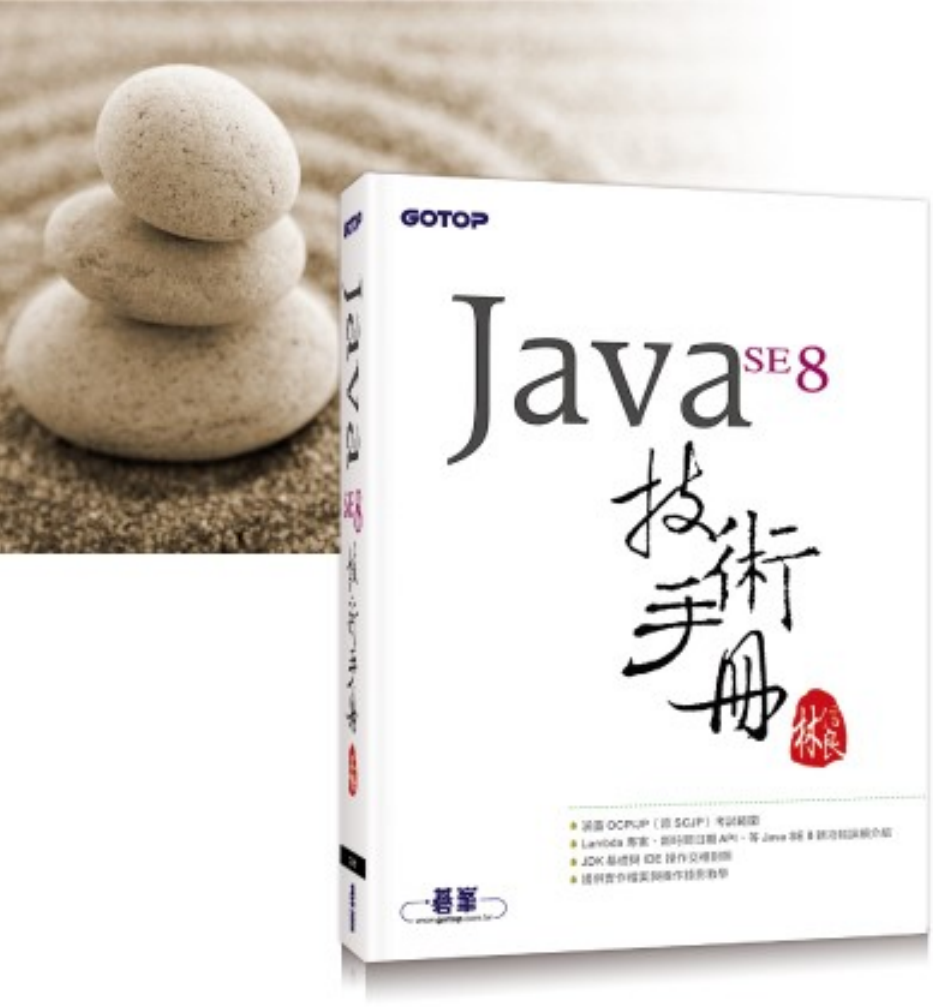


Java SE 8 技術手冊

林信良

- 涵蓋 OCP/JP（原 SCJP）考試範圍
- Lambda 專案、新時間日期 API、等 Java SE 8 新功能詳細介紹
- JDK 基礎與 IDE 操作交相對照
- 提供實作檔案與操作錄影教學

碁峰資訊

CHAPTER

基礎語法

學習目標

- 認識型態與變數
- 學習運算子基本使用
- 瞭解型態轉換細節
- 運用基本流程語法

型態

- Java 可區分為兩大型態系統：
  - 基本型態（Primitive type）
  - 類別型態（Class type）

型態

- 整數
  - 可細分為 **short** 整數（佔 2 個位元組）、 **int** 整數（佔 4 個位元組）與 **long** 整數（佔 8 個位元組）
- 位元組
  - **byte** 型態，長度就是一個位元組
- 浮點數
  - 可分為 **float** 浮點數（佔 4 個位元組）與 **double** 浮點數（佔 8 個位元組）

型態

- 字元
  - **char** 型態用來儲存‘ A’ 、‘ B’ 、‘林’等字元符號
  - 在 JDK8 中， Java 的字元採 Unicode 6.2.0 編碼
  - JVM 實作採 UTF-16 Big Endian，所以每個字元型態佔兩個位元組，中文字元與英文字元在 Java 中同樣都是用兩個位元組儲存
- 布林
  - **boolean** 型態可表示 **true** 與 **false**

型態

- 各種型態可儲存的數值範圍，可以透過 API 來得知：

```
// byte、short、int、long 範圍
System.out.printf("%d ~ %d%n",
        Byte.MIN_VALUE, Byte.MAX_VALUE);
System.out.printf("%d ~ %d%n",
        Short.MIN_VALUE, Short.MAX_VALUE);
System.out.printf("%d ~ %d%n",
        Integer.MIN_VALUE, Integer.MAX_VALUE);
System.out.printf("%d ~ %d%n",
        Long.MIN_VALUE, Long.MAX_VALUE);
// float、double 精度範圍
System.out.printf("%d ~ %d%n",
        Float.MIN_EXPONENT, Float.MAX_EXPONENT);
System.out.printf("%d ~ %d%n",
        Double.MIN_EXPONENT, Double.MAX_EXPONENT);
// char 可表示的 Unicode 範圍
System.out.printf("%h ~ %h%n",
        Character.MIN_VALUE, Character.MAX_VALUE);
// boolean 的兩個值
System.out.printf("%b ~ %b%n",
        Boolean.TRUE, Boolean.FALSE);
```

型態

- 單行註解 //
- 多行註冊 /**/

```
/* 作者：良葛格
   功能：示範 printf()方法
   日期：2011/7/23
 */
public class Demo {
...
```

型態

- 以下使用多行註解的方式是不對的：

```
/*   註解文字 1……bla…bla
     /*
          註解文字 2……bla…bla
     */
*/
```

- System.out.printf() 是什麼？

符號	說明
%%	因為%符號已經被用來作為控制符號前置，所以規定使用%%才能在字串中表示%。
%d	以 10 進位整數格式輸出，可用於 byte、short、int、long、Byte、Short、Integer、Long、BigInteger。
%f	以 10 進位浮點數格式輸出，可用於 float、double、Float、Double 或 BigDecimal。
%e, %E	以科學記號浮點數格式輸出，提供的數必須是 float、double、Float、Double 或 BigDecimal。%e 表示輸出格式遇到字母以小寫表示，如 2.13e+12，%E 表示遇到字母以大寫表示。
%o	以 8 進位整數格式輸出，可用於 byte、short、int、long、Byte、Short、Integer、Long、或 BigInteger。
%x, %X	以 16 進位整數格式輸出，可用於 byte、short、int、long、Byte、Short、Integer、Long、或 BigInteger。%x 表示字母輸出以小寫表示，%X 則以大寫表示。

%s, %S	字串格式符號。
%c, %C	以字元符號輸出，提供的數必須是 byte、short、char、Byte、Short、Character 或 Integer。%c 表示字母輸出以小寫表示，%C 則以大寫表示。
%b, %B	輸出 boolean 值，%b 表示輸出結果會是 true 或 false，%B 表示輸出結果會是 TRUE 或 FALSE。非 null 值輸出是 true 或 TRUE，null 值輸出是 false 或 FALSE。
%h, %H	使用 Integer.toHexString(arg.hashCode())來得到輸出結果，如果 arg 是 null，則輸出 null，也常用於想得到 16 進位格式輸出。
%n	輸出平台特定的換行符號，如果 Windows 下會置換為"\r\n"，如果是 Linux 下則會置換為'\n'，Mac OS 下會置換為'\r'。

```
System.out.printf("example:%.2f%n", 19.234);

System.out.printf("example:%6.2f%n", 19.234);
```

變數

- 想像一下程式中輸出 10 的部份很多，如果想要一次把它改為 20 …

```
System.out.println(10);
System.out.println(3.14);
System.out.println(10);
```

```
int number = 10;
double PI = 3.14;
System.out.println(number);
System.out.println(PI);
System.out.println(number);
```

變數

- 想要宣告何種型態的變數，就使用 `byte`、`short`、`int`、`long`、`float`、`double`、`char`、`boolean` 等關鍵字來宣告
- 變數在命名時有一些規則，它不可以使用數字作為開頭，也不可以使用一些特殊字元，像是 *、&、^、% 之類的字元
- 變數名稱不可以與 Java 的關鍵字（Keyword）同名，例如 `int`、`float`、`class` 等就不能用來作為變數
- 變數名稱也不可以與 Java 保留字（Reversed word）同名，例如 `goto` 就不能用來作為變數名稱

變數

- 在 Java 領域中的命名慣例（Naming convention），通常會以小寫字母開始，並在每個單字開始時第一個字母使用大寫，稱為駝峰式（Camel case）命名法

```
int ageOfStudent;
int ageOfTeacher;
```

變數

- 在 Java 中宣告一個區域變數，就會為變數配置記憶體空間，但不會給這塊空間預設值
- 不可以宣告區域變數後未指定任何值給它之前就使用變數

```
double score;
System.out.println(score);
```

variable score might not have been initialized

(Alt-Enter shows hints)

變數

- 如果在指定變數值之後，就不想再改變變數值，可以在宣告變數時加上 **`final`** 限定
- 如果後續撰寫程式時，自己或別人不經意想修改 `final` 變數，就會出現編譯錯誤

```
final double PI = 3.141596;
```

```
cannot assign a value to final variable PI
----
(Alt-Enter shows hints)
```

```
PI = 3.14;
```

變數

- 字面常數（Literal constant）

```
int number1 = 12;     // 10 進位表示
int number2 = 0xC;    // 16 進位表示，以 0x 開頭
int number3 = 014; // 8 進位表示，以 0 開頭

double number1 = 0.00123;
double number2 = 1.23e-3;

char size = 'S';
char lastName = '林';

char symbol = '\'';

boolean flag = true;
boolean condition = false;
```

變數

忽略符號	說明
\\	反斜線\。
\'	單引號'。
\"	雙引號"。
\uxxxx	以 16 進位數指定 Unicode 字元輸出，x 表示數字。
\xxx	以 8 進位數指定 Unicode 字元輸出，x 表示數字。
\b	倒退一個字元。
\f	換頁。
\n	換行。
\r	游標移至行首。
\t	跳格(按下 Tab 鍵的字元)。

```
System.out.println("\u0048\u0065\u006C\u006C\u006F");
```

變數

- Java SE 7 字面常量表示法

```
int number1 = 1234_5678;
double number2 = 3.141_592_653;

int mask = 0b101010101010;    // 用二進位表示 10 進位整數 2730

int mask = 0b1010_1010_1010;    // 用二進位表示 10 進位整數 2730
```

運算子

- 以下程式碼片段會在文字模式下顯示 7

```
System.out.println(1 + 2 * 3);
```

- 以下程式碼會顯示的是 6 ：

```
System.out.println(2 + 2 + 8 / 4);
```

- 以下程式碼顯示的是 3 ：

```
System.out.println((2 + 2 + 8) / 4);
```

運算子

- % 運算子計算的結果是除法後的餘數

```
int count = 0;
....
count = (count + 1) % 360;
```

運算子

- 比較運算子（Comparison operator）

```
System.out.printf("10 >  5 結果 %b%n", 10 > 5);
System.out.printf("10 >= 5 結果 %b%n", 10 >= 5);
System.out.printf("10 <  5 結果 %b%n", 10 < 5);
System.out.printf("10 <= 5 結果 %b%n", 10 <= 5);
System.out.printf("10 == 5 結果 %b%n", 10 == 5);
System.out.printf("10 != 5 結果 %b%n", 10 != 5);
```

運算子

- 條件運算子（Conditional operator）

```
System.out.printf("該生是否及格?%c%n", score >= 60 ? '是' : '否');

System.out.printf("是否為偶數?%c%n", (number % 2 == 0) ? '是' : '否');


if(number % 2 == 0) {
    System.out.println("是否為偶數?是");
}
else {
    System.out.println("是否為偶數?否");
}
```

運算子

- 邏輯運算

```
int number = 75;
System.out.println(number > 70 && number < 80);
System.out.println(number > 80 || number < 75);
System.out.println(!(number > 80 || number < 75));
```

運算子

- 捷徑運算（Short-Circuit Evaluation）
  - 因為 AND 只要其中一個為假，就可以判定結果為假，所以對 `&&` 來說，只要左運算元（Operand）評估為 `false`，就會直接傳回 `false`，不會再去運算右運算元
  - 因為 OR 只要其中一個為真，就可以判定結果為真，所以對 `||` 來說，只要左運算元評估為 `true`，就會直接傳回 `true`，就不會再去運算右運算元

運算子

- 除數為 0 會發生 `ArithmeticException`，代表除 0 的錯誤，以下運用 `&&` 捷徑運算避免了這個問題

```
if(b != 0 && a / b > 5) {
    // 作一些事...
}
```

運算子

- 位元運算

```
System.out.println("AND 運算：");
System.out.printf("0 AND 0 %5d%n", 0 & 1);
System.out.printf("0 AND 1 %5d%n", 0 & 1);
System.out.printf("1 AND 0 %5d%n", 1 & 0);
System.out.printf("1 AND 1 %5d%n", 1 & 1);

System.out.println("\nOR 運算：");
System.out.printf("0 OR 0 %6d%n", 0 | 0);
System.out.printf("0 OR 1 %6d%n", 0 | 1);
System.out.printf("1 OR 0 %6d%n", 1 | 0);
System.out.printf("1 OR 1 %6d%n", 1 | 1);

System.out.println("\nXOR 運算：");
System.out.printf("0 XOR 0 %5d%n", 0 ^ 0);
System.out.printf("0 XOR 1 %5d%n", 0 ^ 1);
System.out.printf("1 XOR 0 %5d%n", 1 ^ 0);
System.out.printf("1 XOR 1 %5d%n", 1 ^ 1);
```

運算子

- 補數運算是將所有位元 0 變 1， 1 變 0。例如 00000001 經補數運算就會變為 11111110

```
byte number = 0;
System.out.println(~number);
```

運算子

- 左移（<<）與右移（>>）

```
int number = 1;
System.out.printf( "2 的 0 次方: %d%n", number);

number = number << 1;
System.out.printf( "2 的 1 次方: %d%n", number);

number = number << 1;
System.out.printf( "2 的 2 次方: %d%n", number);

number = number << 1;
System.out.printf( "2 的 3 次方: %d%n", number);
```

運算子

- 遞增、遞減運算

```
int i = 0;
i = i + 1;
System.out.println(i);
i = i - 1;
System.out.println(i);
```

```
int i = 0;
i++;
System.out.println(i);
i--;
System.out.println(i);
```

```
int i = 0;
System.out.println(++i);
System.out.println(--i);
```

運算子

```
int i = 0;
int number = 0;
number = ++i;    // 結果相當於 i = i + 1; number = i;
System.out.println(number);
number = --i;    // 結果相當於 i = i - 1; number = i;
System.out.println(number);

int i = 0;
int number = 0;
number = i++;    // 相當於 number = i; i = i + 1;
System.out.println(number);
number = i--;    // 相當於 number = i; i = i - 1;
System.out.println(number);
```

運算子

- 指定運算

指定運算子	範例	結果
+=	a += b	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	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b

型態轉換

- 這個片段編譯時沒有問題…

```
double PI = 3.14;
```

- 如果你寫了個程式片段 …

```
float PI = 3.14;
```

possible loss of precision
 required: float
 found: double

(Alt-Enter shows hints)

型態轉換

- 在程式中寫下一個浮點數時，編譯器預設會使用 **double** 型態
- 編譯器會告知想將 double 長度的資料指定給 float 型態變數，會因為 8 個位元組資料要放到 4 個位元組空間，而遺失 4 個位元組的資料

型態轉換

- 兩種方式可以避免這個錯誤…

```
float PI = 3.14F;

float PI = (float) 3.14;
```

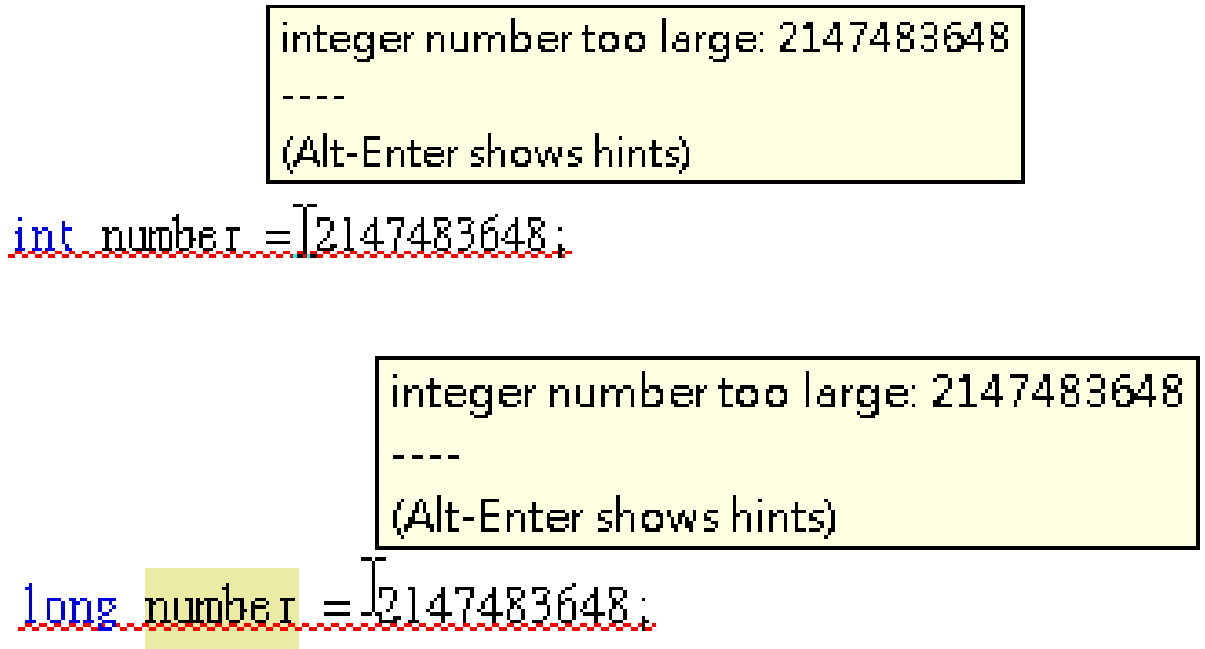
- 使用 (float) 語法告訴編譯器，你就是要將 double 型態的 3.14 指定給 float 變數，別再囉嗦了
- 後果自負…遺失精度而發生程式錯誤了，那絕不是編譯器的問題

型態轉換

- 這沒有問題 …

```
int number = 10;
```

- 但…



型態轉換

- 程式中寫下一個整數時，預設是使用不超過 `int` 型態長度
- 2147483648 超出了 `int` 型態的長度
- 直接告訴編譯器，用 `long` 來配置整數的長度，也就是在數字後加上個 **L**

```
long number = 2147483648L;
```

型態轉換

- 程式中寫下一個整數時，預設是使用不超過 `int` 型態長度

```
byte number = 10;
```

- 不過這樣不行：

```
byte number = 128;
```

型態轉換

- 如果運算式中包括不同型態數值，則運算時以長度最長的型態為主，其它數值自動提昇（Promote）型態

```
int a = 10;
double b = a * 3.14;
```

- a 是 int 型態，而寫下的 3.14 預設是 double，所以 a 的值被提至 double 空間進行運算

型態轉換

- 如果運算元都是不大於 **int** 的整數，則自動全部提昇為 **int** 型態進行運算

```
short a = 1;
short b = 2;
short c = a + b;
```

possible loss of precision
 required: short
 found: int

(Alt-Enter shows hints)

- int 的運算結果要放到 short，編譯器就又會囉嗦遺失精度的問題

型態轉換

- 你要告訴編譯器，就是要將 `int` 的運算結果丟到 `short` ，請它住嘴：

```
short a = 1;
short b = 2;
short c = (short) (a + b);
```

- 這次怎麼又遺失精度？

```
short a = 1;
long b = 2;
int c = a + b;
```

possible loss of precision
 required: int
 found: long

(Alt-Enter shows hints)

型態轉換

- b 是 long 型態，於是 a 也被提至 long 空間中作運算，long 的運算結果要放到 int 變數 c ，自然就會被編譯器囉嗦精度遺失了
- 如果這真的是你想要的，那就叫編譯器住嘴吧！

```
short a = 1;
long b = 2;
int c = (int) (a + b);
```

型態轉換

- 以下你覺得會顯示多少？

```
System.out.println(10 / 3);
```

- 答案是 3，而不是 3.333333....，因為 10 與 3 會在 int 長度的空間中作運算

```
System.out.println(10.0 / 3);
```

型態轉換

- 在玩弄語法？
- `count + 1 > Integer.MAX_VALUE` 永遠不會成立

```
int count = 0;
while(someCondition) {
    if(count + 1 > Integer.MAX_VALUE) {
        count = 0;
    }
    else {
        count++;
    }
    ...
}
```

流程控制

- **if..else** 條件式

```
int input = 10;
int remain = input % 2;
if(remain == 1) { // 餘數為 1 就是奇數
    System.out.printf("%d 為奇數%n", input);
}
else {
    System.out.printf("%d 為偶數%n", input);
}
```

流程控制

- 如果 `if` 或 `else` 中只有一行陳述句，則 `{` 與 `}` 可以省略
- 不過為了可讀性與可維護性而言，現在建議是就算只有一行陳述句，也要撰寫 `{` 與 `}` 明確定義範圍

流程控制

- 不過以下情況倒是省略 { 與 } 後比較有可讀性

```
if(條件式一) {
    ...
}
else {
    if(條件式二) {
        ...
    }
    else {
        ...
    }
}
```

```
if(條件式一) {
    ...
}
else
    if(條件式二) {
        ...
    }
    else {
        ...
    }
```

```
if(條件式一) {
    ...
}
else if(條件式二) {
    ...
}
else {
    ...
}
```

流程控制

```
int score = 88;
char level;
if(score >= 90) {
    level = 'A';
}
else if(score >= 80 && score < 90) {
    level = 'B';
}
else if(score >= 70 && score < 80) {
    level = 'C';
}
else if(score >= 60 && score < 70) {
    level = 'D';
}
else {
    level = 'E';
}
System.out.printf("得分等級：%c%n", level);
```

流程控制

- **switch** 條件式

```
int score = 88;
int quotient = score / 10;
char level;
switch(quotient) {
    case 10:
    case 9:
        level = 'A';
        break;
    case 8:
        level = 'B';
        break;
    case 7:
        level = 'C';
        break;
    case 6:
        level = 'D';
        break;
    default:
        level = 'E';
}
System.out.printf("得分等級：%c%n", level);
```

流程控制

- for 迴圈

```
for(int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

```
for(int j = 1; j < 10; j++) {
   for(int i = 2; i < 10; i++) {
       System.out.printf("%d*%d=%2d ",i, j,  i * j);
   }
   System.out.println();
}
```

流程控制

- **while** 迴圈

```
while(true) {
    int number = (int) (Math.random() * 10);
    System.out.println(number);
    if(number == 5) {
        System.out.println("I hit 5....Orz");
        break;
    }
}
```

❶ 直接執行迴圈

❷ 隨機產生 0 到 9 的數

❸ 如果遇到 5 就離開迴圈

流程控制

- **do..while** 迴圈

```
int number;
do {
    number = (int) (Math.random() * 10);
    System.out.println(number);
} while(number != 5);
System.out.println("I hit 5....Orz");
```

❶ 先隨機產生 0 到 9 的數

❷ 再判斷要不要重複執行

流程控制

- break 可以離開目前 switch、for、while、do..while 的區塊
- 使用於迴圈，break 會結束區塊執行，而 continue 只會略過之後陳述句，並回到迴圈區塊開頭準備下一次迴圈

```
for(int i = 1; i < 10; i++) {
    if(i == 5) {
        break;
    }
    System.out.printf("i = %d%n", i);
}
```

```
for(int i = 1; i < 10; i++) {
    if(i == 5) {
        continue;
    }
    System.out.printf("i = %d%n", i);
}
```