

# 初探 Java

TYIC桃高資訊社

# 第一個 Java 程式

寫好了第一個 Java 程式  
但是自己在寫什麼  
自己也不知道

```
01 // class 名稱必須跟檔案名稱一樣
02 public class Main {
03
04     // Java 程式需要一個主方法(main 方法)，程式從這裡開始執行
05     public static void main(String[] args) {
06
07         // 在 Java 中，使用 System.out.println() 來輸出資料
08         System.out.println("Hello, World!");
09
10     }
11 }
```



java

# 第一個 Java 程式

第 1、4、7 行：

"//" 表示是單行註解(Comment)

程式執行會忽略 "//" 和該行後面的所有文字


```
01  // class 名稱必須跟檔案名稱一樣  
02  public class Main {  
03  
04      // Java 程式需要一個主方法(main 方法)，程式從這裡開始執行  
05      public static void main(String[] args) {  
06  
07          // 在 Java 中，使用 System.out.println() 來輸出資料  
08          System.out.println("Hello, World!");  
09  
10      }  
11  }
```



java

# 第一個 Java 程式

第 2 行：`public`、`class` 都是保留字(Reserved word)  
有著特定功能，之後的課程會說到  
"`Main`" 是類別(`class`)名稱



```
01 // class 名稱必須跟檔案名稱一樣
02 public class Main {
03
04     // Java 程式需要一個主方法(main 方法)，程式從這裡開始執行
05     public static void main(String[] args) {
06
07         // 在 Java 中，使用 System.out.println() 來輸出資料
08         System.out.println("Hello, World!");
09
10     }
11 }
```




java

# 第一個 Java 程式

第 5 行：`public`、`static`、`void` 也都是保留字

"`main`" 是方法名稱，"`args`" 是一個參數(`parameter`)

"`String[]`" 是 `args` 參數的型別(`type`)，之後的課程會說到



```
01 // class 名稱必須跟檔案名稱一樣
02 public class Main {
03
04     // Java 程式需要一個主方法(main 方法)，程式從這裡開始執行
05     public static void main(String[] args) {
06
07         // 在 Java 中，使用 System.out.println() 來輸出資料
08         System.out.println("Hello, World!");
09
10     }
11 }
```



java

# 第一個 Java 程式

第 8 行："System.out.println()" 是一個方法(method) 用來輸出小括號裡面放的是要輸出的東西(引數argument)，這裡放的是「"Hello, World!"」，所以會輸出 "Hello, World!" 而因為這是個表達陳述式(expression statement)，所以結尾須加上分號

```
01 // class 名稱必須跟檔案名稱一樣
02 public class Main {
03
04     // Java 程式需要一個主方法(main 方法)，程式從這裡開始執行
05     public static void main(String[] args) {
06
07         // 在 Java 中，使用 System.out.println() 來輸出資料
08         System.out.println("Hello, World!");
09
10     }
11 }
```



java

# 註解

我們在第一個程式中說過，"**//**" 表示是單行註解

程式會忽略 "**//**" 和該行後面的所有文字

還有另一種註解是多行註解

程式會忽略夾在 "**/\***" 和下一個 "**\*/**" 中間的所有文字

```
System.out.println("會輸出(沒有被單行註解)");  
// System.out.println("不會輸出(被單行註解)");  
System.out.println("會輸出(多行註解前)");  
/*  
System.out.println("不會輸出(被多行註解)");  
System.out.println("不會輸出(被多行註解)");  
System.out.println("不會輸出(被多行註解)");  
*/  
System.out.println("會輸出(多行註解後)");
```

java

```
會輸出(沒有被單行註解)  
會輸出(多行註解前)  
會輸出(多行註解後)
```

output

# 基本輸出

我們在第一個程式中說過

"`System.out.println()`" 是一個用來輸出東西的方法且會換行

如果不想換行可以使用 "`System.out.print()`" 方法

能輸出的也不只文字，如：'a'、2147483647、3.14159、true

```
01 public class Main {
02     public static void main(String[] args) {
03         System.out.println("a");
04         System.out.println('a');
05         System.out.println("2147483647");
06         System.out.println(2147483647);
07         System.out.println("3.14159");
08         System.out.println(3.14159);
09         System.out.println("true");
10         System.out.println(true);
11     }
12 }
```



java

```
[ a
[ a
[ 2147483647
[ 2147483647
[ 3.14159
[ 3.14159
[ true
[ true
```


output

觀察每兩行有什麼差別？



# 基本輸出

```
01 public class Main {
02     public static void main(String[] args) {
03         System.out.println("a");
04         System.out.println(a);
05         System.out.println(2147483647);
06         System.out.println(2147483647);
07         System.out.println(3.14159);
08         System.out.println(3.14159);
09         System.out.println(true);
10         System.out.println(true);
11     }
12 }
```

  
java

```
[ a
[ a
[ 2147483647
[ 2147483647
[ 3.14159
[ 3.14159
[ true
[ true
output
```

顯而易見的，程式碼奇數行有一對雙引號，而偶數行沒有  
這是因為奇數行和偶數行括號裡的東西的資料型態不一樣的關係  
使用一對雙引號 `"` 夾起來的才是字串(**String**)，其餘則不是  
這與資料型別(**Data type**)有關

# 基本資料型別(primitive data types)

Java 中總共有 8 種基本資料型態：

byte 位元組	short 短整數	char 字元	int 整數	long 長整數	float 單精度 浮點數	double 雙精度 浮點數	boolean 布林
直接表示 值的範圍為 -128 到 127 的整數	直接表示 值的範圍為 -32768 到 32767 的整數	放一個字在 一對單引號 裡表示 也可以用 0~65535 的整數表示	直接表示 值的範圍為 $-2^{31}$ 到 $2^{31}-1$ 的整數 $2^{31}-1$ =2147483647	整數後方 加L表示 值的範圍為 $-2^{63}$ 到 $2^{63}-1$ 的整數	小數後方 加f表示 值的範圍 約為 3.4E-38到 3.4E+38	含小數點 直接表示 值的範圍約為 1.7E-308 到 1.7E+308	直接表示 值只有true 和false 分別代表 「真」與 「假」
-1 24	-2222 1024	'a' 99	-2147 83648	999999L -77777L	6.073f -2.88f	-228.0 3.5555	true false

像這樣直接寫下來的叫做字面常數(literal constant)，是值(value)的一種

# Char

`char` 在電腦內部實際上是儲存一個 0 ~ 65535 的整數

所以 `char` 也是數字的一種

這 0 ~ 65535 的整數當中每個數字各自對應了一個字元

而這個對應是根據 **Unicode** 的基本多文種平面(**Basic Multilingual Plane**，簡稱**BMP**、**0號平面**、**Plane 0**)來決定

當中除了前**128**個字元完全兼容 **ASCII(American Standard Code for Information Interchange**，美國標準資訊交換碼)

還有新增中日韓統一表意文字，也就是常見的漢字

以及拉丁字母、特殊字元、中日韓符號和標點、康熙部首等

# ASCII

ASCII 是相當重要的編碼

其中包含了英文字母、數字符號、特殊符號、控制字元

共 128 個字元（編號 0 - 127）

當中較為重要的是：

32：空格(space)

48：0

65：A

97：a

數字 0-9、英文 a-z、A-Z 皆可直接按照順序推下去

# ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

控制字元  
無法顯示

# 數字

8種基本資料型態中，整數表示的有byte、short、char、int、long

這四種不只可以直接以十進位(Decimal)表示，也可用其他進位表示

二進位(Binary)：在二進位數字前加上 "0b"，如 "0b101"

八進位(Octal)：在八進位數字前加上 "0"，如 "07777L"

十六進位(Hexadecimal)：在十六進位數字前加上"0x"，如"0xF4"

二進位：每 2 進位，所以組成只有 0 和 1

八進位：每 8 進位，由 0 - 7 組成

十六進位：每 16 進位，由 0-9和A-F組成，A-F依序代表10-15

# 數字

這六種不只可以直接表示，還可以在數字之間(含十六進位下A-F)

加上下劃線("\_"，**underscore**)讓數字更容易閱讀

如："0b1\_0\_1"、"0\_777"、"1912\_01\_01"、"0xF\_4L"

8種基本資料型態中，小數表示的有 **float** 和 **double**

這兩種不只可以直接表示，還可以使用科學記號來表示

如："3.14E59"、"48763E4"

直接以數字表示且不含小數點時，編譯器始終會視為 **int**

# 變數(Variable)宣告

在 Java 中可以宣告(**declare**)變數，宣告的方式有兩種：

資料型別 變數名稱; // 第一種，未初始化

資料型別 變數名稱 = 值; // 第二種，已初始化

java

第一種是只宣告變數，沒有初始化(**initialization**)，使用前必須初始化

第二種是宣告變數，並初始化變數，且值的資料型別必須和變數相同

兩種都是宣告陳述式(**declaration statement**)，須單獨一行，且結尾須有分號

若是第二種，資料型別可以填入 "**var**" 讓編譯器自動推斷

已經宣告過的變數不可以再宣告。舉例：

```
byte a;  
short b = 0;  
int c = 2147_4836_47;  
long d = 29999999999L;
```

java

```
float e = 1.414f;  
double f = 6.8;  
char g = 'z';  
boolean h = true;
```

java



# 變數賦值運算

在 **Java** 中，賦值(指定，**assign**)給變數的方式如下：

```
變數名稱 = 值
```

java

若變數還沒有初始化，則這行就是初始化變數

若變數已初始化，則這行就是重新賦值給變數，

且值的資料型別必須和變數相同

賦值可以是表達式(**expression**)也可以是表達陳述式。舉例：

```
a = 2;  
b = 4;  
c = -2147_4836_48;  
d = 999999999999999L;
```

java

```
e = 0.99999f;  
f = 0.999999999999999;  
g = ' '; // 空白也是一個字元  
h = false;
```

java


# 變數使用

變數代表一個值

所以任何可以填值的地方都可以填變數

舉例：

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(2);  
        int a = 2;  
        System.out.println(a);  
        a = 0;  
        System.out.println(a);  
        System.out.println(a = 3); // 賦值作為表達式  
        System.out.println(a);  
    }  
}
```

  
java

```
2  
2  
0  
3  
3  
output
```

# 變數命名規則

在 Java 中，變數命名「**一定要**」遵守以下規則：

1. 只能由 **a-z**、**A-Z**、**0-9**、**\$**、**\_** 組成
2. 開頭不能是數字
3. 不能是保留字

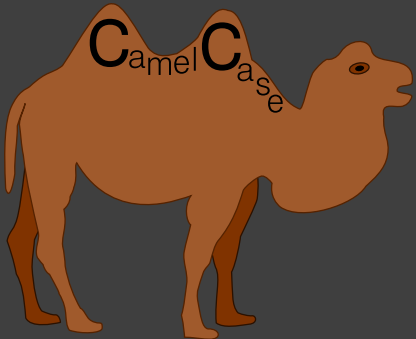
在 Java 中，變數命名「**盡可能**」遵守以下規則：

1. 名稱有意義，避免 **a**、**b**、**c** 這種名稱，除非是臨時變數
2. 使用小駝峰式命名法(lowerCamelCase)

如：**apple**、**applePen**、**penPineappleApplePen**

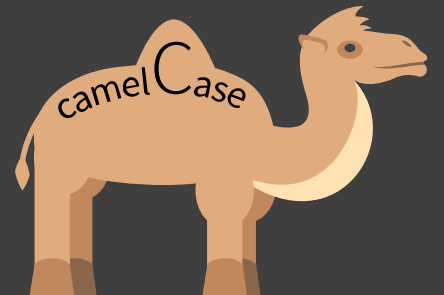


蛇行命名法：  
每個字母皆小寫  
且每個單字中間  
用下劃線連接



大駝峰式命名法  
(Pascal命名法)：  
每個單字首字母大寫  
其餘小寫  
且每個單字中間直接連接

小駝峰式命名法：  
第二個單字起每個單字  
首字母大寫其餘小寫  
且每個單字中間直接連接



# 常數(Constant)

若在宣告變數時加上 **final**，則在初始化後不可以被重新賦值：

```
final 資料型別 變數名稱; // 第一種，未初始化
```

```
final 資料型別 變數名稱 = 值; // 第二種，已初始化
```

java

其餘用法與變數完全一樣

```
public class Main {  
    public static void main(String[] args) {  
        final int a;  
        a = 10;  
        System.out.println(a);  
        a = 100; // Compile error: variable a might already have been assigned  
        System.out.println(a);  
    }  
}
```

java

# 常數命名規則

在 **Java** 中，常數命名規則基本上與變數命名規則一樣  
但建議使用

蛇行命名法(`snake_case`、`lower_case_with_underscores`)  
的變種

**SCREAMING\_SNAKE\_CASE(UPPER\_CASE\_WITH\_UNDERSCORES)**：

每個字母都大寫

且每個單字之間用下劃線連接

如：`PEN`、`APPLE_PEN`、`PEN_PINEAPPLE_APPLE_PEN`、`PI`

# 命名規則

如果沒有遵守命名規則...

```
final double abcde = 3.14_159;  
final int nINeTYniNe = 99;  
System.out.println(abcde);  
System.out.println(nINeTYniNe);
```

java

遵守命名規則後：

```
final double PI = 3.14_159;  
final int ninetyNine = 99;  
System.out.println(PI);  
System.out.println(ninetyNine);
```

java

# 表達式與表達陳述式

表達式(**expression**)：不單獨一行且結尾不須加上分號

表達陳述式(**expression statement**)：單獨一行且結尾須加上分號

有些只能當表達式，而有些只能當陳述式，而有些兩個都可以

如下方程式的第 8 行，"**System.out.println()**" 兩個都可以  
但這裡作為表達陳述式，所以單獨成一行且結尾有分號

```
01 // class 名稱必須跟檔案名稱一樣
02 public class Main {
03
04     // Java 程式需要一個主方法(main 方法)，程式從這裡開始執行
05     public static void main(String[] args) {
06
07         // 在 Java 中，使用 System.out.println() 來輸出資料
08         System.out.println("Hello, World!");
09
10     }
11 }
```



java

# 運算(Operation)

只有基本型別可以進行運算

每個運算都由運算元(operand)及運算子(operator)組成

且每個運算都會返回一個值(結果, result)

以加法運算為例：

```
6 + 8
```

java

其中 6 和 8 為運算元，為參與運算的值

"+" 為運算子，表示運算的類型

運算元和運算子中間的空格可省略，但不省略較易閱讀

運算元的數量及型別，視運算的類型而定

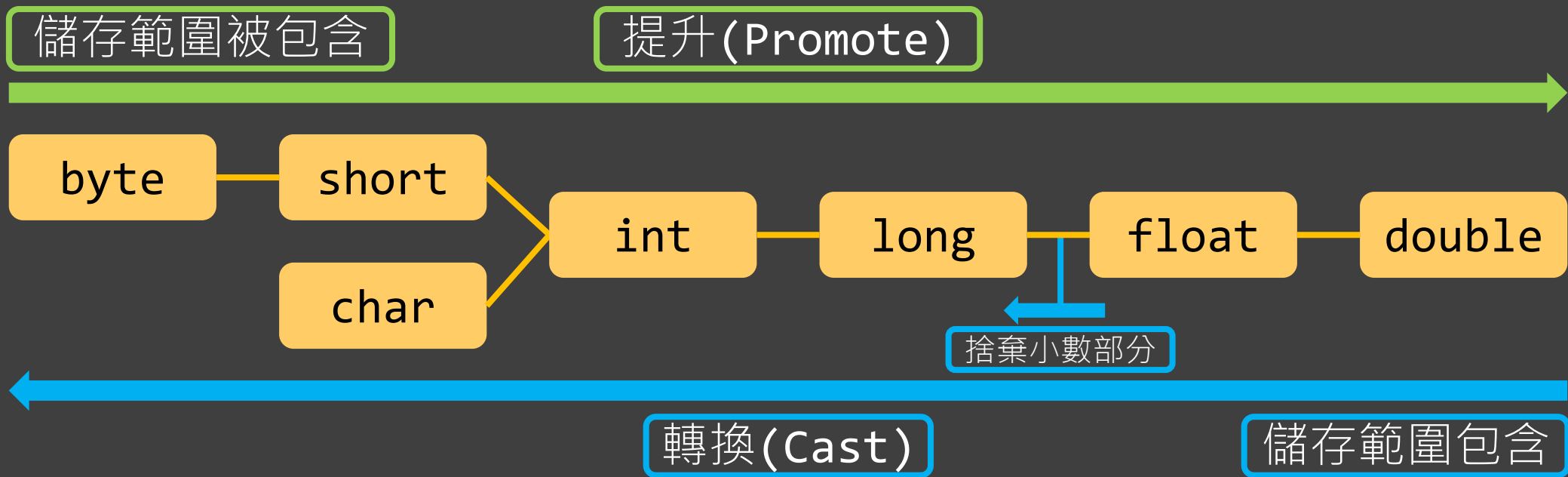
返回結果的型別，視運算的類型和運算元的型別而定

除了遞增遞減運算和指定運算可為表達陳述式，運算只能是表達式



# 數學運算

顯然的，數學運算只有數字才能用(含 char)  
而進行數學運算時，型別比較小的運算子會提升成型別較大的  
而且 byte、short、char 會提升成 int



# 一元數學運算

運算名稱	正數運算	負數運算
格式	+運算元	-運算元
功能	把數字加上正號 = <b>Do Nothing</b>	把數字變 為相反數
結果型別	與運算元型別相同	
範例	<b>+1</b> <b>+(-2)</b>	<b>-1</b> <b>-(-2)</b>

# 二元數學運算

運算名稱	加法運算	減法運算	乘法運算
格式	運算元1 + 運算元2	運算元1 - 運算元2	運算元1 * 運算元2
功能	運算元1 + 運算元2	運算元1 - 運算元2	運算元1 × 運算元2
結果型別	與運算元型別相同		
範例	$\begin{array}{r} 1 + 2 \\ 5 + -9 \end{array}$	$\begin{array}{r} 1 - 2 \\ -5 - 9 \end{array}$	$\begin{array}{r} 1 * 2 \\ -5 * -9 \end{array}$

# 二元數學運算

運算名稱	除法運算	取餘運算
格式	運算元1 / 運算元2	運算元1 % 運算元2
功能	運算元1 ÷ 運算元2	返回 (運算元1/運算元2)*運算元2-運算元1 也就是商趨向0，滿足 運算元1=運算元2×商+結果
結果型別	與運算元型別相同 故int/int得int	與運算元型別相同
範例	4 / 2 17 / -9	4 % 2 17 % -9

# 補充：取餘與取模

在 C/C++/Java 中，"%" 運算子是「取餘」運算

而在 Python 中，"%" 運算子是「取模」運算

取餘運算求商時，商會趨向於 0

而取模運算求商時，商會趨向於負無窮

這同時也影響了運算結果的正負性

若是取餘運算，則運算結果的正負與被除數相同

若是取模運算，則運算結果的正負與除數相同

# 複合指定運算

變數的賦值其實是賦值(指定)運算，而"="則是賦值(指定)運算子  
指定運算子和二元數學運算子可以合在一起，變成複合指定運算子

運算名稱	加法賦值	減法賦值	乘法賦值	除法賦值	取餘賦值
格式	變數 += 值	變數 -= 值	變數 *= 值	變數 /= 值	變數 %= 值
功能	變數=變數+值	變數=變數-值	變數=變數*值	變數=變數/值	變數=變數%值
結果型別	與運算元型別相同				
範例	a += 2 b += -9	c -= 2 d -= - 9	e *= 2 f *= -9	g /= 2 h /= - 9	g %= 2 h %= - 9

# 遞增、遞減運算

運算名稱	遞增運算	遞減運算
格式	變數++ 或 ++變數	變數-- 或 --變數
功能	<p>變數 = 變數 + 1</p> <p>++在後：先回傳變數再加</p> <p>++在前：先加再回傳變數</p>	<p>變數 = 變數 - 1</p> <p>--在後：先回傳變數再加</p> <p>--在前：先加再回傳變數</p>
結果型別	與運算元型別相同	
範例	a++ ++b	c-- --c


```
int a = 10;  
System.out.println(a++); // 11  
System.out.println(a); // 12  
System.out.println(++a); // 13
```

java

# 溢位(Overflow)

如果數值超過了該型別的範圍，那麼數值就會發生溢位變成從範圍的另一端出來。舉例：

```
01    public class Main {
02        public static void main(String[] args) {
03            final int INT_MAX = 2147483647;
04            [int a = INT_MAX + 1;
05             long b = INT_MAX + 1;
06             long c = INT_MAX + 1L;
07             System.out.println(a);
08             System.out.println(b);
09             System.out.println(c);
10        }
11    }
```

  
java

```
-2147483648
-2147483648
2147483648    output
```

觀察這三行  
有什麼區別？



# 溢位

```
03  final int INT_MAX = 2147483647;  
04  [int a = INT_MAX + 1;  
05  long b = INT_MAX + 1;  
06  long c = INT_MAX + 1L;
```

java

-2147483648

-2147483648

2147483648

output

第 4 行：進行加法運算，並將結果賦值給 `int` 變數 `a`。加法運算兩者都是 `int`

故結果為 `int`，但因運算結果 `2147483648` 已超過 `int` 上限 `2147483647`，故發生溢位

結果變成 `-2147483648`，最後將運算結果 `-2147483648` 存入變數 `a`

第 5 行：進行加法運算，並將結果賦值給 `long` 變數 `b`。加法運算兩者都是 `int`

故結果為 `int`，但因運算結果 `2147483648` 已超過 `int` 上限 `2147483647`，故發生溢位

結果變成 `-2147483648`，最後將運算結果 `-2147483648` 提升成 `long` 並存入變數 `b`

第 6 行：進行加法運算，並將結果賦值給 `long` 變數 `c`。加法運算一個是 `int` 一個是 `long`

故 `int` 提升為 `long`，運算結果為 `long`，運算結果 `2147483648` 沒有超過 `long` 上限

並沒有發生溢位，最後將運算結果 `2147483648` 存入變數 `b`

# 轉換

型別小的變型別大的會經過提升

是個自動的過程

而型別大的變型別小的則須進行轉換

是個手動的过程

使用以下方法進行轉換，為表達式

(欲轉換型別) 值

java

若轉換的值

超過欲轉換型別的範圍

則會發生溢位

```
a
65633
a
2
32816.0
32816.5
32816.5
32816.0
output
```

```
01 public class Main {
02     public static void main(String[] args) {
03         char a = 97;
04         int b = 65536 + 97;
05         char c = (char) b;
06         int d = (byte) 258;
07         double e = b / d;
08         double f = (double) b / d;
09         double g = b / (double) d;
10         double h = (double) (b / d);
11         System.out.println(a);
12         System.out.println(b);
13         System.out.println(c);
14         System.out.println(d);
15         System.out.println(e);
16         System.out.println(f);
17         System.out.println(g);
18         System.out.println(h);
19     }
20 }
```



java

觀察哪幾行出現了  
提升、轉換、溢位？

# 提升、轉換、溢位

提升：第 6、7、8、9 行

轉換：第 5、6、8、9、10 行

溢位：第 5、6 行

```
03 char a = 97;
04 int b = 65536 + 97;
05 char c = (char) b;
06 int d = (byte) 258;
07 double e = b / d;
08 double f = (double) b / d;
09 double g = b / (double) d;
10 double h = (double) (b / d); java
```

```
a
65633
a
2
32816.0
32816.5
32816.5
32816.0 output
```

第5行：將65633轉換為char且發生溢位，溢位後變成97(char)並存入char變數c

第6行：將258轉換為byte且發生溢位，溢位後變成2(byte)提升並存入int變數d

第7行：將b(int)/d(int)的結果32816(int)提升並存入double變數e

第8行：將b(double)/d(int提升double)的結果32816.5(double)存入double變數f

第9行：將b(int提升double)/d(double)的結果32816.5(double)存入double變數g

第10行：將b(int)/d(int)的結果32816(int)轉換成double並存入double變數h

# 相等運算

運算名稱	等於	不等於
格式	運算元1 == 運算元2	運算元1 != 運算元2
功能	測試 運算元1 和 運算元2 是否相等	測試 運算元1 和 運算元2 是否不相等
結果型別	boolean	
範例	1 == 2 3 == 3.0	1 != 2 3 != 3.0

# 比較運算

比較運算也只能用在數字(含 char)

運算名稱	大於	大於等於
格式	運算元1 > 運算元2	運算元1 >= 運算元2
功能	測試 運算元1 是否 大於 運算元2	測試 運算元1 是否 大於等於 運算元2
結果型別	boolean	
範例	1 > 2 3.5 > 3.14	1 >= 2 3.5 >= 3.14

# 比較運算

運算名稱	小於	小於等於
格式	運算元1 < 運算元2	運算元1 <= 運算元2
功能	測試 運算元1 是否 小於 運算元2	測試 運算元1 是否 小於等於 運算元2
結果型別	boolean	
範例	1 < 2 3.5 < 3.14	1 <= 2 3.5 <= 3.14

# 邏輯運算

邏輯運算只能用在 `boolean`

運算名稱	否定運算	或運算	且運算
格式	<code>!運算元</code>	<code>運算元1   運算元2</code> <code>運算元1    運算元2</code>	<code>運算元1 &amp; 運算元2</code> <code>運算元 1 &amp;&amp; 運算元2</code>
功能	真變假 假變真	有一真即為真，否則為假  ：兩個運算元都會參與運算   ：若 運算元1 為真 則 運算元2 不參與運算	都為真即為真，否則為假 &：兩個運算元都會參與運算 &&：若 運算元1 為假 則 運算元2 不參與運算
結果型別	<code>boolean</code>		
範例	<code>!true</code> <code>!false</code>	<code>false   false</code> <code>true    false</code>	<code>false &amp; false</code> <code>true &amp;&amp; false</code>

# 三元運算

在 Java 中，只有一種三元運算，格式如下：

```
boolean(條件) ? 條件為真時的返回值 : 條件為假時的返回值
```

java

範例如下：

```
final int INT_MAX = 2147483647;  
final int INT_MIN = -2147483648;  
System.out.println(INT_MAX + 1 == INT_MIN ? "True" : "False");  
System.out.println(INT_MAX == INT_MIN - 1 ? "True" : "False");
```

java

```
True  
True
```

output



# 運算順序

遞增遞減運算(變數++、變數--)

- > 遞增遞減運算(++變數、--變數) = 正負號運算 = 邏輯否定運算
- > 乘法、除法、取餘運算 > 加法、減法運算
- > 比較運算 > 相等運算
- > 三元運算 > 指定運算

遇到同級運算時，除指定運算為右往左，其餘為左往右  
若遇到括號，則括號先算

# 基本輸出

除了前面介紹的"`System.out.print()`"和"`System.out.println()`"外還有一個常用的"`System.out.printf()`"，用來進行格式化輸出

```
System.out.printf("格式化字串", 引數1, 引數2, 引數3, ..., 引數n);
```

java

格式化字串裡的有些格式化符號會依序被後面的引數替換  
若替換的型別不和格式化符號不符，則會出現錯誤

格式化符號	%d	%f	%c	%b	%s	%%	%n
功能	輸出整數	輸出浮點數，可以用" <code>%.nf</code> "指定輸出小數後幾位，預設為 <code>n=6</code>	輸出字元	輸出布林	輸出字串	輸出" <code>%</code> "	換下一行
適用型別	byte、short int、long	float double	char	boolean	string	不適用	不適用

```
System.out.printf("姓名：%s 學號：%d 身高：%.2f%n", "夏禹添", 1234567, 175.1);  
// 姓名：夏禹添，學號：1234567，身高：175.10
```

java

# 字元轉換整數

若要將字元 '0' 到 '9' 轉換成整數

可以藉助前面提到的 **ASCII** :

48 為 '0' 開始依序到 57 為 '9'

所以只要將字元 '0' 到 '9' 減掉 48

即可將字元 '0' 到 '9' 轉換成整數 0 到 9

```
char eight = '8';  
System.out.println((int) eight); // ASCII 值  
System.out.println(eight - 48); // 轉換成整數  
System.out.println(eight - '0'); // '0' 即為 48 java
```

```
56  
8  
8  output
```

# 字串

一對單引號 `' '` 包起來的是字元

而一對雙引號 `" "` 包起來的是字串

每個字串都是 `java.lang.String` 類別(class)的實例(instance)

而不是字元陣列(char array)，且無法更改字串內容

除了使用 `" "` 來代表字串

也可以使用創建(create)實例的方式：`new String("")`

兩者的詳細差異會在以後說明

# 字串

字串可以使用 "+" 運算子(並非加法運算)來串接字串和其他東西  
只要其中有一個運算元是字串，就會將其他運算元變為字串並串接

```
public class Main {  
    public static void main(String[] args) {  
        int i = 1;  
        final String PREFIX = "喜歡你的第";  
        final String SUFFIX = new String("年，我還是沒告白");  
        System.out.println(PREFIX + i++ + "年，我還沒有告白");  
        System.out.println(PREFIX + i++ + SUFFIX);  
        System.out.println(PREFIX + i++ + SUFFIX);  
        System.out.println(PREFIX + i++ + SUFFIX);  
        System.out.println(PREFIX + i++ + SUFFIX);  
        System.out.println(PREFIX + i++ + "年，我終於告白了");  
    }  
}
```

java

喜歡你的第1年，我還沒有告白  
喜歡你的第2年，我還是沒告白  
喜歡你的第3年，我還是沒告白  
喜歡你的第4年，我還是沒告白  
喜歡你的第5年，我還是沒告白  
喜歡你的第6年，我終於告白了 output

6行字串串接中  
int 型別的變數 i  
變為了字串並串接

# 字串

另一種串接字串的方式是使用 `java.lang.StringBuilder`

首先使用 `"new StringBuilder()"` 創建一個新的實例

然後呼叫(call)其方法(method) `"append(arg)"` 來串接字串

特別注意，`StringBuilder` 實例不可以和字串使用 `"+"` 來串接

將上一頁的程式用 `StringBuilder` 來改寫：

```
public class Main {  
    public static void main(String[] args) {  
        int i = 1;  
        final String PREFIX = "喜歡你的第";  
        final String SUFFIX = "年，我還是沒告白";  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append("年，我還沒有告白"));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append("年，我終於告白了"));  
    }  
}
```

喜歡你的第1年，我還沒有告白  
喜歡你的第2年，我還是沒告白  
喜歡你的第3年，我還是沒告白  
喜歡你的第4年，我還是沒告白  
喜歡你的第5年，我還是沒告白  
喜歡你的第6年，我終於告白了

output

java

事實上，上一頁的程式碼編譯時會被編譯器改成這樣的寫法

# 字串

若要比較字串是否相等

應該呼叫字串的 "`equals(arg)`" 方法進行判斷

而不是使用相等運算(`==`、`!=`)來進行判斷

原因會在以後說明

```
final String a = "TYIC";  
final String b = "TYIC";  
final String c = new String("TYIC");  
final String d = new String("TYIC");  
System.out.println(a == b);  
System.out.println(a.equals(b));  
System.out.println(b == c);  
System.out.println(b.equals(c));  
System.out.println(c == d);  
System.out.println(c.equals(d));
```

java

```
true  
true  
false  
true  
false  
true
```

output

# 跳脫字元(Escape character)

反斜線 `"\"` 加上一個特定的字會形成一個有特定功能的跳脫字元

跳脫字元 名稱	換行 (LF)	回車 (CR)	單引號	雙引號	反斜線	製表符	16進位 字元
格式	<code>\n</code>	<code>\r</code>	<code>\'</code>	<code>\"</code>	<code>\\</code>	<code>\t</code>	<code>\u</code> 十六進位
功能	換行	回到該行 最前方	顯示 單引號	顯示 雙引號	顯示 反斜線	加入 <b>tab</b>	顯示 <b>16</b> 進位 代表的字元
範例	<code>\n</code>	<code>\r</code>	<code>\'</code>	<code>\"</code>	<code>\\</code>	<code>\t</code>	<code>\u6843</code> <code>\u9AD8</code>



# 換行

除了格式化符號 `"%n"` 可以換行，跳脫字元 `'\n'` 也可以  
但這兩者有著一些差異：

在非 **Unix** 系統(如 **Windows**)使用 `"\r\n"`(**CR**+**LF**)來表示換行  
而在 **Unix** 及類 **Unix** 系統(如 **Linux**)使用 `"\n"`(**CR**)來表示換行  
格式化符號 `"%n"` 則是會在不同系統下自動變為上述兩者之一

**CR** + **LF** 是打字機的操作：將列印頭移至起始點，並將紙往上移  
大多數程式都遵守寬容原則，即只要 **LF** 就表示換行

# 補充：Unix 家族

Unix 最初是由 AT&T 開發的，並免費授權給學術機構

因此產生了各種 Unix 變種，如 BSD

後來 AT&T 不再授權給學術機構，並跟變體打著作權官司

此時便出現了一個類似於 Unix 的系統：Minix

但因 Minix 只授權於教育使用

林納斯·托瓦茲(Linus Torvalds) 便創造了自由開源系統：Linux

而此時 Unix 變種也已完全重寫，並完全開源自由使用

後來 BSD 演變成了 FreeBSD 和 OpenBSD，以及 macOS

1969  
1971 to 1973  
1974 to 1975  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001 to 2002  
2003  
2004  
2005 to 2007  
2008 to 2009  
2010  
2011 to 2018  
2019 to 2023

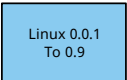
1969  
1971 to 1973  
1974 to 1975  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001 to 2002  
2003  
2004  
2005 to 2007  
2008 to 2009  
2010  
2011 to 2018  
2019 to 2023

Open source

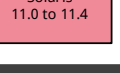
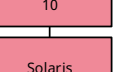
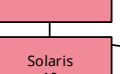
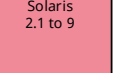
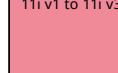
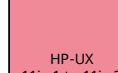
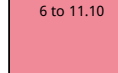
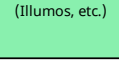
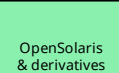
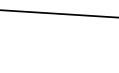
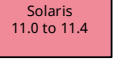
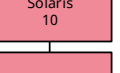
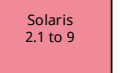
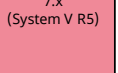
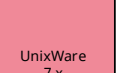
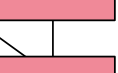
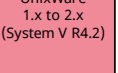
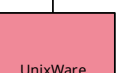
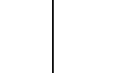
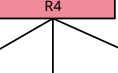
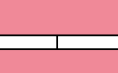
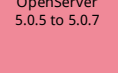
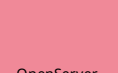
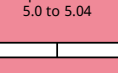
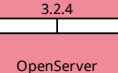
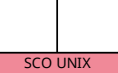
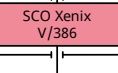
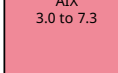
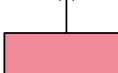
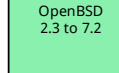
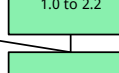
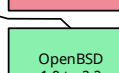
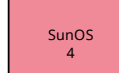
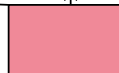
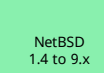
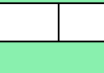
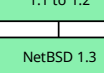
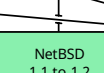
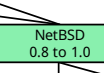
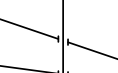
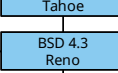
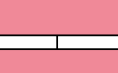
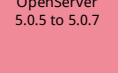
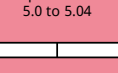
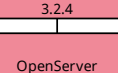
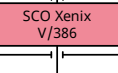
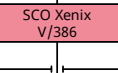
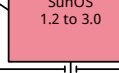
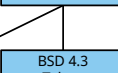
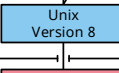
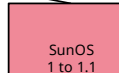
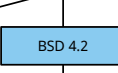
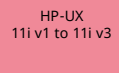
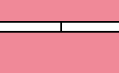
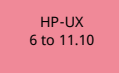
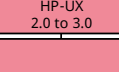
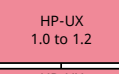
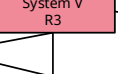
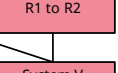
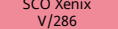
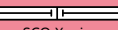
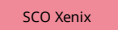
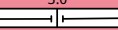
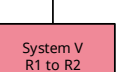
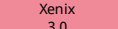
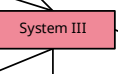
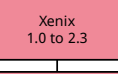
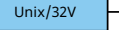
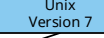
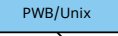
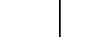
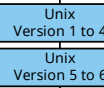
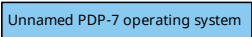
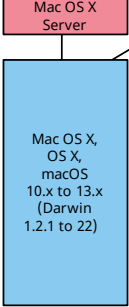
Mixed/shared source

Closed source

Unix-like systems



Apple 持有



# 基本輸入

在 Java 中，常使用 `java.util.Scanner` 來進行輸入  
要使用需要先載入(import) `java.util.Scanner` 套件(package)  
接著還需要先創建一個 `Scanner` 實例，再呼叫他的方法來讀取輸入

```
01 import java.util.Scanner; // 載入套件
02
03 public class Main {
04     public static void main(String[] args) {
05         Scanner scanner = new Scanner(System.in); // 創建新的 Scanner 實例
06         System.out.print("姓名 學號 身高:");
07         String name = scanner.next(); // 讀入下一個字串並存入變數 name
08         int studentId = scanner.nextInt(); // 讀入下一個 int 並存入變數 studentId
09         double height = scanner.nextDouble(); // 讀入下一個 double 並存入變數 height
10         System.out.printf("姓名: %s 學號: %d 身高: %.2f\n", name, studentId, height);
11     }
12 }
```

```
姓名 學號 身高: 張信喆 32767 185.1
姓名: 張信喆 學號: 32767 身高: 185.10 console
```



java

不只是 `next()`、`nextInt()`、`nextDouble()`，其他型別也可以

# 載入

為何 `Scanner` 需要 `import`，而其他的不需要？

因為 `Java` 預設會幫你 `import java.lang.*`  
而 `String` 和 `System` 都屬於 `java.lang` 的一部分

關於套件(`package`)，之後會有更詳細的敘述