第三章 變數與運算式

課前指引

在Java語言中，資料以變數來加以儲存，資料運算則是利用Java所提供的眾多運算子來處理資料。運算子與運算元將組合成運算式，而運算式只要在結尾加上一個「;」就形成了最簡單的單一敘述。

章節大綱

備註：可依進度點選小節

3.1 變數與資料型態

- Java的資料處理基本單位為「變數」
  - 在類別內的變數視為該類別的欄位(fields)
  - 在方法內宣告的變數則為該方法的區域變數(local variables)。

3

3.1.1變數的意義

- 變數代表在程式執行過程中可能會被改變的某個數值。
  - 程式的運作過程是靠眾多變數的變化來完成工作
    - 例如：一個計算長方形面積的程式，至少就必須包含3個變數：長、寬、面積。
- 程式的運作
  - 主要是靠CPU與記憶體的合作來完成，而程式中的變數將存放在記憶體（實際運作時可能會被搬移到CPU的暫存器中）。
    - 因此，以上面的範例來說，在記憶體中，就必須儲存長、寬、面積等3個變數。

4

3.1.1變數的意義

- 變數可宣告為類別的欄位或某一方法的區域變數（還有其他種類的變數，但暫時不討論），我們假設有三種狀況：
  - Case1：長期保留長、寬及面積三個變數。
  - Case2：只想要保留面積變數。
  - Case3：不想保留任何變數，純粹只是要測試Java的「=」與「*」運算子功能。

5

3.1.1變數的意義

- **Case1：**
  - 我們想要長期保留長、寬及面積三個變數。
- **解：**
  - 我們可以先將長方形宣告為類別，然後將長、寬及面積都宣告為長方形類別的變數（即該類別的欄位）
  - 如此一來，這三個變數就屬於該類別所有，依照其保護等級也可以限制不能被其他類別使用。但該類別所屬的方法可使用這三個變數
  - 因此，我們可以在該類別下宣告一個computeArea()方法，在該方法內實現讀入並設定長、寬兩個變數以及計算並設定面積變數。

6

3.1.1變數的意義

- Case2：
  - 我們只想要保留面積變數。
- 解：
  - 我們可以先將長方形宣告為類別，然後將面積宣告為長方形類別的變數（即該類別的欄位），並在該類別下宣告一個computeArea()方法。
  - 然後在computeArea方法內宣告長、寬兩個區域變數。並於方法內設定長、寬變數後，計算並設定面積變數，然後結束computeArea()方法。
  - 當computeArea()方法結束時，長、寬兩個區域變數所佔用的記憶體將會被釋放，但面積由於屬於類別的欄位，因此不會被釋放而得以保留下來。

7

3.1.1變數的意義

- Case3：
  - 我們並不想保留任何變數，純粹只是要測試Java的「=」與「*」運算子功能。
- 解：
  - 我們並不需要宣告任何特定類別，只要在主類別的main方法中宣告長、寬及面積三個區域變數即可
  - 並在main方法中設定長、寬的變數值，然後透過運算式計算出面積變數的值即可。
  - **由於我們尚未介紹類別等物件導向程式設計，因此在本章中，我們都將採用這個案例來解說。**

8

3.1.2 變數的命名

- 變數的命名原則
  - 在第二章已經詳述
  - 例如我們可以將長、寬、面積等命名為length, width, rectArea。

9

3.1.3 變數的宣告

- Java語言規定，任何變數使用前必須先宣告
  - 為了方便起見，一般我們會把區域變數的宣告放在方法的開始處，而其他的運算則放置於變數宣告之後
    - 這樣做的好處在於容易管理
    - 變數一開始就宣告了，後面的運算敘述可以直接取用。
- 宣告變數時，必須宣告該變數的資料型態
  - Java的資料型態分為**原始資料型態**(primitive type)與**非原始資料型態**(non-primitive type)兩類。
    - 本章將以原始資料型態為主來宣告變數。

10

3.1.3 變數的宣告

圖3-1　Java的所有資料型態

11

3.1.3 變數的宣告

- 以原始資料型態宣告變數時，變數的宣告應該以最適當的資料型態為主
  - 宣告不適當的資料型態可能無法負擔未來變數的變化，或者浪費記憶體空間。
  - 不適當的變數宣告，可能導致資料處理結果並非預期想要的結果。

12

3.1.3 變數的宣告

- 舉例來說，若X為int資料型態（整數），Y為float資料型態（浮點數，也就是小數），經過以下運算後，會得到不同的結果。
  - X = 20/5 = 4　　Y = 20/5 = 4
  - **X = 20/6 = 3**　　Y = 20/6 = 3.333333
  - 這是因為在做數學運算時，整數變數無法儲存小數的數值，因此X計算結果與預期有所差距。

老師的叮嚀

上述範例中的Y變數之記憶體內容不可能存放真實的20/6之結果，這是因為**記憶體的長度有限**，不可能完整表達無理數及循環小數。因此，使用電腦計算無理數（例如圓周率）時常會發生誤差，但這些誤差在一般應用時是可接受的，而在特殊應用狀況下（例如航太工業），則必須透過數值方法等更特殊的程式技巧來減少誤差範圍。

13

3.1.4 原始資料型態

原始資料型態名稱	宣告	佔用位元組	說明
位元組	byte	1	-128 ~ 127之整數
短整數	short	2	-32,768 ~ 32,767
整數	int	4	-2,147,483,648 ~ 2,147,483,647
長整數	long	8	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
字元	char	2	16-bit Unicode字元 最小值為'\u0000' (或0) 最大值為'\uffff' (或65,535)
布林	boolean	1	true或false
單倍精準度浮點數	float	4	採用32-bit IEEE 754浮點數格式 精確位數為7位，在正數方面可表達3.4E-38~3.4E+38 （E後面的數字代表10的次方數）
雙倍精準度浮點數	double	8	採用64-bit IEEE 754浮點數格式 精確位數為15位，在正數方面可表達1.7E-308~1.7E+308 （E後面的數字代表10的次方數）

表3-1 Java的原始資料型態

14

3.1.4 原始資料型態

- 字串並非Java的原始資料型態
- 整數資料型態（byte, short, int, long）
  - **整數**資料型態可用來代表正負整數。
  - 在Java程式中的整數，可以宣告為4種資料型態，分別是byte, short, int, long，可表示的數值範圍如表3-1所列
    - 其中最常使用的是int
    - 當數值過大int不足以記錄時，則可以使用long來宣告
    - 而short則是為了節省記憶體時使用。
    - 至於byte則因為所能表達的範圍實在很小，通常較少使用，但由於非常節省記憶體空間，因此若搭配陣列使用的話，則可以大幅減少記憶體需求。

15

3.1.4 原始資料型態

- 浮點數資料型態（float, double）
  - 浮點數是可包含小數的正負數值，分為單精準度與雙精準度兩種
    - 宣告使用float與double。
  - float資料型態的變數佔記憶體4個位元組（32位元），精確位數為7位，在正數方面可表達3.4E-38~3.4E+38
    - （E後面的數字代表10的次方數）。
  - 經宣告為double資料型態的變數將佔用記憶體8個位元組（64位元），精確位數則為15位，在正數方面可表達1.7E-308~1.7E+308。

延伸學習：**實數為何稱為浮點數呢？**
整數資料的小數點，「固定」在最右邊的位元之後，而浮點數表示法的小數點位置則必須由數值與精確度來決定（小數點位置是「浮動」的，因此稱為浮點數表示法）。

小數點表示法	科學記號表示法
7654.321	7.654321E+03
0.004721	4.721000e-03
-123.456	-1.234560e+02

16

3.1.4 原始資料型態

- 布林資料型態（boolean）
  - 布林資料型態的變數只能存放『真(true)』或『假(false)』兩種值
    - 通常在條件判斷時，我們常使用布林資料型態的變數。
    - 然而某些運算式的結果若為布林值，此時也可以將運算式的結果存放在布林資料型態的變數中。
    - 請注意，布林資料型態的變數名稱一般會取isXXXX 之類的名稱，例如isZero，以代表非真即假的特性。
- 字元資料型態（char）
  - char稱為字元資料型態，可以用來儲存單一字元
  - 由於Java使用Unicode來表示字元，因此每個字元資料型態佔用記憶體2個位元組（16位元）。

17

3.1.4 原始資料型態

- 字元資料型態（char）
  - 在Java中指定Unicode字元，可以直接以「'」包裝可列印字元，例如：'H'等
  - 也可以使用數值方式指定字元的Unicode之十進位數值，例如：72。
  - 使用數值方式指定字元的Unicode，除了可以使用十進位來設定之外，可以透過二進位、八進位、十六進位等常數來指定，格式分別為0bttttttttt、0ddd、0xhhhh。
    - 例如以數值0b01001000（二進位）、0110（八進位）、0x0048（十六進位）設定字元內容時，這三種表示法代表的都是字元『H』。
      - （英文字母的Unicode詳見附錄C）

18

3.1.4 原始資料型態

- 字元資料型態（char）
  - 要指定不可列印的**跳脫字元（Escape sequence character）**時，可以搭配特殊符號『\』來表達
    - 例如『\n』代表換行的跳脫字元，能夠使得螢幕游標跳到下一行。
    - 常用的跳脫字元如表3-2所列。
    - 至於使用數值方式指定跳脫字元，則仍只需要指定Unicode對應的數值即可。

19

3.1.4 原始資料型態

跳脫字元	效果
\'	單引號『'』
\"	雙引號『"』
\\	反斜線符號『\』
\a	響鈴（Bell）
\n	換行(Newline)
\b	退格一格（Backspace）
\t	水平跳格，與tab鍵作用相同(Tab)
\f	通知印表機，資料由下一頁開始列印（Format）
\r	回到該行第一格(Carriage Return)
\ddd	8進制數字指定Unicode（例如'H'可表示為'\110'）
\uhhhh	16進制數字指定Unicode（例如'H'可表示為'\u0048'）

表3-2：跳脫字元表

20

3.1.5變數的宣告語法及範例

- 宣告原始資料型態變數的基本語法如下：

  原始資料型態　變數1,變數2,……;

- 範例：

```
int x;          // 宣告x為整數變數
boolean k;      // 宣告k為布林變數，
float p;        // 宣告p為單精準度浮點數變數
double q;       // 宣告q為雙精準度浮點數變數
char c1,c2;     // 宣告c1,c2為字元變數
```

21

3.1.5變數的宣告語法及範例

- 【說　明】
  - 您可以將資料型態當作一種製作箱子的模子
  - 模子可用來製作變數或常數
  - 不同種類的模子製作出來的箱子只能放入特定種類的資料
  - 如果硬要放進不合規定的資料，則可能無法通過編譯，或者資料在放入後箱子後，可能會發生變形而喪失原本的完整資訊。

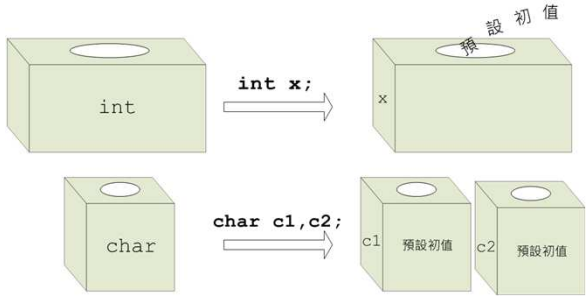


圖3-2 變數宣告示意圖

22

3.1.5變數的宣告語法及範例

- 原始資料型態變數宣告並設定初始值的基本語法如下：

```
原始資料型態　變數1=初始值;
```

- 範例：

```
int x=2000,y=5;     //宣告整數變數x=2000,y=5
boolean k=false;    //宣告布林變數k=false
float p=3.237;      //宣告單精準度浮點數變數p=3.237
char c='r';         //宣告字元變數c='r'，單一字元需透過『'』包裝
```

23

3.1.5變數的宣告語法及範例

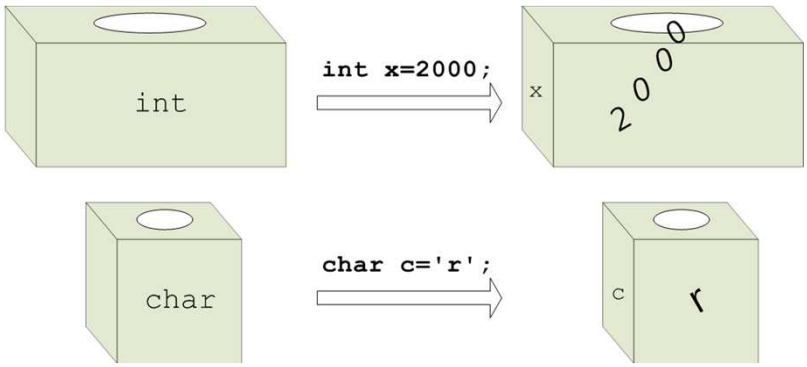


圖3-3 變數宣告並放入初始值示意圖

24

3.1.6常數

- 現實中，光速、圓周率等永遠不會變化。在程式中，同樣有一些不會變化的數值，例如1、3、5.2等，它不論程式執行到何處都不會產生變化（除非硬體故障）。這些不會變化的數值稱為**常數**。
  - 常數在Java中也應該有其資料型態的分別
  - 字元與布林常數和其他語言沒有太大的差別，但對於整數及浮點數，Java語言有其特別規定。
  - 對於一個未指定資料型態的整數常數而言，Java將之視為**int**資料型態，所以下列語法是錯誤的：
    - `long bigNum=10000000000;  //錯誤，因為常數超過int整數範圍`

25

3.1.6常數

- 如果要將整數常數指定為long型態的範圍，則必須在後面加上『L』或『l』以作為標示，一般都採用大寫『L』來標示，如下範例：
  - `long bigNum=10000000000L;  //正確，可通過編譯`
- 對於一個未指定資料型態的浮點數常數而言，Java將之視為**double**資料型態
- 也可以直接指定為float資料型態，只要在其後加上『f』或『F』即可。
- 若在後面加上『d』或『D』則代表該常數為double資料型態，如下範例。
  - `float p1=5.3f;  //5.3為float單倍精準度浮點數資料型態`
  - `float p2=2.0f;  //2.0為float單倍精準度浮點數資料型態`
  - `double p3=7.532D; //7.532為double雙倍精準度浮點數資料型態`
  - `double p4=6.0d;  //6.0為double雙倍精準度浮點數資料型態`

26

3.1.6常數

- 常數代碼
  - 在許多狀況下，為了避免溢位的發生，通常會在程式中，檢查數值是否已達資料型態的範圍極限
    - Java將這些範圍極限常數值記錄於java.lang類別庫的各類別中，使用者可以直接取用該代碼以代表各數值。
    - 事實上，這些代碼就是類別變數(Class Variable)，因此可以直接透過指定類別來存取，而不需要宣告物件實體，各常數代碼如表3-3所列：

27

3.1.6常數

	byte		short	
隸屬類別	java.lang.Byte		java.lang.Short	
最大值	MAX_VALUE	127	MAX_VALUE	32,767
最小值	MIN_VALUE	-128	MIN_VALUE	-32,768

	int		long	
隸屬類別	java.lang.Integer		java.lang.Long	
最大值	MAX_VALUE	2,147,483,647	MAX_VALUE	9,223,372,036,854,775,807
最小值	MIN_VALUE	-2,147,483,648	MIN_VALUE	-9,223,372,036,854,775,808

	float		double	
隸屬類別	java.lang.Float		java.lang.Double	
正數最大值	MAX_VALUE	3.4028235E38	MAX_VALUE	1.7976931348623157E308
正數最小值	MIN_VALUE	1.4E-45	MIN_VALUE	4.9E-324

表3-3 常數代碼

28

3.1.6常數

延伸學習：**溢位**

當某個變數的大小超過所能表示的範圍時，將會產生溢位，此時，變數內容將不是一般預期的結果。
例如宣告為short的變數a若已經為32,767（記憶體內容為0111111111111111），則再將a加1之後，就會發生溢位（它將會是-32768而非+32768，
原因是-32,768的2's補數表示法恰為1000000000000000）。

- 整數常數的特殊表示法
  - 整數常數除了使用十進制數值表示之外，也可以使用二、八、十六進制數值來表示。
  - 除此之外，相對於商用數值的千位數「,」分段符號，也可以改為「_」符號來取代，範例如下：

29

3.1.6常數

- 【範例】：
  - int i=0b00000011;
    - //宣告並設定變數i=3
  - int j=079;
    - //宣告並設定變數j=72（72的八進位表示為79）
  - int k=0x64;
    - //宣告並設定變數k=100（100的十六進位表示為64）
  - int z = 1_000_000;
    - //宣告並設定變數z=1000000

30

3.1.7原始資料型態變數的預設值

- 如果宣告變數時，沒有給定變數的預設值，則Java語言規定變數的預設值為0或false，如表3-4所列。

原始資料型態名稱	預設值
byte	(byte)0
short	(short)0
int	0
long	0L
char	'\u0000'
boolean	false
float	0.0f
double	0.0d

表3-4　Java 的原始資料型態變數預設值

- 【註】：
  - String或其他各類物件的預設值則為null

31

3.1.7原始資料型態變數的預設值

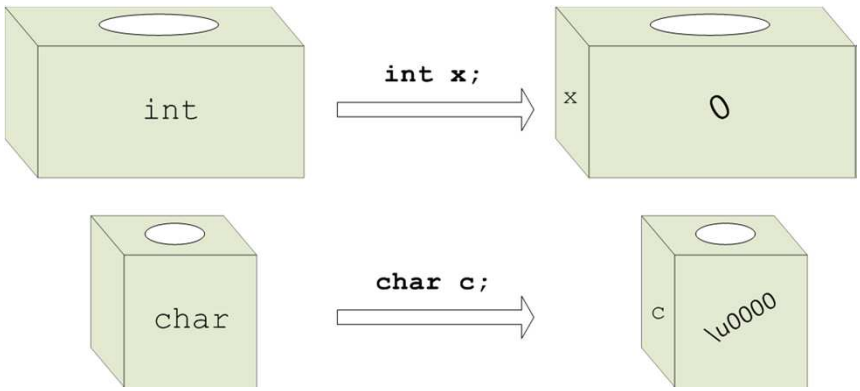


圖3-4 變數宣告未設定初始值示意圖

32

3.1.8不會變動的變數

- 現實世界中的常數是不會變動的一個值，有它本身的意義（例如圓周率、光速）。一般而言，為了彼此的溝通，我們會替該數值取一個符號。
  - 例如圓周率在數學中會以π來代表。
- 符號在程式中是以變數方式來出現，如果這個變數的值在程式中不會產生異動，我們可以將之宣告為不會變動的變數，其語法如下：

```
final  原始資料型態   變數=常數值;
```

- 【語法說明】
  - 將變數宣告為final，則變數一經設定初始的常數值後，就不允許再被修改。因此，我們可以將程式中，不需改變的常數使用此法來宣告。如下範例：
    - final float pi=3.1416f;
    - final float rate=0.0325f;

33

3.1.8不會變動的變數

- 【說明】：
  - pi是圓周率，一般來說，是一個固定值且為無窮小數。而rate可能是做為利率，若在計算貸款、存款的程式中，可能使用於許多公式中。
  - 為了統一管理起見，可以將之宣告為不會變動的變數，如果程式想要修改該值時（例如利率調整或增加圓周率的精確度），只要在宣告處修改數值即可，而不必在使用該數值的程式各處一一修改該值。

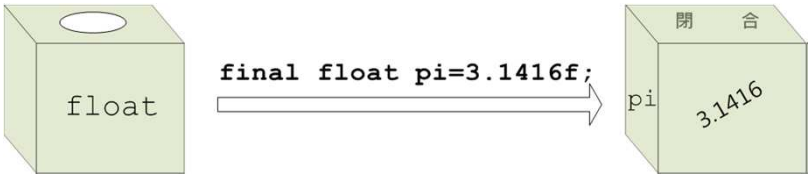


圖3-5 final變數宣告並放入初始值示意圖

34

3.1.8不會變動的變數

- 對於區域變數而言，宣告final變數可以在宣告時不設定初值，而留待後面才設定初值，但一旦設定初值之後，就不能再改變其值。如下範例：

```
final float rate;
rate=0.0325f;
rate=0.028f; //本行程式碼錯誤
             //final變數不可以設定兩次以上的值
```

延伸學習：**final的多用途性**

final關鍵字除了上述功能之外，在類別的繼承時，也可以用來終止類別的繼承。我們將於後面章節中再做說明。

35

3.1.9 字串

- 字串雖然不是原始資料型態的一種，但卻經常被使用。
  - 例如：在Java程式中，不論是讀取使用者於鍵盤輸入的資料，或檔案中的資料，所讀取到的都是字串格式。即使內容為數值，也會被當作字串來處理。

36

3.1.9 字串

- 宣告字串物件
  - 在Java中使用字串，必須透過String類別來處理
  - 「類別的實體變數」稱之為「物件」
  - 因此要存放字串，必須使用String物件來存放。其語法有下列幾種：
    - 使用建構函式（建構子）
    - 用原始資料型態的配置語法

37

3.1.9 字串

- 一、使用建構函式（建構子）

```
String  字串物件名稱  =  new  String(初始值);
```

- 【說明】：
  - 此法可同時設定字串的初始值，如果初始值要設定為空字串，則( )內為空即可。
  - 建構函式是「當該物件生成時，會自動執行的方法」，我們將於後面章節中介紹，而其中的初始值，則為傳遞給建構函式的引數。

38

3.1.9 字串

- 二、用原始資料型態的配置語法

```
String   字串物件名稱;
```

或

```
String   字串物件名稱=初始值;
```

- 【說明】：
  - 此法可設定或不設定字串的初始值。

39

3.1.9 字串

- 設定字串值
  - 由宣告語法可知，設定字串物件內的字串值，可以直接使用『=』來指定即可，如下範例：

```
String str1 = new String ("This is str1");
String str2 = new String();
String str3 = "This is str3";
String str4;
str1 = "This is new str1";
str2 = "This is str2";
str3 = "This is new str3";
str4 = str3;
```

40

3.1.9 字串

- 【說明】
  - 上面的敘述執行完畢時，str1~str3所代表的字串內容為最後設定的狀況，而str4與str3實際上使用相同的字串物件（內容當然也相同）。
  - 值得注意的是，雖然上述的語法都可以宣告字串物件並設定字串內容，但實際上背後的運作則有一些差異，我們將於第11章再深入介紹。

延伸學習：**字串常數**

當程式中出現了字串常數，則一定會產生一個String物件以存放其內容，並且Java採用String Pool及自動記憶體回收機制有效管理眾多字串常數。（我們將於第11章詳述）

41

3.1.9 字串

- 字串的內容
  - 在Java中，字串以類別來宣告，以物件方式呈現，而字串由於是由字元所組成，因此，字串的內容事實上是存放在char[]型態的字元陣列中
    - 該字元陣列則為String類別的成員變數。

42

3.2資料輸入

- 螢幕輸出可以透過System.out所屬類別的print或println方法來完成，而如果要讀入使用者由鍵盤輸入的資料，則比較複雜。
  - 一般而言，有java.util.Scanner類別與java.io.Console類別可完成此項工作。

43

3.2資料輸入

- java.util.Scanner類別
  - 使用Scanner物件可讀取輸入裝置（例如檔案）上的資料
    - 因為鍵盤為標準輸入裝置，因此將引數設定為輸入裝置的代碼System.in即可：
  - 使用Scanner 物件讀取鍵盤輸入的格式如下：

```
import java.util.Scanner;
public class 主類別名稱{
   public static void main(String args[])
   {
      Scanner keyboardInput = new Scanner(System.in);
      String str1;
      str1 = keyboardInput.nextLine();
      ..... 其他程式碼.....
   }
}
```

我們將透過此物件讀取鍵盤的輸入。

讀入一行輸入的文字，並存入str1字串中。

44

3.2資料輸入

- 【說明】：
  - (1)使用java.util.Scanner類別
    - 要透過import java.util.Scanner;載入類別。
  - (2)宣告一個keyboardInput物件變數為Scanner類別的物件（也可以取其他名稱），並透過new產生物件實體
    - 產生物件實體時傳入代表鍵盤的System.in引數，那麼keyboardInput物件就可以用來代表由鍵盤接收資料的物件。
  - 方法nextLine()會回傳鍵盤輸入的一行資料給字串
    - 使用字串物件str1來接收資料。

45

3.2資料輸入

- 範例3-1：ch3_01.java（隨書光碟myJava\ch03\ch3_01.java）

```
1   /*  檔名:ch3_01.java    功能:鍵盤輸入範例    */
2
3   package myJava.ch03;
4   import java.lang.*;
5   import java.util.Scanner;
6
7   public class ch3_01          //主類別
8   {
9       public static void main(String args[])
10      {
11          Scanner keyboardInput = new Scanner(System.in);
12          String str1,str2;
13
14          System.out.print("請輸入第一個字串:");
15          str1 = keyboardInput.nextLine();
16          System.out.print("請輸入第二個字串:");
17          str2 = keyboardInput.nextLine();
18          System.out.println("您所輸入的字串如下:");
19          System.out.println(str1);
20          System.out.println(str2);
21      }
22  }
```

46

3.2資料輸入

- 執行結果：

```
請輸入第一個字串:This is String 1
請輸入第二個字串:This is String 2
您所輸入的字串如下:
This is String 1
This is String 2
```

- 範例說明：
  - 第11行是宣告並產生物件實體keyboardInput
  - 第15行與第17行則是透過物件的nextLine方法取得鍵盤輸入的一行字串。

3.2資料輸入

- java.io.Console類別
  - JDK6提供了新的類別java.io.Console，它主要是針對標準輸出入裝置所設計，可以透過它取得使用者在文字模式(console mode)下的鍵盤輸入。
    - Eclipse並未完整將java.io.Console類別實作並整合到IDE之中，因此發生錯誤。
    - 如果您採用的是第一章介紹的命令列模式來編譯與執行就不會有問題。
  - 使用Console物件讀取鍵盤輸入的格式如下：

3.2資料輸入

```
import java.io.Console;

public class 主類別名稱{
  public static void main(String args[])
  {
     Console console = System.console();
     String str1;
     char[] PW;
     str1 = console.readLine();
     ..... 其他程式碼.....
     PW = console.readPassword(" 請輸入密碼:");
     ..... 其他程式碼.....
  }
}
```

我們將透過此物件讀取鍵盤的輸入。

讀入一行輸入的文字，並存入str1字串中。

讀入一行密碼並存入PW 字元陣列中。密碼不會顯示於螢幕上。

- 【說明】：
  - 由於要使用java.io.Console類別，因此透過import java.io.Console;載入類別。當然您也可以寫為java.io.*一次載入所有屬於該類別庫下的類別。

49

3.2資料輸入

- console是Console類別的物件（也可以取其他名稱）。其方法readLine會回傳鍵盤輸入的一行資料給字串。
  - 使用字串物件str1來接收這些資料。
- readPassword方法的引數"請輸入密碼:"是提示字串，您可以填入任何想要在輸入密碼前出現的字串，由於該方法被「多載」定義，故也可以省略該字串，改用System.out.print輸出提示字串。請見範例3-2。

延伸學習：**多載(overloading)**

多載代表可在呼叫方法時，提供不同種類、數量的引數，我們會在第六章詳述多載技術。

50

3.2資料輸入

- 範例3-2：ch3_02.java（隨書光碟 myJava\ch03\ch3_02.java）

```
1   /*  檔名:ch3_02.java  功能:鍵盤輸入範例 - 不顯示輸入字元   */
2
3   package myJava.ch03;
4   import java.lang.*;
5   import java.io.Console;
6
7   public class ch3_02          //主類別
8   {
9       public static void main(String args[])
10      {
11          Console console = System.console();
12          String str1;
13          char[] PW; // 宣告字元陣列
14
15          System.out.println(" 請輸入帳號:");
16          str1 = console.readLine();
17          PW = console.readPassword(" 請輸入密碼:");
18          System.out.println(" 您所輸入的帳號及密碼如下:");
19          System.out.println(str1);
20          System.out.println(PW);
21      }
22  }
```

51

3.2資料輸入

- 執行結果：

實際上按下了123與Enter，但不會顯示出來

```
請輸入帳號:
abc
請輸入密碼:
您所輸入的密碼如下:
123
```

- 範例說明：
  - (1) 由於readPassword方法的回傳值為字元陣列，故在第13行宣告字元陣列inputPW，詳見陣列一章。
  - (2) 第16行讀取文字時執行的是readLine()方法，所以在螢幕上可以看見使用者輸入的字串abc，這個字串會存入str1字串中。
  - (3) 第17行的提示字串為" 請輸入密碼:"。
  - (4) 第20行的System.out.println也可以輸出字元陣列。

52

3.2資料輸入

- (5) 執行結果中，我們實際上輸入了密碼字元「123」，但螢幕並不會顯示出來，若非我們在第20行將之列印出來，則外人無法得知我們輸入的密碼為何，因此具有保密特性。
- (6) 這個範例無法在Eclipse中正確執行。

延伸學習：**printf**

事實上，java.io.Console類別也提供了輸出的方法printf()，對於C熟悉的讀者可以試試看，例如：console.printf("字串為:"+str1+"\n");，不過由於過去多數人已經習慣使用System.out.println來輸出並且也很簡單，因此，我們並不打算使用Console來進行輸出。

53

3.3運算式（運算子及運算元）

- 程式的目的是解決問題，而手段則是依靠各個變數值的改變，想要變更變數值就必須透過**運算式(Expression)**加以完成。
  - 運算式是由**運算元(Operand)**與**運算子(Operator)**共同組成，常見的數學公式就是最基本的運算式，例如：area=r*r*3.14 。
    - 其中的『r』、『3.14』就是運算元
    - 而『=』、『*』就是運算子。
  - Java提供了許多的運算子，並且允許運算元為一個或多個的『常數』、『變數』、『函式呼叫敘述』或甚至是『其他運算式』的組合，只要運算結果為單一個值的都可以作為運算元。

54

3.3運算式（運算子及運算元）

- 運算式與敘述
  - 敘述是Java程式中，一個完整執行的單元。
  - 下列的運算式，只要加上結尾「;」，即可成為運算式敘述。
    - 1.指定運算式。例如a=7。
    - 2.使用遞增遞減（++或--）的運算式。例如：a++。
    - 3.方法的呼叫。
      - 例如：System.out.println("Hello World!")
    - 4.建立物件的運算式。
      - 例如：String str1 = new String ("This is str1")
  - 除了運算式敘述之外，Java還有下列其他種類的敘述。
    - (1)宣告敘述，例如：int a;或int a=8;
    - (2)流程控制敘述（請見下一章）。

55

3.3運算式（運算子及運算元）

- 敘述與區塊
  - 區塊是由0個以上的敘述所組成，又可以稱之為敘述區塊，它以『{』為開頭，『}』為結尾
  - 區塊可以出現在單一敘述可出現之處（故您也可以將區塊當作敘述的一種，稱之為區塊敘述），以作為一個較大單元的執行區塊。

56

3.3.1 『=』設定運算子

- 『=』運算子是最常見的運算符號
  - 作用是將符號右邊的運算式計算後的值指定給左邊的變數
  - 稱為**設定運算子**或**指定運算子**。
  - 格式：

    變數 = 運算式;

  - 範例：

```
a = 10;
b = 'w';
c = p + q;
str1 = keyboardInput.nextLine();
10 = x + 40;          //錯誤，左邊不可以是數值
f(h) = 15;            //錯誤，左邊不可以是函式呼叫
x + y = z;            //錯誤，左邊不可以是複合運算式
```

  - 說明：
    - 『=』運算子的左邊只能有也必須有唯一的一個變數，不能是數值、函式、其他的複合運算式。

57

3.3.1 『=』設定運算子

Coding 注意事項

在Java語言中，『=』的意義為指定或設定，與數學上的相等有些許不同，對於Java而言，相等代表一種比較，因此必須使用比較運算子的『==』符號。

老師的叮嚀

雖然『=』運算子為運算式中最常見的運算子，但並非所有的運算式都必須包含『=』運算子，例如：a++也是一個運算式，但卻不必使用『=』符號。此外，運算式是一種遞迴式的定義，也就是運算式可以做為其他更大的運算式的某一個運算子，例如：z=x+y。其中『x+y』是一個運算式，而『z=x+y』也是一個運算式，並且x+y為『=』運算子的運算元。

58

3.3.2 算術運算子

- 算術運算子可用來做數學的運算，Java提供的算術運算子共有5種：『+』、『-』、『*』、『/』、『%』，如下表所列：

算術運算子	使用範例	說明
+	a + b	a加b
-	a - b	a減b
*	a * b	a乘b
/	a / b	a除以b
%	a % b	取a除以b的餘數

59

3.3.2 算術運算子

- **範例3-3：**ch3_03.java（隨書光碟 myJava\ch03\ch3_03.java）

```
1   /*  檔名:ch3_03.java     功能:算術運算子範例    */
2
3   package myJava.ch03;
4   import java.lang.*;
5
6   public class ch3_03          //主類別
7   {
8     public static void main(String args[])
9     {
10      float answer;
11      float a=2.1f,b=3.5f,c=4.0f;
12      int x,y;
13      x = 20;
14      y = 7;
15      System.out.print("當x=" + x + "    ");
16      System.out.println("y=" + y + "時");
17      System.out.println("x + y = " + (x+y));
18      System.out.println("x - y = " + (x-y));
19      System.out.println("x * y = " + (x*y));
20      System.out.println("x / y = " + (x/y));
21      System.out.println("x % y = " + (x%y));
22      System.out.println("----------------------");
23      System.out.print("當a=" + a + "   ");
24      System.out.print("b=" + b + "    ");
```

y不可為0

60

3.3.2 算術運算子

```
25  System.out.println("c" + c + "時");
26      answer = b*b-4*a*c;
27      System.out.println("b^2-4ac=" + answer);
28    }
29  }
```

- 執行結果：

```
當x=20     y=7時
x + y = 27
x - y = 13
x * y = 140
x / y = 2
x % y = 6
----------------------
當a=2.1    b=3.5    c4.0時
b^2-4ac=-21.349998
```

第20行的輸出

61

3.3.2 算術運算子

- 範例說明：
  - (1)第20行，由於x、y都是整數資料型態，因此x/y會被轉換成整數資料型態（只能記錄整數部分）。

$$\frac{x}{y} = \frac{20}{7} = 2.857... \xrightarrow{\text{只取整數部分}} 2$$

  - (2)不論是做除法或取餘數時，分母都不可為0。而20%7的結果是餘數6。
  - (3)x*y不可以簡寫為xy，這一點與數學不太一樣。
  - (4)第26行的運算式『answer=b*b-4*a*c』當中，會先做『*』乘法，再做『-』減法，最後才是『=』指定。

62

3.3.3 比較運算子

- Java提供的比較運算子有『==』、『!=』、『>』、『<』、『>=』、『<=』等六種。其運算結果為布林值，也就是true（真）或false（假）。
  - 所以比較運算子常用在條件判斷之用，其餘比較運算子說明如下表所列：

比較運算子	意義	使用範例	說明
= =	等於	x==y	比較x是否等於y
!=	不等於	x!=y	比較x是否不等於y
>	大於	x>y	比較x是否大於y
<	小於	x<y	比較x是否小於y
>=	大於等於	x>=y	比較x是否大於等於y
<=	小於等於	x<=y	比較x是否小於等於y

63

3.3.4 邏輯條件運算子

- 邏輯條件運算子可以將布林資料true或false做某些運算
  - 若將邏輯條件運算子與其他比較運算子搭配使用，運算結果將仍然是布林值，因此常被用來當做條件判斷之用。
  - 在Java中，邏輯條件運算子共有NOT、AND、OR 等3種，其符號分別為『!』、『&&』、『||』，我們以真值表方式來加以介紹。

64

3.3.4 邏輯條件運算子

- (1) ! (NOT)
  - 『!』邏輯條件運算子是一個單元運算子，只需要單一個運算元，它會將緊接著的運算元的值給反相，若輸入的值為true，輸出的值將為false。反之，輸入的值為false，輸出的值將為true。。

X	!X
false	true
true	false

真值表： !X

- (2) && (AND)
  - 當前後兩個運算元都是true，輸出才會是true，其餘的各種情況都是輸出false。

X	Y	X && Y
false	false	false
false	true	false
true	false	false
true	true	true

真值表： X &&Y

65

3.3.4 邏輯條件運算子

- (3) || (OR)
  - 當前後兩個運算元中只要有一個是true，輸出就會是true，只有當兩個運算元都是false時，輸出才會是false。

真值表： X || Y

X	Y	X || Y
false	false	false
false	true	true
true	false	true
true	true	true

66

3.3.4 邏輯條件運算子

- 範例3-4：ch3_04.java（隨書光碟 myJava\ch03\ch3_04.java）

```
1   /*   檔名:ch3_04.java   功能:比較運算子與邏輯運算子範例   */
2
3   package myJava.ch03;
4   import java.lang.*;
5
6   public class ch3_04          //主類別
7   {
8     public static void main(String args[])
9     {
10        int x=10,y=20;
11        boolean a=true,b=false;
12
13        System.out.print(" x=" + x);
14        System.out.println(" y=" + y);
15        System.out.println("---------------");
16        System.out.println("x==y ==> " + (x==y));
17        System.out.println("x!=y ==> " + (x!=y));
18        System.out.println("x>y  ==> " + (x>y));
19        System.out.println("x<y  ==> " + (x<y));
20        System.out.println("x>=y ==> " + (x>=y));
21        System.out.println("x<=y ==> " + (x<=y));
```

67

3.3.4 邏輯條件運算子

```
22        System.out.println("==================");
23        System.out.print(" a=" + a);
24        System.out.println(" b=" + b);
25        System.out.println("---------------");
26        System.out.println("not a    ==> " + !a);
27        System.out.println("a and b  ==> " + (a && b));
28        System.out.println("a or b   ==> " + (a || b));
29        System.out.println("a nand b ==> " + !(a && b));
30        System.out.println("a nor b  ==> " + !(a || b));
31        System.out.println("a and (x<y) ==> " + (a && (x<y)));
32    }
33  }
```

68

3.3.4 邏輯條件運算子

- 執行結果：

```
x=10 y=20
---------------
x==y ==> false
x!=y ==> true
x>y  ==> false
x<y  ==> true
x>=y ==> false
x<=y ==> true
=====================
 a=true b=false
---------------
not a    ==> false
a and b  ==> false
a or b   ==> true
a nand b ==> true
a nor b  ==> false
a and (x<y) ==> true
```

- 範例說明：
  - (1)第16~21行，比較運算子的執行結果會回傳一個布林值。所以比較運算子常做為條件判斷之用。

69

3.3.4 邏輯條件運算子

- (2)在第29~30行中，我們使用3種邏輯運算子，完成更多樣的邏輯運算，例如：nand（not and）、nor（not or）。
- (3)第31行，由於(x<y)將會回傳布林值true，而a也為true，因此 a and (x<y)的結果為true。
  - 在往後的程式設計中，當進行條件判斷時，常常會出現關係比較運算子與邏輯運算子合作以完成所需的狀況。

小試身手3-1

在範例3-4中， boolean a=true,b=false;，請寫一個程式印出a xor b的結果。

xor為互斥運算，其真值表如右，當前後兩個運算元的值不相同時，才會輸出true。

並且a xor b等價於((a && !b)||(!a && b))。

真值表： a xor b

a	b	a xor b
false	false	false
false	true	true
true	false	true
true	true	false

70

3.3.5 位元運算子

- 在位元運算方面，Java提供七種位元運算子
  - 透過這些運算子，我們就可以只針對某些位元(bit)做運算處理。
    - 例如：int整數變數x在記憶體佔用32個位元（4個位元組）長度，若x為1234，則實際在記憶體中的內容如下：

$$x = 1234_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 1101\ 0010_2$$

71

3.3.5 位元運算子

- 我們只要透過位元運算子，就可以針對某一些位元單獨做運算，進行更低階的資料處理。Java提供的位元運算子如下表。

位元運算子	意義	範例	說明
~	反相	~x	將x的所有位元做NOT運算
&	且	x & y	將x與y相對應的位元做AND運算
|	或	x | y	將x與y相對應的位元做OR運算
^	互斥	x^y	將x與y相對應的位元做XOR（互斥）運算
>>	右移	x >> p	將x的內容往右移動p個bit
>>>	右移	x >>> p	將x的內容往右移動p個bit（左邊補0）
<<	左移	x << p	將x的內容往左移動p個bit（右邊補0）

72

3.3.5 位元運算子

- 【註】：XOR（互斥）的真值表如下，當前後兩個運算元的值不相同時，才會輸出1。

真值表：X ^ Y

X	Y	X ^ Y
0	0	0
0	1	1
1	0	1
1	1	0

- 範例3-5：ch3_05.java（隨書光碟 myJava\ch03\ch3_05.java）

73

3.3.5 位元運算子

```
1   /*   檔名:ch3_05.java    功能:位元邏輯及移位運算子範例    */
2
3   package myJava.ch03;
4   import java.lang.*;
5
6   public class ch3_05          //主類別
7   {
8     public static void main(String args[])
9     {
10        short x=100,y=50,p=3;
11        int xx=0;
12
13        System.out.println("p=3");
14        System.out.println("x=01100100");
15        System.out.println("y=00110010");
16        System.out.println("--------------------");
17        System.out.println("not xx   ==> " + ~xx);
18        System.out.println("x and y  ==> " + (x & y));
19        System.out.println("x or y   ==> " + (x | y));
20        System.out.println("x xor y  ==> " + (x ^ y));
21        System.out.println("x >>> p  ==> " + (x >>> p));
22        System.out.println("x << p   ==> " + (x << p));
23    }
24  }
```

74

3.3.5 位元運算子

- 執行結果：

```
p=3
x=01100100
y=00110010
--------------------
not xx    ==> -1
x and y   ==> 32
x or y    ==> 118
x xor y   ==> 86
x >>> p   ==> 12
x << p    ==> 800
```

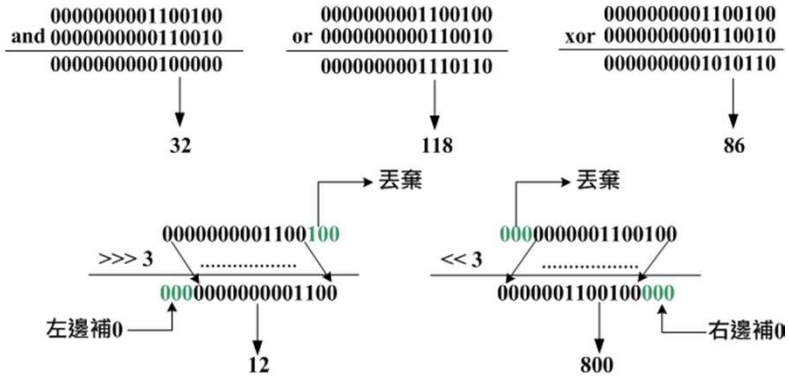
見範例說明(4)

- 範例說明：
  - (1)第10行將x, y宣告為short整數資料型態，所以變數長度為2個位元組。因此x=100=0000000001100100、y=50=0000000000110010。

75

3.3.5 位元運算子

- 範例說明：
  - (2)範例執行結果為何出現這些數字呢？請見以下的實際運算。



76

3.3.5 位元運算子

- 範例說明：
  - (3)善用左移位元符號『<<』可以將數值做2的倍數乘法，例如：100左移3個位元得800，也就是100×2^3=800。
  - (4)xx被宣告為int整數資料型態，佔用4個位元組
    - 由於宣告時指定初值為0（事實上，不指定也可以，因為int資料型態的預設值為0）
    - 因此記憶體為000…000
    - 當透過「~」取反相時，結果為111…111，以2's補數來看待，則為-1。

77

3.3.6複合式指定運算子

- Java除了簡單的指定運算子『=』之外，還提供另一種同時具有運算及指定功能的複合式指定運算子(compound assignments)，此類運算子有很多，如下表所列。

複合式指定運算子	使用方法	功能等同於
+=	i += j;	i = i + j;
-=	i -= j;	i = i - j;
*=	i *= j;	i = i * j;
/=	i /= j;	i = i / j;
%=	i %= j;	i = i % j;
|=	i |= j;	i = i | j;
&=	i &= j;	i = i & j;
^=	i ^= j;	i = i ^ j;
>>=	i >>= j;	i = i >> j;
<<=	i <<= j;	i = i << j;

78

3.3.7 遞增與遞減運算子

- Java提供兩個特別的運算符號：遞增符號「++」、遞減符號「- -」。
  - 遞增符號會將運算元的內容加1
  - 遞減符號會將運算元的內容減1。

遞增與遞減運算子	使用方法	功能等同於
++	a++; ++a;	a = a + 1;
--	a--; --a;	a = a - 1;

79

3.3.7 遞增與遞減運算子

- 『++』和『--』可以放在運算元的後面，也可以放在運算元的前面。放在運算元後面時，代表要做前置運算，如：i ++。放在運算元前面時，代表要做後置運算，例如：++i。兩種運算的意義不同，分述如下：
  - **前置運算（例如：i++）：**
    - 運算元先進行其他運用，再進行加一或減一的動作。
  - **後置運算（例如：++i）：**
    - 運算元先進行加一或減一的動作，再進行其他運用。
- 範例3-6：ch3_06.java（隨書光碟myJava\ch03\ch3_06.java）

80

3.3.7 遞增與遞減運算子

```
1   /*  檔名:ch3_06.java     功能:遞增遞減運算子範例    */
2
3   package myJava.ch03;
4   import java.lang.*;
5
6   public class ch3_06          //主類別
7   {
8       public static void main(String args[])
9       {
10          int i=5,j=10,a,b;
11          int x=5,y=10,c,d;
12
13          a = 1+ i++;
14          b = 1+ j--;
15          c=  1+ ++x;
16          d=  1+ --y;
17
18          System.out.print("i = " + i);
19          System.out.print("\t j = " + j);
20          System.out.print("\t x = " + x);
21          System.out.println("\t y = " + y);
22          System.out.print("a = " + a);
23          System.out.print("\t b = " + b);
24          System.out.print("\t c = " + c);
25          System.out.println("\t d = " + d);
26      }
27  }
```

先執行a=1+i然後再執行i=i+1。

先執行x=x+1再執行c=1+x。

81

3.3.7 遞增與遞減運算子

- 執行結果：

```
i = 6    j = 9    x = 6    y = 9
a = 6    b = 11   c = 7    d = 10
```

- 範例說明：
  - (1)雖然i, j, x, y的執行結果都各自遞增或遞減1了，但由於將『++』與『--』放在不同位置，所以a, b, c, d的執行結果並不相同。
    - 第13行是先執行a=1+i然後再執行i=i+1，所以a=6、i=6。
    - 而第15行程式，則是先執行x=x+1再執行c=1+x，所以x=6、c=7。
    - 第14、16行也是相同的道理。
  - (2)另外，讀者會發現c=1+ ++x若刪除空白並且將1也用變數取代時，讀者可能會對於運算結果產生疑惑，例如：c=p+++q，到底是c=p+(++q)還是c=(p++)+q呢？
    - 這其實和運算子的優先權有關，『++』的運算子優先權比『+』優先權還高，因此，c=p+++q相當於c=(p++)+q。
  - (3)在輸出結果時，我們使用了「\t」跳脫字元來對齊資料，它相當於按下<Tab>鍵的效果。

82

3.3.8其他的單元運算子

- 除了上述的「++」、「--」、「!」、「~」之外，「+」與「-」也可以當做是單元運算子
  - 「+」代表後面為正值
  - 而「-」號則代表將運算元乘以『-1』，與數學式子的負號相同。

83

3.3.9其他運算子

- Java還提供另外一些運算子，分述如下：
  - instanceof物件型別判斷運算子（Type Comparison Operator）
    - instanceof運算子可以檢查一個物件的型別，我們可以透過它測試該物件是否屬於某類別或某類別的子類別，或某一個介面的實作。
    - 它的結果將會是一個布林值，代表所測試的結果。詳見9.6節介紹。
  - 三元條件運算子
    - Java提供了條件運算子『?:』，可以用來當做簡單的if-else判別指令，它有三個運算元，如下列範例，我們會在條件判斷一節中詳加說明。

條件運算子	使用範例	說明
?:	a ? b : c	若a為true，則取b，否則取c

84

3.3.9其他運算子

- **其他運算子**
  - 除了以上所介紹的運算子之外，Java語言還提供了某些運算子如下表
  - 表格中某些運算子，我們早就已經在使用了（例如：『,』分隔運算子），有些則於後面章節使用時，再做明確的說明。

運算子	說明
()	應用廣泛，不同位置有不同涵義，例如：函式宣告的參數串列、提高部分運算式的優先權、控制流程的判斷式、執行強制轉型等
{}	區塊的起始與結束。
[]	指定陣列維度及存取陣列元素。
,	分隔運算子，例如：變數宣告的間隔。
.	存取類別的成員或執行類別的方法。
new	建立新的物件或陣列。
->	Lambda 運算式的分隔符號。

85

3.3.10運算子結合性及優先權

- 當出現z=a*x+y時，我們知道要先做a*x再做+y，最後才是做z=。
  - 因為我們熟背『先乘除後加減』。這就是運算子優先權規定。
- Java也規定了所有的運算子優先權，我們將之整理如表3-5，優先權越高的運算子會越先被處理
  - 如果您不確定運算子的優先權時，最好使用小括號『()』將想要先處理的部分運算式括起來。
- 另外一個會影響運算式最後結果的因素是運算子的結合性
  - 就如同數學上的1+2+3會先計算1+2再計算(1+2)+3，而次方計算則並非如此，如下圖：

86

3.3.10運算子結合性及優先權

$$2^{3^2} = 2^9 = 512$$

圖3-6 結合性對於運算結果的影響

$$\neq 8^2 = 64$$

- 明顯地，『+』號的結合性為由左向右，但次方的結合性為由右向左
- 雖然Java並未提供次方運算子，但同樣也有某些運算子的執行方向是由右向左的
- 有時方向並不會影響最後結果（例如：『+』），但有的時候卻會嚴重導致結果不同（例如：『/』）
- 因此我們將Java的所有運算子之結合性也整理於表3-5中。

87

3.3.10運算子結合性及優先權

- 運算子結合性及優先權

優先權	名稱	運算子	結合性
高	名稱	運算子	結合性
	括號運算子	()	由左而右
	陣列運算子	[]	由左而右
	後置運算子	運算式++ 運算式--	由左而右
	單元運算子	++運算式 --運算式 +運算式 -運算式~ !	由右而左
	乘除運算子	* / %	由左而右
	加減運算子	+ -	由左而右
	位移運算子	<< >> >>>	由左而右
	關係運算子	< > <= >= instanceof	由左而右
	相等判斷	== !=	由左而右
	位元的AND運算	&	由左而右
	位元的XOR運算	^	由左而右
	位元的OR運算	|	由左而右
	邏輯AND運算	&&	由左而右
	邏輯OR運算	||	由左而右
	三元運算子	? :	由右而左
低	指定運算子	= += -= *= /= %= &= ^= |= <<= >>= >>>=	由右而左

表3-5 運算子優先權順序及結合性

88

3.4資料型態的轉換

- 在Java程式中，同一個運算式允許出現不同資料型態的運算
  - 但是可能會發生資料型態轉換的問題
  - 本節將討論資料型態轉換的觀念與方法。

89

3.4.1自動資料型態轉換

- 假設程式中包含下列片段程式，a, b被宣告為int與short型別，當我們要將a的值指定給b時，在編譯時期就會發生錯誤
  - 因為a的值可能超過short所能表達的範圍。

```
int a=20;
short b=10;
b=a;      //編譯時期發生錯誤，因為b的表示範圍比a小
a=b;      //沒有錯誤，但會發生自動資料型態轉換
```

  - **但如果我們將b的值指定給a時，則不會發生錯誤**，因為int的表達範圍足以容納b的值。不過這個時候，事實上編譯器已經為您的程式進行了**「自動資料型態的轉換」**。
  - 因為，對於『=』運算子而言，『=』左右的資料型態必須相同。故當我們要將b的值指定給a時，編譯器會先將b的值提升為int資料型態，然後才指定給a。

90

3.4.1自動資料型態轉換

- 為了避免錯誤，所以我們將自動資料型態的轉換，歸納出下列三種狀況：
  - (1)必須是數值（包含char型態）之間的轉換。
    - 如果是非數值的資料（如boolean或String型態）則不可能發生自動型態轉換。
  - (2)只會將佔用空間較小的型態轉換為佔用空間較大的型態。
    - 例如int可自動轉換為long，float可自動轉換為double。
    - 但int不會自動轉換為short或byte，double也不會自動轉換為float。
    - 並且當float自動轉換為double時，可能會出現些許誤差。
  - (3)浮點數和整數的轉換應該分開來看，以(2)為原則
    - 並且浮點數不會自動轉型為整數，但整數會自動轉型為浮點數。

91

3.4.1自動資料型態轉換

Coding 注意事項

1.整數常數若未特別註明，則資料型態為int。若直接設定給byte,short,char等型態的變數，則會先經過自動轉型後才設定給變數。但在呼叫函式傳遞引數時，則不會進行自動轉型。

2.由於char沒有負數，因此，byte或short雖然使用空間未大於char，但仍不會自動轉型。

92

3.4.2運算式的資料轉換

- 若某一個運算式中包含不同的資料型態，則編譯器會自動依照下列規則進行自動轉型。
  - (1)boolean型態不轉型。
  - (2)char型態轉型為short型態。
  - (3)若運算元包含short與int兩種型態，則全部會轉換為int型態。
  - (4)若運算元包含float浮點數與整數，則會全部轉換為float型態。
  - (5)若運算元包含double浮點數，則會全部轉換為double型態。
  - (6)佔用記憶體空間較少者會轉換為佔用記憶體空間較大的型態。
    - 我們以範例來加以說明：

93

3.4.2運算式的資料轉換

- 範例3-7：ch3_07.java（隨書光碟myJava\ch03\ch3_07.java）

```
1   /*  檔名:ch3_07.java       功能:運算式的自動轉型範例    */
2
3   package myJava.ch03;
4   import java.lang.*;
5
6   public class ch3_07       //主類別
7   {
8       public static void main(String args[])
9       {
10          short s=23;
11          int a=100;
12          char b='a';
13          float c=3.5f;
14          double d=4.8d;  //相當於 double d=4.8;
15
16          System.out.print("result = ");
17          System.out.println((s+a-b+c)+(c*d));
18      }
19  }
```

執行結果

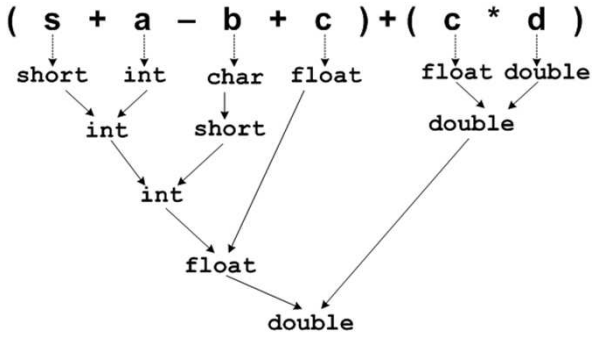
```
result = 46.3
```

會產生自動轉型。

94

3.4.2運算式的資料轉換

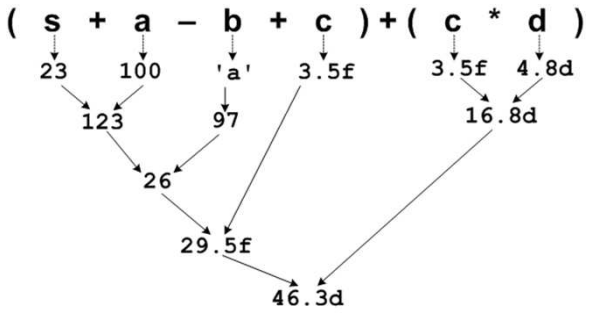
- 範例說明：
  - (1)第17行的運算式為(s+a-b+c)+(c*d)，因此會依照下圖步驟來轉換資料型態。



95

3.4.2運算式的資料轉換

- 範例說明：
  - (2)將上圖之變數以實際變數值代入後，則如下圖。



96

3.4.3強制資料型態轉換

- 自動資料型態轉換非常複雜，並且時常浮點數的自動轉型會出現精確度的問題
  - 例如您如果將範例3-7的第14行double d=4.8d修改為double d=4.82d，則執行結果為result = 46.370000000000005。
  - 為了讓程式設計師更能掌控資料型態轉換的問題，Java也提供使用者強制轉型的機制，格式如下。

```
(強制轉換型態) 運算式或變數;
```

97

3.4.3強制資料型態轉換

- **範例：**
  - 若x=15，y=4皆為int整數變數，z為float浮點數變數
    - 則運算式『z = x / y』
    - 會得到『z=3.0』
    - 因為x與y都是整數，而15/4=3.75，取整數為3，因此z=3
  - 但若是以強制型態轉換來重寫運算式
    - 『z = (float)x / (float)y』
    - 則運算前會先將x、y都轉換為浮點數
    - 而15.0/4.0=3.75，因此z=3.75。

98

3.4.4使用內建類別轉換資料型態

- Java是一個純物件導向的程式語言，對於Java而言，什麼都可以是物件
  - 因此，Java為八種原始資料型態也定義了對應的內建類別，如下表所列：

原始資料型態	對應的內建類別
byte	Byte
short	Short
int	**Integer**
long	Long
char	**Character**
boolean	Boolean
float	Float
double	Double

表3-6 原始資料型態對應之內建類別

99

3.4.4使用內建類別轉換資料型態

- 在這些對應類別中，有一個資料欄位（成員變數）的資料型態為所對應的原始資料型態
  - 例如Integer類別的物件將會有一個int型態的成員變數
    - 但由於封裝等級的緣故，所以我們不能直接以「物件.欄位」存取該變數
    - 而必須透過它的方法來運作這些變數。
- 字串與原始資料型態又該如何進行轉型呢？
  - 在這些內建類別中，有一些被宣告為static的類別方法(Class Method)，使得我們不用產生物件就可以直接利用該方法。
  - 而這些方法中，有一些可以作為字串與原始資料型態轉換之用，整理如下表：

100

3.4.4使用內建類別轉換資料型態

所屬類別	轉換可用之static方法（Class Method）	功能
java.lang.Byte	static byte parseByte(String s)	將字串轉為byte型態
java.lang.Byte	static String toString(byte b)	將byte型態轉為字串
java.lang.Short	static short parseShort(String s)	將字串轉為short型態
java.lang.Short	static String toString(short s)	將short型態轉為字串
java.lang.Integer	static int parseInt(String s)	將字串轉為int型態
java.lang.Integer	static String toString(int i)	將int型態轉為字串
java.lang.Long	static long parseLong(String s)	將字串轉為long型態
java.lang.Long	static String toString(long i)	將long型態轉為字串
java.lang.Float	static float parseFloat(String s)	將字串轉為float型態
java.lang.Float	static String toString(float f)	將float型態轉為字串

表3-7 原始資料型態與字串轉換的可用方法

101

3.4.4使用內建類別轉換資料型態

所屬類別	轉換可用之static方法（Class Method）	功能
java.lang.Double	static double parseDouble(String s)	將字串轉為double型態
java.lang.Double	static String toString(double d)	將double型態轉為字串
java.lang.Boolean	static boolean parseBoolean(String s)	將字串轉為boolean型態
java.lang.Boolean	static String toString(boolean b)	將boolean型態轉為字串
java.lang.Character	static String toString(char c)	將char型態轉為字串

表3-7 原始資料型態與字串轉換的可用方法

- 【註】：以上各方法的封裝等級為public。
- 【註】：boolean parseBoolean(String s)
  - 只有在s="true"時，才會回傳布林值true，否則全部都回傳布林值false。

102

3.4.4使用內建類別轉換資料型態

- 範例3-8：ch3_08.java（隨書光碟 myJava\ch03\ch3_08.java）

```
1  /*  檔名:ch3_08.java      功能:字串轉原始資料型態範例    */
2
3  package myJava.ch03;
4  import java.lang.*;
5  import java.util.Scanner;
6
7  public class ch3_08          //主類別
8  {
9      public static void main(String args[])
10     {
11           Scanner keyboardInput = new Scanner(System.in);
12           String str1;
13           double x;
14
15           System.out.print(" 欲求x 的3 次方, 請輸入x:");
16           str1 = keyboardInput.nextLine();
17           x = Double.parseDouble(str1);
18           System.out.println("x 的3 次方為:" + (x*x*x));
19     }
20 }
```

將字串轉型為double

103

3.4.4使用內建類別轉換資料型態

- 使用內建類別轉換資料型態
  - 執行結果：

```
欲求x的3次方,請輸入x:6.5
x的3次方為:274.625
```

  - 範例說明：
    - 由鍵盤輸入的資料經過nextLine方法讀取後，為字串str1的內容，因此要進行第18行的數值運算前，應該要先進行轉型，故在第17行透過parseDouble方法進行轉型
      - 由於該方法為static方法，故不需要產生物件實體，但執行時則必須指定它屬於哪一個類別。

104

3.4.4使用內建類別轉換資料型態

- String類別的valueOf方法
  - 除了上述的內建類別之外，在String字串類別中，也有利用多載(overload)技術宣告的static valueOf類別方法可以用來將原始資料型態轉換為字串。
  - 請見範例的說明：

105

3.4.4使用內建類別轉換資料型態

- 範例3-9：ch3_09.java（隨書光碟 myJava\ch03\ch3_09.java）

```
1   /*  檔名:ch3_09.java        功能:數值轉字串範例    */
2
3   package myJava.ch03;
4   import java.lang.*;
5
6   public class ch3_09          //主類別
7   {
8       public static void main(String args[])
9       {
10          short a=100;
11          int b=200;
12          float d=3.5f;
13          double e=4.8d;  //相當於 double d=4.8;
14          String s1,s2;
15
16          s1 = String.valueOf(a+b)+String.valueOf(a)+String.valueOf(b);
17          s2 = Float.toString(d)+Double.toString(e);
18          System.out.println("s1 = " + s1);
19          System.out.println("s2 = " + s2);
20      }
21  }
```

執行結果

```
s1 = 300100200
s2 = 3.54.8
```

valueOf()是多載的方法

toString必須指定正確的引數資料型態

106

3.4.4使用內建類別轉換資料型態

- 範例說明：
  - (1)第16行，使用了三次String類別的static方法valueOf將三個數值轉為字串，然後透過「+」進行字串的連結。所以s1="300100200"。
    - 由於valueOf使用了多載技術，使得輸入的引數可以為任何的原始資料型態，故我們不用關心a+b為何種資料型態
    - （事實上，根據之前的自動轉型規則所言，a+b的結果應該是int資料型態）。

107

3.4.4使用內建類別轉換資料型態

- (2)第17行使用的是Float與Double類別的toString方法進行轉型，因此您必須依照傳入引數的型別，決定正確的類別名稱，否則將出現錯誤。
  - 該行同樣可以使用 s2=String.valueOf(d)+String.valueOf(e);來取代。
  - 至於執行結果中，s2="3.54.8"，是因為兩個數值被轉型為字串後進行連結的緣故（"3.5"連結"4.8"）。
- (3)由這個例子可知，使用valueOf將數值轉為字串較為方便，因為不用理會原始數值屬於哪一種資料型態。

108

本章結束

Q&A討論時間

109