Java 使用方法

使用方法

1. 方法的宣告與呼叫

(1). 宣告方法

語法 :

[modifiers] return_type method_identifier ([arguments]) {

method_code_block

}

// method_identifier : 方法名稱

// return_type : 回傳型別，必要

// [modifiers] : 修飾詞，非必要

// [arguments] : 輸入參數，非必要

範例 :

```
   class Shirt1{
        public void display() {

        }
    }

/* 1. 方法名稱之後必加()，是和屬性的區別
 * 2. 回傳型態使用void，表示沒有回傳
 * 3. () 之內沒有內容，表示該方法沒有輸入參數
 */
```

(2). 呼叫方法

如要呼叫類別方法，則必須建立該類別的物件，取得參考物件(遙控器)後，使用「.」運算子，呼叫該方法

範例 :

```
public class ShirtTest {

    public static void main(String[] args) {
        ShirtTest myShirt = new ShirtTest();
        myShirt.display();

    }
```

```
    public void display() {

    }

}
```

2. 使用方法的好處

1. 方法是物件的行為。設計類別時，需要實作方法讓物件可以表現自己的行為。 ex : 上樓、下樓、關門、開門
2. 方法應該具備獨立的功能或邏輯性，讓程式可讀性高，並易於維護
3. 可以增加程式「可重複使用性(re-usable)」，減少重複的程式碼
4. 透過caller 和 worker之間的呼叫，讓不同物件可互動

宣告static 方法和變數

- Java 是物件導向的程式語言，需要以類別(class)產生物件(object)後，才能使用物件的屬性和方法
- static 關鍵字，翻譯為「靜態」，所以「靜態方法或變數」，就是指加上static 修飾詞後的方法或變數
- static 修飾詞在物件導向的程式開發裡是一個很另類的存在，因為在類別設計裡若把屬性或方法加上static修飾詞，則該屬性或方法，使用時就不需要再透過物件生成! 亦即直接使用類別，就能呼叫static 方法和變數

範例 : (沒有 static 時的困境)

```
public class Circle0 {

        private double radius;
        final double PI = 3.1415926;

        public void setRadius(double r) {
        this.radius =r;
        }

        public double getArea() {
            return this.radius * this.radius * PI;
        }

        public static void main(String[]args) {
            Circle0 c1 = new Circle0();
            c1.setRadius(1);
            System.out.println(c1.getArea());
            Circle0 c2 = new Circle0();
            c2.setRadius(10);
            System.out.println(c2.getArea());
        }


}
/* Output :
 * 3.1415926
```

```
 * 314.15926
 */
```

1. 使用static 解決問題

(1) 為了解決「PI變數無法共享」的問題，使用static 欄位

(2) 為了解決「需要使用物件計算面積」的問題，新增static 方法

範例 :

```
public class Circle1 {


    private double radius;
    static final double PI = 3.1415926;


    public void setRadius(double r) {
        this.radius = r;

    }

    public double getArea() {
        return this.radius * this.radius *PI;
    }

    public static double areaFormula(double r) {
        return r *r *PI;
    }

    public static void main(String[] args) {
        System.out.println(Circle1.PI);
        System.out.println(Circle1.areaFormula(1));
        System.out.println(Circle1.areaFormula(10));
    }

}

/* Output :
 * 3.1415926
 * 3.1415926
 * 314.15926
 */
```

2. static 宣告的意義

先前範例，加上static 修飾詞後，該欄位或方法在記憶體裡只會有一份，讓所有物件共享。這唯一的一份資料是放在「類別」裡

- 共享於Circle 類別上的static 方法與屬性，不只 Circle 物件可以使用，其他物件也同樣可以使用

類別內的 「欄位和方法」的宣告都可以使用 static 修飾詞，所以定義在類別內的所有屬性和方法，可以由「static 宣告」的 「有/無」分成兩類 :

(1) 未使用 static 宣告

使用這類的欄位和方法時都必須先產生物件，再使用物件參考(遙控器)去呼叫。因為和「物件」息息相關，所以稱「無static宣告的欄位和方法」為「物件成員(object member)」

(2) 使用 static 宣告

因為只用「類別」名稱就可以呼叫方法和欄位，因此稱「有 static 宣告的欄位和方法」為「類別成員(class member)」

關於「物件成員」和「類別成員」有幾件事要注意 :

(1) 比較

成員分類	使用步驟
類別成員	1. 類別定義(*.class)載入JVM 2. 以類別名稱呼叫
物件成員	1. 類別定義(*.class)載入JVM 2. 使用類別定義產生物件 3. 以物件參考呼叫

(2) 同一類別哩，只要有類別存在就可以使用類別成員，物件成員卻必須以類別建立物件後才能使用，因此先有類別成員，才有物件成

員。所以物件成員可以呼叫類別成員，但類別成員不能呼叫物件成員。犯錯編譯器會顯示 :

non-static variable XXX cannot be referenced from a static context 或

non-static method XXX() cannot be referenced from a static context

(3) 因為每個類別只載入 1 次，所以類別成員在 JVM裡是唯一存在

(4) 因為類別成員比物件成員更早存在，語法上也可以透過物件參考 (遙控器) 取得 static 成員，但不建議，因為容易造成混淆

3. static的宣告時機

- 實務上只要方法 (method) 的內容不涉及物件狀態，亦即不含物件成員，方法的執行結果只和輸入參數有關，就可以考慮使用static 宣告
- 「main」方法，因為被賦予啟動 Java SE 的程式的特殊功能，不屬於物件自己的行為，也是使用「static 」宣告 :

語法：

public static void main (String [] args) {

    method_code_block

}

範例：

```
public class Circle1 {

    private double radius;
    static final double PI = 3.1415926;


    public void setRadius(double r) {
        this.radius = r;

    }

    public double getArea() {
        return this.radius * this.radius *PI;
    }

    public static double areaFormula(double r) {
        return r *r *PI;
    }

    public static void main(String[] args) {
        System.out.println(Circle1.PI);
        System.out.println(Circle1.areaFormula(1));
        System.out.println(Circle1.areaFormula(10));
    }

}

/* Output :
 * 3.1415926
 * 3.1415926
 * 314.15926
 */
```

範例：

```
public class Circle2 {
    private double radius;
    //static final double PI = 3.1415926;

    public void setRadius(double r) {
        this.radius = r;
    }
```

```
    public double getArea() {
        return areaFormula(this.radius);
    }

    public static double areaFormula(double r) {
         return r * r * Math.PI;      // 改用 Math.PI
    }

    public static void main(String[] args) {
        System.out.println(Circle2.areaFormula(1));
        System.out.println(Circle2.areaFormula(10));
    }

}

/* Output :
 * 3.141592653589793
 * 314.1592653589793
*/
```

4. Java 使用 static 的範例

Math 類別 :

1. 指數 : Math.exp(double a)
2. 對數 : Math.log(double a)
3. 三角函數 : Math.sin(double a)
4. 隨機浮點數 : Math.random()
5. 數學常數 : Math.PI

System 類別 :

1. 取得環境變數 : System.getenv()
2. 取得標準輸入輸出串列 : System.out
3. 結束程式 : System.exit()

建立多載的方法

1. 方法的簽名

語法 :

[modifiers] return_type method_identifier ([arguments]) {

}

1. 方法名稱 (method_identifier)
2. 參數 (arguments)

合併為「方法簽名 (method signature)」

- 類別內真正用來識別方法的，不只「方法名稱」，還必須加上「參數」
- 參數的數量可以有零至多個 ，因此要認定「方法參數」是否相同有三個考量方向：
    1. 數量
    2. 順序
    3. 型別
        - 只要有一個不同，就算不同。「參數名稱」無須考量

2. 方法的多載

- 定義：同一個類別內，若有方法名稱相同，但簽名不同 (否則無法編譯) ，就稱為「多載 (overloading)」
- 使用時機： 若類別內有多個方法 「功能相近」，只是傳入的參數型態、數量不同，就可以使用多載 (overloading) 進行程式設計

範例： (使用多載)

```
public class Calculator {

    public static int sum( int i1, int i2) {
        return (i1 + i2);
    }

    public static float sum( float f1, int i1) {
        return (f1 + i1);
    }

    public static float sum( int i1, float f1) {
        return (i1 + f1);
    }


    public static void main(String[] args) {
          int totalOne = Calculator.sum(2, 5);
          System.out.println(totalOne);
          float totalTwo = Calculator.sum(12.9f, 12);
          System.out.println(totalTwo);
          float totalThree = Calculator.sum(12, 12.9f);
          System.out.println(totalThree);
    }

}

/* Output :
 * 7
 * 24.9
 * 24.9
*/
```

範例：(沒使用多載， 必須準確呼叫每一種加總的方法，不能弄錯)

```
public class Calculator2 {
    public static int sumForInt( int i1, int i2) {
        return (i1 + i2);
    }

    public static float sumForFloatAndInt( float f1, int i1) {
        return (f1 + i1);
    }

    public static float sumForIntAndFloat( int i1, float f1) {
        return (i1 + f1);
    }

    public static void main(String[] args) {
          int totalOne = Calculator2.sumForInt(2, 5);
          System.out.println(totalOne);
          float totalTwo = Calculator2.sumForFloatAndInt(12.9f, 12);
          System.out.println(totalTwo);
          float totalThree = Calculator2.sumForIntAndFloat(12, 12.9f);
          System.out.println(totalThree);

    }

}

/* Output :
 * 7
 * 24.9
 * 24.9
*/
```

變數值的傳遞

1. 變數值傳遞的發生場景

Java 在兩種情況時需要傳遞 (pass) 變數 / 參數 :

(1) 由指定運算子「=」右側，將值 (value) 傳遞給左側變數

(2) 透過方法宣告的參數，將值 (value) 由呼叫者 (caller)方法傳遞進入工作者 (worker)方法中

2. Pass By Value = Pass By Copy

傳遞參數 / 變數時，無論「基本型別」，和「參考型別」都是「複製」變數本身的 「值」做傳遞，所以都稱為「Pass by Value」。但兩者對值的定義不同，因為值都會經過複製，所以也稱為「Pass by Copy」

1.參考型別

　若變數屬於參考型別，則複製物件參考 (遙控器) 後進行傳遞。複製前後雖遙控器不同，但指向同一物件。

2.基本型別

　若變數屬於基本型別，因為沒有遙控器概念，因此直接複製變數值，如同影印機複製原稿後產生副本，兩者各自獨立。