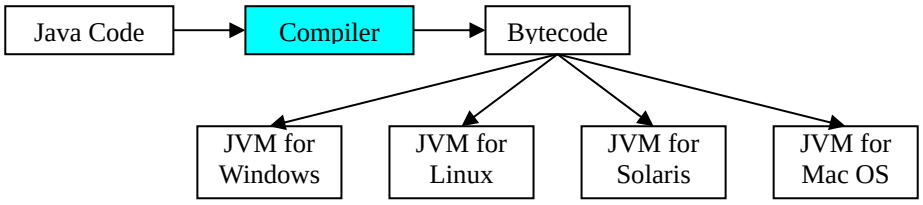
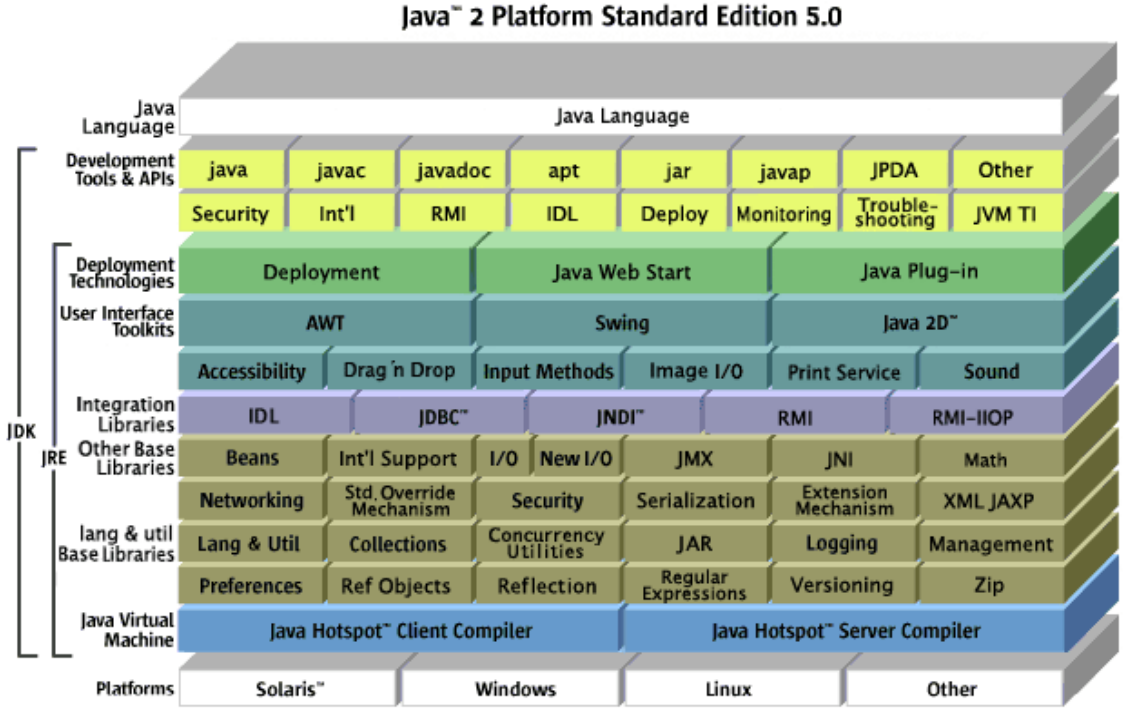
Table of Content

Java 平台簡介

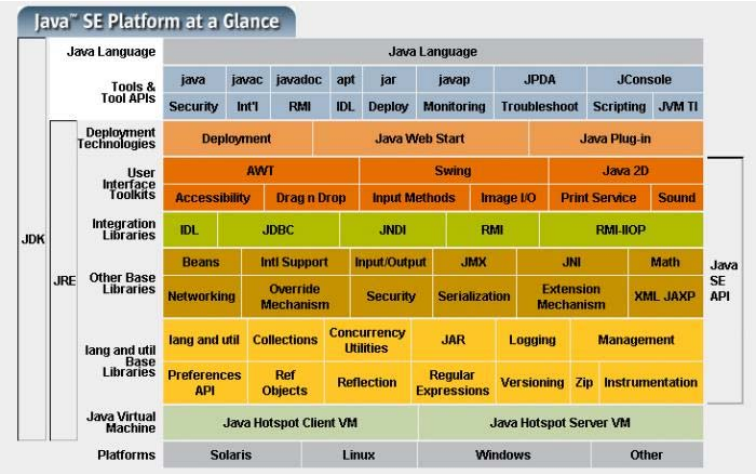
- Java Platform－Java 是一種結合編譯和直譯優點的高街物件導向程式語言，利用建立在各種作業系統(如 Windows, Mac OS, Solaris 等)上的軟體平台(Platform)結合硬體和軟體的執行環境，以期達到所謂”Write Once, Run Anywhere”的跨平台功能；主要包含 Java Virtual Machine 和 Java Application Programming Interface：
  - JVM－Java 編譯程式可以將 Java 原始程式編碼編譯成位元編碼(Bytecode)，為一種虛擬的機器語言，執行此語言的機器便是 JVM；亦即作業系統安裝了 JVM 的直譯程式便可直譯和執行位元碼，因此 Java 的程式碼只需撰寫一次，由安裝在不同系統的 JVM 來執行，達到跨平台的目的。



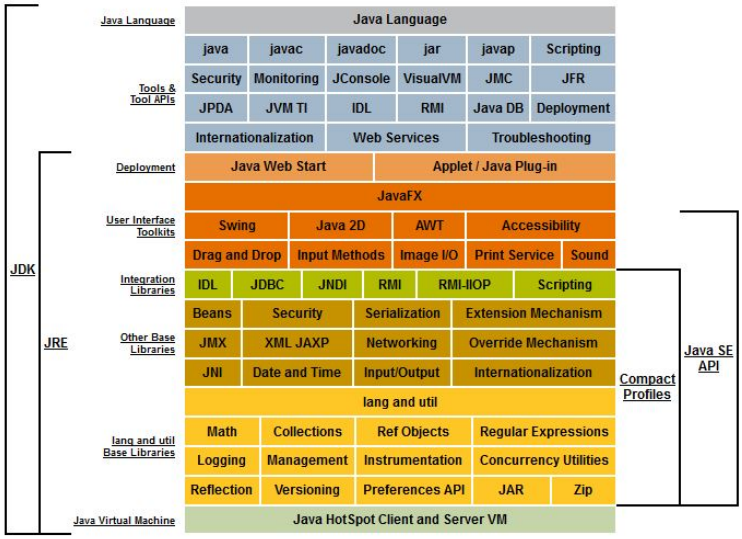
  - Java API－為 Java 軟體元件的集合，提供集合物件、GUI 圖形化介面元件、檔案處理、資料庫存取和網路介面連結等相關的類別和物件，稱之為套件或包裹(Package)，類似過去 C/C++所提供的函式庫；只是各家公司所開發的 C/C++的函式庫彼此多半無法相容，而 Java 則只有一種。
- Java 架構
  - Java 2 Platform Standard Edition 5.0



- Java Platform Standard Edition 6/7



- Java Platform Standard Edition 8



➢ Java 的發展工具－JDK，Java 的發展工具(Java Development Kit; JDK)組件的使用方法和範例包含如下：

命令	描 述
javac	這是一個編譯程式，用於將 Java 的原始程式編譯成為位元組碼的 .class 類別檔。(可用 **set classpath=%classpth%;new_path** 語法設定類別路徑環境，其中%classpath%也可以用一個句點 . 代替) 語法：javac [-classpath *pathes*][source code] 範例：javac Hello.java 或 javac -classpath *path1*;*path2* Hello.java 其中，path1, path2 代表 Hello.java 程式中有使用到的類別套件所在路徑
java	這是一個解譯程式，它最主要的目的是解譯 javac 所編譯後的位元組碼，亦即 .class 類別檔。(類別路徑亦可設進預設的環境變數中) 語法：java [-options] filename 範例：java Hello 或 java -cp %classpath%;*new_path* Hello 或 java -cp .;D: Hello

	(此例中假設 Hello.class 檔已編譯好存在於 D 碟某路徑下，但與預設之 java 執行環境不同，故須以句點.引入執行環境路徑，再加上目前類別檔所在位置方可執行)
appletviewer	appletviewer 的功能，在於能夠讓我們不透過瀏覽器執行 Java Applet 的程式。底下的範例是透過 HelloApplet.html 網頁帶出 Hello.class 程式碼，並將其結果顯示在網頁上。 範例：appletviewer HelloApplet.html

- 其他開發工具
  - NetBeans IDE, http://www.netbeans.org
  - IBM Eclipse, http://www.eclipse.org
  - Borland JBuilder, http://www.borland.com
  - Gel, http://www.gexperts.com/index.html
  - JCreator, http://www.jcreator.com

變數型態與運算子(Variables and Operators)

整數、浮點數、字元及布林變數

➢ **整數資料型態(Integral Types)**

Types	Bit	Range
byte	8	$-2^7 \sim 2^7-1$
short	16	$-2^{15} \sim 2^{15}-1$
int	32	$-2^{31} \sim 2^{31}-1$
long	64	$-2^{63} \sim 2^{63}-1$

- 整數文字值(Integral Literal)

表示法	十進位值	說明
44	44	十進位整數
0256	174	八進位整數
0xef	239	十六進位整數
0x3e6	998	十六進位整數
245L 或 245l	245	將文字值指定成長整數

➢ **浮點數資料型態(Floating Point Types)**

Type	Bit	Range
Float	32	1.40239846e-15 ~ 3.40282347e38
Double	64	4.9406545841246544e-324 ~ 1.79769313486231570e308

- 浮點數文字值(Integral Literal)

Type	Char	Example
Float	F/f	6.7F 或 6.7f

Double	D/d	31.415D 或 31.415d 或 3.1415e1 或 .00567 或 0.0056

- 要在兩種不相容的型態間產生轉換，必須利用強制型態轉換。強制型態轉換就是明確的型態轉換。它的一般格式如下：

  (target-type) value

  例如: b =(double) a ; //a 可能是一個整數或浮點數

- **字元資料型態(Character Type)**

Type	Symbol	Example
Char	''	char a = 'A' ; char b = 65 ; char c = '\u0020' ;
String	""	String a = "Java" ;

- Escape 逸出字元

Escape 字元	Unicode 碼	說明
\b	\u0008	Backspace 空白鍵
\f	\u000C	FF, Form Feed 換頁符號
\n	\u000A	LF, Line Feed 換行符號
\r	\u000D	CR, Enter 鍵
\t	\u0009	Tab, 定位鍵
\'	\u0027	', 單引號
\"	\u0022	", 雙引號
\\	\u005C	\, Backslash 反斜線

- **布林資料型態(Boolean Type)**－只有 true 和 false 兩種
- **常數(Constant)**－常數的定義表示該資料的值為一個固定值，不能再做任何改變，只要在上述的變數宣告前加上 final 並賦予資料即可，例如

  final double PI=3.14159 ;

  final String str = "Java" ;

- Example: 基本資料輸出

```
public class DataTypeTest
{   // 主程式
    public static void main(String[] args) {
        // 變數宣告
        int i = 44;
        int j = 0256;
        int k = 0xef;
        int l = 0x3e6;
        // 顯示結果
        System.out.print("44 = ");
        System.out.println(i);
        System.out.print("0256 = ");
        System.out.println(j);
```

```
        System.out.print("0xef = ");
        System.out.println(k);
        System.out.print("0x3e6 = ");
        System.out.println(l);
        // 變數宣告
        float ii = 25.0F;
        double jj = 0.0123;
        double kk = .00567;
        double ll = 1.25e4;
        // 顯示結果
        System.out.println(ii);
        System.out.println(jj);
        System.out.println(kk);
        System.out.println(ll);
        // 變數宣告
        char a = 'A';
        char b = 65;
        char c = '\u0020';
        String str = "Hello World!" ;
        // 顯示結果
        System.out.println(a);
        System.out.println(b);
        System.out.println(c+str);
        System.out.print("換行符號\n");
        System.out.println("\"Escape\"逸出字元");
    }
}
```

- Example: 利用 BufferedReader 物件進行資料輸入－必須先匯入 java.io 套件

```
import java.io.*;
// 主類別
public class BasicIOTest
{   // 主程式
    public static void main(String[] args) throws Exception {
        // 建立 BufferedReader 的輸入串流物件
        BufferedReader input = new BufferedReader(
                                new InputStreamReader(System.in));
        String str ;
```

```
        char c ;
        int i ;
        long l ;
        float f ;
        double d ;

        System.out.print("Please input a String: ");
        str = input.readLine();   // 讀取一列字串
        System.out.println("以 println 印出輸入的資料: " + str);
        System.out.print("Please input a Character: ");
        c = (char)input.read();   // 讀取一個位元組
        System.out.print("以 print 印出輸入的資料: " + c);
        str = input.readLine();    // 讀取緩衝字元
        System.out.print("  緩衝字元: " + str + "\n");
        System.out.println("Please input an Integer: ");
        str = input.readLine();
        i = Integer.parseInt(str);
        System.out.println("以 println 印出輸入的資料: " + i);
        System.out.print("Please input a Long Integer: ");
        str = input.readLine();
        l = Long.parseLong(str);
        System.out.println("以 println 印出輸入的資料: " + l);
        System.out.print("Please input a float: ");
        str = input.readLine();
        f = Float.parseFloat(str);
        System.out.println("以 println 印出輸入的資料: " + f);
        System.out.print("Please input a double: ");
        str = input.readLine();
        d = Double.parseDouble(str);
        System.out.println("以 println 印出輸入的資料: " + d);
    }
}
```

- Example: 利用 Scanner 物件進行資料輸入

```
import java.util.Scanner;
public class ScannerIOTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in) ;
        String str ;
```

```
        char c ;
        int i ;
        long l ;
        float f ;
        double d ;
        System.out.print("Please input a String: ");
        str = sc.next();   // 讀取一列字串
        System.out.println("以 println 印出輸入的資料: " + str);
        System.out.print("Please input a Character: ");
        c = sc.next().charAt(0);   // 讀取一個位元組
        str = sc.nextLine();      // 讀取緩衝字元
        System.out.print("以 print 印出輸入的資料: " + c);
        System.out.print("  緩衝字元: " + str + "\n");
        System.out.println("Please input an Integer: ");
        i = sc.nextInt();
        System.out.println("以 println 印出輸入的資料: " + i);
        System.out.print("Please input a Long Integer: ");
        l = sc.nextLong();
        System.out.println("以 println 印出輸入的資料: " + l);
        System.out.print("Please input a float: ");
        f = sc.nextFloat();
        System.out.println("以 println 印出輸入的資料: " + f);
        System.out.print("Please input a double: ");
        d = sc.nextDouble();
        System.out.println("以 println 印出輸入的資料: " + d);
    }
}
```

運算子(Operators)

- 算數運算子(Arithmetic Operator)－算數運算子的運算元必須是數字型態，但是可以將他們用在 char(字元)型態，因為對 Java 來說，char 型態基本上是 int 的子集合。
- 關係運算子(Relational Operator)－關聯式運算子決定一個運算元與另一個運算元的關係。尤其是，它們決定相等與順序。
- 布林運算子－布林邏輯運算子只用於 boolean 運算元。所有的二進位邏輯運算子結合兩個 boolean 值來形成結式的 boolean 值。
- 位元運算子(Shift and Bitwise Operator)－Java 定義許多與位元有關的運算子，可應用在整數型態，long、int、short、char 與 byte 上。這些運算子作用在運算元單一位元上。

算數運算子		關係運算子		位元運算子	
+	加法	==	等於	~	位元類 NOT
-	減法	!=	不等於	&	位元類 AND
*	乘法	>	大於	|	位元類 OR
/	除法	<	小於	^	位元類互斥 OR
%	餘數	>=	大於或等於	>>	右移
++	遞增	<=	小於或等於	>>>	右移補零
+=	加法指定	布林運算子		<<	左移
-=	減法指定	&	邏輯的 AND	&=	位元類 AND 指定
*=	乘法指定	|	邏輯的 OR	|=	位元類 OR 指定
/=	除法指定	^	邏輯的 XOR(反 OR)	^=	位元類互斥 OR 指定
%=	餘數指定	||	捷徑 OR	>>=	右移指定
--	遞減	&&	捷徑 AND	>>>=	右移補零指定
		!	邏輯的 NOT	<<=	左移指定

- Example

```
public class OperatorTest
{   // 主程式
    public static void main(String[] args) {
        // 變數宣告
        int x, y, f, c;
        int inc = 10;
        int dec = 10;
        String str1 = "Java 是";
        String str2 = "一種物件導向程式語言";
        System.out.println("負號運算: -7       = " + -7 );
        inc++;     // 遞增
        System.out.println("遞增運算: A++ = " + inc);
        dec--;     // 遞減
        System.out.println("遞減運算: A-- = " + dec);
        System.out.println("乘法運算: 5 * 6 = " + 5*6);
        System.out.println("除法運算: 7.0 / 2.0 = " + 7.0/2.0);
        System.out.println("餘數運算: 7 % 2 = " + 7%2);
        System.out.println("加法運算: 4 + 3 = " + (4+3));
        System.out.println("減法運算: 4 - 3 = " + (4-3));
        // 測試 Pre-/Post-  運算子
        x = 10;
        y = 10;
        System.out.println("x++ = " + x++ +":x = " + x );
```

```
        System.out.println("--y = " + --y +":y = " + y);
        System.out.println("字串連結: " + (str1 + str2));
        // 數學公式
        x = 6;
        y = 3;
        f = x*x-2*x+3;
        System.out.println("x*x-2*x+3 = " + f);
        f = (x+y)*(x+y)+5;
        System.out.println("(x+y)*(x+y)+5 = " + f);
        c = 80;
        System.out.print("攝氏: " + c + "度=  華氏: ");
        System.out.println((9.0 * c) / 5.0 + 32.0);
        // 變數宣告
        int a = 7;
        int b = 5;
        boolean blnA = a > b;
        boolean blnB = a == b;
        // 測試關係運算子
        System.out.println("小於:7<5 結果為  " + (a < b));
        System.out.println("大於:7>5 結果為  " + (a > b));
        System.out.println("小於等於:7<=5 結果為  " + (a <= b));
        System.out.println("大於等於:7>=5 結果為  " + (a >= b));
        System.out.println("等於:7==5 結果為  " + (a == b));
        System.out.println("不等於:7!=5 結果為  " + (a != b));
        // 測試條件運算子
        System.out.println("A 條件運算式: " + blnA);
        System.out.println("B 條件運算式: " + blnB);
        System.out.println("NOT 條件運算: !A  結果為  " + (!blnA));
        System.out.println("AND 條件運算: A && B  結果為  " + (blnA && blnB));
        System.out.println("OR 條件運算: A || B  結果為  " + (blnA || blnB));
        System.out.println("XOR 條件運算: A ^ B  結果為  " + (blnA ^ blnB));
        // 變數宣告
        int aa = 1;    // 0001
        int bb = 2;    // 0010
        int cc = 3;    // 0011
        int dd = 16; // 1000
        // 測試位元運算子
        System.out.println("NOT 運算: ~A = " + (~aa));
```

```
        System.out.println("左移運算: C << 2 = " + (cc << 2));
        System.out.println("右移運算: B >> 1 = " + (bb >> 1));
        System.out.println("無符號右移運算: D >>> 1 = " + (dd >>> 1));
        System.out.println("AND 運算: A & C = " + (aa & cc));
        System.out.println("XOR 運算: A ^ B = " + (aa ^ bb));
        System.out.println("OR 運算: A | B = " + (aa | bb));
    }
}
```

控制流程(Control Flow)

- Java 的流程控制結構(Control Structure)－在未導入物件導向程式(Object Oriented Programming)設計之前，傳統的程式設計概念以結構化程式設計(Structured Programming)為基礎，使用由上而下的設計(Top-down Design)來分析問題，將程式分解成階層架構(Hierarchical Architecture)的模組(Module)，每個模組擁有獨立功能的程式碼，只有單一的進入點和離開點，並用循序(Sequential)、選擇(Selection)及迭代(Iteration)等三種流程控制結構來整合。

條件控制敘述(Conditional Control)

- **if** 是否選擇條件敘述

```
if (conditional expression) {
    Statement ;
    ……
}
```

- **if...else** 二選一條件敘述

```
if (conditional expression) {
    Statement ;
    ……
} else {
    Statement ;
    ……
}
```

- **if...else if...else** 多選一條件敘述

```
if (conditional expression) {
    Statement ;
    :
} else if {
```

```
        Statement ;
        ……
} else {
        Statement ;
        ……
}
```

➢ **switch** 多選一條件敘述(下式中 case 後的 option 為單一字元或整數)

```
switch (option expression)
{
        case option1:
                Statement 1 ;
                break ;
        case option 2:
                Statement 2 ;
                break ;
         ……
        default:
                Statement ;
}
```

➢ **?...:**條件敘述運算子－分別以 “?”, “:” 代替 “if”, “else”

```
variable = (conditional express) ? variable 1 : variable 2 ;
```

➢ Example

```
public class ControlFlowTest
{   //  主程式
    public static void main(String[] args) {
            int score = 60;
            // if 條件敘述
            System.out.println("Score: " + score);
            if ( score >= 60 ) {
                    System.out.print("JavaSE Design");
                    System.out.println("Pass!:)");
            }
            score = 56;
            System.out.println("Score: " + score);
            if ( score < 60 )
                    System.out.println("Failed!:(");
            // if/else 條件敘述
            if ( score >= 60 ) {
```

```
            System.out.print("JavaSE Design");
            System.out.println("Pass!:)");
        } else {
            System.out.print("JavaSE Design");
            System.out.println("Failed!:(");
        }
        char grade = 'C';
        int age = 40;
        // if/else 多選一條件敘述
        System.out.println("Passenger Age: " + age);
        if ( age <= 18 )
            System.out.println("Student Rate: NT$12");
        else
            if ( age >= 65 )
                System.out.println("Senior Rate: NT$8");
            else
                System.out.println("Regular Rate: NT$15");
        // switch 多選一條件敘述
        System.out.println("Student GPA: " + grade);
        switch (grade) {
            case 'A':
                System.out.println("The Score Passing 80 points");
                break;
            case 'B':
                System.out.println("The Score Between 70~79 points");
                break;
            case 'C':
                System.out.println("The Score Between 60~69 points");
                break;
            default:
                System.out.println("The Score Under 60 points");
        }
        String str;
        int hour = 20;
        // ?:條件敘述運算子
        System.out.println("24-hour-Format Time: " + hour);
        str = (hour >= 12) ? " PM" : " AM";
        hour = (hour >= 12) ? hour-12 : hour;
```

```
            System.out.println("Current Time: " + hour + str);
        }
}
```

迴圈控制敘述(Loop Control)

- **for** 迴圈敘述－執行迴圈前後不檢查條件運算式，又稱計數迴圈(counting loop)

```
for (initialization ; ending condition ; iteration) {
        Statement ;
}
```

  - 一般(…)內之語句大致如下格式

    int i=0; i<10; i++

- **while** 迴圈敘述－執行迴圈前檢查條件運算式

```
while (condition) {
        Statement ;
}
```

- **do…while** 迴圈敘述－執行迴圈後檢查條件運算式

```
do {
        Statement ;
} while (condition) ;
```

- **break** 與 **continue** 指令－利用 break 強迫終止迴圈區塊內之敘述，利用 continue 繼續執行下一個迴圈區塊。
- 巢狀迴圈(Nested Loop)－迴圈之中含有迴圈。
- Example－測試迴圈指令(含 1.遞增及遞減 for 迴圈做數字加總、2.while 迴圈列出華式攝氏溫度換算、3.do…while 迴圈計算一條 100 單位的繩索可對折幾次、4.do…while 迴圈配合 break 計算階乘，以及 5.巢狀迴圈列出九九乘法表)

```
public class LoopTest
{   // 主程式
    public static void main(String[] args) {
// 測試 for 迴圈
            int total = 0;
            // 遞增 for 迴圈敘述
            for (int i = 1; i <= 10; i++ ) {
                System.out.print("Number: " + i + " ");
                total += i;
            }
            System.out.println("\nSummary from 1 to 10: " + total);
            System.out.println(" ---------------- ");
            total = 0;   // 重設總和變數
```

```
        // 遞減 for 迴圈敘述
        for (int i = 10; i >= 1; i-- ) {
            System.out.print("Number: " + i + " ");
            total += i;
        }
        System.out.println("\nSummary from 10 to 1: " + total);
// 溫度對照表
        double c = 30;
        double f;
        System.out.println("C        F");
        // while 迴圈敘述
        while ( c <= 100 ) {
            f = (9.0 * c) / 5.0 + 32.0;
            System.out.println(c + "       " + f);
            c += 10;
        }
// 繩索對折次數計算
        int count = 0; // 計算次數
        float len = 100.0f;
        // do/while 迴圈敘述
        do {
            System.out.println(count + " Length: " + len);
            count++;
            len /= 2.0;   // 對折繩索
        } while ( len > 20.0 );
        System.out.println("Folding Number: " + count);
        System.out.println("Final Length: " + len);
// 階乘計算
        int num = 0;
        float result = 1.0f;
        // do/while 迴圈敘述
        do {
            System.out.println("Number: " + num);
            if (num == 0) result = 1.0f ;
            else result *= num;
            num++;
            if (num > 5 ) break;     // 跳出迴圈
        } while ( true );
```

```
            System.out.println("5! = " + result);
// 九九乘法表
            // 顯示標題列
            System.out.print("       ");
            for (int i = 1; i <= 9; i++ )
                System.out.print(i + "          ");
            System.out.println();
            // 巢狀迴圈-第一層 while 迴圈
            int row = 0, col=0 ;
            while (row <= 9 ) {
                // 顯示欄標題
                System.out.print(row + " ");
                for (col = 1; col <= 9; col++ ) {
                    // 第二層 for 迴圈
                    System.out.print(row + "*" + col + "=");
                    System.out.print(row*col + " ");
                    if ( (row*col ) < 10 && col != 1 )
                        System.out.print(" ");// 調整顯示位置
                }
                row++; // 計數器變數加一
                System.out.println();
            }
        }
}
```

陣列(Array)

- **物件陣列(Object Array)**－陣列是一組相似型態的變數，以一個共同的名字來存取。任何型態的陣列皆可產生，同時也可以一個或以上的維度。
- 一維陣列：一維陣列是一串相似型態的變數。要產生陣列，首先必須產生一個型態的陣列變數。一般宣告一維陣列物件的格式如下：

  type [ ] array-name = { *constant set* } ; //直接賦予陣列值

  或

  type [ ] array-name = new type[size] ; //將 new 與一維陣列一起使用

  或

  type [ ] array-name ; //利用 new 動態配置陣列所需的記憶體

  (或 **type array-name[ ] ;**)

  array-name = new type[size] ;

  - 一維陣列實例

```
public class Array1DTest
{   // 主程式
    public static void main(String[] args) {
            // 建立 int 陣列
            int[] temp;   // 宣告陣列變數
            int[] tips = {150, 300, 500};
            // 建立 double 陣列
            double[] scores = new double[6];
            scores[0] = 66.5;   scores[1] = 78.9;
            scores[2] = 97.3;   scores[3] = 86.3;
            scores[4] = 45.8;   scores[5] = 64.5;
            // 使用迴圈顯示陣列值和計算總和
            int total = 0;
            for (int i=0; i < tips.length; i++) {
                total += tips[i];
                System.out.println(tips[i]);
            }
            System.out.println("小費總計: " + total);
            // 使用亂數產生整數亂數
            temp = tips;       // 複製陣列
            double seed;
            for(int i=0; i<temp.length; i++) {
                seed = Math.random() ; // 產生 0.0~1.0 之間的隨機亂數
                System.out.println("原始陣列值:"+temp[i] + "\t 亂數值:" + seed + "\t 乘積
取整數:"+(int)(temp[i]*seed)) ;
            }
            // 使用迴圈顯示陣列值和計算平均
            double sum = 0.0, average;
            for (int i=0; i < scores.length; i++) {
                sum += scores[i];
                System.out.println(scores[i]);
            }
            average = sum/(double)scores.length ;
            System.out.print("成績總和: " + sum);
            System.out.print("\t 平均成績: ");
            System.out.print(average + "\n");
            // String 物件陣列
            int n = 5 ; // 改變 n 的值可改變宣告之陣列大小
```

```
        String[] name = new String[n];
        name[0] = "王建民";
        name[1] = "郭泓志";
        name[2] = "鈴木一郎";
        name[3] = "Randy Johnson";
        name[4] = "曹錦輝";
        // 使用迴圈顯示陣列內容
        for (int i=0; i< n; i++) {
            System.out.print("\""+name[i]+"\"(字串長度:");
            System.out.println(name[i].length() + ")");
        }
    }
}
```

- Math 物件為數學函式庫，所提供的 random()方法可產生 0~1 之間的隨機亂數，其他如 abs(a)傳回 a 的絕對值，sin(a)、cos(a)、tan(a)可產生 a 的三角函數值，asin(a)、acos(a)、atan(a)計算 a 的反三角函數，pow(a,b)計算 a 的 b 次方值，sqrt(a)、cbrt(a)計算 a 的平方根及三次方根，exp(a)得到 a 的自然指數值，log(a)則取 a 的自然對數值等等，其他如四捨五入、取最大最小值、整數進位及各種基本數學運算的函數，詳見 JavaAPI 套件文件說明。
- (int)(a)會對 a 的數值進行強制轉型， a 可能是實數，轉型後只取整數值。

➢ 多維陣列實際上是陣列的陣列。要宣告一個多維陣列變數，必須利用另一組方括號來指定每個額外的索引。

type [ ] [ ] array-name = { {*constant set*}, { *constant set*}, ...} ;

或

type [ ][ ] array-name = new type[size_1][size_2] ;

或

type [ ] [ ] array-name = new type [size_1] [ ] ;

for (i = 0; i < array-name.length; i++) array-name[i] = new type [size_2]

- 二維陣列實例

```
public class Array2DTest
{   // 主程式
    public static void main(String[] args) {
        // 建立二維陣列
        String[][] username = new String[3][2];
        username[0][0] = "Joe";
        username[0][1] = "Manager";
        username[1][0] = "Jane";
        username[1][1] = "Engineer";
```

```
        username[2][0] = "Tom";
        username[2][1] = "Customer";
        // 使用巢狀迴圈顯示陣列值
        for (int j=0; j < username.length; j++) {
            for (int i=0; i < username[i].length; i++)
                System.out.print(username[j][i] + "\t");
            System.out.println();
        }
        // 動態配置陣列大小
        int m, n ;
        m = 9 ;
        int[][] c = new int[m][] ;
        n = 9 ;
        for(int i=0; i<c.length; i++)
            c[i] = new int[n] ;
        // 使用巢狀迴圈顯示九九乘法表
        for(int i=0; i<m; i++) {
            for(int j=0; j<n; j++) {
                c[i][j] = (i+1) * (j+1) ;
                System.out.print((i+1) + "*" + (j+1) + "=" + c[i][j] + "\t") ;
            }
            System.out.print("\n") ;
        }
        // 使用巢狀迴圈計算總和
        int total, sum;
        int[][] scores = {{54,68},{67,78},{89, 93}};
        total = 0;
        for (int j=0; j < scores.length; j++) {
            sum = 0;
            for (int i=0; i < scores[j].length; i++) {
                System.out.print(scores[j][i] + " ");
                sum += scores[j][i];
            }
            total += sum;
            System.out.println("成績小計: " + sum);
        }
        System.out.println("成績總和: " + total);
    }
```

```
}
```

- 動態陣列配置的目的在於可以隨時依照實際需求賦予陣列剛好夠用的大小，以節省記憶體。
- 宣告動態陣列的主要觀念在於陣列的位址的配置。當一個二維陣列被宣告如 c[][]時，代表該物件獲得了陣列的位址，但仍無實際大小，如同買到建地而尚未蓋起房子一樣。而 c[m][]被設定時，代表第一階的大小被設定，也取得了第二階的位址，像是第一層樓蓋好之後可以準備蓋第二層一樣。
- 配置第二階陣列大小時並不能像配置第一階大小的方式直接給值，而是要利用宣告的方式，針對每一個第一階元素分配其對應的下一階元素大小；即想像 c[m]是一個已被宣告的一階陣列物件，此時要賦予其大小。

➢ **foreach 迴圈** ： 利用 foreach 迴圈存取陣列資料

```
data_type [] array_name = { array_data } ;
for (data_type  variable  :  array_name ) {
    Statement with variable ;
}
```

- 使用 foreach 迴圈時要配合已宣告的陣列使用。執行迴圈時，陣列 array_name 的元素資料直接賦予給相同資料型態 data_type 的 variable 變數
- 迴圈中不會使用到迴圈計數子

```
public static void main(String[] args) {
   double[] test = {0,Math.PI/6,Math.PI/3,Math.PI/2,2*Math.PI/3,5*Math.PI/6,Math.PI};
   for(double rad: test) {   System.out.println("sin(" + rad + ") = " + Math.sin(rad));   }
}
```

重構(Refactor)

副程式的觀念

➢ 重構是將程式中具有特定功能的部分獨立出來成為類別方法或副程式，而主程式的部份則呼叫這些方法或副程式出來使用，其目的便是要讓程式更具結構化。

- 所謂副程式主要分為兩種，一種是宣告為 void 而沒有傳回值的函式(Subroutine)，一種則是宣告為變數型態必須有 return 指令傳回計算值的函數(Function)。這樣的概念在物件導向設計裡，則衍生為類別中的**方法(Method)**。
- 嚴格來講，主程式可定義為 Java 類別中一個名為 main 的函式，其傳入的引數為一個字串陣列。
- 傳入副程式的引數除非是傳入位址，例如陣列的傳址，否則傳入的值並不會隨著副程式的計算而改變。

➢ 茲以先前測試迴圈之 LoopTest 範例程式，分別將五個部分獨立出來改成副程式，並在主程式中呼叫使用。惟須注意的是，主程式的格是屬於一種靜態宣告的函式(static void)，由靜態宣告的函式所呼叫的函式，也必須宣告成靜態函式或函數，關於這些

觀念將於後續之物件導向設計中說明。

```
public class RefactorLoopTest {
    public static void main(String[] args) {
        forTest(10) ;                                   // 1.for 迴圈測試
        ConvertTemp(30, 100) ;                          // 2.溫度轉換表
        FoldRope(100.0f, 20.0f) ;                       // 3.繩索對折次數計算
        System.out.println("5! = " + nFactorial(5)) ;   // 4.階乘計算
        Table_99() ;                                    // 5.九九乘法表
    }
// 測試 for 迴圈
    public static void forTest(int bound) {
        int total = 0;
        // 遞增 for 迴圈敘述
        for (int i = 1; i <= bound; i++ ) {
            System.out.print("Number: " + i + " ");
            total += i;
        }
        System.out.println("\nSummary from 1 to 10: " + total);
        System.out.println(" ---------------- ");
        total = 0;   // 重設總和變數
        // 遞減 for 迴圈敘述
        for (int i = bound; i >= 1; i-- ) {
            System.out.print("Number: " + i + " ");
            total += i;
        }
        System.out.println("\nSummary from 10 to 1: " + total);
    }
// 溫度對照表
    public static void ConvertTemp(double cBottom, double cTop) {
        double c = cBottom;
        double f;
        System.out.println("C         F");
        while ( c <= cTop ) {// while 迴圈敘述
            f = CtoF(c);
            System.out.println(c + "       " + f);
            c += 10;
        }
    }
```

```
// 溫度轉換公式
    public static double CtoF(double c) {
        double f = (9.0 * c) / 5.0 + 32.0;
        return f ;
    }
// 繩索對折次數計算
    public static void FoldRope(float ropeLen, float sectLen){
        int count = 0; // 計算次數
        float len = ropeLen ;
        do {// do…while 迴圈敘述
            System.out.println(count + " Length: " + len);
            count++;
            len /= 2.0;   // 對折繩索
        } while ( len > sectLen );
        System.out.println("Folding Number: " + count);
        System.out.println("Final Length: " + len);
    }
// 階乘計算
    public static float nFactorial(int n) {
        int num = 0;
        float result = 1.0f;
        do {// do…while 迴圈敘述
            System.out.println("Number: " + num);
            if (num == 0)     result = 1.0f ;
            else       result *= num;
            num++;
            if (num > n ) break;     // 跳出迴圈
        } while ( true );
        return result ;
    }
// 九九乘法表
    public static void Table_99() {
        // 顯示標題列
        System.out.print("       ");
        for (int i = 1; i <= 9; i++ )
            System.out.print(i + "          ");
        System.out.println();
        // 巢狀迴圈-第一層 while 迴圈
```

```
            int row = 0, col=0 ;
            while (row <= 9 ) {
                // 顯示欄標題
                System.out.print(row + " ");
                for (col = 1; col <= 9; col++ ) {
                    // 第二層 for 迴圈
                    System.out.print(row + "*" + col + "=");
                    System.out.print(row*col + " ");
                    if ( (row*col ) < 10 && col != 1 )
                        System.out.print(" ");// 調整顯示位置
                }
                row++; // 計數器變數加一
                System.out.println();
            }
        }
}
```

- 重構後，將原程式分離出五組主要的副程式或函式，與主程式一齊被包含在整個類別之中。
- 第一組計算累加值，輸入累加次數。
- 第二組溫度轉換表又分出一個函數來計算溫度轉換公式，呼叫時輸入對照表的最低及最高攝氏溫度。
- 第三組計算階乘的函數，輸入階乘數直接傳回計算值。
- 第四組計算繩索對折次數，輸入繩索長度及對折完後之長度。
- 第五組則不必輸入任何引數，自動印出九九乘法表。

參數傳遞－傳值與傳址

- 在程式設計裡，變數的儲存是以位址的記錄被存放在記憶體進行運算。亦即每個變數具備了數值(Value)和位址(Address)兩個部份，當宣告了一個變數時，電腦便賦予一個獨立的位址，當變數要進行運算時，電腦就到記憶體中找到這個位址把其中的數值取出來計算。
- **傳值**－在程式裡要將各副程式呼叫出來使用，有些是需要用引數傳入提供運算的變數資料，此時就必須把合適的變數值傳送進去。對於一般非陣列物件型態的純量變數而言，如上例，這樣的傳遞是將一個“數值”傳入引數，這些引數相當於副程式中所宣告的變數，已經具有了不同的“位址”，故在副程式中的運算過程裡，代表放在不同位址的兩個變數，因此副程式外部的變數值並不會被影響。
- **傳址**－當引數是宣告成陣列型態時，以一維陣列 c 為例，如果引數的宣告是(int[] c)，呼叫副程式時傳入的變數是(d)，則代表讓副程式中的 c 陣列與副程式外的 d 陣列共用同一段位址，此時若副程式中的 c 陣列數直被改變，則意味著 d 陣列的值也跟著

改變。換言之，傳址時會把該段位址下存放的所有數值全部傳過去。

- 傳值或傳址參數

傳遞方式	說明
傳值呼叫 (Call by Value)	將變數值傳入方法，在方法需要另外配置記憶體空間來儲存參數值，因為擁有不同的記憶體空間，所以並不會更改呼叫變數的值。
傳址呼叫 (Call by Reference)	將變數實際儲存的記憶體空間位址傳入，所以在方法中更改參數值時也會同時變更原呼叫的變數值。

- Java 的參數依照不同的資料型態有不同的預設傳址方式

資料型態	方式	說明
int, char, double 等基本資料型態	傳值	基本資料型態的參數傳遞都是使用傳值方式
String 物件	傳值	不論是否使用 new 運算子建立的字串都是傳值，因為這種字串物件不能更改字串內容
StringBuffer 物件	傳址	StringBuffer 物件可以更改字串內容，所以參數傳遞是使用傳址方式
Array 陣列	傳址	Java 陣列是一種物件，其參數傳遞方式是傳址方式

- 下例中提供了以上幾種傳遞參數的方式

```
public class CallMethod
{
    public static void main(String[] args) {
        int c = 1;                      // 數字
        boolean b = true;               // 布林
        String s = "陳金鋒";            // 字串
        String s1 = new String("郭泓志"); // 字串物件
        // StringBuffer 物件
        StringBuffer s2 = new StringBuffer("道奇");
        int no[] = { 1, 2, 3 };   // 陣列
        System.out.println("methodA 前:" + c + "-" + b);
        // 呼叫類別方法
        methodA(c, b);
        System.out.println("methodA 後:" + c + "-" + b);
        System.out.print("methodB 前:");
        System.out.println( no[1] + "-" + s);
        // 呼叫類別方法
        methodB(no, s);
        System.out.print("methodB 後:");
        System.out.println(no[1] + "-" + s);
        System.out.println("mehtodC 前:"+ s1 + "-" + s2);
```

```
            // 呼叫類別方法
            methodC(s1, s2);
            System.out.println("methodC 後:"+ s1 + "-" + s2);
        }
    // 類別方法: int 和 boolean 型態參數為傳值
        public static void methodA(int c, boolean b) {
            c++;
            b = false;
            System.out.println("methodA 中:" + c + "-" + b);
        }
        // 類別方法: 陣列與字串物件參數為傳址
        public static void methodB(int temp[], String a) {
            temp[1] = 100;
            a = "鈴木一郎";
            System.out.print("methodB 中:");
            System.out.println(temp[1] + "-" + a);
        }
        // 類別方法: 字串物件傳值, StringBuffer 為傳址
        public static void methodC(String a, StringBuffer b) {
            a = "王建民";
            b.append("Yankee");
            System.out.println("methodC 中:" + a + "-" + b);
        }
}
```

- 參數傳遞前後值之列印結果

```
methodA 前:1-true
methodA 中:2-false
methodA 後:1-true
methodB 前:2-陳金鋒
methodB 中:100-鈴木一郎
methodB 後:100-陳金鋒
mehtodC 前:郭泓志-道奇
methodC 中:王建民-道奇 Yankee
methodC 後:郭泓志-道奇 Yankee
```

- 由傳遞的參數在進入副程式或類別方法的過程中可發現，如為傳值者，傳遞前後的值不因傳遞中的變化而改變；但如果是傳址的參數，傳入的過程等於共用相同的位址，因此傳遞前後的值會因此而改變。

➢ 陣列傳值－因陣列內存在著一個數值的集合，故利用傳址的方式傳遞整個集合。如

果要讓上述之 c 陣列中的某元素(c[3])取得一個外部外部陣列的傳入值，則引數的宣告必須是要求一個單一數值變數，而副程式外部傳入的變數如 d[1]，在副程式中讓 c[3]元素等於副程式外部的變數 d[1]元素值。

■ 以下範例測試陣列傳值與傳址的關係，用不同的方式處理傳入的陣列引數。

```
public class ArrayTest {
    public static void main(String[] args) {
        System.out.println("陣列傳值測試") ;
        int[] a = {1, 2, 3, 4, 5} ;
        int[] b = {10,20,30,40,50} ;
        ArrayTest(a, b, b[3]) ;
        for(int i=0; i<a.length; i++) {
            System.out.println("a["+i+"]="+a[i]+"\tb["+i+"]="+b[i]) ;
        }
    }
    public static void ArrayTest(int[] a, int[] b, int eValue) {
// 副程式中, d 的計算等同於 a 的計算, 但 c 與 b 是彼此獨立的
        int[] c = new int[a.length] ;    // 另外宣告新的物件, 分派新的位址
        int[] d = a ;                    // 直接和 a 共用相同的位址和數值
        int[] e = new int[1] ;
        for(int i=0; i<a.length; i++) {
            c[i] = b[i] ;                // 只有複製 b 的值
            c[i] = c[i] + a[i] ;
            d[i] = d[i] * 2 ;
        }
        e[0] = eValue ;
        e[0] = e[0] * 2 ;
        eValue = eValue * eValue ;
    }
}
```

■ 上例中的執行結果如下，主程式中 a 陣列的值改變了，但 b 陣列則不變

```
陣列傳值測試
e[0]=80  eValue=1600<=In Subprogram
a[0]=2   b[0]=10
a[1]=4   b[1]=20
a[2]=6   b[2]=30
a[3]=8   b[3]=40
a[4]=10  b[4]=50
```

檔案處理概念－File I/O

匯入套件概念

- Java 是一種以物件導向為基礎而開發出來的語言，前述之基本語法是由一個標準套件 java.lang 所提供，此套件在編譯程式時為預設值，不需要在程式中另行設定。但未了處理其他各種不同性質的問題，此預設套件並不敷使用，因此提開發了各種套件來解決，這些套件包含了許多的類別與使用介面，每一種類別均提供了許多的方法，就像是前述範例中利用 Math 提供的 sqrt()方法計算平方根一樣。
- Java 在處理檔案的問題上，則提供了 File 類別物件，而處理檔案資料時又必須考慮到傳送資料的串流，這些相關的類別均由 java.io 套件所提供，因此要使用這些類別時，在程式的前端要匯入套件名稱，即加上
  import java.io.* ;
  編譯時程式才知道要去哪裡找出這些類別。

串流基礎

- 串流(stream)觀念最早使用在 UNIX 作業系統，串流模型如同水管的水流，並沒有考慮資料來源、型態等，當程式開啟一個來源的輸入串流(例如檔案、記憶體和緩衝區等)為循序存取串流(Sequential Access Streams)，如水流般依序讀取和寫入資料。
- Java 提供了一套用來處理串流資訊的函式套件，Java I/O 套件，全名是 Java Input/Output(輸入/輸出)，即應用程式的資料輸入與輸出，在 Java 類別函式庫(Class Library)是使用串流模型來處理資料的輸入與輸出。
- Java 的 java.io 套件提供多種串流類別，基本上 Java 串流類別分成兩大類：字元串流(CharacterStream)和位元組串流(ByteStream)，各分為輸入/輸出 2 種串流類別。
  - 字元串流(CharacterStream)－字元串流是是一種適合人類閱讀(Human-readable)的串流，Reader/Writer 兩個類別分別讀取和寫入 16 位元的字元資料，屬於字元串流的父抽象類別。在 java.io 套件提供多種繼承自 Reader/Writer 的子類別，並包含各種情況的串流資料處理，其名稱都是使用 Reader 和 Writer 結尾，例如:
    - BufferReader/BufferWriter：處理緩衝區 I/O。
    - InputStreamReader/OutputStreamWriter：InputStreamReader 在讀取位元組資料後將它轉成字元資料，OuputStreamWriter 是將字元轉換成位元組資料。
    - FileReader/FileWriter：處理檔案的 I/O。
  - 位元組串流(ByteStream)－位元組串流是一種電腦格式(Machine-formatted)串流，可以讀取和寫入 8 位元的位元組資料，也就是處理二進位資料的執行檔、圖檔和聲音等，其父抽象類別的輸入/輸出串流名稱為 InputStream/OutputStream 類別。java.io 套件擁有多種繼承自 InputStream/OutputStream 的子類別，其名稱都是使用 InputStream/OutputStream 結尾，例如:

- FileInputStream/FileOutputStream：處理檔案的 I/O。
- DataInputStream/DataOutputStream：讀取和寫入基本資料型態的資料。
- BufferInputStream/BufferedOutputStream：處理緩衝區 I/O。

檔案(File)與資料夾目錄(Directory)處理

- File 物件－java.io 套件提供 File 類別(java.io.File)建立物件，取得檔案或資料夾的相關資訊，例如取得檔名、檔案大小、檔案路徑、檔案屬性或目錄下之檔案列表等，基本的宣告指令範例如下

```
File f = new File(String str) ;
```

  - 其中 String 表示建構子中應輸入字串參數，為檔案或資料夾目錄路徑。
  - 要使用這個類別時，程式檔的最前面必須執行匯入 java.io 的動作，如同要在螢幕狀態輸入串流資料使用 BufferReader 一樣。
- 如同宣告陣列後可以利用 length 方法得到陣列大小一樣，宣告成 File 的物件變數也擁有許多的方法可以對輸入參數 str 所代表的檔案或目錄資料進行處理。詳細資料可參考 Java 的 API 文件，基本常用到的方法如：
  - 顯示資料夾資訊

boolean	isDirectory()－檢視參數 str 所代表的資訊是否為一個目錄
String[]	list()－如果參數 str 為一個目錄，將目錄下所有的檔名存在一個字串陣列裡
File[]	listFiles()－將目錄下所有的檔案存在一個檔案陣列裡，即陣列裡的每一個元素均記錄著檔案相關資訊

  - 顯示檔案資訊的方法如下

boolean	exists()－判斷檔案或資料夾是否存在
boolean	isFile()－判斷是否是一個檔案
String	getAbsolutePath()－取得絕對路徑
String	getName()－取得檔名字串
String	getPath()－取得所在位置相對路徑
long	length()－傳回檔案大小
boolean	canRead()－判斷檔案是否可被讀取
boolean	canWrite()－判斷檔案是否可以覆寫
long	lastModified()－傳回檔案最後一次修改的時間
boolean	setReadOnly()－設定檔案為唯讀

  - 更改檔案或資料夾名稱的方法如下

boolean	renameTo(File dest)－更換檔名為 dest

  - 刪除檔案的方法如下

boolean	delete()－刪除檔案

  - 建立資料夾目錄的方法如下

boolean	mkdir()－建立新資料夾

- Example: 檢查檔案目錄資訊範例

```
import java.io.File;
import java.util.Scanner;
public class FileTest {
      public static void main(String[] args) {
            Scanner sc = new Scanner(System.in) ;
            String[] filenames; // 宣告字串陣列
            System.out.println("1. 檔案目錄判斷測試") ;
            System.out.println("請輸入檔名或目錄名稱") ;
            File file = new File(sc.next());   // 宣告 File 物件
// 判斷是否是目錄
            if ( file.isDirectory() ) {
                  filenames = file.list();   // 取得檔案和目錄清單
                  for ( int i = 0; i < filenames.length; i++ ) {// 使用迴圈顯示清單內容
                        System.out.println(filenames[i]);
                  }
            } else {
                  System.out.println("[" + file + "]不是目錄!");
            }
// 判斷檔案是否存在
            System.out.println("2. 檔案目錄存在判斷") ;
            if ( file.exists() ) {
                  if ( file.isFile() )
                        System.out.println("["+file+"]是檔案");
                  else if ( file.isDirectory() )
                        System.out.println("["+file+"]是目錄");
                  // 顯示檔案資訊
                  System.out.println("絕對路徑 :" +
                              file.getAbsolutePath());
                  System.out.println("絕對的檔案路徑 :" +
                              file.getAbsoluteFile());
                  System.out.println("名稱:"+file.getName());
                  System.out.println("上層路徑:"+file.getParent());
                  System.out.println("檔案路徑:"+file.getPath());
                  System.out.println("尺寸:"+file.length());
                  System.out.println("可讀:"+file.canRead());
                  System.out.println("可寫:"+file.canWrite());
            } else {
                  System.out.println("[" + file + "]不存在!");
```

```
            }
// 更改檔案名稱
            System.out.println("3. 檔案名稱更改測試") ;
            System.out.println("請輸入原始檔名") ;
            File fs = new File(sc.next());
            System.out.println("請輸入欲更改之檔名") ;
            File fd = new File(sc.next());
            if ( fs.exists() ) {       // 檢查來源是否存在
                System.out.println("舊檔名: "+fs) ;
                if ( !fd.exists() ) { // 檢查目的是否不存在
                    boolean success=fs.renameTo(fd); // 更改名稱
                    System.out.println("更改名稱成功!"+success+" 新檔名:"+fd);
                } else {
                    System.out.println("[" + fd + "]已經存在!");
                }
            } else {
                System.out.println("[" + fs + "]不存在!");
            }
// 刪除檔案
            System.out.println("4. 檔案刪除測試") ;
            System.out.println("請輸入欲刪除之檔名") ;
            File kill = new File(sc.next());
            if ( kill.exists() ) {
                System.out.println("刪除檔案..." + kill);
                boolean success = kill.delete();   // 刪除檔案
                System.out.println("刪除檔案成功" + success);
            } else {
                System.out.println("[" + kill+ "]檔案不存在!");
            }
// 建立目錄
            System.out.println("5. 建立目錄測試") ;
            System.out.println("請輸入欲建立之目錄") ;
            File dir = new File(sc.next());
            if ( !dir.exists() ) { // 目錄是否存在
                boolean success = dir.mkdir(); // 建立目錄
                System.out.println("目錄["+dir+"]建立"+success);
            } else {
                System.out.println("[" + dir + "]目錄存在!");
```

```
            }
        }
}
```

執行情形

```
D:\Course_Data\Example>java -cp .;D: FileTest
1. 檔案目錄判斷測試
請輸入檔名或目錄名稱
FileTest.txt
[PackageTest.txt]不是目錄!
2. 檔案目錄存在判斷
[FileTest.txt]是檔案
絕對路徑 : D:\Course_Data\Example\FileTest.txt
絕對的檔案路徑 : D:\Course_Data\Example\FileTest.txt
名稱:FileTest.txt
上層路徑:null
檔案路徑:FileTest.txt
尺寸:267
可讀:true
可寫:true
3. 檔案名稱更改測試
請輸入原始檔名
FileTest.txt
請輸入欲更改之檔名
FileTest.bak
舊檔名: FileTest.txt
更改名稱成功!true  新檔名:FileTest.bak
4. 檔案刪除測試
請輸入欲刪除之檔名
FileTest.bak
刪除檔案...FileTest.bak
刪除檔案成功 true
5. 建立目錄測試
請輸入欲建立之目錄
FileTest
目錄[FileTest]建立 true
```

一般文字檔案串流處理

- Java 在 Java.io 函式庫套件中提供了 BufferReader、BufferWriter、FileReader、

FileWriter 等函式來處理文字檔案的串流，故執行前要先匯入以上套件，且必須在使用這些函式的程式類別後面加上 **throws Exception** 指令處理有發生資料存取例外之情形。

- 寫入文字檔案－寫入檔案之方式是先開啟檔案的 FileWriter 串流，然後使用緩衝器 BufferedWriter 加速處理，主要方式如下

```
BufferedWriter output = new BufferedWriter(new FileWriter(file)) ;
```

  - 其中 *file* 是檔案名稱(含所在之路徑)，接著就可寫出串流的方式以 output.write 方法將字串寫入。

- 讀取文字檔案－讀取檔案之方式是先開啟檔案的 FileReader 串流，然後使用緩衝器 BufferedReader 加速處理，主要方式如下

```
BufferedReader input = new BufferedReader(new FileReader(file)) ;
```

  - 其中 *file* 是 File 物件之檔案，接著就可使用 **while** 迴圈配合 input.readLine()方法讀取檔案內容，並檢查讀取字串是否為 null 以判斷是否讀到檔尾。如

```
while ( (str = input.readLine()) != null) { ……}
```

- 檔案複製－結合 FileReader 及 FileWriter 的用法分別開啟來源和目的的檔案串流，即可複製檔案，主要的複製迴圈如下

```
while ( (ch = input.read()) != -1)
    output.write(ch) ;
```

  - 使用 read()方法讀取整數，然後使用 write()方法寫入目的檔案。

- Example:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Scanner;
public class FileIOTest {
    public static void main(String[] args) throws Exception {
// 串流寫入測試
        Scanner sc = new Scanner(System.in) ;
        System.out.println("串流寫入測試\n 請輸入測試文字檔名[例
如:IOStreamTest.txt]") ;
        String file = sc.next() ;
        String str1 = "串流輸出輸入測試\n";
        String str2 = "Stream I/O Test!\n";
        // 建立 BufferedWriter 的輸出串流物件
        BufferedWriter output = new BufferedWriter(new FileWriter(file));
        System.out.println("正在寫入檔案..." + file);
```

```
            output.write(str1);         // 寫入字串
            output.write(str2);         // 寫入字串
            output.close();             // 關閉串流
            System.out.println("寫入檔案成功..." + file);
// 串流讀取測試
            System.out.println("串流讀取測試\n 讀取文字檔"+file+"內容") ;
            File name = new File(file);   // 建立 File 物件
            if ( name.exists() ) {   // 建立 BufferedReader 的輸入串流物件
                BufferedReader input = new BufferedReader(new FileReader(name));
                String str;
                // 讀取資料
                while ( (str = input.readLine()) != null ) {
                    System.out.println(str);
                }
                input.close();          // 關閉串流
            } else {
                System.out.println("檔案[" + name + "不存在!");
            }
// 檔案複製測試
            System.out.println("文字檔案複製測試") ;
            System.out.println("請輸入欲複製檔名[例如:"+file) ;
            String sour_path = sc.next();
            System.out.println("請輸入複製後檔名[例如:OutputFile.txt]") ;
            String dest_path = sc.next();
            File sour = new File(sour_path);   // 建立 File 物件
            File dest = new File(dest_path);
            if ( sour.exists() ) {   // 建立 BufferedReader 的輸入串流物件
                BufferedReader input = new BufferedReader(new FileReader(sour));
                // 建立 BufferedWriter 的輸出串流物件
                BufferedWriter duplicate = new BufferedWriter(new FileWriter(dest));
                int ch;
                System.out.println("正在複製檔案..." + dest);
                // 複製檔案內容
                while ( (ch = input.read()) != -1 ) {
                    duplicate.write(ch);
                }
                input.close();       // 關閉串流
                duplicate.close();
```

```
                System.out.println("複製檔案成功..." + dest);
            } else {
                System.out.println("來源檔案["+sour+"不存在!");
            }
        }
}
```

執行情形

```
D:\Course_Data\Example>java -cp .;M: FileIOTest
串流寫入測試
請輸入測試文字檔名[例如:IOStreamTest.txt]
IOStreamTest.txt
正在寫入檔案...IOStreamTest.txt
寫入檔案成功...IOStreamTest.txt
串流讀取測試
讀取文字檔 IOStreamTest.txt 內容
串流輸出輸入測試
Stream I/O Test!
文字檔案複製測試
請輸入複製後檔名[例如: IOStreamTest.txt]
IOStreamTest.txt
請輸入複製後檔名[例如:OutputFile.txt]
OutputFile.txt
正在複製檔案...OutputFile.txt
複製檔案成功...OutputFile.txt

D:\Course_Data\Example>type OutputFile.txt
串流輸出輸入測試
Stream I/O Test!
```

二進位檔案串流處理

- 對於非文字的檔案存取，java.io 套件的函式庫提供了處理位元組檔案串流的類別，FileInputStream 與 FileOutputStream，對應 Reader/Writer 類別方法來處理讀取和寫入位元組或位元組陣列(參考 java.io.InputStream 及 java.io.OutputStream)，將檔案內容以二進位元逐一進行存取。
  - 檔案處理方式如下

```
FileOutputStream output = new FileOutputStream(fileString) ;
FileInputStream input = new FileInputStream(nameFile) ;
```

  - 上述兩列程式碼分別建立輸出和輸入串流，其來源和目的都是檔案，參數可以

是 **fileString** 的檔案路徑字串或 **nameFile** 的 File 物件。

- Example－複製二進位串流檔案

```
import java.io.*;
public class FileStreamIOTest {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in) ;
        System.out.println("請輸入欲讀取之二進位串流檔案") ;
        String srcName = sc.next();
        System.out.println("請輸入欲寫入之二進位串流檔案") ;
        String destName = sc.next();
        // 建立 FileInputStream 的輸入串流物件
        FileInputStream input = new FileInputStream(srcName);
        // 建立 FileOutputStream 的輸出串流物件
        FileOutputStream output = new FileOutputStream(destName);
        int ch;
        while ( (ch = input.read()) != -1 ) {// 讀取字元
            output.write((char) ch);      // 寫入字元
        }
        input.close();          // 關閉輸入串流
        output.close();         // 關閉輸出串流
        // 建立檔案物件
        File src = new File(srcName);
        System.out.println("原始檔案路徑: " + src.getAbsolutePath());
        File dest = new File(destName);
        System.out.println("目的檔案路徑: " + dest.getAbsolutePath());
    }
}
```

執行過程－複製一個 Excel 檔

```
請輸入欲讀取之二進位串流檔案
CLA_sal.xls
請輸入欲寫入之二進位串流檔案
CLA_sal1.xls
原始檔案路徑: D:\Course_Data\Example\CLA_sal.xls
目的檔案路徑: D:\Course_Data\Example\CLA_sal1.xls
```

例外處理－Exception

- 例外處理的目的在於避免程式執行時因不正常的資料存取而造成程式無法繼續運

作，例如在處理檔案串流時必須在主程式宣告列加上 Exception 指令，因為處理的串流的類別被設定成會丟出例外訊息。

- 在 Java 產生的例外物件屬於 Throwable 類別或其子類別的實例，Throwable 類別擁有 2 個直接繼承的子類別，
  - Error 類別－其子類別屬於 JVM 的嚴重錯誤，這種錯誤會導致程式終止執行，所以並沒有辦法使用例外處理來處理這種錯誤。
  - Exception 類別－其子類別是各種例外物件，也是例外處理可以處理的部分，事實上，部分例外物件也是一種錯誤，只是錯誤沒有嚴重到需要終止程式執行，例外處理就是在防止程式終止執行，並且作一些補救操作。

例外處理敘述(try … catch … finally)

- Java 語言例外處理程式敘述分為 try、catch、finally 三個程式區塊，可以處理特定的例外物件：

```
try {
      ……
}
catch (XXXException e) {
      // Exception
      ……
}
 :    // Other catches
finally {
      ……
}
```

  - try 程式區塊－在 try 區塊的程式碼檢查是否產生例外，當例外產生時，就會丟出指定例外類型的物件。
  - catch 程式區塊－在 try 程式區塊的程式碼如果丟出例外，Java 程式需要準備一到多個 catch 程式區塊處理不同類型的例外，傳入參數 e 是例外類型的物件(繼承自 Throwable 類別)，可以取得例外的相關資訊，其相關方法如下表所示:
    String getMessage() – 傳回例外說明字串
    void printStackTrace() – 顯示程式呼叫的執行過程
  - finally 程式區塊－finally 程式區塊可有可無，主要是用來執行程式善後，不論例外是否產生，都會執行此區塊的程式碼。如果沒有定義 finally 區塊，程式在處理完 catch 到的例外後仍會繼續執行。

丟出例外敘述(throw)

- Java 程式碼中可以使用 throw 指令丟出例外

```
throw ThrowableObject;
```

- 例如

```
throw new ArithmeticException(“The value is 0”) ;
```

➢ Example:

```
public class Ex01_throwException
{
    public static void main(String[] args) {
        try {// 例外處理程式敘述
            String str = args[0];   // 產生超過陣列範圍例外
            // 產生除以零的例外
            for (int i = 2; i > -1; i-- )
                System.out.println("計算結果: " + 6/i);
        } catch( ArithmeticException e ) {   // 處理除以零的例外
            System.out.println("例外說明: "+e.getMessage());
            System.out.print("例外原因: ");
            e.printStackTrace();
        } catch( ArrayIndexOutOfBoundsException e ) {   // 處理超過陣列範圍例外
            System.out.println("例外說明: "+e.getMessage());
            System.out.print("例外原因: ");
            e.printStackTrace();
        }
        System.out.println("未執行 finally,程式繼續執行...") ;
        double result;
        Scanner sc = new Scanner(System.in) ;
        System.out.println("請輸入三個數值計算 a*b/c") ;
        try {   // 取得參數值
            double a = Double.parseDouble(sc.next());
            double b = Double.parseDouble(sc.next());
            double c = Double.parseDouble(sc.next());
            result = cal(a, b, c);   // 呼叫方法
            System.out.println("計算結果: " + result);
        } catch( IllegalArgumentException e ) { //處理 IllegalArgumentException 例外
            System.out.println("例外說明: "+e.getMessage());
            System.out.print("例外原因: ");
            e.printStackTrace();
        } catch( ArrayIndexOutOfBoundsException e ) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } finally {
```

```
                System.out.println("錯誤處理結束");
            }
        }
// 副程式: 計算 a*b/c 的值
    static double cal(double a, double b, double c) throws IllegalArgumentException {
            double value;
            if ( c == 0 ) {   // 丟出 IllegalArgumentException 例外
                throw new IllegalArgumentException("c 不能是 0!");
            } else {
                value = a*b/c;
                if ( value < 0 ) {   // 丟出 IllegalArgumentException 例外
                    throw new IllegalArgumentException("運算結果小於 0");
                }
            }
            return value;
    }
}
```

■ 執行結果

```
D:\Course_Data\Example>java ExceptionTest
D:\Course_Data\Example>java -cp .;g: ExceptionTest aaa
計算結果: 3
計算結果: 6
例外說明: / by zero
例外原因: java.lang.ArithmeticException: / by zero
            at ExceptionTest.main(ExceptionTest.java:29)
未執行 finally,程式繼續執行...
請輸入三個數值計算 a*b/c
6
2
0
例外說明: c 不能是 0!
例外原因: java.lang.IllegalArgumentException: c 不能是 0!
            at ExceptionTest.cal(ExceptionTest.java:63)
            at ExceptionTest.main(ExceptionTest.java:47)
錯誤處理結束

D:\Course_Data\Example>java -cp .;g: ExceptionTest
例外說明: 0
```

```
例外原因: java.lang.ArrayIndexOutOfBoundsException: 0
        at ExceptionTest.main(ExceptionTest.java:26)
未執行 finally,程式繼續執行...
請輸入三個數值計算 a*b/c
6
-2
3
例外說明:  運算結果小於 0
例外原因: java.lang.IllegalArgumentException:  運算結果小於 0
        at ExceptionTest.cal(ExceptionTest.java:67)
        at ExceptionTest.main(ExceptionTest.java:47)
錯誤處理結束

D:\Course_Data\Example>java -cp .;g: ExceptionTest
例外說明: 0
例外原因: java.lang.ArrayIndexOutOfBoundsException: 0
        at ExceptionTest.main(ExceptionTest.java:26)
未執行 finally,程式繼續執行...
請輸入三個數值計算 a*b/c
6
6
2
計算結果: 18.0
錯誤處理結束
```

集合物件基本觀念與應用－Collection

- 集合物件是一個物件的容器，可以用來處理一些不定數量的資料，集合物件簡單的說就是物件導向程式的資料結構，Java 提供了主要的集合物件有 Collection、List、Set、Map，此處介紹較基本的 List。
- Java 的集合由 java.util 套件提供，因此在使用集合物件的各種物件之前，要先匯入 java.util.*套件(如同處理檔案要匯入 java.io.*一樣)。在此提出應用於處理未知範圍的陣列資料時的集合物件。亦即當處理的資料元素數目並不確定時，無法宣告一個確定範圍的陣列來儲存，此時可利用集合的觀念來處理，程式的處理上就像是準備了一個沒有限制的空桶子來收集資料，需要存放時就把資料丟進去，而程式會自動把這些資料依照存放的順序先後安排編號，使用時再依序取出來。

集合介面基礎概念

- 集合物件(Collections)是指一組相關的物件集合，將這組物件集合視為單一物件，在

集合物件中的物件稱為元素(Elements)，集合物件有很多種，其儲存的元素可能允許重複，有些集合物件的元素會進行排序。

- 集合介面(Collection Interface)是處理集合物件中儲存的物件，提供一致的物件操作力式來新增、刪除和搜尋元素的方法，此處主要用的兩種
  - Collection 介面：這是類別架構的根介面，不過並沒有任何類別是直接實作 Collection 介面，但是提供了 Iterator 類別用以存取集合的方法。
  - List 介面：實作 List 介面的集合物件可以擁有重複元素，元素是以循序方式存入，即以類似陣列索引方式來存取元素。

List 介面的集合類別

- List 介面繼承自 Collection 介面，實作此介面的集合類別是一種循序集合物件(Ordered Collection)，允許重複元素，元素擁有索引位置，可以使用類似陣列索引方式來存取元素。
  - **循序集合物件(Ordered Collection)**和**排序集合物件(Sorted Collection)**並不相同，循序集合物件是如同陣列以元素位置的索引來排列，排序集合物件是以元素值的大小進行排列。
  - 主要提供放置集合元素的常用方法如下

boolean	add(E o)－把元素加入集合
void	clear()－清除集合內的元素
boolean	contains(Object o)－判斷集合內是否有某個特定的元素
boolean	equals(Object o)－判斷集合是否與指定的集合相等(元素及索引均要一致)
boolean	isEmpty()－判斷集合是否為空集合
Iterator<E>	iterator()－走訪集合內的元素
boolean	remove(Object o)－移除集合內某個被指定的元素
int	size()－傳回集合內的元素數目
E	get(int index)－取得指定索引的元素
int	indexOf(Object o)－取得元素的索引值
void	set(int index, Object o)－重新設定已加入的某一個特定索引位置的元素值
Object[]	toArray()－將集合轉成陣列物件

- ArrayList 類別－元素如其名是以類似陣列方式來儲存，元素使用索引位置依序存入，只需將元素新增或插入 ArrayList 物件，並不用事先宣告物件尺寸，如同一種可自動調整陣列尺寸的動態陣列，例如：

```
ArrayList alist = new ArrayList() ;
```

上述程式碼使用建構子建立 ArrayList 物件儲存字串元素，在新增 ArrayList 物件的變數後，Java 程式可以使用而迴圈配合索引取得每一個物件元素，例如

```
for (int i = 0; i < alist.size(); i++)
    System.out.print(alist.get(i) + " ") ;
```

  - 上述程式碼使用 size()方法取得元素數，再用 get()方法配合索引位置取出元素。

- 利用 add 方法增加元素時，如果只輸入元素名稱，表示從 0 開始自動增加索引值，亦可在輸入時自行指定索引值，即 add(index, element)。
- 當移除集合中的某個元素後，對應的索引也會自動由下面一個元素遞移補上。
- 如果有重複的元素，仍會有不同的索引，但此時若要查詢此重複元素的索引值時，則傳回最先出現的元素索引。

➢ LinkedList 類別－這是類似鏈結串列(Linked Lists)資料結構的類別，各節點如同火車掛車廂一般，將每一個車廂的節點連結起來，串列節點擁有指向下一個節點的指標，最後一個節點的指標是指向 null，表示沒有下一個節點。

- LinkedList 增加了處理首尾元素的方法：

void	addFirst(E o)－在串列集合最前面增加元素
void	addLast(E o)－在串列集合最後面增加元素
E	getFirst()－取得串列集合的第一個元素
E	getLast()－取得串列集合的最後一個元素
E	removeFirst()－移除串列集合的第一個元素
E	removeLast()－移除串列集合的最後一個元素

Iterator 輸出集合物件元素

➢ Collections 集合物件可以使用 Iterator 的一致走訪方式來輸出集合物件內的元素。

- Iterator 提供一致方法來走訪集合物件的元素或刪除元素。
- 實作 List 介面的集合物件除了可以使用 Iterator 外，還可以使用 ListIterator，除了使用一致走訪方法外，還可以雙向走訪集口物件的元素，即從頭到尾，或是從尾到頭進行走訪。

➢ Iterator 介面輸出元素－Iterator 介面提供以下方法輸出介面元素

boolean	hasNext()－判斷目前在集合中的位置後面是否還有下一個元素
E	next()－傳回集合中的下一個元素
int	nextIndex()－傳回集合中的下一個元素的索引
void	remove()－移除上一個傳回的元素

- Iterator 的宣告方式如下(一般和 List 的相關物件一齊宣告)，

  Iterator iterator_name = iterator_method

  例如，假如已經存在一個 ArrayList 的變數 alist，則

  Iterator iterator = alist.iterator() ;
- 在取得 Iterator 後，只需呼叫其方法，就可以配合 while 迴圈走訪集合物件的元素，先使用 hasNext()方法檢查是否有下一個元素，並使用 next()方法取得此元素，再依序走訪和刪除元素，如下表所示:

```
while (iterator.hasNext()) {
    ……
    System.out.print(“ “ + iterator.next()) ;
    ……
```

```
}
```

- ListIterator 介面輸出元素－ListIterator 繼承 Iterator 的功能外，更提供了雙向走訪集合元素的功能
  - ListIterator 的宣告方式和 Iterator 類似

    ListIterator listiterator_name = listiterator_method

    例如，假如已經存在一個 LinkedList 的變數 llist，則

    ListIterator listiterator = llist.listiterator() ;
  - 取得 ListIterator 後的使用方式則和 Iterator 一樣，但是多了可以處理向前走訪元素的相關方法。
  - ListIterator 增加了以下幾個方法，

void	add(E o)－增加元素 Inserts the specified element into the list (optional operation).
boolean	hasPrevious()－走訪集合時判定前一個元素是否存在
E	previous()－走訪集合的下一個元素
int	previousIndex()－走訪集合的下一個元素索引
void	set(E o)－重新設定剛走訪過的元素值

  - 回顧 List 介面物件提供 listIterator()方法，可以取得 ListIterator<E>介面物件，例如 ArrayList<String>物件 alist 可以使用上表方法取得 ListIterator 介面物件，如下所示:

    ListIterator<String> iterator = alist.listIterator(0) ;

    上述程式碼將 ArrayList 物件的元素轉換成 ListIterator<E>介面物件，泛型型態為 String，然後就可以使用上表之介面方法雙向走訪、新增、取代和刪除元素。
- Example: 綜合測試以上所提到的類別方法

```
// import java.util.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
public class CollectionTest {
      public static void main(String[] args) {
            String[] team = {"洋基", "鈴木一郎", "王建民", "曹錦輝", "郭泓志", "陳金鋒", "
松井秀喜",
            "Kobe Bryan", "Sharq O'Neil", "陳金鋒", "胡金龍", "姜建銘"} ;
            String[] bullpen = {"Mike Mussina", "Carl Pavano", "Randy Johnson",
            "Jose Veras", "Mike Myers", "Mariano Rivera", "王建民"} ;
//  測試 ArrayList,印出所有元素,並移除部份元素
//             List player = new ArrayList() ;
```

```
            ArrayList player = new ArrayList() ;
            System.out.println("測試 ArrayList,印出所有元素,並移除部份元素") ;
            for(int i=0; i<team.length; i++) {
                  player.add(team[i]) ;     // 加入元素
            }
            player.set(0,"Yankee") ;       // 將第一個元素值做修改
            for(int i=0; i<player.size(); i++) {
                  System.out.println("The player index " + i + " is: " + player.get(i)) ;
                  if(player.get(i).equals("Kobe Bryan") || player.get(i).equals("Sharq
O'Neil")) {
                        System.out.println("This is not the baseball player.....be removed") ;
                        player.remove(i) ;   // 移去指定索引的元素質
                        i-- ;
                  }
            }
            System.out.println("The final number in the collection is " + player.size()) ;
//  測試 LinkedList,在集合首尾加元素
//             List pitcher = new LinkedList() ;
            System.out.println("測試 LinkedList,在集合首尾加元素") ;
            LinkedList pitcher = new LinkedList() ;
            for(int i=0; i<bullpen.length; i++) {
                  pitcher.add(bullpen[i]) ;
            }
            pitcher.addFirst("王建民") ;     // 在串列集合之前再加一個元素
            pitcher.addLast("姜建銘") ;      // 在串列集合之後再加一個元素
//  測試 Iterator,印出 ArrayList 保留下來的部份,並再移除部分元素
            Iterator it = player.iterator() ;
            if(player.isEmpty()) {
                  System.out.println("Nothing in the collection") ;
            } else {
                  System.out.println("利用 Iterator 走訪 ArrayList 保留下來的元素,並再移除
部分元素") ;
                  while(it.hasNext()){
                        String name = (String)it.next() ;     // 將走訪的元素值轉型為字串以
便找出其索引
                        System.out.println("The player index " + player.indexOf(name) + ": "
+ name) ;
                        if(name.equals("松井秀喜") || name.equals("鈴木一郎")) {
```

```
                        System.out.println("This guy is not from Taiwan...be
removed") ;
                        it.remove() ;    // 移除走訪到的元素
                    }
                }
            }
// 測試 ListIterator 對集合元素進行雙向走訪
            ListIterator lit = pitcher.listIterator() ;
            System.out.println("以 ListIterator 向後走訪集合") ;
            while(lit.hasNext()) {
                System.out.println("The player index " + lit.nextIndex() + ":" + lit.next()) ;
            }
            System.out.println("以 ListIterator 向前走訪集合") ;
            while(lit.hasPrevious()) {
                System.out.println("The player index " + lit.previousIndex() + ":" +
lit.previous()) ;
            }
    }
}
```

執行過程

```
測試 ArrayList,印出所有元素,並移除部份元素
The player index 0 is: Yankee
The player index 1 is: 鈴木一郎
The player index 2 is: 王建民
The player index 3 is: 曹錦輝
The player index 4 is: 郭泓志
The player index 5 is: 陳金鋒
The player index 6 is: 松井秀喜
The player index 7 is: Kobe Bryan
This is not the baseball player.....be removed
The player index 7 is: Sharq O'Neil
This is not the baseball player.....be removed
The player index 7 is: 陳金鋒
The player index 8 is: 胡金龍
The player index 9 is: 姜建銘
The final number in the collection is 10
測試 LinkedList,在集合首尾加元素
利用 Iterator 走訪 ArrayList 保留下來的元素,並再移除部分元素
```

```
The player index 0: Yankee
The player index 1: 鈴木一郎
This guy is not from Taiwan...be removed
The player index 1: 王建民
The player index 2: 曹錦輝
The player index 3: 郭泓志
The player index 4: 陳金鋒
The player index 5: 松井秀喜
This guy is not from Taiwan...be removed
The player index 4: 陳金鋒
The player index 6: 胡金龍
The player index 7: 姜建銘
以 ListIterator 向後走訪集合
The player index 0:王建民
The player index 1:Mike Mussina
The player index 2:Carl Pavano
The player index 3:Randy Johnson
The player index 4:Jose Veras
The player index 5:Mike Myers
The player index 6:Mariano Rivera
The player index 7:王建民
The player index 8:姜建銘
以 ListIterator 向前走訪集合
The player index 8:姜建銘
The player index 7:王建民
The player index 6:Mariano Rivera
The player index 5:Mike Myers
The player index 4:Jose Veras
The player index 3:Randy Johnson
The player index 2:Carl Pavano
The player index 1:Mike Mussina
The player index 0:王建民
```