物導2014年Ch1-6重點整理

Chapter 1 Getting Started

1. method: objects interact with one another by means of actions
2. two types of Java program
   - Applications: start with main method
   - Applet: run from Web browser
3. *System.out.println*
   - *System.out*: object
   - *println*: method
4. l-value:位置. r-value:內容
5.
   - byte-code
     (1) machine language for a fictitious computer (Java Virtual Machine)
     (2) can be used on any computer
6. Class Loader(linker): a program that connects the byte-code of the classes
7. *javac*: compile command
   - e.g *javac FirstProgram.java*
8. Identifier: name of a variable
   - identifiers have no maximum length
9. keywords(reserved words): predefined meaning in Java
   - e.g public, class, void, static
10. predefined identifiers: defined in libraries
    - e.g. *System*, *String*, *println*
11. <重要！>Primitive Types

TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
boolean	true or false	1 byte	not applicable
char	single character (Unicode)	2 bytes	all Unicode characters
byte	integer	1 byte	−128 to 127
short	integer	2 bytes	−32768 to 32767
int	integer	4 bytes	−2147483648 to 2147483647
long	integer	8 bytes	−9223372036854775808 to 9223372036854775807
float	floating-point number	4 bytes	$-3.40282347 \times 10^{+38}$ to $-1.40239846 \times 10^{-45}$
double	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

12. Assignment

- *temperature = 98.6;* //98.6預設為*double*
- *a=2;* //2預設為*int*
- *int intVal = 2.99;* //Illegal

```
byte   a = 1;        //1應該為int，不是應該塞不進byte嗎？但是compiler自動把他轉為byte大小
short  b = 1;
int    c = 1;
long   d = 1;
float  e = 1.0;      //error, 因為1.0為預設double，塞不進float
double f = 1.0;

a = 100000;          //error, 因為byte塞不下這麼大的數字
b = 100000;          //error, 因為short塞不下這麼大的數字
c = 100000;
d = 100000;
d = 100000000L;      //OK
e = 1.0f             //OK
```

13. Constants (literal)

- 例如光速常數：

```
<java>
public static final double c =3.0E8;   //常數不要寫在main函式裡面
<C>
const double c=3.0E8;
```

14. 兩種變數類型做四則運算後會變大者, e.g

$$byte \pm short \rightarrow short$$
$$byte \times short \rightarrow short$$
$$byte \div short \rightarrow short$$

15. Precedence Rules

- First: the unary operators: +, -, ++, —, and ! (right-to-left)
- Second: the binary arithmetic operators: *, /, and % (left-to-right)
- Third: the binary arithmetic operators: + and - (left-to-right)

16. 非class的基本資料形態若要使用則必須先給他一個明確的初始值
    class裡面的data member如果沒有明確的給他初始值，則boolean給false，剩下全部補0

17. garbage collection

- 若沒有變數指向某物件，該物件則由java的garbage collection負責刪除並把記憶體還給系統
- <C> malloc來配置，free來釋放
- <C++> new來配置，delete來釋放

<java> new來配置，可將指標變數設為null，最後自動由garbage collection處理掉

Chapter 2: Console Input and Output

1. legacy code: code that is “old fashioned” but too expensive to replace
   - e.g printf
2. Importing Packages and Classes
   - *java.lang* package is automatically imported
   - e.g *java.text.NumberFormat*
     - *java.text*: package
     - *NumberFormat*: class name
3. Scanner Class: keyboard input
   - need to *import java.util.Scanner*
   - e.g. *Scanner keyboard = new Scanner(System.in);*
   - e.g. *int numberOfPods = keyboard.nextInt();*
   - e.g. *String word = keyboard.next();*
   - e.g. *String line = keyboard.nextLine();*

Chapter 3: Flow of Control

1. switch: 只能用於*char*, *int*, *short*, *byte*, *String*
2. conditional operator

```
if (n1 > n2)     max = n1;
else       max = n2;
max = (n1 > n2) ? n1 : n2; //same as the two lines above

if (a>b)
      if(c>d) max = c;
      else  max = d;
else
      if(e>f) max = e;
      else  max = f;

max = (a>b) ?
            (c>d) ? c:d
            :
            (e>f) ? e:f;
```

3. *==* with Strings: to see if they are stored in the same memory location
4. logical operators
   (1) short-circuit (lazy evaluation): *&&*, *||*
   - *&&*：檢查第一個不成立，就不檢查第二個
   - *||*：檢查第一個成立，就不檢查第二個

```
public static void main(String[] args) {
      int a=1;
      int x=20;

      if(a>10 && x>a++){     //前者做完不成立，後者不用做

      }
      System.out.println(a);  //print出1
}
```

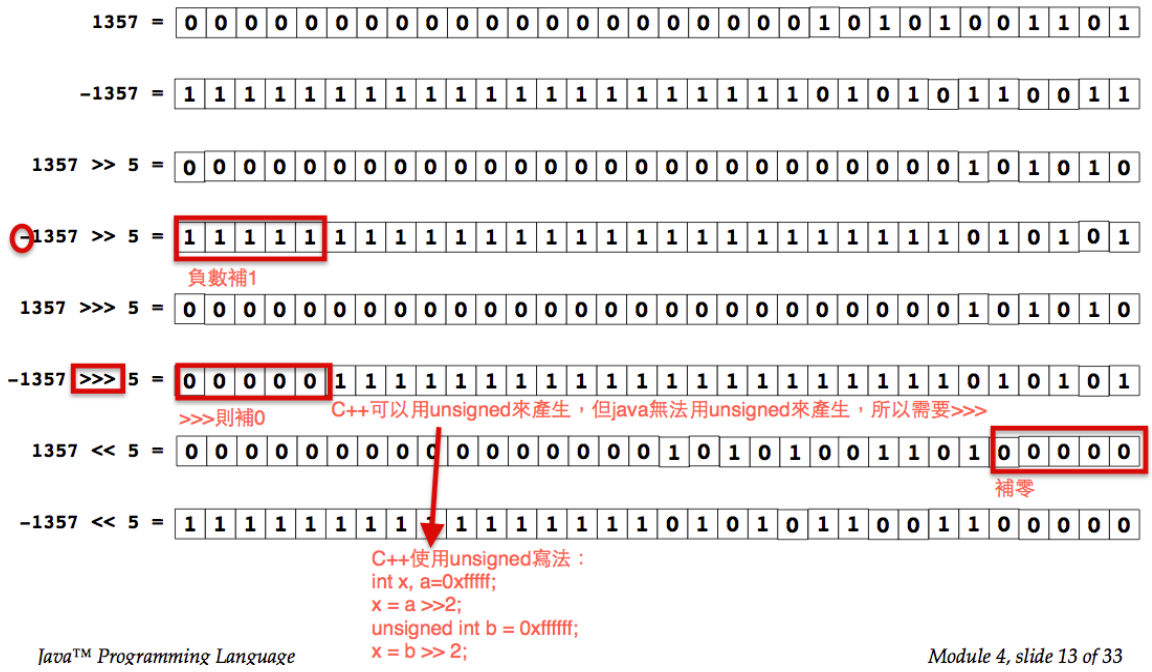
   (2) boolean operator: not(*!*), and(*&*), or(*|*), xor(*^*)

```
public static void main(String[] args) {
      int a=1;
      int x=20;

      if(a>10 & x>a++){     //前者做完不成立，後者還是要做

      }
      System.out.println(a);  //print出2
}
```

- *&*兩側為boolean expression，*&*則扮演logical operator
- *&*兩側為int，*&*則扮演bitwise logical operator

5. shift operators
   (1) signed right-shift operator *>>*
   - e.g. *128 >> 1* returns $128/2^1 = 64$, *-256 >> 4* returns $256/2^4 = -16$

   (2) unsigned right-shift operator *>>>*
   (3) signed left-shift operator *<<*
   (4) unsigned left-shift operator *<<<*



6. Precedence

Highest Precedence

PRECEDENCE	ASSOCIATIVITY
From highest at top to lowest at bottom. Operators in the same group have equal precedence.	
Dot operator, array indexing, and method invocation., `[ ]`, `( )`	Left to right
++ (postfix, as in x++), -- (postfix)	Right to left
The unary operators: +, -, ++ (prefix, as in ++x), -- (prefix), and !	Right to left
Type casts (Type)	Right to left
The binary operators *, /, %	Left to right
The binary operators +, -	Left to right
The binary operators <, >, <=, >=	Left to right
The binary operators ==, ! =	Left to right
The binary operator &	Left to right
The binary operator |	Left to right
The binary operator &&	Left to right
The binary operator ||	Left to right
The ternary operator (conditional operator) ?:	Right to left
The assignment operators =, *=, /=, %=, +=, -=, & =, |=	Right to left

Lowest Precedence

7. side effects: an expression changes something, such as value of a variable. 某指令狀態前與後，若變數值有改變就是side effect
   - e.g. assignment, increment, decrement
   - pass by value沒有side effect，因為不會改變到原本傳進去變數的值
   - pass by reference有side effect，因為會改變到原本船進去變數的值，不過java沒有pass by reference這種東西
   - example 1:

```
main(){
      int x=10;
      f(x);
      print(x); //10
      //no side effect
}

f(int a){
      a++;
}
```

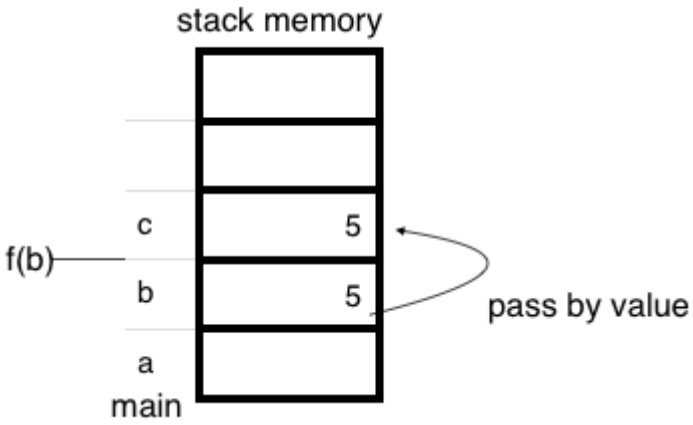
   - example 2:

```
public static void main(String[] args) {
      String x = "abc";       //x points to another memory block stored "abc"
      f(x);
      System.out.print(x);    //abc
}
```

```
public static void f(String a){
      a="def";                //a points to another memory block stored “def"
                              //因為String immutable , 所以a指向別的物件去了
}
```

8. stack memory during pass by value

```
public static void main(String[] args){
      int a;
      int b=5;
}

public static void f(int c){

}
```



9. ***exit***: end the program immediately
10. assertion checks: asserts something about the state of a program
    - if evaluates to false, program ends, and outputs an assertion failed error message
11. random numbers

```
import java.util.Random;
Random rnd = new Random();
int i = rnd.nextInt(10);                    //Random number from 0 to 9
double d = rnd.nextDouble();                // 0 <= d < 1

//亦可使用Math class的random()                    //random()輸出值>=0, <1
Math.random()*100                           //若要產生0~100的亂數
Math.random()*50+50                         //若要產生50~100的亂數
int x = (int)(Math.random()*6)+1;           //模擬dice的運作
char y = (char)(Math.random()*26+'a');      //a到z間任取一個字母
```

12. special loop flow control
    - ***break [label];***

```
outer:
      do{
            statement;
            do {
                  statement;
                  if (boolean expression){
                        break outer;     //跳出外面的do while回圈
                  }
                  statement;
            } while (boolean expression)
      } while (boolean expression)
```

    - ***continue [label];***

```
do{
      statement;
      if (boolean expression){
            continue;  //回到do的地方執行
      }
      statement;
} while (boolean expression)
```

    - ***label: statement;*** //where statement should be a loop

Chapter 4: Defining Classes I

1. programmer-defined types: typedef, enum, struct, class(blueprint for objects)
2. members = data items + methods
3. fields = instance variables = data items = attributes
4. variables
   (1) local/automatic/temporary/stack variables
   - declared within a method
   - no default value

   (2) no global variable
   (3) parameters of primitive type
   - 呼叫函式傳入的變數叫做formal parameters/parameters(參數)
   - 被呼叫函式接收的變數叫做actual parameters/argument(引述)

```
void main(String[] argv{
      int a;
      f(a);       //a為parameter
}

f(int b){         //b為argument
      b++
}
```

   - 若int傳入double類型，會發生automatically type cast
   - this
     (1) Explicit name for the calling object.指標變數(reference)指向自身物件.
     (2) must be used if same name is used in the method
     (3) static method do not contain this

   (4) member and class variables are automatically initialized

```
public class HelloJava {
      static int x;
      public static void main(String[] args) {
            int a;
            System.out.println(a); //error, because not initialized
            System.out.println(x); //no error, because x is a member
      }
}
```

5. bottom-up testing: first test all the methods invoked by that method, and then test the method itself. 被呼叫的函式全部先測試好，才能使用該函式
6. information hiding
   (1) the practice of separating how to use a class from the details of its implementation
   (2) ADT (abstract data type): data type using information hiding
7. encapsulation
   (1) data and methods of a class are combined into a single unit

(2) API (application programming interface): description of how to use a class

8. accessor methods: obtain value of an object's instance variable
9. mutator methods: change the value of an object's instance variable
10. overloading: two or more methods with same name in the same class
    - need different signatures: types of parameters
11. construct
    (1) need *new* operator to construct
    (2) if no constructor, Java automatically create a default/no-argument constructor
    (3) if include constructor, Java will not provide default constructor
        - therefore, user need to provide no-argument constructor
    (4) default constructor
        - boolean: false
        - primitive types: zero

class types: null

Chapter 5: Defining Classes II

1. static method: without calling object (所以static methods沒有this)
   - 不屬於任何一個class的method
   - e.g public static double pow(double base, double exponent)
   - static method can't access private data member and member functions
2. static variable: variable with only one copy for a class
   - static method can access static variable
   - static variable不透過constructor而建構，所以若要對static variable做初始化動作，可以寫個static block，來對所有static variable處理，如下：

```
static{

}
```

   - should always be defined private, unless it is also a defined constant
     - e.g *Math.PI* is a *public static final double*
3. Wrapper Class
   (1) boxing: primitive type to object of its wrapper class
     - e.g. *Integer integerObject = new Integer(42);*
     - automatic boxing: *Integer integerObject = 42;*

   (2) unboxing: object of a wrapper class to primitive type
     - e.g. *int i = integerObject.intValue();*
     - automatic unboxing: *int i = integerObject;*

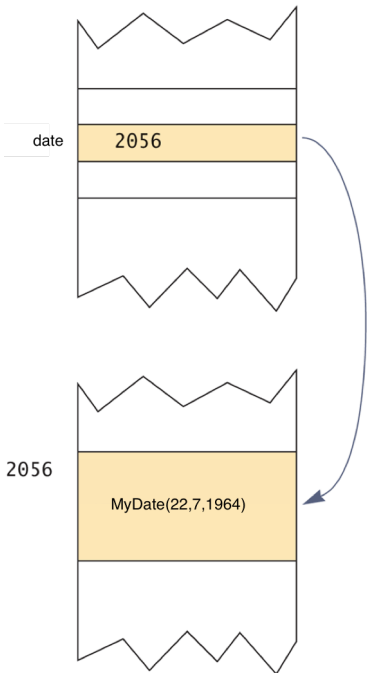
   (3) conversions:
     - convert string to number. e.g. *Integer.parseInt("123");*
     - convert number to string. e.g. *Double.toString(123.99);*
4. <重要！>reference assignment operator *=*

   e.g. *MyDate date = new MyDate(22, 7, 1964);*

   (1) new會在heap記憶體區段配置MyDate大小的空間

   (2) MyDate constructor初始化data member

   (3) 回傳自己在heap記憶體區段的位置，並指定給在stack記憶體區段的reference變數date



5. class parameters: any change made to the object named by the parameter will be made to the object named by the argument
6. anonymous object: an object whose reference is not assigned to a variable
7. *null* (constant)
   - not an object
   - placeholder for a reference that does not name any memory location
8. copy constructor: a constructor with a single argument of the same type as the class
   - primitive type: copy directly
   - class type: must creates a new one (prevent privacy leak)
   - deep copy: a copy of an object which has no references in common, except for immutable objects 所有物件都不share，除了無法修改內容的物件例外可以share
   - shallow copy: not a deep copy
9. packages: a group of classes
   (1) to make a package
   - group all classes into a single folder
   - add *package package_name* to the beginning of each class file

   (2) *java.lang* is automatically imported. It includes
   - *Math*
   - *String*
   - wrapper classes

   (3) package directories
   e.g. assume I have a file under the directory (\libraries\newlibraries)
   - a package class in the directory (\libraries\newlibraries\utitlities\numericstuff) needed to be imported

- type import *utilities.numericstuff.**
- *utilities.numericstuff* is a CLASSPATH

(4) packages prevent name clashes (a situation in which two classes have the same name)

Chapter 6: Arrays

1. Declaration: ***BaseType[] ArrayName = new BaseType[size]***
   e.g. ***double[] score = new double[5];***
   e.g. ***double[] score = {56,88,55,99,33};***
2. array is an object
3. *length* instance variable
   - return numbers of elements in the array
   - e.g. ***score.length***
4. character array to String

```
char[] a = {'A', 'B', 'C'};
String s = a;                    // illegal! Only C++ can do this

// OK! makes a new string by String constructor, s1 is "ABC"
String s1 = new String(a);

String s2 = new String(a, 0, 2)  // OK! s2 is "AB"

System.out.println(a);           // prints out ABC

int[] ar = {1,2,3,4,5};
System.out.print(ar);       // prints out the address of ar, not 12345
```

5. arrays with class base type
   - e.g. ***Date[] holidayList = new Date[20];*** //doesn't create 20 Date objects, ***null*** for all indexes
     - 所以總共要new 20次(20個Date) + 1次(Date array) = 21次
6. array parameters
   - e.g. ***public static void doubleElements (double[] a);***
   - e.g. ***public static void main (String[] args)***
     - when using terminal, ***java SomeProgram Hi ! there***
     - ***args[0]*** is "Hi"
     - ***args[1]*** is "!"
     - ***args[2]*** is "there"
7. return array
   - e.g. ***public static int[] {returns anArray;}*** //returns reference of the array, but has privacy leak

```
//deep copy for accessor to prevent privacy leak
public int[] getArray(){
      double[] temp = new double[count];
      for (int i = 0; i < count; i++)
            temp[i] = anArray[i];
      return temp;
}
```

8. for each loop
   - *for (ArrayBaseType VariableName : ArrayName)*
     Statement
   - *for (double element : arr)* //access all elements in arr, and print them
     println(element);
   - *VariableName* is a copy from the element in *ArrayName*, assignment to *VariableName* doesn't change the element in *ArrayName*
9. ellipsis
   - *Type*... *ArrayName*
   - e.g. *public static Type methodName(Type*... *ArrayName)*
     - *methodName* can take variable number of parameters
   - for example:

```
private static void m1(int[] a){
      for(int i=0; i< a.length; i++){
            System.out.println(a[i]);
      }
}

/* m2 can't overload m1 because they have same type of parameter(which is
array) */
private static void m2(int... a){      //a is an array
      //no need to pass by array from main to m2
      for(int i=0; i<a.length; i++){
            System.out.println(a[i]);
      }
}

public static void main(String[] args) {
      m1(1);            //error, need to pass by array
      m1(1,2);    //error, need to pass by array

      int[] b = {1};
      m1(b);      //ok!

      m2(1);      //ok, 當1傳入m2時，同時也被打包成array，成為陣列a的元素
      m2(1,2);    //ok, 當1,2傳入m2時，同時也被打包成array，成為陣列a的元素
      m2(1,2,3);        //ok
      m2(1,2,3,4);      //ok
      m2(b);            //ok
}
```

10. *enum*: 直接列舉所有此種自訂類型的值

- *enum TypeName {VALUE_1, VALUE_2, …, VALUE_N}*

```
enum WorkDay{MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};
WorkDay meetingDay, availableDay;
meetingDay = WorkDay.THURSDAY;
availableDay = null;
System.out.println(meetingDay);  //outputs is THURSDAY, not String value
```

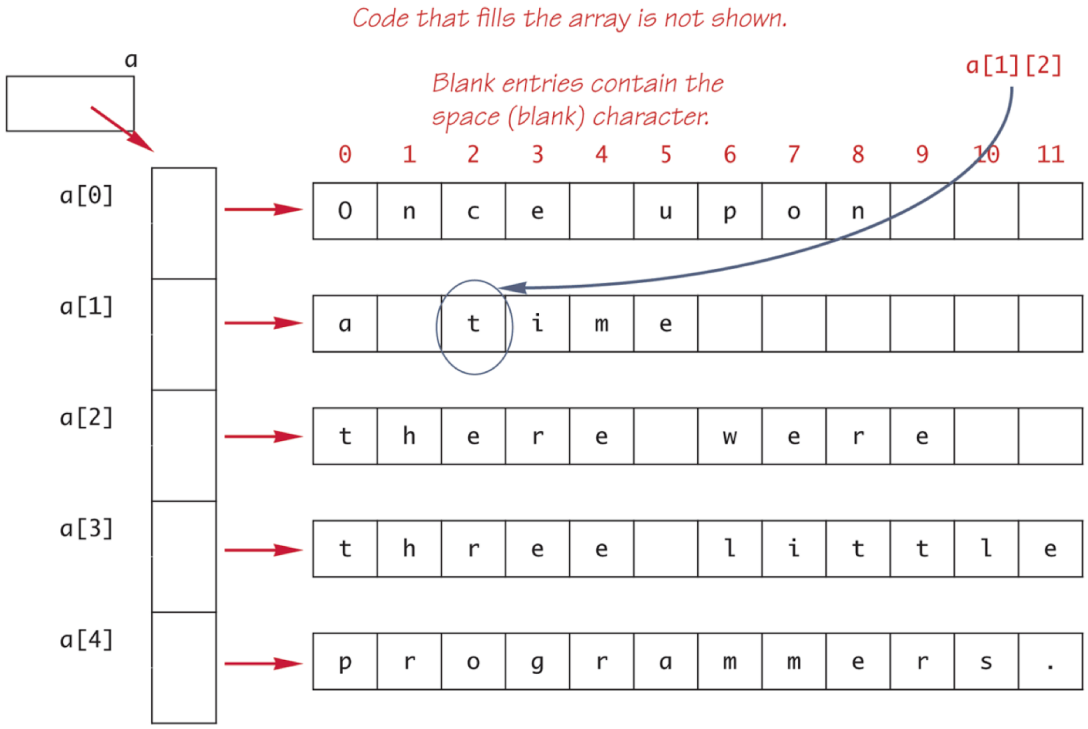
- *==* or *equals* method to compare two variables or constants
  - *values* method: returns entire enum as an array
  - e.g. *WorkDay[] day = WorkDay.values();* //day[0] is MONDAY, day[1] is TUESDAY…
- can be used with *switch* statement
- enum with data members and methods

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),     //每一項都是constant object
    VENUS   (4.869e+24, 6.0518e6),     //後面括號的數字相當於直接去呼叫了
    EARTH   (5.976e+24, 6.37814e6),    //constructor
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7),
    PLUTO   (1.27e+22,  1.137e6);

    private final double mass;   // in kilograms (data member)
    private final double radius; // in meters    (data member)
    Planet(double mass, double radius) {    //constructor
        this.mass = mass;
        this.radius = radius;
    }
    public double mass()   { return mass; }       //enum methods
    public double radius() { return radius; }

    // universal gravitational constant  (m^3 kg^-1 s^-2)
    public static final double G = 6.67300E-11;

    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

11. Multidimensional Arrays

- e.g. char ***[][] a = a[5][12];*** //宣告了一個空間為5的陣列，每項元素指向一個空間為12的char陣列，可以想成是row為5，column為12的陣列

```
char[][] a = new char[5][12];
```



- e.g. ***int[][] b = new int[4][];*** //宣告了一個空間為4個陣列，每項元素指向一個int的陣列（每個陣列的大小可能不一，稱之為ragged array）

```
int[][] b = new int[4][];
b[0] = new int[3];
b[1] = new int[5];
```

- 若想要將多維陣列都確實有物件，總共需要產生多少物件？
  - e.g. ***new int[4][5];*** //共 1 + 4 = 5個物件（因為是int，所以只需要配置array物件）
  - e.g. ***new A[3][5];*** //共 1 + 3 + 3*5 = 19個物件

	說明
1	A[0]~A[2]的一個陣列
3	A[0][0]~A[0][4], A[1][0]~A[1][4], A[2][0]~A[2][4] 三個陣列
3*5	二維陣列內所有元素需要配置之物件A的數量

  - e.g. ***new A[3][5][7];*** //共 1 + 3 + 3*5 + 3*5*7 = 124個物件
  - e.g. ***new A[3][5][7][9];*** //共 1 + 3 + 3*5 + 3*5*7 + 3*5*7*9 = 1069個物件

12. copying array: ***System.arraycopy()*** method

- ***System.arraycopy(sourceArr, sourceArr starting index, destinationArr, destinationArr starting index, element numbers);***

```
int sourceArr[] = {1,2,3,4,5};
int destinationArr[] = {6,7,8,9,10,11,12};
System.arraycopy(sourceArr, 0, destinationArr, 0, sourceArr.length);
```

13. static import:

- <SE5.0 doc> allows unqualified access to static members without inheriting from the type containing the static members
- <wiki> allows members (fields and methods) defined in a class as `public static` to be used in Java code without specifying the class in which the field is defined.

```
static import java.lang.Math.PI;
public static void main(String[] args){
      System.out.println(Math.PI);      //if no static import
      System.out.println(PI);           //with static import, no need to
                                        //specify class name
}
```