介面 interface

> 介紹Java中interface的用法，及其物件導向意義。

何謂介面？ What is Interface?

舉個現實中的例子，冰箱跟烤箱都是常見的電器設備，然而台灣電力公司要如何供給一般家庭所需的電力呢？ 沒錯，就是110/220伏特的交流電，而且有一個標準的電力接頭，冰箱跟烤箱只需要接受這個電力規格，就可以使用，不管內部作了什麼處理，使用者只需要為他們接上制訂好的電源。

所以我們可以說，冰箱跟烤箱擁有相同的『行為』，在這裡指的就是接受並運用這個110/220伏特的交流電。

介面，描述不同類別的共通行為。

為什麼需要介面？ Why Interface?

假設電器設備指的是可以從台灣電力公司取得需要的電源。

冰箱跟烤箱都可以接受台灣電力公司的電源，這兩個東西都可以視為電器設備，但這兩個東西在繼承關係上可說是相差很遠的。

例如：

物件 -> 冷藏設備 -> 冰箱。

物件 -> 加熱設備 -> 烤箱。

由於**Java不允許多重繼承**(任何類別只能有一個父類別)，所以沒辦法讓冰箱既是冷藏設備又是電器設備。

無法多重繼承：

```
class 冰箱 extends 冷藏設備,電器設備{  // 編譯錯誤，不允許多重繼承
}
```

從現實中來看，冰箱跟烤箱只有『使用台電提供的電』這個共通的行為，因此我們把電器設備定義為介面，如此一來冰箱跟烤箱都可以『實作』這個介面，從而具有這個介面的特性，台電也很清楚的知道只要送出的電可以讓『電器設備』這個介面使用就好，任何需要電的設備只要符合這個介面制訂的規則，就可以獲得電力。

```
class 冰箱 extends 冷藏設備 implements 電氣設備{ // 繼承冷藏設備，實作電器設備這個介面
    // 成員宣告
}
```

如此一來，冰箱就可以視為電器設備，當作電器設備來使用。

宣告介面 Declare interface

在Java中，我們可以把介面interface當成一種很特別的類別。

宣告方式：

```
[修飾子] interface 介面名稱 {
    // 成員定義
}
```

範例程式：

```
public interface Power{
    void getPowerFromTP();
}
```

介面的宣告就是使用關鍵字interface，跟類別很像都是走差不多的格式，其中方法的定義只能宣告『方法的原型』，就像在『抽象』的章節中提到的抽象方法一樣。

實作介面 implements interface

利用關鍵字implements，來讓類別實作某個介面，類別需要定義好該介面所制定的方法。

範例程式：

```
class Refrigerator implements Power{
    public void getPowerFromTP(){
        System.out.println("轉換..轉換....")
    }
}
```

必須實作『所有』該介面宣告的方法，否則會編譯錯誤。

此時這個類別就可以被視為該介面來使用。

```
Power obj = new Refrigerator();
obj.getPowerFromTP();
```

執行結果：

```
轉換..轉換....
```

實作多個介面

在Java不允許多重繼承，但同一個類別可以實作多個介面，算是彌補了不能多重繼承所帶來的不方便。

實作多個介面用『,』隔開，寫上介面名稱，並且該類別必須實作每個介面所制定的方法。

使用格式：

```
class 類別名稱 implements 介面1,介面2,...介面n{
    // 成員定義
}
```

範例程式：

```
interface A{
    void Amethod();
}
interface B{
    void Bmethod();
}
class MyClass implements A,B {
    public void Amethod(){

    }
    public void Bmethod(){

    }
}
```

內訂的修飾子

在interface中定義的欄位、方法都有規定好的修飾子，可以寫也可以不寫，但不能衝突。

介面方法的修飾：

```
public abstract 回傳型態 方法名稱();
```

程式設計師可以自己寫，但不能與之衝突。

```
interface MyInterface{
    public abstract void A();  // ok
    public void B();           // ok，編譯完會自行在執行檔加上 public abstract
    void C();                  // ok，編譯完會自行在執行檔加上 public abstract
    private void D();       // no ok, 編譯錯誤，修飾子不正確
    void E(){};             // no ok, 編譯錯誤，只能定義原型，不行定義方法本體
}
```

介面欄位的修飾：

```
public static final 資料型態 變數名稱 = 值;
```

跟方法一樣，可以省略修飾子，但不能與之衝突。

```
interface Qoo{

    int value = 10;
    public int value2 = 20;
    public static int value3 = 30;
    public static final int value4 = 40;
    // 以上四行敘述，編譯完成後在執行檔中修飾子均為 public static final

    int value5;  // 編譯錯誤，必須給訂初始值
    private int value6 = 60; // 編譯錯誤，private與public衝突
}
```

介面欄位的存取

上面提到，可以在介面中宣告資料欄位並且修飾子限定是public static final，要如何使用呢？

```
介面名稱.欄位名稱;
```

嗯，就把介面當成一種特殊的類別，就是存取靜態欄位的方法。

程式範例：

```
interface Power{
    int value = 10;
}
class Test{
    public static void main(String[] args){
        System.out.println(Power.value);
    }
}
```

執行結果：

```
10
```

要注意的是在介面中的欄位都是public static final的修飾，所以不能改變其值，一般都是當作該介面的常數來使用。

同名欄位的存取

一個類別可以實作多個介面，最多繼承一個類別，所以加上自己定義的欄位，可能會有許多相同名字的變數，該如何存取？

```
interface Inter1 {
    int value = 10;
}
interface Inter2 {
    int value = 20;
}
class Father{
    int value = 30;
}
class A extends Father implements Inter1,Inter2{
    int value = 40;
}
```

存取每個同名變數的範例程式：

```
class Test {
    public static void main(String[] args) {

        A a = new A();
        System.out.println( Inter1.value );  // 以介面名稱存取，因為是static修飾
        System.out.println( Inter2.value );  // 同上
        System.out.println( ((Father)a).value ); // 先把物件a轉型成該父類別，再存取
        System.out.println( a.value );  // 直接以物件存取

    }// end of main(String[])
}// end of class Test
```

執行結果：

```
10
20
30
40
```

同名方法的實作

一個類別可以實作多個介面，但介面中如果有相同的方法呢？

```
interface A{
    void method();
}
interface B{
    void method();
    int method(int a);
}
class MyClass implements A,B{
    public int method(int a) {
        //...
    }
    public void method() {
        //...
    }
}
```

根據方法的多載，相同名稱、不同參數即視為不同的方法，一模一樣的方法實作一個即可，關鍵點是每個不同的方法都必須實作。

介面的繼承

沒錯，介面也可以繼承，而且還允許**多重繼承**！

```
interface 介面名稱 extends 介面1,介面2,...,介面n{

}
```

彼此繼承的結果就是，實作該介面的類別必須實作每個定義的方法。

程式範例：

```
interface A{
    void a();
}
interface B{
    void b();
}
interface C extends A,B{  // 介面的繼承，繼承多個介面以逗號『,』隔開
    void c();
}
class MyClass implements C{ // 必須實作介面C及其所有父類別定義的方法
    public void a() {
    }
    public void b() {
    }
    public void c() {
    }
}
```

抽象類別實作介面

抽象類別也是個類別，當然也可以實作介面。

然而抽象類別中可以定義該介面宣告的方法的本體，也可以不定義。沒定義的話就是由繼承這個抽象類別的子類別要實作所有抽象方法。

程式範例：

```
interface A{
    void a();
    void b();
}
abstract class AbsClass implements A{
    public void b(){
        System.out.println("hello b~");
    }
    abstract void c();
}
class MyClass extends AbsClass{

    public void a() {
    }
    // 方法b()已經在AbsClass實作，MyClass不需要再實作，當然也可以再覆寫b()
    public void c() {
    }

}
```

原則就是，非抽象類別要實作所有未定義的方法。

抽象類別與介面的比較 Abstract Classes Compared to Interfaces

抽象類別與介面有點像，兩個都不行被實體化成物件。

抽象類別

用於被繼承，子類別要實作定義的抽象方法。

抽象類別中可以定義完整的方法(方法本體)，也可以只定義方法原型。

可以定義完整的資料欄位供繼承類別使用。

設計中心以資料為主體。

介面

用於被實作，子類別要實作定義的方法。

介面中只能定義方法原型，不能有方法本體。

方法的修飾子必為public abstract，欄位的修飾子必為public static final，可省略不可衝突。

定義的資料欄位用於作為常數使用。(因為修飾子為public static final)

設計中心以方法(行為)為主體。

一般來說有共同的概念可以繼承相同的抽象父類別，

若只是行為相同以介面來設計會比較恰當。

實務上先考慮介面的劃分會比較方便，畢竟類別可以繼承多個介面，需要用到層層的欄位概念再使用類別去繼承。