Java SE 個人筆記

黃懷慶

更新日期：2026/02/06

目錄

一、Java 的主程式架構

```
package 資料夾;

import 資料夾.物件;
import 資料夾.資料夾.物件;
……

class 主程式物件名稱 extends 物件 implements 介面A, 介面B, … {
        //宣告全域變數……
        前贅詞 資料型態 變數名 ……

        public static void main(String [] args){
                //宣告區域變數
                資料型態 變數名 ……

                內容;
        }                                                    【main函式】
}
```


有main函式的class就是主程式
也只有主程式的class才能夠執行

二、基本資料型態

1. 基本的資料型態：

   整　數：long > int > short > byte

   浮點數：double > float

   文　字：char(字元，'文字')

   布　林：boolean

2. 包覆類別：

   整　數：Long > Integer > Short > Byte

   浮點數：Double > Float

   文　字：String(字串，"文字") > Char

   布　林：Boolean

   ＊包覆類別為 java.lang 下的類，裡面有一些 function 使用，如：型態轉換。

3. 自動辨識型別的特殊屬性：

var 是 Java 10 中 新增的 特殊屬性，特點是會根據 內容 自動推斷屬性的類型。

語法範例：

```
var number = 10;  // 編譯器會推斷 number 是 int 類型
var name = "Hello";  // 編譯器會推斷 name 是 String 類型
```

限制：

A. 僅能用於 區域變數。
B. 宣告的時候 必須 賦值，且 其值 必須能夠推斷類型。

```
var x;  // 編譯錯誤，缺少初始化值
var obj = null;  // 編譯錯誤，無法推斷 null 的具體類型
```

三、註解

單行註解：

```
//註解內容
```

多行註解：

```
/*註解內容*/
/*
  註解內容
  註解內容
*/
```

四、跳脫字元、強制轉換與資料輸出

1. 跳脫字元：

\t	Tab
\n	換行

＊此為字串中的一些特殊符號。

2. 強制轉換字元：

「'」、「"」和「\」是 java 的特殊符號，要在字元或字串中儲存，需要使用「\」強制轉換。

字元	'	char a1 = '\''
	"	char a2 = '\"'
	\	char a3 = '\\'
字串	'	String b1 = "\'"
	"	String b2 = "\""
	\	String b3 = "\\"

3. 資料輸出：

(1) 同行輸出：

```
System.out.print(輸出內容);
```

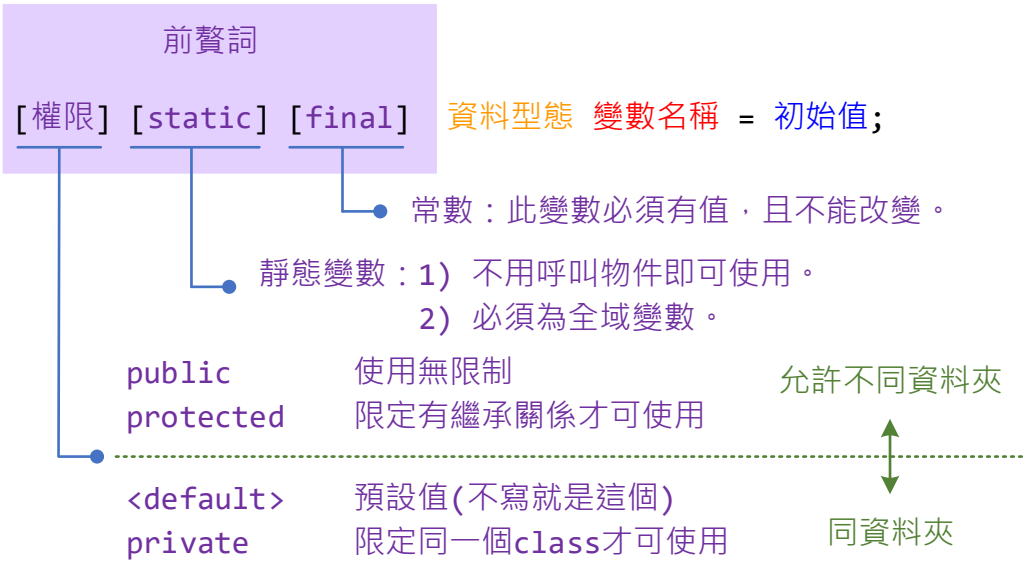
(2) 換行輸出：

```
System.out.println(輸出內容);
```

五、變數宣告

1. 基本：

(1) 語法：



(2) 範例：

```
public static final double pi = 3.14;
```

2. 一維陣列：

(1) 靜態宣告：


```
變數[0] = 值;
變數[1] = 值;
.....
變數[n] = 值;
```

＊陣列的變數計數從 0 開數。

(2) 動態宣告：

前贅詞

[權限] [static] [final] 資料型態 [ ] 變數名稱 = {值, 值, ……, 值};

給幾個值，陣列長度就是多少

＊陣列的變數計數從 0 開數。

3. 二維陣列：

(1) 靜態宣告：

```
前贅詞 資料型態 [ ][ ] 變數名稱 = new 資料型態[長度][長
度]
變數[0][0] = 值;
變數[0][1] = 值;
……
變數[n][m] = 值;
```

＊陣列的變數計數從 0 開數。

(2) 動態宣告：

前贅詞 資料型態 [ ] 變數名稱 = {{值, …, 值}, …, {值, …, 值}};

給幾個值，陣列長度就是多少

＊陣列的變數計數從 0 開數。

六、運算子

1. 算數運算子：

+	加法
-	減法
*	乘法
/	除法
%	求餘數

2. 指定運算子：

=	將「右邊的值」給予「左邊的變數」
+=	如：x+=y 為 x = x + y 的簡寫
-=	如：x-=y 為 x = x - y 的簡寫
*=	如：x*=y 為 x = x * y 的簡寫
/=	如：x/=y 為 x = x / y 的簡寫
%=	如：x%=y 為 x = x % y 的簡寫
++	x++，x = x + 1 的簡寫。特性為先運算，後加 1。
	++x，x = x + 1 的簡寫。特性為先加 1，後運算。
--	x--，x = x – 1 的簡寫。特性為先運算，後減 1。
	--x，x = x – 1 的簡寫。特性為先減 1，後運算。

3. 關係運算子 1：

&&	and，左右兩邊都為 True，回傳 True
||	or ，左右其一為 True，回傳 True
^	xor，左右兩邊都為 True 或 False，回傳 True
!	not，反轉 True 和 False。
	True → Fales、False → True

＊關係運算子會回傳 True 或 False。

4. 關係運算子 2：

>	大於　　，左邊 > 右邊 回傳 True
<	小於　　，左邊 < 右邊 回傳 True
>=	大於等於，左邊 >= 右邊 回傳 True
<=	小於等於，左邊 <= 右邊 回傳 True
==	等於　　，左右兩邊相同 回傳 True (物件：比較物件是否相同)
equals	等於　　，左右兩邊相同 回傳 True (比較物件數值是否相同；包覆類別用)
!=	不等於　，左右兩邊不相同 回傳 True

＊關係運算子會回傳 True 或 False。

＊「==」和「equals」的差異詳見下表：

比較對象	==	equals
一般資料型態	值	無此方法
包覆類別	比較物件是否相同	比較物件的值是否相同

案例說明：

(1)　s1 和 s2 為 『值 = abc』的 2 個不同 String 物件

(2)　i1 和 i2 為 『值 = 1』　的 2 個不同 Integer 物件

```
public static void main(String[] args) {
    String s1 = new String("abc");
    String s2 = new String("abc");
    boolean b1 = (s1 == s2);
    System.out.println("String 比較測試");
    System.out.println("==     : " + b1);
    System.out.println("equals: " + s1.equals(s2));

    System.out.println("");

    Integer i1 = new Integer(1);
    Integer i2 = new Integer(1);
    boolean b3 = (i1 == i2);
    System.out.println("Integer 比較測試");
    System.out.println("==     : " + b3);
    System.out.println("equals: " + i1.equals(i2));
}
```

```
String 比較測試
==     : false
equals: true

Integer 比較測試
==     : false
equals: true
```

5. 位元運算子：

&	對二進位的每一格進行 and 判斷 5: 0000 0101 2: & 0000 0010 0: 0000 0000
|	對二進位的每一格進行 or 判斷 5: 0000 0101 2: | 0000 0010 7: 0000 0111
^	對二進位的每一格進行 xor 判斷 5: 0000 0101 7: ^ 0000 0111 3: 0000 0010
~	補數，正負數反轉 正整數: 0 1 2 ..... 負整數: -1 -2 -3 .....

6. 位移運算子：

<<	對二進位的每一格向左移動 5 << 2 5: 0000 0101 20: 0001 0100 遞補
>>	對二進位的每一格向右移動 5 >> 2 5: 0000 0101 7: 0000 0001 遞補

七、邏輯判斷

1. if 判斷式：

```
if (布林條件式) {
    內容;
}
else if (布林條件式) {
    內容;
}
……以此類推
else {
    內容;
}
```

2. switch 判斷式：

```
switch (變數) {
    case 值 A:
        內容;
        break;      ← 沒有 break 將會繼續往下執行
    case 值 B:
        內容;
        break;
    ……以此類推
    case 值 N:
        內容;
        break;
    default:        ← 其他情況：變數 不符合 (值 A ~ 值 N) 會執行這裡
        內容;
        break;

}
```

＊變數類型：整數(int、short、byte)、字元(char)、字串(String)。

3. 三元運算子：

(1) 語法：

```
布林條件式 ? True 儲存的值 : False 儲存的值
```

(2) 範例：

```
cn = (chi >= 60 ? "及格" : "不及格");
```

＊當 chi >= 60，則 cn = "及格"，否則 cn = "不及格"

八、迴圈

1. for 迴圈：

(1) 語法：

```
for (變數 = 初始值 ; 布林條件式 ; 迴圈計算式) {
    內容;
}
```

＊符合布林條件式才會執行迴圈內容。

＊迴圈每次跑完後會執行迴圈計算式。

(2) 範例：

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

2. for-each 迴圈：

只有陣列類型的變數才可使用的 Java 特殊迴圈，
其運算是「逐一取出陣列中的值」。

```
for (陣列值的資料型態 變數名稱 : 陣列名稱) {
    內容;
}
```

3. while 迴圈：

```
while (布林條件式) {
    內容;
}
```

＊與 for 不同的是「沒有執行次數限制」，
若內容沒有改變「布林條件式的內容」或加上「迴圈控制」，將造成無限迴圈。

4. do-while 迴圈：

```
do {
    內容;
} while (布林條件式)
```

＊與 while 迴圈不同的是「無論是否符合條件」迴圈內容都「至少會執行一次」。
利用此特性可以用來記錄錯誤訊息。

5. 迴圈控制：

(1) bleak：離開迴圈。
(2) continue：離開「該次」迴圈。

九、物件

依照程式撰寫時的需要，可以將需要的功能單獨寫成好幾個 class。
在主程式中要使用時，只要先 import 進來後，再「初始化」該 class 就可以使用了。

1. 物件初始化：

(1) 語法：

```
class名稱 物件變數名稱 = new class名稱(參數, ……);
```

相同

或有繼承關係
父類別(左)、子類別(右)

(2) 執行的動作：

記憶體中建立索引位置 → 記憶體中產生變數欄位 → 找尋建構式並執行

2. 物件的撰寫：

```
[權限] class 物件名稱 {
    //宣告變數……
    int chi;
    int eng;

    //建構式，可以有很多個，但參數數量不能相同
    物件名稱(資料類型 參數名, ……){
        //用 this.變數名 指向全域變數
        //直接使用 變數名 通常用來指 建構式的參數
        內容;
    }
    物件名稱(int chi){
        this.chi = chi;
    }
    物件名稱(int chi, int eng){
        this.chi = chi;
        this.eng = eng;
    }

    //函式
    類型 函式名稱(資料型態 參數, ……){
        內容;
        return 回傳變數;    ← void不用寫
    }
}
```

＊建構式的名稱必須跟物件名稱相同。

十、方法

Java 的方法允許 Overloading，也就是「方法名稱相同」且「參數的數量不同」。

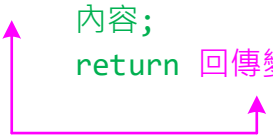
其方法依照有無回傳值分成以下兩類：

1. 不傳值的方法：

```
權限 void 方法名稱(資料型態 參數, ……){
        內容;
}
```

2. 傳值的方法：

```
權限 類型 函式名稱(資料型態 參數, ……){
        內容;
        return 回傳變數;
}
```



資料形態要相同

＊方法的類型可以是：

1) int、String 等基本資料型態。
2) class。

十一、 陣列的常用方法

使用前需要 impory java.util.Arrays。

1. 由小到大排序：

(1) 語法：

```
Arrays.sort(陣列名稱);
```

(2) 範例：

```
int [] x = {5, 2, 4, 3, 1};
Arrays.sort(x);      //此時 x 的值變成 {1, 2, 3, 4, 5}
```

2. 搜尋：

(1) 語法：

```
Arrays.binarySearch(陣列名稱, 查詢數值)
```

＊有查到會回傳「陣列的 index」，否則回傳「-」+「應該排在第幾個位置」。

(2) 範例：

```
int [] x = {1, 2, 3, 4, 5};
System.out.println(Arrays.binarySearch(x, 3));  //回傳 2
                                                //x[0] = 1
                                                //x[0] = 2
                                                //x[0] = 3 ←
                                                //x[0] = 4
                                                //x[0] = 5
System.out.println(Arrays.binarySearch(x, 8));  //回傳 -6
                                                //x[0] = 1
                                                //x[0] = 2
                                                //x[0] = 3
                                                //x[0] = 4
                                                //x[0] = 5
                                                        6 ← 第 6 個
```

十二、 繼承

1. 語法：

【父類別】

```
[權限] class 父類別名稱 {
    //宣告變數

    //建構式
    父類別名稱(資料型態 父參數A, ……){
        內容;
    }

    //方法
    類型 方法名稱(資料型態 參數, ……){
        內容;
        return 回傳函數值;
    }
}
```

【子類別】

```
[權限] class 子類別名稱 extends 父類別名稱 {
    //宣告變數

    //建構式
    子類別名稱(資料型態 父參數A, ……, 資料型態 子參數B, ……){
        super(資料型態 父參數A, ……);  ← 一定要在第一行
        內容;
    }

    //方法
    類型 方法名稱(資料型態 參數, ……){
        內容;
        return 回傳函數值;
    }
}
```

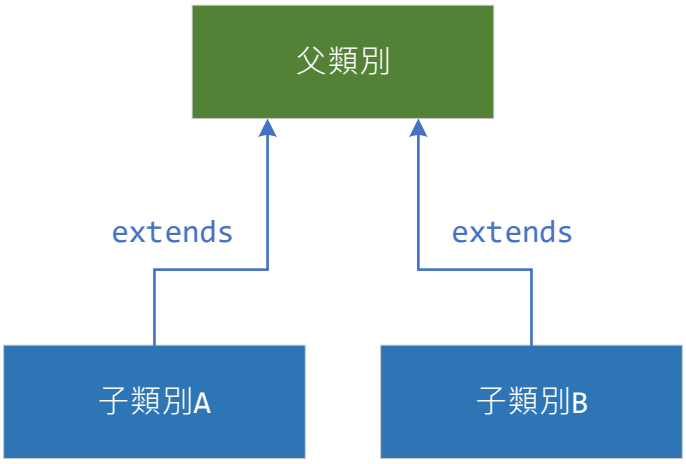
2. 父類別、子類別與繼承限制：

(1) 被繼承的 class 稱為父類別。

(2) 繼承的 class 稱為子類別，可以直接使用父類別的「變數」和「方法」。

(3) 一個子類別，只能繼承一個父類別。

(4) 一個父類別，可以被很多子類別繼承。



3. 建構式：

(1) 子類別會繼承父類別的建構式。

(2) 執行順序：

父類別建構式 → 子類別建構式

(3) 子類別的建構式獲得傳入的參數後，必須先將相關參數傳給父類別的建構式。

```
super(參數, ……);
```

(4) 範例：

【父類別建構式】

```
superClass(String name, int chi){
        this.name = name;
        this.chi = chi;
}
```

【子類別建構式】

```
subclass(String name, int chi, int eng){
        super(name, chi);  ← 一定要在第一行
        this.eng = eng;
}
```

4. 方法：

(1) 子類別可以改寫(override)父類別的方法，
此時該方法的「名稱」和「參數」都要相同。

(2) 當子類別要改寫方法時，

a. 改寫後方法的權限要相同或是更大。

b. 當這個方法有回傳值時，改寫時可以用『super.方法名稱()』
取得改寫前的回傳值。

(3) 範例：

【父類別的方法】

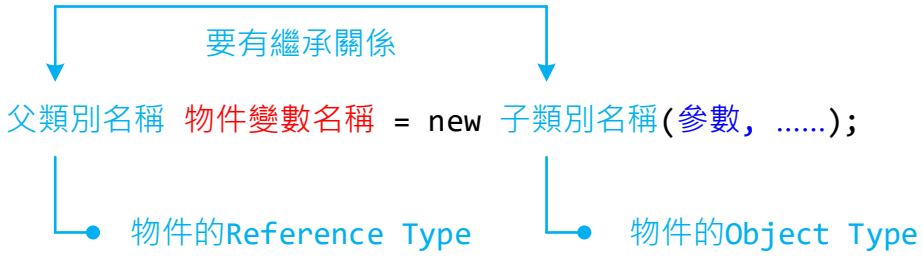
```
protected String show(){
    return "姓名" + name;
}
```

【子類別的方法】

```
public String show(){
    return super.show() + "國文分數" + chi;
}
```

十三、 多型

1. 語法：



2. 優點：

透過多型宣告的機制，可以方便我們管理方法。

(1) 透過父類別來統一建立物件的方法。
(2) 透過子類別來個別建立方法的內容。

例如：
先透過父類別來統一建立「保險商品需要的功能方法」，如：檢核理賠條件。
再讓繼承這個父類別的個別保險商品，用 override 的方式來「個別設定理賠檢核條件」。

3. 缺點：

使能使用「繼承」或「override」的方法。

4. 編譯的順序：

檢查兩個類別是否有繼承關係 → 判斷使用的方法是否存在父類別中

十四、 抽象類別與抽象方法

1. 語法：

```
[權限] abstract class 類別名稱 {
        //宣告變數

        //建構式
        類別名稱(資料型態 參數, .....){
                內容;
        }

        //抽象方法
        abstract void 函式名稱(資料型態 參數, .....);

        //一般函式
        類型 函式名稱(資料型態 參數, .....) {
                內容;
        }
}
```

2. 抽象類別：

(1) 可以宣告抽象方法。
(2) 抽象類別中「不能初始化(new)物件」。

3. 抽象方法：

(1) 沒有內容的方法。
(2) 被繼承時，子類別必須要 override 該方法。
(也就是一定要提供方法的內容)

十五、 interface (介面)

1. 語法：

【介面】

```
interface 介面名稱 extends 介面A, 介面B, ..... {
        //只能宣告常數
        public final 常數名稱 = 值;
                     ↑ 可省略
        //只能建立抽象方法
        public abstract 類型 函式名稱(資料類型 參數, .....);
                        ↑ 可省略
}
```

interface可以繼承許多interface

【類別】

```
[權限] class 類別名稱 implements 介面A, 介面B, ..... {
        //宣告變數
        //建構式
        類別名稱(資料型態 參數A, .....){
                內容;
        }

        //函式
        類型 函式名稱(資料型態 參數, .....){
                內容;
                return 回傳函數值;
        }
}
```

可以實作許多interface

2. 特點：

(1) 介面是一種特殊的父類別，在介面裡面只能有「常數」和「抽象方法」。
(2) 當子類別實作介面時，必須要 override 介面中的抽象方法。

3. 綜合範例：

```
public class ABC extends Class_A implements Interface_A, Inteface_B
{
    ……
}
```

十六、 內部類別 與 匿名類別

1. 內部類別：

(1) 在一個 class 的裡面，再寫一個 class，裡面的這個 class 就是「內部類別」。
(2) 「內部類別」不能「繼承」，而是與「變數」和「方法」相同，
都是「外部類別」的一個屬性。
(3) 語法：

```
class 外部類別 extends 物件 implements 介面, ……
{
    //內部類別
    class 內部類別 A1
    {
        //巢狀內部類別
        class 內部類別 A2
        {
            //巢狀內部類別中的方法
            類型 方法 A2_1(資料型態 參數, ……)
            {
                內容;
            }
        }

        //內部類別中的方法
        類型 方法 A1_1(資料型態 參數, ……)
        {
            內容;
        }
    }

    //內部類別
    static class 內部類別 B1
    {
        //內部類別的方法
        類型 方法 B1_1(資料型態 參數, ……)
        {
            內容;
        }
    }
}
```

(4) 宣告內部類別的物件 與 使用方法：

a. 沒有 static：

```
外部類別.內部類別 A1 物件變數 = new 外部類別().new 內部類別 A1();
物件變數.方法 A1_1();
```

＊當 class 沒有 static 時，宣告物件必須每個都要 new。

b. 有 static：

```
外部類別.內部類別 B1 物件變數 = new 外部類別().內部類別 B1();
物件變數.方法 B1_1();
```

＊內部類別 B1 有「ststic」，所以宣告物件時內部類別 B1 不需要 new。

(5) 若內部類別中的方法有 static 時，這個內部類別也必須有 ststic，
否則編譯不會過。

2. 匿名類別：

(1) 「匿名類別」就是沒有給名稱的類別，其特性是「只能使用一次」。
(2) 匿名類別的常見用途：

a. 與「抽象類別」搭配使用，也就是「透過匿名類別來 override」。
b. 在沒有主程式的類別中，可以透過匿名類別來進行前置運算。

(3) 語法：

a.
```
抽象類別 物件變數 = new 抽象類別(參數, ……)
{
    override 抽象方法;
};
```

b.
```
static {
    內容;
}
```

十七、 enum (列舉)

1. enum 是「用來保存專有名詞(String)」的 class，其儲存結構「類似陣列」。
2. 只能保存 String 型態的資料，且內容有嚴格規定：

   (1) 不允許「數字開頭」。
   (2) 不允許「中間有空格」

3. 內部類別中可以有多個 enum。
4. 語法：

   (1)
   ```
   enum 列舉類名稱 A
   {
         字串 A1, 字串 A2, 字串 A3, ……
   }
   ```

   (2)
   ```
   class 外部類別
   {
         enum 列舉類名稱 B
         {
             字串 B1, 字串 B2, 字串 B3, ……
         }
         enum 列舉類名稱 C
         {
             字串 C1, 字串 C2, 字串 C3, ……
         }
   }
   ```

(3) enum 列舉類名稱 {

```
        /** 變數名稱 A 的說明 */                ←若有說明，在 eclipse 中可以
        物件 A(數值 A, 數值 B, ……);                 在使用時顯示說明內容
        /** 變數名稱 B 的說明 */                藉此知道這個的內容是啥
        物件 B(數值 A, 數值 B, ……);
        ……
                /****** 以下為設定區域 ******/
        //宣告變數
        private 類型 A 參數 A;
        private 類型 B 參數 B;
        ……
        //建構式：設定上面所有變數名稱的各個數值對應的參數和類型
        private 列舉名稱(類型 A 參數 A, 類型 B 參數 B, ……) {
            this.參數 A = 參數 A;
            this.參數 B = 參數 B;
            ……
        }
        //getting：設定單獨取值得方法
        public 類型 get 變數 A {
            return 變數 A;
        }
        public 類型 get 變數 B {
            return 變數 A;
        }
        ……
    }
```

＊使用列舉類名稱.物件 A，就可取得物件 A 設定的中的各個參數值。

5. 使用方式：

在一般 class 中要使用時，需要透過 values()來將內容轉移到陣列中保存。

```
(1)  列舉名稱 A [] 物件名稱 a = 列舉名稱 A.values();
(2)  外部類別.列舉名稱 B [] 物件名稱 b = 外部類別.列舉名稱 B.values();
     外部類別.列舉名稱 C [] 物件名稱 c = 外部類別.列舉名稱 C.values();
```

之後就如同陣列一樣，透過「物件名稱」就可以取出相對應的 String 資料了。

```
System.out.print(物件名稱 a[0]);  //輸出「字串 A1」
```

6. 範例：

```
public enum enumClass {
    /** 測試物件 A */
    enumObject("王曉明", 50, 40);

    //宣告參數
    private String name;
    private int chi;
    private int eng;
    //建構式：設定參數
    private enumClass(String name, int chi, int eng) {
        this.name = name;
        this.chi = chi;
        this.eng = eng;
    }
    //參數的 getting 方法
    public String getName() {
        return name;
    }
    public int getChi() {
        return chi;
    }
    public int getEng() {
        return eng;
    }
}
```

【使用】

```
System.out.println(enumClass.enumObject);
System.out.println(enumClass.enumObject.getName());
System.out.println(enumClass.enumObject.getChi());
System.out.println(enumClass.enumObject.getEng());
```

十八、 錯誤處理

錯誤處理是 Java 的安全檢查機制，當程式執行中出現錯誤時，錯誤處理機制可以拋出錯誤訊息，或是讓程式設計師自行決定要如何處理。

1. 錯誤訊息的分類：

Java 將錯誤訊息依照嚴重的程度，分成以下三種：

(1) ERROR：

a. 嚴重錯誤，會強制中斷程式，並拋出錯誤訊息。
b. 錯誤訊息皆繼承 java.lang.Error。
c. 此類錯誤在編譯或執行階段都有可能遇到。
例如：變數重複宣告，就是屬於此類的錯誤。

(2) Runtime Exception：

a. Java 預先決定好的「運行時錯誤訊息」。
b. 錯誤訊息屬於 java.lang.RuntimeException，
繼承自 java.lang.Exception。
c. 此類錯誤 Java 會「自動檢查」，並拋出錯誤訊息。
例如：除法算式的分母為 0，就是屬於此類錯誤。

(3) Checked Exception：

a. 需要程式設計師「自行」決定的錯誤類型，
使用時需要將該錯誤類型 import 進來。
b. 繼承自 java.lang.Exception 的錯誤訊息，
除了 java.lang.RuntimeException 外，都屬於此類型。
c. 錯誤訊息共有 2 種處理方式：
1) 使用「throws」拋出錯誤訊息。
2) 使用「try-catch」捕捉錯誤，並決定後續處理程式。

2. throws (拋出錯誤)：

(1) 遇到錯誤時，直接顯示錯誤訊息，並中斷程式。

(2) 語法：

```
class 主程式 extends 物件 implements 介面, ……
{
    public static void main(String[] args) throws 錯誤類型, ……
    {
        可能遇到錯誤的內容;
        內容;
    }
}
```

＊需要拋出許多不同的錯誤類型時，用「,」分隔即可。

＊錯誤類型根據可能出現錯誤的程式而有所不同。
如下面的範例中，File 會對應 IOException。

(3) 範例：

```
import java.io.File;
import java.io.IOException;

public class t1 extends baba implements int1, int2 {

    public static void main(String[] args) throws IOException {
        File file = new File("c:/a.txt");
        file.createNewFile();
    }
}
```

【執行後訊息】

```
Exception in thread "main" java.io.IOException: 系統找不到指定的路徑。
    at java.io.WinNTFileSystem.createFileExclusively(Native Method)
    at java.io.File.createNewFile(Unknown Source)
    at test.t1.main(t1.java:10)
```

＊使用 File 中的 crateNewFile()會有 Checked Exception，
此處選擇拋出錯誤，就是遇到錯誤時，直接顯示「預設的錯誤訊息」。

＊例如要新增檔案時，該目錄不存在，就會出現錯誤。

3. try-catch (捕捉錯誤)：

(1) 遇到錯誤時，自行決定要如何處理這個錯誤。

(2) 語法：

```
class 主程式 extends 物件 implements 介面, ……
{
    public static void main(String[] args)
    {
        try {
            可能遇到錯誤的內容;
        } catch(錯誤類型 A1 | 錯誤類型 A2 錯誤變數) {
            //錯誤變數會儲存錯誤訊息
            //可以自行決定要如何處理
            //多重錯誤的處理方法 A
        } catch(錯誤類型 B 錯誤變數) {
            //多重錯誤的處理方法 B
        }

        //出現錯誤訊息後，要繼續執行的內容;
    }
}
```

＊遇到錯誤後，一般來說會中斷程式。
但是 tyr-catch 可以讓程式繼續執行，
只要將後續程式寫在 try-catch 後面就可以了。
(Java 7 以後)

＊有多個錯誤類型時：
1) 寫在一個 catch 中，並用「|」分隔開來。
2) 寫成多個 catch，分開個別處裡。

(3) 範例：

```
import java.io.File;
import java.io.IOException;

public class t1 extends baba implements int1, int2 {

    public static void main(String args[]) {
        File file = new File("c:/abc/a.txt");

        try {
            file.createNewFile();
        } catch (IOException | ArithmeticException e) {
            System.out.println("錯誤: " + e);
        }

        System.out.println("try-catch 後繼續執行");
    }
}
```

【執行後訊息】

```
錯誤: java.io.IOException: 系統找不到指定的路徑。
try-catch 後繼續執行
```

十九、 泛型

「泛型」是一種「不定型態」的資料類型，當「宣告物件」時，再決定它的資料類型。

1. 建立單一泛型：

(1) 語法：

```
class 泛型物件<類型代號>
{
    //宣告變數
    類型代號 類型變數名稱;

    //建構式
    泛型物件(類型代號 參數A, 資料型態 參數B, ……)
    {
        this.類型變數名稱 = 參數A;
        //其他內容
    }
}
```

(2) 範例：

```
public class book<N> {
    N bookName;
    Integer bookPrice;

    book(N bookName, int bookPrice){
        this.bookName = bookName;
        this.bookPrice = bookPrice;
    }

    void getbook() {
        System.out.println("書名: " + bookName);
        System.out.println("售價: " + bookPrice);
    }
}
```

2. 建立多重泛型：

(1) 語法：

```
class 泛型物件<類型代號 A, 類型代號 B, ……>
{
    //宣告變數
    類型代號 A 類型變數名稱 A;
    類型代號 B 類型變數名稱 B;

    //建構式
    泛型物件(類型代號 A 參數 A, 類型代號 B 參數 B, 資料型態 參數, ……)
    {
        this.類型變數名稱 A = 參數 A;
        this.類型變數名稱 B = 參數 B;
        //其他內容
    }
}
```

(2) 範例：

```
public class cat<N, A> {
    N catName;
    A catAge;

    cat(N catName, A catAge) {
        this.catName = catName;
        this.catAge = catAge;
    }

    void getCat() {
        System.out.println("名稱: " + catName);
        System.out.println("年齡: " + catAge);
    }
}
```

3. 使用泛型：

(1) 語法：

```
泛型物件<資料類型, ……> 物件變數 = new 泛型物件<>(參數, ……);
```

只能使用包覆類別

如：Integer、String、Double、……

(2) 範例：

```
public class t1 extends baba implements int1, int2 {

    public static void main(String args[]) {
            book<String> b1 = new book<>("javaSE", 500);
            cat<String, Integer> c1 = new cat<>("nono", 5);

            b1.getbook();
            c1.getCat();
    }
}
```

＊book 的不定型 N 宣告為 String，故宣告時第一個參數需為 String。

＊cat 的不定型 N 宣告為 String，故宣告時第一個參數需為 String。
不定型 A 宣告為 Integer，故宣告時第一個參數需為 Integer。

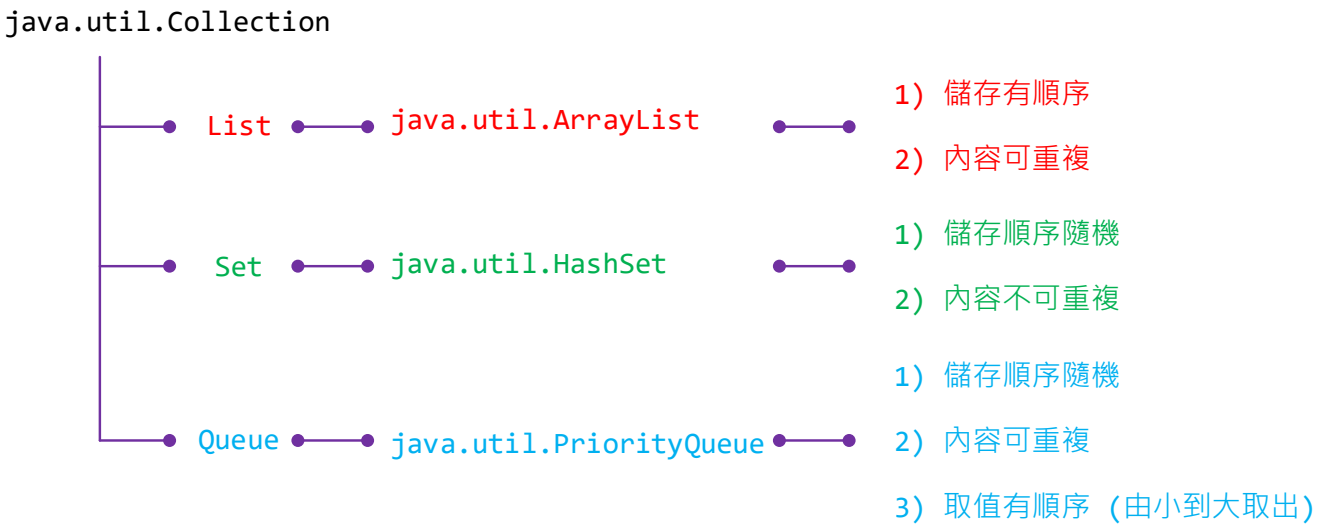
二十、 集合

集合(Collection)是泛型的一種，其都繼承 java.util.Collection。

當我們要用陣列儲存「不確定數量的」數據資料時，就會使用到集合。

換句話說，集合就是不需要事先決定長度的「動態陣列」。

1. 常見的集合的種類：



2. ArrayList：

List 的一種，特性為「儲存有順序」且「資料可重複」。

使用時需要 import java.util.ArrayList 和 java.util.List。

(1) 宣告語法：

```
List<資料類型> ArrayList 變數 = new ArrayList<>();
```

＊資料類型必須為包覆類型。

＊初始長度預設為 10 個，會自動檢查夠不夠用，且會自動添加。

＊如果要於 非同步中 進行 add() 加入資料時，
建議使用
```
List<資料類型> ArrayList 變數 = new CopyOnWriteArrayList<>();
```
以避免資料缺失。

(2) 常用方法：

a. ArrayList 變數.add(value)：

新增資料

範例：
```
arraylist.add("1");
```

b. ArrayList 變數.remove(index)：

刪除特定資料 (後面資料往前遞補)

範例：
```
arraylist.remove(2);
```

c. ArrayList 變數.set(index, value)：

修改資料

範例：
```
arraylist.set(3, "5");
```

d. ArrayList 變數.get(index)：

取出資料

範例：

```
System.out.println(arraylist.get(0));
```

e. ArrayList 變數.size()：

查詢長度

範例：

```
int length = arraylist.size();
```

f. ArrayList 變數.indexOf(value)：

查詢指定 value 的 index (回傳第一個找到的 index)

範例：

```
int index = arraylist.indexOf("1");
```

g. ArrayList 變數.toArray(陣列變數)：

將資料轉存成陣列

＊陣列變數的「資料型態」和「長度」需要與「ArrayList 變數」相同。

範例：

```
ArrayList<String> arraylist = new ArrayList<>();
……
int length = arraylist.size();
String[] array = new String[length];
arraylist.toArray(array);
```

h. ArrayList 變數.contains(value)：

查詢是否包含指定資料，回傳 True(是) / False(否)

範例：

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> addressList = new ArrayList<>();
        addressList.add("台北");
        addressList.add("台中");
        addressList.add("台南");
        addressList.add("高雄");
        addressList.add("屏東");
        addressList.add("花蓮");

        Boolean chkInd1 = addressList.contains("台北");
        Boolean chkInd2 = addressList.contains("澎湖");

        System.out.println("chkInd1 檢核結果: " + chkInd1); //true
        System.out.println("chkInd2 檢核結果: " + chkInd2); //false
    }
}
```

(3) 範例：

```
import java.util.List;
import java.util.ArrayList;

public class t1 {
    public static void main(String args[]) {
        //宣告
        ArrayList<String> arraylist = new ArrayList<>();
        //新增
        arraylist.add("1");
        arraylist.add("2");
        arraylist.add("3");
        arraylist.add("4");
        //刪除
        arraylist.remove(2);
        //修改
        arraylist.set(2, "5");
        //長度
        int length = arraylist.size();
        //取出
        for (int i = 0 ; i < length ; i++) {
            String value = arraylist.get(i);
            System.out.println("index: "   + i +
                               ", value: " + value);
        }
        //轉存成陣列
        String[] array = new String[length];
        arraylist.toArray(array);
    }
}
```

3. HashSet：

Set 的一種，特性是「隨機儲存」且「資料不重複」。

使用時需要 import java.util.HashSet 和 java.util.Set。

(1) 宣告語法：

```
Set<資料類型> HashSet 變數 = new HashSet <>();
```

＊資料類型必須為包覆類型。

(2) 常用方法：

a. HashSet 變數.add(value)：

新增資料

範例：

```
hashset.add(1);
```

b. HashSet 變數.remove(value)：

刪除指定資料

範例：

```
hashset.remove(1);
```

c. HashSet 變數.contains(value)：

查詢是否包含指定資料，回傳 True(是) / False(否)

範例：

```
boolean exist = hashset.contains(3);
```

d. HashSet 變數.size()：

查詢長度

範例：

```
int length = hashset.size();
```

e. HashSet 變數.toArray(陣列變數)：

將資料轉存成陣列

＊陣列變數的「資料型態」和「長度」需要與「HashSet 變數」相同。

範例：

```
Set<Integer> hashset = new HashSet<>();
……
int length = hashset.size();
Integer[] array = new Integer[length];
hashset.toArray(array);
```

f. 取出資料：

利用 for-each 來遍歷 HashSet 取出資料。

範例：

```
for (Integer i : hashset) {
      System.out.println(i);
}
```

(3) 範例：

```
import java.util.HashSet;
import java.util.Set;

public class t1 {

    public static void main(String args[]) {
            Set<Integer> hashset = new HashSet<>();
            //新增
            hashset.add(1);
            hashset.add(2);
            hashset.add(3);
            //刪除
            hashset.remove(2);
            //查詢
            boolean exist = hashset.contains(3);
            System.out.println("是否包含 3: " + exist);
            //長度
            int length = hashset.size();
            //遍歷讀取
            for (Integer i : hashset) {
                System.out.println(i);
            }
            //轉存陣列
            Integer[] array = new Integer[length];
            hashset.toArray(array);
            for (Integer i : array) {
                System.out.println(i);
            }
    }
}
```

4. PriorityQueue：

Queue 的一種，特性是「儲存順序隨機」、「資料可重複」和「由小到大取出資料」。

使用時要 import java.util.PriorityQueue。

(1) 宣告語法：

```
PriorityQueue <資料類型> PriorityQueue 變數 = new PriorityQueue <>();
```

＊資料類型必須為包覆類型。

(2) 常用方法：

a. PriorityQueue 變數.add(value)：

新增資料

範例：

```
priority.add(1);
```

b. PriorityQueue 變數.remove(value)：

刪除指定資料，若有多筆相同資料，刪除排序最前的一個

範例：

```
priority.remove(1);
```

c. PriorityQueue 變數.contains(value)：

是否包含指定 value，回傳 True(是) / Fales(否)

範例：

```
boolean exist = priority.contains(1);
```

d. PriorityQueue 變數.size()：

查詢長度

範例：

```
int length = priority.size();
```

e. PriorityQueue 變數.poll()：

讀取並移除資料，順序為「由小到大」，若無資料回傳 Null

範例：

```
Integer i;
while ((i = priority.poll()) != null) {
     System.out.println(i);
}
```

f. PriorityQueue 變數.toArray(陣列變數)：

將資料轉存成陣列

＊陣列變數的「資料型態」和「長度」需要與「PriorityQueue 變數」相同。

＊其轉存出來的資料為「隨機排序」。

範例：

```
PriorityQueue<Integer> priority = new PriorityQueue<>();
Integer[] array = new Integer[length];
priority.toArray(array);
for (Integer i : array) {
     System.out.println(i);
}
```

(3) 範例：

```
import java.util.PriorityQueue;
import java.util.Queue;

public class t1 {
    public static void main(String args[]) {
        PriorityQueue<Integer> priority = new PriorityQueue<>();
        //新增
        priority.add(4);
        priority.add(2);
        priority.add(1);
        priority.add(2);
        priority.add(3);
        //查詢
        boolean exist = priority.contains(2);
        System.out.println(exist);
        System.out.println("--------------");
        //刪除
        priority.remove(2);
        //長度
        int length = priority.size();
        //轉存陣列
        Integer[] array = new Integer[length];
        priority.toArray(array);
        for (Integer i : array) {
            System.out.println(i);
        }
        //讀取並刪除資料
        System.out.println("--------------");
        Integer i;
        while ((i = priority.poll()) != null) {
            System.out.println(i);
        }
    }
}
```

二十一、 Map<K, V>

Map 是泛型的一種，使用 Key-Value 的形式來保存資料，透過不重複的 Key 來存取 Value。

1. Key 區：

   索引區，資料型態為「String」。

   其保存的內容「不可重複」。

2. Value 區：

   值區，資料型態為「包覆類別」。

   其保存的內容「可以重複」。

3. HashMap：

   (1) 特點：

   用「亂數排序」的方式來儲存 Key 區。

   (2) 宣告語法：

   ```
   Map<String, 包覆類別> HashMap 變數 = new HashMap<>();
   ```

   (3) 常用方法：

   a. HashMap 變數.put("key", value)：

   新增資料

   範例：

   ```
   hashmap.put("a", 1);
   ```

   b. HashMap 變數.remove("key")：

   刪除指定資料

   範例：

   ```
   hashmap.remove("a ");
   ```

c. HashMap 變數.get("key")：

讀取指定資料，無此 Key 會回傳 Null

範例：

```
Integer value = hashmap.get("a");
```

d. HashMap 變數.size()：

查詢長度

範例：

```
int length = hashmap.size();
```

e. HashMap 變數.keySet()：

輸出 key 區資料，資料型態為 Set

範例：

```
Set<String> keys = hashmap.keySet();
```

(4) 範例：

```
import java.util.Map;
import java.util.Set;
import java.util.HashMap;

public class t1 extends baba implements int1, int2 {
      public static void main(String args[]) {
          Map<String, Integer> hashmap = new HashMap<>();
          //新增
          hashmap.put("a", 1);
          hashmap.put("b", 1);
          hashmap.put("c", 2);
          hashmap.put("d", 3);
          //刪除
          hashmap.remove("c");
          //長度
          int length = hashmap.size();
          //輸出 key
          Set<String> keys = hashmap.keySet();
          //讀取 value
          for (String s : keys) {
              Integer value = hashmap.get(s);
              System.out.println("key: " + s +
                                 ", value: " + value);
          }
      }
}
```

二十二、　foreach 方法和 Lambda 表達式

「foreach 方法」和「Lambda 表達式」是 Java 8 新增的功能，
其作用是簡化 List、Set、Queue 和 Map 等集合中，使用 for-each 時的程式碼。

1. 語法：

```
集合變數.foreach(Lambda 表達式);
```

2. Lambda 表達式：
   (1) 語法結構：

   a. 一般型態：

```
集合變數中的每一個元素，一格有幾個變數就要有幾個參數。
↓                          如：List 為 x ；Map 為 (x, y)。
參數 -> 內容;
        ↑
        每次要執行的內容，會在其中使用參數來代指集合變數的每一個元素。

參數 -> {
            內容 A;
            內容 B;
            ……
        }
```

   b. 方法引用：

```
儲存資料用的物件類
↓
類名稱::方法名
        ↑
        「類」中的「getting 方法」
```

   ＊Lambda 表達式的更精簡寫法，代表調用「類」中的「某個方法」。

(2) 無參數的一般型態：

```
() -> 內容;
() -> {
            內容 A;
            內容 B;
            ……
      }
```

(3) 單一參數的一般型態：

```
x -> 內容;
x -> {
            內容 A;
            內容 B;
            ……
      }
```

(4) 多參數的一般型態：

```
(x, y, ……) -> 內容;
(x, y, ……) -> {
                  內容 A;
                  內容 B;
                  ……
               }
```

範例：

```java
public class t1 {
    public static void main(String[] args) {
       List<Integer> list = new ArrayList<>(1, 2, 3);
       list.add(1);
       list.add(2);
       list.add(3);
       //單一參數 - 單一內容
       list.forEach(ls -> System.out.println(ls));
       //單一參數 - 多內容
       list.forEach(ls -> {
          ls = ls * 2;
          System.out.println(ls);
       });
    }
 }
```

二十三、 stream API

stream API 是 Java 8 新加入的功能，可以對集合變數中的元素進行操作，以減少程式碼的撰寫與優化執行效率。
此 API 有 單執行緒(stream) 和 多執行緒(parallelStream) 兩種版本，
在此將先以單執行緒版進行介紹。

1. filter (過濾器)：

(1) 功能：

對集合中的元素進行過濾，輸出符合條件的元素。

(2) 語法：

依照輸出元素的數量，可以分成 2 大類：

a. 單一元素：

```
元素類型 變數 = 集合變數.stream().filter(變數 -> 通過條件)
                        .findFirst()      ← 輸出第一個元素
                        .orElse(輸出值); ← 無符合時的輸出值
                                            通常用 null
```

b. 集合：

```
集合類型 變數 = 集合變數.stream().filter(變數 -> 通過條件)
                        .collect(輸出的集合類型);
                                 ↑
                                 如：Collectors.toList()
                                     Collectors.toSet()
```

(3) 範例：

```java
List<Integer> L1 = Arrays.asList(1,2,3,4,5,6);
// 過濾器(1): 取得符合條件的第一個元素
Integer output1;
output1 = L1.stream().filter(x -> x % 2 == 0)
                     .findFirst().orElse(null);
System.out.println(output1);    // 2
// 過濾器(2): 將所有符合條件的元素存成 List
List<Integer> output2;
output2 = L1.stream().filter(x -> x % 2 == 0)
                     .collect(Collectors.toList());
System.out.println(output2);    // [2, 4, 6]
```

2. sorted (排序)：

(1) 功能：

對集合內的元素進行排序，預設由小到大，但也可以自訂排序方式，如：由大到小。

(2) 語法：

a. 由小到大：

```
集合類型 變數 = 集合變數.stream()
                        .sorted()      ← 集合只有一個欄位
                        .collect(輸出的集合類型);
```

```
                                                用 Lambda 方法引用
                                    集合有多個欄位      設定比較的元素
集合類型 變數 = 集合變數.stream()          ↓                  ↓
                        .sorted(Comparator.comparing(排序元素))
                        .collect(輸出的集合類型);
```

b. 由大到小：

```
集合類型 變數 = 集合變數.stream()          ↓ 集合只有一個欄位
                       .sorted(Comparator.reverseOrder())
                       .collect(輸出的集合類型);
```

```
                                              用 Lambda 方法引用
                                    集合有多個欄位    設定比較的元素
集合類型 變數 = 集合變數.stream()       ↓          ↓
                .sorted(Comparator.comparing(排序元素).reversed())
                .collect(輸出的集合類型);
                         ↑
                         如：Collectors.toList()
                             Collectors.toSet()
```

(3) 範例 1：單一元素 的排序

```
List<Integer> L2 = Arrays.asList(1,5,2,6,4,3);
// 排序(1): 由小到大排序，不用加排序條件
List<Integer> output3;
output3 = L2.stream().sorted()
                     .collect(Collectors.toList());
System.out.println(output3);    // [1, 2, 3, 4, 5, 6]

// 排序: 由大到小排序，使用 Comparator.reverseOrder()
output3 = L2.stream().sorted(Comparator.reverseOrder())
                     .collect(Collectors.toList());
System.out.println(output3);    // [6, 5, 4, 3, 2, 1]
```

(4) 範例 2：多元素-指定特定元素的排序

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class streamTest {

    public static void main(String[] args) {
        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員B", 16));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 14));
        userList.add(new User("學員E", 18));

        System.out.println("年齡由小到大排序");
        List<User> userSort1 = userList.stream()
                                .sorted(Comparator.comparing(User::getAge))
                                .collect(Collectors.toList());
        for(User user : userSort1) {
            System.out.println(user);
        }
        System.out.println("年齡由大到小大排序");
        List<User> userSort2 = userList.stream()
                                .sorted(Comparator.comparing(User::getAge).reversed())
                                .collect(Collectors.toList());
        for(User user : userSort2) {
            System.out.println(user);
        }
    }

}
```

```
<terminated> streamTest [Ja
年齡由小到大排序
[學員A 10]
[學員C 12]
[學員D 14]
[學員B 16]
[學員E 18]
年齡由大到小大排序
[學員E 18]
[學員B 16]
[學員D 14]
[學員C 12]
[學員A 10]
```

3. map（映射）：

(1) 功能：

取出集合變數中，符合指定規則的元素。

(2) 語法：

```
                                                    去除重複
                                                    ↓
集合類型 變數 = 集合變數.stream().map(取出規則).distinct()
                                                    .collect(輸出的集合類型);
                                                    ↑
                                                    如：Collectors.toList()
                                                        Collectors.toSet()
```

(3) 範例：

```java
public class main {
    public static void main(String[] args) {
        List<User> data = new ArrayList<>();
        data.add(new User("a",50));
        data.add(new User("a",40));
        data.add(new User("b",60));
        data.add(new User("e",10));
        data.add(new User("d",30));

        // 取出 data 中的全部不重複的 Name
        List<String> output = data.stream().map(User::getName)
                                  .distinct()
                                  .collect(Collectors.toList());
        System.out.println(output.toString()); // [a, b, e, d]
    }
}
```

4. sum (加總)：

(1) 功能：

針對集合變數中，各個物件的某個數值進行加總。

(2) 語法：

```
                                                類型必須是 int
                                                ↓
int 變數 = 集合變數.stream().mapToInt(加總元素).sum();

                                                    類型必須是 long
                                                    ↓
long 變數 = 集合變數.stream().mapToLong(加總元素).sum();

                                                        類型必須是 Double
                                                        ↓
double 變數 = 集合變數.stream().mapToDouble(加總元素).sum();
                                                        ↑
                                                        使用「方法引用」
                                                        設定要加總的元素
```

(3) 範例：

```java
public class main {
    public static void main(String[] args) {
        List<User> data = new ArrayList<>();
        data.add(new User("a",50, 1, 2.5));
        data.add(new User("e",10, 2, 1.5));
        data.add(new User("d",30, 4, 6.5));

        //加總(sum):
        int sumInt = data.stream().mapToInt(User::getAge).sum();
        System.out.println(sumInt);  //90

        long sumLong = data.stream().mapToLong(User::getChg).sum();
        System.out.println(sumLong);  //7

        double sumDouble = data.stream().mapToDouble(User::getEng).sum();
        System.out.println(sumDouble);  //10.5
    }
}
```

5. max（最大值）：

(1) 功能：

取出集合變數中，各個物件內指定元素最大的物件。

(2) 語法：

```
物件類型 變數 = 集合變數.stream()
                        .max(Comparator.comparing(要比較元素)).get();
                                                  ↑
                                                  使用「方法引用」
                                                  設定要比較的元素
```

(3) 範例：

```java
public class main {
    public static void main(String[] args) {
        List<User> data = new ArrayList<>();
        data.add(new User("a", 50, 1, 2.5));
        data.add(new User("e", 10, 2, 1.5));
        data.add(new User("d", 30, 4, 6.5));

        User maxObject;
        /* 1 */
        maxObject = data.stream()
                        .max(Comparator.comparing(User::getChg)).get();
        Long maxChg = maxObject.getChg();
        System.out.println(maxChg); // 50
        /* 2 */
        maxObject = data.stream()
                        .max(Comparator.comparing(User::getName)).get();
        String maxName = maxObject.getName();
        System.out.println(maxName); // e
    }
}
```

6. min（最小值）：

(1) 功能：

取出集合變數中，各個物件內指定元素最小的物件。

(2) 語法：

```
物件類型 變數 = 集合變數.stream()
                        .min(Comparator.comparing(要比較元素)).get();
                                                  ↑
                                                  使用「方法引用」
                                                  設定要比較的元素
```

(3) 範例：

```java
public class main {
    public static void main(String[] args) {
        List<User> data = new ArrayList<>();
        data.add(new User("a", 50, 1, 2.5));
        data.add(new User("e", 10, 2, 1.5));
        data.add(new User("d", 30, 4, 6.5));

        User minObject;
        /* 1 */
        minObject = data.stream()
                        .min(Comparator.comparing(User::getAge)).get();
        int minAge = minObject.getAge();
        System.out.println(minAge); // 10
        /* 2 */
        minObject = data.stream()
                        .min(Comparator.comparing(User::getName)).get();
        String minName = minObject.getName();
        System.out.println(minName); // a
    }
}
```

7. foreach（迴圈）：

(1) 功能：

針對集合變數中的元素，進行 foreach。

(2) 語法：

```
集合變數.stream().foreach(變數 -> {
    內容;                                    ← 此為匿名類別
});
```

(3) 範例：

```
4  public class t1 {
5      public static void main(String[] args) {
6          // 取 1 ~ 10
7          List<Integer> intList = new ArrayList<>();
8          for(int i = 1 ; i <= 10 ; i++) {
9              intList.add(i);
10         }
11
12         // 透過 foreach 輸出
13         intList.stream().forEach(i -> {
14             System.out.println(i);
15         });
16     }
17 }
```

```
Console × Problems Progress Debug Shell
<terminated> t1 [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (2023年6月5日 下午8:26:06)
1
2
3
4
5
6
7
8
9
10
```

8. isPresent (是否存在)

(1) 功能：

若要判斷 List 中，是否存在某個值，可以使用此方法達成。

A. 透過 filter　　篩選條件

B. 透過 findAny　　設定輸出全部元素

C. 透過 isPresent 判斷是否存在:

數量 = 0 : 不存在 : 回傳 false

數量 > 0 : 存　在 : 回傳 true

(2) 語法

```
Boolean 布林變數 = 集合變數.stream().foreach(變數 -> 通過條件)
                                    .findAny()      ← 輸出全部元素
                                    .isPresent();   ← 是否存在
                                                      true  = 存在
                                                      false = 不存在
```

(3) 範例

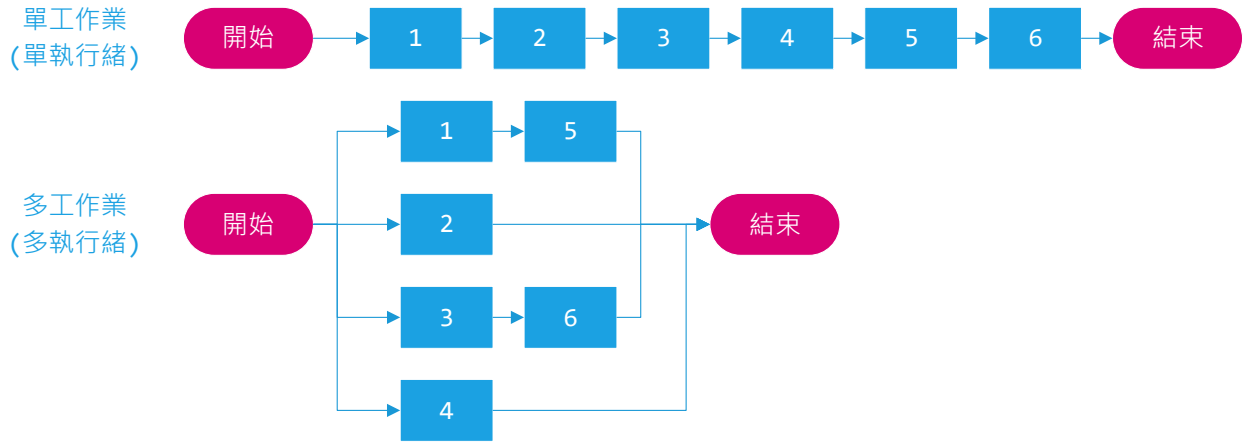
```java
import java.util.ArrayList;

public class StreamTest {

    public static void main(String[] args) {
        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員B", 16));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 14));

        Boolean age12Exists = userList.stream().filter(user -> user.getAge() == 12)
                                               .findAny().isPresent();
        System.out.println("是否存在12歲的學員:" + age12Exists);
    }
}
```

```
Console × Problems Progress Debug Shell
<terminated> streamTest [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (2023年8月10日 下午8:35:54)
是否存在12歲的學員: true
```

9. 單執行緒 (stream) 與 多執行緒 (parallelStream)：

parallelStream 為 stream 的 多執行緒 版本，兩者可用的方法皆相同。

如果 需要處理的運算龐大，可以考慮使用『parallelStream』。
將可以有效地節省時間。



範例：
進行 1～1 千萬 相加的作業
stream 耗時 65 毫秒
parallelStream 耗時 36 毫秒，此方法明顯效率較好。

```java
public class t1 {
    public static void main(String[] args) {
        List<Long> L1 = new ArrayList<>();
        for(long i=1;i<=10000000;i++) {
            L1.add(i);
        }

        System.out.println("*** stream啟動 ***");
        long start1 = System.currentTimeMillis();
        Long output1 = L1.stream().mapToLong(Long::longValue).sum();
        System.out.println("sum=" + output1);
        System.out.println("耗時：" + (System.currentTimeMillis()-start1) + "毫秒");

        System.out.println();

        System.out.println("*** parallelStream啟動 ***");
        long start2 = System.currentTimeMillis();
        Long output2 = L1.parallelStream().mapToLong(Long::longValue).sum();
        System.out.println("sum=" + output2);
        System.out.println("耗時：" + (System.currentTimeMillis()-start2) + "毫秒");

    }
}
```

```
Console × Problems Progress Debug Shell
<terminated> t1 [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (2023年6月5日 下午7:49:26)
*** stream啟動 ***
sum=50000005000000
耗時：65毫秒

*** parallelStream啟動 ***
sum=50000005000000
耗時：36毫秒
```

二十四、　集合變數的排序

JAVA 對於集合變數的排序，不只有一種方法，
除了上述介紹的『stream().sorted()』　　(需要另外導出來)，
『Collections.sort()』和『集合變數.sort()』也是常見的方法。(直接改集合的順序)

1. 單一元素的排序：

(1) 方法一：

由小到大：

```
Collections.sort(集合變數, Comparator.comparing(排序元素));
```

由大到小：

```
Collections.sort(集合變數,
                 Comparator.comparing(排序元素).reversed());
```

(2) 方法二：

由小到大：

```
集合變數.sort(Comparator.comparing(排序元素));
```

由大到小：

```
集合變數.sort(Comparator.comparing(排序元素).reversed());
```

範例 1：使用『Collections.sort()』

```java
import java.util.ArrayList;

public class SortTest {

    public static void main(String[] args) {

        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員B", 16));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 14));

        // 使用一般的Lambda表達式
        // 由小到大對 User age 排序
        Collections.sort(userList, Comparator.comparing(user -> user.getAge()));
        userList.forEach(user -> {
            System.out.println(user.getName() + " " + user.getAge() + "歲");
        });

        System.out.println();

        // 使用Lambda的方法引用
        // 由大到小對 User age 排序
        Collections.sort(userList, Comparator.comparing(User::getAge).reversed());
        userList.forEach(user -> {
            System.out.println(user.getName() + " " + user.getAge() + "歲");
        });


    }

}
```

```
<terminated> SortTest [
學員A 10歲
學員C 12歲
學員D 14歲
學員B 16歲

學員B 16歲
學員D 14歲
學員C 12歲
學員A 10歲
```

範例 2：使用『集合變數.sort()』

```java
import java.util.ArrayList;

public class SortTest {

    public static void main(String[] args) {

        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員B", 16));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 14));

        // 由小到大對 User age 排序
        userList.sort(Comparator.comparing(User::getAge));
        userList.forEach(user -> {
            System.out.println(user.getName() + " " + user.getAge() + "歲");
        });

        System.out.println();

        // 由大到小對 User age 排序
        userList.sort(Comparator.comparing(User::getAge).reversed());
        userList.forEach(user -> {
            System.out.println(user.getName() + " " + user.getAge() + "歲");
        });


    }

}
```

```
<terminated> Sor
學員A 10歲
學員C 12歲
學員D 14歲
學員B 16歲

學員B 16歲
學員D 14歲
學員C 12歲
學員A 10歲
```

2. 兩個元素的排序：

如果要「先依照 A 元素排序」後，「再依照 B 元素排序」
那麼需要先個別設定元素排序方法後，再透過『thenComparing』來執行

```
//1. 設定排序元素: 各別元素 由小到大 或 由大到小 於這裡設定
Comparator<變數類型> 排序變數 A = Comparator.comparing(排序元素 A); ← 由小到大
Comparator<變數類型> 排序變數 B = Comparator.comparing(排序元素 B).reversed();
                                                                ↑ 由大到小

// 2. 進行排序: 先依照 A 排序，再依照 B 排序
Collections.sort(集合變數, 排序變數 A.thenComparing(排序變數 B));
```

範例 1：Collections.sort() 的案例示範

```java
import java.util.ArrayList;

public class streamTest {

    public static void main(String[] args) {
        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員F", 12));
        userList.add(new User("學員B", 16));
        userList.add(new User("學員E", 18));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 12));
        ////////////////////////////////////////////////////////////////////////////////
        // 多欄位的排序 要先設定個別欄位的排序
        Comparator<User> userSortByAge1  = Comparator.comparing(User::getAge);
        Comparator<User> userSortByName1 = Comparator.comparing(User::getName).reversed();

        System.out.println("年齡由小到大排序，姓名由大到小排序");
        Collections.sort(userList, userSortByAge1.thenComparing(userSortByName1));
        for(User user : userList) {
            System.out.println(user);
        }
        ////////////////////////////////////////////////////////////////////////////////
        Comparator<User> userSortByAge2 = Comparator.comparing(User::getAge).reversed();
        Comparator<User> userSortByName2 = Comparator.comparing(User::getName).reversed();

        System.out.println("年齡由大到小大排序，姓名由大到小排序");
        Collections.sort(userList, userSortByAge2.thenComparing(userSortByName2));
        for(User user : userList) {
            System.out.println(user);
        }
    }

}
```

```
<terminated> streamTest [Java Application] C:\Pro
年齡由小到大排序，姓名由大到小排序
[學員A 10]
[學員F 12]
[學員D 12]
[學員C 12]
[學員B 16]
[學員E 18]
年齡由大到小大排序，姓名由大到小排序
[學員E 18]
[學員B 16]
[學員F 12]
[學員D 12]
[學員C 12]
[學員A 10]
```

範例 2：stream().sort() 也可以做到兩個元素的排序

```java
import java.util.ArrayList;

public class streamTest {

    public static void main(String[] args) {
        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員F", 12));
        userList.add(new User("學員B", 16));
        userList.add(new User("學員E", 18));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 12));
        ////////////////////////////////////////////////////////////////////////////////
        // 多欄位的排序 要先設定個別欄位的排序
        Comparator<User> userSortByAge1  = Comparator.comparing(User::getAge);
        Comparator<User> userSortByName1 = Comparator.comparing(User::getName).reversed();

        System.out.println("年齡由小到大排序，姓名由大到小排序");
        List<User> userList1 = userList.stream()
                                       .sorted(userSortByAge1.thenComparing(userSortByName1))
                                       .collect(Collectors.toList());
        for(User user : userList1) {
            System.out.println(user);
        }
        ////////////////////////////////////////////////////////////////////////////////
        Comparator<User> userSortByAge2 = Comparator.comparing(User::getAge).reversed();
        Comparator<User> userSortByName2 = Comparator.comparing(User::getName).reversed();

        System.out.println("年齡由大到小大排序，姓名由大到小排序");
        List<User> userList2 = userList.stream()
                                       .sorted(userSortByAge2.thenComparing(userSortByName2))
                                       .collect(Collectors.toList());
        for(User user : userList2) {
            System.out.println(user);
        }
    }

}
```

```
<terminated> streamTest [Java Application] C:\Progr
年齡由小到大排序，姓名由大到小排序
[學員A 10]
[學員F 12]
[學員D 12]
[學員C 12]
[學員B 16]
[學員E 18]
年齡由大到小大排序，姓名由大到小排序
[學員E 18]
[學員B 16]
[學員F 12]
[學員D 12]
[學員C 12]
[學員A 10]
```

3. 含有 null 的排序：

如果 集合變數 中，排序元素 的值 可能存在 Null，
為了避免出現 NullPointException，
需要在使用『Comparator.comparing()』設定 排序元素 的同時，
設定要如何處理 Null

語法： 以『集合變數.sort()』為範例

(1) Null 放最前面

```
集合變數.sort(Comparator.comparing(排序元素,
                                   Comparator.nullsFirst(String::compareTO)));
```

(2) Null 放最後面

```
集合變數.sort(Comparator.comparing(排序元素,
                                   Comparator.nullsLast(String::compareTo)));
```

範例：

```
public class SortTest2 {

    public static void main(String[] args) {
        List<User> userList = new ArrayList<>();
        userList.add(new User("學員A", 10));
        userList.add(new User("學員B", 16));
        userList.add(new User(null, 18));
        userList.add(new User("學員C", 12));
        userList.add(new User("學員D", 14));

        // 含有 null 的排序
        // 1. nullsFirst: null 放前面
        userList.sort(Comparator.comparing(User::getName, Comparator.nullsFirst(String::compareTo)));
        userList.forEach(user -> {
            System.out.println(user.getName() + " " + user.getAge() + "歲");
        });

        System.out.println();

        // 2. nullsLast: null 放後面
        userList.sort(Comparator.comparing(User::getName, Comparator.nullsLast(String::compareTo)));
        userList.forEach(user -> {
            System.out.println(user.getName() + " " + user.getAge() + "歲");
        });

    }

}
```

```
null 18歲
學員A 10歲
學員B 16歲
學員C 12歲
學員D 14歲

學員A 10歲
學員B 16歲
學員C 12歲
學員D 14歲
null 18歲
```

二十五、 存取檔案(I/O)

Java 中對於檔案的操作都是屬於 java.io.*的內容，在使用時都需要處理 IOException。

1. 新增與刪除：

(1) 宣告物件：

```
File 檔案變數 = new File("檔案或資料夾路徑");
```

(2) 常用方法：

a. 檔案變數.exists()：

判斷檔案(資料夾)是否存在，回傳 True(存在) / False(不存在)

範例：

```
File file = new File("a.txt");
boolean fileExist = file.exists();
```

b. 檔案變數.createNewFile()：

新增檔案

範例：

```
File file = new File("a.txt");
file.createNewFile();
```

c. 檔案變數.mkdir()：

新增資料夾

範例：

```
File dir = new File("a");
dir.mkdir();
```

d. 檔案變數.delete()：

刪除檔案(資料夾)

範例：

```
File file = new File("a.txt");
file.delete();
```

(3) 範例：

```
import java.io.File;
import java.io.IOException;

public class t1 {
    public static void main(String args[]) throws IOException {
        File file = new File("a.txt");
        //檔案是否存在
        boolean fileExist = file.exists();
        System.out.println(fileExist);
        //新增檔案
        if(!fileExist) {
            file.createNewFile();
            System.out.println("檔案新增成功!");
        }
        //刪除檔案
        if(fileExist) {
            file.delete();
            System.out.println("檔案刪除成功!");
        }

        File dir = new File("a");
        //資料夾是否存在
        boolean dirExist = dir.exists();
        System.out.println(dirExist);
        //新增資料夾
        if(!dirExist) {
            dir.mkdir();;
            System.out.println("資料夾新增成功!");
        }
        //刪除資料夾
        if(dirExist) {
            dir.delete();;
            System.out.println("資料夾刪除成功!");
        }
    }
}
```

2. 寫入：

(1) 宣告物件：

```
FileWriter 檔案路徑變數 = new FileWriter("檔案路徑");
BufferedWriter 寫入變數 = new BufferedWriter(檔案路徑變數);
```

＊由於 BufferedWriter 無法讀取檔案，
故先透過 FileWriter 讀取後，在串接 BufferedWriter 給使用。

(2) 常用方法：

a. 寫入變數.write("要寫入的字串")：

將字串寫入檔案中

＊如果寫入的內容需要換行，則需要在字串中加入「\n」換行符號。

＊若檔案已存在，則寫入時會將原有內容覆蓋。

範例：

```
FileWriter file = new FileWriter("a.txt");
BufferedWriter bw = new BufferedWriter(file);
bw.write("這是第一行");
bw.write("\n 換行了");
```

b. 寫入變數.close()：

釋放寫入資源，寫入時要下達此指令，才會將寫入內容保存到檔案中。

範例：

```
FileWriter file = new FileWriter("a.txt");
BufferedWriter bw = new BufferedWriter(file);
……
bw.close();
```

(3) 範例：

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class t1 {
    public static void main(String args[]) throws IOException {
        //確認檔案是否存在，不存在就建立檔案
        File chk_file = new File("a.txt");
        if(!chk_file.exists()) {
            chk_file.createNewFile();
        }
        //宣告寫入的物件變數
        FileWriter file = new FileWriter("a.txt");
        BufferedWriter bw = new BufferedWriter(file);
        //寫入
        bw.write("這是第一行");
        bw.write("，繼續寫入不會換行");
        bw.write("\n 換行了");
        //寫入結束
        bw.close();
    }
}
```

3. 讀取：

(1) 宣告物件：

```
FileReader 檔案路徑變數 = new FileReader("檔案路徑");
BufferedReader 讀取變數 = new BufferedReader(檔案路徑變數);
```

＊由於 BufferedReader 無法讀取檔案，
故先透過 FileReader 讀取後，在串接 BufferedReader 給使用。

(2) 常用方法：

a. 讀取變數.readLine：

從檔案中讀取一行的文字內容，若沒有內容則回傳 Null

＊若文字中有多行文字，需要搭配 while 來讀取全部內容。

＊若要將檔案中的所有文字保存到 String 中，則需要手動加入換行記號 "\n"。

範例：

```
FileReader file = new FileReader("a.txt");
BufferedReader br = new BufferedReader(file);
String read;
while((read = br.readLine()) != null) {
    System.out.println(read);
}
```

b. 讀取變數.close()：

釋放讀取資源

範例：

```
FileReader file = new FileReader("a.txt");
BufferedReader br = new BufferedReader(file);
……
br.close();
```

(3) 範例：

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class t1 {
    static String word = "";
    public static void main(String args[]) throws IOException {
        //宣告讀取變數
        FileReader file = new FileReader("a.txt");
        BufferedReader br = new BufferedReader(file);
        //逐行讀取
        String read;
        while((read = br.readLine()) != null) {
            word = word + read + "\n";
        }
        System.out.println(word);
        //釋放資源
        br.close();
    }
}
```

4. 複製檔案、刪除檔案：

(1) 建立來源檔：

```
File 來源檔變數 = new File("來源檔路徑");
```

範例：

```
File input = new File("a.txt");
```

(2) 建立目標檔：

```
File 目標檔變數 = new File("目標檔路徑");
```

範例：

```
File output = new File("b.txt");
```

(3) 取得 Path 的方法：

```
Path 來源路徑變數 = 來源檔變數.toPath();
Path 目標路徑變數 = 目標檔變數.toPath();
```

範例：

```
Path inputPath = input.toPath();
Path outputPath = output.toPath();
```

(4) 複製檔案：

```
Files.copy(來源路徑變數, 目標路徑變數);
```

範例：

```
Files.copy(inputPath, outputPath);
```

(5) 刪除檔案：

```
Files.delete(刪除檔案的路徑變數);
```

範例：

```
Files.delete(output.toPath());
```

(6) 範例：

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class t1 {
    public static void main(String args[]) throws IOException,
                                                   InterruptedException {
        //建立來源檔、目標檔
        File input = new File("a.txt");
        File output = new File("b.txt");
        //path
        Path inputPath = input.toPath();
        Path outputPath = output.toPath();
        //檔案複製
        Files.copy(input.toPath(), output.toPath());
        System.out.println("複製");
        //暫停 10 秒
        Thread.sleep(10000);
        //檔案刪除
        Files.delete(output.toPath());
        System.out.println("刪除");
    }
}
```

二十六、 JDBC

JDBC 是 Java 與 Database 溝通的方法，其操作屬於 java.sql.*的內容，
在使用時都需要處理 SQLException。

1. 掛載 Database Driver：

依照使用的 Database 不同，需要掛載的 Driver 也不同，
Driver 請跟廠商要，或去官網抓。

在 Java Web 和 Java 3.0 以前，需要用語法手動掛載 Driver 的 Class。
其 Class 的路徑依照使用的 Database 而有所不同：

語法：

```
Class.forName("Driver Class 的路徑");
```

範例：(MySQL)

```
Class.forName("com.mysql.jdbc.Driver");
```

2. 連線 Database：

(1) 語法：

```
String URL = "連線路徑/DataBase";
String user = "連線帳號";
String password = "連線密碼";
Connection 連線變數 = DriverManager.getConnection(URL, user, password);
```

(2) 範例：(MySQL)

```
String URL = "jdbc:mysql://localhost:3306/test";
String User = "root";
String Password = "root";
Connection conn = DriverManager.getConnection(URL, User, Password);
```

3. 執行 SQL：（查詢以外）

(1) 宣告傳送變數：

```
PreparedStatement 傳送變數;
```

(2) 輸入 SQL 語法：

```
傳送變數 = 連線變數.perpareStatement("SQL 語法");
```

＊使用 PreparedStatement 的好處是支援預編譯。
也就是 SQL 語法中，相關數值可以先用「?」代表，之後在設定「?」代表什麼值。

(3) 設定傳送值：

若使用預編譯，則需要使用以下語法，輸入「?」代表的值。

語法：

```
傳送變數.set 資料型態(index, value);
```

＊例如：傳送 String 型態，使用 傳送變數.setString(index, value);
傳送 int 型態，使用 傳送變數.setInt(index, value);
傳送 double 型態，使用 傳送變數.setDouble(index, value);

＊index 依照「?」由左往右數（1～）。

(4) 執行：

「?」代表的值輸入完成後，就需要將 SQL 指令送出。

語法：

```
傳送變數.execute();
```

(5) 範例：

```
//新增
String sql = "insert into polf values (?, ?)";
PreparedStatement ps;
ps = conn.prepareStatement(sql);
ps.setString(1, "bbb");
ps.setInt(2, 50);
ps.execute();
//修改
sql = "update polf set name = ? where name = ?";
ps = conn.prepareStatement(sql);
ps.setString(1, "aaa");
ps.setString(2, "bbb");
ps.execute();
//刪除
sql = "delete from polf where name = ?";
ps = conn.prepareStatement(sql);
ps.setString(1, "aaa");
ps.execute();
```

4. 查詢 SQL：

(1) 宣告查詢變數：

```
PreparedStatement 傳送變數 = 連線變數.perpareStatement("SQL 查詢語法");

ResultSet 查詢變數 = 傳送變數.executeQuery();
```

＊查詢到的全部資料，會用 ResultSet 的資料型態儲存起來。

(2) 指標常用方法：

獲得查詢資料後，由於一次只能輸出一筆資料，故要用以下方法，
控制指向第幾筆資料：

【若無資料，會回傳 False】

a. 查詢變數.first()：

指向「第一筆」資料

b. 查詢變數.last()：

指向「最後一筆」資料

c. 查詢變數.next()：

指向「下一筆」資料

d. 查詢變數.previous()：

指向「上一筆」資料

e. 查詢變數.absoulte(index)：

指向「第 index 筆」資料

(3) 取得資料：

當指向想要取得的該筆資料後，要將欄位的值取出並儲存。

語法：

```
資料型態 變數 = 查詢變數.get 資料型態("欄位名稱");
```

＊例如：取出 String 型態，使用 查詢變數.getString("欄位名稱");
取出 int 型態，使用 查詢變數.getInt("欄位名稱");
取出 double 型態，使用 查詢變數.getDouble("欄位名稱");

(4) 如何遍歷資料：

```
while (查詢變數.next()) {
    取出每一筆的欄位值;
}
```

＊當指標在第一筆時，使用 while 迴圈，搭配 next()即可遍歷查詢資料。

(5) 範例：

```
//取得查詢
String sql = "select * from polf";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
//輸出查詢資料
String name;
int age;
///遍歷
while (rs.next()) {
    name = rs.getString("name");
    age = rs.getInt("age");
    System.out.println(name + ", " + age);
}
System.out.println("-------------");
///第一筆
rs.first();
name = rs.getString("name");
age = rs.getInt("age");
System.out.println(name + ", " + age);
System.out.println("-------------");
///最後一筆
rs.last();
name = rs.getString("name");
age = rs.getInt("age");
System.out.println(name + ", " + age);
System.out.println("-------------");
///第 n 筆
int n = 3;
rs.absolute(n);
name = rs.getString("name");
age = rs.getInt("age");
System.out.println(name + ", " + age);
```

二十七、 執行緒(Thread)

本章節是介紹控制執行狀態的指令，這些指令都屬於 java.util.Thread。

1. 主執行緒與副執行緒：

(1) 在主程式中執行的就是「主執行緒」。
(2) 在主程式中，執行其他 Class 中的方法，就是「副執行緒」。

2. 程式暫停：

(1) 語法：

```
Thread.sleep(毫秒);
```

＊1 秒 = 1000 毫秒

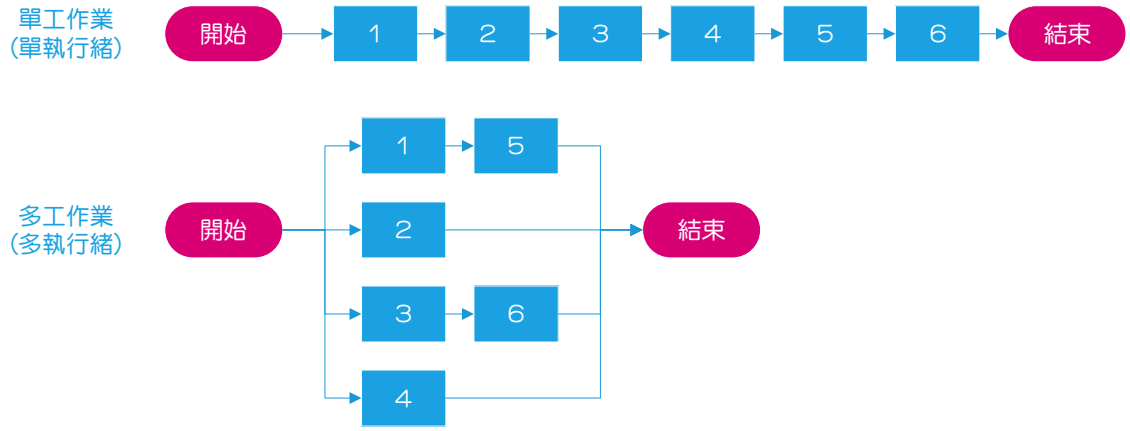
(2) 範例：

```
public class t1 {
    public static void main(String args[]) throws
                                              InterruptedException  {
        System.out.println("程式啟動");
        for (int i = 1 ; i <= 5 ; i++) {
            Thread.sleep(1000);
            System.out.println(i + "秒鐘");
        }
        System.out.println("程式結束");
    }
}
```

3. 多工作業：

一般來說，Java 的執行緒都是單工作業，也就是當很多 user 要使用某個方法時，需要排隊輪流使用，等上一個執行完了，才能讓下一個執行。

而多工作業，就是將某個副執行緒，開闢多個執行窗口，讓 user 同時使用，先跑完的就先執行。



(1) 語法：

【副 Class】

```
public class 副程式 implements Runnable  ← 要實作 Runnable
{
    //宣告變數
    R
    //建構式

    //多工作業
    @Override
    public void run() {        ← Override Runnable 的 run()
        要執行多工的內容;          並在裡面寫多工的內容
    }
}
```

【主程式】

```
public class 主程式 {
    public static void main(String args[])  {
        ……
        副程式 物件變數 = new 副程式(參數, ……);
        new Thread(物件變數).start();      ← 先串接 Thread，
    }                                          在透過 Thread 的 start()
}                                              啟動多工處理
```

(2) 範例：

【副 Class】

```
class thread implements Runnable {
    @Override
    public void run() {
        for (int i = 1 ; i < 500 ; i++) {
            System.out.println(i);
        }
    }
}
```

【主程式】

```
public class t1  {
    public static void main(String args[]) {
        thread s = new thread();
        for(int i = 1 ; i < 10 ; i++) {
            new Thread(s).start();
        }
    }
}
```

4. 取消同步化：

當進行多工作業時，其中的某個方法不能多工作業，就要將此方法取消同步化。

(1) 語法：

```
【副 Class】
public class 副執行緒 implements Runnable
{
    //宣告變數

    //建構式

    //多工作業
    @Override
    public void run() {
        要執行多工的內容;
        取消同步化方法(參數, ……);  ← 執行此方法要逐一乖乖排隊
    }

    //方法
    synchronized 前贅詞 類型 取消同步化方法(資料型態 參數, ……) {
        內容;
    }
}
```

(2) 範例：

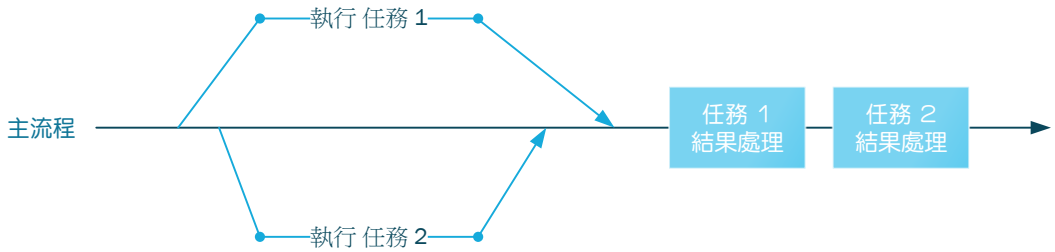
【副 Class】

```
class thread implements Runnable {
    @Override
    public void run() {
        for (int i = 1 ; i < 100 ; i++) {
            System.out.println(i);
            syn();
        }
    }

    synchronized static void syn() {
        System.out.println("A");
        System.out.println("B");
        System.out.println("C");
        System.out.println("D");
    }
}
```

【主程式】

```
public class t1  {
    public static void main(String args[]) {
        thread s = new thread();
        for(int i = 1 ; i < 10 ; i++) {
            new Thread(s).start();
        }
    }
}
```

二十八、 Future：異步任務 - 異步阻塞 【Java 5 舊寫法】



1. 語法架構：

(1) 執行緒數量不可過多，會導致系統負擔過大

(2) 有回傳值的任務，要用 Future 接收回傳，並透過 get()來取出回傳值 (阻塞主線程)

啟動多工

```
public static void main(String[] args) {
   (示範 2 種方法)
   1. 固定執行緒數量
   ExecutorService 任務變數 = Executors.newFixedThreadPool(執行緒數量);
   2. 不固定執行緒數量，動態調整
   缺點: 可能造成執行池過大，增加系統負擔
   ExecutorService 任務變數 = Executors.newCachedThreadPool();
```

提交任務

```
   (示範 3 種方法)
   1. execute: 無回傳值 的 單次任務
   任務變數.execute(無回傳值函式(參數, ...));

   2. submit: 有回傳值的 單次任務
   Future<Map<String, Object>> future = 任務變數.submit(有回傳值函式(參數, ...));
   // 先透過 get() 方法取得 單次任務的回傳變數
   // 再透過 Map 的 get(參數名) 取得回傳值
   Map<String, Object> 回傳變數 = future.get();

   3. invokeAll: 有回傳值的 大量任務
            先將要執行的任務準備好後，再一次全部執行
   List<Callable<Map<String, Object>>> taskList = new ArrayList<>();
   taskList.add(有回傳值函式(參數, ...));
   List<Future<Map<String, Object>>> futureList = 任務變數.invokeAll(taskList);
   for(Future<Map<String, Object>> future : futureList) {
      // 先透過 get() 方法取得 單次任務的回傳變數
      Map<String, Object> 回傳變數 = future.get();
      // 再透過 Map 的 get(參數名) 取得回傳值
   }
```

關閉多工

```
   任務變數.shutdown();
}
```

無回傳值任務函式

```
private static Runnable 無回傳值函式(參數型態 參數名稱, ...) {
   Runnable output = () -> {
      // 要執行的內容
   };
   return output;
}
```

有回傳值任務函式

```
private static Callable<Map<String, Object>> 有回傳值函式(參數型態 參數名稱, ...) {
   Callable<Map<String, Object>> output = () -> {
      Map<String, Object> map = new HashMap<>();
      // 要執行的內容
      // 透過 Map 設定回傳值
      return map;
   };
   return output;
}
```

2. 範例 1 execute：

```java
import java.util.concurrent.ExecutorService;

public class ExecutorTest1 {

    public static void main(String[] args) throws Exception {
        // 設定多工任務池: 固定2執行緒
        ExecutorService service = Executors.newFixedThreadPool(2);
        // 設定任務(單一無回傳值): 設定9個任務
        for(int i=1 ; i<=15 ; i++) {
            service.execute(showMsg("任務" + i));
        }
        // 關閉多工
        service.shutdown();
    }

    private static Runnable showMsg(String msg) throws Exception {
        Runnable output = () -> {
            System.out.println(msg);
        };
        return output;
    }
}
```

```
<terminated> Execu
任務2
任務1
任務3
任務4
任務6
任務5
任務7
任務8
任務9
任務11
任務10
任務12
任務13
任務14
任務15
```

3. 範例 2 submit：

```java
import java.util.ArrayList;

public class ExecutorTest2 {
    public static void main(String[] args) throws Exception {
        // 設定多工任務池: 固定2執行緒
        ExecutorService service = Executors.newFixedThreadPool(2);
        // 設定任務(單一有回傳值): 設定9個任務
        List<Future<Map<String, Object>>> futureList = new ArrayList<>();
        for(int i=1 ; i<=9 ; i++) {
            String user = "學員" + i;
            int age = i * 4;
            Future<Map<String, Object>> future = service.submit(chkAge(user,age));
            futureList.add(future);
        }
        // 顯示任務結果
        for(Future<Map<String, Object>> future : futureList) {
            // 透過 get() 方法取得回傳值
            Map<String, Object> valueMap = future.get();
            System.out.println(valueMap.get("user") + " " + valueMap.get("age") + "歲" +
                               "，是否成年:" + future.get().get("value"));
        }
        // 關閉多工
        service.shutdown();
    }

    private static Callable<Map<String, Object>> chkAge(String user, int age) throws Exception {
        Callable<Map<String, Object>> output = () -> {
            Map<String, Object> map = new HashMap<>();
            map.put("user",user);
            map.put("age", age);
            if(age >= 18) {
                map.put("value", Boolean.TRUE);
            } else {
                map.put("value", Boolean.FALSE);
            }
            System.out.println(user + " 啟動年齡判斷");
            return map;
        };
        return output;
    }
}
```

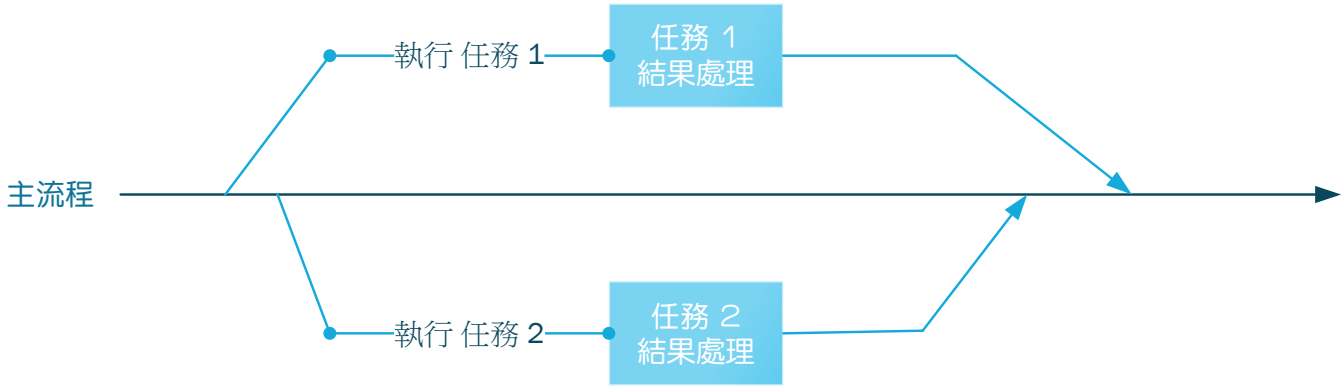
```
<terminated> ExecutorTest2 [Java Application] C:\Pro
學員1 啟動年齡判斷
學員2 啟動年齡判斷
學員3 啟動年齡判斷
學員1 4歲，是否成年: false
學員5 啟動年齡判斷
學員4 啟動年齡判斷
學員6 啟動年齡判斷
學員2 8歲，是否成年: false
學員8 啟動年齡判斷
學員7 啟動年齡判斷
學員9 啟動年齡判斷
學員3 12歲，是否成年: false
學員4 16歲，是否成年: false
學員5 20歲，是否成年: true
學員6 24歲，是否成年: true
學員7 28歲，是否成年: true
學員8 32歲，是否成年: true
學員9 36歲，是否成年: true
```

4. 範例 3 invokeAll：

```java
import java.util.ArrayList;

public class ExecutorTest3 {
    public static void main(String[] args) throws Exception {
        // 設定多工任務池: 固定2執行緒
        ExecutorService service = Executors.newFixedThreadPool(5);
        // 設定任務(單一有回傳值): 設定9個任務
        List<Callable<Map<String, Object>>> taskList = new ArrayList<>();
        for(int i=1 ; i<=9 ; i++) {
            String user = "學員" + i;
            int age = i * 4;
            taskList.add(chkAge(user,age));
        }
        List<Future<Map<String, Object>>> futureList = service.invokeAll(taskList);
        // 顯示任務結果
        for(Future<Map<String, Object>> future : futureList) {
            // 透過 get() 方法取得回傳值
            Map<String, Object> valueMap = future.get();
            System.out.println(valueMap.get("user") + " " + valueMap.get("age") + "歲" +
                               "，是否成年:" + future.get().get("value"));
        }
        // 關閉多工
        service.shutdown();
    }

    private static Callable<Map<String, Object>> chkAge(String user, int age) throws Exception {
        Callable<Map<String, Object>> output = () -> {
            Map<String, Object> map = new HashMap<>();
            map.put("user",user);
            map.put("age", age);
            if(age >= 18) {
                map.put("value", Boolean.TRUE);
            } else {
                map.put("value", Boolean.FALSE);
            }
            System.out.println(user + " 啟動年齡判斷");
            return map;
        };
        return output;
    }
}
```

```
<terminated> ExecutorTest3 [Java Application] C
學員1 啟動年齡判斷
學員5 啟動年齡判斷
學員4 啟動年齡判斷
學員2 啟動年齡判斷
學員9 啟動年齡判斷
學員3 啟動年齡判斷
學員8 啟動年齡判斷
學員7 啟動年齡判斷
學員6 啟動年齡判斷
學員1 4歲，是否成年: false
學員2 8歲，是否成年: false
學員3 12歲，是否成年: false
學員4 16歲，是否成年: false
學員5 20歲，是否成年: true
學員6 24歲，是否成年: true
學員7 28歲，是否成年: true
學員8 32歲，是否成年: true
學員9 36歲，是否成年: true
```

二十九、　CompletableFuture：異步任務 - 異步不阻塞 【Java 8 新寫法】



＊新增/修改/刪除 作業時，不可使用 異步處理，

會導致 失敗時 無法全部進行 Rollbeck。（多執行緒非同次交易）

1. 宣告 任務集合
   用於紀錄 後續需要等待完成的任務

```
List<CompletableFuture<?>> futureList = new ArrayList<>();
```

2. 設定任務，任務設定完畢後就會開始執行（不會阻塞主線程）
   1. 無回傳值任務
      (1) *runAsync*：設定無回傳值的任務
      (2) 執行後，要將任務 加入 任務集合中

```
CompletableFuture<Void> future =
    CompletableFuture.runAsync(() -> 無回傳任務(參數));
futureList.add(future);
```

   2. 有回傳值任務 + 將回傳結果 進行處理（無回傳值）
      (1) *supplyAsync*：設定有回傳值的任務
      (2) thenAccept：執行完畢後，對結果進行處理
      (3) 因為無回傳值，所以 回傳型態 是 Void
      (4) 執行後，要將任務 加入 任務集合中

```
CompletableFuture<Void> future1 =
    CompletableFuture.supplyAsync(() -> 有回傳任務(參數))
        .thenAccept(result -> {
            你要進行的邏輯處理
        });
futureList.add(future1);
```

3. 有回傳值任務 + 將回傳結果 加工為 新結果 (有回傳值)
   (1) *supplyAsync*：設定有回傳值的任務
   (2) thenApply：執行完畢後，對結果進行處理 (要 return 回傳值)
   (3) 執行後，要將任務 加入 任務集合中

```
CompletableFuture<String> future2 =
    CompletableFuture.supplyAsync(() -> 有回傳任務(參數))
        .thenApply(result -> {
            你要進行的邏輯處理
            return result;
        });
futureList.add(future2);
```

4. 有回傳值任務 + 不做後續處理
   (1) *supplyAsync*：設定有回傳值的任務
   (2) 執行後，要將任務 加入 任務集合中
   (3) 結果的處理，可以於 allOf 再進行處理

```
CompletableFuture<String> future3 =
    CompletableFuture.supplyAsync(() -> 有回傳任務(參數));
futureList.add(future3);
```

3. 等待任務完成
   (1) allOf 裡面是放「要等待完成的任務」
   (2) thenRun：額外處理，這裡進行 任務三 的 結果處理
   (3) 取得 任務回傳值：
   任務變數.join()
   (4) 如果沒有設定，任務設定完畢後，主線程就會直接結束 (異步任務還沒執行完畢)

```
CompletableFuture.allOf(futureList.toArray(CompletableFuture[]::new))
    .thenRun(() -> {
          resultList.add(future3.join());
    })
.join();
```

4. 範例

```
任務設定完畢
啟動任務 T002
啟動任務 T001
啟動無參數任務 T004
啟動任務 T003
啟動任務 T002 - 執行完畢
啟動任務 T003 - 執行完畢
啟動無參數任務 T004 - 執行完畢
啟動任務 T001 - 執行完畢
任務完成，結果清單：[使用者ID：T002, 使用者ID：T001, 使用者ID：T003]
```

```java
/** 異步不阻塞 任務範例 **/
public void RunCompletableFutrue() {  1 个用法
    // 設定 結果回存變數
    List<String> resultList = new CopyOnWriteArrayList<>();

    // 設定 任務集合
    List<CompletableFuture<?>> futureList = new ArrayList<>();

    // 設定任務，每個任務設定後就會執行
    // 任務 1: 有回傳值 + thenAccept (處理任務結果 + 無回傳值)
    CompletableFuture<Void> future1 = CompletableFuture.supplyAsync(() -> printfUserId("T001"))
            .thenAccept( String result -> {
                resultList.add(result);

            });
    futureList.add(future1);
    // 任務 2: 有回傳值 + thenApply (處理任務結果 + 有回傳值)
    CompletableFuture<String> future2 = CompletableFuture.supplyAsync(() -> printfUserId("T002"))
            .thenApply( String result -> {
                resultList.add(result);
                return result;
            });
    futureList.add(future2);
    // 任務 3: 有回傳值 (全部完成後 再處理結果)
    CompletableFuture<String> future3 = CompletableFuture.supplyAsync(() -> printfUserId("T003"));
    futureList.add(future3);
    // 任務 4: 無回傳值
    CompletableFuture<Void> future4 = CompletableFuture.runAsync(() -> printfVoid( jobNo: "T004"));
    futureList.add(future4);

    // 試著插入其他邏輯
    System.err.println("任務設定完畢");

    // 等待所有任務完成
    CompletableFuture.allOf(futureList.toArray(CompletableFuture[]::new))
            .thenRun(() -> {
                resultList.add(future3.join());
            })
            .join();

    // 輸出結果
    System.err.println("任務完成，結果清單：" + resultList);
}
/** 有回傳值的任務 **/
public String printfUserId(String userId) {  3 个用法
    System.out.println("啟動任務 " +  userId);
    try {
        Thread.sleep( millis: 1000); // 模擬耗時
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
    System.out.println("啟動任務 " +  userId + " - 執行完畢");
    return "使用者ID：" + userId;
}
/** 無回傳值的任務 **/
public void printfVoid(String jobNo) {  1 个用法
    System.out.println("啟動無參數任務 " + jobNo);
    try {
        Thread.sleep( millis: 1000); // 模擬耗時
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
    System.out.println("啟動無參數任務 " + jobNo + " - 執行完畢");
}
```

5. 實際應用：

透過 非同步 同時抓取資料

```
CompletableFuture<PolfEntity> polfFuture = CompletableFuture.supplyAsync(() -> loadDataService.getPolf());
CompletableFuture<RspoEntity> rspoFuture = CompletableFuture.supplyAsync(() -> loadDataService.getRslf());
CompletableFuture<ChswEntity> chswFuture = CompletableFuture.supplyAsync(() -> loadDataService.getChsw());

PolfEntity polf = polfFuture.join();
RspoEntity rspo = rspoFuture.join();
ChswEntity chsw = chswFuture.join();
```

```
CompletableFuture<List<ColfEntity>> colfFuture = CompletableFuture.supplyAsync(() -> loadDataService.getColf());
CompletableFuture<List<RscoEntity>> rscoFuture = CompletableFuture.supplyAsync(() -> loadDataService.getRsco());
CompletableFuture<List<PldfEntity>> pldfFuture = CompletableFuture.supplyAsync(() -> loadDataService.getPldf());

List<ColfEntity> colfEntityList = colfFuture.join();
List<RscoEntity> rscoEntityList = rscoFuture.join();
List<PldfEntity> pldfEntityList = pldfFuture.join();
```

三十、 時間 API

1. 獲取目前電腦日期：

```
Date 目前時間變數 = new Date();
```

＊new Date()可以取得目前的電腦時間，其輸出的資料型態為 Date。

＊此方法的時間為完整格式，例如：
Mon Aug 26 14:57:08 CST 2019

2. 轉換輸出格式：

```
String 儲存變數 = new SimpleDateFormat("時間格式").format(目前時間變數);
```

3. 常見的時間格式：

(1) yyyy/MM/dd：

西元年/月/日

範例：
2019/8/26

(2) ahh:mm:ss：

12 小時制的時間

範例：
下午 03:04:15

(3) HH:mm:ss：

24 小時制的時間

範例：
15:04:15

4. 範例：

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class t1  {
    public static void main(String args[]) {
        //取得目前時間
        Date today = new Date();
        System.out.println(today);
        //輸出西元年
        String d1 = new SimpleDateFormat("yyyy/MM/dd").format(today);
        System.out.println(d1);
        //12 小時制的時間
        String d2 = new SimpleDateFormat("ahh:mm:ss").format(today);
        System.out.println(d2);
        //24 小時制的時間
        String d3 = new SimpleDateFormat("HH:mm:ss").format(today);
        System.out.println(d3);
    }
}
```

【輸出】

```
Mon Aug 26 15:04:15 CST 2019
2019/8/26
下午 03:04:15
15:04:15
```

三十一、 字串 API

1. length (查詢字串長度)：

```
int 儲存變數 = 字串變數.length();
```

範例

```
String s = "1234 5678";
int length = s.length();          // 輸出 9
```

2. substring (輸出部分字串)：

```
String 儲存變數 = 字串變數.substring(開頭 index, 結尾 index);
```

＊index 從 0 起算。 (0～)
＊包含開頭 index，但是不包含結尾 index。 (包前不包後)

範例

```
String s = "0123456789";
System.out.println(s.substring(2, 5));  // 輸出 234
```

3. replace (取代)

將 指定的文字 替換成 特定文字。

```
String 儲存變數 = 字串變數.replace(目標文字, 取代文字);
```

範例

```
String s = "abcd abcd";
System.out.println(s.replace("c","C"));   // 輸出 abCd abCd
```

4. compareTo (字串比大小)：

回傳值：
1 = 前字串 > 後字串
0 = 前後字串內容相同
-1 = 前字串 < 後字串

```
Boolean 回傳值 =  前字串.compareTo(後字串);
```

範例

```
String date1 = "113/05/21";
String date2 = "114/01/01";
System.out.println(date1.compareTo(date2));   // 輸出 -1
```

5. matches (字串模糊比對)：

進行字串的模糊比對。
回傳值：True = 命中 / False = 無命中

```
Boolean 回傳值 =  String 變數.matches("正規表達式");
```

＊JAVA 的正規表達式中，萬用字元為『.*』
如：找『6 開頭保單』使用『6.*』

＊因為 matches 只能使用正規表達式，
所以 如果想要與 Informix 的 matches 有相同功能，
可以使用
```
Boolean 回傳值 =  String 變數.matches(比較變數.replace("*",".*"));
```
將『*』轉換為『.*』

範例

```
String matchesPolicyNo = "6.*";
String chkPolicyNo = "600000000001";
Boolean result1 = chkPolicyNo.matches(matchesPolicyNo);
System.out.println(result1);      // 輸出 true

String matchesPlanClassCode = "0***";
String chkPlanClassCode = "0121";
Boolean result2 =
chkPlanClassCode.matches(matchesPlanClassCode.replace("*", ".*"));
System.out.println(result2);      // 輸出 true
```

6. StringBuilder (高性能的字串處理器)

- String 和 StringBuilder 的差異比較

	String	StringBuilder
可變性	不可變，每次修改都是建立新物件	可變，修改不創建新物件
字串拼接	使用 +	使用 append(…)
性能	性能較差	性能較高
轉變為 String	直接是 String	需要使用 toString()轉型

- StringBuilder 的常用方法

  - append(String str)

    將 指定的字符串 追加到 末尾

    範例

    ```
    StringBuilder sb = new StringBuilder("Hello");
    sb.append(" World");         // sb 現在是 "Hello World"
    ```

  - insert(int offset, String str)

    在 指定位置 插入 字符串

    範例

    ```
    StringBuilder sb = new StringBuilder("Hello World");
    sb.insert(5, " Java");       // sb 現在是 "Hello Java World"
    ```

  - delete(int start, int end)

    刪除從 start 索引到 end 索引的字符（不包含 end）

    範例

    ```
    StringBuilder sb = new StringBuilder("Hello Java World");
    sb.delete(6, 11);             // sb 現在是 " Hello World"
    ```

  - replace(int start, int end, String str)

    用 指定的字符串 替換從 start 到 end 索引的字符

    範例

    ```
    StringBuilder sb = new StringBuilder("Hello World");
    sb.replace(6, 11, "Java");   // sb 現在是 "Hello Java"
    ```

  - toString()

    將 StringBuilder 轉換為 String

    範例

    ```
    StringBuilder sb = new StringBuilder("Hello");
    String str = sb.toString();
    ```

  - length()

    返回 StringBuilder 目前的字符長度

    範例

    ```
    StringBuilder sb = new StringBuilder("Hello");
    System.out.println(sb.length());    // 輸出 5
    ```

三十二、　數字表示法

1. 語法：

```
DecimalFormat 格式變數 = new DecimalFormat("數字格式");
```

2. 數字格式：

以下三種符號有其特殊意義，其他的就會直接出現。

(1)「0」:

該格一定要有值，沒值會補 0。

(2)「#」:

該格不一定有值，沒值就不會顯示。

(3)「%」:

此數字為百分數，會數字自動「乘以 100」，並將「%」顯示在最後。

3. 範例：

```
import java.text.DecimalFormat;

public class t1  {
    public static void main(String args[]) {
        double d1 = 987.654321;
        double d2 = 6543;
        double d3 = 0.0328;

        DecimalFormat df;
        //取到小數點後 1 位 (會四捨五入)
        df = new DecimalFormat("####0.0");
        System.out.println(df.format(d1));
        //取到小數點後 2 位
        df = new DecimalFormat("####0.00");
        System.out.println(df.format(d1));
        //取到小數點後 3 位 (會自動補 0)
        df = new DecimalFormat("####0.000");
        System.out.println(df.format(d2));
        //取得小數點後 1 位的百分比 (有%會自動乘 100)
        df = new DecimalFormat("##0.00%");
        System.out.println(df.format(d3));
    }
}
```

【輸出】

```
987.7
987.65
6543.000
3.28%
```

三十三、　精確的數字運算

在 Java 中，浮點數的設計上是為了快速提供近似值，故會有運算結果不夠精確的問題，不適合在商業上使用。

因此在需要精確數字運算的場合，可以透過 java.math.BigDecimal 來進行運算。

1. 宣告：

```
BigDecimal 儲存變數 = new BigDecimal(String value);
```

＊為求數值夠精確，value 的型態建議為 String。

2. 常用方法：

(1) 運算：

名稱	語法
加法　A + B	儲存變數 A.add(儲存變數 B)
減法　A - B	儲存變數 A.subtract(儲存變數 B)
乘法　A * B	儲存變數 A.multiply(儲存變數 B)
除法　A / B	儲存變數 A.divide(儲存變數 B)

(2) 兩數比較：

```
int 回傳值 = 儲存變數 A.compareTO(儲存變數 B);
```

名稱	回傳值
大於　A > B	1
等於　A == B	0
小於　A < B	-1

(3) 與 0 比較：

```
int 回傳值 = 儲存變數 A.signum();
```

名稱	回傳值
正數　A > 0	1
為 0　A == 0	0
負數　A < 0	-1

(4) 轉型成數字：

```
(a)  int    回傳值 = 儲存變數.intValue();
(b)  long   回傳值 = 儲存變數.longValue();
(c)  float  回傳值 = 儲存變數.floatValue();
(d)  double 回傳值 = 儲存變數.doubleValue();
```

範例：

```
public class t1  {
    public static void main(String args[]) {
        BigDecimal num1 = new BigDecimal("2.4");
        BigDecimal num2 = new BigDecimal("1.2");
        BigDecimal num3 = new BigDecimal("-1.2");
        BigDecimal num4;
        int result;
        //加法
        num4 = num1.add(num2);
        System.out.println("加法: " + num4);
        //減法
        num4 = num1.subtract(num2);
        System.out.println("減法: " + num4);
        //乘法
        num4 = num1.multiply(num2);
        System.out.println("乘法: " + num4);
        //除法
        num4 = num1.divide(num2);
        System.out.println("除法: " + num4);
        //兩數比較：大於，返回 1
        result = num1.compareTo(num2);
        System.out.println("大於回傳: " + result);
        //兩數比較：等於，返回 0
        result = num2.compareTo(num2);
        System.out.println("等於回傳: " + result);
        //兩數比較：小於，返回 -1
        result = num2.compareTo(num1);
        System.out.println("小於回傳: " + result);
        //與 0 比較：正數，返回 1
        result = num1.signum();
        System.out.println("正數回傳: " + result);
        //與 0 比較：為 0，返回 0
        result = num2.signum();
        System.out.println("為 0 回傳: " + result);
        //與 0 比較：負數，返回 -1
        result = num3.signum();
        System.out.println("負數回傳: " + result);
    }
```

(5) 四捨五入、無條件進位、無條件捨去

```
BigDecimal 儲存變數 = new BigDecimal(String value);
儲存變數 = 儲存變數.setScale(小數位數, 捨入型態);
捨入型態: BigDecimal.ROUND_HALF_UP    【四捨五入】
          BigDecimal.ROUND_UP         【無條件進位】
          BigDecimal.ROUND_DOWN       【無條件捨去】
```

範例：

```
3  public class t1 {
4      public static void main(String[] args) {
5          BigDecimal decimal = new BigDecimal("10.2356");
6          BigDecimal t1 = decimal.setScale(2, BigDecimal.ROUND_HALF_UP);
7          BigDecimal t2 = decimal.setScale(1, BigDecimal.ROUND_UP);
8          BigDecimal t3 = decimal.setScale(1, BigDecimal.ROUND_DOWN);
9          System.out.println("原始:" + decimal.doubleValue());
10         System.out.println("四捨五入  (2位):" + t1.doubleValue());
11         System.out.println("無條件進位(1位):" + t2.doubleValue());
12         System.out.println("無條件捨去(1位):" + t3.doubleValue());
13     }
14 }
```

```
Console × Problems Progress Debug Shell
<terminated> t1 [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (2023年6月9日 上午6:56:22)
原始:10.2356
四捨五入  (2位):10.24
無條件進位(1位):10.3
無條件捨去(1位):10.2
```

三十四、　Field：取得 與 修改 物件屬性(值)

在某些情況下，我們需要針對 某個物件 的 所有欄位值 進行調整

如：當 欄位值 = Null 時，需要 將其改成 空格
　　此時，可以透過 Field 來達成這個目的
　　先 取得 物件的所有欄位
　　再 逐一檢查 該欄位的欄位值 是否為 Null
　　　如果為 Null 就改成 空格

1. 語法示範：

```
// 取得欄位，並存成 Field 陣列
Field[] fieldList = 物件變數.getClass().getDeclaredFields();

// 遍歷 Field 陣列: 取得 與 修改 欄位值
for(Field field : fieldList) {
    // 要 取得/修改 欄位值，必須先將權限設定為可訪問
    field.setAccessible(true);
    // (A) get 方法: 取得欄位值
    Object obj 變數 = field.get(物件變數);
    // (B) set 方法: 修改欄位值
    field.set(物件變數, 修改值);
}
```

2. 範例：

```
1  import java.lang.reflect.Field;
2
3  public class ObjectTest {
4
5      public static void main(String[] args) throws Exception {
6          User user = new User(null,35);
7          System.out.println("調整前: ");
8
9          // 獲取 DTO 中 所有欄位的類型 並儲存起來
10         Field[] fieldList = user.getClass().getDeclaredFields();
11         // foreach 檢查所有欄位: 依序輸出所有欄位值
12         for(Field field : fieldList) {
13             // 如果要用 get() 取得欄位值，需要先設定權限為「可訪問」
14             field.setAccessible(true);
15             // 測試: 列印出欄位值
16             Object obj = field.get(user);
17             System.out.println(field + ": " + obj);
18             // 修改欄位值: 使用 set() 給值
19             if(field.get(user) == null) {
20                 field.set(user, " ");
21             }
22
23         }
24         System.out.println("調整後: ");
25         for(Field field : fieldList) {
26             field.setAccessible(true);
27             System.out.println(field + ": " + field.get(user));
28         }
29     }
30 }
```

```
<terminated> ObjectTest [Java Application] C:\Program Files\Java\jre-1.8
調整前:
private java.lang.String User.name: null
private int User.age: 35
調整後:
private java.lang.String User.name:
private int User.age: 35
```

三十五、　附錄：透過反射執行 Method

透過 反射執行 Method 的技巧，可以用來 並行處理多個相同 INPUT、OUTPUT 的方法。

下面範例是模擬執行核保訊息

- DataModelDto：INPUT 變數，屬性為 變更前後資料 及 試算結果 等東西。
- CheckMethodService：保存 核保訊息 Method 的 Service。
- CheckMethodDto：核保訊息方法的回傳值，有兩個屬性。
  (1) resultShow：是否要顯示。
  (2) checkResult：要顯示時，紀錄 核保訊息 的代碼、文字、等級 的 變數。
- List<CheckResultDto>：回傳時，用 List 拋出 核保訊息的集合。

```
public class CheckMainService {  3 个用法
    /**
     * 檢查所有核保訊息，並返回需要顯示的檢查結果列表
     * @param dataModelDto 包含核保數據的傳輸物件
     * @return 需要顯示的檢查結果列表（List<CheckResultDto>）
     */
    public List<CheckResultDto> checkAllMessages(DataModelDto dataModelDto) {  1 个用法
        // 初始化結果列表
        List<CheckResultDto> results = new ArrayList<>();

        // 使用反射動態調用所有核保訊息檢核方法
        Method[] methods = CheckMethodService.class.getMethods(); // 獲取CheckMethodService中所有public方法
        Arrays.stream(methods)
                .filter( Method m -> m.getName().startsWith("check")) // 篩選出以"check"開頭的方法
                .parallel() // 並行處理提高效率
                .forEach( Method m -> { // 對每個符合條件的方法進行處理
                    try {
                        // 動態調用方法：
                        // 1. 創建CheckMethodService實例
                        // 2. 傳入dataModelDto參數
                        // 3. 將返回結果轉型為CheckMethodDto
                        CheckMethodDto result = (CheckMethodDto) m.invoke(new CheckMethodService(), dataModelDto);

                        // 如果結果標記為需要顯示，則加入結果列表
                        if (result.getResultShow()) {
                            results.add(result.getCheckResult());
                        }
                    } catch (Exception e) {
                        throw new RuntimeException(m.getName() + " 執行錯誤: " + e.getMessage());
                    }
                });

        return results;
    }
}
```

三十六、　附錄：資料為空判斷 【spring】

＊此方法為 spring 的相關方法，在使用 spring boot 進行開發時可以使用

- StringUtils.isEmpty(字串) 【spring 已棄用】

  >> import org.springframework.util.StringUtils;

  當資料的型態為 String，可以透過 此方法來檢查 資料是否為空
  >> true = 資料為空 / false = 資料不為空

  資料為空 定義：
  >> isEmpty 使用時，空格資料 非空白

  (1)　""
  (2)　null

  範例：

```
String st1 = null;
if (StringUtils.isEmpty(st1)) {
    System.out.println("空白");    // 進入這邊
} else {
    System.out.println("非空白");
}

String st2 = "";
if (StringUtils.isEmpty(st2)) {
    System.out.println("空白");    // 進入這邊
} else {
    System.out.println("非空白");
}

String st3 = "123";
if (StringUtils.isEmpty(st3)) {
    System.out.println("空白");
} else {
    System.out.println("非空白");  // 進入這邊
}
```

- StringUtils.isBlank(字串)

>> import org.apache.commons.lang.StringUtils;　　【spring boog 2】

>> import org.apache.commons.lang3.StringUtils;　　【spring boot 3】

當資料的型態為 String，可以透過 此方法來檢查 資料是否為空
>> true = 資料為空 / false = 資料不為空

資料為空 定義：
>> isBlank 使用時，空格資料 也算是 空白

(1)　""
(2)　" "
(3)　null

範例：

```
String st1 = null;
if (StringUtils.isBlank(st1)) {
    System.out.println("空白");   // 進入這邊
} else {
    System.out.println("非空白");
}

String st2 = "";
if (StringUtils.isBlank(st2)) {
    System.out.println("空白");   // 進入這邊
} else {
    System.out.println("非空白");
}

String st3 = " ";
if (StringUtils.isBlank(st3)) {
    System.out.println("空白");   // 進入這邊
} else {
    System.out.println("非空白");
}

String st4 = "abc";
if (StringUtils.isBlank(st4)) {
    System.out.println("空白");
} else {
    System.out.println("非空白");  // 進入這邊
}
```

- CollectionUtils.isEmpty(集合)

  >> import org.springframework.util.CollectionUtils;

  當資料的型態為 集合 (如：List)，可以透過 此方法來檢查 資料是否為空
  >> true = 資料為空 / false = 資料不為空

  資料為空 定義：

  (1) [] // 空集合
  (2) null

  範例：

```
List<String> ls1 = null;
if (CollectionUtils.isEmpty(ls1)) {
    System.out.println("空白");   // 進入這邊
} else {
    System.out.println("非空白");
}

List<String> ls2 = new ArrayList<>();
if (CollectionUtils.isEmpty(ls2)) {
    System.out.println("空白");   // 進入這邊
} else {
    System.out.println("非空白");
}

List<String> ls3 = new ArrayList<>();
ls3.add("1");
ls3.add("2");
if (CollectionUtils.isEmpty(ls3)) {
    System.out.println("空白");
} else {
    System.out.println("非空白");  // 進入這邊
}
```