|  |  |
| --- | --- |
|  | [modifier1] interface 介面名稱 {      [modifier2] type 方法名稱(type param..)  } |

在 Java 中您可以使用介面(interface),定義介面的的關鍵字是 interface,語法則如下所示:

其中:

* modifier1 被使用來決定此介面相對於其它類別或介面的存取權。
* modifier2 是介面中定義的方法之權限設定,如果沒有定義,預設是 public。
* type 用來定義方法的返回值, param 用來定義參數列型態與名稱。

以實際的例子來看看如何定義一個介面,如程式碼 12-1 所示:

|  |  |
| --- | --- |
| 1  2  3  4 | public interface ContainerListener {      public void doHello();      public void doGoodbye();  } |

*程式碼 12-1 ContainerListener.java*

在第 2 行與第 3 行只定義了方法權限、返回值型態、方法名稱與空的參數列,而不用實作方法本體,撰寫介面時也是以\*.java 檔案撰寫,編譯過後也是產生\*.class 檔案。一個類別上可以實作多個介面。

• 實作介面

在定義類別時,可以一併使用 implements 關鍵字來指定要實作的介面,例如設計兩個類別,它們都實作了ContainerListener,會在被加入某個容器或被移除時顯示訊息,直接以程式碼 12-2、12-3 作為示範:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Some implements ContainerListener { //使用 implements 關鍵字來指定要實作的介面      public void doHello() {          System.out.println("Some 物件被加入...");      }        public void doGoodbye() {          System.out.println("Some 物件被移除...");      }  } |
|  |  |

*程式碼 12-2 Some.java*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Other implements ContainerListener{      public void doHello() {          System.out.println("Other 物件被加入...");      }        public void doGoodbye() {          System.out.println("Other 物件被移除...");      }  } |

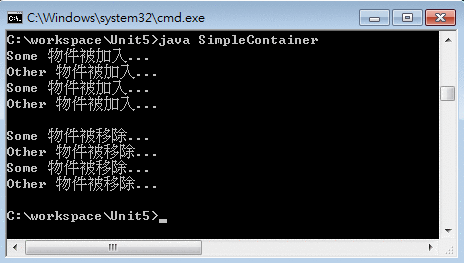
*程式碼 12-3 Other.java*

當您取得實作介面的某個物件之後,您可以將它的操作介面轉換為所實作的介面,如此就可以使用介面上所規範的方法來操作物件,例如程式碼 12-4 實作一個簡單的容器,物件的加入或移除都會呼叫 doHello()與 doGoodbye()方法。

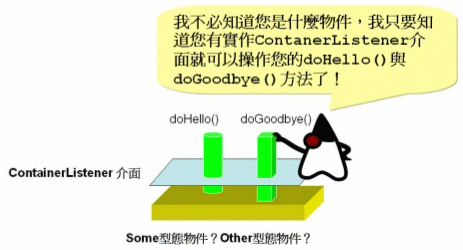
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | public class SimpleContainer {      private Object[] objArr;      private int index = 0;        // 預設 10 個物件空間      public SimpleContainer() {          objArr = new Object[10];      }      public SimpleContainer(int capacity) {          objArr = new Object[capacity];      }      // 加入物件      public void add(Object o) {  **ContainerListener listener = (ContainerListener) o; // 轉換操作介面**  **listener.doHello(); //呼叫介面上規範的方法**          objArr[index] = o;          index++;      }      // 移除物件      public void remove(int i) {  **ContainerListener listener = (ContainerListener) get(i); // 轉換操作介面**          objArr[i] = null;  **listener.doGoodbye(); // 呼叫介面上規範的方法**      }      public int length() {          return index;      }      public Object get(int i) {          return objArr[i];      }      public static void main(String[] args) {          SimpleContainer container =                  new SimpleContainer(4);          // 加入物件          container.add(new Some());          container.add(new Other());          container.add(new Some());          container.add(new Other());          System.out.println();          // 移除物件          container.remove(0);          container.remove(1);          container.remove(2);          container.remove(3);      }  } |

*程式碼 12-4 SimpleContainer.java*

執行結果如下所示:

[](http://www.codedata.com.tw/book/java-basic/images/12-3.png)*圖12-3 程式碼 12-4 執行結果*

雖然 Some 與 Other 兩個是不同的類別,但由於它們都實作了 ContainerListener 介面,因而在 SimpleContaner 類別的add()與 remove()方法中,都可以在轉換操作介面之後,順利的操作 Some 物件與 Other 物件的 doHello() 與doGoodbye()方法。

[](http://www.codedata.com.tw/book/java-basic/images/12-4.png)*圖12-4 利用介面規範操作方法*

• 介面的繼承

介面也可以進行繼承的動作,同樣也是使用"extends"關鍵字,例如:

|  |  |
| --- | --- |
| 1  2  3 | public interface 名稱 extends 介面 1, 介面 2 {      // ...  } |

不同於類別的是,介面可以同時繼承多個介面,如果有一個A 介面繼承了 B、C 介面,則實作 A 介面的類別,對於 B、C 介面中規範的方法也必須一併實作。