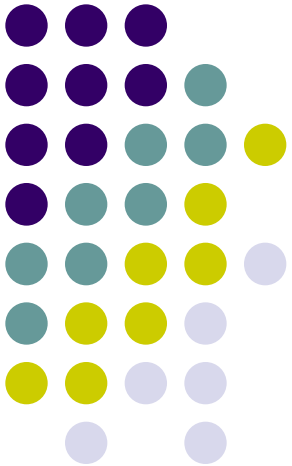
第四章
運算子、運算式與敘述

認識運算式與運算子

學習各種常用的運算子

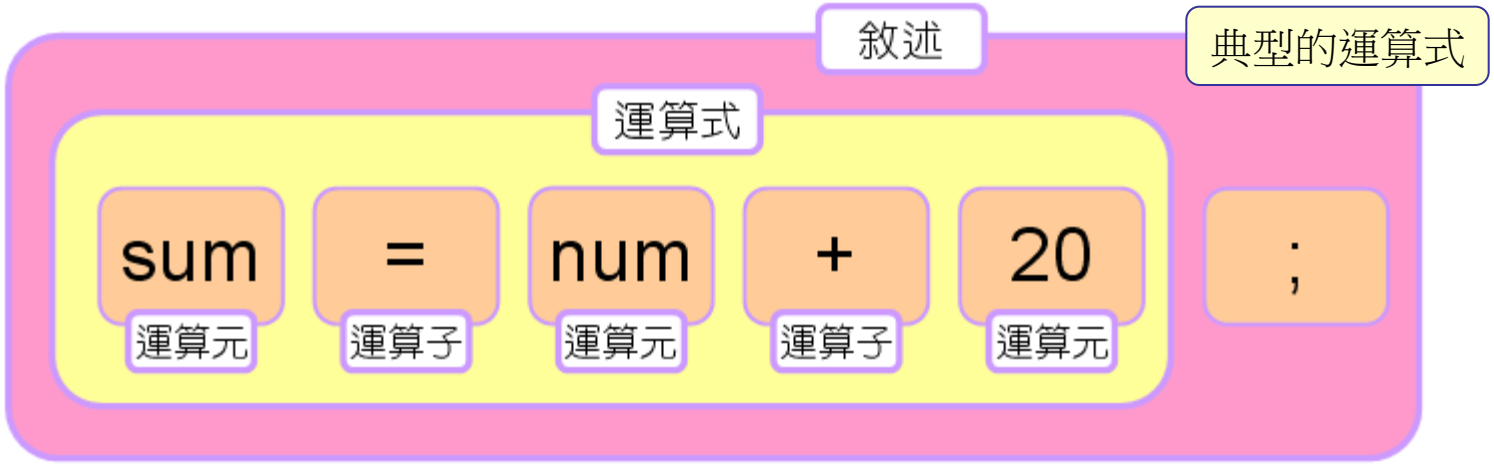
認識運算子的優先順序

學習如何進行運算式之資料型態的轉換



認識運算式

- 運算式由運算元（operand）與運算子（operator）組成
- 運算元可以是變數或是常數
- 運算子就是數學上的運算符號
  - 如「+」、「-」、「*」、「/」等



算術運算子(1/2)

- 算術運算子在數學上經常會使用到
- 算術運算子的成員：

• 算術運算子

運算子	代表意義	範例	說明
+	加法	a+b	計算 a 與 b 相加
-	減法	a-b	計算 a 與 b 相減
*	乘法	a*b	計算 a 與 b 相乘
/	除法	a/b	計算 a 與 b 相除
%	取餘數	a%b	計算 a 除以 b 的餘數

```
b=c*3;
a=a*a;
17*5;
```
*

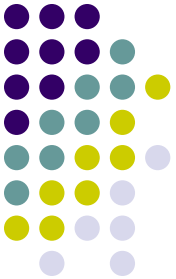
```
age=age%5;
c=a%b;
45%7;
```
%

```
6+2;
b=a+15;
sum=a+b+c;
```
+

```
age=age-1;
c=a-b;
54-12;
```
-

```
b=a/6;
d=c/d;
3/8;
```
/

要注意資料型態的變化



算術運算子(2/2)

```
01  // Ch4_1, 算術運算子的使用
02  public class Ch4_1{
03     public static void main(String[] args){
04        int a=9, b=5;
05
06        System.out.printf("%d + %d=%d\n",a,b,a+b);     // 相加
07        System.out.printf("%d - %d=%d\n",a,b,a-b);     // 相減
08        System.out.printf("%d * %d=%d\n",a,b,a*b);     // 相乘
09        System.out.printf("%d / %d=%d\n",a,b,a/b);     // 相除
10        System.out.printf("%d %% %d=%d\n",a,b,a%b);    // 取餘數
11     }
12  }
```

執行結果：

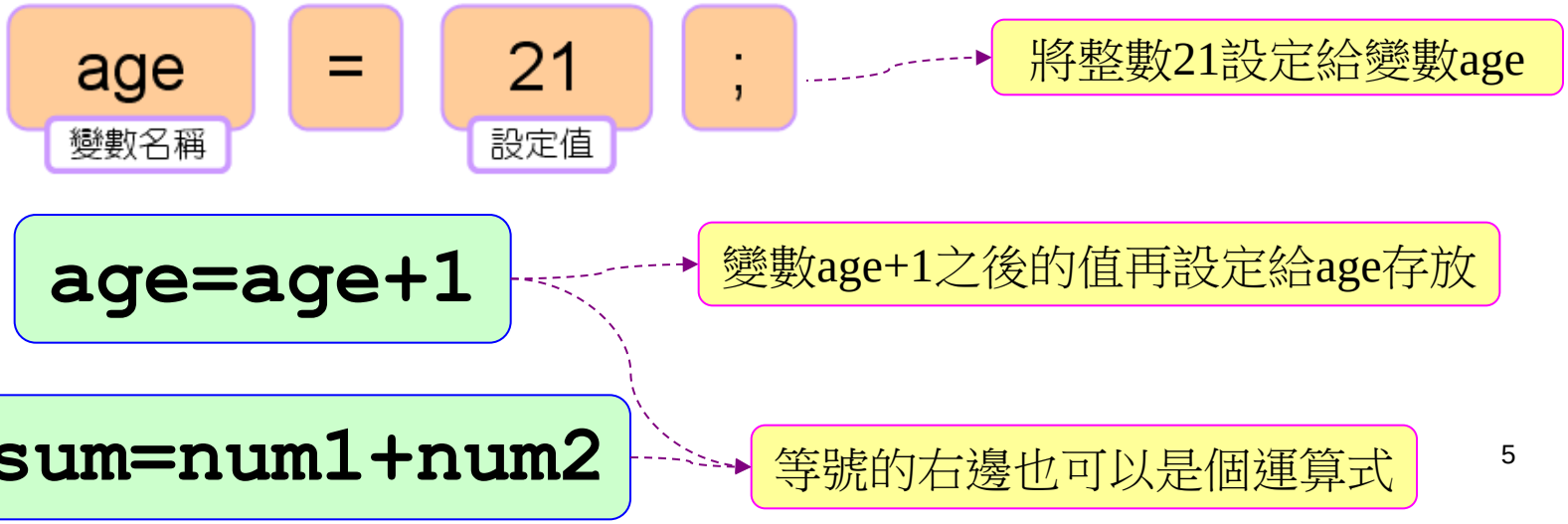
```
9 + 5=14
9 - 5=4
9 * 5=45
9 / 5=1
9 % 5=4
```

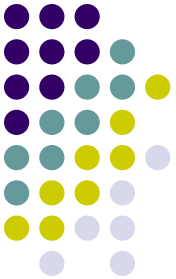
設定運算子(1/3)

- 將變數設值可使用設定運算子

設定運算子	意義
=	設定

- 等號（=）並不是「等於」，而是「設定」



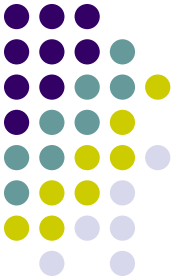


設定運算子(2/3)

- 複合設定運算子可讓程式碼簡潔

• 複合設定運算子

運算子	代表意義	範例	說明
=	直接設定	a=b	將 b 的值設定給 a 存放
+=	以和設定	a+=b	將 a+b 的值存放到 a 中，等同於 a=a+b
-=	以差設定	a-=b	將 a-b 的值存放到 a 中，等同於 a=a-b
*=	以積設定	a*=b	將 a*b 的值存放到 a 中，等同於 a=a*b
/=	以商設定	a/=b	將 a/b 的值存放到 a 中，等同於 a=a/b
%=	以取餘數設定	a%=b	將 a%b 的值存放到 a 中，等同於 a=a%b



設定運算子 (3/3)

- 下面的範例示範了複合設定運算子的使用：

```
01  // Ch4_2, 複合設定運算子的使用
02  public class Ch4_2{
03     public static void main(String[] args){
04        int a=5, b=3;
05
06        a+=4;   // 相當於 a=a+4
07        b-=1;   // 相當於 b=b-1
08        System.out.printf("a=%d\n",a);
09        System.out.printf("b=%d\n",b);
10     }
11  }
```

執行結果：

```
a=9
b=2
```

遞增與遞減運算子(1/3)

- 遞增與遞減運算子的成員：

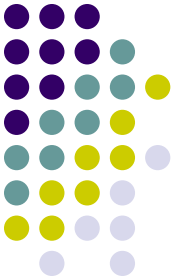
• 遞增與遞減運算子

運算子	代表意義	範例	說明
++	遞增運算	++a	遞增運算在前。先將 a 加 1，再傳回 a 的值
		a++	遞增運算在後。先傳回 a 的值，a 再加 1
--	遞減運算	--a	遞減運算在前。先將 a 減 1，再傳回 a 的值
		a--	遞增運算在後。先傳回 a 的值，a 再減 1

- 想讓變數a加上1，其敘述如下

```
a=a+1;          // a 加 1 後再設定給 a 存放
a+=1;           // 利用複合運算子將 a 加 1
a++;            // 利用++運算子
++a;            // 利用++運算子
```

這4種寫法
都可以



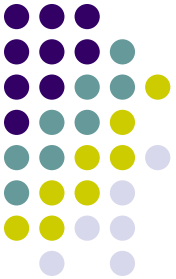
遞增與遞減運算子(2/3)

- 下面的程式是使用遞增運算子的範例：

```
01  // Ch4_3, 遞增運算子「++」
02  public class Ch4_3{
03      public static void main(String[] args){
04          int a=5,b=5;
05
06          System.out.printf("++a 的傳回值: %d\n",++a);  //遞增運算子在前
07          System.out.printf("執行完++a 之後，a= %d\n",a);
08          System.out.printf("b++的傳回值: %d\n",b++);  //遞增運算子在後
09          System.out.printf("執行完 b++之後，b= %d\n",b);
10      }
11  }
```

執行結果：

```
++a 的傳回值: 6
執行完++a 之後, a= 6
b++的傳回值: 5
執行完 b++之後, b= 6
```



遞增與遞減運算子(3/3)

- 下面的程式比較a++與++b的不同

```
01  int a=5,b;
02  b=--a-3;
03  System.out.printf("a=%d, b=%d\n",a,b);
```

就等於
```
a=a-1;
b=a-3;
```

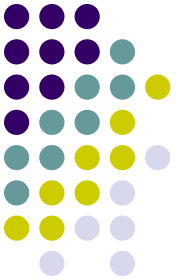
```
01  int a=5,b;
02  b=(a--)-3;    // 刻意加了括號，也可以寫成 b=a---3，但不好閱讀
03  System.out.printf("a=%d, b=%d\n",a,b);
```

就等於
```
b=a-3;
a=a-1;
```

```
01  int a=5,b=2,c;
02  b+= a++;          // 同時使用了+=和++運算子
03  c=(a++)*(++a);    // ++運算子用了兩次
04  System.out.printf("a= %d, b=%d, c=%d\n",a,b,c);
```

就等於
```
b+=a;
a++;
```

過多遞增/遞減運算子
反而造成閱讀困難

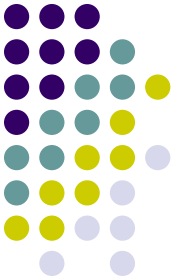


關係運算子 (1/2)

- 關係運算子可以判斷條件式是否成立

  - 算術運算子

運算子	代表意義	範例	說明
>	大於	5>8	不成立，回應 false
<	小於	5<8	成立，回應 true
>=	大於等於	5>=5	成立，回應 true
<=	小於等於	6<=8	成立，回應 true
==	等於	7==8	不成立，回應 false
!=	不等於	7!=8	成立，回應 true



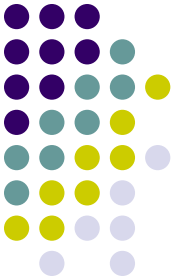
關係運算子 (2/2)

- 關係運算子的使用範例：

```
01  // Ch4_4, 關係運算子
02  public class Ch4_4{
03      public static void main(String[] args){
04          System.out.printf("5>=4: %b\n",5>=4);    // 大於等於運算子
05          System.out.printf("6<=6: %b\n",6<=6);    // 小於等於運算子
06          System.out.printf("8!=7: %b\n",8!=7);    // 不等於運算子
07          System.out.printf("5==4: %b\n",5==4);    // 等於運算子
08      }
09  }
```

執行結果：

```
5>=4: true
6<=6: true
8!=7: true
5==4: false
```



邏輯運算子 (1/2)

- 邏輯運算子與真值表：

• 邏輯運算子

運算子	代表意義	範例	說明
&&	AND，且	x>5 && x<10	&& 兩邊都是 true 則傳回 true
||	OR，或	x<4 || x>8	|| 兩邊任一是 true 則傳回 true
!	NOT，取反運算	!true	!true 傳回 false，！false 傳回 true

邏輯運算子的成員

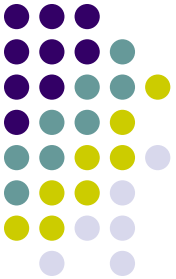
邏輯運算子	意義
&&	AND，且
||	OR，或

AND與OR真值表

AND	T	F
T	T	F
F	F	F

OR	T	F
T	T	T
F	T	F

```
3>0 && 5>3       // 3>0 和 5>3 皆為 true，因此運算結果為 true
5>8 || 7!=0      // 7!=0 為 true，所以運算結果為 true
!(3>5)           // 3>5 為 false，取反之後得到 true
```



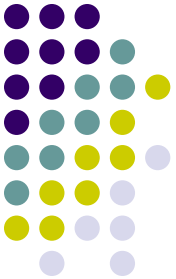
邏輯運算子(2/2)

- 邏輯運算子的範例 ：

```
01  // Ch4_5, 邏輯運算子
02  public class Ch4_5{
03      public static void main(String[] args){
04         boolean a=true, b=false;
05         System.out.printf("%b || %b = %b\n",a, b, a||b); // 邏輯運算子 OR
06         System.out.printf("%b && %b = %b\n",a, b, a&&b); // 邏輯運算子 AND
07         System.out.printf("!%b == %b\n",a,!a);   // 邏輯運算子 NOT
08      }
09   }
```

執行結果：

```
true || false = true
true && false = false
!true == false
```

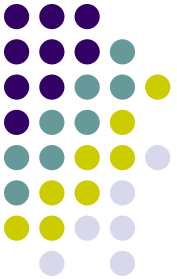


位元運算子(1/2)

- 位元運算子以a=7，b=13做說明 ：

・位元運算子 （假設 a=7 (0b0111), b=13 (0b1101)）

運算子	代表意義	範例	說明
&	位元 AND	a & b = 5	0111 & 1101 = 0101 = 5
|	位元 OR	a | b = 15	0111 | 1101 = 1111 = 15
~	位元 NOT	~a = -8	~00111=11000=-8， 取1的補數
^	位元 XOR	a ^ b = 10	0111 ^ 1101 = 1010 = 10
>>	位元右移	a >> 2= 1	0111 >> 2 = 0001 = 1
<<	位元左移	a << 2 = 28	0111 << 2 = 11100 = 28



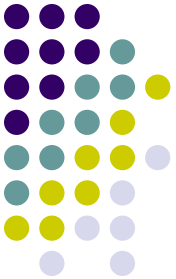
位元運算子(2/2)

- 位元運算子的使用 ：

```
01  // Ch4_6,位元運算子
02  public class Ch4_6{
03      public static void main(String[] args){
04          int a=7, b=13;
05          System.out.printf("%d & %d = %d\n",a, b, a&b);    // 位元 AND
06          System.out.printf("%d | %d = %d\n",a, b, a|b);    // 位元 OR
07          System.out.printf("!%d = %d\n",a, ~a);            // 位元 NOT
08          System.out.printf("%d ^ %d = %d\n",a, b, a^b);    // 位元 XOR
09          System.out.printf("%d >> 2 = %d\n",a, a>>2);      // 右移兩個位元
10          System.out.printf("%d << 2 = %d\n",a, a<<2);      // 左移兩個位元
11      }
12  }
```

執行結果：

```
7 & 13 = 5
7 | 13 = 15
!7 = -8
7 ^ 13 = 10
7 >> 2 = 1
7 << 2 = 28
```



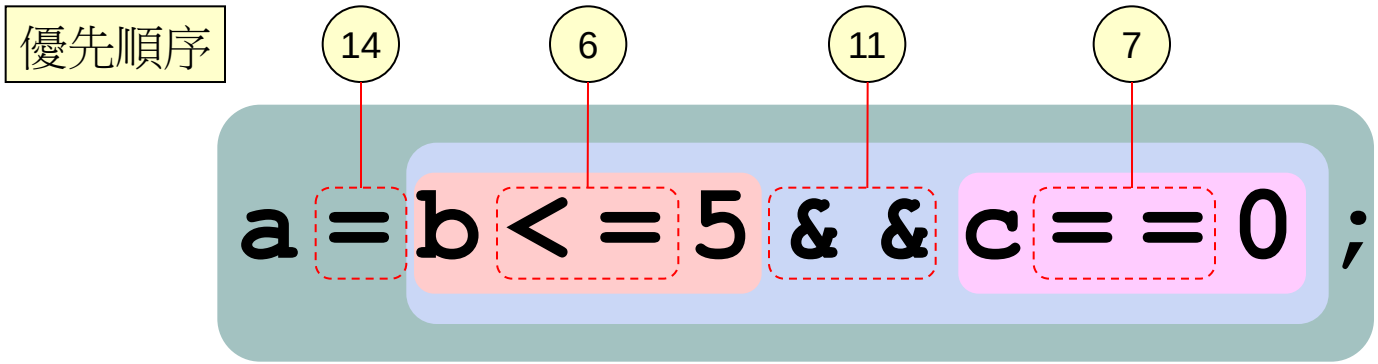
運算子的優先順序列表

Level	Operator	Description	Associativity
16	() [] .	parentheses array access member access	left-to-right
15	++ --	unary post-increment unary post-decrement	left-to-right
14	+ - ! ~ ++ --	unary plus unary minus unary logical NOT unary bitwise NOT unary pre-increment unary pre-decrement	right-to-left
13	() new	cast object creation	right-to-left
12	* / %	multiplicative	left-to-right
11	+ - +	additive string concatenation	left-to-right
10	<< >> >>>	shift	left-to-right
9	< <= > >= instanceof	relational	left-to-right
8	== !=	equality	left-to-right
7	&	bitwise AND	left-to-right
6	^	bitwise XOR	left-to-right
5	|	bitwise OR	left-to-right
4	&&	logical AND	left-to-right
3	||	logical OR	left-to-right
2	?:	ternary	right-to-left
1	= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	right-to-left
0	->	lambda expression arrow	right-to-left

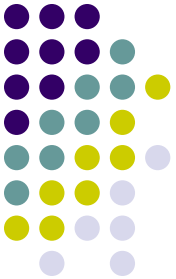
數字愈小表示
優先順序愈高

運算子的優先順序

- 運算子優先順序的範例：



1. 先計算 b<=5 (<=的優先順序為6)
2. 再計算 c==0 (==的優先順序為7)
3. 然後進行&&運算 (&&的優先順序為11)
4. 最後再把運算結果設給變數 a 存放 (= 的優先順序為14)



結合性

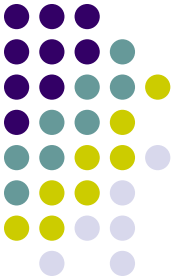
- 結合性是指相同優先順序之運算子的執行順序
- 算術運算子的結合性為「由左至右」

```
a=b+d/3*6;      //  結合性可以決定運算子的處理順序
```

d會先除以3再乘以6得到的結果
加上b後，將整個值給a存放

- 適時加上括號可以提高運算子的優先順序

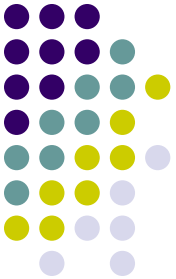
```
12-2*6/4+1;          //  未加括號的運算式
(12-2*6)/(4+1);      //  加上括號的運算式
```



運算式的組成

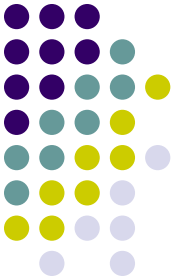
- 運算式是由常數、變數或是其它運算元與運算子所組合而成
- 下面的例子，均是屬於運算式

```
-18                 // 由一元運算子「-」與整數18組成
sum+6               // 由變數sum、算術運算子與整數6組成
a+b-c/(d*3-9)       // 由變數、整數與算術運算子組成
```



運算式的資料型態轉換 (1/3)

- Java處理型態轉換的規則：
  - 佔用位元組較少的型態轉換成位元組較多的型態
    - 字元型態會轉換成int型態（字元會取其unicode碼）
    - short型態（2 bytes）遇上int型態（4 bytes），會轉換成int型態
    - int型態會轉換成float型態
    - 運算式中若某個運算元的型態為double，則另一個運算元也會轉換成double型態
  - 布林型態不能轉換成其它的型態



運算式的資料型態轉換 (2/3)

- 型態轉換的範例：

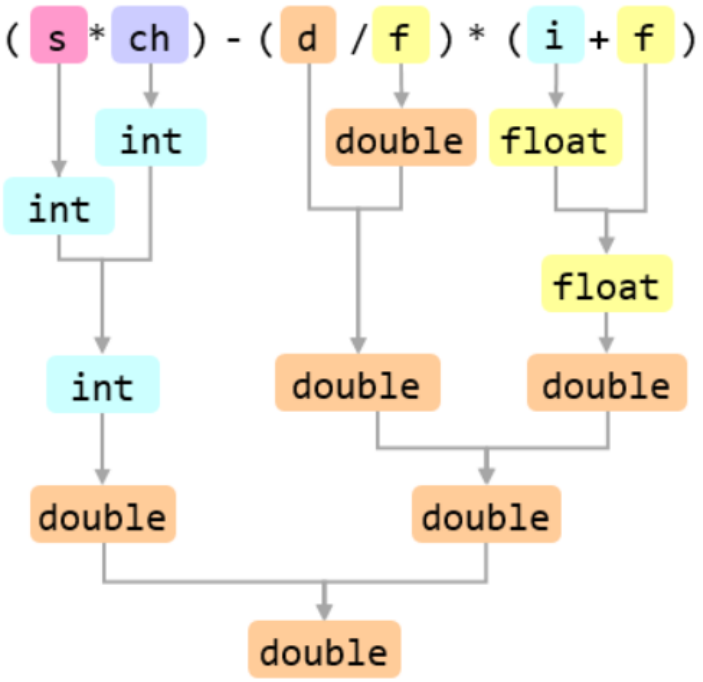
```
01  // Ch4_7,運算式的型別轉換
02  public class Ch4_7{
03      public static void main(String[] args){
04          char ch='m';
05          short s=-5;
06          int i=6;
07          float f=9.7f;
08          double d=1.76;
09                System.out.print("(s*ch)-(d/f)*(i+f)=");
10          System.out.printf("%7.3f",(s*ch)-(d/f)*(i+f));  // 印出結果
12      }
13  }
```

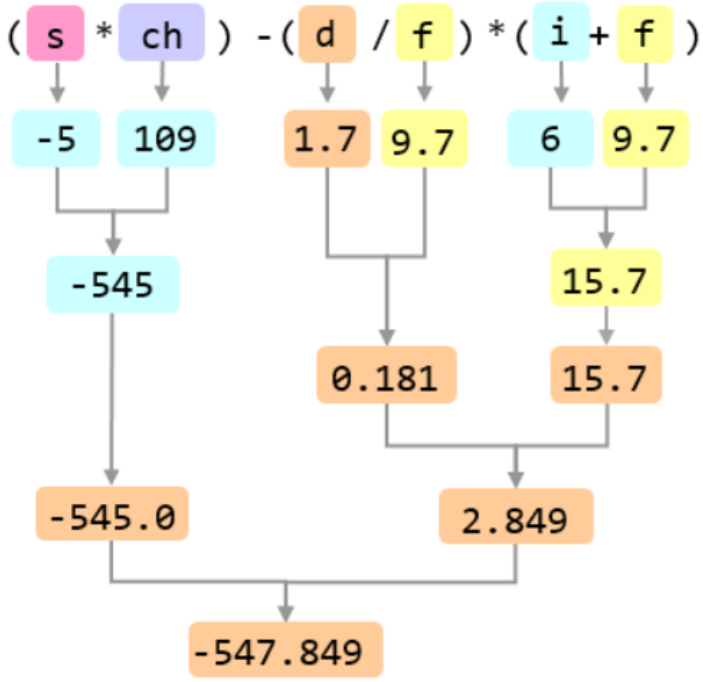
- 執行結果：

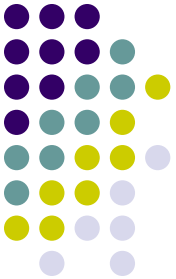
```
(s*ch)-(d/f)*(i+f)=-547.849
```

運算式的資料型態轉換 (3/3)

- 下圖為Ch4_7內變數的資料型態轉換過程解說：







強制型別轉換 (1/2)

- 強制將變數轉換至其他型別的語法：

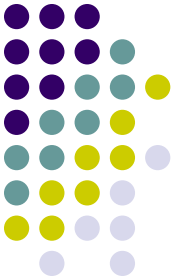
變數轉換型別的語法

```
(type) var;    // 將 var 轉換成 type 型別
```

- 將變數強制轉換成另一種型別時，變數原先的型別並不會被改變
- 將較大型別的變數設給較小型別的變數存放時，稱為縮小轉換（Narrowing cast）
- 縮小轉換可能會遺失資料精度，Java不會自動做轉換，此時就必須進行強制型別轉換

強制型別轉換時可能會發生溢位

```
int a=129;
byte b=(byte)a;
```



強制型別轉換 (2/2)

- 下面的範例說明整數是如何進行強制轉換成浮點數：

```
01  // Ch4_8, 強制型別轉換
02  public class Ch4_8{
03     public static void main(String[] args){
04        int a=25;
05        int b=9;
06
07        System.out.printf("a=%d, b=%d\n",a,b);    // 印出 a、b 的值
08        System.out.printf("a/b=%d\n",a/b);       // 印出 a/b 的值
09        System.out.printf("(float)a/b=%6.3f\n",(float)a/b);
10     }
11  }
```

執行結果：

```
a=25, b=9
a/b=2
(float)a/b= 2.778
```

可改寫成 a/(float)b 或是 (float)a/(float)b

將 a 轉換成浮點數之後，再除以 b

不可寫成 (float)(a/b)


-The End-