第 12 章

抽象類別 (Abstract Class)
介面 (Interface)
內部類別(Inner Class)

12 - 1 抽象類別 (Abstract Class)

- 前一章的 Shape 類別只是一個抽象的概念,
  程式中並不會有 Shape 的物件, 而只會使用它的衍生類別如 Circle、Rectangle 等, 來建立物件。
- 因此需要一種方法, 可以讓類別的使用者知道, Shape 這個類別並不能用來產生物件。

12-1-1 甚麼是抽象類別？

- Java 提供抽象類別 (Abstract Class) 的機制，其用途即是標註某個類別僅是抽象的概念，不應該用以產生物件。
- 只要在類別的名稱之前加上 abstract 存取控制字符，該類別就會成為抽象類別
- Java 編譯器將會禁止任何產生此物件的動作。

12-1-1 甚麼是抽象類別？

- 舉例來說, 在上述的範例中的 Shape 類別就可以改成這樣：

```
abstract class  Shape {
  protected double x,y;

  public Shape(double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

甚麼是抽象類別？

程式 Abstract.java 抽象類別無法建立物件

```
01 abstract class Parent { // 抽象類別
02 }
03
04 class Child extends Parent { // 子類別
05 }
06
07 public class Abstract {
08   static public void main(String argv[]{
09     Parent p = new Parent(); // 企圖建立抽象類別的物件
10   }
11 }
```

執行結果

```
Abstract.java:9: Parent is abstract; cannot be instantiated
    Parent p = new Parent(); // 企圖建立抽象類別的物件
               ^
1 error
```

12-1-2 抽象方法 (Abstract Method)

- Land 也是標註為抽象類別的好對象：

```
abstract class Land {  // 父類別
  double area() {      // 計算面積
    return 0;
  }
}
```

抽象方法 (Abstract Method)

程式 AbstractLand.java 定義抽象方法及抽象方法

```
01 abstract class Land {      // 父類別
02   abstract double area(); // 計算面積的抽象方法
03 }
04
05 class Circle extends Land { // 圓形的土地
06   int r; // 半徑（單位：公尺）
07
08   Circle(int  r) { // 建構方法
09     this.r = r;
10   }
11
12   double area() {  // 實作抽象方法 （也就是重新定義父類別中的方法）
```

抽象方法 (Abstract Method)


```
13      return 2 * 3.14 * r * r;
14    }
15  }
16
17  class Square extends Land { // 正方形的土地
18    int side; // 邊長（單位：公尺）
19
20    Square(int side) { // 建構方法
21      this.side = side;
22    }
23
24    double area() {      // 實作抽象方法 （也就是重新定義父類別中的方法）
25      return side * side;
26    }
27  }
```

抽象方法 (Abstract Method)

- 擁有抽象方法的類別一定要標註為抽象類別。
- 抽象方法代表這個方法要到子類別才會真正實作, 表示其所屬的類別並不完整, 自然就不應該拿來產生物件使用

12-1-3 抽象類別、抽象方法與繼承關係

- 對於一個擁有抽象方法的抽象類別來說，若其子類別並沒有實作其中的所有抽象方法，這個子類別也必須定義為抽象類別。

程式 WrongAbstractChild.java 自動成為抽象類別的子類別

```
01 abstract class Parent { // 抽象類別
02   abstract void show(); // 抽象方法
03 }
04
05 class Child extends Parent { // Parent 的子類別
06   // 沒有實作show, 自動成為抽象類別
07 }
08
09 class Grandson extends Child { // Child 的子類別
10   void show({ // 實作了抽象方法
```

抽象類別、抽象方法與繼承關係


```
11      System.out.println("我有實作抽象方法");
12    }
13 }
14
15 public class WrongAbstractChild {
16   static public void main(String argv[]{
17     Parent p = new Child(); // 企圖建立抽象類別的物件
18   }
19 }
```

執行結果

```
WrongAbstractChild.java:5: Child is not abstract and does not
override abstract method show() in Parent
class Child extends Parent { // 子類別
^
1 error
```

抽象類別、抽象方法與繼承關係

程式 AbstractChild.java 抽象子類別

```
01 abstract class Parent { // 抽象類別
02   abstract void show(); // 抽象方法
03 }
04
05 abstract class Child extends Parent { // Parent 的子類別
06   // 沒有實作show, 自動成為抽象類別
07 }
08
09 class Grandson extends Child { // Child 的子類別
10   void show() { // 實作了抽象方法
```

抽象類別、抽象方法與繼承關係

```
11      System.out.println("我有實作抽象方法");
12    }
13 }
14
15 public class AbstractChild {
16    static public void main(String argv[]){
17      Parent p = new Grandson(); // 建立子類別的物件
18      p.show();
19    }
20 }
```

執行結果

```
我有實作抽象方法
```

12 - 2 介面 (Inter face)

- 12-2-1 定義介面
  - 介面的命名和類別一樣
  - 在介面中只能定義方法的型別(傳回值)及參數型別
  - 介面通常代表某種特性

```
interface 介面名稱 {
  // 介面中的方法
}
```

定義介面

- 舉例來說，要計算地價時，當然要算出土地的面積，而「計算面積」可能是很多類別需要的功能，所以可以定義一個計算面積的介面：

```
interface Surfacing {
  double area();              // 計算面積的方法
}
```

12-2-2 介面的實作


```
interface Surfacing {
   double area();              // 計算面積的方法
}

class Circle implements Surfacing {
   ...
   public double area() {
      // 計算圓面積並回傳
   }
}
```

介面的實作

- 沿用上一章 Shape 類別及 Circle 類別的繼承架構, 並讓 Circle 實作 Surfacing 介面：

程式 ShapeArea.java 實作 Surfacing 介面

```
01 interface Surfacing {
02   double area();            // 計算面積的方法
03 }
04
05 class Shape {               // 代表圖形原點的類別
06   protected double x,y;     // 座標
07
08   public Shape(double x,double y) {
09     this.x = x;
10     this.y = y;
11   }
12
```

介面的實作

```
13    public String toString() {
14      return "圖形原點：(" + x + ", " + y + ")";
15    }
16  }
17
18  class Circle extends Shape implements Surfacing {
19    private double r;       // 圓形半徑
20    final static double PI = 3.14159;   // 圓周率常數
21
22    public Circle(double x,double y,double r) {
23      super(x,y);           // 呼叫父類別建構方法
24      this.r = r;
25    }
26
27    public double area() {// 計算圓面積
28      return PI*r*r;
29    }
```

介面的實作


```
30
31    public String toString() {
32      return "圓心：(" + x + ", " + y + ")、半徑：" + r +
33                  "、面積：" + area();
34    }
35  }
36
37  public class ShapeArea {
38    public static void main(String[] argv) {
39      Circle c = new Circle(5,8,7);
40      System.out.println(c.toString());
41    }
42  }
```

執行結果

```
圓心：(5.0, 8.0)、半徑：7.0、面積：153.93791
```

12-2-3 介面中的成員變數

- 介面也可以擁有成員變數，不過在介面中宣告的成員會自動擁有 static public final 的存取控制，而且必須在宣告時即指定初值。
- 在介面中僅能定義由所有實作該介面的類別所共享的常數。

12-2-3 介面中的成員變數

程式 InterfaceMember.java 在介面中使用成員變數

```
01 interface Surfacing {
02   double area();          // 計算面積的方法
03   double PI = 3.14159;    // 定義常數
04 }
05
06 class Shape {             // 代表圖形原點的類別
07   protected double x,y; // 座標
08
09   public Shape(double x,double y) {
10     this.x = x;
11     this.y = y;
12   }
13
```

介面中的成員變數


```
14    public String toString() {
15      return "圖形原點：(" + x + ", " + y + ")";
16    }
17  }
18
19  class Circle extends Shape implements Surfacing {
20    private double r;       // 圓形半徑
21
22    public Circle(double x,double y,double r) {
23      super(x,y);           // 呼叫父類別建構方法
24      this.r = r;
25    }
26
```

介面中的成員變數


```
27    public double area() {// 計算圓面積
28      return PI*r*r;
29    }
30
31    public String toString() {
32      return "圓心：(" + x + ", " + y + ")、半徑：" + r +
33             "、面積：" + area();
34    }
35  }
36
```

介面中的成員變數


```
37 public class InterfaceMember {
38   public static void main(String[] argv) {
39     Circle c = new Circle(3,6,2);
40     System.out.println(c.toString());
41     System.out.println("圓周率：" + Surfacing.PI);
42     System.out.println("圓周率：" + c.PI);
43   }
44 }
```

執行結果

```
圓心：(3.0, 6.0)、半徑：2.0、面積：12.56636
圓周率：3.14159
圓周率：3.14159
```

12 - 3 介面的繼承

- 12- 3 - 1 簡單的繼承
  - 就是使用extends保留字從指定的介面延伸

程式 SimpleInheritance.java 簡單的介面繼承關係

```
01 interface P { // 父介面
02   int i = 20;
03
04   void show();
05 }
06
07 interface C extends P { // 子介面
08   int getI();
09 }
10
11 public class SimpleInheritance implements C { // 實作介面
12   public void show() { // 實作由C繼承P而來的方法
```

簡單的繼承

```
13      System.out.println("變數 i 的內容：" + i);
14    }
15
16    public int getI() { // 實作C所定義的方法
17      return i;
18    }
19
20    public static void main(String[] argv) {
21      SimpleInheritance s = new SimpleInheritance();
22      s.show();
23    }
24  }
```

執行結果

```
變數 i 的內容：20
```

12-3-2 介面的多重繼承

- 可以同時繼承多個父介面,將多項特性併在一起。

程式 MultipleInheritance.java 繼承多個介面

```
01 interface P1 { // 父介面
02   int i = 20;
03
04   void showI();
05 }
06
07 interface P2 { // 父介面
08   int j = 30;
09
```

介面的多重繼承


```
10    void showJ();
11 }
12
13 interface C extends P1,P2 { // 子介面
14    void show();
15 }
16
17 public class MultipleInheritance implements C { // 實作介面C
18    public void showI() { // 實作由C繼承P1而來的方法
19       System.out.println("變數 i 的內容：" + i);
20    }
21
22    public void showJ() { // 實作由C繼承P2而來的方法
23       System.out.println("變數 j 的內容：" + j);
24    }
```

介面的多重繼承


```
25
26    public void show() { // 實作C所定義的方法
27      showI();
28      showJ();
29    }
30
31    public static void main(String[] argv) {
32      MultipleInheritance s = new MultipleInheritance();
33      s.show();
34    }
35  }
```

執行結果

```
變數 i 的內容：20
變數 j 的內容：30
```

繼承多個同名的方法

- 介面可以繼承多個父介面。

程式 NameConflict.java 繼承多個同名的方法

```
01 interface P1 { // 父介面
02   int i = 20;
03
04   void show();
05 }
06
07 interface P2 { // 父介面
08   int j = 30;
09
10   void show();
11 }
12
13 interface C extends P1,P2 { // 子介面
14   void show(String s); // 多重定義的版本
15 }
```

繼承多個同名的方法

```
16
17 public class NameConflict implements C { // 實作介面C
18   public void show() { // 實作由P1與P2繼承來的方法
19     show(""); //呼叫下面的 show(String s) 方法
20   }
21
22   public void show(String s) { // 實作C中多重定義的方法
23     System.out.println(s + "i:" + i + ",j:" + j);
24   }
25
26   public static void main(String[] argv) {
27     NameConflict s = new NameConflict();
28     s.show();
29   }
30 }
```

執行結果

```
i:20,j:30
```

繼承多個同名的成員變數

- 繼承多個同名的方法並不會有問題, 但是繼承多個同名的成員變數就有點問題了。

程式 WhoseMember.java 繼承多個同名的成員

```
01 interface P1 { // 父介面
02   int i = 20;
03
04   void show();
05 }
06
07 interface P2 { // 父介面
08   int i = 30;
09
10   void show();
11 }
12
```

繼承多個同名的成員變數

```
13 interface C extends P1,P2 { // 子介面
14   void show(String s); // 多重定義的版本
15 }
16
17 public class WhoseMember implements C { // 實作介面
18   public void show() { // 實作由P1與P2C繼承來的方法
19     show("");
20   }
21
22   public void show(String s) { // 實作C中多重定義的方法
23     System.out.println(s + "i:" + i); // 誰的i?
24   }
25
26   public static void main(String[] argv) {
27     WhoseMember s = new WhoseMember();
28     s.show();
29   }
30 }
```

繼承多個同名的成員變數

- 由於介面 P1 與介面 P2 都確確實實有一個同名的成員變數 i, 所以無法決定第 23 行究竟要顯示的是介面 P1 還是介面 P2 的 i, 因此在編譯程式時, 會發生如下的錯誤：

執行結果

```
WhoseMember.java:23: reference to i is ambiguous, both variable i
in P1 and variable i in P2 match
    System.out.println(s + "i:" + i); // 誰的i?
                                  ^
1 error
```

繼承多個同名的成員變數

- 必須在程式中明確的冠上介面名稱,
  才能讓編譯器知道所指的到底是哪一個 i：

程式 SameMemberName.java 明確指定介面

```
22    public void show(String s) { // 實作C中多重定義的方法
23      System.out.println(s + "P1.i=" + P1.i + ", P2.i=" + P2.i);
24    }
```

執行結果

```
P1.i=20, P2.i=30
```

實作多重介面

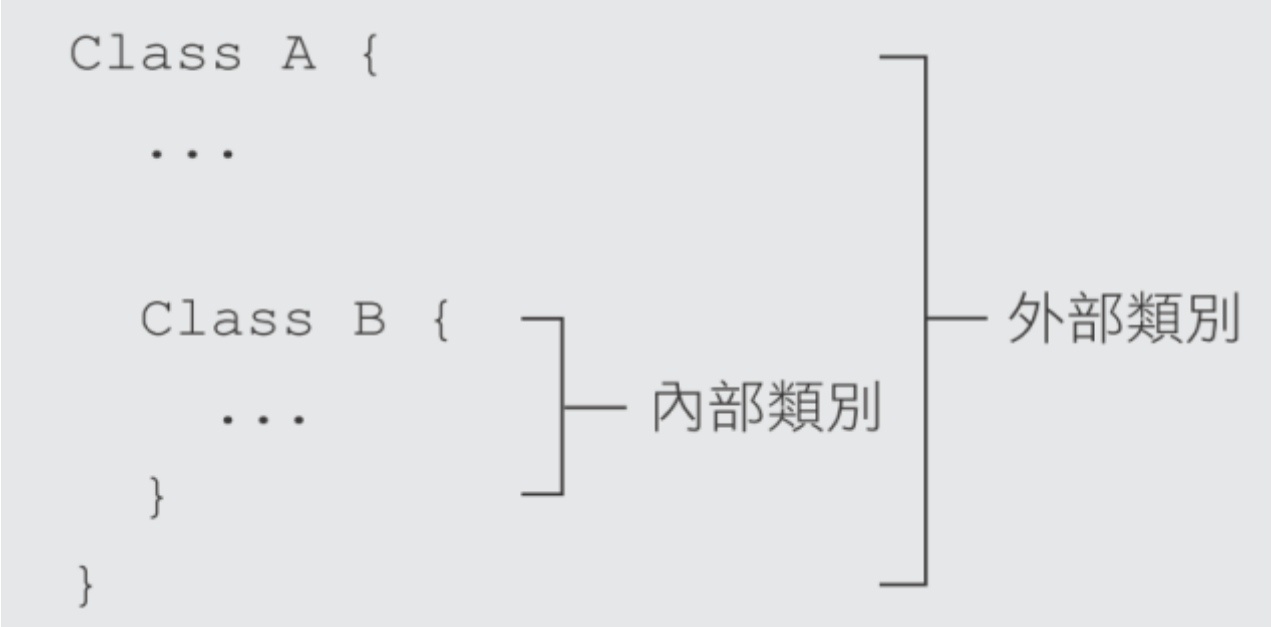
- 單一類別也可以同時實作多個介面，這時會引發的問題就如同單一介面繼承多個介面時一樣。

12 - 4 內部類別 (Inner Class)

- 12-4-1 甚麼是內部類別？
- 根據 Java 語言規格,
  『定義在另一個類別內部的類別』
  稱為巢狀類別,
  其中未被宣告為 static 的巢狀類別就稱為內部類別, 宣告為 static 的則稱為靜態巢狀類別 。

12-4-1 甚麼是內部類別？

- 相對於內部類別而言，包含住它的類別則稱為外部 (Outter) 類別，或稱外層類別。



甚麼是內部類別？

- 最大的好處是內部類別可以直接存取外部類別的所有成員，包括 private 成員在內。

程式 InnerClass.java 示範在內部類別中存取外部類別的成員，以及如何使用內部類別

```
01 class Outter {    // 外部類別
02   private int i = 1, j = 2; // 實體變數
03   static int k = 3;         // 靜態變數
04
05   class Inner {     // 內部類別
06     int j = 4, k = 5;  // 遮蓋了外部變數 j、k
07     void print() {
```

甚麼是內部類別？


```
08          System.out.print(i);                // 存取外部變數 i
09          System.out.print(Outter.this.j); // 存取被遮蓋的外部實體變數 j
10          System.out.print(Outter.k);         // 存取被遮蓋的外部靜態變數 k
11          System.out.print(j);                // 存取內部變數 j
12        }
13      }
14
15      void callInner() { // 外部類別的方法
16        Inner in = new Inner();   // 在外部類別的方法中, 必須先建立內部物件
17        in.print();               // 然後才能用它來呼叫內部類別的方法
18      }
19  }
```

甚麼是內部類別？


```
20
21 public class InnerClass {
22   public static void main(String[] argv) {
23     Outter or = new Outter();           // 建立外部物件
24     or.callInner();                     // 呼叫外部物件的方法
25     Outter.Inner ir = or.new Inner(); // 用外部物件建立內部物件
26     ir.print();
27   }
28 }
```

執行結果

```
1234
1234
```

甚麼是內部類別？

- 若只想建立內部物件，或是只想呼叫內部物件的方法，也可改用以下 2 種簡潔的寫法：

```
Outter.Inner ir = new Outter().new Inner();   // 直接建立外部及內部物件，
                                                 且外部物件用完即丟
new Outter().new Inner().print();   //直接建立外、內部物件並呼叫內部方法，
                                      且內外物件用完即丟
```

12-4-2 匿名類別 (Anonymous Class)

- 匿名類別(Anynomous Class),
  它只有類別的本體, 但沒有類別的名稱。
  也可說匿名類別是：在使用物件時, 才同時定義類別並產生物件的類別。
- 主要是用來臨時定義一個某類別的子類別, 並用以產生物件；由於該子類別用完即丟, 所以不需要指定名稱。

```
new 父類別或介面名稱() {
    // 匿名類別的定義內容
}
```

匿名類別 (Anonymous Class)

程式 AnonyDemo.java

```
01 public class AnonyDemo {
02
03   public static void main(String[] args) {
04     final int a= 10;
05
06     (new Object() {    // 匿名類別
07       int b =10000;   // 匿名類別的成員
08       public void show() {   // 匿名類別的方法
09         System.out.println ("匿名類別：");
10         System.out.println ("this   ->b= " +b);
11         System.out.println ("main()->a= " +a);
12       }
13     }).show();     // 產生匿名類別物件後
14   }                    即呼叫其 show() 方法
15 }
```

執行結果

```
匿名類別：
this   ->b= 10000
main()->a= 10
```

匿名類別 (Anonymous Class)

程式 AnonyFace.java 用匿名類別來實作介面並產生物件

```
01 interface Face {    // 定義 Face 介面
02    void smile();
03 }
04
05 public class AnonyFace {
06   public static void main(String[] argv) {
07
08     // 實作 Face 介面的匿名類別, 並建立物件傳回給變數 c
09     Face c = new Face() {
10       public void smile() {  // 實作介面中的方法
11         System.out.print("^_^");
12       }
13     };
14     c.smile(); // 以 c 物件執行匿名類別中實作的
15   }                smile() 方法
16 }
```

執行結果

```
^_^
```

匿名類別 (Anonymous Class)

程式 AnonyFace2.java 用匿名類別來實作介面、產生物件、並執行其方法

```
... ... 略 (同前一程式)
07
09      new Face() {
10          public void smile()
11              { System.out.print("^_^"); }
12      }.smile();
13    }
14  }
```

12-4-3 Lambda 運算式

- 從 Java 8 開始，若是匿名類別要實作的方法只有一個，則可改用 Lambda 來更加簡化程式，此時只需撰寫方法的參數及程式主體即可。Lambda 特別適用在並行處理及事件驅動的程式設計上。

(方法的參數) -> 方法的主體

Lambda 運算式

- 在箭頭 (->) 的前、後分別是方法的參數及主體, 參數可以有 0 到多個, 若有多個則以逗號分開, 若只有一個則可省略小括號。
- 參數

```
()                  // 無參數
a                   // 1 個參數 (小括號可省略)
(a, b)              // 2 個參數
(int a, int b)      // 指定型別的參數
```

Lambda 運算式

- 主體的部份可以是運算式或程式區塊,
  如果是運算式,
  則表示要將運算的計算結果傳回。
  如果是程式區塊, 則要以 { } 括起來,
  區塊中也可用 return 來傳回一個值。

```
() -> 58                                // 無參數, 傳回 58
a -> a * a                              // 1 個參數, 傳回 a * a 的結果
(a, b) -> { return a * b; }             // 2 個參數, 傳回 a * b 的結果
```

Lambda 運算式

- 如果程式區塊中只有一個「呼叫無傳回值的方法」的敘述，那麼也可省略{}。

```
n -> System.out.println(n);     // println() 無傳回值, 因此可省略 { }
```

Lambda 運算式

程式 LambdaFace.java 用 Lambda 來實作介面並產生物件

```
01 interface Face {    // 定義 Face 介面
02    void smile();
03 }
04
05 public class LambdaFace {
06   public static void main(String[] argv) {
07
08     Face c = () -> System.out.print("^_^"); // 用 Lambda 建立匿名
                                                 類別並產生物件
09     c.smile();  // 輸出：^_^
10   }
11 }
```

12-4-4 靜態巢狀類別 (Static Nested Class)

- 靜態巢狀類別 (Static Nested Class) 就是加上 static 的巢狀類別,
  可以直接透過『外部類別名稱.內部類別名稱』來存取內部類別中的靜能成員,
  或是直接建立內部物件,
  而不用先建立外部物件。

靜態巢狀類別(Static Nested Class)

程式 StaticNested.java 靜態巢狀類別的應用

```
01 class Outter {
02   static class Inner { // 靜態巢狀類別
03     int i = 1;
04     static int j = 2;      // 靜態變數
05     static void add(int x) { j += x; } // 靜態方法
06     void print() { System.out.print(i + "," + j); }
07   }
08 }
09 public class StaticNested {
10   public static void main(String[] argv) {
```

靜態巢狀類別(Static Nested Class)

```
11      Outter.Inner a = new Outter.Inner(); // 直接建立內部物件
12      a.i = 3;                  // 以內部物件存取該物件中的一般變數
13      Outter.Inner.j = 4;       // 以類別名稱存取靜態變數
14      Outter.Inner.add(5);      // 以類別名稱呼叫靜態方法
15      a.print();                // 以內部物件執行一般方法, 以輸出 i,j 的值
16    }
17  }
```

輸出結果

```
3,9
```

12-5 綜合演練

- 12-5-1 撰寫通用於多種類別的程式
  - 定義代表可比較特性的介面

程式 Sorting.java 定義負責比較的介面

```
01 interface ICanCompare {
02   int compare(ICanCompare i); // 進行比較
03 }
```

定義提供排序功能的類別

程式 Sorting.java (續) 提供排序功能的 Sort 類別

```
...
05 class Sort { // 提供排序功能的類別
06   static void bubbleSort(ICanCompare[] objs) { // 氣泡排序法
07     for(int i = objs.length - 1;i > 0;i--) {
08       for(int j = 0;j < i;j++) {
09         if(objs[j].compare(objs[j + 1]) < 0) {
10           ICanCompare temp = objs[j];
11           objs[j] = objs[j + 1];
12           objs[j + 1] = temp;
13         }
14       }
15     }
16   }
17 }
```

定義實作ICanCompare介面的類別

程式 Sorting.java (續) 要被排序的 Land 類別及其子類別

```
19 abstract class Land implements ICanCompare { // 父類別
20   abstract double area(); // 計算面積的抽象方法
21   public int compare(ICanCompare i) { // 實作介面的compare方法
22     Land l = (Land) i;
23     return (int)(this.area() - l.area()); // 依據面積比較大小
24   }
25 }
26
27 class Circle extends Land { // 圓形的土地
28   int r; // 半徑（單位：公尺）
29
30   Circle(int  r) { // 建構方法
31     this.r = r;
32   }
33
```

定義實作ICanCompare介面的類別

```
34    double area() { // 重新定義抽象方法
35      return 3.14 * r * r;
36    }
37
38    public String toString() {
39      return "半徑：" + r + ",面積：" + area() + "的圓";
40    }
41  }
42
43  class Square extends Land { // 正方形的土地
44    int side; // 邊長（單位：公尺）
```

定義實作ICanCompare介面的類別


```
45
46    Square(int  side) { // 建構方法
47      this.side = side;
48    }
49
50    double area() { // 重新定義抽象方法
51      return side * side;
52    }
53
54    public String toString() {
55      return "邊長：" + side + ",面積：" + area() + "的正方形";
56    }
57  }
```

撰寫測試程式

程式 Sorting.java（續）測試排序的主程式

```
59 public class Sorting {
60
61   public static void main(String[] argv) {
62     Land[] lands = {
63       new Circle(5),
64       new Square(3),
65       new Square(2),
66       new Circle(4)
67     };
68
69     for(Land l : lands) {
70       System.out.println(l);
71     }
```

撰寫測試程式


```
73        Sort.bubbleSort(lands);
74        System.out.println("排序後...");
75
76        for(Land l : lands) {
77          System.out.println(l);
78        }
79      }
80  }
```

執行結果

```
半徑：5,面積：78.5的圓
邊長：3,面積：9.0的正方形
邊長：2,面積：4.0的正方形
半徑：4,面積：50.24的圓
```

```
排序後...
半徑：5,面積：78.5的圓
半徑：4,面積：50.24的圓
邊長：3,面積：9.0的正方形
邊長：2,面積：4.0的正方形
```

12-5-2 擔任物件之間的溝通橋樑

程式 Stopwatch.java 使用介面做為物件之間溝通的橋樑

```
01 interface TimesUp {
02   void notifyMe(); // 通知時間已到的方法
03 }
04
05 class Timer { // 碼錶類別
06   static void startTimer(int seconds,TimesUp obj) {
07     // 開始計時
08     for(int i = 0;i < seconds;i++);
09     obj.notifyMe(); // 通知碼錶使用者
10   }
11 }
```

擔任物件之間的溝通橋樑


```
12
13 class watchUser implements TimesUp {// 要使用碼錶的類別
14   public void notifyMe() {
15     System.out.println("時間到");
16   }
17 }
18
19 public class Stopwatch {
20
21   public static void main(String[] argv) {
22     watchUser w = new watchUser();
23     Timer.startTimer(1000,w);
24   }
25 }
```