

類別的成員 → 都可分成 類別等級(類別載入就有生命) 與物件等級(需要 new 才有生命)

```
public class Static01 {

    public int x; //物 , 死
    public static int y; //類 , 活

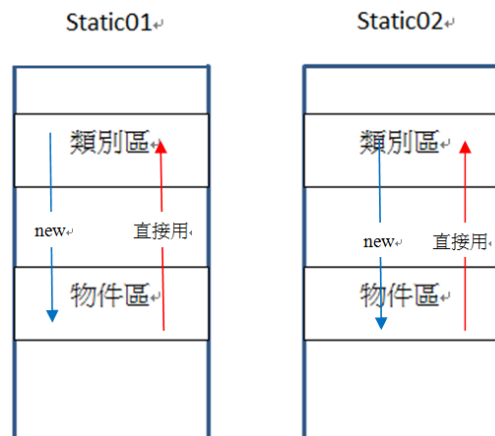
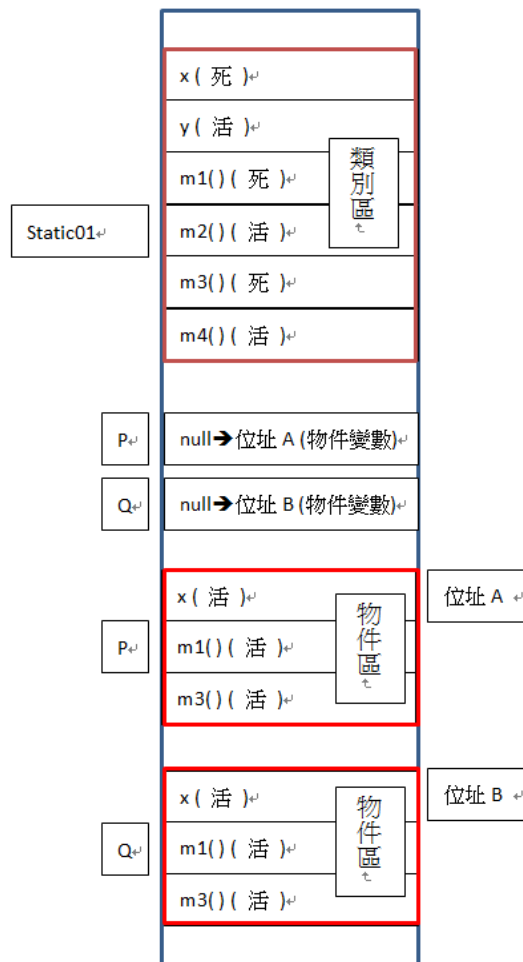
    public void m1() { //物 , 死
        x = 10;
        y = 20;
        m3();
        m4();
    }

    public static void m2() { //類 , 活
        // x=10;
        y = 20;
        // m3();
        m4();
        //要確定活才可呼叫==>自己 new 自己
        Static01 p = new Static01();
        p.x = 20;
        p.m3();
    }

    public void m3() { //物 , 死
        x = 123;
        this.x = 123;
        y = 456;
        Static01.y = 456;
        m1();
        this.m1();
        m2();
        Static01.m2();
    }

    public static void m4() { //類 , 活
        // x = 123;
        // this.x = 123;
        y = 456;
        Static01.y = 456;
        // m1();
        // this.m1();
        m2();
        Static01.m2();
    }

}
```

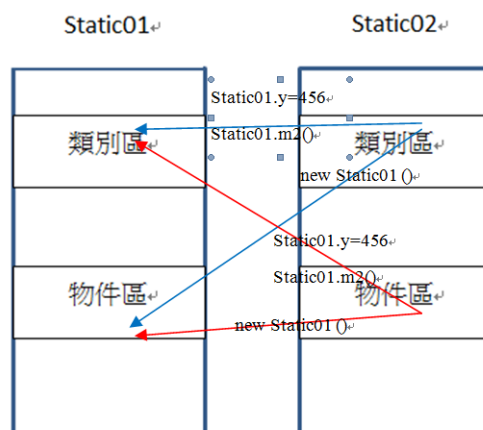


```
public static void 類別成員與物件成員() {

    Static01 P = new Static01();
    P.x = 123;
    Static01.y = 888;
    P.m1();
    Static01.m2();

    Static01 Q = new Static01();
    Q.x = 456;
    Static01.y = 999;
    Q.m1();
    Static01.m2();

}
```



```

public class Static02 {
    public int a; //物，死
    public static int b; //類，活

    public void n1() { //物，死
        //使用自己
        a = 10;
        b = 20;
        n3();
        n4();
        //使用別的類別
        Static01.y = 20;
        Static01.m4();
        Static01 q = new Static01();
        q.x = 10;
        q.m3();
    }
    public static void n2() { //類，活
        //使用自己
        // a=10;
        b = 20;
        // n3();
        n4();

        Static02 p = new Static02();
        p.a = 10;
        p.n3();
        //使用別的類別
        Static01.y = 20;
        Static01.m4();

        Static01 q = new Static01();
        q.x = 10;
        q.m3();
    }
    public void n3() { //物，死
    }
    public static void n4() { //類，活
    }
}

```

```

public class Static03 extends Static01 {

    public int a; //物，死
    public static int b; //類，活

    public void n1() { //物，死
        //使用自己
        a = 10;
        b = 20;
        n3();
        n4();
        //使用父類別繼承而來的成員
        x = 10;
        y = 20;
        m3();
        m4();
    }

    public static void n2() { //類，活
        //使用自己
        // a=10;
        b = 20;
        // n3();
        n4();

        Static03 p = new Static03();
        p.a = 10;
        p.n3();

        //使用父類別繼承而來的成員
        y = 20;
        m4();
        p.x = 10;
        p.m3();
    }

    public void n3() { //物，死
    }

    public static void n4() { //類，活
    }
}

```

物件等級的方法可使用		類別等級的方法可使用	
1. 類別等級的屬性 2. 類別等級的方法 3. 物件等級的屬性 4. 物件等級的方法		1. 類別等級的屬性 2. 類別等級的方法	
	生	死	copy
類別變數(static)	當 class 被載入時 VarDemo2.y=20 或 VarDemo2 p	當 class 不再被需要時 專案結束	一個 class 只有一份 copy · 這個 class 的所有物件 共享這份 copy
物件變數	當 object 被建立時 new VarDemo2();	當 object 不再被參考時 p=null	每個 instance 有自己的一份 copy
區域變數	當程式進入 local block 時	當程式離開 local block 時	無此概念
區塊變數	當程式進入 小區塊 時	當程式離開 小區塊 時	無此概念

```

public class Run {

    public String 名字;
    public int 公里數;
    public static int 總公里數;

    public void 跑(String n, int miles) {
        名字 = n;
        公里數 = miles;
        總公里數 = 總公里數 + miles;
        秀狀態();
    }

    public void 秀狀態() {

        System.out.println(名字);
        System.out.println("跑" + 公里數 + "公里");
        System.out.println("總共已跑" + 總公里數 + "公里");
    }

    public static void 清除總公里數() {
        總公里數 = 0;
    }
}

```



```

public static void 接力賽() {
    Run 爸1 = new Run();
    爸1.跑("爸爸1", 10);
    Run 哥1 = new Run();
    哥1.跑("哥哥1", 20);
    Run 弟1 = new Run();
    弟1.跑("弟弟1", 30);

    Run.清除總公里數();
    System.out.println("=====");

    Run 爸2 = new Run();
    爸2.跑("爸爸2", 30);
    Run 哥2 = new Run();
    哥2.跑("哥哥2", 40);
    Run 弟2 = new Run();
    弟2.跑("弟弟2", 90);

    Run.清除總公里數();
}

```

```

爸爸1
跑10公里
總共已跑10公里
哥哥1
跑20公里
總共已跑30公里
弟弟1
跑30公里
總共已跑60公里

```

```

爸爸2
跑30公里
總共已跑30公里
哥哥2
跑40公里
總共已跑70公里
弟弟2
跑90公里
總共已跑160公里

```

String 類別

char	<code>charAt(int index)</code> 返回指定索引處的 char 值。
------	----------------------------------------------------

Integer 類別

static int	<code>parseInt(String s)</code> 將字元串參數作為有符號的十進制整數進行解析。
------------	-----------------------------------------------------------

static String	<code>toString(int i)</code> 返回一個表示指定整數的 String 物件。
---------------	--------------------------------------------------------

Character 類別

static char	<code>toUpperCase(char ch)</code> 使用取自 <code>UnicodeData</code> 檔案的大小寫映射資訊將字元參數轉換為大寫。
-------------	------------------------------------------------------------------------------------------

String 類別

<code>String</code>	<code>toUpperCase()</code> 使用預設語言環境的規則將此 String 中的所有字元都轉換為大寫。
---------------------	------------------------------------------------------------------

1. char charAt(int index)
 <1>.String a="abcde"
 <2>.char b
 <3>.int c=2
 b=a.charAt(c)
 答案 → 'c'

2. static int parseInt(String s)
 <1>.Integer
 <2>.int b
 <3>.String c="123"
 b=Integer.parseInt(c)
 答案 → 123

3. static String toString(int i)
 <1>.Integer
 <2>.String b
 <3>.int c=123
 b=Integer.toString(c)
 答案 → "123"

4. static char toUpperCase(char ch)
 <1>.Character
 <2>.char b
 <3>.char c='a'
 b=Character.toUpperCase(c)
 答案 → 'A'

5. String toUpperCase()
 <1>.String a="abc"
 <2>.String b
 <3>.
 b=a.toUpperCase()
 答案 → "ABC"

String 類別

<code>String</code>	<code>concat(String str)</code> 將指定字元串連接到此字元串的結尾。
---------------------	------------------------------------------------------

String a=" abc" · String c=" xyz"

<code>int</code>	<code>indexOf(String str)</code> 返回指定子字元串在此字元串中第一次出現處的索引。
------------------	--------------------------------------------------------------

String a=" abcxyzabc" · String c=" xyz"

<code>String[]</code>	<code>split(String regex)</code> 根據給定 正則表達式 的比對拆分此字元串。
-----------------------	---------------------------------------------------------------------------

String a=" ab · cd · ef"

<code>String</code>	<code>substring(int beginIndex)</code> 返回一個新的字元串，它是此字元串的一個子字元串。
---------------------	--------------------------------------------------------------------

String a=" abcdefg" · c=3

<code>static String</code>	<code>valueOf(double d)</code> 返回 <code>double</code> 參數的字元串表示形式。
----------------------------	----------------------------------------------------------------------

double c=4.156

Character 類別

<code>static boolean</code>	<code>isDigit(char ch)</code> 確定指定字元是否為數字。
-----------------------------	-----------------------------------------------

char c=' a'

<code>static char</code>	<code>toLowerCase(char ch)</code> 使用取自 UnicodeData 檔案的大小寫映射資訊將字元參數轉換為小寫。
--------------------------	---------------------------------------------------------------------------------------------

char c=' A'

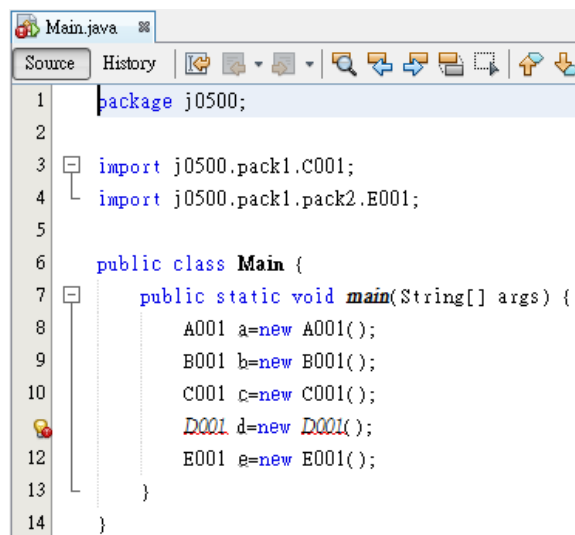
Math 類別

<code>static int</code>	<code>max(int a, int b)</code> 返回兩個 <code>int</code> 值中較大的一個。
-------------------------	------------------------------------------------------------------

int c1=4 · int c2=5

Java 的 Application 程式架構分成四個區

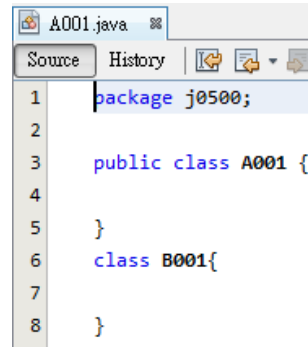
package 區，import 區，主類別區，附屬類別區



```

1 package j0500;
2
3 import j0500.pack1.C001;
4 import j0500.pack1.pack2.E001;
5
6 public class Main {
7     public static void main(String[] args) {
8         A001 a=new A001();
9         B001 b=new B001();
10        C001 c=new C001();
11        D001 d=new D001();
12        E001 e=new E001();
13    }
14 }

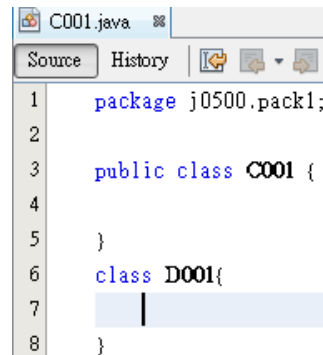
```



```

1 package j0500;
2
3 public class A001 {
4
5 }
6 class B001{
7
8 }

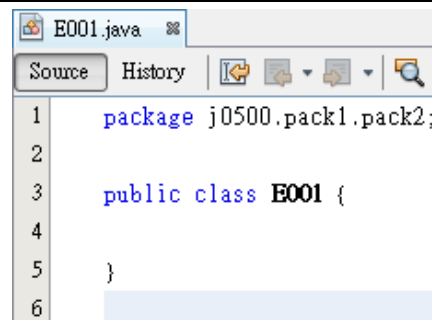
```



```

1 package j0500.pack1;
2
3 public class C001 {
4
5 }
6 class D001{
7
8 }

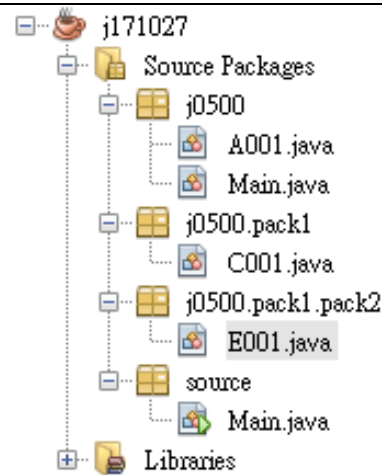
```



```

1 package j0500.pack1.pack2;
2
3 public class E001 {
4
5 }
6

```



package 區→此區不一定要有→0, 1

用來定義此 java 是隸屬於那一個 程式包裹(資料夾)

package j0500; →存放在 j0500\ 目錄下

package j0500.pack1; →存放在 j0500\pack1 目錄下

import 區→環境變數須先設定好

1. 將程式所用到的類別 import 進來
2. JVM 會自動(default) imports 兩個 packages 內的所有 classes→java.lang 及 the default package (the work directory)
3. 在一個程式內不限制有多少個 import 敘述句
4. 所有 import 敘述句都必須放在任何類別區段之上，否則會編譯錯誤

import j0500.pack1.*; →import 屬於 j0500.pack1 下的所有公開的類別

import j0500.pack1.C001; →明確的指定只要 C001 這個類別

注意 * 號並不會在 running 時影響 performance

主要類別區 (一個程式模組只能有一個類別加上 public)

1. 主要類別的名稱要與檔名一樣
2. 主要類別才可加 public 封裝修飾字，成為一個公開的類別，允許其它 Package 內的類別來 import 它

附屬類別區

1. 一個 java 程式檔內允許宣告多個一般類別
2. 所宣告的類別是要讓此程式的**主要類別(public)**所使用，所以也稱為**附屬類別**

import 的用法

```
import source.pack1.Import1;
//import source.pack1.Import3;
//import static source.pack1.Import4.x;
//import static source.pack1.Import4.s1;
import static source.pack1.Import4.*;

public static void import的用法() {
    //第一種用法 正常
    Import1 a = new Import1();
    a.x = 10;
    a.s1();

    //第二種用法 麻煩
    source.pack1.Import2 b = new source.pack1.Import2();
    b.x = 20;
    b.s1();
    source.pack2.Import2 c = new source.pack2.Import2();
    c.x = 10;
    c.s1();

    //static 複習
    //Import3.x = 30;
    //Import3.s1();
    source.pack1.Import3.x = 30;
    source.pack1.Import3.s1();

    //第三種用法 5.0 以上版本 新式用法 適用於 static 成員
    x = 30;
    s1();
}
```

```
package source.pack1;

public class Import1 {

    public int x;

    public void s1() {
    }
}
```

```
package source.pack1;

public class Import2 {

    public int x;

    public void s1() {
        System.out.println("我愛你");
    }
}
```

```
package source.pack2;

public class Import2 {

    public int x;

    public void s1() {
        System.out.println("我恨你");
    }
}
```

```
package source.pack1;

public class Import3 {

    public static int x = 4;

    public static void s1() {
    }
}
```

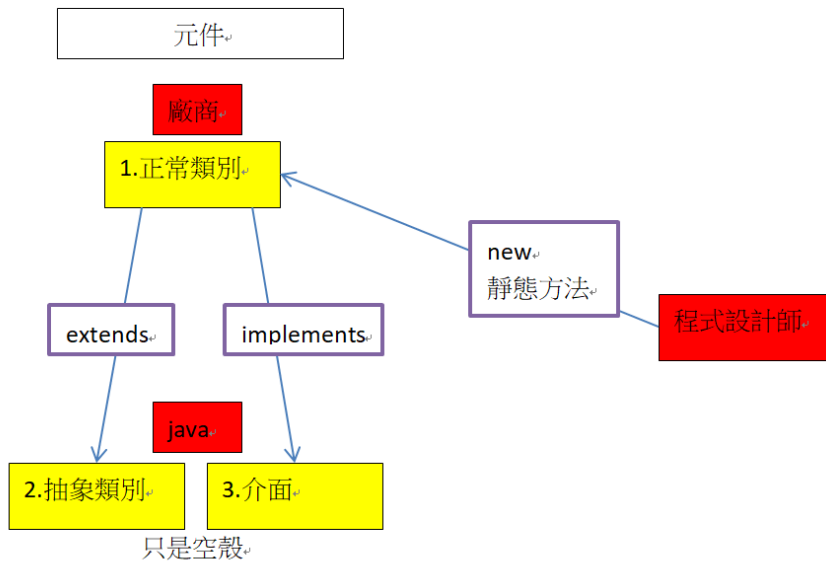
```
package source.pack1;

public class Import4 {

    public static int x = 4;

    public static void s1() {
    }
}
```


類別 與 類別之間的關係→[類別，抽象的類別，介面]，[繼承，實作，new，包覆]



```

public abstract class Conn1 {

    public abstract void conn();

    public void close() {
        System.out.println("close");
    }
}

public class MsSQL extends Conn1 {

    public void conn() {
        System.out.println("MsSQL連接了");
    }
}

public class MySql extends Conn1 {

    public void conn() {
        System.out.println("MySQL連接了");
    }
}

public class Oracle extends Conn1 {

    public void conn() {
        System.out.println("Oracle連接了");
    }
}
  
```

```

public interface Conn2 {

    void conn();

    void close();
}

public class MsSQL implements Conn2 {

    public void conn() {
        System.out.println("MsSQL連接了");
    }

    public void close() {
        System.out.println("MsSQL-close");
    }
}

public class MySql implements Conn2 {

    public void conn() {
        System.out.println("MySQL連接了");
    }

    public void close() {
        System.out.println("MySQL-close");
    }
}

public class Oracle implements Conn2 {

    public void conn() {
        System.out.println("Oracle連接了");
    }

    public void close() {
        System.out.println("Oracle-close");
    }
}
  
```

```

public static void 連接資料庫_程式設計師1() {
    source.pack3.Conn1 b = new source.pack3.MsSQL();
    b.conn();
    source.pack3.Conn1 c = new source.pack3.Oracle();
    c.conn();
    source.pack3.Conn1 d = new source.pack3.MySql();
    d.conn();
    ///////////////////////////////////
    source.pack3.Conn1 a;
    a = new source.pack3.MsSQL();
    a.conn();
    //
    a = new source.pack3.Oracle();
    a.conn();
    //
    a = new source.pack3.MySql();
    a.conn();
}

```

MsSQL連接了
Oracle連接了
MySQL連接了
MsSQL連接了
Oracle連接了
MySQL連接了

```

public static void 連接資料庫_程式設計師2() {
    source.pack4.Conn2 b = new source.pack4.MsSQL();
    b.conn();
    source.pack4.Conn2 c = new source.pack4.Oracle();
    c.conn();
    source.pack4.Conn2 d = new source.pack4.MySql();
    d.conn();
    ///////////////////////////////////
    source.pack4.Conn2 a;
    a = new source.pack4.MsSQL();
    a.conn();
    //
    a = new source.pack4.Oracle();
    a.conn();
    //
    a = new source.pack4.MySql();
    a.conn();
}

```

MsSQL連接了
Oracle連接了
MySQL連接了
MsSQL連接了
Oracle連接了
MySQL連接了

```

public class Father1 {

    public void 麵店() {
        System.out.println("牛肉麵");
    }

    public void 自助餐() {
    }
}
class Mother1 {

    public void 咖啡廳() {
    }

    public void 超市() {
    }
}

class Friend1 {

    public void 美容院() {
    }
}

class Son1 {

    public void 飲料店() {
        System.out.println("紅茶");
    }
}

```

```

public abstract class Father3 { //抽象的類別

    public void 麵店() {
        System.out.println("牛肉麵");
    }

    public abstract void 自助餐(); //抽象的方法
}
class Mother3 {

    public void 咖啡廳() {
    }

    public void 超市() {
    }
}
class Friend3 {

    public void 美容院() {
    }
}
class Son3 extends Father3 {

    public void 飲料店() {
        System.out.println("紅茶");
    }

    public void 麵店() { //覆寫
        System.out.println("義大利麵");
    }

    public void 自助餐() { //實作
        System.out.println("自助餐");
    }
}

```

```

public class Father2 {

    public void 麵店() {
        System.out.println("牛肉麵");
    }

    public void 自助餐() {
    }
}
class Mother2 {

    public void 咖啡廳() {
    }

    public void 超市() {
    }
}
class Friend2 {

    public void 美容院() {
    }
}
class Son2 extends Father2 {

    public void 飲料店() {
        System.out.println("紅茶");
    }

    public void 麵店() { //覆寫
        System.out.println("義大利麵");
    }
}

```

```

public static void 類別之間的關係() {
    Son2 p = new Son2();
    p.麵店();
}

```

義大利麵

```
public abstract class Father4 { //抽象的類別
    public void 麵店() {
        System.out.println("牛肉麵");
    }
    public abstract void 自助餐(); //抽象的方法
}
interface Mother4 { //介面
    void 咖啡廳();
    void 超市();
}
interface Friend4 { //介面
    void 美容院();
}
class Son4 extends Father4 implements Mother4, Friend4 {
    public void 飲料店() { //兒子自己的
        System.out.println("紅茶");
    }
    public void 麵店() { //覆蓋爸爸
        System.out.println("義大利麵");
    }
    public void 自助餐() { //實作爸爸
        System.out.println("自助餐");
    }
    public void 咖啡廳() { //實作媽媽
        System.out.println("拿鐵");
    }
    public void 超市() { //實作媽媽
        System.out.println("日常用品");
    }
    public void 美容院() { //實作朋友
        System.out.println("洗頭");
    }
}
```

類別 (new) 類別

類別 (new) 子類別 (extends) 父類別 1

類別 (new) 子類別 (extends) 父類別 1 (implements) 父介面 1，父介面 2，父介面 3

extends 與 implements 的差別

```

public class 鳥1 {
    public void 飛() {
        System.out.println("鳥飛");
    }
}
class 魚1 {
    public void 游泳() {
        System.out.println("魚游泳");
    }
}
class 飛魚1 extends 鳥1 {
    public void 飛() {
        System.out.println("飛魚飛");
    }
}
class 飛機1 extends 鳥1 {
    public void 飛() {
        System.out.println("飛機飛");
    }
}
class 潛水艇1 extends 魚1 {
    public void 游泳() {
        System.out.println("潛水艇游泳");
    }
}
class 人1 extends 魚1 {
    public void 游泳() {
        System.out.println("人游泳");
    }
}

```

```

public interface 飛2 {
    void 飛();
}
interface 游泳2 {
    void 游泳();
}
abstract class 人類 {
    String 名字;
    int 身高;
    int 體重;

    public abstract void 吃();
}
class 鳥2 implements 飛2 {
    public void 飛() {
        System.out.println("鳥飛");
    }
}
class 魚2 implements 游泳2 {
    public void 游泳() {
        System.out.println("魚游泳");
    }
}
class 飛魚2 extends 魚2 implements 飛2 {
    public void 飛() {
        System.out.println("飛魚飛");
    }
}
class 飛機2 implements 飛2 {
    public void 飛() {
        System.out.println("飛機飛");
    }
}

class 潛水艇2 implements 游泳2 {

    public void 游泳() {
        System.out.println("潛水艇游泳");
    }
}

```

new 的關係

建立類別(元件) 2. 建立物件 3. 產生物件實體 4. 使用物件的屬性與方法

```
//class Object{
//
//}
public class Person1 {

    public String 名字="AAA";
    public int 身高=162;
    public int 體重=50;
    public static int 人數;

    public Person1() {
        super();
        名字="BBB";
        身高=172;
        體重=60;
    }

    public static void 我愛你() {
        System.out.println("我愛你");
    }

    public void 秀名字身高體重人數() {
        System.out.println("名字=" + 名字);
        System.out.println("身高=" + 身高);
        System.out.println("體重=" + 體重);
        System.out.println("目前人數=" + 人數);
    }
}
```

```
public static void 由類別產生物件1() {
    Person1 人1 = new Person1();
    人1.名字 = "劉德華";
    人1.身高 = 183;
    人1.體重 = 70;
    Person1.人數++;
    人1.秀名字身高體重人數();
    Person1.我愛你();
    System.out.println(人1 instanceof Person1);
    System.out.println(人1.getClass().getName());

    System.out.println("=====");
    Person1 人2 = new Person1();
    人2.名字 = "張學友";
    人2.身高 = 171;
    人2.體重 = 60;
    Person1.人數++;
    人2.秀名字身高體重人數();
    Person1.我愛你();
    System.out.println(人2 instanceof Person1);
    System.out.println(人2.getClass().getName());
}
```

```
名字=劉德華
身高=183
體重=70
目前人數=1
我愛你
true
java05_new與繼承.Person1
=====
名字=張學友
身高=171
體重=60
目前人數=2
我愛你
true
java05_new與繼承.Person1
```

初始化的順序

1. 預設初始化
2. 明顯的初始化
3. 建構函數初始化

```

public class Person2 { //加了建構函數

    public String 名字;
    public int 身高;
    public int 體重;
    public static int 人數;

    public Person2() { //建構函數多載
        super();
        人數++;
    }

    public Person2(String n, int h, int w) { //建構函數多載
        super();
        名字 = n;
        身高 = h;
        體重 = w;
        人數++;
    }

    public static void 我愛你() {
        System.out.println("我愛你");
    }

    public void 秀名字身高體重人數() {
        System.out.println("名字=" + 名字);
        System.out.println("身高=" + 身高);
        System.out.println("體重=" + 體重);
        System.out.println("目前人數=" + 人數);
    }
}

```

```

public static void 由類別產生物件2() {
    Person2 人1;
    人1 = new Person2("劉德華", 183, 70);
    人1.秀名字身高體重人數();
    Person1.我愛你();

    System.out.println("=====");
    Person2 人2 = new Person2("張學友", 171, 60);
    人2.秀名字身高體重人數();
    Person2.我愛你();

    System.out.println("=====");
    Person2 人3 = new Person2();
    人3.秀名字身高體重人數();
    Person2.我愛你();
}

```

```

名字=劉德華
身高=183
體重=70
目前人數=1
我愛你

```

```

=====
名字=張學友
身高=171
體重=60
目前人數=2
我愛你

```

```

=====
名字=null
身高=0
體重=0
目前人數=3
我愛你

```

```

public class Person3 {
    public String 名字;
    public int 身高;
    public int 體重;
    public static int 人數;

    public String 眼睛;
    public String 鼻子;
    public String 嘴巴;

    public Person3() {
        super();
        人數++;
    }
    public Person3(String n, int h, int w) {
        super();
        名字 = n;
        身高 = h;
        體重 = w;
        人數++;
    }

    public static void 我愛你() {
        System.out.println("我愛你");
    }
    public static void 我恨你() {
        System.out.println("我恨你");
    }

    public void 秀名字身高體重人數() {
        System.out.println("名字=" + 名字);
        System.out.println("身高=" + 身高);
        System.out.println("體重=" + 體重);
        System.out.println("目前人數=" + 人數);
    }
}

```

```

public static void 由類別產生物件4() {
    Person4 人1 = new Person4();
    人1.名字 = "賴玉珊";
    人1.身高 = 160;
    人1.體重 = 50;
    人1.頭.眼睛 = "藍色";
    人1.頭.鼻子 = "很挺";
    人1.頭.嘴巴 = "很大";

    // 人1.秀名字身高體重人數();
    System.out.println(人1);

    Person4.說.我愛你();
    Person4.說.我恨你();
}

```

名字=賴玉珊
 身高=160
 體重=50
 眼睛=藍色
 鼻子=很挺
 嘴巴=很大
 目前人數=1

我愛你
 我恨你

```

public class Person4 {

    public String 名字;
    public int 身高;
    public int 體重;
    public static int 人數;
    public Head4 頭 = new Head4(); // 必需先 new
    public static Say4 說 = new Say4(); //必需先 new

    public Person4() {
        super();
        人數++;
    }

    public Person4(String n, int h, int w) {
        super();
        名字 = n;
        身高 = h;
        體重 = w;
        人數++;
    }

    public void 秀名字身高體重人數() {
        System.out.println("名字=" + 名字);
        System.out.println("身高=" + 身高);
        System.out.println("體重=" + 體重);
        System.out.println("目前人數=" + 人數);
        System.out.println("眼睛=" + 頭.眼睛);
        System.out.println("鼻子=" + 頭.鼻子);
        System.out.println("嘴巴=" + 頭.嘴巴);
    }

    public String toString() {
        return "名字=" + 名字 + "\n"
            + "身高=" + 身高 + "\n"
            + "體重=" + 體重 + "\n"
            + "眼睛=" + 頭.眼睛 + "\n"
            + "鼻子=" + 頭.鼻子 + "\n"
            + "嘴巴=" + 頭.嘴巴 + "\n"
            + "目前人數=" + 人數 + "\n";
    }
}

class Head4 {

    public String 眼睛;
    public String 鼻子;
    public String 嘴巴;
}

class Say4 {

    public void 我愛你() {
        System.out.println("我愛你");
    }

    public void 我恨你() {
        System.out.println("我恨你");
    }
}

```


物件的內容 → toString() → StringBuffer, StringBuilder, 八個包裝類別, File, Date, 集合 → 已有覆寫

```

232 * </pre></blockquote>
233 *
234 * @return a string representation of the object.
235 */
236 public String toString() {
237     return getClass().getName() + "@ " + Integer.toHexString(hashCode());
238 }
239

```

```

public static void 物件的內容() {

    Person2 人1 = new Person2("賴玉珊", 162, 50); //自訂類別
    int[] ar1 = new int[5]; //陣列

    StringBuffer str1 = new StringBuffer("abc"); //StringBuffer
    StringBuilder str2 = new StringBuilder("xyz"); //StringBuilder
    Integer num1 = new Integer(123); //八個包裝類別
    Double num2 = new Double(456.78);
    Date d1 = new Date(); //日期
    System.out.println("人1=" + 人1);
    System.out.println("ar1=" + ar1);
    System.out.println("str1=" + str1);
    System.out.println("str2=" + str2);
    System.out.println("num1=" + num1);
    System.out.println("num2=" + num2);
    System.out.println("d1=" + d1);
}

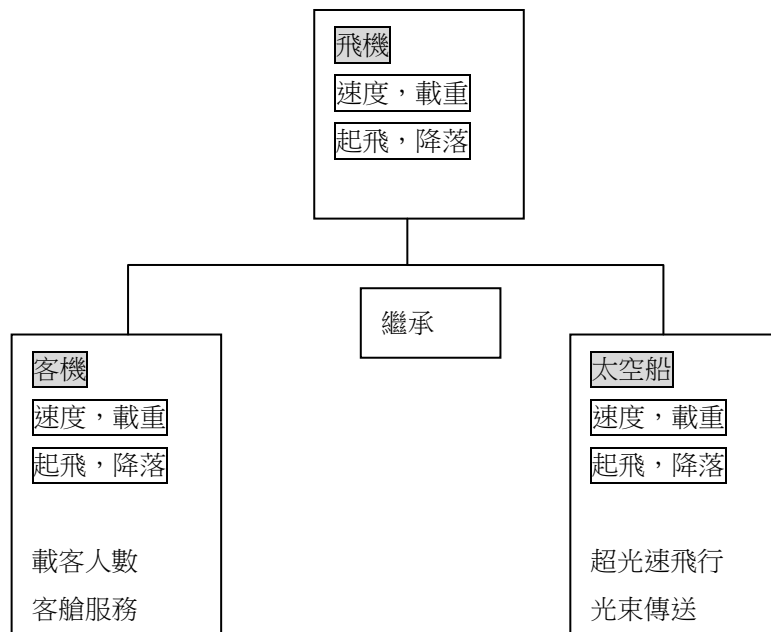
```

```

人1=source.Person2@19e0bfd
ar1=[I@139a55
str1=abc
str2=xyz
num1=123
num2=456.78
d1=Thu Jun 02 08:15:54 CST 2016

```

繼承的關係



```

public class Plane { //飛機

    public int 速度;
    public int 載重 = 1000;

    public void 起飛() {
        速度 = 10;
        System.out.println("我是飛機-起飛速度:" + 速度);
    }

    public void 降落() {
    }

}
  
```

```

public class Airplane extends Plane {

    public int 載客人數;
    public double 載重 = 10000.0; //遮蔽

    public void 客艙服務() {
        System.out.println("我是客艙服務");
    }

    public void 起飛() { //覆蓋
        速度 = 100;
        System.out.println("我是客機-起飛速度:" + 速度);
    }

}
  
```

```

public class Aircraft extends Plane { //太空船

    public void 超光速飛行() {
        速度 = 10000;
        System.out.println("太空船-超光速飛行, 速度=" + 速度);
    }

    public void 光速傳送() {
    }

}
  
```

```
public static void 繼承1() {
    Plane 飛機1 = new Plane();
    飛機1.起飛();

    Airplane 客機1 = new Airplane();
    客機1.起飛();

    Aircraft 太空船1 = new Aircraft();
    太空船1.超光速飛行();
    太空船1.起飛();
}
```

我是飛機-起飛速度:10
我是客機-起飛速度:100
太空船-超光速飛行, 速度=10000
我是飛機-起飛速度:10

```
public static void 繼承2() {
    Plane 飛機1 = new Plane();
    飛機1.起飛();

    Plane 客機1 = new Airplane();
    客機1.起飛();

    Plane 太空船1 = new Aircraft();
    太空船1.起飛();
}
```

我是飛機-起飛速度:10
我是客機-起飛速度:100
我是太空船-起飛速度:1000

```
public static void 繼承3() {
    Plane 飛機1;

    飛機1 = new Plane();
    飛機1.起飛();

    飛機1 = new Airplane();
    飛機1.起飛();

    飛機1 = new Aircraft();
    飛機1.起飛();
}
```

```
public static void 繼承4() {
    Plane[] 飛機 = {new Plane(), new Airplane(), new Aircraft()};

    for (Plane x : 飛機) {
        x.起飛();
    }
}
```

is-a → extends , implements

has -a → instance variable

1. Inheritance : 通常為 “is a ” 的關係, 比如 狗 is a 動物
2. Composition : 通常為 “has a ” 的關係, 比如一樣東西由一系列組件構成 (車有輪胎、窗子、發動機。。。)

```
// Dog5 is a Animal5 , Cat5 is a Animal5
// Dog5 has a Tail5 , Beagle5 has a Tail5
// Beagle5 is a Dog5 , Beagle5 is a Jumper5
```

```
public class Animal5 {
}

class Tail5 { //尾巴
}

interface Jumper5 {
}

class Dog5 extends Animal5 {

    Tail5 tail;
}

class Cat5 extends Animal5 {
}

class Beagle5 extends Dog5 implements Jumper5 {
}
```

```
public static void instance的關係1() {
    //Animal1 是父類別 , Dog1 是子類別
    Dog5 p = new Dog5();
    System.out.println("p instanceof Dog5 =" + (p instanceof Dog5)); //兒子 - true
    System.out.println("p instanceof Animal5 =" + (p instanceof Animal5)); //父親 - true
    System.out.println("p instanceof Cat5 =" + (p instanceof Cat5)); //沒有任何關係 , 所以錯誤
}
```

```
public static void instance的關係2() {
    //Animal1 是父類別 , Dog1 是子類別
    Dog5 p = new Dog5();
    呼叫instance的關係(p);
}

//Compiler 時 會認為 p 是 Object 的型態 與 Cat5 有繼承關係
//run 時才會挖出 p 真正的內容是 Dog5
public static void 呼叫instance的關係(Object p) {
    System.out.println("p instanceof Dog5 =" + (p instanceof Dog5)); //兒子 - true
    System.out.println("p instanceof Animal5 =" + (p instanceof Animal5)); //父親 - true
    System.out.println("p instanceof Cat5 =" + (p instanceof Cat5)); //不相關 -false
}
```

java05_new 與繼承.doc		21 / 22	2019/9/21
1. String concat(String str) <1>.String a="abc" <2>.String b <3>.String c="xyz" b=a.concat(c) 答案➡"abcxyz"	2. int indexOf(String str) <1>.String a="abcxyzabc" <2>.int b <3>.String c="xyz" b=a.indexOf(c) 答案➡3	3. String[] split(String regex) <1>.String a="ab , cd , ef" <2>.String[] b <3>.String c=" , " b=a.split(c) 答案➡b[0]="ab" b[1]="cd" b[2]="ef"	
4. String subString(int beginIndex) <1>.String a="abcdefg" <2>.String b <3>.int c=3 b=a.subString(c) 答案➡"defg"	5. static String valueOf(double d) <1>.String <2>.String b <3>.double c=4.156 b=String.valueOf(c) 答案➡"4.156"	6. static boolean isDight(char ch) <1>.Character <2>.boolean b <3>.char c='a' b=Character.isDight(c) 答案➡false	
7. static char toLowerCase(char ch) <1>.Character <2>.char b <3>.char c='A' b=Character.toLowerCase(c) 答案➡'a'	8. static int max(int a , int b) <1>.Math <2>.int b <3>.int c1=4 , int c2=5 b=Math.max(c1 , c2) 答案➡5		

作業

```
public class Student {  
  
}  
  
public class TestStudent {  
  
    public static void main(String[] args) {  
  
        Student s1 = new Student("甲", 78, 75, 79);  
        Student.寫檔案(s1.toString());  
        Student s2 = new Student("乙", 77, 68, 45);  
        Student.寫檔案(s2.toString());  
        Student s3 = new Student("丙", 69, 99, 45);  
        Student.寫檔案(s3.toString());  
        Student s4 = new Student("丁", 65, 69, 33);  
        Student.寫檔案(s4.toString());  
        Student s5 = new Student("戊", 46, 79, 68);  
        Student.寫檔案(s5.toString());  
  
    }  
}  
//Student.txt  
//甲,78.0,75.0,79.0,232.0,77.33333333333333  
//乙,77.0,68.0,45.0,190.0,63.33333333333336  
//丙,69.0,99.0,45.0,213.0,71.0  
//丁,65.0,69.0,33.0,167.0,55.666666666666664  
//戊,46.0,79.0,68.0,193.0,64.33333333333333
```