副程式 (結構化)

1. 主程式(主呼叫者) 與 副程式 (被呼叫者)
2. 方法(無回傳值 void) 或 函數(有回傳值)
3. 實際參數 與 型式參數
4. 傳值➔傳 原始資料型態➔byte · short · int · long · float · double · boolean · char (值)
   傳址➔傳 類別 (位址) 或 陣列 (位址)
5. 回傳值➔回傳 原始資料型態➔byte · short · int · long · float · double · boolean · char (值)
   回傳址➔回傳 類別 (位址) 或 陣列 (位址)

public void abc( ){
 	
}
	public 值 abc( ){
	
	}
		public 址 abc( ){
		
		}

public void abc(值){
 	
}
	public 值 abc(值){
	
	}
		public 址 abc(值){
		
		}

public void abc(址){
 	
}
	public 值 abc(址){
	
	}
		public 址 abc(址){
		
		}

```java
//主呼叫者 ( 主程式 )
public static void 兩數相加() {
    int x;
    int y;
    int sum;
    x = 4;
    y = 5;
    //呼叫方法--------
    sumxy(x, y);  //實際參數
    //呼叫函數--------
    sum = sumab(x, y); //實際參數
}
//被呼叫者  ( 副程式 ) 方法
public static void sumxy(int x, int y) { // 形式參數
    int sum;
    sum = x + y;
    System.out.println("sum = " + sum);
}
//被呼叫者  ( 副程式 ) 函數
public static int sumab(int a, int b) { // 形式參數
    int sum;
    sum = a + b;
    return sum;
}
```

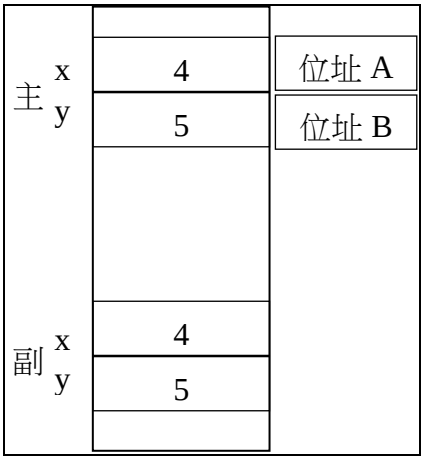
傳值

```
public static void 加1_傳值() {

    int x = 4;
    int y = 5;

    System.out.println("傳值加1前 x=" + x + " y=" + y);
    傳值(x, y);
    System.out.println("傳值加1後 x=" + x + " y=" + y);
}

public static void 傳值(int x, int y) {
    x = x + 1;
    y = y + 1;
}
```



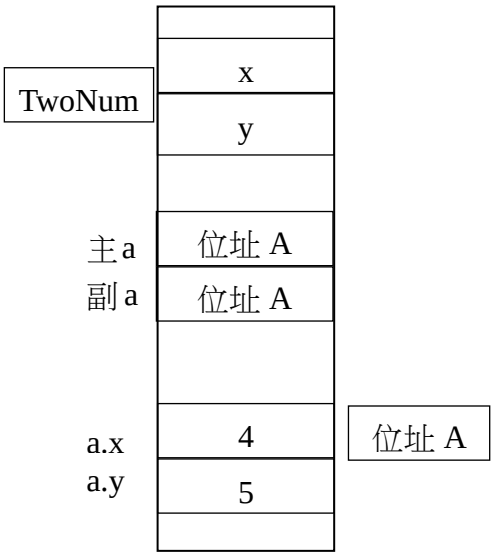
傳址

```
public class TwoNum {

    public int x;
    public int y;
}
```

```
public static void 加1_傳址() {
    TwoNum a = new TwoNum();
    a.x = 4;
    a.y = 5;
    System.out.println("傳址加1前 a.x=" + a.x + " a.y=" + a.y);
    傳址(a);
    System.out.println("傳址加1後 a.x=" + a.x + " a.y=" + a.y);
}

public static void 傳址(TwoNum a) {
    a.x = a.x + 1;
    a.y = a.y + 1;
}
```



傳址-其他變形

```
public static void 傳址_變數() {
    TwoNum a = new TwoNum();  //a.x  a.y
    a.x = 4;
    a.y = 5;
    System.out.println("a.x=" + a.x + " a.y=" + a.y);  //4 ,5

    TwoNum b;
    b = a;
    //b = new TwoNum();
    System.out.println("b.x=" + b.x + " b.y=" + b.y);  //4 ,5
}
```

```
public static void 傳址_參數() {
    TwoNum a = new TwoNum();
    a.x = 4;
    a.y = 5;
    System.out.println("a.x=" + a.x + " a.y=" + a.y); //4,5
    接收址(a);
}

public static void 接收址(TwoNum b) {
    //  b = new TwoNum();
    System.out.println("b.x=" + b.x + " b.y=" + b.y); //4,5
}
```

```
public static void 傳址_不同的變數() {
    TwoNum a = new TwoNum();  //a.x  a.y
    a.x = 4;
    a.y = 5;
    System.out.println("a.x=" + a.x + " a.y=" + a.y);  //4 ,5

    TwoNum b;
    b = a;

    b.x = 10;
    b.y = 20;
    System.out.println("a.x=" + a.x + " a.y=" + a.y); //10,20
}
```

傳址➔多型應用

```java
public interface Call7 {

    void 叫();
}
abstract class Animal7 {

    public abstract void 叫();
}


class Dog7 implements Call7 {

    public void 叫() {
        System.out.println("狗叫");
    }
}
class Cat7 implements Call7 {

    public void 叫() {
        System.out.println("貓叫");
    }
}
class Bird7 implements Call7 {

    public void 叫() {
        System.out.println("鳥叫");
    }
}
```

```java
public static void 傳址_多型1() {

    Dog7 狗 = new Dog7();
    Cat7 貓 = new Cat7();
    Bird7 鳥 = new Bird7();
    接收址1(狗);
    接收址1(貓);
    接收址1(鳥);
}

public static void 接收址1(Dog7 a) {
    a.叫();
}

public static void 接收址1(Cat7 a) {
    a.叫();
}

public static void 接收址1(Bird7 a) {
    a.叫();
}
```

```java
public static void 傳址_多型2() {
    Call7 狗 = new Dog7();
    Call7 貓 = new Cat7();
    Call7 鳥 = new Bird7();
    接收址2(狗);
    接收址2(貓);
    接收址2(鳥);
}

public static void 接收址2(Call7 a) {
    a.叫();
}
```

```
狗叫
貓叫
鳥叫
```

回傳址➔多型應用

```java
public static void 回傳1() {
    //回傳 值
    int sum;
    int x = 4;
    int y = 5;
    sum = 回傳1_值(x, y);
    System.out.println("sum=" + sum);
    //回傳 址
    TwoNum a = null;
    a = 回傳1_址();
    System.out.println("a.x=" + a.x + " a.y=" + a.y);
}

public static int 回傳1_值(int x, int y) {
    return x + y;
}

public static TwoNum 回傳1_址() {
    TwoNum a = new TwoNum();
    a.x = 10;
    a.y = 20;
    return a;
}
```

```
sum=9
a.x=10 a.y=20
```

```java
public class GetCall7 {

    public static Call7 getDog7Instance() {
        return new Dog7();
    }

    public static Call7 getCat7Instance() {
        return new Cat7();
    }

    public static Call7 getBird7Instance() {
        return new Bird7();
    }
}
```

```java
public static void 回傳2_址() {
    Call7 ani;
    ani = GetCall7.getDog7Instance();
    ani.叫();
    ani = GetCall7.getCat7Instance();
    ani.叫();
    ani = GetCall7.getBird7Instance();
    ani.叫();
}
```

```
狗叫
貓叫
鳥叫
```

工廠函數➔方法是 static，並且有回傳值

1. 工廠模式是為了解耦：把對象的創建和使用的過程分開。就是 Class A 想調用 Class B ，那麼 A 只是調用 B 的方法，而至於 B 的實例化，就交給工廠類。

解耦通俗地說就是兩個東西原來互相影響，現在讓他們獨立發展

2. 其次，工廠模式可以降低代碼重復。如果創建對象 B 的過程都很復雜，需要一定的代碼量，而且很多地方都要用到，那麼就會有很多的重復代碼。我們可以將這些創建對象 B 的代碼放到工廠裡統一管理。既減少了重復代碼，也方便以後對 B 的創建過程的修改維護。(當然，也可以把這些創建過程的代碼放到類的建構函數裡，同樣可以降低重復率，而且建構函數本身的作用也是初始化對象。不過，這樣也會導致建構函數過於復雜，做的事太多，不符合 java 的設計原則。)
3. 由於創建過程都由工廠統一管理，所以發生業務邏輯變化，不需要找到所有需要創建 B 的地方去逐個修正，只需要在工廠裡修改即可，降低維護成本。同理，想把所有調用 B 的地方改成 B 的子類 B1，只需要在對應生產 B 的工廠中或者工廠的方法中修改其生產的對象為 B1 即可，而不需要找到所有的 new B ( ) 改為 new B1()。
4. 另外，因為工廠管理了對象的創建邏輯，使用者並不需要知道具體的創建過程，只管使用即可，減少了使用者因為創建邏輯導致的錯誤。
5. 舉個例子：一個資料庫工廠：可以回傳一個資料庫實例，可以是 mysql，oracle 等。這個工廠就可以把數據庫連接需要的用户名，地址，密碼等封裝好，直接回傳對應的資料庫對象就好。不需要調用者自己初始化，減少了寫錯密碼等等這些錯誤。調用者只負責使用，不需要管怎麼去創建、初始化對象。
6. 還有，如果一個類有多個建構方法 (建構的重寫)，我們也可以將它抽出來，放到工廠中，一個建構方法對應一個工廠方法並命名一個友好的名字，這樣我們就不再只是根據參數的不同來判斷，而是可以根據工廠的方法名來直觀判斷將要創建的對象的特點。這對於使用者來說，體驗比較好。
7. 工廠模式適用的一些場景

<1>.對象的創建過程/實例化準備工作很復雜，需要初始化很多參數、查詢數據庫等。

<2>.類本身有好多子類，這些類的創建過程在業務中容易發生改變，或者對類的調用容易發生改變。

```java
public static void 工廠函數() {
    //1
    A001 a = new A001();
    //2
    A001 b = A001.getA001Instance();
    //3
    A001 c = ClassManager.getA001Instance();
    //4
    ClassManager q = new ClassManager();
    A001 d = q.createA001Instance();
}
```

```java
public class A001 {

    public A001() {

    }

    public static A001 getA001Instance() {
        return new A001();
    }
}

class B001 {

}

class C001 {

}
```

```java
class ClassManager {

    public static A001 getA001Instance() {
        return new A001();
    }

    public A001 createA001Instance() {
        return new A001();
    }

    public static B001 getB001Instance() {
        return new B001();
    }

    public B001 createB001Instance() {
        return new B001();
    }

    public static C001 getC001Instance() {
        return new C001();
    }

    public C001 createC001Instance() {
        return new C001();
    }
}
```

陣列參數的傳遞

```
public static void 傳陣列_一維() {
    int[] ar1 = {1, 2, 3, 4, 5};
    接收陣列_一維(ar1);
    for (int x : ar1) {
        System.out.print(x + " ");
    }
    System.out.println();
}

public static void 接收陣列_一維(int[] ar1) {
    for (int i = 0; i < ar1.length; i++) {
        ar1[i]++;
    }
}
```

```
2 3 4 5 6
```

```
public static void 傳陣列_二維() {
    int[][] ar1 = {{1, 2, 3}, {4, 5, 6}};
    接收陣列_二維(ar1);
    for (int[] y : ar1) {
        for (int x : y) {
            System.out.print(x + " ");
        }
        System.out.println();
    }

}

public static void 接收陣列_二維(int[][] ar1) {
    for (int i = 0; i < ar1.length; i++) {
        for (int j = 0; j < ar1[i].length; j++) {
            ar1[i][j]++;
        }
    }
}
```

```
2 3 4
5 6 7
```

```
public static void 接收陣列_一維() {
    int[] ar1;
    ar1 = 回傳陣列_一維();
    for (int x : ar1) {
        System.out.print(x + " ");
    }
}

public static int[] 回傳陣列_一維() {
    int[] ar1 = {1, 2, 3, 4, 5};
    return ar1;
}
```

```
1 2 3 4 5
```

```
public static void 接收陣列_二維() {
    int[][] ar1;
    ar1 = 回傳陣列_二維();
    for (int[] y : ar1) {
        for (int x : y) {
            System.out.print(x + " ");
        }
        System.out.println();
    }
}

public static int[][] 回傳陣列_二維() {
    int[][] ar1 = {{1, 2, 3}, {4, 5, 6}};
    return ar1;
}
```

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

字串參數的傳遞

```java
public static void 傳字串1() {
    StringBuilder str = new StringBuilder("abc");
    System.out.println("呼叫前 str=" + str);
    接收字串1(str);
    System.out.println("呼叫後 str=" + str);
}

public static void 接收字串1(StringBuilder str) {
    str.append("xyz");
}
```

```
呼叫前 str=abc
呼叫後 str=abcxyz
```

```java
public static void 傳字串2() {
    String str = "a";
    接收字串2(str);
    str = "null";
    接收字串2(str);
    str = "A";
    接收字串2(str);
    str = null;
    接收字串2(str);
}

public static void 接收字串2(String str) {
    switch (str) {
        case "a":
            System.out.println("a");
            break;
        case "null":
            System.out.println("null");
            break;
        case "A":
            System.out.println("A");
        default:
            System.out.println("other");
    }
}
```

```
str=a
str=null
str=A
```

```
public static void 傳值與傳參考() {
    Bike myBMW = new Bike();
    int tmpMile = 68;
    int tmpCount = 2;
    myBMW.setProperties(tmpMile, tmpCount);
    Wheel myWL = new Wheel();
    myWL.thickness = 2.72F;
    myWL.size = 40.5F;
    myBMW.setWheel(myWL);
    myBMW.showProperties();
    System.out.println("========================");
    //以下是為檢驗所傳入的參數，其內容是否可能在函數內被改變
    System.out.println("tmpMile = " + tmpMile);
    System.out.println("tmpCount = " + tmpCount);
    System.out.println("myWL.size =" + myWL.size);
}
```

```
里程數=68 輪子數量=2
輪子 厚度=2.72 直徑=55.5
========================
tmpMile = 68
tmpCount = 2
myWL.size =55.5
```

```
public class Wheel {

    public float thickness; //厚度
    public float size;      //直徑
}

class Bike {

    public int mileage;     //里程數
    public int wheelCount;  //輪子數量
    public Wheel theWheel;  //輪子

    public void setWheel(Wheel WL) {
        WL.size = WL.size + 15;
        theWheel = WL;   //theWheel 和 WL 參考同一份物件實體
    }

    public void setProperties(int mile, int count) {
        mileage = mile;
        wheelCount = count;
        mile = mile + 99;  //此敘述句僅為測試而作
        count = count + 99;  //此敘述句僅為測試而作
    }

    public void showProperties() {
        System.out.println("里程數=" + mileage + " 輪子數量=" + wheelCount);
        System.out.println("輪子 厚度=" + theWheel.thickness + " 直徑=" + theWheel.size);
    }
}
```

作業

```
public static void 練習1() { //傳 3 個變數
  String name="賴玉珊";
  String address="新北市";
  String tel="29018251";
}

public static void 接收1() {

}

public static void 練習2() { //傳 陣列 data

}

public static void 接收2() {

}

public static void 練習3() { //傳 物件 Data

}

public static void 接收3() {

}
```

```
public static void 練習4() {

}

public static String 回傳4() { //回傳 變數

}

public static void 練習5() {

}

public static String[] 回傳5() { //回傳 陣列 data

}

public static void 練習6() {

}

public static Data 回傳6() { //回傳 物件 Data

}
```