為甚麼Java提供Interface

雖然程式語言提供了基本資料型別,但由於各種應用都有其特定的資料結構需求,因此程式語言都提供使用者自訂型別的能力。型別自訂後,其使用的方法和基本資料型態類似。Class就是一種使用者自定的型別。Java提供了instanceof的保留字,用以判斷某reference所指到的物件,其型態和某Class是否相容:

```
Object ref;
ref = new Bird();
if (ref instanceof Animal) { // correct
    System.out.println("ref is currently pointing to an Animal Object.");
}
```

在物件導向的觀念裡,物件可以具有多個型別。例如「附有橡皮擦的鉛筆」,可以當成是「書寫工具」,也可以當成是「擦拭工具」。物件可有多種型別的觀念,不僅在日常生活中常見,在軟體開發上也有實際的需求。要使物件具有多種型別,可透過繼承來達成。例如Bird物件就同時具有Bird和Animal兩種型別。由於Java的Class只能有單一繼承,因此像「附有橡皮擦的鉛筆」同時具有「書寫工具」和「擦拭工具」兩種互不相關的型別,就無法透過Class的單一繼承來達成了。

許多語言提供Class的多重繼承,但Java考量諸如下面的多重繼承問題,選擇不引進Class多重繼承:

假設B繼承A,C繼承A,D又多重繼承B,C, 該語言又使用virtual function則

- 如果B, C都有overwrite A的methodM方法, 而A ref指到D類別的物件, 請問透過ref傳遞methodM訊息時, 應該使用B還是C的methodM?

在不引進Class多重繼承的前提下,為了讓物件具有多種型態,Java提供了Interface(界面)的觀念。Interface可視為沒有實作的自訂型別,和Class用來作為Object的模板,有所不同。Class可以宣告實作多個Interface,而Interface之間可以有多重繼承。

Java有關Interface的語法

- 宣告Interface

  ```
  public interface Listener {
      double PI = 3.14149; // 同public static final
      void listen(); // 同public abstract
  }
  public interface Runnalbe {
      int PERIOD = 10;
      void run();
  }
  public interface AnotherRun {
      int PERIOD = 20;
      void run();
      int run(int);
  }
  ```

  注意上述函數宣告沒有{},也就是說沒有實作的意思。

- Interface的繼承

  ```
  public interface ActionListener extends Listener {
  }
  public interface MultiInterface extends Listener, Runnalbe {
  }
  ```

- Class實作Interface的宣告

  ```
  public class A implements Listener {
      public void listen() {
      }
  }
  public class B implements Listener, Runnable {
  ```

```
        public void listen() {
        }
        public void run() {
        }
    }
    public class C implements MultiInterface {
        public void listen() {
        }
        public void run() {
        }
    }
    public class D extends A implements Runnable, AnotherRun {
        public void run() {
        }
        public int run(int period) {
        }
    }
```

Interface如同Class一樣可以作為一種型態的宣告,因此如下的判斷都是正確的

```
D ref = new D();
ref instanceof D; // true
ref instanceof Runnable; // true
ref instanceof AnotherRun; // true
ref instanceof A; // true
ref instanceof Listener; // true
```

Interface中宣告的變數具有以下特質

- public。所謂Interface(界面)指的是外界觀看某物件時,所能看到的表象以及溝通的管道,因此Interface內的成員一定是public。也就是說即便宣告時沒寫public關鍵字,Compiler也會幫我們加上去。
- static。既然Interface沒有實作,就不可能透過Interface產生物件。換言之,Interface內的變數一定是屬於Class,而不屬於Object。
- final。Interface可視為一種約定或契約,我們自然不希望裡面的variable可以隨便更改。

Interface中宣告的method具有以下特質

- public。同變數說明。
- abstract。Interface沒有實作,裡面定義的method只是宣告而已。沒有實作的method,在Java裡用abstract這個關鍵字來表達。有關abstract的詳細說明,請見下一節

當Interface繼承多個Interface,或Class實作多個Interface時,如果有多個同名的函數或變數時,應該如何處理? 例如Runnable和AnotherRun這兩個界面都定義了變數PERIOD和方法run。

- 相同變數名稱:由於interface內的變數具有static的性質,因此使用這些變數時,必須加上Interface的名稱才行,如Runnable.PERIOD,AnotherRun.PERIOD,因此不會造成任何混淆。
- 相同函數名稱:如果signature(參數個數,型態以及傳回值型態)完全相同,則Class只要實作一次即可,例如Runnable和AnotherRun均定義void run(),因此Class D只要實作一次就好了。如果同名函數符合Overloading,把它們分別當成不同的method即可。如果參數完全相同,但傳回值不同,則違反了Overloading的原則,會產生Compile Error。

Abstract Class and Method

只有參數宣告,沒有實作的方法,稱為abstract method。某些情況下,雖然有實作,但我們希望強迫子類別必須override該方法時,也可以宣告為abstract method。Interface裡的方法一定沒有實作,因此必然為abstract method。

如果Class裡有一個以上的abstract method,則該class必須宣告為abstract。有時候即使沒有abstract method,也可以宣告該class為abstract。我們不可以直接new該class的物件,只能new其子類別物件。

```
public abstract class AbstractExample {
    int x;
    public void abstract abstractMethod() {
```

```
    }
    public AbstractExample() {
        x = 5;
    }
}
public class SubClass extends AbstractExample {
    public void abstractMethod() { // must override this method, or SubClass be declared as abstract class
        x = 10;
    }
}
public class Main {
    public static void main(String[] argv) {
        AbstractExample a = new SubClass(); // correct
        a.abstractMethod(); // virtual function, call SubClass's abstractMethod
        a = new AbstractExample(); // Compile error, you can't new abstract class
    }
}
```

綜合以上所述,可列出以下幾點特徵

- 具有abstract method的class必須宣告為abstract class。
- 繼承abstract class的子類別必須override所有父類別的abstract method,否則子類別也必須宣告為abstract class。
- 實作Interface A的Class必須實作A裡的所有method,否則必須宣告自己為abstract class。
- 不能直接new abstract class,只能new其非abstract class的子類別。