

# Operator

運算子

Java Fundamental



# Java 運算子 (Operators)

- ◆ Java 「運算式」是由「運算元」和「運算子」組成。

Expression = Operands + Operators

運算式 = 運算元 + 運算子

- ◆ 運算式的範例：

- `int c = a + b - 1`
- `int d = a >= b`
- `boolean e = a > b && a > 1`

- ◆ 變數 `a`、`b`、`1` 是運算元；「`+`」、「`-`」、「`>=`」、「`>`」和「`&&`」為運算子。

運算子  
3 \* 4  
運算元

# Java 運算子 (Operators)

- ◆ 運算子可對一個、兩個或三個運算元執行動作。
- ◆ 依照所需的運算元數目，可分為：
  - 一元運算子(unary operator)，例：`a++`、`--a`
  - 二元運算子(binary operator)，例：`a = 2`
  - 三元運算子(ternary operator)，例：`(x>0) ? x-1 : (x+1)`
- ◆ 依照運算子擺放位置，可分為
  - 前置(prefix)運算子：運算子擺放在運算元之前，例：`++i`
  - 後置(postfix)運算子：運算子擺放在運算元之後，例：`i++`
  - 中置(infix)運算子：運算子擺放在中間，例：`a + b`

運算子  
3 \* 4  
運算元

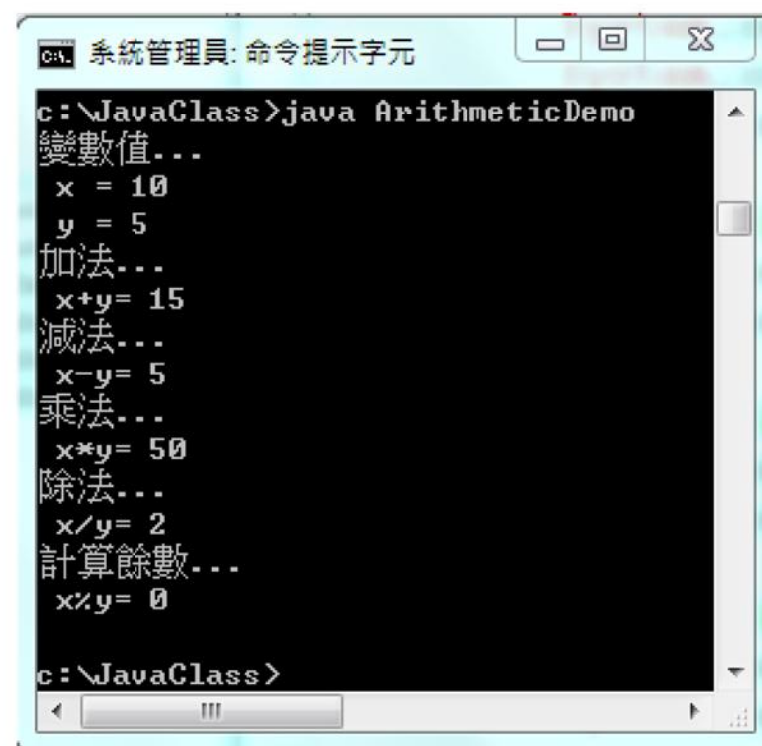
# 算數運算子 (Arithmetic Operator)

運算子	說明	範例	類別
+	正號	<b>+</b> 2 → 取得正數 2	單元
	加法	2 <b>+</b> 1 = 3	雙元
-	負號	<b>-</b> 2 → 取得負數 2	單元
	減法	2 <b>-</b> 1 = 1	雙元
*	乘法	2 * 3 = 6	雙元
/	除法	4 / 2 = 2	雙元
%	餘數	3 % 2 = 1，求得 3 除以 2 的餘數	雙元
++	遞增	前序遞增：int A = 0；A = <b>++</b> A； 表示 A 先加上 1 之後再指派給A所以A=1。	單元
		後序遞增：int A = 0；A = A <b>++</b> ； 表示A(此時的A=0) 會先指派給 A 之後才會執行 <b>++</b> 的動作， <b>但A仍然是0</b> 。	
--	遞減	前序遞減：int A = 0；A = <b>--</b> A； 表示A先減1之後再指派給A所以 A = - 1	單元
		後序遞減：int A = 0；A = A <b>--</b> ； 表示 A(此時的A=0)會先 指派給 A 之後才會執行 <b>--</b> 的動作但 A 仍然是 0。	



# 算數運算子(Arithmetic Operator)

```
01 public class ArithmeticDemo {
02     public static void main(String[] args) {
03         //宣告變數與值
04         int x = 10;
05         int y = 5;
06         System.out.println("變數值...");
07         System.out.println(" x = " + x);
08         System.out.println(" y = " + y);
09         //加法示範
10         System.out.println("加法...");
11         System.out.println(" x+y= " + (x+y));
12         //減法示範
13         System.out.println("減法...");
14         System.out.println(" x-y= " + (x-y));
15         //乘法示範
16         System.out.println("乘法...");
17         System.out.println(" x*y= " + (x*y));
18         //除法示範
19         System.out.println("除法...");
20         System.out.println(" x/y= " + (x/y));
21         //模數運算
22         System.out.println("計算餘數...");
23         System.out.println(" x%y= " + (x%y));
24     }
25 }
```



The screenshot shows a Windows Command Prompt window titled "系統管理員: 命令提示字元". The command prompt shows the execution of the Java program `java ArithmeticDemo` from the directory `c:\JavaClass`. The output of the program is displayed as follows:

```
c:\JavaClass>java ArithmeticDemo
變數值...
 x = 10
 y = 5
加法...
 x+y= 15
減法...
 x-y= 5
乘法...
 x*y= 50
除法...
 x/y= 2
計算餘數...
 x%y= 0
c:\JavaClass>
```

# 算數運算子 後置運算

◆  $X = X + 1;$                        $\Rightarrow$  遞增運算                       $X++$  ,  $++X$

◆  $X = X - 1;$                        $\Rightarrow$  遞減運算                       $X--$  ,  $--X$

◆ 後置遞增、後置遞減

- $\text{int var} = \text{age}++;$



$\text{int var} = \text{age};$   
 $\text{age} = \text{age} + 1;$

```
01 public class UnaryOperatorDemo {
02     public static void main(String[] args) {
03         int age = 10;
04         int var = age++;
05         System.out.println("age="+age);
06         System.out.println("var="+var);
07     }
08 }
```

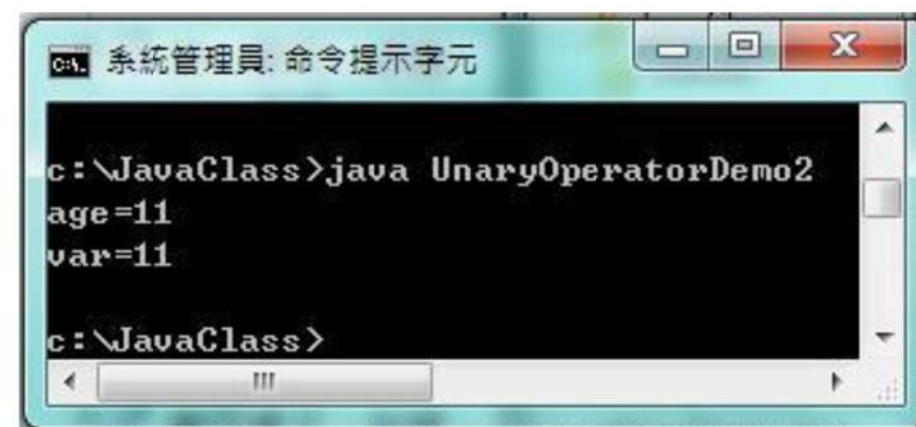
```
C:\JavaClass>java UnaryOperatorDemo
age=11
var=10
C:\JavaClass>
```

# 算數運算子 前置運算

## ◆ 前置遞增、前置遞減

- `int var = ++age;`       $\Rightarrow$       `age = age + 1`  
   `int var = age;`

```
01 public class UnaryOperatorDemo2 {  
02     public static void main(String[] args) {  
03         int age = 10;  
04         int var = ++age;  
05         System.out.println("age="+age);  
06         System.out.println("var="+var);  
07     }  
08 }
```



```
c:\JavaClass>java UnaryOperatorDemo2  
age=11  
var=11  
  
c:\JavaClass>
```



# 算數運算子 遞增遞減運算

```
int x = 17;  
  
System.out.println("Before Prefix X = " + ++x);  
System.out.println("After Prefix X = " + x);  
  
x = 29;  
  
System.out.println("Before Postfix X = " + x--);  
System.out.println("After Postfix X = " + x);
```

```
Before Prefix X = 18  
After Prefix X = 18  
Before Postfix X = 29  
After Postfix X = 28
```



# 算術運算與自動轉型規則

---

## ◆ 算術運算與自動轉型

1. **Java** 會將運算式中所有的運算元自動轉型成最大的運算元型別來運算。
2. 若所有的運算元都小於 `int`, 程式會自動轉型成 `int` 來運算。
  - 1) `short a = 1 ;`
  - 2) `short b = 2 ;`
  - 3) `short c = (short) (a+b);`

# 文字串接 (String Concatenation)

## ◆ 文字串接

➤ 「+」中只要有一個運算元是字串，另一個運算元也會被轉成字串。

1. `System.out.println(5+6+"str");`      `//11str`

2. `System.out.println("str"+5+6);`      `//str56`

```
public class Concatenate{  
    public static void main(String[] args){  
        String str1 = "Java ";  
        String str2 = "Operators";  
        int i = 1;  
        double d = 1.2;  
        String str3 = str1 + i + d;  
        System.out.println(str1.concat(str2));  
        System.out.println(str3);  
    }  
}
```

-----輸出-----  
Java Operators  
Java 11.2

會將 **i** 與 **d** 數字轉成字串後再串接

將 **str1** 與 **str2** 兩個字串串接起來

# 算數運算子 (Arithmetic Operator)

---

## ◆ 整數 / 整數 (取商數)

- $10 / 3 = 3 \cdots 1$  餘數捨棄
- $10 / 0 = \text{throws ArithmeticException}$  分母不可為0

## ◆ 運算元 (除數或被除數) 中至少有一為浮點數

- $10.0 / 3 = 3.33333334$  不丟棄餘數
- $10.0 / 0.0 = \text{infinity}$
- $-10.0 / 0.0 = \text{-infinity}$
- $0.0 / 0.0 = \text{NaN}$

分母可為0



# 算數運算子 (Arithmetic Operator)

---

## ◆ 整數 % 整數(取餘數)

- $10 \% 0 = \text{throws ArithmeticException}$  分母不可為0
- $11 \% 3 = 2$
- $11 \% (-3) = 2$
- $(-11) \% 3 = -2$
- $(-11) \% (-3) = -2$

## ◆ 若有一個以上的運算元為浮點數

- $10.0 \% 0.0 = \text{NaN}$
- $0.0 \% 0.0 = \text{NaN}$
- $(5/0.0)\%2.0 = \text{NaN}$  ( $\infty \% 2.0$ )
- $8.0 \% 3 = 2.0$
- $6.5 \% 3.2 = 0.1$  (商數須為整數)
- $2.0\%(5/0.0) = 2.0$  ( $2.0\%\infty$ )

# 關係運算子 (Relational Operator)

關係運算子傳回值為布林值 (true或false)

運算子	說明	運算元	範例	運算子類別
<b>==</b>	等於	布林 數字 字元 物件	$a == b$	雙元
<b>!=</b>	不等於		$a != b$	雙元
<b>&gt;</b>	大於	數字,字元	$a > b$	雙元
<b>&lt;</b>	小於	數字,字元	$a < b$	雙元
<b>&gt;=</b>	大於等於	數字,字元	$a >= b$	雙元
<b>&lt;=</b>	小於等於	數字,字元	$a <= b$	雙元

# 關係運算子 (Relational Operator)

- ◆ 關係運算子會得到true或false的答案。
- ◆ 浮點數不是精準數，建議不要將浮點數用在「==」上。

```
1. public class Compare{
2.     public static void main(String[] args){
3.         int x = 50, y = 50;
4.         boolean b1 = x > y;
5.         boolean b2 = x == y;
6.         boolean b3 = x != y;
7.         System.out.println(b1);
8.         System.out.println(b2);
9.         System.out.println(b3);
10.    }
11. }
```

比較 x 與 y 是否相等，是的話  
回傳 **true**；否則回傳 **false**

x 與 y 的值都是 **50**，所以回  
傳 **false**

-----輸出-----

false

true

false



# 邏輯運算子 (Logical operator)

◆ 運算元為布林值(true或false)

◆ 傳回值為布林值

運算子	說明	範例	運算子類別
&	AND	a & b	雙元
	OR	a   b	雙元
^	XOR	a ^ b	雙元
!	NOT	! a	單元
(捷徑)Short-circuit Logical Operator			
&&	AND	a && b	雙元
	OR	a    b	雙元

# 邏輯運算子 (Logical operator)

```
01 public class ConditionalDemo {
02     public static void main(String[] args) {
03         int i = 2;
04         int j = 3;
05         int k = 4;
06         System.out.println("變數值...");
07         System.out.println(" i = " + i);
08         System.out.println(" j = " + j);
09         System.out.println(" k = " + k);
10         //&&運算
11         System.out.println("且...");
12         System.out.println(" i<j 且 j<k "+((i<j)&&(j<k)));
13         //||運算
14         System.out.println("或...");
15         System.out.println(" i<j 或 k>j "+((i<j)|| (k>j)));
16         //&&運算
17         System.out.println("非...");
18         System.out.println(" i<j 的非 "+(! (i<j)));
19     }
20 }
```



```
c:\JavaClass>java ConditionalDemo
變數值...
i = 2
j = 3
k = 4
且...
i<j 且 j<k true
或...
i<j 或 k>j true
非...
i<j 的非 false
c:\JavaClass>
```

# 邏輯運算子 (Logical operator)

## ◆ XOR (^) :

- XOR與OR運算的結果大致相同。
- 只有當所有條件式都是True時，XOR結果是False。

## ◆ NOT (!) :

- True碰到NOT符號會得到False；False碰到NOT符號會變成True。

```
1. class XOR_NOT{
2.     public static void main(String[] args){
3.         boolean furniture=true, decoration=true;
4.
5.         System.out.println(furniture ^ decoration);
6.         System.out.println(!(furniture ^ decoration));
7.     }
8. }
```

true ^ true 得到結果為 false

furniture ^ decoration 結果為 false，  
前面加上「!」運算符號後會得到 true

-----輸出-----

false

true



# 捷徑邏輯運算子 (Short-circuit Logical Operator)

## ◆ 一般邏輯運算子 $\&$ , $|$ , $\wedge$ , $!$

AND	true	false	OR	true	false	XOR	true	false	NOT	
true	true	false	true	true	true	true	false	true	true	false
false	false	false	false	true	false	false	true	false	false	true

## ◆ 捷徑邏輯運算子(short-circuit) $\&\&$ , $||$

- 當左式已經足以判斷整個運算的結果時，右式就不必做了。
- 若左式不足以判斷整個運算的結果時，右式仍然要做。

$\&\&$	X
true	X
false	false

$  $	X
true	true
false	X

# 捷徑邏輯運算子 (Short - Circuit Logical Operator)

◆ `a=10; b=5`

`(a++ > 10) & (b-- < 5)`  $\Rightarrow$  **`a=11; b=4`**

`(a++ > 10) && (b-- < 5)`  $\Rightarrow$  **`a=11; b=5`**

◆ `if ((10 / i) > 2) {...} int i = 0;`  $\Rightarrow$  **throws `ArithmeticException`**

◆ `if( i == 0 || (10/i) > 2) {...}`  $\Rightarrow$  **true**

◆ `if( i != 0 && (10/i) > 2) {...}`  $\Rightarrow$  **false**

# 指定運算子(Assignment Operator)

運算子	說明	範例	類別
=	基本指定	$a = b$	單元
+=	加法指定	$a += b \rightarrow a = a + b$	雙元
-=	減法指定	$a -= b \rightarrow a = a - b$	雙元
*=	乘法指定	$a *= b \rightarrow a = a * b$	雙元
/=	除法指定	$a /= b \rightarrow a = a / b$	雙元
%=	餘數指定	$a \% = b \rightarrow a = a \% b$	雙元
&=	AND 指定	$a \& = b \rightarrow a = a \& b$	雙元
=	OR 指定	$a  = b \rightarrow a = a   b$	雙元
^=	XOR 指定	$a \wedge = b \rightarrow a = a \wedge b$	雙元
>>=	算數右移指定	$a >> = b \rightarrow a = a >> b$	雙元
<<=	算數左移指定	$a << = b \rightarrow a = a << b$	雙元
>>>=	邏輯右移指定	$a >>> = b \rightarrow a = a >>> b$	雙元



# 運算子的優先順序

## ◆ 運算子優先權

高



低

運算子分類	運算子
分隔運算子	() [] {} , .
一元運算子(單一運算元)	++ -- + (正號) - (負號) ~ !
建造 / 轉型運算子	new (type)expr
算術運算子	* / % + -
位移運算子	<< >> >>>
關係運算子	< <= > >=
型別運算子	instanceof
相等運算子	== !=
位元邏輯運算子	& ^
邏輯運算子	&&
條件運算子	? :
指定運算子	= += -= *= /= %= &= ^=  = >>= <<= >>>=

Q & A