Java SE 個人筆記

黃懷慶

2025/09/01

目錄

[一、 Java的主程式架構 5](#_Toc207703129)

[二、 基本資料型態 5](#_Toc207703130)

[1. 基本的資料型態： 5](#_Toc207703131)

[2. 包覆類別： 5](#_Toc207703132)

[3. 自動辨識型別的特殊屬性： 6](#_Toc207703133)

[三、 註解 6](#_Toc207703134)

[四、 跳脫字元、強制轉換與資料輸出 7](#_Toc207703135)

[1. 跳脫字元： 7](#_Toc207703136)

[2. 強制轉換字元： 7](#_Toc207703137)

[3. 資料輸出： 7](#_Toc207703138)

[五、 變數宣告 8](#_Toc207703139)

[1. 基本： 8](#_Toc207703140)

[2. 一維陣列： 8](#_Toc207703141)

[3. 二維陣列： 9](#_Toc207703142)

[六、 運算子 10](#_Toc207703143)

[1. 算數運算子： 10](#_Toc207703144)

[2. 指定運算子： 10](#_Toc207703145)

[3. 關係運算子 1： 10](#_Toc207703146)

[4. 關係運算子 2： 11](#_Toc207703147)

[5. 位元運算子： 13](#_Toc207703148)

[6. 位移運算子： 13](#_Toc207703149)

[七、 邏輯判斷 14](#_Toc207703150)

[1. if判斷式： 14](#_Toc207703151)

[2. switch判斷式： 14](#_Toc207703152)

[3. 三元運算子： 14](#_Toc207703153)

[八、 迴圈 15](#_Toc207703154)

[1. for迴圈： 15](#_Toc207703155)

[2. for-each迴圈： 15](#_Toc207703156)

[3. while迴圈： 15](#_Toc207703157)

[4. do-while迴圈： 16](#_Toc207703158)

[5. 迴圈控制： 16](#_Toc207703159)

[九、 物件 17](#_Toc207703160)

[1. 物件初始化： 17](#_Toc207703161)

[2. 物件的撰寫： 18](#_Toc207703162)

[十、 方法 19](#_Toc207703163)

[1. 不傳值的方法： 19](#_Toc207703164)

[2. 傳值的方法： 19](#_Toc207703165)

[十一、 陣列的常用方法 20](#_Toc207703166)

[1. 由小到大排序： 20](#_Toc207703167)

[2. 搜尋： 20](#_Toc207703168)

[十二、 繼承 21](#_Toc207703169)

[1. 語法： 21](#_Toc207703170)

[2. 父類別、子類別與繼承限制： 22](#_Toc207703171)

[3. 建構式： 22](#_Toc207703172)

[4. 方法： 23](#_Toc207703173)

[十三、 多型 24](#_Toc207703174)

[1. 語法： 24](#_Toc207703175)

[2. 優點： 24](#_Toc207703176)

[3. 缺點： 24](#_Toc207703177)

[4. 編譯的順序： 24](#_Toc207703178)

[十四、 抽象類別與抽象方法 25](#_Toc207703179)

[1. 語法： 25](#_Toc207703180)

[2. 抽象類別： 25](#_Toc207703181)

[3. 抽象方法： 25](#_Toc207703182)

[十五、 interface (介面) 26](#_Toc207703183)

[1. 語法： 26](#_Toc207703184)

[2. 特點： 27](#_Toc207703185)

[十六、 內部類別 與 匿名類別 28](#_Toc207703186)

[1. 內部類別： 28](#_Toc207703187)

[2. 匿名類別： 29](#_Toc207703188)

[十七、 enum (列舉) 30](#_Toc207703189)

[十八、 錯誤處理 33](#_Toc207703190)

[1. 錯誤訊息的分類： 33](#_Toc207703191)

[2. throws (拋出錯誤)： 34](#_Toc207703192)

[3. try-catch (捕捉錯誤)： 35](#_Toc207703193)

[十九、 泛型 37](#_Toc207703194)

[1. 建立單一泛型： 37](#_Toc207703195)

[2. 建立多重泛型： 38](#_Toc207703196)

[3. 使用泛型： 39](#_Toc207703197)

[二十、 集合 40](#_Toc207703198)

[1. 常見的集合的種類： 40](#_Toc207703199)

[2. ArrayList： 41](#_Toc207703200)

[3. HashSet： 44](#_Toc207703201)

[4. PriorityQueue： 47](#_Toc207703202)

[二十一、 Map<K, V> 50](#_Toc207703203)

[1. Key區： 50](#_Toc207703204)

[2. Value區： 50](#_Toc207703205)

[3. HashMap： 50](#_Toc207703206)

[二十二、 foreach方法和Lambda表達式 53](#_Toc207703207)

[1. 語法： 53](#_Toc207703208)

[2. Lambda表達式： 53](#_Toc207703209)

[(1) 語法結構： 53](#_Toc207703210)

[(2) 無參數的一般型態： 54](#_Toc207703211)

[(3) 單一參數的一般型態： 54](#_Toc207703212)

[(4) 多參數的一般型態： 54](#_Toc207703213)

[二十三、 stream API 55](#_Toc207703214)

[1. filter (過濾器)： 55](#_Toc207703215)

[2. sorted (排序)： 56](#_Toc207703216)

[4. sum (加總)： 59](#_Toc207703217)

[5. max (最大值)： 60](#_Toc207703218)

[6. min (最小值)： 61](#_Toc207703219)

[7. foreach (迴圈)： 62](#_Toc207703220)

[8. isPresent (是否存在) 63](#_Toc207703221)

[9. 單執行緒 (stream) 與 多執行緒 (parallelStream)： 64](#_Toc207703222)

[二十四、 集合變數的排序 65](#_Toc207703223)

[1. 單一元素的排序： 65](#_Toc207703224)

[2. 兩個元素的排序： 67](#_Toc207703225)

[3. 含有null的排序： 69](#_Toc207703226)

[二十五、 存取檔案(I/O) 70](#_Toc207703227)

[1. 新增與刪除： 70](#_Toc207703228)

[2. 寫入： 72](#_Toc207703229)

[3. 讀取： 74](#_Toc207703230)

[4. 複製檔案、刪除檔案： 76](#_Toc207703231)

[二十六、 JDBC 78](#_Toc207703232)

[1. 掛載Database Driver： 78](#_Toc207703233)

[2. 連線Database： 78](#_Toc207703234)

[3. 執行SQL：（查詢以外） 79](#_Toc207703235)

[4. 查詢SQL： 81](#_Toc207703236)

[二十七、 執行緒(Thread) 84](#_Toc207703237)

[1. 主執行緒與副執行緒： 84](#_Toc207703238)

[2. 程式暫停： 84](#_Toc207703239)

[3. 多工作業： 85](#_Toc207703240)

[4. 取消同步化： 87](#_Toc207703241)

[二十八、 Future：異步任務 – 異步阻塞 【Java 5 舊寫法】 89](#_Toc207703242)

[二十九、 CompletableFuture：異步任務 – 異步不阻塞 【Java 8 新寫法】 92](#_Toc207703243)

[三十、 時間API 96](#_Toc207703244)

[1. 獲取目前電腦日期： 96](#_Toc207703245)

[2. 轉換輸出格式： 96](#_Toc207703246)

[3. 常見的時間格式： 96](#_Toc207703247)

[三十一、 字串API 98](#_Toc207703248)

[1. length (查詢字串長度)： 98](#_Toc207703249)

[2. substring (輸出部分字串)： 98](#_Toc207703250)

[3. replace (取代) 98](#_Toc207703251)

[4. compareTo (字串比大小)： 98](#_Toc207703252)

[5. matches (字串模糊比對)： 99](#_Toc207703253)

[6. StringBuilder (高性能的字串處理器) 99](#_Toc207703254)

[三十二、 數字表示法 101](#_Toc207703255)

[1. 語法： 101](#_Toc207703256)

[2. 數字格式： 101](#_Toc207703257)

[三十三、 精確的數字運算 103](#_Toc207703258)

[1. 宣告： 103](#_Toc207703259)

[2. 常用方法： 103](#_Toc207703260)

[(1) 運算： 103](#_Toc207703261)

[(2) 兩數比較： 103](#_Toc207703262)

[(3) 與0比較： 103](#_Toc207703263)

[(4) 轉型成數字： 103](#_Toc207703264)

[(5) 四捨五入、無條件進位、無條件捨去 106](#_Toc207703265)

[三十四、 Field：取得 與 修改 物件屬性(值) 107](#_Toc207703266)

[三十五、 附錄：透過反射執行Method 108](#_Toc207703267)

# Java的主程式架構



# 基本資料型態

## 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

## 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。

## 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：

1. 僅能用於 區域變數。
2. 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

# 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

\*/

# 跳脫字元、強制轉換與資料輸出

## 跳脫字元：

|  |  |
| --- | --- |
| \t | Tab |
| \n | 換行 |

＊此為字串中的一些特殊符號。

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 = "\\" |

## 資料輸出：

1. 同行輸出：

System.out.print(輸出內容);

1. 換行輸出：

System.out.println(輸出內容);

# 變數宣告

## 基本：

1. 語法：



1. 範例：

public static final double pi = 3.14;

## 一維陣列：

1. 靜態宣告：



＊陣列的變數計數從0開數。

1. 動態宣告：



＊陣列的變數計數從0開數。

## 二維陣列：

1. 靜態宣告：



＊陣列的變數計數從0開數。

1. 動態宣告：



＊陣列的變數計數從0開數。

# 運算子

## 算數運算子：

|  |  |
| --- | --- |
| + | 加法 |
| - | 減法 |
| \* | 乘法 |
| / | 除法 |
| % | 求餘數 |

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。 |

## 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。

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 |

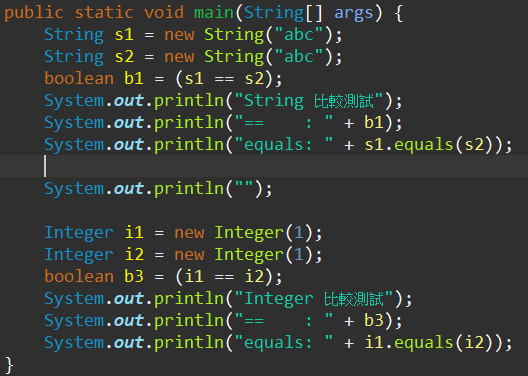
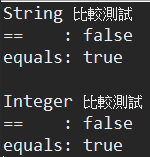
＊關係運算子會回傳True或False。

＊「==」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

位元運算子：

|  |  |
| --- | --- |
| & | 對二進位的每一格進行 and 判斷 |
| | | 對二進位的每一格進行 or 判斷 |
| ^ | 對二進位的每一格進行 xor 判斷 |
| ~ | 補數，正負數反轉 |

位移運算子：

|  |  |
| --- | --- |
| << | 對二進位的每一格向左移動 |
| >> | 對二進位的每一格向右移動 |

# 邏輯判斷

## 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;

}

## 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)。

## 三元運算子：

1. 語法：

布林條件式 ? True儲存的值 : False儲存的值

1. 範例：

cn = (chi >= 60 ? "及格" : "不及格");

＊當chi >= 60，則cn = "及格"，否則cn = "不及格"

# 迴圈

## 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。

1. 範例：

int i;

for (i = 1 ; i < 10 ; i++) {

System.out.println(i);

}

## 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;

}

## 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。

## 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。

## 迴圈控制：

1. bleak：離開迴圈。
2. continue：離開「該次」迴圈。

# 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。

## 物件初始化：

1. 語法：



1. 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

## 物件的撰寫：



＊建構式的名稱必須跟物件名稱相同。

# 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：

## 不傳值的方法：

權限 void 方法名稱(資料型態 參數, ……){

內容;

}

## 傳值的方法：



＊方法的類型可以是：  
 1) int、String等基本資料型態。  
 2) class。

# 陣列的常用方法

使用前需要impory java.util.Arrays。

## 由小到大排序：

1. 語法：

Arrays.sort(陣列名稱);

1. 範例：

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

## 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」。

1. 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

# 繼承

## 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;

}

}

## 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。



## 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

1. 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, ……);

1. 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;  
}

## 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，
3. 改寫後方法的權限要相同或是更大。
4. 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。
5. 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;

}

# 多型

## 語法：



## 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」。

## 缺點：

使能使用「繼承」或「override」的方法。

## 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

# 抽象類別與抽象方法

## 語法：



## 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」。

## 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)

# interface (介面)

## 語法：

【介面】



【類別】



## 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  
{  
 ……  
}

# 內部類別 與 匿名類別

## 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;

}

}

}

1. 宣告內部類別的物件 與 使用方法：
2. 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。

1. 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。

1. 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。

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：
3. 與「抽象類別」搭配使用，也就是「透過匿名類別來override」。
4. 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。
5. 語法：
6. 抽象類別 物件變數 = new 抽象類別(參數, ……)

{

override抽象方法;

};

1. static {

內容;

}

# 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：
3. 不允許「數字開頭」。
4. 不允許「中間有空格」
5. 內部類別中可以有多個enum。
6. 語法：
7. enum 列舉類名稱A

{

字串A1, 字串A2, 字串A3, ……

}

1. 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, ……

}

}

1. 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。

1. 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」

1. 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());

# 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。

## 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：
2. 嚴重錯誤，會強制中斷程式，並拋出錯誤訊息。
3. 錯誤訊息皆繼承java.lang.Error。
4. 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。
5. Runtime Exception：
6. Java預先決定好的「運行時錯誤訊息」。
7. 錯誤訊息屬於java.lang.RuntimeException，  
   繼承自java.lang.Exception。
8. 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。
9. Checked Exception：
10. 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。
11. 繼承自java.lang.Exception的錯誤訊息，  
    除了java.lang.RuntimeException外，都屬於此類型。
12. 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。

## 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。

1. 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。

## 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。

1. 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

# 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。

## 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

}

}

1. 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);

}

}

## 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

}

}

1. 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);

}

}

## 使用泛型：

1. 語法：



1. 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。

# 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」。

## 常見的集合的種類：



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。

1. 常用方法：
2. ArrayList變數.add(value)：

新增資料

範例：  
arraylist.add("1");

1. ArrayList變數.remove(index)：

刪除特定資料 (後面資料往前遞補)

範例：  
arraylist.remove(2);

1. ArrayList變數.set(index, value)：

修改資料

範例：  
arraylist.set(3, "5");

1. ArrayList變數.get(index)：

取出資料

範例：  
System.out.println(arraylist.get(0));

1. ArrayList變數.size()：

查詢長度

範例：  
int length = arraylist.size();

1. 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");

1. 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);

1. 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);

}

}

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。

1. 常用方法：
2. HashSet變數.add(value)：

新增資料

範例：  
hashset.add(1);

1. HashSet變數.remove(value)：

刪除指定資料

範例：  
hashset.remove(1);

1. 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);

1. HashSet變數.size()：

查詢長度

範例：  
int length = hashset.size();

1. 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);

1. 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);  
}

1. 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);

}

}

}

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。

1. 常用方法：
2. PriorityQueue變數.add(value)：

新增資料

範例：  
priority.add(1);

1. 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);

1. 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);

1. PriorityQueue變數.size()：

查詢長度

範例：  
int length = priority.size();

1. 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);  
}

1. 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);  
}

1. 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);

}

}

}

# 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。

## Key區：

索引區，資料型態為「String」。

其保存的內容「不可重複」。

## Value區：

值區，資料型態為「包覆類別」。

其保存的內容「可以重複」。

HashMap：

* 1. 特點：

用「亂數排序」的方式來儲存Key區。

* 1. 宣告語法：

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

* 1. 常用方法：

1. HashMap變數.put("key", value)：

新增資料

範例：  
hashmap.put("a", 1);

1. HashMap變數.remove("key")：

刪除指定資料

範例：  
hashmap.remove("a ");

1. 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");

1. HashMap變數.size()：

查詢長度

範例：  
int length = hashmap.size();

1. HashMap變數.keySet()：

輸出key區資料，資料型態為Set

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

* 1. 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);

}

}

}

# 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。

## 語法：

集合變數.foreach(Lambda表達式);

## Lambda表達式：

### 語法結構：

1. 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;

……

}

1. 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」。

### 無參數的一般型態：

() -> 內容;

() -> {

內容A;

內容B;

……

}

### 單一參數的一般型態：

x -> 內容;

x -> {

內容A;

內容B;

……

}

### 多參數的一般型態：

(x, y, ……) -> 內容;

(x, y, ……) -> {

內容A;

內容B;

……

}

範例：

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);

       });

    }

 }

# 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。

## 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。

1. 語法：

依照輸出元素的數量，可以分成2大類：

1. 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

1. 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()

1. 範例：

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]

## 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。

1. 語法：
2. 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);

1. 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()

1. 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]

1. 範例2：多元素-指定特定元素的排序



map (映射)：

1. 功能：

取出集合變數中，符合指定規則的元素。

1. 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()

1. 範例：

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]

    }

}

## 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。

1. 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

1. 範例：

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

    }

}

## 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。

1. 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

1. 範例：

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

    }

}

## 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。

1. 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

1. 範例：

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

    }

}

## foreach (迴圈)：

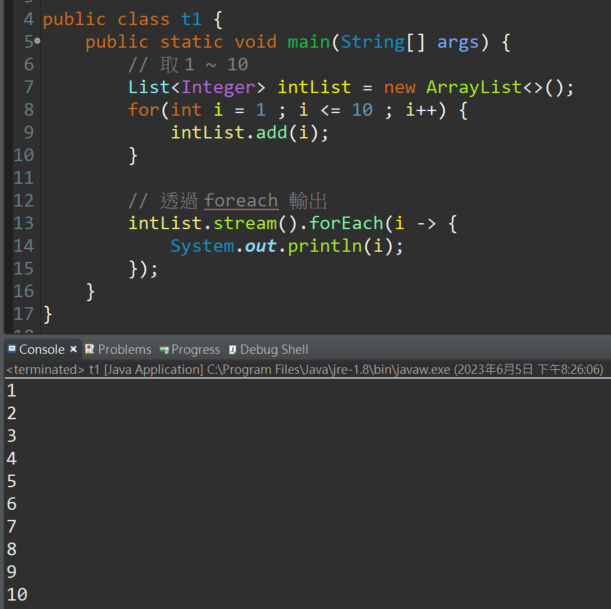
* 1. 功能：

針對集合變數中的元素，進行foreach。

* 1. 語法：

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

* 1. 範例：



## 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。

* + 1. 透過filter 篩選條件
    2. 透過findAny 設定輸出全部元素
    3. 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
  1. 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

* 1. 範例



## 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。

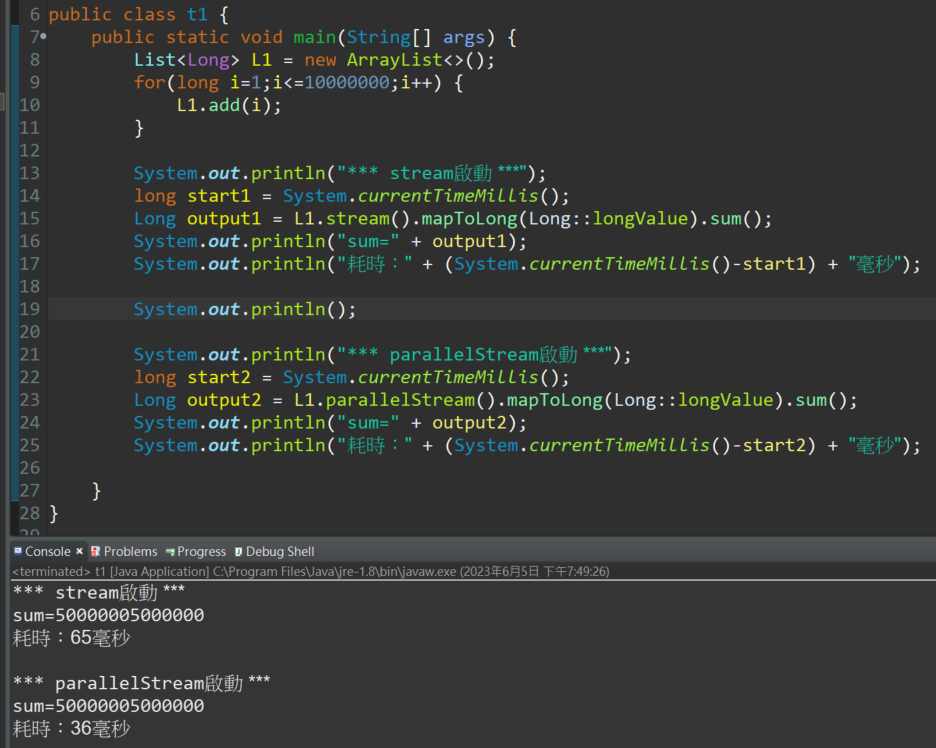
如果 需要處理的運算龐大，可以考慮使用『parallelStream』。  
將可以有效地節省時間。



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。



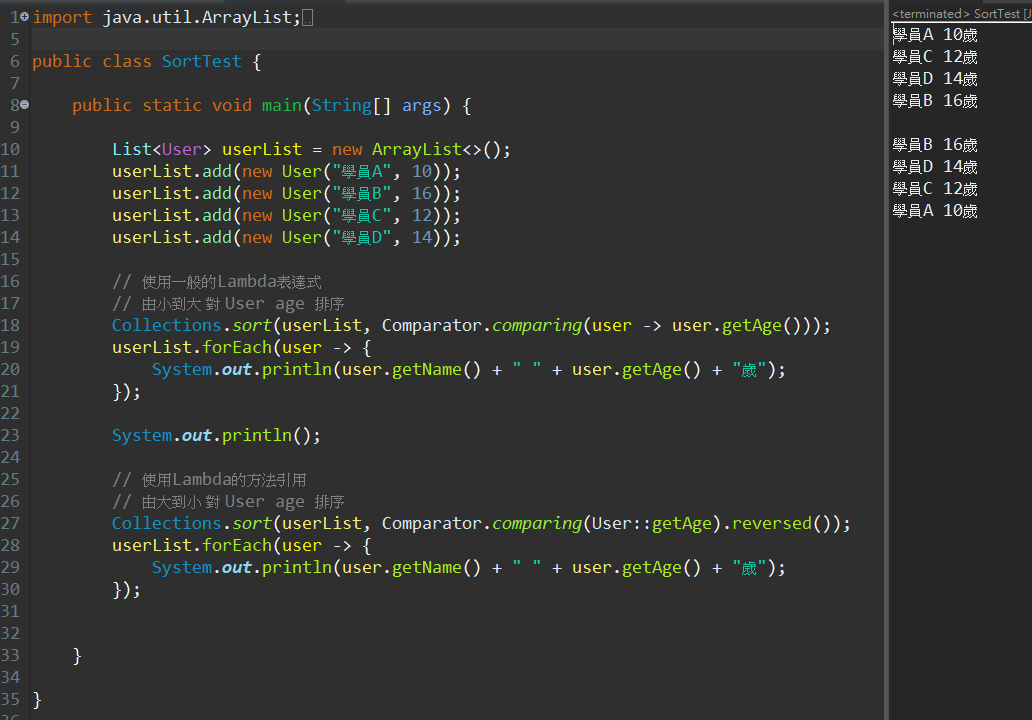
# 集合變數的排序

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()](#_排序(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)

## 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()』



範例2：使用『集合變數.sort()』



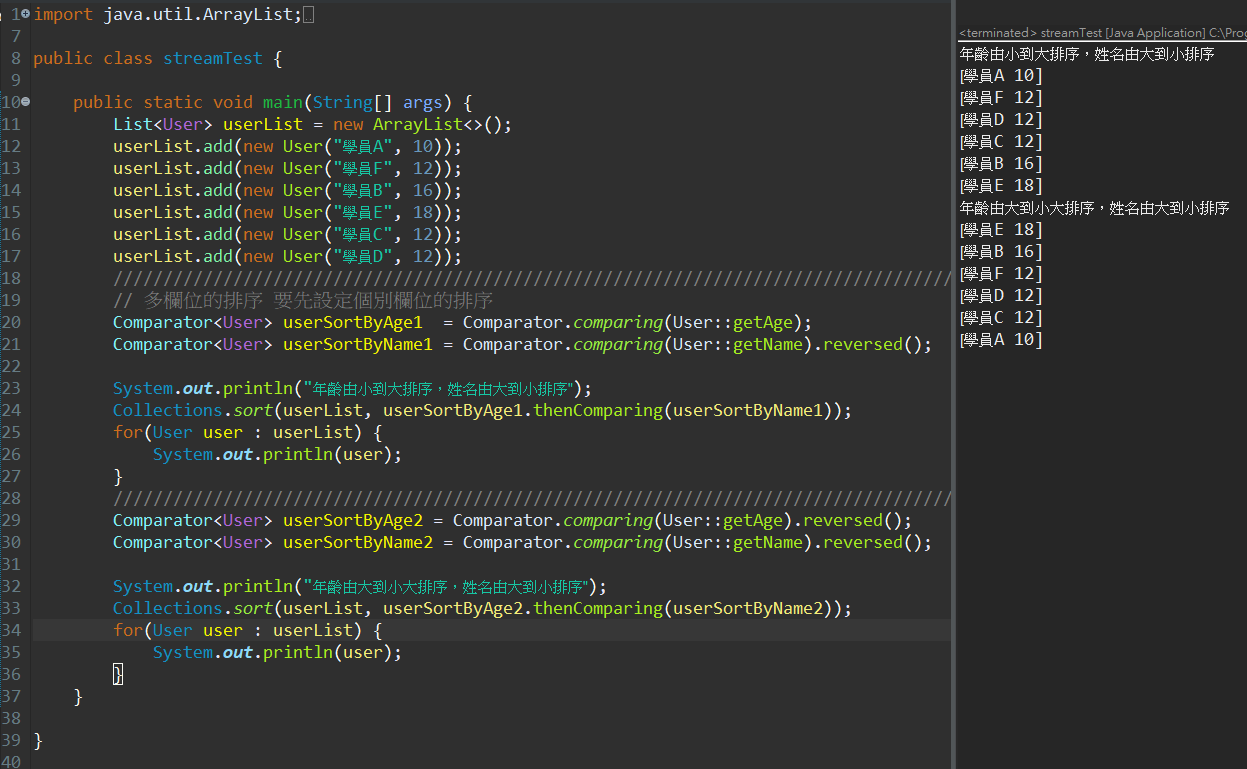
兩個元素的排序：

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

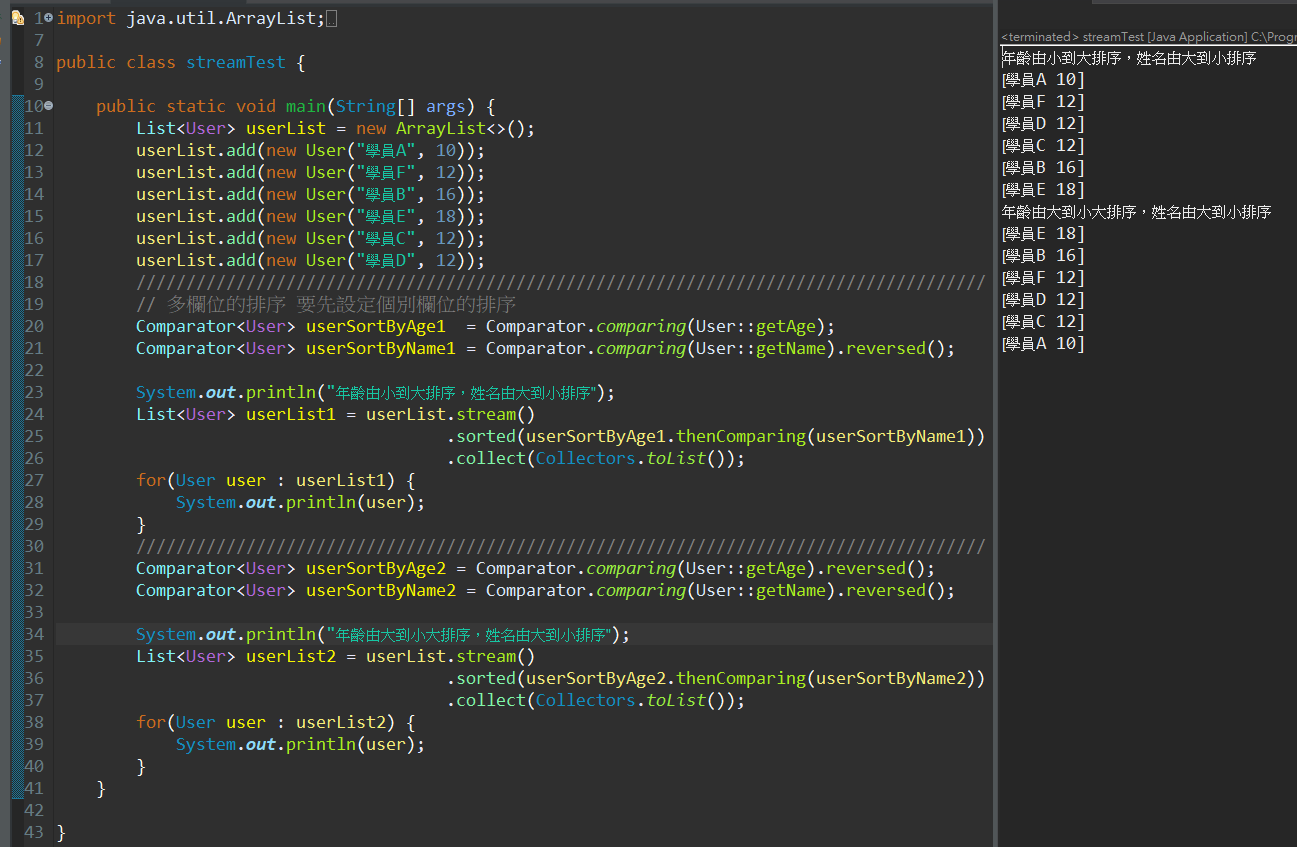
//1. 設定排序元素: 各別元素 由小到大 或 由大到小 於這裡設定  
Comparator<變數類型> 排序變數A ← 由小到大  
 = Collections.sort(集合變數, Comparator.comparing(排序元素A));  
Comparator<變數類型> 排序變數B ← 由大到小  
 = Collections.sort(集合變數, Comparator.comparing(排序元素B).reversed());

// 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



範例2：stream().sort() 也可以做到兩個元素的排序



## 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：



# 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。

## 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");

1. 常用方法：
2. 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();

1. 檔案變數.createNewFile()：

新增檔案

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

1. 檔案變數.mkdir()：

新增資料夾

範例：  
File dir = new File("a");  
dir.mkdir();

1. 檔案變數.delete()：

刪除檔案(資料夾)

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

1. 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!");

}

}

}

## 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。

1. 常用方法：
2. 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");

1. 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();

1. 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();

}

}

## 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。

1. 常用方法：
2. 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);  
}

1. 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();

1. 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();

}

}

## 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");

1. 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");

1. 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();

1. 複製檔案：

Files.copy(來源路徑變數, 目標路徑變數);

範例：  
Files.copy(inputPath, outputPath);

1. 刪除檔案：

Files.delete(刪除檔案的路徑變數);

範例：  
Files.delete(output.toPath());

1. 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");

}

}

# 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。

## 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");

連線Database：

1. 語法：



1. 範例：(MySQL)

String URL = "jdbc:mysql://localhost:3306/test";

String User = "root";

String Password = "root";

Connection conn = DriverManager.getConnection(URL, User, Password);

執行SQL：（查詢以外）

1. 宣告傳送變數：

PreparedStatement 傳送變數;

1. 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。

1. 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～）。

1. 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();

1. 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();

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。

1. 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】

1. 查詢變數.first()：

指向「第一筆」資料

1. 查詢變數.last()：

指向「最後一筆」資料

1. 查詢變數.next()：

指向「下一筆」資料

1. 查詢變數.previous()：

指向「上一筆」資料

1. 查詢變數.absoulte(index)：

指向「第index筆」資料

1. 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");

1. 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。

1. 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);

# 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。

## 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」。

程式暫停：

1. 語法：

Thread.sleep(毫秒);

＊1秒 = 1000毫秒

1. 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");

}

}

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

1. 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();

}

}

}

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;

}

}

1. 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();

}

}

}

# 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

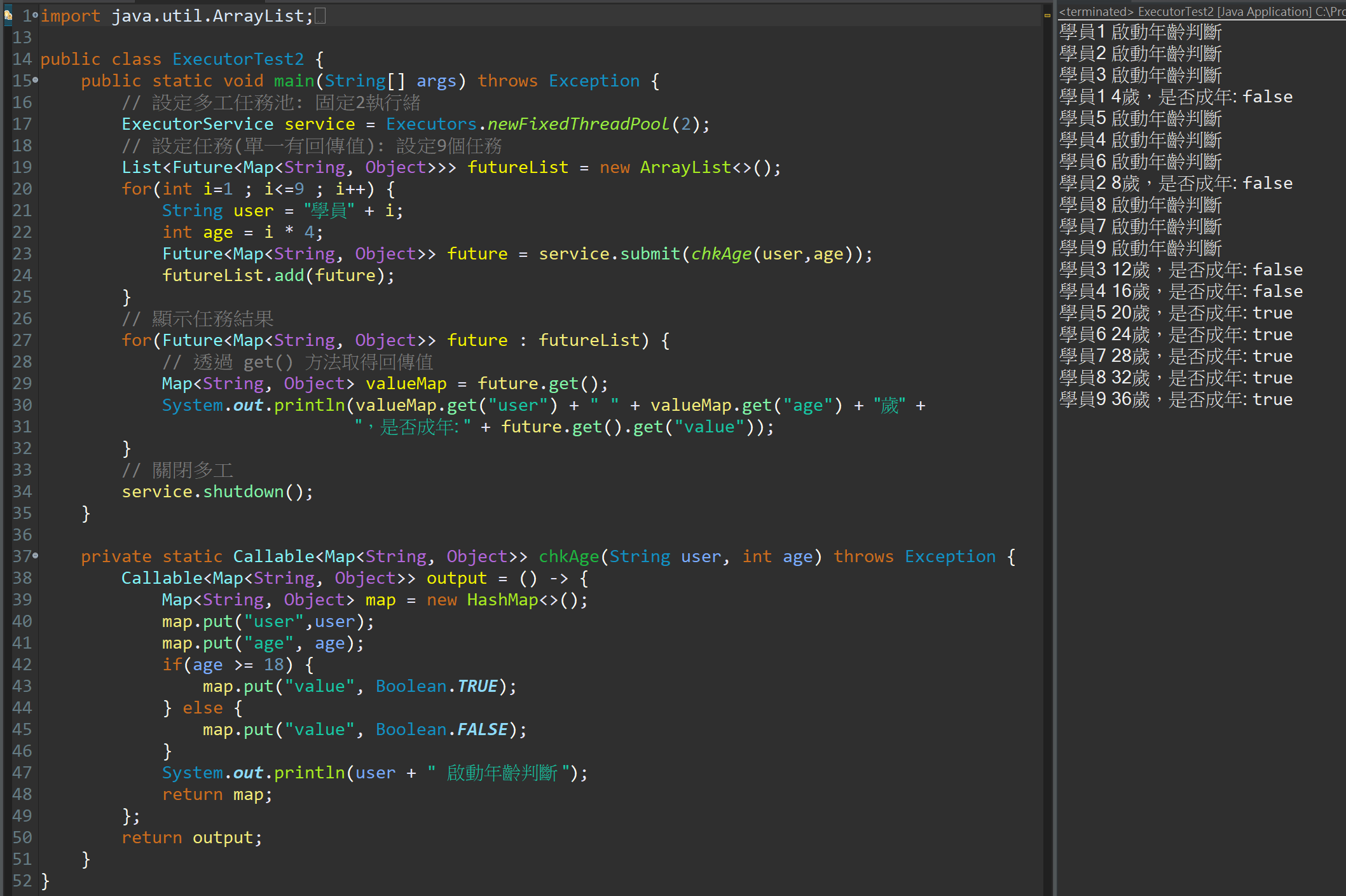
(2) 有回傳值的任務，要用Future接收回傳，並透過get()來取出回傳值 (阻塞主線程)



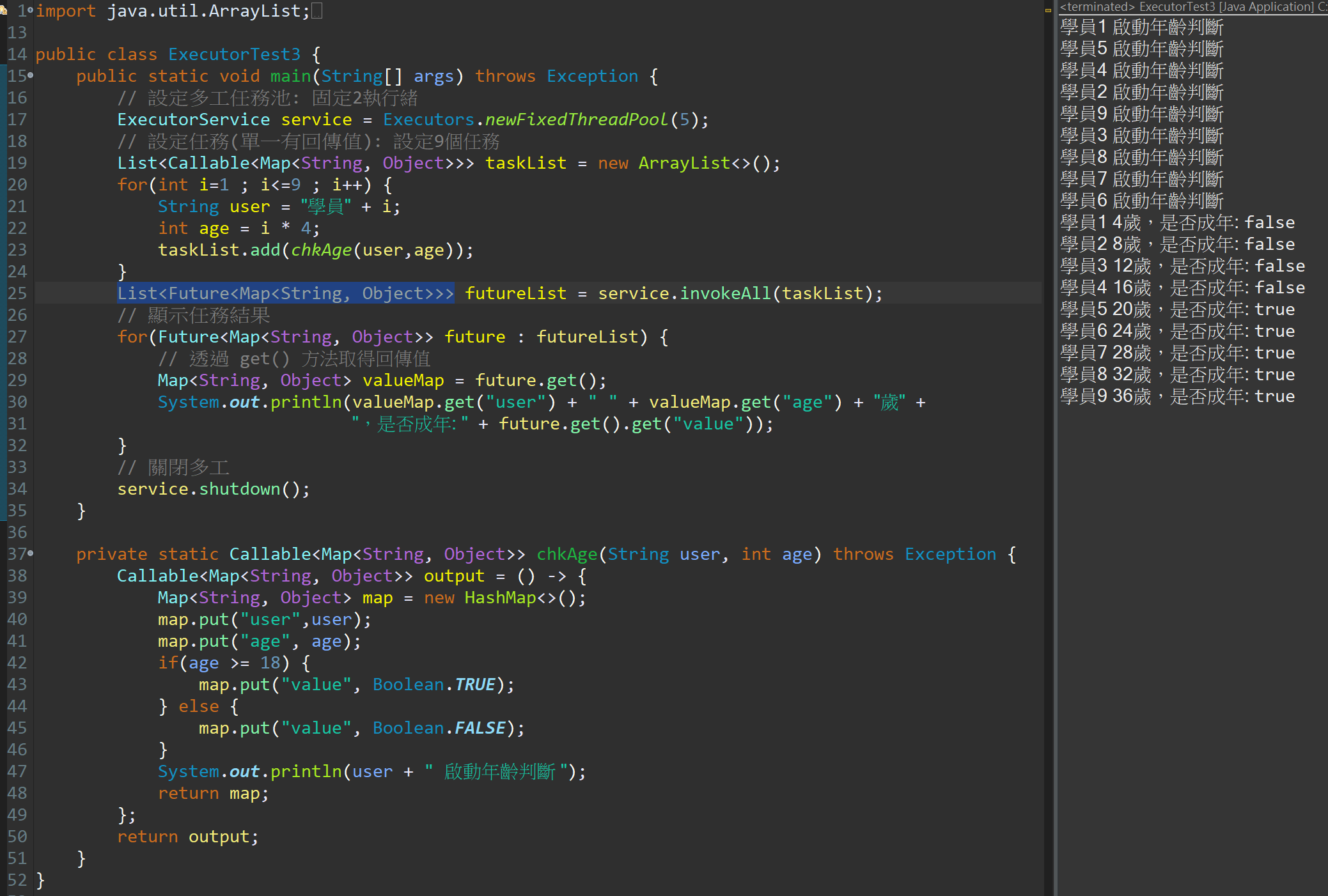
1. 範例1 execute：



1. 範例2 submit：



1. 範例3 invokeAll：



# CompletableFuture：異步任務 – 異步不阻塞 【Java 8 新寫法】



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<>();

1. 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);

* 1. 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);

* 1. 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

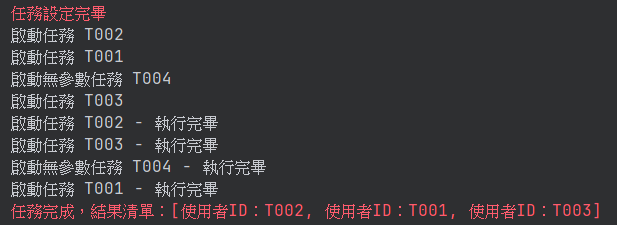
CompletableFuture<String> future2 =   
 CompletableFuture.*supplyAsync*(() -> 有回傳任務(參數))  
 .*thenApply*(result -> {  
 你要進行的邏輯處理  
 return result;  
 });  
futureList.add(future2);

* 1. 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);

1. 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();

1. 範例  
   



# 時間API

## 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

轉換輸出格式：

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

常見的時間格式：

1. yyyy/MM/dd：

西元年/月/日

範例：  
2019/8/26

1. ahh:mm:ss：

12小時制的時間

範例：  
下午03:04:15

1. HH:mm:ss：

24小時制的時間

範例：  
15:04:15

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

# 字串API

## 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

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

## 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

## 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.comparaTo(後字串);

範例

String date1 = "113/05/21";

String date2 = "114/01/01";

System.*out*.println(date1.compareTo(date2)); // 輸出 -1

## 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

## 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

# 數字表示法

## 語法：

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

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。

1. 「#」：

該格不一定有值，沒值就不會顯示。

1. 「%」：

此數字為百分數，會數字自動「乘以100」，並將「%」顯示在最後。

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%

# 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。

## 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。

## 常用方法：

### 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) |

### 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 |

### 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 |

### 轉型成數字：

1. int 回傳值 = 儲存變數.intValue();
2. long 回傳值 = 儲存變數.longValue();
3. float 回傳值 = 儲存變數.floatValue();
4. 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);

    }

### 四捨五入、無條件進位、無條件捨去

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

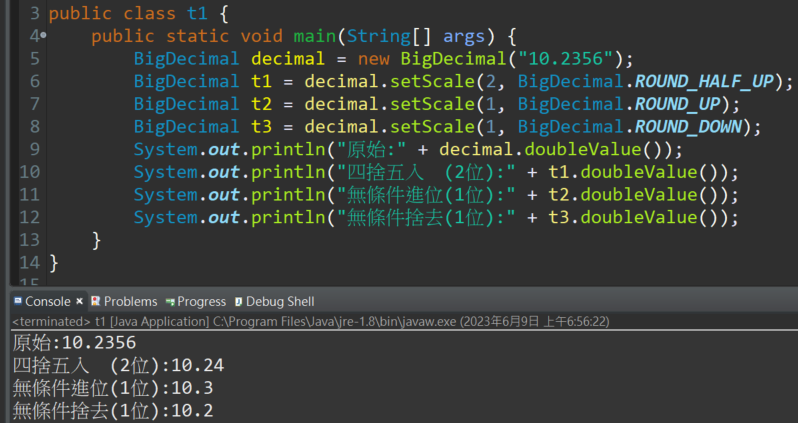
儲存變數 = 儲存變數.setScale(小數位數, 捨入型態);

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：



# 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

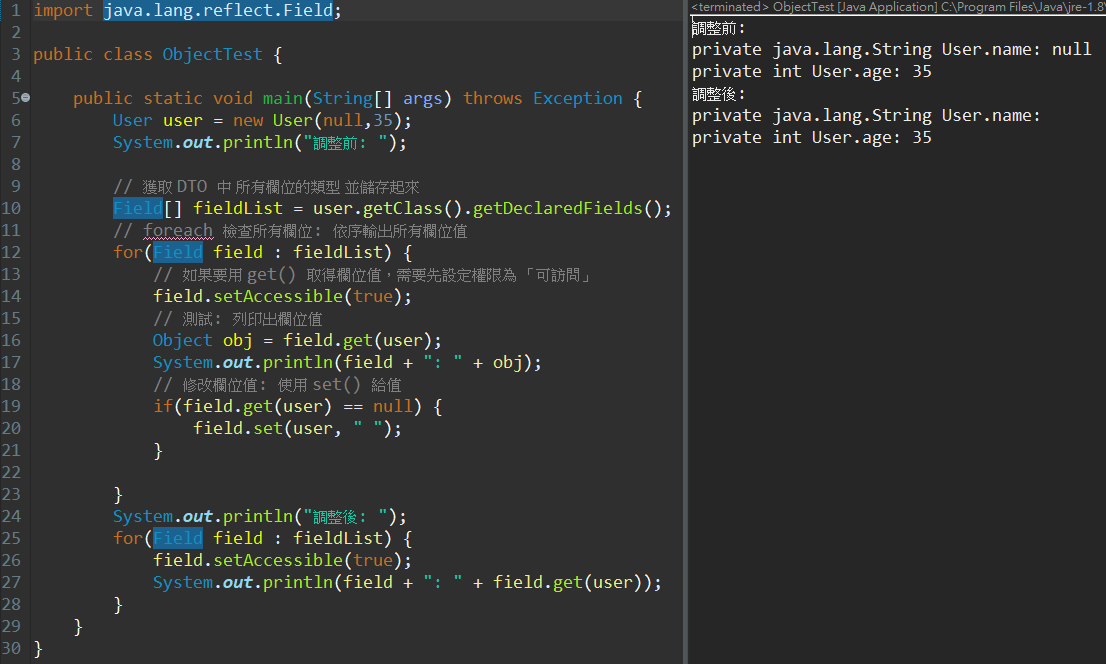
如：當 欄位值 = Null 時，需要 將其改成 空格  
　　此時，可以透過 Field 來達成這個目的  
　　先 取得 物件的所有欄位  
　　再 逐一檢查 該欄位的欄位值 是否為 Null  
　　 如果為 Null 就改成 空格

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);  
}

1. 範例：



# 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。

