多載➔參數的 型態 · 個數 · 順序 要不相同

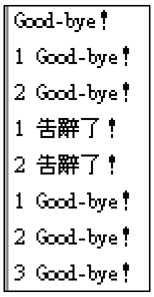
```java
public class OverDemo1P {

    protected void m1(double x) {
    }
}

class OverDemo1C extends OverDemo1P {

    public void m1() {
    }

    private static double m1(char x) {
        return 2.2;
    }

    public void m1(int x) {
    }

    final void m1(int x, float y) {
    }

    protected int m1(float x, int y) {
        return 1;
    }
}
```

複雜的多載

1. 標準 與 放寬

```java
public class OverDemo2 {

    public void sayGoodbye() {
        System.out.println("Good-bye！");
    }

    public void sayGoodbye(int times) {
        for (int x = 1; x <= times; x++) {
            System.out.println(x + " Good-bye！");
        }
    }

    public String sayGoodbye(String msg, int times) {
        for (int x = 1; x <= times; x++) {
            System.out.println(x + " " + msg);
        }
        return msg;
    }
}
```

```java
public static void 多載() {
    OverDemo2 cyh = new OverDemo2();
    cyh.sayGoodbye();
    cyh.sayGoodbye(2);
    cyh.sayGoodbye("告辭了！", 2);

    byte b = 3;
    //傳入 byte 也可，會自動轉型為 int(放寬)
    cyh.sayGoodbye(b);
}
```

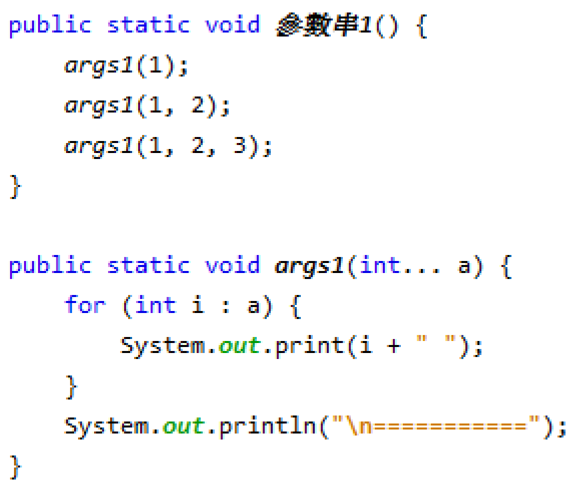


```
Good-bye！
1 Good-bye！
2 Good-bye！
1 告辭了！
2 告辭了！
1 Good-bye！
2 Good-bye！
3 Good-bye！
```

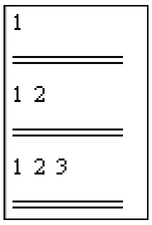
2. 參數串

<1>.以陣列方式接收，...只能在中間，不能在左右

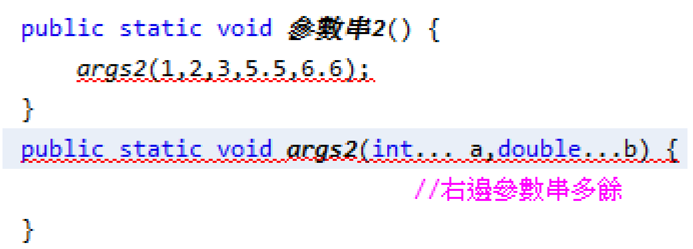
<2>.一個括號內只允許放一個參數串，並且放在參數的最右邊（因為參數串是不固定的）



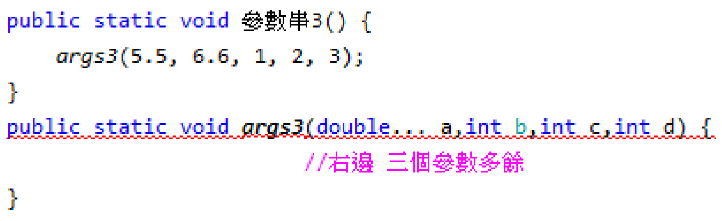
```java
public static void 參數串1() {
    args1(1);
    args1(1, 2);
    args1(1, 2, 3);
}

public static void args1(int... a) {
    for (int i : a) {
        System.out.print(i + " ");
    }
    System.out.println("\n===========");
}
```



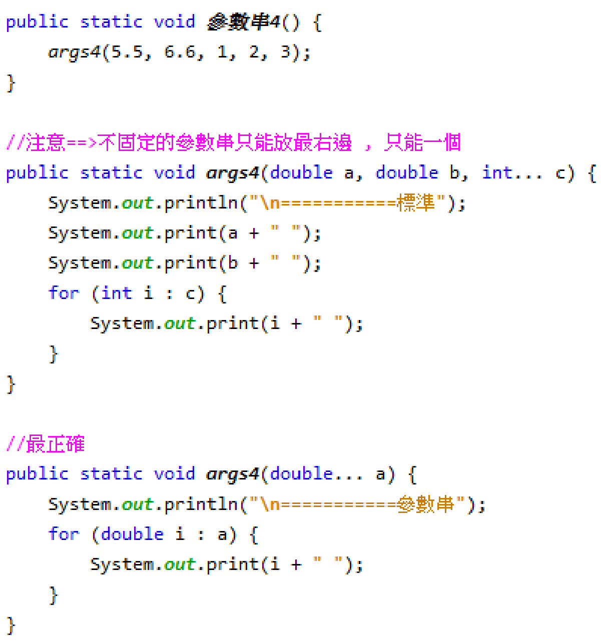
```
1
===========
1 2
===========
1 2 3
===========
```



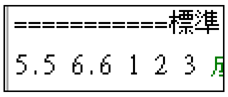
```java
public static void 參數串2() {
    args2(1,2,3,5.5,6.6);
}
public static void args2(int... a,double...b) {
                              //右邊參數串多餘
}
```



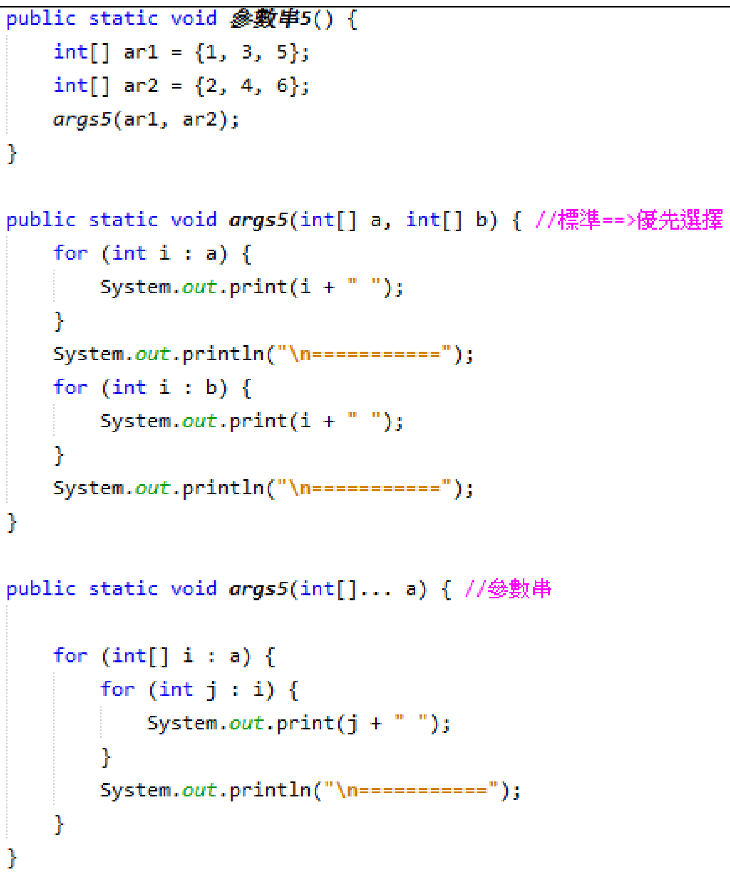
```java
public static void 參數串3() {
    args3(5.5, 6.6, 1, 2, 3);
}
public static void args3(double... a,int b,int c,int d) {
                          //右邊 三個參數多餘
}
```



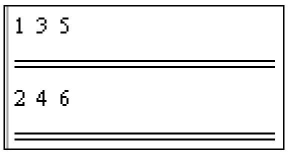
```java
public static void 參數串4() {
    args4(5.5, 6.6, 1, 2, 3);
}

//注意==>不固定的參數串只能放最右邊 ， 只能一個
public static void args4(double a, double b, int... c) {
    System.out.println("\n===========標準");
    System.out.print(a + " ");
    System.out.print(b + " ");
    for (int i : c) {
        System.out.print(i + " ");
    }
}

//最正確
public static void args4(double... a) {
    System.out.println("\n===========參數串");
    for (double i : a) {
        System.out.print(i + " ");
    }
}
```



```
===========標準
5.5 6.6 1 2 3
```



```java
public static void 參數串5() {
    int[] ar1 = {1, 3, 5};
    int[] ar2 = {2, 4, 6};
    args5(ar1, ar2);
}

public static void args5(int[] a, int[] b) { //標準==>優先選擇
    for (int i : a) {
        System.out.print(i + " ");
    }
    System.out.println("\n===========");
    for (int i : b) {
        System.out.print(i + " ");
    }
    System.out.println("\n===========");
}

public static void args5(int[]... a) { //參數串

    for (int[] i : a) {
        for (int j : i) {
            System.out.print(j + " ");
        }
        System.out.println("\n===========");
    }
}
```



```
1 3 5
===========
2 4 6
===========
```

3. 包裝型別➔8 個基本型別都各有一個相對應的包裝類別

byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

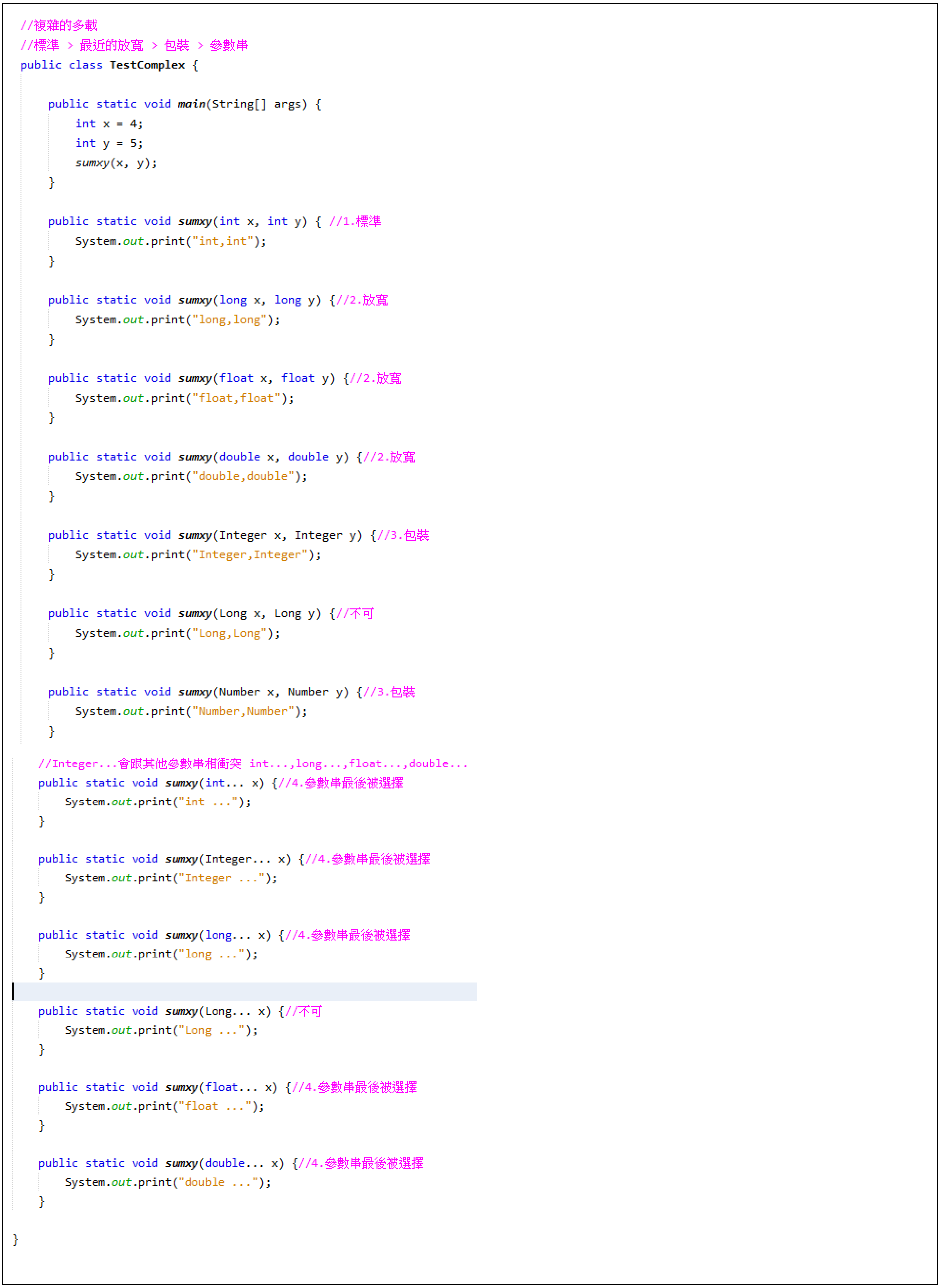
1. 數值變成字串
2. 字串剖析數值
3. 數值包成物件
4. 物件拆解成數值
5. 字串包成物件

```java
public static void 包型類別1() {
    //數值變成字串
    String s1 = Integer.toString(123); //"123"
    String s2 = String.valueOf(123);

    //字串剖析成數值
    int i1 = Integer.parseInt("123");  //123

    //數值包成物件
    Integer I1 = new Integer(123); //1.4
    Integer I2 = 123; //5.0版 以後 會自動包

    //物件拆解成數值
    int i2 = I1.intValue();  //1.4
    int i3 = I1;  //5.0版 以後會 自動拆解

    //字串包成物件
    Integer I3 = new Integer("123"); //失去字串特性
    Integer I4 = Integer.valueOf("123"); //失去字串特性

    int i4 = I3;  //5.0 拆解 變 數字

}
```

```java
public static void 包型類別2() {
    //自動解
    Integer a = 4;
    s1(a);
    //自動包
    int b = 4;
    s2(b);
}

public static void s1(int x) { //基本型別
    System.out.println(x);
}

public static void s2(Integer x) {//包裝型別
    System.out.println(x);
}
```

```java
public static void 包型類別3() {

    if (s1("true")) {
        System.out.println(true);
    }
}

public static Boolean s1(String x) {
    Boolean y = new Boolean(x);
    return y;
}
```

1.標準 > 2.放寬> 3.包裝 > 4.參數串 (int... 與 Integer...只能擇其一，否則會無法辯識)



```java
//複雜的多載
//標準 > 最近的放寬 > 包裝 > 參數串
public class TestComplex {

    public static void main(String[] args) {
        int x = 4;
        int y = 5;
        sumxy(x, y);
    }

    public static void sumxy(int x, int y) { //1.標準
        System.out.print("int,int");
    }

    public static void sumxy(long x, long y) {//2.放寬
        System.out.print("long,long");
    }

    public static void sumxy(float x, float y) {//2.放寬
        System.out.print("float,float");
    }

    public static void sumxy(double x, double y) {//2.放寬
        System.out.print("double,double");
    }

    public static void sumxy(Integer x, Integer y) {//3.包裝
        System.out.print("Integer,Integer");
    }

    public static void sumxy(Long x, Long y) {//不可
        System.out.print("Long,Long");
    }

    public static void sumxy(Number x, Number y) {//3.包裝
        System.out.print("Number,Number");
    }

   //Integer...會跟其他參數串相衝突 int...,long...,float...,double...
   public static void sumxy(int... x) {//4.參數串最後被選擇
       System.out.print("int ...");
   }

   public static void sumxy(Integer... x) {//4.參數串最後被選擇
       System.out.print("Integer ...");
   }

   public static void sumxy(long... x) {//4.參數串最後被選擇
       System.out.print("long ...");
   }

   public static void sumxy(Long... x) {//不可
       System.out.print("Long ...");
   }

   public static void sumxy(float... x) {//4.參數串最後被選擇
       System.out.print("float ...");
   }

   public static void sumxy(double... x) {//4.參數串最後被選擇
       System.out.print("double ...");
   }

}
```

覆蓋(Override)，修改從父類別繼承來的方法➔不小，同，同，同，子集

```
public abstract class OverDemo3P {

    public int x;
    private int y;  //子類別看不到

    void abc(int x, float y) {
    }

    final void xyz() {//子類別無法覆蓋
    }

    public static int m1() {//子類別也要 static
        return 1;
    }

    public abstract void m2();

    protected void m3(float z) {
    }

    private void m4(int x) {//子類別看不到
    }

    private final void m5() {//子類別看不到
    }
}
```

```
class OverDemo3C extends OverDemo3P {

    public float x;
    public int y;

    public void abc(int x, float y) {
    }

    void xyz() { //不能覆寫 , 因為父親是 final

    }
    public static int m1() {//因為父親是 static 我也要 static
        return 1;
    }

    public void m2() {
    }

    protected final void m3(float z) {//依然有覆蓋父親
    }

    private void m4(int x) {//自行定義
    }

    void m5() {//自行定義
    }
}
```

Covariant 回傳

當一個子類別想要變更繼承而來的實作(覆寫)時，子類別所定義的函式，其回傳值的型態必須和繼承來的型態一樣，但在 java5，還被允許可以在覆寫函式上變更回傳型別，只要新的回傳型別是回原函式回傳型別的子型別，也就是 Covariant 回傳

```
class One {
}

class Two extends One {
}

class Three extends Two {
}

class Four{

}

public class OverDemo4P {

    public One abc() {
        return new One();
    }
}

class OverDemo4C extends OverDemo4P {

    public Two abc() {
        return new Two();
    }
}
```

```
public interface A {

    public void doSomething(String thing);
}

class AImpl implements A {

    public void doSomething(String msg) {
    }
}
```

```
class B {

    public A doit() {
        return new AImpl();
    }
    public String execute() {
        return "abc";
    }
}
class C extends B {

    public AImpl doit() {
        return new AImpl();
    }
    public Object execute() {
        return new Object();
    }
}
```

指派關係

```
public class 動物 {
}

class 狗 extends 動物 {
}

class 貓 extends 動物 {
}

class 吉娃娃 extends 狗 {
}

class 植物 {
}
```

```
public static void 指派1() {

    動物 ani = new 動物();
    狗 dog = new 狗();
    貓 cat = new 貓();
    植物 plant = new 植物();
    String s = "abc";

    ani = dog;          //子轉父
    ani = cat;          //子轉父

    dog = (狗) ani;   //父轉子 要轉型
    cat = (貓) ani;   //父轉子 要轉型

    //彼此之間沒有關係不能轉型
    plant = (植物) ani;
    s = (String) ani;

    //兄弟關係 不行
    dog = (狗) cat;
    cat = (貓) dog;
}
```

```
public static void 指派2() {
    動物 ani = new 動物(); //父
    動物 dog = new 狗(); //子1
    動物 cat = new 貓(); //子2
}
```

```
public static void 指派3() {
    動物 ani; //父
    ani = new 動物();
    ani = new 狗();
    ani = new 貓();
}
```

多型(Polymorphism)➔建立在 繼承 與 覆蓋 的基礎上

```
public class Poly01P {

    public void 習慣() {
        System.out.println("父-抽煙");
    }
}

class Poly01C extends Poly01P {

    public void 習慣() {
        System.out.println("子1-吃檳榔");
    }
}

class Poly02C extends Poly01P {


    public void 習慣() {
        System.out.println("子2-喝酒");
    }
}
```

```
public static void 覆蓋() {
    Poly01P a = new Poly01P();
    a.習慣();
    Poly01C b = new Poly01C();
    b.習慣();
    Poly02C c = new Poly02C();
    c.習慣();
}
```

```
父-抽煙
子1-吃檳榔
子2-喝酒
```

```
public static void 多型1() {
    Poly01P a = new Poly01P();
    a.習慣();
    Poly01P b = new Poly01C();
    b.習慣();
    Poly01P c = new Poly02C();
    c.習慣();
}
```

```
父-抽煙
子1-吃檳榔
子2-喝酒
```

```
public static void 多型2() {
    Poly01P a;
    a = new Poly01P();
    a.習慣();
    a = new Poly01C();
    a.習慣();
    a = new Poly02C();
    a.習慣();
}
```

```
父-抽煙
子1-吃檳榔
子2-喝酒
```

```
public static void 多型應用_動態繫結1() {
    Scanner scanner = new Scanner(System.in);
    int input;
    Poly01P r;
    while (true) {
        System.out.print("請問要 :  1.抽煙 2.吃檳榔 3.喝酒 4.其它. 離開 :");
        input = scanner.nextInt();

        if (input == 1) {
            r = new Poly01P();
        } else if (input == 2) {
            r = new Poly01C();
        } else if (input == 3) {
            r = new Poly02C();
        } else {
            break;
        }
        r.習慣();
        System.out.println();
    }
    System.out.println();
}
```

```
請問要 :  1.抽煙 2.吃檳榔 3.喝酒 4.其它. 離開 :
1
抽煙

請問要 :  1.抽煙 2.吃檳榔 3.喝酒 4.其它. 離開 :
2
吃檳榔

請問要 :  1.抽煙 2.吃檳榔 3.喝酒 4.其它. 離開 :
3
喝酒
```

```java
public abstract class Traffic1 {

    protected static int miles;

    public abstract void speedUp();
}

class Airplane1 extends Traffic1 {

    public void speedUp() {
        miles += 15;
        System.out.println("駕駛飛機,加速中,前進" + miles + "公里");
    }
}

class Car1 extends Traffic1 {

    public void speedUp() {
        miles += 2;
        System.out.println("駕駛車子,加速中,前進" + miles + "公里");
    }
}
```

```java
public static void 多型應用_動態繫結2() {
    Scanner scanner = new Scanner(System.in);
    int input;

    Traffic1 r;
    Car1 mycar = new Car1();
    Airplane1 myAirplane = new Airplane1();

    while (true) {
        System.out.print("請問要駕駛 : 1.車子 2.飛機 3.其它. 離開 :");
        input = scanner.nextInt();

        if (input == 1) {
            r = mycar;
        } else if (input == 2) {
            r = myAirplane;
        } else {
            break;
        }
        r.speedUp();
        System.out.println();
    }
    System.out.println();
}
```

```
請問要駕駛 : 1.車子 2.飛機 3.其它. 離開 :
1
駕駛車子,加速中,前進2公里

請問要駕駛 : 1.車子 2.飛機 3.其它. 離開 :
2
駕駛飛機,加速中,前進17公里
```

```java
public class Grade {

    public int[] grades;

    public final void setGrades(int[] g) { //不得覆蓋此方法
        if (grades == null) {
            grades = g;
        } else {
            System.out.println("考試次數已確定，不得更改！");
        }
    }

    public double average() {
        if (grades == null) {
            return 0;
        }
        int tot = 0;
        for (int i = 0; i < grades.length; i++) {
            tot = tot + grades[i];
        }
        return (double) tot / grades.length;
    }
}

class Grade2 extends Grade {

    public double average() {
        if (grades == null) {
            return 30;
        }
        int tot = 0;
        for (int i = 0; i < grades.length; i++) {
            if (grades[i] >= 45 && grades[i] < 60) {
                grades[i] = 60;
            }
            tot = tot + grades[i];
        }
        return (double) tot / grades.length;
    }
}
```

```java
public static void 多型應用3() {

    int[] theGrades = {46, 58, 52, 76, 49, 67, 74, 81};
    Grade stud = new Grade();
    stud.setGrades(theGrades);
    System.out.println("(原始)平均 = " + stud.average());
    stud = new Grade2();
    stud.setGrades(theGrades);
    System.out.println("(加分)平均 = " + stud.average());
}
```

```
(原始)平均 = 62.875
(加分)平均 = 67.25
```

多型與覆蓋不一樣的地方

1. 多型只有 物件等級的方法 會用到子類別

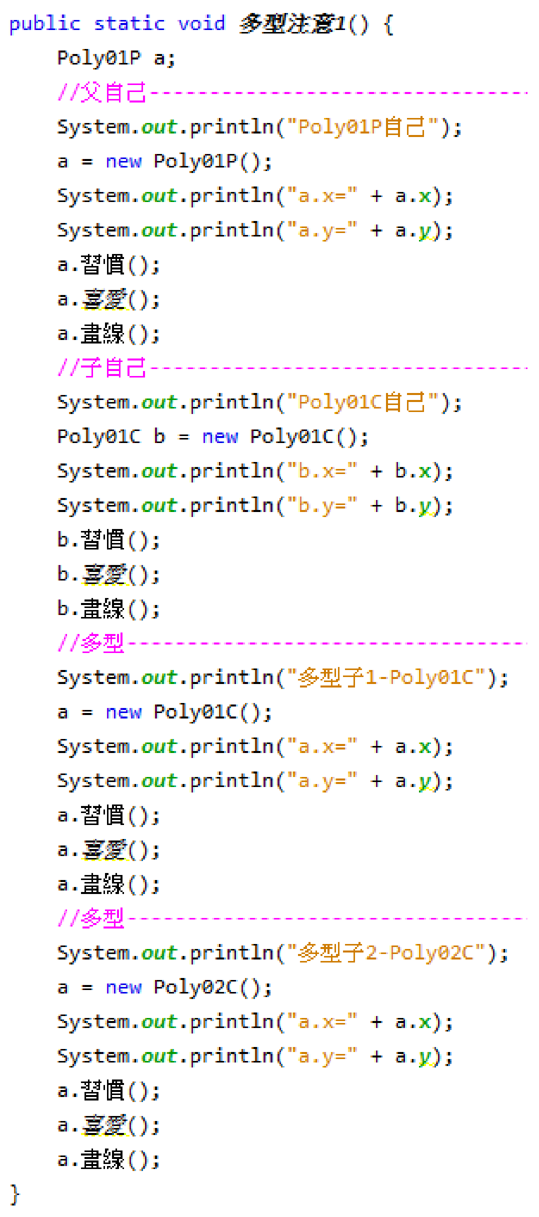
   其餘　　物件等級的屬性，類別等級的屬性，類別等級的方法 ➔一律使用到父類別

2. 子類別只有繼承，沒有覆蓋的物件等級的方法，一樣延用父類別的方法

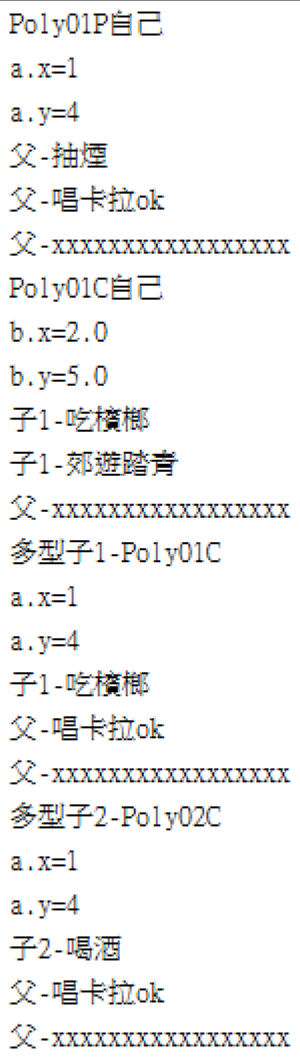
```java
public class Poly01P {

    public int x = 1;
    public static int y = 4;

    public void 習慣() {
        System.out.println("父-抽煙");
    }
    public static void 喜愛() {
        System.out.println("父-唱卡拉ok");
    }
    public void 畫線() {
        System.out.println("父-xxxxxxxxxxxxxxxxx");
    }
}
```

```java
class Poly01C extends Poly01P {

    public double x = 2.0;
    public static double y = 5.0;

    public void 習慣() {
        System.out.println("子1-吃檳榔");
    }
    public static void 喜愛() {
        System.out.println("子1-郊遊踏青");
    }
    public void 討厭() {
        System.out.println("子1-吃苦瓜");
    }
}
```

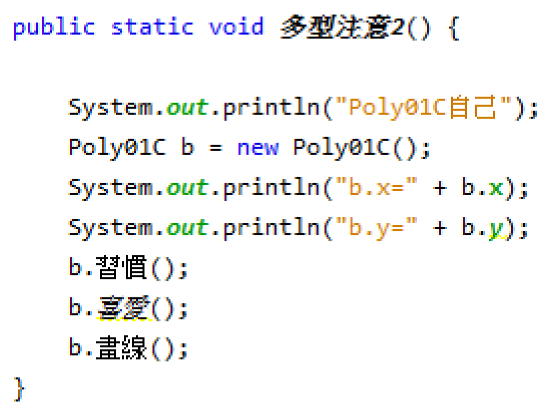
```java
class Poly02C extends Poly01P {

    public double x = 3.0;
    public static double y = 6.0;

    public void 習慣() {
        System.out.println("子2-喝酒");
    }
    public static void 喜愛() {
        System.out.println("子2-釣魚");
    }
}
```



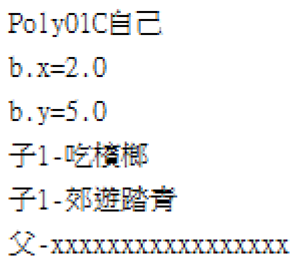
```java
public static void 多型注意1() {
    Poly01P a;
    //父自己--------------------------------
    System.out.println("Poly01P自己");
    a = new Poly01P();
    System.out.println("a.x=" + a.x);
    System.out.println("a.y=" + a.y);
    a.習慣();
    a.喜愛();
    a.畫線();
    //子自己--------------------------------
    System.out.println("Poly01C自己");
    Poly01C b = new Poly01C();
    System.out.println("b.x=" + b.x);
    System.out.println("b.y=" + b.y);
    b.習慣();
    b.喜愛();
    b.畫線();
    //多型--------------------------------
    System.out.println("多型子1-Poly01C");
    a = new Poly01C();
    System.out.println("a.x=" + a.x);
    System.out.println("a.y=" + a.y);
    a.習慣();
    a.喜愛();
    a.畫線();
    //多型--------------------------------
    System.out.println("多型子2-Poly02C");
    a = new Poly02C();
    System.out.println("a.x=" + a.x);
    System.out.println("a.y=" + a.y);
    a.習慣();
    a.喜愛();
    a.畫線();
}
```



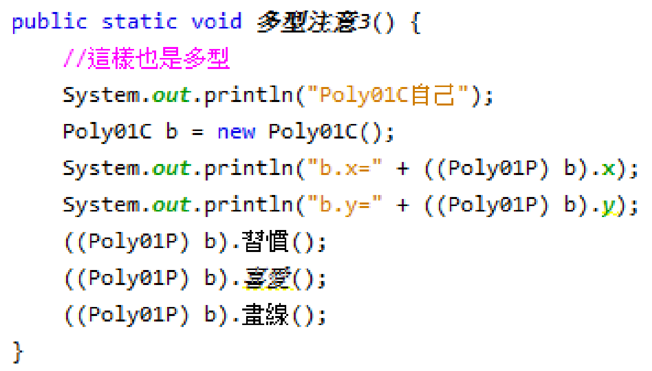
```
Poly01P自己
a.x=1
a.y=4
父-抽煙
父-唱卡拉ok
父-xxxxxxxxxxxxxxxxx
Poly01C自己
b.x=2.0
b.y=5.0
子1-吃檳榔
子1-郊遊踏青
父-xxxxxxxxxxxxxxxxx
多型子1-Poly01C
a.x=1
a.y=4
子1-吃檳榔
父-唱卡拉ok
父-xxxxxxxxxxxxxxxxx
多型子2-Poly02C
a.x=1
a.y=4
子2-喝酒
父-唱卡拉ok
父-xxxxxxxxxxxxxxxxx
```



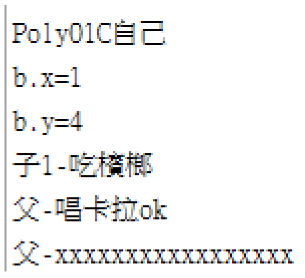
```java
public static void 多型注意2() {

    System.out.println("Poly01C自己");
    Poly01C b = new Poly01C();
    System.out.println("b.x=" + b.x);
    System.out.println("b.y=" + b.y);
    b.習慣();
    b.喜愛();
    b.畫線();
}
```



```
Poly01C自己
b.x=2.0
b.y=5.0
子1-吃檳榔
子1-郊遊踏青
父-xxxxxxxxxxxxxxxxx
```



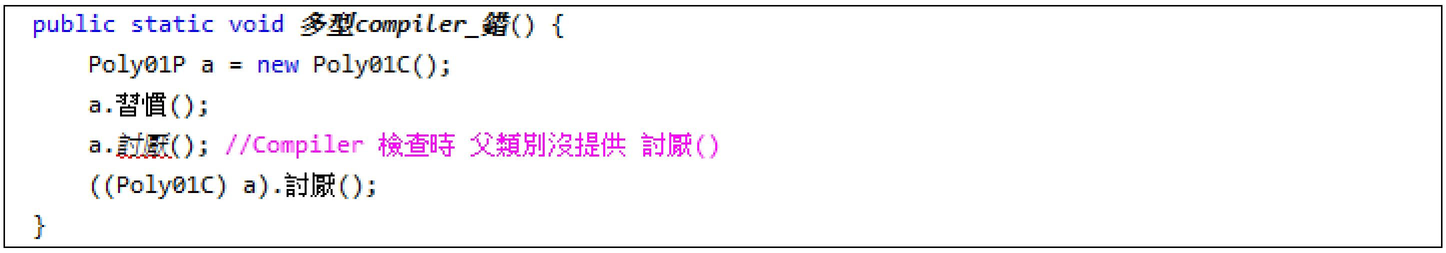
```java
public static void 多型注意3() {
    //這樣也是多型
    System.out.println("Poly01C自己");
    Poly01C b = new Poly01C();
    System.out.println("b.x=" + ((Poly01P) b).x);
    System.out.println("b.y=" + ((Poly01P) b).y);
    ((Poly01P) b).習慣();
    ((Poly01P) b).喜愛();
    ((Poly01P) b).畫線();
}
```



```
Poly01C自己
b.x=1
b.y=4
子1-吃檳榔
父-唱卡拉ok
父-xxxxxxxxxxxxxxxxx
```

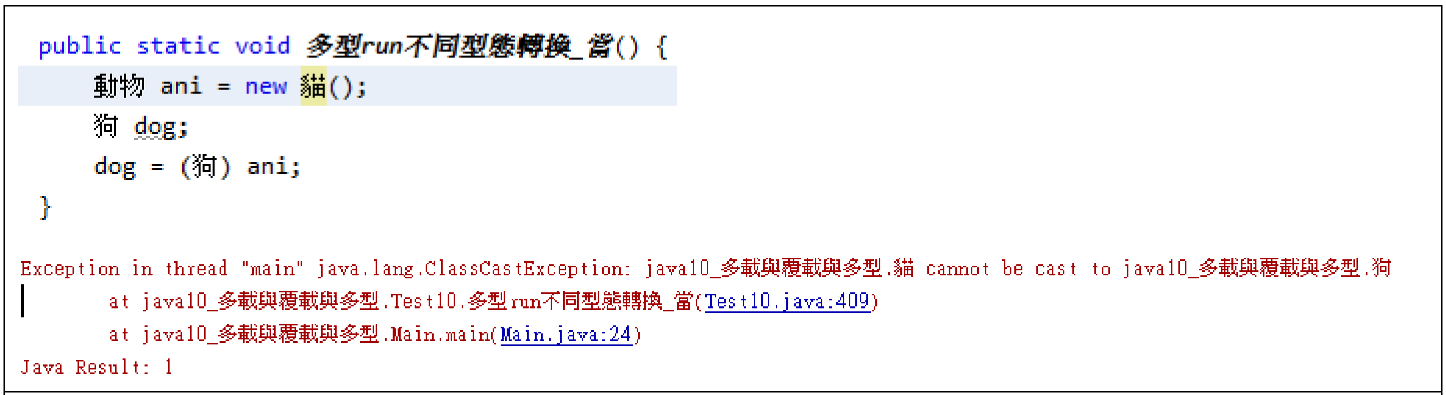
多型的錯

1. compiler 時 a 會認為是父親的型態

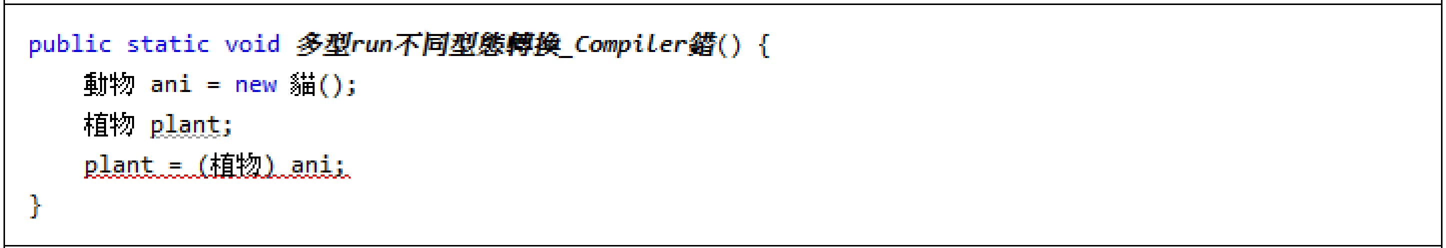


```
public static void 多型compiler_錯() {
     Poly01P a = new Poly01C();
     a.習慣();
     a.討厭(); //Compiler 檢查時 父類別沒提供 討厭()
     ((Poly01C) a).討厭();
}
```

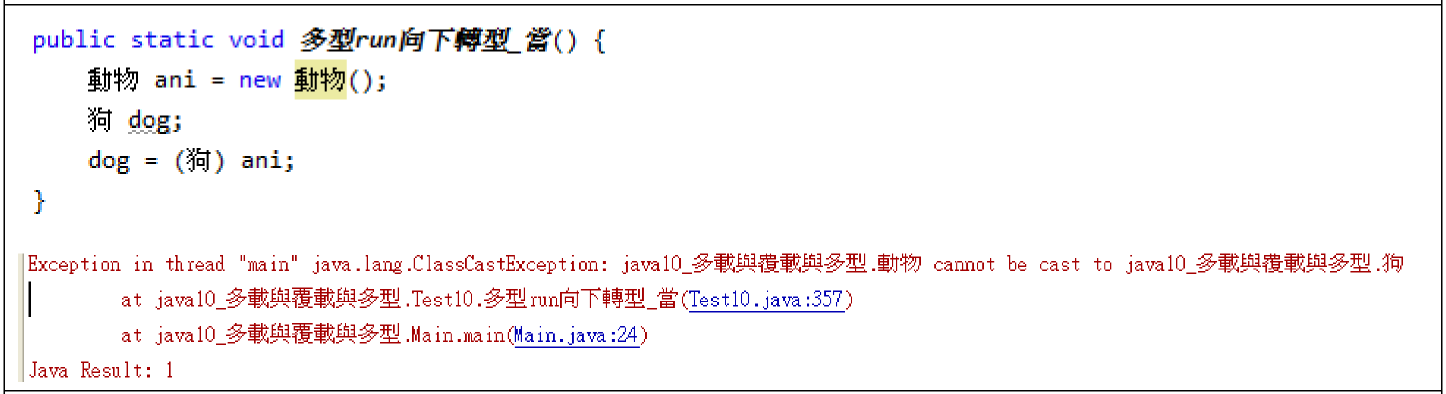
2. Run-time 時➔compiler 時檢查到 (ani➔型態是動物) 與 (dog➔型態是狗) 之間有繼承關係，但執行時要把 ani 的實體 貓 指派給 dog (雖然有經過轉型的動作) 但還是會當掉，因為 貓與狗 無繼承關係



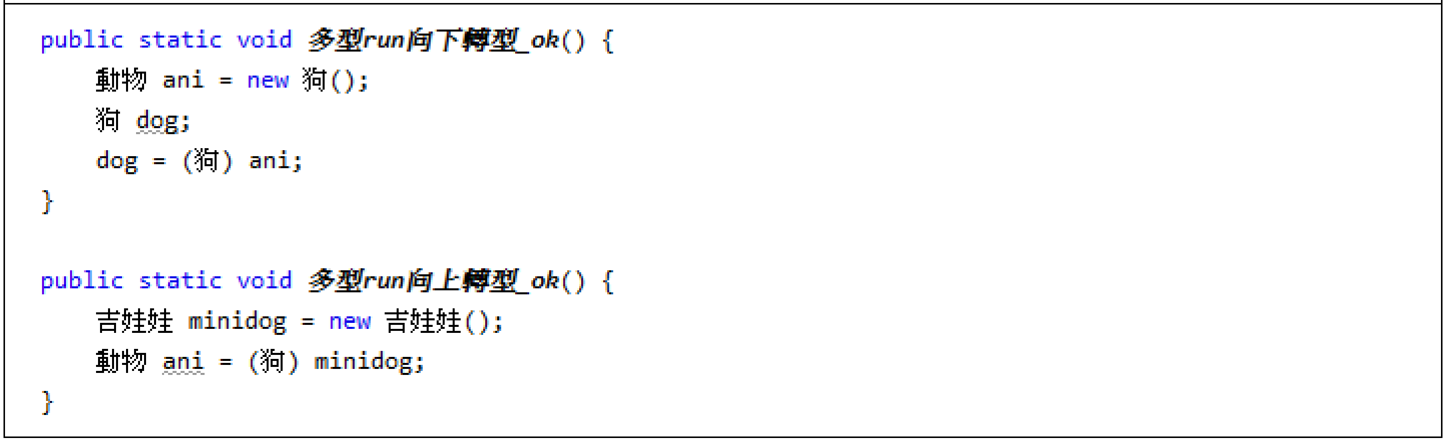
```
public static void 多型run不同型態轉換_當() {
     動物 ani = new 貓();
     狗 dog;
     dog = (狗) ani;
}

Exception in thread "main" java.lang.ClassCastException: java10_多載與覆載與多型.貓 cannot be cast to java10_多載與覆載與多型.狗
        at java10_多載與覆載與多型.Test10.多型run不同型態轉換_當(Test10.java:409)
        at java10_多載與覆載與多型.Main.main(Main.java:24)
Java Result: 1
```



```
public static void 多型run不同型態轉換_Compiler錯() {
     動物 ani = new 貓();
     植物 plant;
     plant = (植物) ani;
}
```



```
public static void 多型run向下轉型_當() {
     動物 ani = new 動物();
     狗 dog;
     dog = (狗) ani;
}

Exception in thread "main" java.lang.ClassCastException: java10_多載與覆載與多型.動物 cannot be cast to java10_多載與覆載與多型.狗
        at java10_多載與覆載與多型.Test10.多型run向下轉型_當(Test10.java:357)
        at java10_多載與覆載與多型.Main.main(Main.java:24)
Java Result: 1
```



```
public static void 多型run向下轉型_ok() {
     動物 ani = new 狗();
     狗 dog;
     dog = (狗) ani;
}

public static void 多型run向上轉型_ok() {
     吉娃娃 minidog = new 吉娃娃();
     動物 ani = (狗) minidog;
}
```

```java
public class Foo {
}

class Alpha extends Foo {
}

class Beta extends Alpha {
}

class Detla extends Beta {
}
```

```java
public static void 多型run向下轉型1() {
    Foo f = new Foo();
    Alpha x = new Alpha();
    x = (Alpha) f;
    f = x;
}

public static void 多型run向下轉型2() {
    Beta f = new Beta();
    Foo x = (Detla) f;
}

public static void 多型run向上轉型1() {
    Beta f = new Beta();
    Foo x = (Alpha) f;
}
```

```java
public class Poly04P {

    public int var1 = 100;

    protected void showVar1() {
        System.out.println("Poly04P 定義的 showVar1() var1 = " + var1);
    }
}

class Poly04C extends Poly04P {

    public double var1 = 1111.111;

    public void showVar1() {
        System.out.println("Poly04C 定義的 showVar1() var1 = " + var1);
    }
}
```

```
public static void 覆蓋與多型() {
    Poly04P obj1 = new Poly04P();
    obj1.showVar1();
    System.out.println("obj1.var1 =" + obj1.var1);
    System.out.println("=================================");
    Poly04C obj2 = new Poly04C();
    obj2.showVar1();
    System.out.println("obj2.var1 =" + obj2.var1);
    System.out.println("=================================");
    Poly04P obj3 = new Poly04C();
    obj3.showVar1();
    System.out.println("obj3.var1 =" + obj3.var1);
}
```

```
Poly04P 定義的 showVar1() var1 = 100
obj1.var1 =100
=================================
Poly04C 定義的 showVar1() var1 = 1111.111
obj2.var1 =1111.111
=================================
Poly04C 定義的 showVar1() var1 = 1111.111
obj3.var1 =100
```

多型的父類別可以改成 “抽象的類別” 或 “介面”

```
public interface Poly05P {

    void 習慣();
}

class Poly05C implements Poly05P {

    public void 習慣() {
        System.out.println("抽煙");
    }
}
class Poly06C implements Poly05P {

    public void 習慣() {
        System.out.println("吃檳榔");
    }
}
class Poly07C implements Poly05P {

    public void 習慣() {
        System.out.println("喝酒");
    }
}
```

```
public static void 多型_父親是介面() {
    Poly05P a;
    a = new Poly05C();
    a.習慣(); //抽煙
    a = new Poly06C();
    a.習慣(); //吃檳榔
    a = new Poly07C();
    a.習慣(); //喝酒
}
```

```java
public class OverDemo5 {

    public int x;

    public OverDemo5(int x) {
        this.x = x;
    }

    public int sum(OverDemo5 obj) {
        return this.x + obj.x;
    }

    public boolean compareto(OverDemo5 obj) {
        if (this.x == obj.x) {
            return true;
        }
        return false;
    }

    public int compare(OverDemo5 obj1, OverDemo5 obj2) {
        if (obj1.x == obj2.x) {
            return 0;
        }
        if (obj1.x > obj2.x) {
            return 1;
        }
        return -1;
    }
}
```

```java
public static void 主人與客人1() {

    OverDemo5 a = new OverDemo5(10);
    OverDemo5 b = new OverDemo5(20);
    System.out.println("a.x+b.x=" + (a.x + b.x));
    System.out.println("a.sum(b)=" + a.sum(b));
    System.out.println("b.sum(a)=" + b.sum(a));

    System.out.println("a.compareto(b)=" + a.compareto(b));
    System.out.println("b.compareto(a)=" + b.compareto(a));

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

```
a.x+b.x=30
a.sum(b)=30
b.sum(a)=30
a.compareto(b)=false
b.compareto(a)=false
a.compare(a,b)=-1
b.compare(a,b)=-1
```

```java
public class OverDemo6 {

    public int x;

    public OverDemo6(int x) {
        this.x = x;
    }
    public int sum(OverDemo6 obj) {
        if (obj != null) {
            return this.x + obj.x;
        }
        return 0;
    }
    public boolean compareto(OverDemo6 obj) {
        if (obj != null) {
            if (this.x == obj.x) {
                return true;
            }
        }
        return false;
    }
    public int compare(OverDemo6 obj1, OverDemo6 obj2) {
        if (obj1 != null && obj2 != null) {
            if (obj1.x == obj2.x) {
                return 0;
            }
            if (obj1.x > obj2.x) {
                return 1;
            }
        }
        return -1;
    }
    public boolean equals(Object obj) {
        if ((obj != null && obj instanceof OverDemo6)) {
            if ((x == ((OverDemo6) obj).x)) {
                return true;
            }
        }
        return false;
    }
}
```

```java
public static void 主人與客人2() {

    OverDemo6 a = new OverDemo6(10);
    OverDemo6 b = new OverDemo6(10);
    Integer c = new Integer(10);

    System.out.println("a.sum(b)=" + a.sum(b));
    System.out.println("a.compareto(b)=" + a.compareto(b));
    System.out.println("a.compare(a,b)=" + a.compare(a, b));
    System.out.println("a.equals(b)=" + a.equals(b));
    //不同型態，會當
    System.out.println("a.equals(c)=" + a.equals(c));
}
```

```
a.sum(b)=20
a.compareto(b)=true
a.compare(a,b)=0
a.equals(b)=true
a.equals(c)=false
```

Object 的三個重要方法

```
public class OverDemo7 {

    public int x;

    public OverDemo7(int x) {
        this.x = x;
    }

    // String , 八個包裝類別  有覆寫
    public boolean equals(Object obj) {
        if ((obj != null && obj instanceof OverDemo7)) {
            if ((x == ((OverDemo7) obj).x)) {
                return true;
            }
        }
        return false;
    }

    //StringBuffer ,StringBuilder , 八個包裝類別 , File , Date...  有覆寫
    public String toString() {
        return String.valueOf(x);
    }

    //String , 八個包裝類別  有覆寫
    public int hashCode() {
        return x;
    }
}
```

```
public static void equals的覆寫() {
    //自訂類別
    OverDemo7 u1 = new OverDemo7(10);
    OverDemo7 u2 = new OverDemo7(10);
    System.out.println("u1.equals(u2)=" + u1.equals(u2));
    System.out.println("u2.equals(u1)=" + u2.equals(u1));
    //String
    String s1 = "abc";
    String s2 = "abc";
    System.out.println("s1.equals(s2)=" + s1.equals(s2));
    //八個包裝類別
    Integer i1 = new Integer(10);
    Integer i2 = new Integer(10);
    System.out.println("i1.equals(i2)=" + i1.equals(i2));
}
```

```
u1.equals(u2)=true
u2.equals(u1)=true
s1.equals(s2)=true
i1.equals(i2)=true
```

```java
public static void toString的覆寫() {
    //自訂類別
    OverDemo7 u = new OverDemo7(10);
    System.out.println("u.x=" + u.x);
    System.out.println("u.toString()=" + u.toString());
    System.out.println("u=" + u);
    //StringBuffer , StringBuilder
    StringBuffer sb = new StringBuffer("abc");
    System.out.println("sb=" + sb);
    //八個包裝類別
    Integer i = new Integer("123");
    System.out.println("i=" + i);
    //Date
    Date d = new Date();
    System.out.println("d=" + d);
    //File
    File f = new File("c:\\file.txt");
    System.out.println("f=" + f);
}
```

```
u.x=10
u.toString()=10
u=10
sb=abc
i=123
d=Tue Jun 27 10:23:47 CST 2017
f=c:\file.txt
```

```java
public static void hashCode的覆寫() {
    //自訂類別
    OverDemo7 u1 = new OverDemo7(10);
    OverDemo7 u2 = new OverDemo7(10);
    System.out.println("u1.hashcode()=" + u1.hashCode());
    System.out.println("u2.hashcode()=" + u2.hashCode());
    //String
    String s1 = "abc";
    String s2 = "abc";
    System.out.println("s1.hashCode()=" + s1.hashCode());
    System.out.println("s2.hashCode()=" + s2.hashCode());
    //八個包裝類別
    Integer i1 = new Integer(10);
    Integer i2 = new Integer(10);
    System.out.println("i1.hashCode()=" + i1.hashCode());
    System.out.println("i2.hashCode()=" + i2.hashCode());
}
```

```
u1.hashcode()=10
u2.hashcode()=10
s1.hashCode()=96354
s2.hashCode()=96354
i1.hashCode()=10
i2.hashCode()=10
```