Java 認識參考型別與操作物件

使用物件參考

1. 建立遙控器的概念

要使用「物件」，必須使用該物件的「物件參考 (object reference)變數」或簡稱「物件參考」或「參考變數」。概念上類似使用「遙控器」由遠端操控「電子產品」。

2. 由類別建構物件

三個程序 :

1. 宣告 (Declaration)

   語法 :

   ```
   Classname reference (物件參考名稱) ;
   ```

   ex :

   ```
   Shirt  myShirt ;
   // Shirt 為類別名稱
   // myShirt 為物件參考變數
   ```

2. 實例化 (Instantiation)

   語法 :

   ```
   new  Classname () ;
   ```

   ex :

   ```
   new   Shirt() ;
   // Shirt 為類別名稱，將使用該類別產生物件實例
   ```

3. 將實例指定給物件參考，完成初始化 (Initialization)

   語法 :

   ```
   reference = new Classname () ;
   ```

   ex :

   ```
   myShirt  =  new Shirt () ;
   // 因為無法直接觸碰記憶體裡的 Shirt 物件，故使用物件參考 myShirt 來控制
   ```

- 取得物件參考 (object reference)後，等同取得「遙控器」，可以控制實例化時在記憶體產生的物件。使用「.」運算子存取物件欄位或呼叫物件方法 :

  ex :

  ```
  int  shirtId  =  myShirt.shirtId ;
  myShirt.display() ;
  ```

- 基本 / 參考型別對照表

  Java 是重視型別的程式語言，任何「變數」都需要宣告型別。

項目	型別	變數	指派運算子	記憶體內容
基本型別範例	int	x	=	10
參考型別範例	Shirt	myShirt	=	new Shirt ()

3. 對不同物件使用不同參考

若不同種類物件，甚至同種但不同的物件，都該使用各自的物件參考 / 遙控器對物件進行控制：

範例：

```
Shirt myShirt1 = new Shirt() ;
myShirt1.display() ;
Shirt myShirt2 = new Shirt() ;
myShirt2.display() ;
Trousers myTrousers = new Trousers() ;
myTrousers.display() ;
```

4. JVM 記憶體分類

Java 記憶體分三大區塊：

1. Global (全域)
   - 保存 static 的類別成員變數
2. Stack (堆疊)
   - 保存基本型別 (primitive type) 的變數和變數內容 (value) 的地方
   - 保存參考型別 (reference type)的變數的地方
3. Heap (堆積)
   - 保存參考型別 (reference type)的變數內容 (instance)的地方

分類	變數 & 變數值	Stack (堆疊)	Heap (堆積)
基本型別	變數	*	
	值	*	
參考型別	變數 (物件參考)	*	
	值 (物件實例)		*

範例：

```
public class Reference {
```

```
    private int  price;

    public static void main(String[] args) {
      Reference shirt1 = new Reference ();
      Reference shirt2 = new Reference ();
      shirt1 = shirt2;        // shirt1 改指向 shirt2 的物件實例
      shirt1.price = 1000;    // 沒有任何物件參考使用，一段時間後會GC
      shirt2.price = 500;
      System.out.println("Shirt price : "+ shirt1.price);
    }

}
 /* Output :
  * Shirt price : 500
 */
```

使用 String 類別

Java 的基本型別已經有字元 (char) ，但 char 只能使用單一字元。對於兩個以上字元組成有落差。Java 特別提供 String 類別供程式設計師使用

1. String 類別支援非標準的語法

String 類別屬於參考型別，使用時會產生物件。為避免相同字串物件產生太多而造成記憶體空間浪費，Java 內建字串的快取 (cache) 機制，將字串存在於「字串池 (String Pool)」中 ，讓相同字串可以在大部分情況下直接由字串池中重複取用且用畢歸還，避免內容相同的字串物件一再產生。

String 物件可以不需要使用「new」關鍵字進行實例化，改直接以類似宣告和初始char 變數的方式進行。

ex :

char c = 'J' ;

String s1 = " Java " ;

String s2 = " Java " ;

2. String 類別是immutable(不可改變)

範例 :

```
String  name1 = "Jim"                                    //使用字面常量 (String literal)
時，將自動生成 String Object
String name2 = name1 + " is teaching" ;                   // 使用運算子「+」讓字串相連
String name3 = name1.concat(" is teaching") ;             // 也可以使用 String 類別內件方
法 「concat」讓字串相連
```

- 注意！因為字串是 「immutable (不可改變)」，字串相連並非讓「原字串物件」連接新字串，而是「將原字串複製一份」後再加上相連的字串

- 注意 ! String 類別是 「immutable」，亦即實例 (instance)一旦建立後就無法變更其狀態的觀念非常重要

範例 :

```
public class ImmutableDemo {

    public static void main(String[] args) {
        String name1 = "Jim";
        name1 = name1.concat(" is teaching");
        System.out.println(name1);

        String name2 = "Jim";
        name2.concat(" is teaching");  // concat 後的結果沒有指回原遙控器(物件參考)，等同做白工，
                                       // 因為遙控器(物件參考)仍指向改變前的物件。所以 name2 = Jim
        System.out.println(name2);

    }

}

/* Output :
 * Jim is teaching
 * Jim
*/
```

- 最大的差別是否將字串 concat ( )後的結果指回原物件參考，不然等於做白工

3. String 類別的其他方法

1. 使用「+」或 方法「concat」相連字串
2. 使用 length() 取得字串長度
3. 使用 toUpperCase() 或 toLowerCase() 將字串內的字元轉換為全部大寫或全部小寫
4. 使用 trim() 去除字串前後空白
5. 使用 substring () 由字串內取出部分字串
6. 使用 endsWith() 判斷字串結尾

範例 :

```
public class StringTest {

    public static void main(String[] args) {
        String name = "Jim Tzeng ";
```

```
        int length = name.length();
        System.out.println(length);
        name = name.trim();
        String lc = name + " TEACHES".toLowerCase();
        lc =(name + " TEACHES").toLowerCase();
        System.out.println(lc);
        String lastName = name.substring(4);
        System.out.println(lastName);
        System.out.println(lc.substring(4,12));
        boolean end = name.endsWith("Tzeng");
        System.out.println(end);
        }

}

/* Output :
 * 10
 * jim tzeng teaches
 * Tzeng
 * tzeng te
 * true
 */
```

使用 StringBuilder 類別

String 類別，immutable (不可改變)的特性和影響。 Java 為字串提供一個mutable (可改變)的選項，StringBuilder類別

StringBuilder 類別基本特性 :

1. 大部分方法都回傳自己的參照，沒有實例化的成本 。

- 因為String 是immutable ，因此若方法是回傳字串的，如 : subString()、trim()等，其實都是回傳一個新物件，因此增加了實例化的成本。

2. 必須使用「new」關鍵字進行物件實例化
3. 提供字串存取的擴充方法 : append()、insert() 、delete()
4. StringBuilder 類別可以透過 append()方法一直增加長度。若程式開發時已經約略知道字串最大長度，則建立時可以提供最佳化「初始長度 (initial capacity)」設定，依次預留足夠長度， 可以避免字串增長時造成的效能損失

即便 StringBuilder類別有許多優異的特質， String 類別依然不可缺少。理由是 :

1. 使用 immutable 物件較安全
2. String 類別在 Java 一推出時即存在，有眾多類別依然需要
3. 擁有比 StringBuilder 類別更多的方法

範例 1 :

```
public class StringBuilderDemo {

    public static void main(String[] args) {
        StringBuilder sb1 = new StringBuilder(8);
        sb1.append("jim");
        sb1.append(" ");
        sb1.append("tzeng");
        System.out.println("sb1: " + sb1.toString());
        System.out.println("sb1 object capacity: " + sb1.capacity());
        System.out.println("sb1 sub string: " + sb1.substring(0,5));
        System.out.println("sb1 sub string: " + sb1.substring(0,10));

    }

}


/*  Output :
 *  sb1: jim tzeng
 *  sb1 object capacity: 18
 *  sb1 sub string: jim t
 *  Exception in thread "main" java.lang.StringIndexOutOfBoundsException: start
0, end 10, length 9
 *  at
java.base/java.lang.AbstractStringBuilder.checkRangeSIOOBE(AbstractStringBuilder
.java:1810)
 *  at
java.base/java.lang.AbstractStringBuilder.substring(AbstractStringBuilder.java:1
070)
 *  at java.base/java.lang.StringBuilder.substring(StringBuilder.java:87)
 *  at StringBuilderDemo.main(StringBuilderDemo.java:12)
*/
```

範例 2 :

```
public class StringBuilderDemo {

    public static void main(String[] args) {

       StringBuilder sb2 = new StringBuilder();
        sb2.append("123456789");
        sb2.insert(3,"-");
        sb2.insert(7, "-");
        System.out.println("sb2: " + sb2.toString());
        StringBuilder sb3 = new StringBuilder("12345678");
        sb3.delete(3, 5);
        System.out.println("sb3: " + sb3.toString());
    }
```

```
}

/*  Output :
 *  sb2: 123-456-789
 *  sb3: 123678
*/
```

Java API 文件介紹

- API 是 Application Programming Interface的縮寫，一般翻譯為「應用程式介面」
- 提供給應用程式開發人員的說明文件，程式開發人員可以由文件中了解程式碼如何使用與呼叫，無須了解底層的原始碼為何、或理解其內部工作機制的細節

基本型別的包覆類別

1. 包覆類別的由來與使用

- 參考型別的優勢在於生成物件後，可以擁有屬性和方法，可以從事很多方便的處理
- 基本型別只能用於單純的計算機計算
- Java 是物件導向的程式語言，以參考型別為主；因此，在java.lang的套件下，建立一套和八種基本型別對應的8個參考型別。提供基本型別常見的加減乘除四則運算、比較等操作的替代方案
- 特色在於將各自對應的基本型別是為核心，將以之物件型態「包覆」，也稱為基本型別的「包覆類別 (wrapper class)」

基本型別	包覆類別
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean