Java 程式設計講義

目次

物件導向程式設計講義

Chapter 1　Java 的變數及運算式

1. 基本程式概念複習

- 對於電腦來說，它真正所懂得的語言只有一種，就是機器語言。所謂的機器語言，其實就是以特定的數字來表示電腦所能進行的各個動作，這些數字就稱為機器碼 (Machine Code)。
- 高階語言不但比較容易理解，也不會像是低階語言所寫出來的程式那樣繁瑣。不過電腦並無法看懂這樣的程式，需要一個轉譯的動作，將使用高階語言所撰寫的程式轉換成電腦所能看懂的機器語言，然後才能依此執行。這個轉換的動作就是由各種程式語言的編譯器 (Compiler) 所進行。
- 物件導向程式語言：隨著軟體技術的不斷演進，程式語言又從高階語言改良，隨之而流行的就是物件導向程式語言 (Object-Oriented Language)，比較著名的有 Smalltalk、C++、以及本課程的主題 Java。
- Java 編譯程式輸出的程式碼稱為位元組碼(ByteCode)，它並不是可執行的程式碼，而必須由虛擬機器(Java Virtual Machine，簡稱 JVM)解譯執行。
- Java 虛擬機器(Java Virtual Machine)，用來解譯及執行位元組碼的程式，位元組碼透過 JVM 解譯後，可以在安裝 JVM 的硬體上，執行該程式
- 程式範例

```
/*
    檔案名稱：  Test.java
*/
public class Test
{  // 主程式
   public static void main(String[] args)
   {
       System.out.println("第一個Java 應用程式");  // 顯示訊息
   }
}
```

- 大小寫有別 - 在 Java 裡，英文字母的大小寫是不一樣的，因此，Hello 與 hello 是不一樣的
- 類別定義：
  1. 定義類別時，以 class 關鍵字為開頭，接著是類別的名稱。
  2. 類別的前面可以加 public 關鍵字或不加任何關鍵字，兩者有不同的意義。
  3. 如果一個程式檔案只有一個類別定義時，類別名稱就是檔案名稱。
  4. 類別的主體以大括號『{ }』包圍起來。
- 起始方法定義：
  1. main()方法必須宣告為 public（公開）、static（靜態）、void（沒有回傳值）。
  2. 其形式參數必須為 String 型別的一維陣列。
  3. 方法的主體也以大括號『{ }』包圍起來。

- 程式結尾符號 - 『；』 ，每一行程式敘述都以『；』符號結尾
- 多行註解（或稱區塊註解）：
  1. 以 /* 為起始符號、以 */ 為終止符號。
  2. 註解符號 /* 和 */ 之間可放入任何文字，以輔助程式設計者了解程式碼在做什麼。
  3. 所有的程式註解都會被編譯器忽略。
  4. 不能有巢狀註解。 /* ***** /* 錯誤的註解 */ ***** */
- 單行註解：
  以 // 為起始符號，終點為該行的最後一個字元，沒有終止符號。

2. 變數

- 變數的目的是儲存程式執行中的一些暫存資料，程式設計者只需記住變數名稱，而且知道名稱表示一個記憶體位置中的資料，至於這個記憶體位置到底有哪裡？並不用傷腦筋，因為這是編譯程式的工作。
- 簡單的說，程式語言的變數是使用有意義的名稱代表數字的記憶體位址。
- Java 擁有 8 種基本資料型態 byte、int、short、long、float、double、char 和 boolean。例如：整數變數宣告的範例，如下所示：

  int *score;*

  int *i, j, score;*
- 宣告變數的用意：
  1. 設定變數所佔的記憶體大小
  2. 不會發生資料因無心而蓋掉的錯誤(相同名稱的變數不能重複宣告)
- 一個常數值符（literal）就表示一個數值，變數所代表的數值是可以變更的。

  例： *j=36;*

3. 變數的命名規則

- 識別字必須以英文字母開頭, 大小寫均可。另外, 也可以用 “_” 或是 “$” 這兩個字元開頭。像是 “3am” 或是 “!age” 就不能作為變數的名字。
- 跟著開頭字元之後的, 可以是符合前一條規則的字元, 或者是阿拉伯數字 0 ~ 9。像是 "apple" 或是 "apple1" 都可以作為變數的名字, 但 "apple!" 就不行。

4. Java 的關鍵字：(只要是關鍵字，必定是小寫)

- boolean、char、byte、short、int、long、float、double
- if、else、switch、break、continue、return、case、do、while、for、goto、const
- new、this、super、void、class、extends、import、package、implements、instanceof、interface
- try、catch、finally、throw、throws

- public、private、protected、default、final、abstract、static
- native、strictfp、synchronized、transient、volatile

5. 常數

- 「常數」(Named Constants)是指一個變數在設定初始值後，就不會變更其值，簡單的說，就是在程式中使用一個名稱代表一個固定值。
- Java 的常數宣告和指定初值的變數宣告相同，只需在前面使用 final 關鍵字，如下所示：

```
final double PI = 3.1415926;
public class Test2
{
   public static void main(String[] args)
    {   // 常數宣告
        final double PI = 3.1415926;
        // 變數宣告
        double area;
        double r = 20.0;
        // 計算面積
        area = PI * r * r;
        System.out.print("圓面積: ");
        System.out.println(area);
    }
}
```

6. 變數的命名原則

- 為了程式撰寫的一致性和易於閱讀，Java 習慣的命名原則，如下表所示：

	習慣命名原則	範例
常數	使用英文大寫字母和底線”_”符號	MAX_SIZE、MIN_SIZE
變數	英文小寫字母開頭，如果 2 個英文字組成，第 2 個英文字以大寫開頭	size、screenSize、myAccountNumber
類別	英文大寫字母開頭，如果是 2 個英文字組成，第 2 個英文字也使用大寫開頭	LargeRoom、SmallRoom
函數（方法）	英文小寫字母開頭，如果是 2 個英文字組成，第 2 個英文字使用大寫開頭	pressButton、scrollScreen

7. 資料型態

- Java 的資料型態分為「基本」(Primitive)和「參考」(Reference)兩種資料型態，如下：
- 基本資料型態：變數有 byte、short、int、long、float、double、char 和 boolean 共 8 種資料型態。

- 參考資料型態：變數值是一個記憶體位置，這個位置值是物件儲存的位置。

8. 基本資料型態範圍及初始值：

Char	'\u0000'	16	'\u0000' ~ '\uFFFF' *	0 ~ 65535
Byte	0	8	$-2^7 \sim 2^7-1$	-128 ~ 127
Short	0	16	$-2^{15} \sim 2^{15}-1$	-32768 ~ 32767
Int	0	32	$-2^{31} \sim 2^{31}-1$	
Long	0L	64	$-2^{63} \sim 2^{63}-1$	
Float	0.0F	32		
Double	0.0D	64		
Boolean	false	1	true / false	
所有參考型別	Null			

- 在程式碼直接使用包含 0、正整數和負整數的「整數值」，可以使用十進位、八進位和十六進位表示，如下所示：
  - 八進位："0"開頭的整數值，每個位數值為 0~7 的整數。
  - 十六進位："0x"開頭的數值，位數值為 0~9 和 A~F（或 a~f）。

	十進位值	說明
44	44	十進位整數
0256	174	八進位整數
0xef	239	十六進位整數
0x3e6	998	十六進位整數

- Java 語言可以使用數值字尾型態的字元將文字值指定成 long 長整數
  - 長整數表示法：

    ***long** a = 123;* //以一般整數設定給長整數變數

    ***long** b = 123**L**;* //以長整數設定給長整數變數
  - long 型別的數值不可設定給 int 型別的變數：

    ***int** c = 123**L**;* //錯誤的設值

- 「浮點數資料型態」(Floating Point Types）是指整數加上小數，例如：3.1415926、123.567等，依照長度的不同（即變數佔用的記憶體位元數），分為 2 種浮點數的資料型態。
  - 在 Java 程式碼直接使用浮點數值預設是 double 資料型態，而不是 float。可以使用科學符號的”e”或”E”符號代表 10 為底的指數，一些浮點數的範例，如下表所示：

	十進位值	說明
0.0123	0.0123	浮點數
.00567	.00567	浮點數
1.25e4	12500.0	使用 e 指數科學符號的浮點數

  - double 型別的值不能設定給 float 型別的變數。

    ***float** ft1 = 1.23**f**;*

    ***float** ft2 = 1.234**F**;*

    ***float** ft3 = 1.234;* //錯誤

    ***float** ft3 = 10**E**2;* //錯誤

- 「字元資料型態」(Char Type）是「無符號」(Unsigned）的 16 位元整數所表示的 Unicode 字元，Unicode 字元是使用 2 個位元組表示字元，可以用來取代 ASCII 字元，使用一個位元組的表示方式。
  - char 的常數值符為單引號括起來的字元或某個特定的 Unicode character(萬國碼字元)，其中\u????是 16 進制的數字，且一定得是四個，不可多不可少。

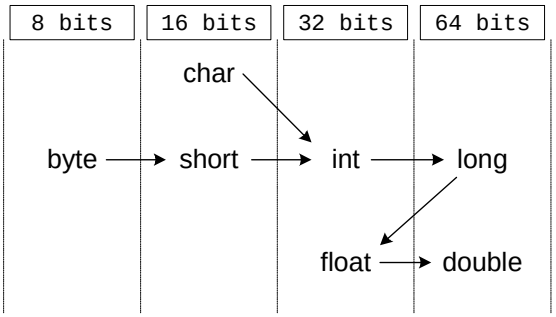
    例: char c = '\u0020'; // 空白字元（Space）。
  - char 也可宣告為 0~65535 的數字而不需加上單引號’ (0 或-0 都不會產生錯誤) 。

- 字串文字值就是一個字串，字串是 0 或多個依序的字元文字值使用雙引號「"」括起的文字內容。
  - 在 Java 語言的字串是一種字串物件，屬於參考資料型態，目前程式碼的字串主要是使用在 System.out.println()或 println()方法的參數。
  - String 類別初始化的預設值為 null。
  - String = “” 和 String = null 意思不同(String a=null;System.out.print(a);//*會 print 出”null”*) 。

- Java 提供 Escape 逸出字元，這是使用"\"符號開頭的字串，可以顯示一些無法使用鍵盤輸入的特殊字元，如下表所示：
  - 合法的跳脫字元【 **b**ig **f**oot **n**eed **r**ed **t**ie 】。

	Unicode 碼	說明
\b	\u0008	Backspace，Backspace 鍵
\f	\u000C	FF，Form feed 換頁符號
\n	\u000A	LF，Line feed 換行符號
\r	\u000D	CR，Enter 鍵
\t	\u0009	Tab 鍵，定位符號
\'	\u0027	「'」單引號
\"	\u0022	「"」雙引號
\\	\u005C	"\"符號

9. 基本型別轉換



- 同資料型態，小體積的型別值可以設定給大體積的型別變數，反之則否。
- boolean 型別不同於其它基本型別，不能和其它基本型別相互成為設定值。
- char 型別可以視為 int 型別，但所有整數型別皆不能視為 char 型別。
- 所有整數型別皆可以設定給所有浮點數型別，反之則否。
- -0.0 == 0.0 is true。
- 不同型別在做數學運算前，compiler 會自動將小的型別改成大的型別。
- int 與 String 的轉換為：Integer.toString() / Integer.parseInt()
- 常見轉型：

floating-point types 轉成 integral types	小數點後面直接截掉

10. 運算式

- 運算子的優先順序

	說明
()	括號
!、-、++、--	條件運算子 NOT、算數運算子負號、遞增和遞減
*、/、%	算術運算子的乘、除法和餘數
+、-	算術運算子加和減法
<<、>>、>>>	位元運算子左移、右移和無符號右移
>、>=、<、<=	關係運算子大於、大於等於、小於和小於等於
==、!=	關係運算子等於和不等於
&	位元運算子 AND
^	位元運算子 XOR
|	位元運算子 OR
&&	條件運算子 AND
||	條件運算子 OR
?:	條件控制運算子
=、op=	指定運算子

- 運算子遇到何種運算元時會做轉型的情況與規則：

一個運算子 ex： byte b,a=3; ~~b=a+1~~; → b=(byte)(a+1)	◆ 若 operand(運算子)是 byte、short、char 則會轉成 int type，否則維持原 type。 ◆ plus、minus：+、- ◆ bit-wise：~ ◆ increment / decrement：++、--
兩個運算子 ex： byte a=1; byte b=2; ~~byte c=a+b~~; → byte c = (byte)(a+b)	◆ 若其中一個 operand 是 double/float/long，則將另一個自動轉型為同 type，否則兩者皆改為 int。 ◆ *、/、% ◆ addition、subraction：+、- ◆ integer bitwise：&、^、| ➢ compound assignment operators：+=
三個運算子	◆ 條件運算子 a = x ? b : c a = (3>2)?0:1;

- 運算時的型別自動轉換表

	byte	short	char	int	long	float	double
byte	int	int	int	int	long	float	double
short		int	int	int	long	float	double
char			int	int	long	float	double
int				int	long	float	double
long					long	float	double
float						float	double
double							double

- 關係與條件運算子-條件運算子

	範例	說明
!	! op	NOT 運算，傳回運算元相反的值，true 成 false，false 成 true
&&	op1 && op2	AND 運算，連結的 2 個運算元都為 true，運算式為 true
||	op1 || op2	OR 運算，連結的 2 個運算元，任一個為 ture，運算式為 true
&	op1 & op2	如果 op1 和 op2 為布林資料型態的變數，此時的運算子如同&&，如果運算元為數字就是下一節的位元運算
|	op1 | op2	如果 op1 和 op2 為布林資料型態的變數，此時的運算子如同||，如果運算元為數字就是下一節的位元運算
^	op1 ^ op2	XOR 運算，連結的 2 個運算元，只需任一個為 true，結果為 true，如果同為 false 或 true 時結果為 false

- && 和 || 稱為快捷運算子（short circuit）

  例1:

  ```
  (6<5) && (39>17) && (a==7)
  ```

  例2:

  (3<9) || (7<12) || (b<=a)

物件導向程式設計講義

Chapter 2　Java 的流程控制及陣列

1. if 流程控制：

語法	注意事項
If (exp) { } Else { }	● **exp 須為 boolean type**。(a==b 可以，但 a=b 不行，要注意) ● else 可有可無，若無明顯{}分開，則是跟著上一個離自己最近的 if。

程式範例：

```
public class Test
{
   public static void main(String[] args)
   {
      int score = 70;
      if ( score >= 60 )
      {   System.out.print("Java 2 程式設計");
          System.out.println("範例教本-及格！");
      }
      else    System.out.println("需要重修Java 2！");
   }
}
```

2. switch 流程控制：

switch (exp) {
　case lable1：
statement1
　default：default
statement
}
	● exp 須為 byte、short、char、int 其中一種 type。
	● case與case間若無break，則會繼續執行下一個case。
	● deralut 可有可無。
	● 不可以兩個 case 放在同一行(~~case 1,2 : statement1~~)

程式範例：

```
public class Test
{
    public static void main(String[] args)
   {
      char grade = 'C';
      switch (grade)
      {  case 'A':    System.out.println("學生成績超過80分");
                      break;
         case 'B':    System.out.println("學生成績為70~79分");
                      break;
         case 'C':    System.out.println("學生成績為60~69分");
                      break;
         default:     System.out.println("學生成績低於60分");
      }
   }
}
```

3. while 流程控制：

While (condition) { }	● condition 須為 boolean type。
Do { } while (condition)**;**	● condition 須為 boolean type。 ● 至少會執行一次。 ● **注意不要遺忘 while 之後的分號。**

4. for 流程控制：

for (init ; boolean ;
increasing)
{
statement
}
	● init 可以是 0 或 1 個以上的 exp，以逗號分隔。若是在 init 中宣告 var，則只能夠有一個 exp (int i=0,j=0;//*算一個*)。但 type 必須相同。
	● boolean 可以是 0 或 1 個 boolean exp，若是空的表示 forever true。
	● increasing 可以是 0 或 1 個以上的 exp，以逗號分隔。
	● init、boolean、increasing 都可以省略，但是兩個分號不可省略。
	● for(a;b;c){d}➔執行時為 abdcbdc…，a 只有在一開始執行一次而已。

程式範例：

```
public class Test
{
   public static void main(String[] args)
   {
      int total = 0;
      // 遞增 for 迴  敘述
      for (int i = 1; i <= 10; i++ )
      {  System.out.print("數值: " + i + " ");
         total += i;
      }
      System.out.println("\n 從小到大的總和: " + total);
   }
}
```

5. 其他：

- break 用於 while、do、for、switch。用以跳出。

  範例:

```
while(true)
{
   if(i==7)
      break;
   System.out.print(i+"\t");
   i++;
}
```

- continue 只能用於 while、do、for，不可於 switch。用以停止目前輪迴，開始下一輪迴。

```
for(int j=0; j<10; j++)
{
     if(j==4)
          continue;      // 跳到迴圈的起始處
     System.out.print(j + "\t");
}
```

- 迴圈標籤

```
landMark:
for (...)
{
   for (...)
   {
      ...
      continue landMark;
      ...
      break landMark;
      ...
   }
   ...
}
```

6. 物件導向基礎

- 參照(引用；reference):用以操控物件的識別字；即物件變數。

例: 1. String s; *//產生一份 String 的 reference*

2. String s = "asdf" *//此時沒有產生實際的物件, 如將訊息傳送給 s, 則發生執行時錯誤.*

3. String s = new String("abcd")

- 物件的生存空間

例: {

String s = new String("a string");

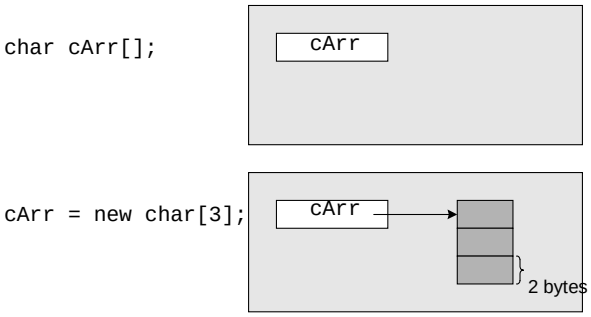
}

  - 當離開了 s 的生存空間，s 的 reference 即消失， 但 s 所指的物件仍繼續佔用記憶體。
  - 如何讓這些物件不會毫無節制佔用記憶體？

- 資源回收機制（garbage collector）
  - 一種會在適當時機逐一檢視透過 new 所產生的物件，並在這些物件不再被引用時，便釋放它們的機制。

7. 陣列：

- 一維陣列
  - 陣列-「多個擁有相同名稱且同型別的變數集合」
  - 陣列元素以陣列名稱，後接一對中括弧“[]”，中括弧內放入位置編號。
  - 位置編號一般也稱為索引，索引一律由 0 開始，接著是 1、2、3…至元素個數減一。
  - 陣列的索引必須是大於或等於 0 的整數，或結果為整數的運算式。
  - 陣列變數其實是一種參考變數（reference variable），陣列名稱是指向陣列實體的一個參考變數。



  - 使用陣列的敘述：

```
int[] iArr = new int[3];     //宣告並配置陣列
iArr[1] = 123;               //設定第2 個元素值為 123
iArr[2] = 456;               //設定第3 個元素值為 456
//將第2 和第3 個元素值相加後設定給第1 個元素
iArr[0] = iArr[1] + iArr[2];
```

  - 取得陣列的長度

    陣列名稱.length
  - 以大括號配置陣列並設值

```
Int[]  myArray = {10, 20, 30, 40, 50, 60};
```

  - 使用陣列元素: 元素預設值
    - 陣列在配置後，元素的內容就會被填入預設值。
    - 陣列的基底型態為整數者預設值為 0。
    - 浮點數則為 0.0f 或 0.0d。
    - boolean 型別為 false。
    - 物件為 null。
- 物件陣列
  - Java 語言的陣列元素可以是物件，例如：Grade 成績類別，如下：

```
class Grade {
    int math;
    int english;
```

```
};
```

- Grade 類別的物件陣列宣告，如下所示：

```
Grade[] students = new Grade[3];
```

- 接著我們需要建立每一個陣列元素的 Grade 物件，如下：

```
for ( int i = 0; i < 3; i++ )
    students[i] = new Grade();
```

- 程式範例:

```
class Grade {
    int math;
    int english;
};
// 主類別
public class Ch2_1_1 {
    // 主程式
    public static void main(String[] args) {
        int sum = 0;   // 總分

        Grade[] students = new Grade[3];   // 建立物件陣列
        for ( int i = 0; i < 3; i++ )
            students[i] = new Grade();
        // 指定物件陣列的值
        students[0].math = 56; students[0].english = 78;
        students[1].math = 66; students[1].english = 91;
        students[2].math = 83; students[2].english = 67;

        sum = students[0].math + students[1].math +
              students[2].math;
        System.out.println("數學成績總分: " + sum);

        sum = students[0].english + students[1].english +
              students[2].english;
        System.out.println("英文成績總分: " + sum);
    }
}
```

- 二維陣列
  - 「二維陣列」（Two-dimensional Array）屬於一維陣列的擴充，如果將一維陣列視為一度空間，二維陣列就是一個二度空間的平面。
  - 在日常生活的二維陣列非常常見，只要屬於平面的表格，大都可以轉換成二維陣列來儲存資料。
  - 多維陣列-宣告

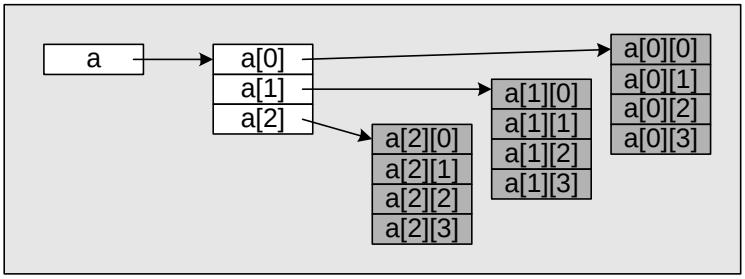
    在 Java 語言的 Array 物件可以包含其它 Array 物件，這就是 Java 語言的二維陣列或多維陣列，如下所示：

```
int[][] scores = { { 54, 68 },
                   { 67, 78 },
                   { 89, 93 } };
```

- JVM 中的二維陣列

  宣告並配置：

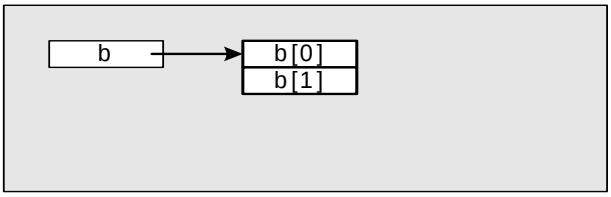
  int[][] arr = new int[3][4];



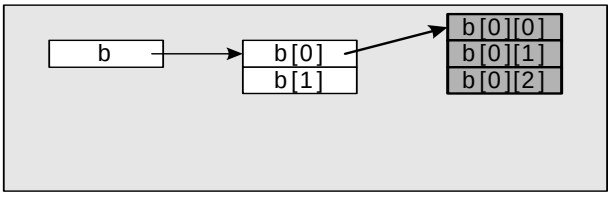
- 建立陣列實體，再指定給參照變數

```
int b[][];
```

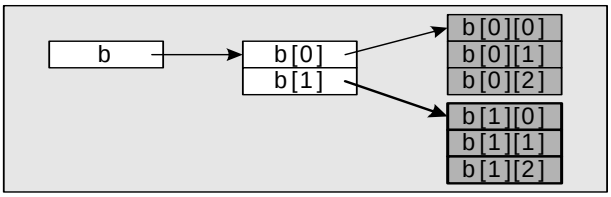

```
b = new int[2][];
```




```
b[0] = new int[3];
```




```
b[1] = new int[3];
```



- 非矩形二維陣列

int[][] a = {{1, 3, 5}, {2, 4}, {9, 8, 7, 6}};

	行0	行1	行2	行3
列0	1	3	5	
列1	2	4		
列2	9	8	7	6

物件導向程式設計講義

Chapter 3　物件導向程式設計及物件的建構

1. 物件

- 一般所講的「物件」，其實就是有形體的東西。
- 每個物件都有它特別的外形、尺寸和功能
- 物件擁有「屬性」和「功能」
- 由於觀點的不同，每個人對相同的一個物件之描述也可能不盡相同。

2. 物件的三個特性

- 狀態（State）：物件所有「屬性」（Attributes）目前的狀態值，屬性是用來儲存物件的狀態，例如：老師的年資、姓名、身高和體重等屬性。
- 行為（Behavior）：行為是物件可見部分提供的服務，可以作什麼事，例如：老師可以教書、出作業和回答問題等。
- 識別字（Identity）：識別字是用來識別不同的物件，每一個物件都擁有獨一無二的識別字，Java 語言是使用物件參考（Reference）作為物件的識別字，簡單的說，就是物件實際儲存的記憶體位址。

3. 類別

- 類別是由一群具有相同屬性與相同行為的物件，在依據某個特性分類以後，所形成的集合。
- Java 的類別是用來建立物件，它是一種使用者自行定義的資料型態。
- 同一個類別可以當作範本建立無數個物件實例，每一個物件都屬於類別的實例，或直接稱為物件。
- Java 物件使用變數儲存狀態稱為「屬性」（Property）或「成員變數」，各種行為的程序和函數，在 Java 稱為方法（Methods）。
- 範例：

```
class Teacher
{  //  成員變數
      int years;        // 年資
      String name;     // 姓名
      int height;       // 身高
      int weight;       // 體重
   //  成員方法
      public void teach()  {  ...  }            //教書
      public void giveHomework()  {  ...  }   //出作業
      public void answer()  {  ...  }           //回答問題
}
```

- 類別的修飾字

  public－公開類別，宣告成此種類別可以被任何類別所使用。

  無修飾字（default）類別，此種類別僅能被同一套件（package）內的類別使用。

  final－此種類別不可被繼承。

  abstract－此為抽象類別的修飾字。此種類別至少擁有一個抽象方法，不可用於直接建立物件。

4. 建立物件

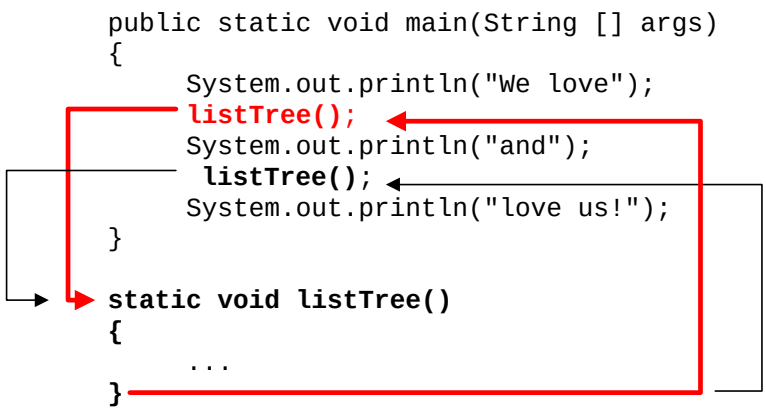
- 宣告物件　　*Teacher t;*
- 建立物件實體　　*t = new Teacher();*
- 宣告並建立物件實體　　*Teacher t = new Teacher();*

5. Java 的方法

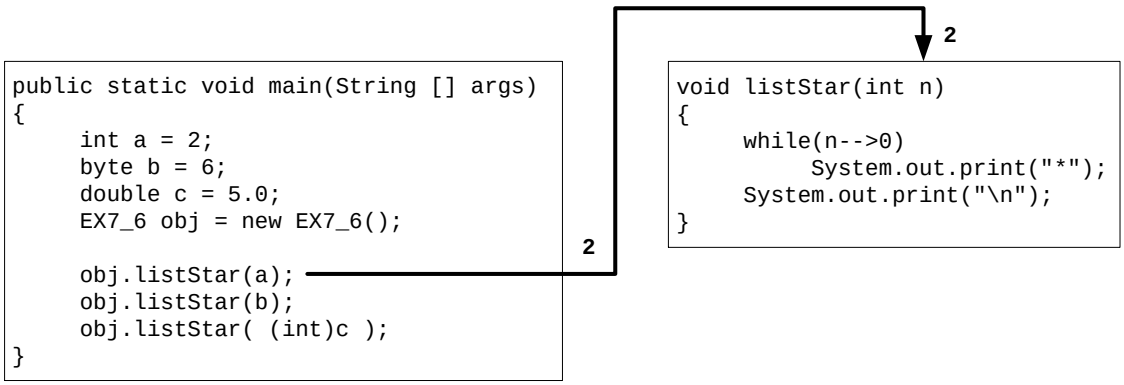
- Java 的方法可以分為兩種，如下所示：

  屬於類別的「類別方法」(Class Methods)

  物件的「實例方法」(Instance Methods)
- Java 方法的參數列是資訊傳遞的機制，可以從外面將資訊送入程序的黑盒子，參數列是方法的使用介面。
- 一個方法如果擁有參數列，在呼叫方法時，傳入不同的參數就可以產生不同的執行結果。
- 呼叫方法時的程式走向



- 參數只傳遞數值



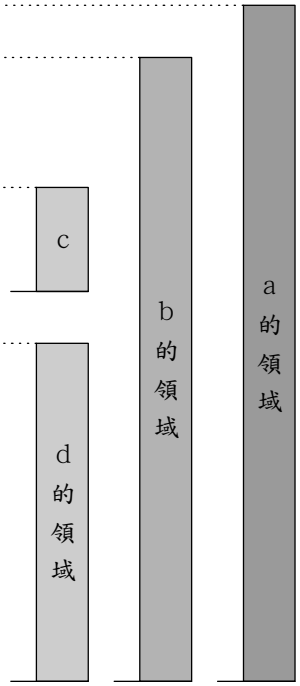
6. 方法多載

- 同一類別中，若定義數個相同名稱的方法，而各方法所需的參數不同時，稱為方法多載（Overloading）。
- 多載方法是以傳入的參數個數及參數型別做為呼叫的判斷依據。
- 多載方法的形式參數的個數及型別相同時，為非法定義。
- 多載的目的是因應不同的傳遞資料，讓方法更有彈性。

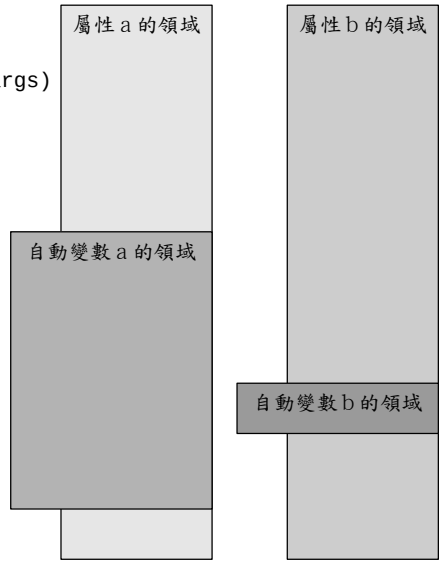
7. 變數範圍

- 區域變數範圍（Local Variable Scope）：在方法內宣告的變數，只能在宣告程式碼後的程式碼使用（不包括宣告前），在方法外的程式碼無法存取此變數。
- 方法參數範圍（Method Parameter Scope）：傳入方法的參數變數範圍是整個方法的程式區塊，在方法外的程式碼並無法存取。
- 成員變數範圍（Member Variable Scope）：不論是 static 的類別變數或是沒有宣告 static，整個類別的程式碼都可以存取此變數。

```
static void methodName(int a)
{
     int b = 2;

     System.out.println("a:"+a);
     System.out.println("b:"+b);

     for(int c = 0; c<10; c++)
     {
          if(c==3)
               System.out.println("c:"+c);
     }

     int d = 4;

     if(d==4)
     {
          System.out.println("d:"+d);
          int e = 5;
          System.out.println("e:"+e);
     }

     {
          int f = 6;
          System.out.println("f:"+f);
     }
}
```




```
class EX7_12
{
     int a =10;

     public static void main(String [] args)
     {
          EX7_12 obj = new EX7_12();
          obj.myMethod(1);
     }

     void myMethod(int a)
     {
          ...

          {
               {
                    int b=2;
                    ...
               }
               ...
          }
     }
     int b = 20;
}
```



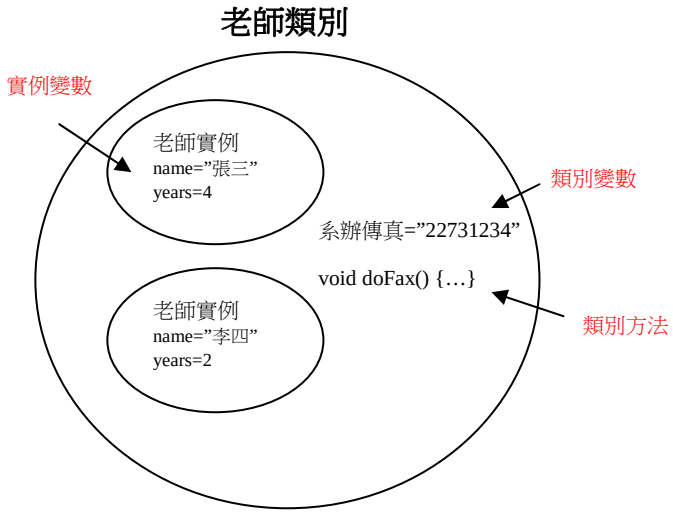
8. 類別成員

- 在此之前的類別成員宣告都沒有使用 static 宣告的變數和方法，因為在類別宣告 static 的類別變數和方法後，建立物件並不會替每一個物件實例建立獨立的類別變數和方法，只有不是宣告成 static 部分才會建立獨立的實例變數和方法。
- 類別變數和方法是屬於類別，所有物件都是使用同一份類別變數和呼叫同一個類別方法。

- 實例屬性及方法： 屬於實例的屬性 及方法。

  類別（靜態）屬性：屬於類別的屬性，為所有的實例共用，而不屬於任何物件。

  類別（靜態）方法：屬於類別的操作，為所有的實例共用，而不屬於任何物件。

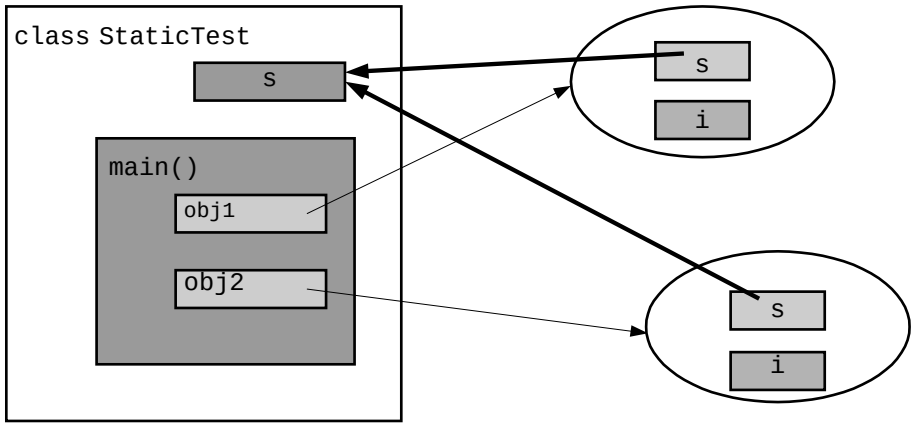


- 存取一個類別的 public static 成員

  在一個成員的前面加上類別名稱及點運算子 (.)

  例： Math.random()

- 類別變數和實例變數的區別




```
// 範例：StaticTest.java
class StaticTest
{
      int i = 0;
      static int s = 100;

      public static void main(String [] args)
      {
     StaticTest obj1 = new StaticTest();
     StaticTest obj2 = new StaticTest();
```

```
        System.out.println(++obj1.i);
        System.out.println(++obj1.s);

        System.out.println(obj2.i += 10);
        System.out.println(obj2.s += 10);

        System.out.println("obj1.s=\t" + obj1.s);
        System.out.println("obj2.s=\t" + obj2.s);
        System.out.println(" StaticTest.s=" + StaticTest.s);
        System.out.println("s=\t" + s);
          }
}
```

9. 建構子

- 如果希望如同基本資料型態在宣告時指定變數初值，在類別宣告需要新增建構子，建構子是物件的初始方法，在建立物件時會自動呼叫此方法。
- 建構子與類別同名，例如：類別 Teacher 的建構子方法是 Teacher()。
- 建構子沒有傳回值。

範例：

```
class Teacher
 {
    int years;        // 年資
    String name;     // 姓名
    public Teacher(int years, String name) // 建構子
    {     this.years = years;
          this.name = name;
    }
    public void teach()  {  ...  }              //教書
 }
```

- 建立物件時呼叫建構子

```
Teacher t = new Teacher(5, "張三" )；
```

- 建構子支援方法的「多載」（Overload），也就是說可以擁有多個同名建構子方法，只是擁有不同的參數型態和參數個數。

範例：

```
class Teacher
 {
    int years;        // 年資
    String name;     // 姓名
    public Teacher() // 建構子
    {     this(1, "張三");
    }
    public Teacher(String name) // 建構子
    {     this(1, name);
    }
     public Teacher(int years, String name) // 建構子
    {     this.years = years;
```

```
        This.name = name;
    }
    public void teach()  {  ...  }          //教書
}
```

- 建構子的修飾字可以為 public、protected、private 或無修飾字。
- 建構子修飾字不使用 static 和 abstract。
- 方法可以和建構子同名。有回傳型別者為方法；反之，無回傳型別者為建構子。

物件導向程式設計講義

Chapter 4　字串、Math 類別及包裝類別

1. 字串的產生

- 字串其實就是 String 物件, 所以要宣告一個字串, 就等於是宣告一個指到 String 物件的參照，然後再利用 new 產生 String 物件，也可以直接讓 String 型別的參照變數指向字串常數。

```
String str = new String("我愛Java!");
String str = "我愛Java!";
```

- String 物件為不可變更的（immutable）物件，其實體內的字元或字串長度都不可變更。
- String 物件的長度可以透過 **length()**方法取得。
- 利用 String 物件的 charAt()方法可以使用索引值取得字串裡的某個字元

範例：

```
class Test
{
   public static void main(String [] args)
   {
        String str = "我愛人人";

       int len = str.length();

       for(int i=len-1; i>=0; i--)
            System.out.print( str.charAt(i) );
       System.out.println();

    }

}
```

2. 字串的連接

- 使用加號「+」　　str1 = str2 + str3;
- 使用「+=」　　str1 += "ABC";
- 使用 concat()方法　　str1.concat(str2)；
- String 物件為不可變更的（immutable）物件，呼叫其物件方法都不會對原物件的內容造成影響。

範例：

```
class Test
{
  public static void main(String [] args)
  {
       String str1 = "東西";
       String str2 = str1 + "當舖當";

       System.out.println( str2.concat(str1) );
```

```
        System.out.println(str1);
        System.out.println(str2);
    }
}
```

3. String 類別的尋找字元和子字串的方法

int indexOf (String str)	尋找字串 str 的位置。
int indexOf (String str, int from)	從 from 開始尋找字串 str。
int lastIndexOf (String str)	尋找最後一個字串 str 的位置。
int lastIndexOf (String str, int from)	從 from 開始尋找最後一個 str。

範例：

```
public class Test
{  // 主程式
   public static void main(String[] args)
   {  // 字串物件宣告
      String str= new String("東西當舖當東西");

      System.out.println(str.indexOf("當"));
      System.out.println(str.indexOf("當", 3);
      System.out.println(str.indexOf("東西"));
      System.out.println(str.lastIndexOf("東西"));
   }
}
```

4. String 類別的擷取子字串方法

public String **substring**(int beginIndex)
public String **substring**(int beginIndex, int endIndex)

範例：

```
class Test
{
   public static void main(String [] args)
   {
        String str = "感覺 Happy 就是幸福！";

        System.out.println("str = " + str);
        System.out.println("str.substring(9) = " + str.substring(9));
        System.out.println("str.substring(2, 7) = " + str.substring(2, 7));
   }
}
```

- 整合擷取子字串方法及尋找子字串

範例：

```
public class Test
{
  public static void main(String[] args)
   {
        String str = "How are you";
```

```
            int firstspace = str.indexOf(' ');
            int secondspace = str.indexOf(' ',firstspace+1);
            System.out.println( str.substring(0,firstspace) );
            System.out.println( str.substring(firstspace+1,secondspace) );
            System.out.println( str.substring(secondspace+1) );
        }
    }
```

5. 字串的比較

- 以比較運算子「==」判斷兩個 String 物件時，是判斷兩者是否指向相同的參考。
- String 類別的 equals()方法，其功能是比較兩個 String 物件的內容而不是參照。

  字串一.equals(字串二)

- 如果要忽略英文字母的大小寫，可以使用 equalsIgnoreCase()方法。

範例：

```
class Test
{
   public static void main(String [] args)
   {
      String s1 = "java";
      String s2 = "JAVA";
      if (s1.equals(s2))
            System.out.println("兩者相等(分大小寫)");
      else
            System.out.println("兩者不等(分大小寫)");

      if (s1.equalsIgnoreCase(s2))
            System.out.println("兩者相等(不分大小寫)");
      else
            System.out.println("兩者不等(不分大小寫)");
    }
}
```

6. String 類別的其它常用方法

```
public static String valueOf (Object obj)
public String toLowerCase ()
public String toUpperCase ()
public String replace (char oldChar, char newChar)
```

範例：

```
class Test
{
   public static void main(String [] args)
   {
      String s = "jAVA PrograMMing";
      String first = s.substring(0,1).toUpperCase();
      String remainder = s.substring(1).toLowerCase();
      System.out.println(first+remainder);
   }
```

```
}
```

7. Math 數學類別

- Java API 的 Math 類別提供數學常數和各種數學函數的類別方法，在 Java 程式提供亂數、計算最大值、最小值、三角和指數等數學函數，因為是類別方法，在呼叫方法時需要指明類別 Math。

例如：計算 900.0 的平方根

Math.sqrt(900.0)

- Java 的亂數產生器 **Math.random()**

*(int) (Math.random() * 6)* // 產一個介於 0 – 5 的整數

- Math 類別的常數

靜態常數	說明
double E	尤拉常數，自然對數的底數。
double PI	圓周率。

- Math 類別的三角函數方法

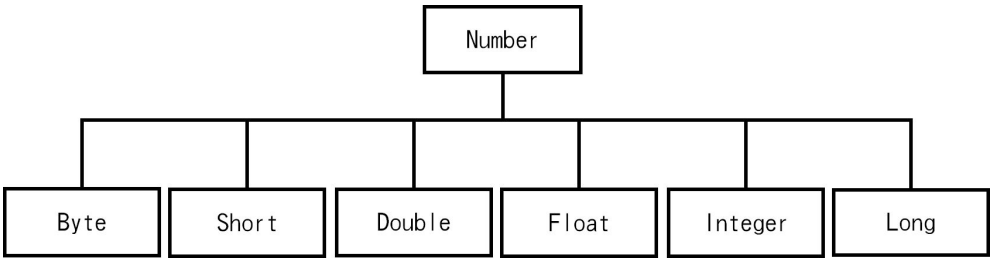
三角函數方法	說明
double acos(double a)	求反餘弦。
double asin(double a)	求反正弦。
double atan(double a)	求反正切。
double atan2(double y, double x)	以 x 和 y 為座標，求反正切。
double cos(double a)	以 a 為徑度求餘弦。
double sin(double a)	以 a 為徑度求正弦。
double tan(double a)	以 a 為徑度求正切。

- Math 類別的其它常用方法

方法	說明
double ceil(double a)	取得不小於 a 的 double 型別之整數。
double exp(double a)	求得 e 的 a 次方。
double floor(double a)	取得不大於 a 的 double 型別之整數。
double log(double a)	以 e 為底求對數值。
double max(double a, double b)	回傳 a 和 b 中較大者。
double min(double a, double b)	回傳 a 和 b 中較小者。
double pow(double a, double b)	求 a 的 b 次方。
double random()	取得介於 0.0 ~ 1.0（小於 1.0）之間的亂數值。
long round(double a)	四捨五入求整數值。
double sqrt(double a)	求 a 的平方根。

8. 包裝類別

- 可以包裝基本資料型別數值的類別，這些類別稱為包裝器（Wrappers）。
- 對應基本資料型別的包裝器類別為：Boolean、Byte、Short、Character、Integer、Long、Float 和 Double 八個類別。
- 包裝器物件是不可變更的（immutable），物件建立後，其包裝的數值就不可改變。
- Byte、Short、Integer、Long、Float 和 Double 六個類別都是和數值相關的類別，它們都繼承 **Number** 抽象類別。



- 型態包裝類別（Type-Wrapper）類別簡單的說是一種 Java 基本資料型態的物件版，雖然 Java 語言已經提供基本資料型態，但是我們需要重複類別的原因，如下所示：
  - 當程式需要使用物件時，可以使用型態包裝類別儲存資料。
  - 型態類別物件提供常數和方法可以取得資料範圍和轉換成其它資料型態。

9. 字串轉換成基本型別

類別	*方法*	*說明*
Boolean	boolean getBoolean(String s)	若字串 s 轉換成小寫後為 "true" 時，則傳回布林值 true，否則傳回 false。
Byte	byte parseByte(String s)	轉換成 Byte 型別。
Short	short parseShort(String s)	轉換成 Short 型別。
Integer	int parseInt(String s)	轉換成 Integer 型別。
Long	long parseLong(String s)	轉換成 Long 型別。
Float	float parseFloat(String s)	轉換成 Float 型別。
Double	double parseDouble(String s)	轉換成 Double 型別。

10. 自動包裝(Autoboxing)與解裝(Unboxing)

- 之前的寫法:

```
int i = 10;                          // 轉成包裝器物件
Integer j = new Integer(i);          // 取得基本型別的值
int k = j.intValue();
```

- 有 Autoboxing 之後的寫法:

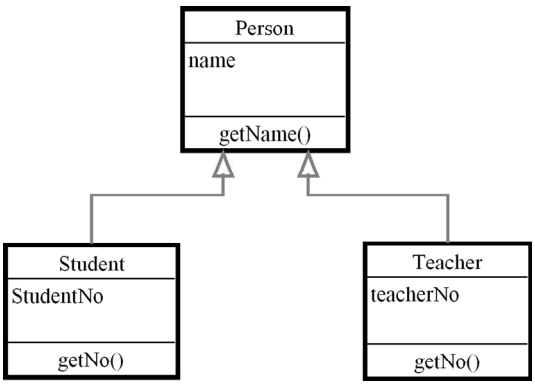
```
int i = 10;
Integer j = i;
int k = j;
```

- 便利之餘要注意的是，每個基本型別的值都只自動包裝到對應的包裝器物件，跟自動型別轉換沒有直接的關係。
- 自動包裝與解裝的另一個問題是，可能在方法多載時造成曖昧不明的情況。發生這種情況時，可暫時忘掉自動包裝與解裝，再推敲結果。

物件導向程式設計講義

Chapter 5　繼承、物件型別轉換、多型及抽象類別

1. 繼承關係（Inheritance）



- Student 和 Teacher 類別是繼承自 Person 類別，我們稱 Student 和 Teacher 類別為繼承類別的「子類別」（Subclass）或「延伸類別」（Derived Class），繼承的 Person 類別稱為「父類別」（Superclass）或「基礎類別」（Base Class）。

範例：

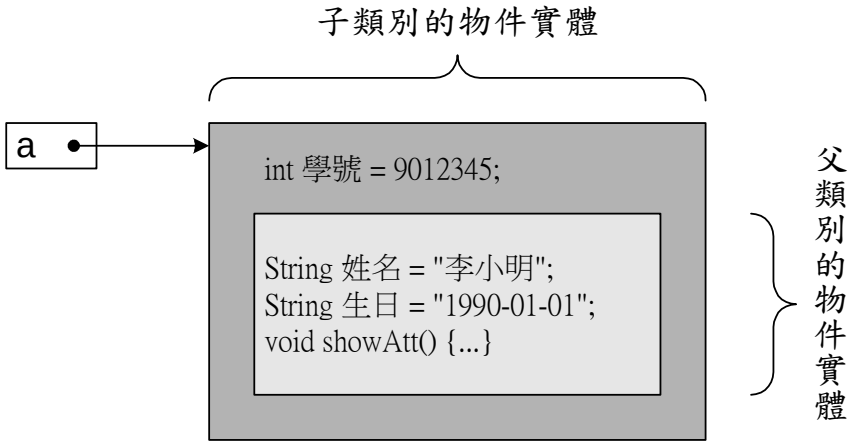
```
class Person {
        public String Name;
        public void eating()
        {
            System.out.println(Name+" eat slowly…");
        }
}
class Student extends Person{
 }
class Teacher extends Person {
}
public class Test  {
      public static void main(String args[])
      {     Student s = new Student();
            s.Name = "Jack";

            Teacher t = new Teacher();
            t.Name = "Mary";

            s.eating();
            t.eating();
      }
}
```

- 預設繼承類別
  - 所有類別的基礎類別 Object
  - 下列兩行的類別定義方式是相同的：

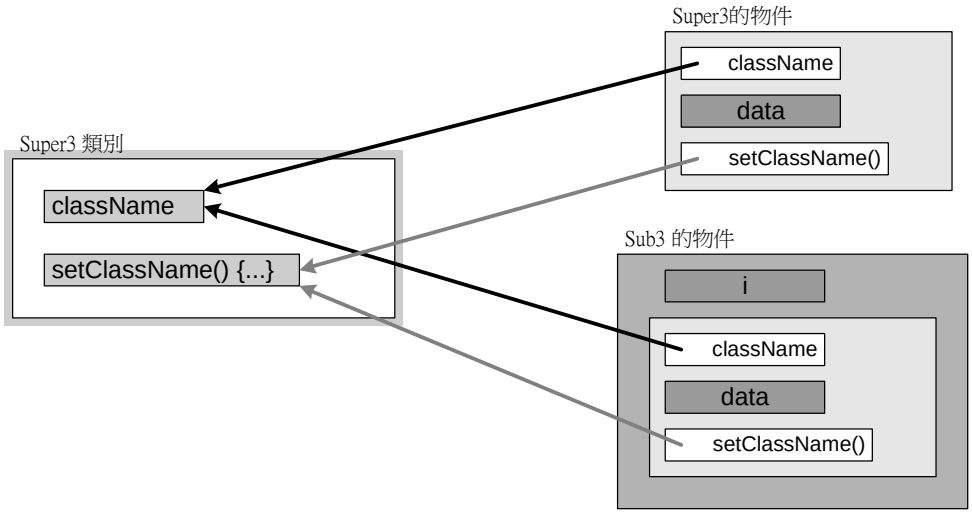
    class A {}

    class A extends Object {}

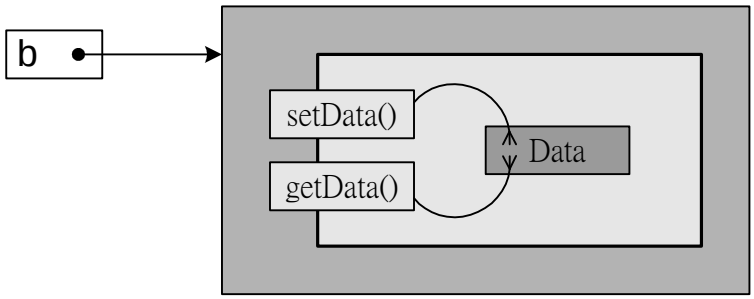
- 基本繼承
  - 非 private 修飾的物件成員都可以被繼承
  - 子類別的物件實體包含父類別的物件實體



- 類別成員與繼承
  - 類別屬性經過繼承之後，還是只有一份資料儲存區。
  - 這份資料儲存區是父類別、子類別、父類別的物件及子類別的物件所共有。
  - 繼承之後，類別成員都還是只有一個



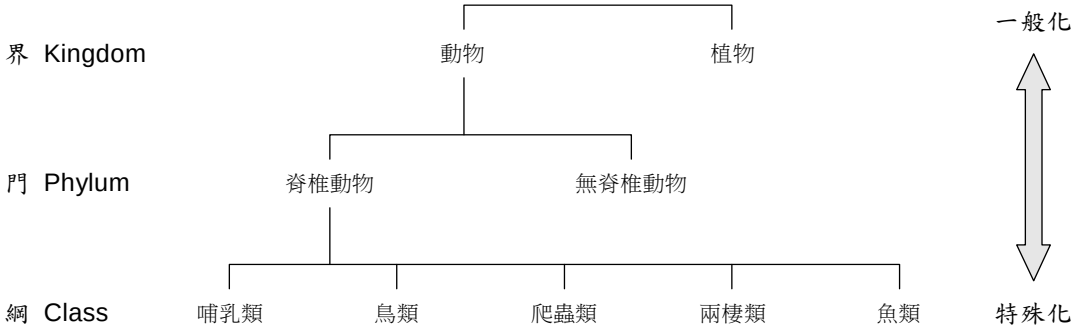
- private 成員不被繼承



  - 以 private 宣告的父類別之物件成員不被子類別所繼承。
  - 不被繼承的成員存在子類別物件中的父類別物件內，只是子類別物件無權使用。
  - 子類別物件可以透過父類別的 public 方法操縱父類別的 private 屬性。

2. 物件的型別轉換

- 子類別物件也屬於父類別物件



小紅是一條魚。

小紅是一隻脊椎動物。

小紅是一隻動物。

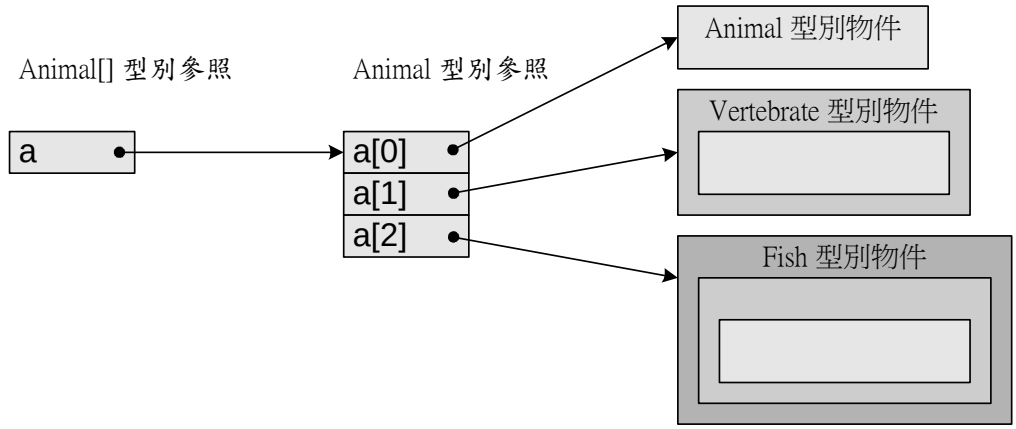
範例:

```
class Animal {}
class Vertebrate extends Animal {}
class Fish extends Vertebrate {}

class Test
{
    public static void main(String [] args)
    {
       Fish a = new Fish();
       Vertebrate b = new Fish();
       Animal c = new Fish();
       //Fish d = new Vertebrate(); //錯誤
       //Vertebrate e = new Animal();   //錯誤
```

```
        System.out.println("a=" + a);
        System.out.println("b=" + b);
        System.out.println("c=" + c);
    }
}
```

- 異質集合

異質集合的物件陣列




```
class Animal {}
class Vertebrate extends Animal {}
class Fish extends Vertebrate {}

class Test
{
    public static void main(String [] args)
    {
        Animal[] a = new Animal[3];
        a[0] = new Animal();
        a[1] = new Vertebrate();
        a[2] = new Fish();

        for(int i=0; i<a.length; i++)
            System.out.println("a[" + i + "]=" + a[i]);
    }
}
```

- instanceof 運算子

instanceof 運算式會得到一個布林值，若物件名稱所指向的物件實體屬於欲判斷的類別則回傳 true；反之，回傳 false。

物件名稱 instanceof *類別名稱*

```
class Animal {}
class Vertebrate extends Animal {}
class Fish extends Vertebrate {}

class Test
{
```

```
    public static void main(String [] args)
    {
        Fish 小紅 = new Fish();            //小紅是一條魚
        Animal 大黃 = new Animal();        //大黃是一隻動物

        System.out.println("小紅是脊椎動物: " +
                                (小紅 instanceof Vertebrate));
        System.out.println("小紅是動物: " +
                               (小紅 instanceof Animal));
        System.out.println("大黃是脊椎動物: " +
                               (大黃 instanceof Vertebrate));
        System.out.println("大黃是魚類: " +
                               (大黃 instanceof Fish));
    }
}
```

- 物件型別轉換，其實是針對指向物件的參照變數，而不是針對物件實體。物件實體只要被建立，其佔用的記憶體位置及大小都不會被變更。

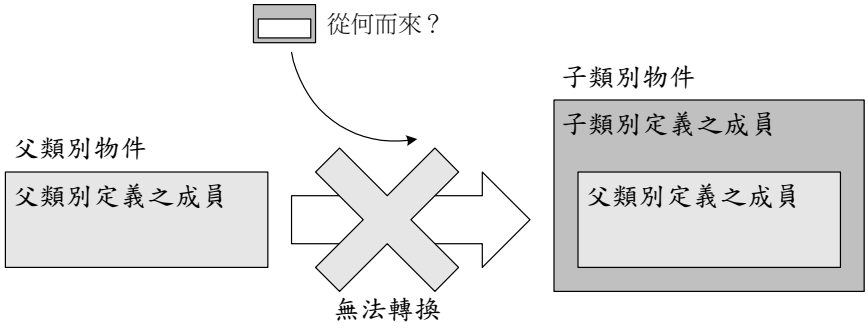
- 物件的強制型別轉換之語法

  (目的類別)物件參照變數

- 如果強制型別轉換是將參照變數的型別轉成子類別型別，則稱為向下轉型(Downcast)。

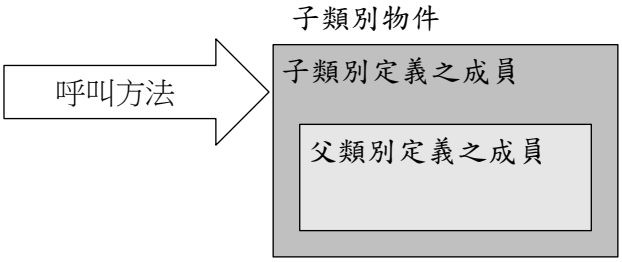
```
class Animal {}
class Vertebrate extends Animal {}
class Fish extends Vertebrate {}

class Test
{
   public static void main(String [] args)
   {
        Fish f = new Fish();
        Animal a = f;
        //Vertebrate v = a;          //錯誤
        Vertebrate v = (Vertebrate)a;
        Fish f2 = (Fish)a;

        System.out.println("a=" + a);
        System.out.println("v=" + v);
        System.out.println("f2=" + f2);
   }
}
```

- 父類別物件不具備子類別定義之成員，無法轉換成子類別物件。



3. 方法的覆蓋

- 子類別的重新定義物件方法稱為方法的覆蓋（Overriding）。
- 方法覆蓋時，下列各項都必須和父類別定義的方法相同：
  - 方法的回傳型別。
  - 方法名稱。
  - 形式參數列中的型別及順序。
- 覆蓋的原理：先碰到子類別定義之成員




```
class Dog
{
      void bark()//物件方法 bark()
      {
            System.out.print("汪汪");
      }
}

class WolfDog extends Dog
{
     void bark()  //覆蓋父類別方法
     {
            System.out.println("汪～汪～嗚～");
     }
}

class Doberman extends Dog
{
      void bark()//覆蓋父類別方法
```

```
        {
            super.bark();   //呼叫父類別定義的方法
            System.out.println("～汪汪汪～");
        }
}

class Test
{
    public static void main(String [] args)
    {
        //建立物件並呼叫其方法
        new WolfDog().bark();
        new Doberman().bark();
        new Dog().bark();
    }
}
```

4. 多型（Polymorphism）

- 多形是物件導向程式設計的重要觀念，可以讓應用程式更容易擴充，因為不需要針對不同資料型態分別建立類別，而是繼承一個基礎類別來建立同名方法，如此就可以處理不同資料型態，如果有新的資料型態，也只需新增繼承的子類別即可。

- 使用多型（介面相同）

```
class Dog
{
        void bark()//物件方法 bark()
        {
            System.out.print("汪汪");
        }
}

class WolfDog extends Dog
{
    void bark()  //覆蓋父類別方法
    {
            System.out.println("汪～汪～嗚～");
    }
}

class Doberman extends Dog
{
        void bark()//覆蓋父類別方法
```

```
    {
        super.bark();   //呼叫父類別定義的方法
        System.out.println("～汪汪汪～");
    }
}

class Test
{
    public static void main(String [] args)
    {
        Dog[] d = new Dog[3];
        d[0] = new WolfDog();
        d[1] = new Doberman();
        d[2] = new Dog();

        for(int i=0; i<d.length; i++)
            d[i].bark();          //呼叫方法
        }
}
```

5. 抽象類別與方法

- Java 的類別宣告成 abstract 表示是一個抽象類別，抽象類別不能用來建立物件，只能繼承抽象類別宣告子類別。
- 在抽象類別同時可以使用 abstract 宣告方法為抽象方法，表示方法只有原型宣告，實作的程式碼是在子類別建立，而且繼承的子類別一定要實作這些抽象方法。
- 抽象類別是子類別的原型宣告，抽象方法是宣告子類別使用介面的方法，如果類別擁有抽象方法，表示類別一定是抽象類別。例如：抽象類別 Shape，如下所示：

```
abstract class Shape
{
   public double x;
   public double y;
   abstract void area();
}
```

接著宣告 Circle 類別繼承 Shape 類別，如下所示：

```
class Circle extends Shape
{
   public double r;
   public Circle(double x, double y, double r) { … }
   public void area()
   { System.out.println("圓面積: " + 3.1416*r*r); }
}
```

上述子類別 Circle 定義圓形，除了圓心座標外，新增成員變數半徑 r，並且實作 area()方法計算圓面積。

物件導向程式設計講義

Chapter 6 基本輸出入與檔案處理

1. 檔案與資料夾處理

- File 類別可以處理作業系統的檔案和資料夾，讓 Java 程式取得指定資料夾的檔案和子資料夾清單、檔案資訊、建立資料夾和刪除檔案等操作。

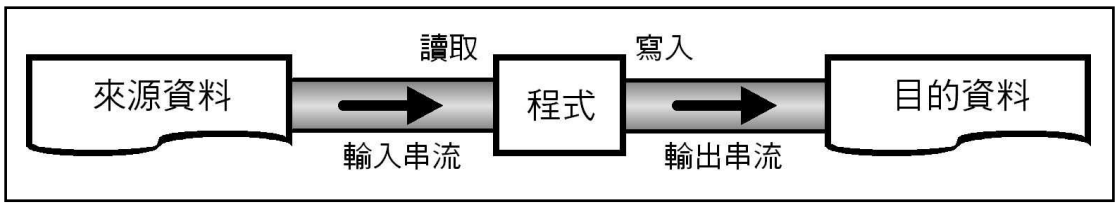
- 在 Java 程式只需建立 File 物件，就可以取得檔案或資料夾的相關資訊，如下所示：
  File f = new File(String);

範例:

```
import java.io.*;
public class Test
{  // 主程式
   public static void main(String[] args)
   {  // 宣告 File 物件
      File file = new File(args[0]);
      // 檔案是否存在
      if ( file.exists() )
      { if ( file.isFile() )
            System.out.println("["+file+"]是檔案");
        else if ( file.isDirectory() )
                 System.out.println("["+file+"]是目錄");
      }
      else
        System.out.println("[" + file + "]不存在!");
   }
}
```

2. Java 的輸入/輸出串流

- Java I/O 套件的全名是 Java Input/Output（輸入/輸出），即應用程式的資料輸入與輸出，在 Java 類別函式庫（Class Library）是使用「串流」（Stream）模型來處理資料的輸入與輸出。
- 串流（Stream）觀念最早使用在 Unix 作業系統，串流模型如同水管的水流，當程式開啟一個來源的輸入串流（例如：檔案、記憶體和緩衝區等），Java 程式可以從輸入串流依序讀取資料，如下圖所示



- Java 的 java.io 套件提供多種串流類別，基本上，Java 串流類別分成兩大類：
  「字元串流」（Character Stream）
  「位元組串流」（Byte Stream）

字元串流（Character Stream）

- 字元串流是 Java 1.1 版才支援的串流類別，這是一種適合「人類閱讀」（Human-readable）的串流，Reader/Writer 兩個類別分別讀取和寫入 16 位元的字元資料，屬於字元串流的父抽象類別。

位元組串流（Byte Stream）

- 位元組串流是一種「電腦格式」（Machine-formatted）串流，可以讀取和寫入 8 位元的位元組資料，也就是處理二進位資料的執行檔、圖檔和聲音等，其父抽象類別的輸入/輸出串流名稱為 InputStream/OutputStream 類別。

3. 標準輸出與輸入串流

- Java 語言的標準輸出和輸入是指 System 類別的 System.out 和 System.in 子類別，不過，System 類別並不是屬於 java.io 套件，而是屬於 java.lang 套件。
- 在本章之前的 Java 程式範例早已經大量使用 System.out 的 println()和 print()方法輸出資料，如下所示：

  System.out.println("….");
  System.out.print("….");

- 在 Java 程式將字串輸出到螢幕顯示就是開啟 System.out 標準輸出的 OutputStreamWriter 串流，我們可以使用緩衝器類別的 BufferedWriter 串流加速資料處理，如下所示：

  BufferedWriter output = new BufferedWriter(new
  OutputStreamWriter(System.out));

範例:

```
import java.io.*;
public class Test
{  // 主程式
   public static void main(String[] args)
                                          throws Exception
   {  // 建立 BufferedWriter 的輸出串流物件
      BufferedWriter output = new BufferedWriter(
                     new OutputStreamWriter(System.out));
      String str = "Java 2 程式設計範例教本 2e";
```

```
        output.write(str);    // 輸出字串
        output.close();       // 關閉串流
    }
}
```

- Java 程式從鍵盤輸入資料是開啟 System.in 標準輸出的 InputStreamReader 串流，同樣可以使用緩衝器類別 BufferedReader 串流加速資料處理，如下所示：

BufferedReader input = new BufferedReader(new
InputStreamReader(System.in));

範例:

```
import java.io.*;
public class Test
{  // 主程式
   public static void main(String[] args)
                                          throws Exception
   {  // 建立 BufferedReader 的輸入串流物件
      BufferedReader input = new BufferedReader(
                        new InputStreamReader(System.in));
      String str;
      System.out.print("請輸入資料: ");
      System.out.flush();        // 清除緩衝區
      str = input.readLine();   // 讀取一列
      input.close();             // 關閉串流
      System.out.println("輸入的資料是: " + str);
   }
}
```

4. 寫入與讀取文字檔案

- 當目標串流是一個檔案，在 Java 程式可以開啟檔案的 FileWriter 串流，然後使用緩衝器 BufferedWriter 串流來加速資料處理，如下所示：

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

範例:

```
import java.io.*;
public class Test
{  // 主程式
   public static void main(String[] args)
                                          throws Exception
   {  String file = "data.txt";
      String str1 = "Java 2 程式設計範例教本 2e\n";
      String str2 = "This is a pen.\n";
      // 建立 BufferedWriter 的輸出串流物件
      BufferedWriter output = new BufferedWriter(
```

```
                                    new FileWriter(file));

            output.write(str1);     // 寫入字串
            output.write(str2);     // 寫入字串
            output.close();         // 關閉串流
        }
    }
```

- Java 程式可以開啟檔案的 FileReader 串流來讀取檔案內容，同樣可以使用緩衝器 BufferedReader 串流來加速資料處理，如下所示：

  BufferedReader input = new BufferedReader(new FileReader(name));

範例:

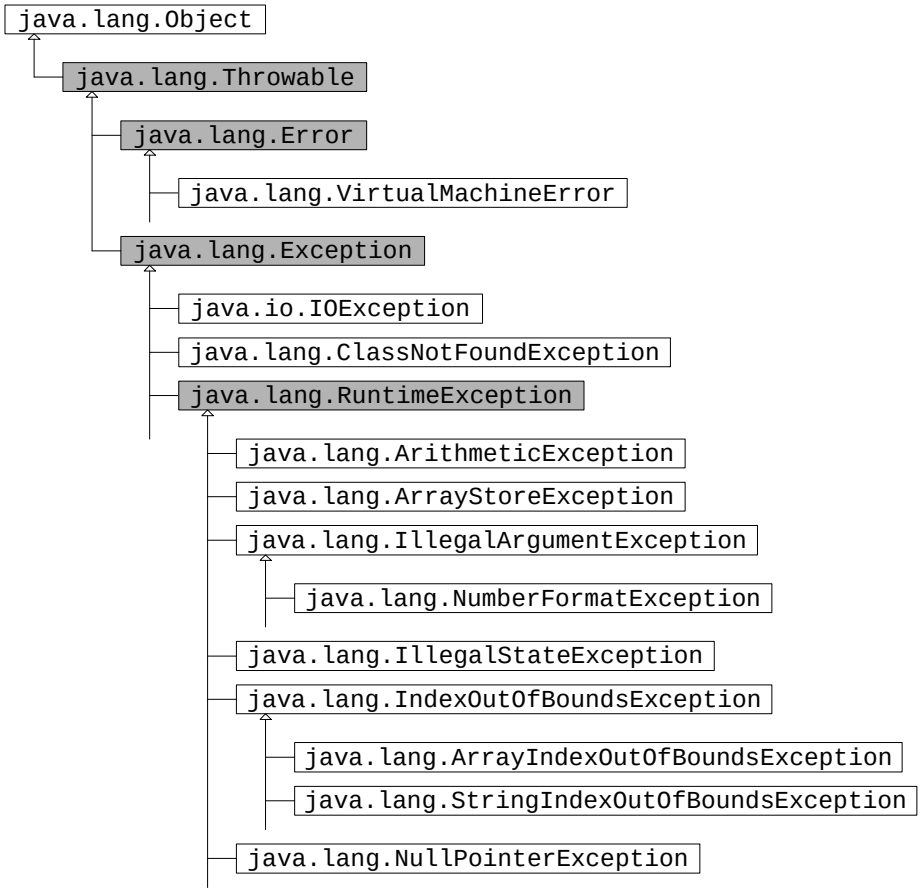
```
import java.io.*;
public class Test
{  // 主程式
   public static void main(String[] args)
                                              throws Exception
   {  String file = "data.txt";
      File name = new File(file);  // 建立 File 物件
      if ( name.exists() )
      {  // 建立 BufferedReader 的輸入串流物件
         BufferedReader input = new BufferedReader(
                                    new FileReader(name));
         String str;
         // 讀取資料
         while ( (str = input.readLine()) != null )
            System.out.println(str);
         input.close();        // 關閉串流
      }
      else
         System.out.println("檔案[" + name + "不存在!");
   }
}
```

物件導向程式設計講義

Chapter 7　例外處理與執行緒

1. Java 例外處理

- 錯誤或例外都是 java.lang.Throwable 的延伸類別
  - Error 及其延伸類別：其為嚴重的錯誤，通常捕捉到也無法處理，因此也很少去捕捉。
  - Exception 及其延伸類別（不包含 RuntimeException 及其延伸類別）：這類例外在一般情況很可能發生，大多屬於環境問題
  - RuntimeException 及其延伸類別：此類例外為程式的執行期錯誤，例如，某數除以 0、陣列索引值超出範圍等。



- 常見的 RuntimeException（Unchecked Exception）

執行期例外	說明
ArithmeticException	數學運算時的例外。例如：某數除以 0。
ArrayIndexOutOfBoundsException	陣列索引值超出範圍。
NegativeArraySizeException	陣列的大小為負數。
NullPointerException	物件參照為 null，並使用物件成員時所產生的例外。
NumberFormatException	數值格式不符所產生的例外。

- try-catch 敘述用以捕捉並處理例外

```
try{
  //例外測試區
}
catch(例外型別  例外名){
  //例外處理區
}
```

- 只要是 Throwable 或其延伸類別之物件，都可以使用 try-catch 敘述捕捉。
- try-catch 敘述可以使用多個 catch 區塊

```
try {
  //例外測試區
}
catch(例外型別1  例外名1){
  //例外處理區1
}
catch(例外型別2  例外名2){
  //例外處理區2
}
...

catch(例外型別N  例外名N){
  //例外處理區N
}
```

  - 頂多只會有一個 catch 區塊內的敘述被執行。
  - 特化的例外型別須放在較前的 catch 敘述內。

- finally 區塊
  - 不論有沒有例外發生，finally 區塊內的敘述皆會執行。

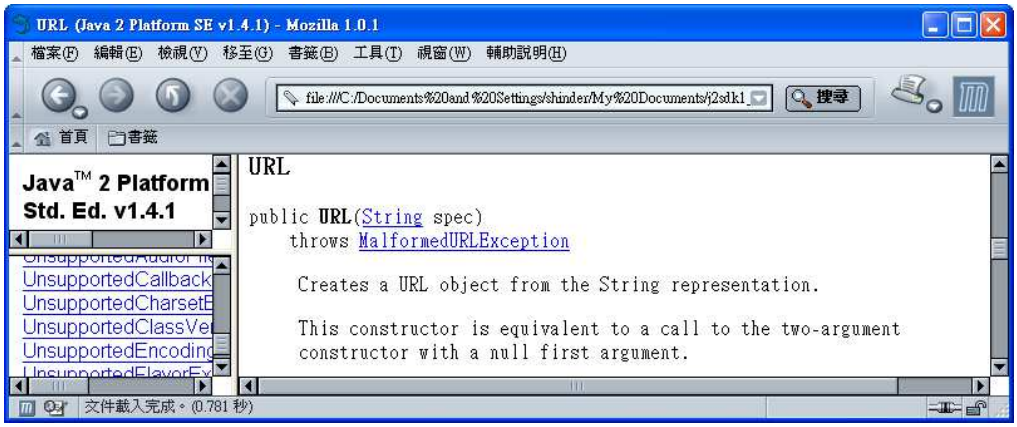
```
try
{
  //例外測試區
}
catch(例外型別  例外名)
{
  //例外處理區
}
finally
{
  //鐵定執行區
}
```

- finally 區塊不會執行或不會完全執行：
  - 有其它例外在 finally 區塊中發生。
  - 在 try 或 catch 區塊使用 System.exit()離開程式。
  - 未執行至 finally 區塊，執行緒即進入結束狀態（dead state）。

- 使用 throw 敘述在程式中丟出例外物件

```
throw new RuntimeException("除數為零");
throw new Exception("找不到檔案");
```

- throws 關鍵字
  - throws 關鍵字使用於方法或建構子定義的標頭，用來指出例外發生時，由方法（或建構子）丟出的例外型別。
  - throws 之後可以接多個例外型別名，表示方法執行過程中可能丟出屬於這些例外型別的物件。
  - 方法不可以使用 throw 丟出不包含在 throws 宣告的例外型別之物件
  - URL 的建構子定義使用到 throws 關鍵字，丟出 MalformedURLException 型別的例外。



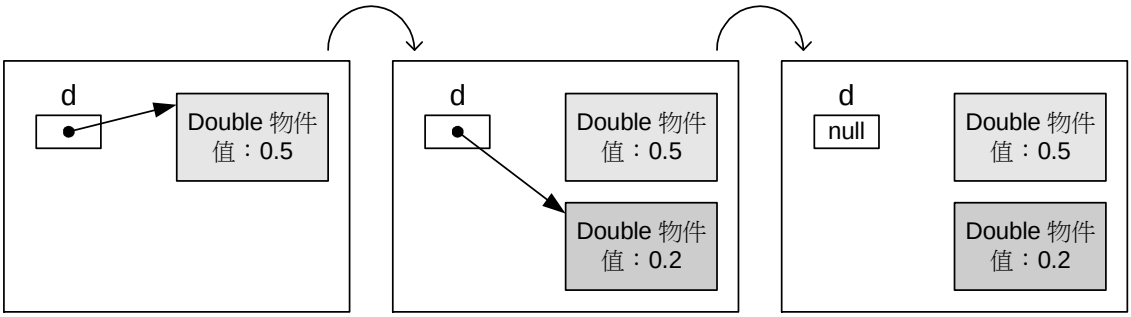
2. 資源回收機制

- 在 C/C++程式中，記憶體資源被配置後若不再使用，需由程式設計者釋放。
- 在 Java，記憶體資源的釋放問題由資源回收（Garbage Collection）機制管控。
- 當物件不再被參照變數（reference variable）參考（refer）時，它就會被認為是不再使用的 garbage。

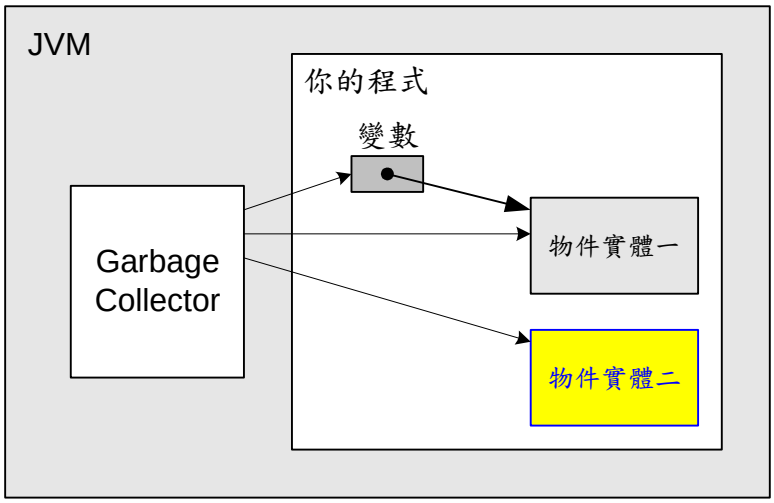
  Double d = new Double(0.5);

  d = new Double(0.2);

  d = null;



- Garbage Collector 在程式一開始執行時就會監視（trace）每一個變數和物件實體，當物件實體不被參考時，此物件實體即成為 Unreachable Object。



- finalize()方法
  - Garbage Collection 機制在終結物件並釋放記憶體之前，會先呼叫物件中的 finalize()方法。
  - finalize()方法定義於 Object 類別，不過為空方法，主要是讓延伸類別覆蓋之用。

    protected void finalize()throws Throwable

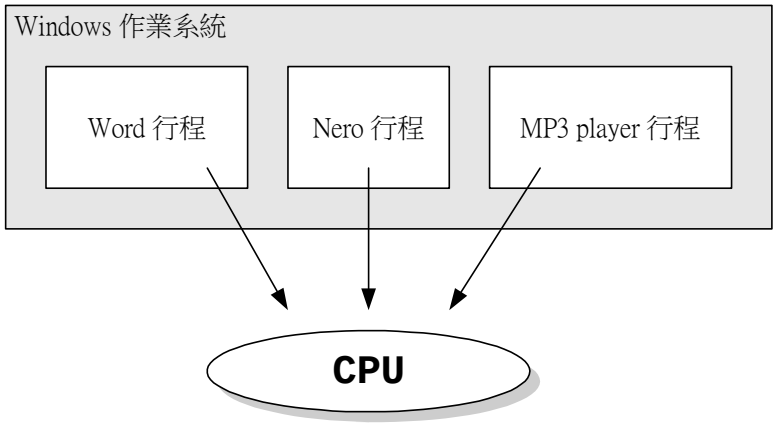
  - finalize()方法不提供多載的功能。而且 finalize()不接受傳入參數及回傳值，所以一律以 void 定義，並且無參數。
  - 建議 Garbage Collector 進行資源回收的方式：

System.gc();

Runtime.getRuntime().gc();

3. 行程與執行緒

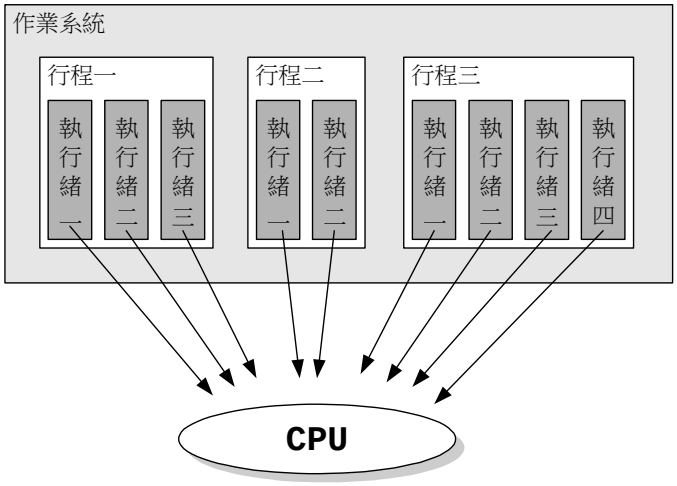
- 多個行程使用單一 CPU



  - 不同的行程所佔有的記憶體資源不同，各自獨立互不干擾。

- 大部份的個人電腦只有一個 CPU，所有行程都必須透過系統取得 CPU 的使用權。
- 每個行程輪流使用 CPU 的情形就像是每個行程都同時執行一樣。

- 多行程多執行緒使用單一 CPU



- 執行緒是行程 的 程 ， 程 行程 的 ，同時 CPU 的使用權。
- main() 就是 Java 用程 的 執行緒 執行緒，Main Thread 。
- 多執行緒程 必須 執行緒 執行緒的 行。
- Java 的執行緒都必須是 java.lang.Thread 的 。

4. Thread

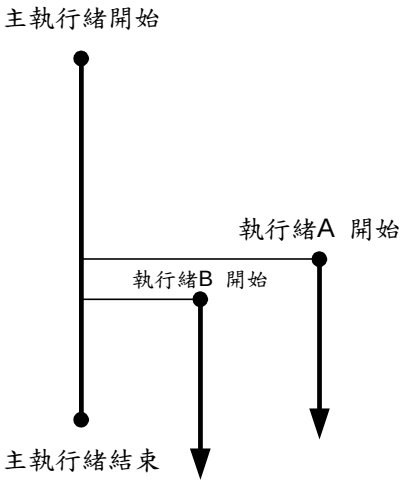
- Thread 的

	執行緒，同時 執行緒的 。
	取得執行緒的 。
	執行緒 的時 的 執行緒 ， 。
	執行緒 時所執行的 。
	執行緒 的 。

- 一個執行緒時，必須 的 start() ， 是 的 run() 。
- Thread 的 run() ， 執行緒，只是一 的 。
- 執行緒 ，執行緒 ， 是 。

- 呼叫 start()方法後執行緒才開始執行

```
public static void main(String [] args)
{
     EX12_1 ta = new EX12_1("執行緒A");
     EX12_1 tb = new EX12_1("執行緒B～");

     ta.start();
     tb.start();

     System.out.println("主執行緒要結束了～");
}
```



5. Runnable 介面

- 執行緒欲繼承 Thread 以外的類別時，可以利用 Runnable 介面建立。
- Runnable 介面只宣告一個 run()方法，所以實作介面時只要實作 run()方法即可。
- 實作 Runnable 介面的類別，還是必須依賴 Thread 類別的建構子才能建立一個執行緒物件。
- 使用 Runnable 型別物件建立 Thread 物件的建構子

使用 Runnable 物件的 Thread 建構子	說明
Thread(Runnable target)	以實作 Runnable 介面的類別物件 target 建立執行緒物件。
Thread(Runnable target, String name)	以實作 Runnable 介面的類別物件 target 建立執行緒物件，執行緒的名稱設定為 name。

- 使用 Runnable 型別物件建立 Thread 物件，兩個物件是獨立的，不過可以看成是「Thread 物件在啟動之後，呼叫 Runnable 型別物件的 run()方法」。

```
final RunnableTest rt = new RunnableTest();
Thread mt = new Thread(){
   public void run()
   {
        rt.run();
   }
};
```