認識變數與 Java 基本型別

認識 Java 的基本型別、變數和常數

1. 何謂變數

數學 : X 是一個變數，要藉由「=」左右兩邊相同的前提，計算出 X的值

Java : 使用 Java 進行運算時，因為不同「型別」的變數在電腦中將使用不同大小的記憶體空間，必須先在程式碼中透過「宣告」的方

式明確告知電腦在程式中使用的變數的「型別」

範例 :

```
public class MyMath {
    public static void main(String[] args) {
        int x = 1;
        System.out.println(x+2);
        System.out.println(x*4);
        x = 6;
        System.out.println(x/3);
    }

}

/* Output :
 * 3
 * 4
 * 2
 */
```

- Java 領域裡，型別分兩大類 :

  (1)「基本型別 (Primitive Type)」的變數用於數學基礎運算、邏輯判斷、字元處理等

  (2)「參考型別 (Reference Type)」的變數用於物件導向程式設計裡的物件 (objects) 的儲存

2. Java 的基本型別

類型 (types)	型別	位元組 (bytes)	位元數（bits)	最小值	最大值
整數 Integral	byte	1	8	-128	127
	short	2	16	-2^{15}	$2^{15}-1$
	int	4	32	-2^{31}	$2^{31}-1$
	long	8	64	-2^{63}	$2^{63}-1$
浮點數 Floating point	float	單精確度，32-bit 浮點數		依 IEEE 754 標準	
	double	雙精確度，64-bit 浮點數		依 IEEE 754 標準	
字元 Textual	char	2	16	'\u0000' – '\uffff'	
布林值 Logical	boolean	1	8	true , false	

3. 字面常量

如在程式碼中直接寫下「1、1.0、3.14159、'T'」這樣的數值或文字，未經過變數的宣告與初始化，這類數值或文字就稱之為「字面常量/常數」(Literal constant)。可以是 :

(1) 字元

(2) 字串

(3) 整數 : 預設是 int ，若背後緊接著 l 或 L ，表示 long 型別

(4) 浮點數 : 預設 double，若背後緊接著 f 或 F，表示 float型別

(5) 符號

(6) true / false

- 字面常量和變數一樣都會使用記憶體空間，不同的是，藉由變數的存在，記憶體位址裡的數值或文字還可以反覆使用；而字面常量一旦執行且生成，因為無法反覆使用，就只能等待被回收

範例 :

```
public class LiteralConstantDemo {
    public static void main(String[] args) {
        System.out.println('J');
        System.out.println("Java");
        System.out.println(10);
        System.out.println(10.10);
        System.out.println("+");
        System.out.println(true);

    }

}

/* Output :
 * J
 * Java
```

```
 * 10
 * 10.1
 * +
 * true
 */
```

4. 使用變數的目的

範例 :

```
public class WithoutVariable {

    public static void main(String[] args) {
        // 給一個半徑 (r=5)，計算其圓周 : PI * 2r
       System.out.println("圓周 = " + 3.14159 * 2 * 5);
       // 給一個半徑 (r=5)，計算其面積 : PI * r * r
       System.out.println("面積 = " + 3.14159 * 5 * 5);
       // 半徑變2倍，計算其圓周
       System.out.println("圓周 = " + 3.14159 * 2 *(2 * 5));
    // 半徑變2倍，計算其面積
       System.out.println("面積 = " + 3.14159 *(2 * 5) * (2 * 5));
    }

}

/* Output :
 * 圓周 = 31.4159
 * 面積 = 78.53975
 * 圓周 = 62.8318
 * 面積 = 314.159
*/
```

問題發現 :

(1) 程式裡有很多「字面常量」反覆出現

(2) PI = 3.14159 ，多打幾次很容易打錯

(3) 以後若要改半徑，要改很多地方。漏改、錯改都會造成 bug

範例 :

```
public class WithVariable {

    public static void main(String[] args) {
        final double PI = 3.14159;
        int r = 5 ;
        System.out.println("圓周 = " + PI * 2 * r);
        System.out.println("面積 = " + PI * r * r);
```

```
        r = 10 ;
        System.out.println("圓周 = " + PI * 2 * r);
        System.out.println("面積 = " + PI * r * r);
    }

}
/* Output :
 * 圓周 = 31.4159
 * 面積 = 78.53975
 * 圓周 = 62.8318
 * 面積 = 314.159
 */
```

問題發現：

(1) 字面常量改宣告為變數，放在計算式前面

(2) PI 被宣告成常數，可以直接拿來使用，不用擔心被修改

(3) 日後改半徑 r 時，輕鬆簡單！

範例：

```
public class WithVariable2 {

    public static void main(String[] args) {
        int r  = 5 ;
        showResult(r);
        r  = 10 ;
        showResult(r);
    }

    private static void showResult(int r) {
        final double PI = 3.14159 ;
        System.out.println("圓周 =  " + PI * 2 * r);
        System.out.println("面積 =  " + PI * r * r);
    }
}

/* Output :
 * 圓周 =  31.4159
 * 面積 =  78.53975
 * 圓周 =  62.8318
 * 面積 =  314.159
 */
```

5. 變數與常數

「字面常量」不需經過「宣告」即可在程式中使用，但無法追蹤及繼續使用

「變數 (variable)」因為會隨程式執行而變動，為了追蹤及繼續使用，需要經過「宣告」程序：

(1) 定義一個英文名稱代表該變數，以小寫開頭。若是使用複合字，為了清楚區隔每個單字，則第二個以後的單字都大寫開頭，「駝峰命

名字元」

(2) 定義資料型別

(3) 若變數是類別的欄位，可以再加上修飾詞 (modifiers)

```
int intNum;               //宣告一個整數變數
double dblNum;           // 宣告一個倍精度浮點數變數
float x = 10 , y = 20;   // 同時宣告多個變數屬同一型別，型別不重複
```

- 經過「宣告」後，系統就會配置一塊記憶體空間供其使用
- 秘訣記法：

  對於周遭不在意的人，經常很難記住他 / 她的名子，就像路人甲、路人乙。若在意某個人，需要了解他 / 她的行蹤及改變，就會記得這個人的「名字」，就是「變數名稱」對於變數的意義
- 一旦將數值指定給變數之後，就「不允許再改變」，則可以在宣告變數時使用 final 關鍵字來限定

範例：

```
final double PI = 3.14;    // 使用 final來限定的變數，目的是不希望其他的程式碼來變動它的值
PI = 3.141519;       // compile error !
```

6. 變數的有效範圍

Java 的變數有兩種：

(1) 實例變數 (Instance variable)，亦即類別屬性或欄位 (field)

- 有效範圍在整個實例 (instance)內
- 使用前若未給值 (未初始化)，依其型別，Java 將給不同預設 (default) 值

型別分類	基本型別	預設值
整數	byte、short、int、long	0
浮點數	float、double	0.0
字元	char	空字元，為 ' ' 或 '\u0000'
邏輯	boolean	false

(2) 區域變數 (Local variable)

- 有效範圍在宣告的方法或特定程式碼區塊 { } 內。若變數名稱和外圍變數的名稱相同，將由區域變數覆蓋實例變數
- 宣告的型態前不能再有修飾詞，如 public 等
- 使用前若未給值 (初始化)，將編譯失敗

範例：

```
public class Shirt {
    public int size ;          //宣告實例變數 int size。因為宣告時沒有一併初始化(給值)，將由java給預設值 0
    public double price = 100.5 ;      //宣告實例變數 double price = 100.5;
    public void display() {
        int size = 5 ;                // 宣告區域變數 int size = 5 ;
        System.out.println(size);     // 因為區域變數 size和實例變數名稱相同，因此覆蓋外圍的實例變數，故印出數值5
        System.out.println(price); // 印出實例變數 price 的數值100.5

        /*if (5 >2) {
            int size = 9; // compile error。因為在方法的程式區塊範圍內， Java不允許再有同名變數
            System.out.println(size);
        }*/
    }

    public static void main(String[] args) {
        new Shirt().display();

    }
}

/* Output :
 * 5
 * 100.5
 */
```

- 秘訣：

  Java 的變數在「被使用前」，無論實例或區域變數，都必須有值。差別在於：

  a. 實例變數：若沒先給值，Java會給預設值

  b. 區域變數：若沒先給值，就完蛋了 !!!!!!!

7. 字元型別

範例：

```
public class CharTest {
    public static void main(String[] args) {
        char c1 ='A';     //是char的宣告型態，故印出字元，不會是數字
        System.out.println(c1);
        char c2 = 65;     // ASCII 英文字母大寫 A，對應到 65 。是char的宣告型態，故印出字元，不會是數字
```

```
        System.out.println(c2);
        int i = 65 ;
        System.out.println(i);
    }
}

/* Output :
 * A
 * A
 * 65
 */
```

- 對 Java 而言，「字元型別 (Textual Type)」裡的每個「字元 (char)」都是一種「圖形」。Java在儲存char時，並非直接儲存圖形 'A'，而是轉成位元碼儲存

8. 使用二進位的字面常量顯示整數

若數值以「0b/0B」開頭，表示將以二進位的書寫方式表現數字

範例：

```
public class BinaryLiteralsDemo {

    public static void main(String[] args) {
        byte b1 = 2;
        byte b2 = 0b10;
        // = 2*(1) + 1 *(0) = 2
        byte b3 = 0B101011;
        // = 32*(1) + 16*(0) + 8*(1) + 4*(0) + 2*(1) + 1*(1) = 43
        System.out.println(b1 + "," + b2 + "," + b3);
    }

}

/* Output :
 * 2,2,43
 */
```

9. 使用底線提高數字常量的可讀性

以「_」區隔數字，增加數字常量 (numeric literals)的可讀性。

範例：

```
public class NumericLiteralsDemo {

    public static void main(String[] args) {
        int i1 = 1234567;
        int i2 = 1_234_567;
        System.out.println(i1 == i2);
        double d1 = 1234567.1234567;
```

```
        double d2 = 1_234_567.123_456_7 ;
        System.out.println(d1 == d2);

    }

}

/* Output :
 * true
 * true
 */
```

使用運算子

1. 常用的運算子

- 算術運算子

運算子	運算子意義	int a = 9, b = 4	運算結果
+	加法	a + b	13
-	減法	a - b	5
*	乘法	a * b	36
/	除法	a / b	2
%	取餘數	a % b	1

- 簡潔運算子

運算子	原式	簡潔運算式
+=	a = a + b	a += b
-=	a = a - b	a -= b
*=	a = a * b	a * = b
/=	a = a / b	a /= b
%=	a = a % b	a %= b

- 遞增 / 遞減運算子

  (1) i ++ : 先執行整個敘述後，再將 i 的值加 1

  (2) ++ i : 先將 i 的值加 1 ，再執行整個敘述

  (3)「遞增 / 遞減」運算子「在前面」，「遞增 / 遞減」運算子「先處理」

「遞增 / 遞減」運算子「在後面」，「遞增 / 遞減」運算子「後處理」

運算子	意義	int i = 3; int a = 0;	結果	
			i	a
++	變數值加 1	i ++	4	
--	變數值減 1	i --	2	
		a = i ++	4	3
		a = ++ i	4	4
		a = i --	2	3
		a = -- i	2	2

2. 運算子的處理順序

處理順序 :

(1) 括號 () 內先處理

(2) 遞增 / 遞減運算子在字首

(3) 算術運算子 (先乘除，後加減。 注意！字串相連使用的 + 號，也在這範圍內)

(4) 關係運算子 (< 、> 、<= 、>= 、== 、!=)

(5) 條件運算子 (&& 、|| 、!)

(6) 三元運算子 (? :)

(7) 指派運算子 (= 、+= 、-= 、*= 、/= 、%=)

(8) 遞增 / 遞減運算子在字尾

範例 :

```
public class OperatorsTest {
    public static void main(String[] args) {
        int count = 20 ;
        int a , b , c , d ;
        a = count++ ;
        b = count ;
        c = ++count ;
        d = count + 1;
        System.out.println(a+b+c+d);
        System.out.println("Result=" + a + b + c + d);
        System.out.println(a + b + "Result=" + c + d);
        System.out.println("Result=" + a + (b + c) + d);
    }

}
```

```
/* Output :
 * 86
 * Result=20212223
 * 41Result=2223
 * Result=204323
 */
```

使用升等和轉型

使用運算子不難，甚至小時候在數學的時候都可能已經滾瓜爛熟。加上程式語言的型別後，就要特別注意「型別轉型」的問題，否則常常會有跌破眼鏡的結果。

範例 :

```
public class Test {
    public static void main(String[] args) {
        int number1 = 10 ;
        System.out.println(number1 / 3);

        double number2 = 10 ;
        System.out.println(number2 / 3);

    }

}

/* Output :
 * 3
 * 3.3333333333333335
 */
```

1. 型別的升等

當 Java 遇到運算式有變數型別不對等或不一致的情況時，會做型別的自動升等 (Automatic Promotion)。主要為 :

(a) 運算式內變數型別不一致時，小型別的將自動提升和大型別一致，以正確保留運算結果

(b) 較小型別指定給較大型別時 (ex : 以大型別宣告)，小型別的數值將自動提升和大型別的宣告型別一致，以滿足宣告要求

(c) 將整數型態 (byte / short / int /long)指定給浮點數型態 (float / double)時，整數型態會自動提升為浮點數型態，以保留小數點後的位

數

範例 1 :

```
public class PromotionTest {
```

```
    public static void main(String[] args) {
        int n1 = 10;
        System.out.println(n1 /3);       //因為字面常量3被當成int，變數n1也是int，故結果
就是int，所以印出數值3
        double n2 = n1;                  //變數 n1是int，變數n2是double，將較小型別指定
給較大型別時，n1將被提升和n2一致                                          //皆為
double，以滿足宣告型別
        System.out.println(n2 /3);      // 因為字面常量3是int，變數n2是double，運算式內
變數型別出現不一致的情況；所以字
                                        //面常量3會被提升至double使其和n2一致，結果就會是
double

    }

}

/* Output :
 * 3
 * 3.3333333333333335
 */
```

範例 2：

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

        /* 將小型別指定給大型別 */
        int y1 = 10 ;
        long y2 = y1 ;
        float z1 = 10.0f ;
        double z2 = z1 ;

        /* 將整數指定給浮點數 */
        float e = 2 ;
        double f = 2 ;
        float g = 2L ;
        double h = 2L ;

        /* 無法將浮點數型態指定給整數型態!!!浮點數將遺失!!!! */
        int a = 2.34f ;      //無法編譯
        long b = 2.34f ;     //無法編譯
        int c = 2.34f ;      //無法編譯
        long d = 2.34f ;     //無法編譯

    }
}

/* 注意 !!!!! 型態長度大小 : double > float > long > int > short > byte */
```

2. 型別的轉型

Java 轉型 (casting)顧名思義是將原先型別轉換成新的型別。語法是變數前加一個「()」，裡面放入要轉型的目標：

語法：

(target_type) value

- 「轉型」通常用於將「大」型別轉成「小」型別。就基本型別而言，轉小型別可以減少記憶體的使用。但必須注意轉型後的數值是否和先前相同
- Java 在使用指派運算子「=」時將因運算式內的型別不一致而發動自動升等，讓小型別轉大型別。有需要時也可以自己利用轉型語法發動。

範例 1：(小型別轉為大型別)

```
int number = 10 ;
System.out.println( (double)number / 3);
```

範例 2：

```
public class CastingTest {
    public static void main(String[] args) {
        int i1 = 53 ;
        int i2 = 47 ;
        byte b3 ;
        b3 = i1 + i2 ; // compile error ! 。因為i1+i2將得到int型別，長度為4bytes，無法指定給byte型別(長度為1byte)
        b3 = (byte)(i1 + i2);
        System.out.println(b3);
    }
}
```

- 秘訣：

  變數型別如杯子，驗數值如杯裡的水，編譯器檢查時只管杯子的大小，不允許將大杯子的水倒入小杯子。但撰寫程式碼的我們，卻可以知道大杯子裡的水雖然會一直改變 (因為是變數)，但其實只有一些些，因此倒進小杯子後不會有滿出 (overflow)的問題，所以可以大膽使用「轉型」

3. 暫存空間對算術運算的影響

Java 在算術運算時會先將指派運算子「右側的運算過程及結果」先放在「暫存空間」，才會丟給指派運算子「左側的變數」

範例 1：

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

```
        int x = 3 * 4 ;
        System.out.println(x); // 3*4使用的暫存空間為int型別的大小(4 bytes)，宣告型別
int承接計算結果，過程正確，輸出結                                        //  果:12

        int a  = 55555 * 66666 ;
        System.out.println(a) ; // 55555 * 66666使用的暫存空間為int型別的大小(4
bytes)，但計算結果=3703629630，長度
                                  // 超出 4 bytes，故發生溢位(overflow)，所以輸出結果
為:-591337666

        long b = 55555 * 66666 ;
        System.out.println(b); //雖然宣告改為long，但問題是 55555 * 666666的「計算過
程」就已經發生溢位，因此只是將溢位的
                                 // 結果指定給 long b，輸出結果依然是:-591337666

        long c = (long)(55555 * 66666) ;
        System.out.println(c);  // 將溢位的結果轉型成long，不影響輸出結果

        long d = ((long)55555) * 66666 ;
        System.out.println(d);  //因為計算前先將其中一個int轉型為long，所以暫存空間被加大
為long型別的大小(8 bytes)，因
                                  //此計算過程不會發生溢位，故得到正確輸出結果:3703629630

    }

}

/* Output :
 * 12
 * -591337666
 * -591337666
 * -591337666
 * 3703629630
 */
```

範例 2：

```
public class TempSpaceDemo2 {
    public static void main(String[] args) {

        short a, b;
        a = 1 ;
        b = 2 ;

        short c ;

        c = a + b ;  //error ! 。變數a和b雖然都是short，但進行計算時仍然使用長度為int型別
長度的暫存空間，故無法將計算結果
                     //指定給short的變數c。編譯失敗訊息為:Type mismatch: cannot
convert from int to short

        c = (short)(a + b); //使用轉型將int的計算結果縮小為short

        int d ;     //或將結果改用宣告為int的變數d承接
```

```
        d = a + b ;

    }
}
```