
12          obj.show2();          //可執行繼承的函式
13          //obj.show3();                       不能執行子類別新增的函式
14      }
15  }
16
17  class CA
18  {
19      protected int varA;
20      public void show1()
21      {
22          System.out.println("父類別show1()執行中");
23      }
```


8.2.7父類別型態變數參考子類別物件實體

```
24      public void show2()
25      {
26          System.out.println("父類別show2()執行中");
27      }
28  }
29
30  class CB extends CA   // 子類別CB繼承自類別CA
31  {
32      protected int varB;
33      @Override public void show1()
34      {
35          System.out.println("子類別改寫的show1()執行中");
36      }
37      public void show3()
38      {
39          System.out.println("子類別新增的show3()執行中");
40      }
41  }
```

- 執行結果：

```
子類別改寫的show1()執行中
父類別show2()執行中
```

8.2.7父類別型態變數參考子類別物件實體

- 範例說明：
  - (1)父類別CA定義了show1()與show2()兩個methods。子類別CB定義了show1()與show3()兩個methods，由於CB繼承了CA，因此show1()為改寫的method。
  - (2)第10行，宣告變數型態為父類別，但new產生實體時為子類別，因此，父類別變數obj將指向子類別物件實體。
  - (3)第11行，obj可以執行子類別改寫的method。
  - 第12行，obj可以執行子類別繼承自父類別的method。
  - 第13行，obj不能執行子類別新增的method。

51

8.2.7父類別型態變數參考子類別物件實體

- 【實用範例8-8】：假設我們有一個圖形類別，當中包含area變數以作為存放圖形面積之用。
  - 由於計算面積按照圖形的種類而有各種不同的公式，故我們將計算面積的函式保留交由繼承的子類別改寫來完成。
  - 請建立一個圖形類別的物件陣列，計算面積後，將陣列依照面積排序，最後輸出陣列內各物件的面積。
- **範例8-8：**ch8_08.java（隨書光碟myJava\ch08\ch8_08.java）

52

8.2.7父類別型態變數參考子類別物件實體

```
1   /*  檔名:ch8_08.java       功能:父類別型態變數參考子類別物件實體的應用  */
6   public class ch8_08          //主類別
7   {
8       public static void main(String args[])
9       {
10            CShape objArr[] =  new CShape[4];
11            objArr[0] = new CRect(10.0,20.0);
12            objArr[1] = new CRect(5.0,15.0);
13            objArr[2] = new CCircle(3.0);
14            objArr[3] = new CCircle(6.0);
15
16            for(int i=0;i<objArr.length;i++)
17              objArr[i].computeArea();
18
19            System.out.println("物件的面積如下");
20            CShape.show(objArr);          //顯示面積
21            CShape.sortByArea(objArr);    //進行排序
22            System.out.println("物件依面積排序後如下");
23            CShape.show(objArr);          //顯示面積
24      }
25  }
26
```

父類別型態變數參考子類別物件實體

執行子類別改寫的計算面積公式

8.2.7父類別型態變數參考子類別物件實體

```
27  class CShape
28  {
29      protected double area;
30
31      public static void show(CShape objArr[])
32      {
33            for(int i=0;i<objArr.length;i++)
34                System.out.print(objArr[i].area + "\t");
35            System.out.println();
36      }
37      public static void sortByArea(CShape objArr[])   //排序
38      {
39           int k,times;
40           CShape temp;
41           k = objArr.length - 1;
42           while(k!=0)
43           {
44             times = 0;
45             for(int i=0;i<=k-1;i++)
46             {
```

8.2.7父類別型態變數參考子類別物件實體

```
47              if(objArr[i].area > objArr[i+1].area)
48              {
49               temp = objArr[i];     objArr[i] = objArr[i+1];
50               objArr[i+1] = temp;   times = i;
51              }
52            }
53            k = times;
54          }
55      }
56      public void computeArea() {}
57  }
58
59  class CRect extends CShape    // 類別CRect繼承自類別CShape
60  {
61      protected double length,width;
62      public CRect(double i,double j)
63      {
64            length = i;
65            width = j;
66      }
67      @Override public void computeArea()
68      {
69            area = length * width;
70      }
71  }
72
```

55

8.2.7父類別型態變數參考子類別物件實體

```
73  class CCircle extends CShape    // 類別CCircle繼承自類別CShape
74  {
75      protected double radius;
76      protected final double pi=3.14;
77      public CCircle(double i)
78      {
79            radius = i;
80      }
81      @Override public void computeArea()
82      {
83            area = radius * radius * pi;
84      }
85  }
```

- 執行結果：

```
物件的面積如下
200.0   75.0    28.26   113.04
物件依面積排序後如下
28.26   75.0    113.04  200.0
```

56

8.2.7父類別型態變數參考子類別物件實體

- 範例說明：
  - (1)父類別CShape定義一個area變數用來存放面積。並定義了三個methods，其中兩個為static methods，可以由類別直接執行，sortByArea()是排序method，同第六章所介紹之氣泡排序，只不過此處依據的是物件的area欄位。另一個非static method為計算面積函式computeArea()，內容為空，它可以被繼承。
  - (2)子類別CRect繼承了父類別CShape，並且改寫computeArea()。另一個子類別CCircle亦如此。但兩者的method內容則按照圖形計算面積公式而有所不同。
  - (3)第10~14行，宣告時使用父類別型態宣告物件陣列，陣列元素內為物件的參考，應該指向父類別物件實體，但也可以指向子類別物件實體。
  - (4)第16~17行，使用迴圈計算面積，當中，i=0, i=1時，執行的是第67~70行的method。而當i=2, i=3時，執行的則是第81~84行的method。

57

8.2.8 UML圖中的類別繼承

- 在UML中，有一種空心三角箭頭是用來表示一般化(generalization)關係，它恰可用來對應類別的繼承。
  - 請注意，子類別與父類別是一種is a kind of的關係，亦即子類別是父類別的一種
    - 例如人類是哺乳類的一種，哺乳類中可能定義了乳房數量，可能定義了哺乳動作，這對於人類而言，都是有意義的。而人類可能還多了如行走等操作，而行走操作並不適用於其他哺乳類的子類別（例如鯨豚類）。

58

8.2.8 UML圖中的類別繼承

- 若以範例8-8 例，我們可以將其類別繼承的UML 圖繪製如下：

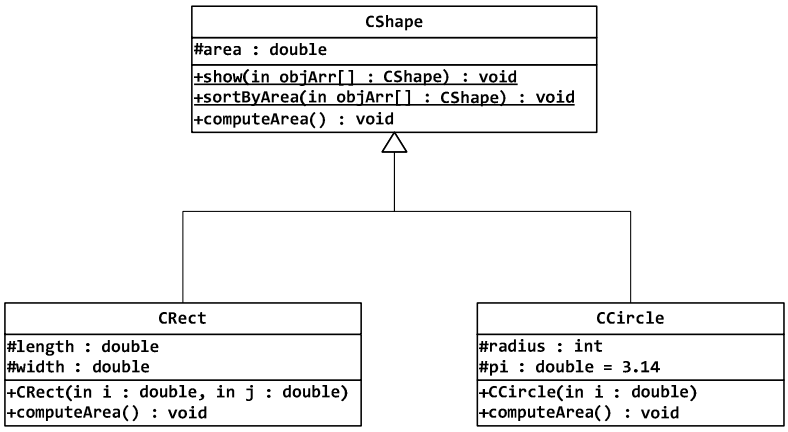


圖8-4 範例8-8的UML類別繼承圖（含底線的操作是靜態操作）

59

8.2.9 繼承的誤用與濫用

- 物件導向設計是在模擬真實世界的情況來進行系統的設計
  - 由於真實世界是非常複雜的，我們不可能完全模擬真實世界的情況，因此這是一個由簡而繁的過程。
  - 換句話說，在系統設計之初，我們對於真實世界的模擬會在系統發展的過程中不斷地因著我們對問題的理解加深而修正。
    - 尤其是在繼承這個環節特別容易出現，因為繼承這個概念與現實生活中的分類可能不完全能夠對應，況且我們對於現實生活中的分類也不一定很清楚。

60

8.2.9 繼承的誤用與濫用

- 舉例來說，一個真實世界與物件導向設計最大的區別就如同橢圓形與圓形的問題。
  - 在數學定義上，圓形是橢圓形的一種，而如果程式設計師很直覺地讓圓形類別去繼承橢圓形類別，那麼橢圓形的長軸和短軸（或者兩個焦點）在圓形裡又是什麼？探討這個問題可以得出一個結論
    - 也就是物件導向設計的繼承（B繼承A），
      除了B is a kind of A之外，
      A的所有屬性與操作對於B而言也應該是有意義的。

61

8.2.9 繼承的誤用與濫用

- 另一個例子我們以前面所提過的人類及哺乳類為例
  - 我們原本以為人類繼承哺乳類，但如果後來應用中出現了猴子時，該怎麼辦呢？因為猴子有尾巴屬性，其他則和人類的屬性相同，操作也都相同，此時猴子該放在哪裡呢？
    - 答案是應該設計一個靈長類，
      靈長類繼承哺乳類，
      人類繼承靈長類，猴子類繼承靈長類。
  - 這是需求或問題產生了變化而導致我們的設計也應該隨著改變。
  - 或者是我們對於問題理解得更深入而導致我們必須改變設計。

62

8.2.9 繼承的誤用與濫用

- 有時候，我們對於問題領域的誤解可能會做出錯誤的設計。
  - 舉例來說，假設我們有老鷹、麻雀、、等等的物件，因此設計出了鳥類這個類別，並且在此類別中定義了飛翔操作。
    - 後來卻發現，鴕鳥不會飛，那麼就該檢討這個飛翔操作是否應該定義在鳥類這個類別當中，或許在生物界，鳥類的定義並不是以會不會飛翔來定義。

63

8.2.9 繼承的誤用與濫用

- 修正設計有時候會付出較大的代價，尤其是當你已經將設計透過程式實現後才發現設計需要修正。
- 一個有用的方式稱之為重構（Refactoring）指得是重新修正程式碼達到原本既有的功能。
  - 而重構的目的則在於當需求改變時，能夠更有效率地完成設計的變更與實現。
  - 重構有許多方法，例如分解大函式、拆解大物件、抽取父類別等等，這些方法還會依據一些現代軟體設計理念來完成（例如單一職責原則）。
- 而重構本身已經是一門學問，足以用一本以上的專書來討論
  - 建議讀者在閱讀完本書之後，有了千行以上的程式設計經驗再來學習重構的技巧。

64

8.3 final修飾字的用途

- final是一個修飾字，它可以出現在下列三個位置，在不同位置有不同的意義：
  - (1)宣告變數時：代表該變數為只能設定一次然後就不可改變數值。（若為物件變數，則參考只能指向第一次指向的物件實體）
    - 使用於區域變數時，只能設定一次然後就不可改變數值
    - 使用於函式的參數時，則引數傳遞後，接收端函式內不可改變其參數值。
    - 使用於定義成員變數時，除了第一次設定之外，包含繼承後都不能再改變其值。

65

8.3 final修飾字的用途

- (2)宣告於類別的method時：代表繼承後，不可被改寫(override)。
- (3)宣告於類別時：代表該類別不可被繼承（終止繼承）。
  - 例如Java的String內建類別就被宣告為final，因此我們無法繼承String類別
    - （所幸，String類別已經非常完整，應該提供的字串相關methods功能大多已提供了）。

66

8.3 final修飾字的用途

- 範例：使用final宣告變數

```
class CA
{
   public final int var;   //宣告於成員變數
   public void func(final int i)   //宣告於參數
   {
          final int j=10;    //宣告於區域變數
   }
}
```

var只能設定一次,以後不可以再改變

i不可以再被設定

j不可以再被設定

67

8.3 final修飾字的用途

- 範例：使用final宣告類別的method

```
class CA
{
   public final void show()
   {
      ……
   }
}
```

show()不可以被改寫

- 範例：使用final宣告類別

```
final class CA
{
   ……
}
```

CA類別不可以被繼承

68

8.3 final修飾字的用途

- 【觀念範例8-9】：final在不同位置的修飾用途
  - **範例8-9：**ch8_09.java（隨書光碟 myJava\ch08\ch8_09.java）

```
1   /*   檔名:ch8_09.java          功能:final在不同位置的修飾用途    */
6   public class ch8_09          //主類別
7   {
8       public static void main(String args[])
9       {
10          final String str = new String ("final修飾字的示範");
11          //str = new String(".....");
12          System.out.println(str);
13          CB obj = new CB();
14          obj.add();
15          obj.show();     //執行繼承自父類別的show()
16      }
17  }
18
```

錯誤，因為str參考已被設定過，不可指向其他物件實體

8.3 final修飾字的用途

```
19  class CA
20  {
21      public final int var1;
22      public int var2;
23      public CA()
24      {
25          var1 = 10;
26          var2 = 10;
27      }
28      public void add()
29      {
30          //var1++;
31          var2++;
32      }
33      public final void show()   //不可被改寫的函式
34      {
35          System.out.println("var1=" + var1 + "\tvar2=" + var2);
36      }
37  }
38
```

錯誤，因為var1已經在建構子內設定過一次

8.3 final修飾字的用途

```
39  class CB extends CA   // 子類別CB繼承自類別CA
40  {
41      public void add()
42      {
43          //var1 = var1 + 10;
44          var2 = var1 + var2;
45      }
46      //public void show(){}
47  }
48
49  final class CC  //類別不可被繼承
50  {
51      public int var3;
52  }
53
54  //class CD extends CC{}
```

錯誤，因為var1已經在父類別建構子內設定過一次

錯誤，因為show不可被改寫

錯誤，因為CC不可被繼承

- 執行結果：

```
final修飾字的示範
var1=10 var2=20
```


8.3 final修飾字的用途

- 範例說明：
  - (1)第49~52行的CC類別不可被繼承，故第54行不合法。
  - (2)第33~36行的show()不可被改寫，故第46行不合法。
  - (3)第21行的成員變數var1被宣告為final，並且在建構子內已經設定過第一次，故之後不能再被設定，即使是被繼承後也不允許再被設定。
  - (4)第10行的str是一個物件參考，它指向一個字串物件實體。由於被宣告為final，故不能再指向另一個字串物件實體。

小試身手8-4

請將範例8-9的第11行改為str = ".....";，然後重新編譯，看看是否可通過編譯？如果無法通過編譯，請暫時留下您的疑問，我們會在第11章講解String類別時，為您解答這個問題。

8.4 Object類別

- Java之所以被稱為純物件導向程式語言，除了它支援物件導向的三大特色之外，還有兩個重要的原因
  - 第一個原因是所有的函式必定屬於某一個類別（C++由於支援C，故有些函式不一定屬於某一個類別）。
  - 第二個原因是Java的所有類別都有父類別。
- 嚴格來說，在Java中，除了巢狀類別（後面章節介紹）及Object類別之外，所有的類別都有父類別。
  - 其實，當編譯Java程式時，編譯器若發現某個類別沒有宣告繼承自哪一個類別的話，它會自動將Object類別當作該類別的父類別。

73

73

8.4 Object類別

- 故所有的類別都是Object類別的子類別或孫類別或…更多層繼承之後的衍生類別。故範例8-9的類別繼承可如下圖所示：

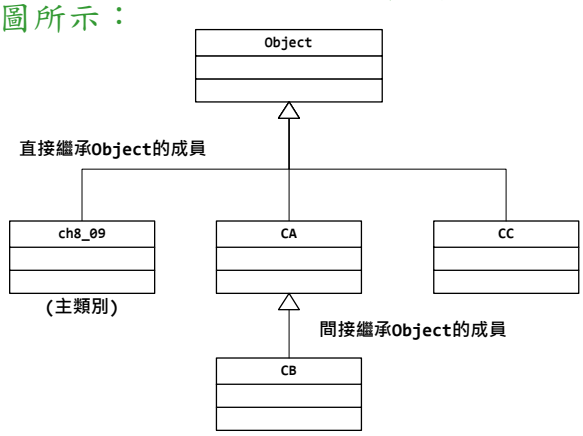


圖8-4 所有類別都繼承自Object類別(範例8-9類別繼承圖)

74

8.4 Object類別

- Object類別的完整名稱為java.lang.Object類別。
  - 除了類別會經由直接或間接繼承自Object類別之外，連陣列也同樣繼承了Object類別。
  - Object類別中沒有final method，故所有的類別都擁有Object類別的成員，並且可以改寫，下表是Object類別的methods：

函式宣告	回傳值說明	說明
boolean equals(Object obj)	回傳布林值，用以判定該物件參考是否與obj參考同一個物件實體	見後面說明
Class<?> getClass()	回傳一個泛型類別，可使用Class類別的物件變數接收回傳值	見後面說明
String toString()	回傳字串物件參考	見後面說明
protected Object clone()	Object代表一個物件變數	複製物件，但須先實作Cloneable介面

表8-1 Object的methods

75

8.4 Object類別

函式宣告	回傳值說明	說明
int hashCode()	回傳整數，代表hashcode	須了解資料結構之雜湊表
protected void finalize()	無回傳值	垃圾回收使用
void notify()	無回傳值	多執行緒使用
void notifyAll()	無回傳值	多執行緒使用
void wait() void wait(long timeout) void wait(long timeout, int nanos)	無回傳值	多執行緒使用

表8-1 Object的methods(續)

76

8.4 Object類別

- 我們僅介紹以下三種methods使用方法
  - equals (Object obj)物件相等判斷
  - getClass()取得物件所屬類別
  - toString()轉換為字串函式

- **equals (Object obj)物件相等判斷**
  - 【觀念範例8-10】： Object類別的equals()的練習。
    - **範例8-10：**ch8_10.java（隨書光碟myJava\ch08\ch8_10.java）

77

8.4 Object類別

```
1   /*   檔名:ch8_10.java          功能:物件的比較    */
6   public class ch8_10           //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass(5);
11          CMyClass Y = new CMyClass(5);
12          CMyClass Z = X;
13          System.out.print("物件X與物件Y  ");
14          if(X.equals(Y))
15             System.out.println("兩物件相等");
16          else
17             System.out.println("兩物件不相等");
18          System.out.print("物件X與物件Z  ");
19          if(X.equals(Z))
20             System.out.println("兩物件相等");
21          else
22             System.out.println("兩物件不相等");
23      }
24  }
25
```

如果X與Y參考同一個物件實體，則回傳true

78

8.4 Object類別

```
26  class CMyClass
27  {
28      private int Var;
29      public CMyClass(){}
30      public CMyClass(int i)
31      {
32          Var = i;
33      }
34  }
```

- 執行結果：

```
物件X與物件Y 兩物件不相等
物件X與物件Z 兩物件相等
```

- 範例說明：
  - 這個範例和範例7-13非常類似，只不過我們在判定物件參考是否相等時，採用的是繼承而來的equals()函式。

79

8.4 Object類別

- **getClass()取得物件所屬類別**
  - getClass會回傳物件所屬的類別，它回傳的是一個Class型態的泛型，我們可以用一個內建Class類別的物件變數來接收，並且該物件變數也可以直接由println()輸出內容，請見下列的範例。
  -
  - 【觀念範例8-11】： Object類別的getClass()的練習。
    - **範例8-11：**ch8_11.java（隨書光碟myJava\ch08\ch8_11.java）

80

8.4 Object類別

```
1   /*  檔名:ch8_11.java        功能:getClass() method  */
6   public class ch8_11         //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyClass X = new CMyClass();
11          Class Y;
12          Y = X.getClass();
13          System.out.println("X屬於" + Y +"的物件");
14          String str = new String("..");
15          Y = str.getClass();
16          System.out.println("str屬於" + Y +"的物件");
17      }
18  }
19
20  class CMyClass
21  {
22      private int var;
23  }
```

回傳X所屬類別

8.4 Object類別

- 執行結果：

```
C:\>java myJava.ch08.ch8_11
X屬於class myJava.ch08.CMyClass的物件
str屬於class java.lang.String的物件
```

- 範例說明：
  - 不論是自訂類別或Java的內建類別，都可以使用getClass()，因為它是繼承自Object類別。

8.4 Object類別

- **toString()轉換為字串函式**
  - toString()可以將物件內容轉換為字串，不過這個字串通常沒有什麼用途，所以大多數的類別都會改寫toString()
    - 例如第三章介紹的Integer,Float等等的Java內建類別都改寫了這個method，使得能夠傳回所需要的字串。請見下列的範例。
  - 【觀念及實用範例8-12】：toString()的改寫。
    - **範例8-12：**ch8_12.java（隨書光碟myJava\ch08\ch8_12.java）

83

8.4 Object類別

```
1   /*  檔名:ch8_12.java          功能:toString()的改寫  */
2
3   package myJava.ch08;
4   import java.lang.*;
5
6   public class ch8_12         //主類別
7   {
8       public static void main(String args[])
9       {
10          CRect X = new CRect();
11          CCircle Y = new CCircle();
12          String str = new String();
13          str = X.toString();
14          System.out.println("CRect類別物件執行toString():\t" + str);
15          str = Y.toString();
16          System.out.println("CCircle類別物件執行toString():\t"+ str);
17      }
18  }
19
```

84

8.4 Object類別

```
20  class CRect
21  {
22      private int length,width;
23  }
24
25  class CCircle
26  {
27      private int radius;
28      @Override public String toString()
29      {
30          return "CCircle類別是用來存放圓形";
31      }
32  }
```

改寫toString()

- 執行結果：

```
C:\>java myJava.ch08.ch8_12
CRect類別物件執行toString():   myJava.ch08.CRect@3238c403
CCircle類別物件執行toString():  CCircle類別是用來存放圓形
```

- 範例說明：
  - 本範例的CRect沒有改寫toString()，所以印出的資訊與類別有關，而CCircle則改寫了toString()，使得該method有些用途。

85

8.5 抽象類別

- 物件導向的重點之一為多型(Polymorphism)，多型又稱同名異式
  - 顧名思義，它代表使用同一個method名稱，但卻可以有不同的method內容。
- 除了我們先前所介紹的method多載(Overload)與改寫(Override)外，在本節中，我們將介紹Java語言中，另一種與method多功能化有關的技巧，稱為抽象方法(abstract method)
  - 它與C++的純虛擬函式(pure virtual function)功能類似
  - 抽象方法的重新定義就是在進行method的改寫(Override)動作。

86

8.5.1抽象方法簡介

- 回顧範例8-8的第56行，我們定義了一個computeArea()，但method內容是空的
  - 所以如果繼承時未加以改寫，則該computeArea()根本就是多餘的method。
  - 為了提醒繼承者必須改寫該函式，我們可以在method前面加上abstract修飾字。
  - 如果我們將method定義的語法區分為method宣告（第一行）與method實作區（其餘部分）
    - 則使用abstract修飾的method並不需要包含method實作區，並且必須在宣告的後面加上「;」分號結尾。

87

8.5.1抽象方法簡介

- 如果一個類別包含了任一個abstract method，則該類別也必須被宣告為abstract類別。
  - 加上abstract修飾的method稱為**抽象方法（抽象函式; abstract method）**
  - 加上abstract修飾的類別則稱為**抽象類別(abstract class)**。

88

8.5.1抽象方法簡介

- 其語法格式如下：

```
[封裝等級] abstract class 類別名稱    //定義抽象類別
{
   [封裝等級] abstract 回傳值型態 method名稱(參數串);    //宣告抽象方法
                :
                :
   [封裝等級] [修飾字] 資料型態 field名稱;    //宣告field
                :
                :
   [封裝等級] [修飾字] 回傳值型態 method名稱(參數串)   //method宣告區
   {
        method的內容     //method實作區
   }
                :
                :
}
```

若包含abstract method，則類別必須宣告為abstract class

abstract method沒有實作區

別忘了結尾加;

39

8.5.1抽象方法簡介

- 【語法說明】
  - (1)包含抽象方法的類別一定要宣告為抽象類別。
    - 而抽象類別無法使用new產生物件實體。
  - (2)抽象類別可包含抽象method與一般method。
    - 抽象類別也可以沒有抽象method，但即使如此，它仍無法使用new產生物件實體。
  - (3)由於抽象類別一定要被繼承，而抽象方法一定要被改寫，故其封裝等級不能是private。
  - (4)抽象方法沒有method實作區。
  - (5)抽象類別有建構子。

90

8.5.1抽象方法簡介

- (6)繼承抽象類別同樣使用extends關鍵字來達成，但繼承抽象類別必須實作（改寫）抽象方法
  - 如果有任何一個抽象方法沒有被實作，則必須將子類別也宣告為抽象類別，以便強迫更下層的類別在繼承它之後，實作剩餘沒有被實作的抽象方法。
  - 只有當所有的成員method都被實作之後，且該類別未被宣告為abstract，才能夠使用new產生物件實體。

- 【觀念及實用範例8-13】：以範例8-8的CShape,CRect,CCircle類別為例，將computeArea()宣告為抽象方法。
  - **範例8-13：**ch8_13.java（隨書光碟 myJava\ch08\ch8_13.java）

91

8.5.1抽象方法簡介

```
1  /*   檔名:ch8_13.java        功能:抽象類別與抽象方法  */
2
3  package myJava.ch08;
4  import java.lang.*;
5
6  public class ch8_13    //主類別
7  {
8      public static void main(String args[])
9      {
10         //CShape obj =  new CShape();
11         CRect obj1 = new CRect(5.0,15.0);
12         CCircle obj2 = new CCircle(3.0);
13         obj1.computeArea();        //計算面積
14         obj2.computeArea();        //計算面積
15         System.out.print("矩形");
16         obj1.show();
17         System.out.print("圓形");
18         obj2.show();
19     }
20 }
21
```

執行結果

```
矩形area=75.0
圓形area=28.26
```

錯誤，抽象類別不能產生物件實體

92

8.5.1抽象方法簡介

包含了抽象方法，所以必須宣告為抽象類別

```
22
23  abstract class CShape    //定義抽象類別
24  {
25      protected double area;
26
27      public void show()
28      {
29          System.out.println("area=" + area);
30      }
31      public abstract void computeArea(); //宣告抽象方法
32  }
33
34  class CRect extends CShape    // 子類別CRect繼承自類別CShape
35  {
36      protected double length,width;
37      public CRect(double i,double j){ length = i; width = j; }
38      @Override public void computeArea()
39      {
40          area = length * width;
41      }
42  }
43
```

這是一個抽象方法

改寫，實作computeArea()

8.5.1抽象方法簡介

```
43  class CCircle extends CShape    // 類別CCircle繼承自類別CShape
44  {
45      protected double radius;
46      protected final double pi=3.14;
47      public CCircle(double i) { radius = i; }
48      @Override public void computeArea()
49      {
50          area = radius * radius * pi;
51      }
52  }
```

改寫，實作computeArea()

- 範例說明：
  - 本範例的CRect與CCircle繼承了抽象類別CShape，並實作了抽象方法computeArea()。故這兩個類別可以產生物件實體。

8.5.1抽象方法簡介

- 【觀念範例8-14】：抽象類別的多層繼承。
  - **範例8-14：**ch8_14.java（隨書光碟 myJava\ch08\ch8_14.java）

```
1   /*   檔名:ch8_14.java        功能:抽象類別的多層繼承    */
2
3   package myJava.ch08;
4   import java.lang.*;
5
6   public class ch8_14          //主類別
7   {
8       public static void main(String args[])
9       {
10          CMyRect obj = new CMyRect(5.0,15.0);
11          obj.computeArea();        //計算面積
12          obj.show();
13      }
14  }
15
```

執行結果

```
矩形area=75.0
```

CMyRect非抽象類別，所以可以產生物件實體


```
16  abstract class CShape    //定義抽象類別
17  {
18      protected double area;
19
20      public abstract void show();         //宣告抽象方法
21      public abstract void computeArea(); //宣告抽象方法
22  }
23
24  abstract class CRect extends CShape
25  {   //
26      protected double length,width;
27      public CRect(double i,double j){ length = i; width = j; }
28      @Override public void computeArea()   //改寫，實作computeArea()
29      {
30          area = length * width;
31      }
32  }
33
34  class CMyRect extends CRect   // 子類別CMyRect繼承自類別CRect
35  {
36      public CMyRect(double i,double j)
37      {
38          super(i,j);
39      }
40      @Override public void show()          //改寫，實作show()函式
41      {
42          System.out.println("矩形area=" + area);
43      }
44  }
```

子類別CRect繼承自類別CShape，但未完全實作全部的抽象方法，故仍為抽象類別

改寫尚未實作的show()

8.5.1抽象方法簡介

- 範例說明：
  - 本範例的CRect繼承了抽象類別CShape，但只實作了抽象方法computeArea()，而未實作抽象方法show()，故CRect仍為抽象類別。
  - 而CMyRect繼承了CRect，並且實作了抽象方法show()，此時所有的抽象方法都已經實作了，故CMyRect類別可以產生物件實體。

97

8.5.2 抽象類別型態變數

- 雖然抽象類別無法建立物件實體，但我們仍可以宣告抽象類別型態的變數。
  - 而這樣的變數有何用途呢？
  - 在前面曾經提及，父類別物件變數可以指向子類別物件實體。
  - 因此，抽象類別型態的變數只能用於指向子類別物件實體
  - 請看下面的範例。
  - 【實用範例8-15】：改寫範例8-8，將CShape宣告為抽象類別，其中computeArea()為抽象方法。
    - **範例8-15：**ch8_15.java（隨書光碟myJava\ch08\ch8_15.java）

98

8.5.2抽象類別型態變數

```
1   /*  檔名:ch8_15.java          功能:抽象類別型態的變數  */
6   public class ch8_15           //主類別
7   {
8       public static void main(String args[])
9       {
10          CShape objArr[] =  new CShape[4];//抽象型態的變數陣列
11          objArr[0] = new CRect(10.0,20.0);//變數參考子類別物件實體
12          objArr[1] = new CRect(5.0,15.0);
13          objArr[2] = new CCircle(3.0);
14          objArr[3] = new CCircle(6.0);
15
16          for(int i=0;i<objArr.length;i++)
17              objArr[i].computeArea(); //執行子類別改寫的計算面積公式
18
19          System.out.println("物件的面積如下");
20          CShape.show(objArr);          //顯示面積
21          CShape.sortByArea(objArr);    //進行排序
22          System.out.println("物件依面積排序後如下");
23          CShape.show(objArr);          //顯示面積
24      }
25  }
```


8.5.2抽象類別型態變數

```
26
27  abstract class CShape
28  {
:       ……同範例8-8第29~55行……
56      public abstract void computeArea();
57  }
58
59  class CRect extends CShape    // 子類別CRect繼承自類別CShape
60  {
:       ……同範例8-8第61~70行……
71  }
72
73  class CCircle extends CShape    // 子類別CCircle繼承自類別CShape
74  {
:      ……同範例8-8第75~84行……
85  }
```

- **執行結果：（同範例8-8）**

```
物件的面積如下
200.0   75.0    28.26   113.04
物件依面積排序後如下
28.26   75.0    113.04  200.0
```

8.5.2抽象類別型態變數

- 範例說明：
  - 我們僅將CShape類別改為抽象類別，以及將computeArea()宣告為抽象方法。objArr[]是使用CShape抽象類別宣告的變數陣列，而後，我們將陣列元素指向子類別的物件實體。

101

8.5.3 UML的抽象類別圖

- 在UML的類別圖中，抽象類別
  - 會以斜體字來表示類別名稱
  - 當中的抽象方法也會用斜體字來呈現。

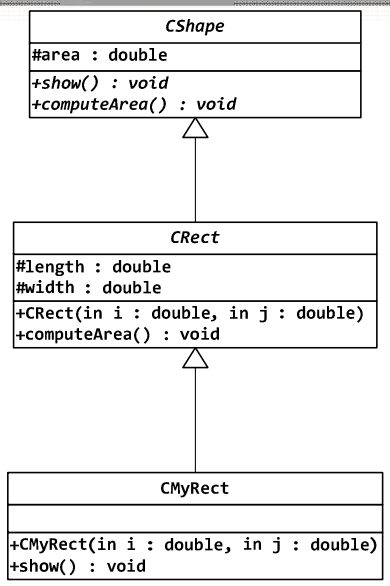
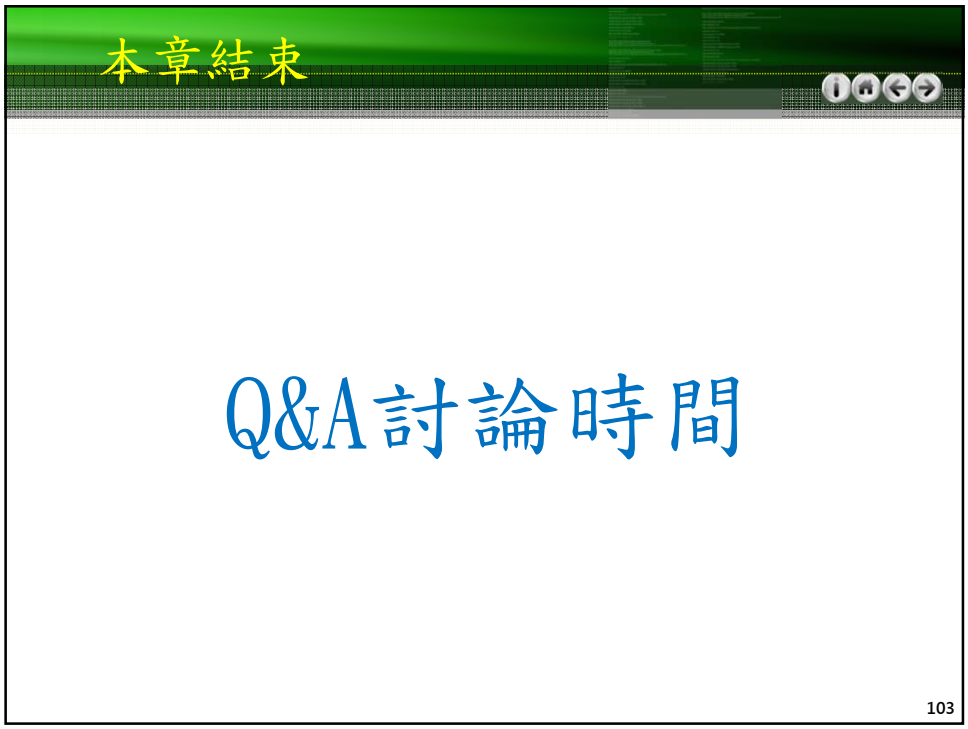


圖8-6 範例8-14的UML類別圖（CShape及CRect為抽象類別）

102


本章結束
Q&A討論時間
103