



J01-10

使用 **Method** 和 **Method Overloading**

曾瑞君 (Jim_Tzeng)

學習目標

1. 宣告Method, 使用參數並回傳結果
2. 宣告 static methods & variables
3. 建立 overloaded method
4. 了解Java如何傳遞(pass)參數/變數



Edited by Ruei-Jiun Tzeng



1/4

宣告 **Method**, 使用參數並回傳結果

Edited by Ruei-Jiun Tzeng

如何宣告Method


- 語法

```
[modifiers] return_type method_identifier ( [arguments] ) {  
    method_code_block  
}
```

- 範例

```
class Shirt {  
    public void display () {  
        ...  
    } // end of display method  
}
```

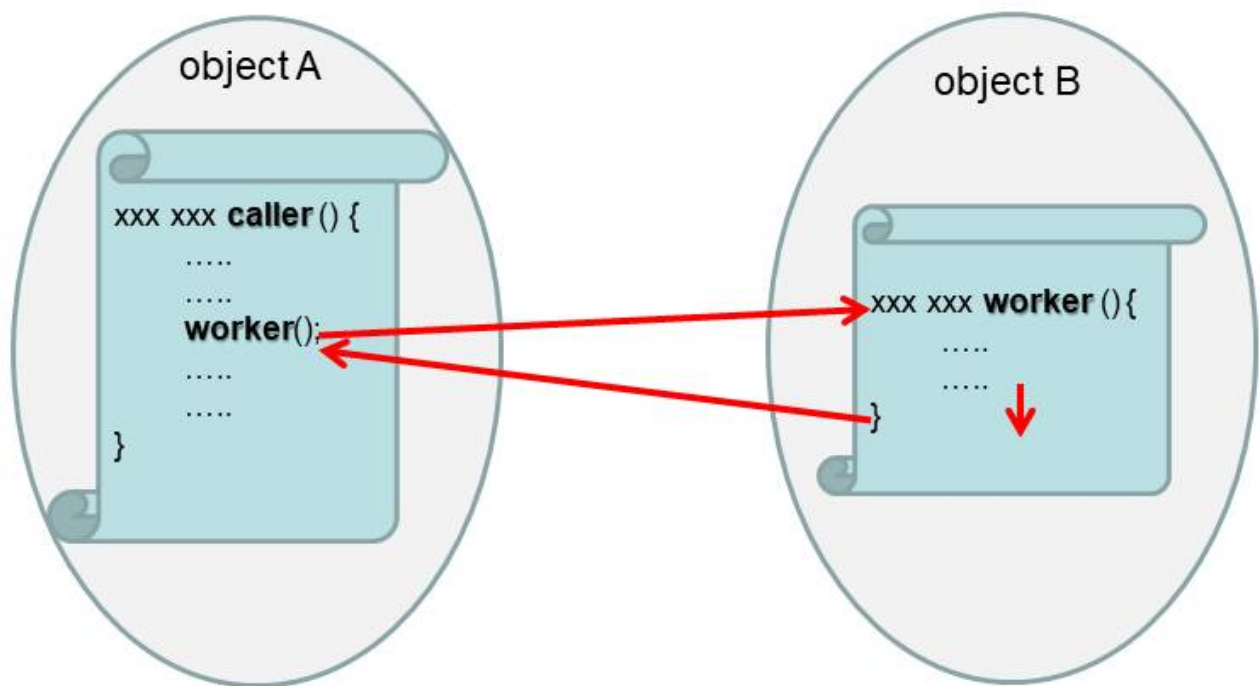
Edited by Ruei-Jiun Tzeng



其它class如何呼叫method?

```
public class ShirtTest {  
    public static void main (String args[]) {  
        Shirt myShirt = new Shirt();  
  
        myShirt.display();  
    }  
}
```

Caller(呼叫者) & Worker(被呼叫者)



Method 帶參數

Worker

```
public void setFloor( int targetFloor ) {  
    while (currentFloor != targetFloor) {  
        if (currentFloor < targetFloor) {  
            up();  
        } else {  
            down();  
        }  
    }  
}
```

Caller

```
Elevator elevator = new Elevator();  
elevator.setFloor( 4 ); // Take elevator to the 4 floor
```

Method 有回傳

Worker

```
public class Elevator {  
    private boolean doorOpen=false;  
    .....  
    public boolean isDoorOpen() {  
        return doorOpen ;  
    }  
}
```

Caller

```
Elevator elevator = new Elevator();  
boolean isOpen = elevator.isDoorOpen() // Is door open?
```


呼叫自家 method

```
public class Elevator {  
    private boolean doorOpen=false;  
    public boolean isDoorOpen() {  
        return this.doorOpen ;  
    }  
    public void openDoor() {  
        // Check if door already open  
        if ( ! this.isDoorOpen() ) {  
            // ...  
        }  
    }  
}
```

可加可不加，表示“自己”

Edited by Ruei-Jiun Tzeng



使用method的好處

- 一個method通常具備獨立功能或邏輯性，讓程式可讀性高，並易於維護
- 增加程式「可重複使用性 (re-usable)」
- 讓不同物件之間可互動 (caller & worker)



2/4

宣告 static methods & variables

Edited by Ruei-Jiun Tzeng

沒有 static 時的情況

- 先來看一段範例：

```
public class Circle {  
  
    private double radius;  
    final double PI = 3.1415926;  
  
    public void setRadius(double r) {  
        this.radius = r;  
    }  
  
    public double getArea() {  
        return this.radius * this.radius * PI;  
    }  
}
```

Edited by Ruei-Jiun Tzeng

沒有 static 時的情況

- 回想一下，我們曾說過 class 是藍圖，用來製作出不同的 object。所以我們來建立 Circle object：

```
public static void main(String[] args) {  
  
    // 建立一個 Circle:c1物件, 其半徑=1, 並取得其面積  
    Circle c1 = new Circle();  
    c1.setRadius(1);  
    System.out.println(c1.getArea());  
  
    // 建立一個 Circle:c2物件, 其半徑=10, 並取得其面積  
    Circle c2 = new Circle();  
    c2.setRadius(10);  
    System.out.println(c2.getArea());  
  
}
```

Edited by Ruei-Jiun Tzeng

沒有 static 時的問題

現在我們看到2個問題：

1. 每個被產生出來的 Circle object，都會有一個資料成員 PI，都會佔據記憶體空間；該資料成員PI是一個常數，每個 Circle物件都有一份，是不是有點浪費？不能只有1份然後大家共享嗎？
2. 再者，圓面積計算是大家都知道的公式，每次都必須 new Circle() 物件嗎？是不是有點浪費記憶空間？可以不使用物件嗎？可以比照Math.random()或是Math.round()，使用 Circle.getArea(r) 直接輸入半徑，求得面積嗎？



使用 static 來解決問題

- 為了解決問題1【PI變數無法共享】，我們使用 static field；
- 為了解決問題2【需要以物件 new Circle() 計算面積】，我們使用 static method。
- 所以class改版如后：

解決方案

原 field 直接加上 static

原 method 不能直接加上 static。
static method 內只能使用 static 的 fields 或 methods

另外建立 static method，需要的半徑，則由方法傳入

```
public class Circle {  
    private double radius;  
    static final double PI = 3.1415926;  
  
    public void setRadius(double r) {  
        this.radius = r;  
    }  
  
    double getArea() {  
        return this.radius * this.radius * PI;  
    }  
  
    //此為公式，結果只和輸入參數有關  
    static double areaFormula(double r) {  
        return r * r * Math.PI;  
    }  
}
```

Edited by Ruei-Jiun Tzeng

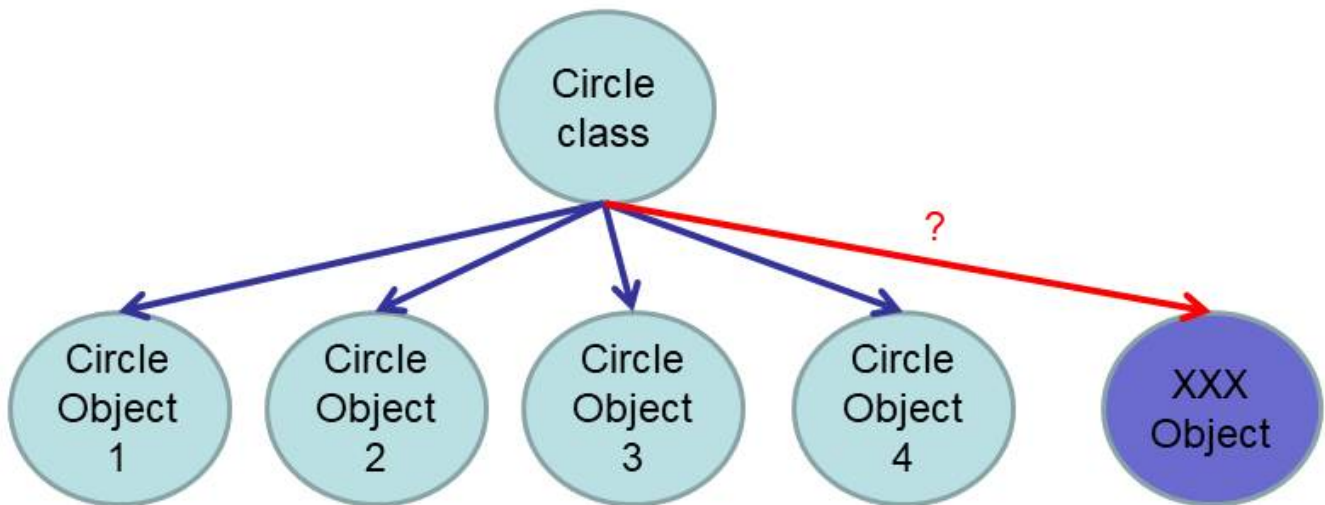
使用解決方案

- 現在，所有 Circle object 都共用同一個 PI 變數；若只是想計算圓面積，我們可以改由 Circle class 提供的【公式】來計算，不需要再new出一個新 object：

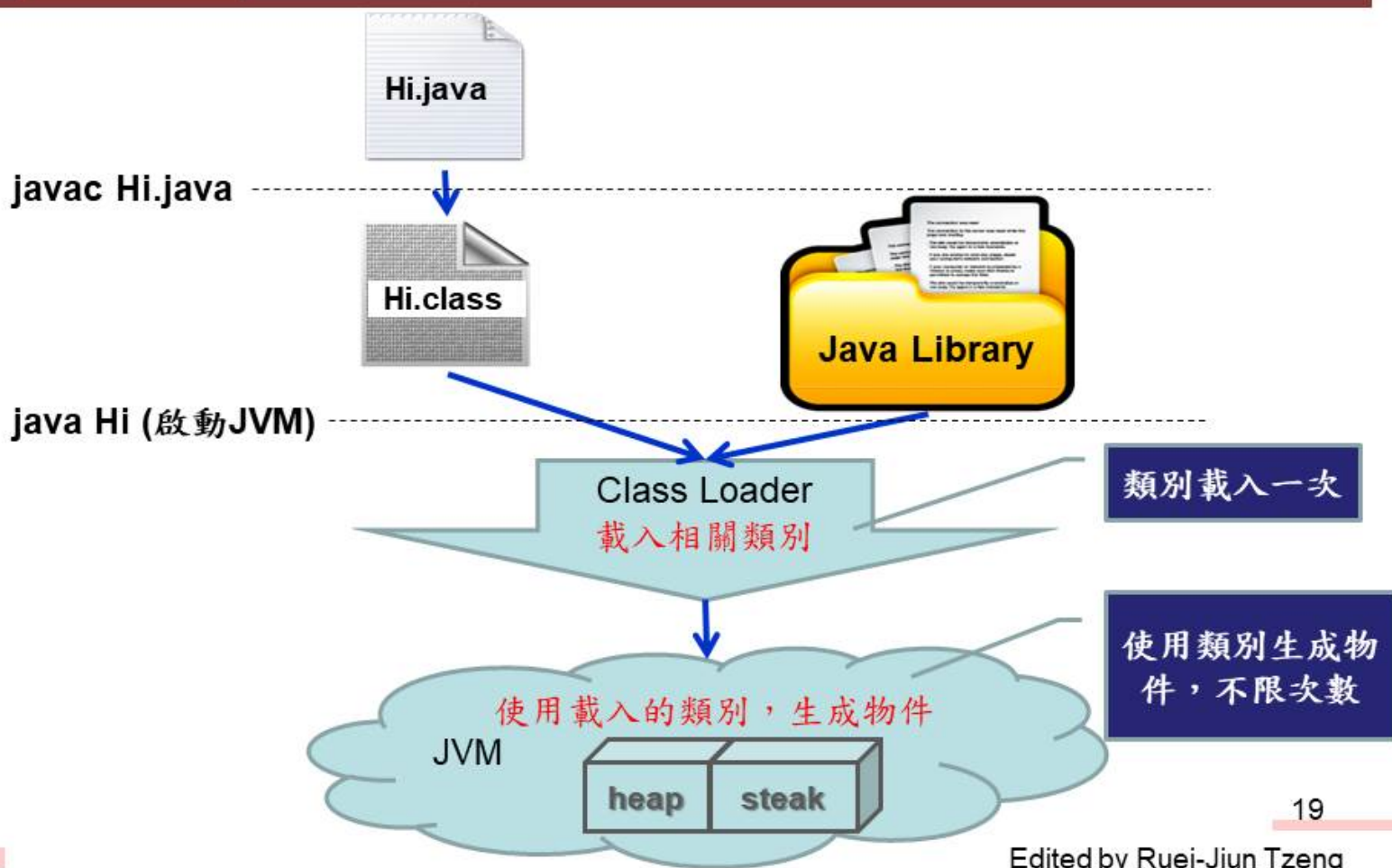
```
public static void main(String[] args) {  
  
    // 印出圓周率PI  
    System.out.println( Circle.PI );  
  
    // 已知半徑 100, 但不想藉由產生 Circle 物件求得面積  
    System.out.println( Circle.areaFormula(100) );  
  
}
```


哪裡適合放共享的資訊？

- 若一個類別生成了4個物件，哪裡適合放共享資訊？
- 其他物件，也可以共用?!



Java 執行示意





static 宣告的意義

因為類別內的欄位和方法的宣告都可以使用static修飾詞，所以可以由「有無static宣告」將「欄位和方法」分成兩類：

- 未使用static宣告：

這類型的類別的欄位和方法使用時都必須先產生物件，再使用物件參考(遙控器)去呼叫，因此稱「物件成員 (Object Member)」

- 使用static宣告：

因為只用類別名稱就可以呼叫方法和欄位，因此稱「類別成員 (Class Member)」

static 宣告的注意事項

- 因為類別先載入記憶體才能以之生成物件；所以物件成員可以使用類別成員。
- 反之，若類別成員使用自己的物件成員，將陷入「類別尚未載入記憶體，卻必須使用該類別生成的物件的矛盾」，故編譯器將 compile 失敗：

non-static variable XXX cannot be referenced from a static context

non-static method XXX() cannot be referenced from a static context

- 因為每個類別只載入1次，所以類別成員在JVM裡是唯一存在。



- 成員 => 欄位或方法
- 有類別(先)，才有物件(後)。
- 依時間順序，後者可以使用前者。
 - 物件成員可以使用物件成員。
 - 物件成員可以使用類別成員。
 - 類別成員可以使用類別成員。
 - 類別成員 **不**可以使用物件成員。



static 宣告的意義

雖然語法上也可以透過物件 (object) 取得 static 成員，但不建議，因為容易混淆：

Circle c1 = new Circle();	建議使用：	不建議使用：
	Circle.PI	c1.PI
	Circle.areaFormula(1)	c1.areaFormula(1)

static 宣告的使用時機

- static 資料與函式的作用之一，是提供**公用類別函式**，例如將數學常用常數或計算公式，以static宣告並撰寫，之後我們可以透過類別名稱來管理與取用這些函式，例如像Math.exp()、Math.log()、Math.sin()等等的static常數或函式等，事實上，像PI這個常數，在Math.PI就有定義，我們可以直接呼叫使用。

公式：結果只和輸入參數有關 (無關所在class/object)

- 若method的內容不涉及物件狀態，亦即不含物件成員，也可以考慮使用static宣告。

讓程式碼更完整...

```
public class Circle {  
  
    private double radius;  
  
    void setRadius(double r) {  
        this.radius = r;  
    }  
  
    double getArea() {  
        return areaFormula(this.radius); //讓原方法呼叫公式  
    }  
  
    //此為公式，結果只和輸入參數有關  
    static double areaFormula(double r) {  
        return r * r * Math.PI;  
    }  
}
```

non-static method,
可以呼叫 static method

最常見的static方法

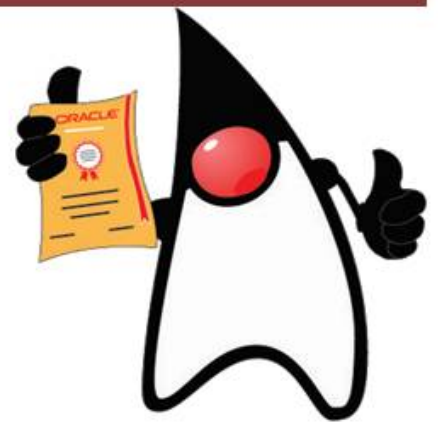
```
public static void main(String[] args) {  
    ...  
}
```

- 作為Java SE程式執行的進入點，有特殊目的，不涉及物件狀態。
- 過去將 Shirt、ShirtTest class 分開，是為避免混淆，其實是可以將該方法直接放在Shirt class中。程式進入點，放哪裡都可以。



Java 內 static 範例

- Math class:
 - 指數
 - 對數
 - 三角函數
 - 隨機浮點數
 - 數學常數，如 (Math.PI)
- System class:
 - 取得環境變數
 - 取得標準輸入輸出串列
 - 結束程式：System.exit()



3/4

建立 Overloaded Method

Method Signature

- 先前談過宣告method的語法：

```
[modifiers] return_type method_identifier ( [arguments] ) {...}
```

- 其中，方法名稱(method_identifier)，和參數(arguments)，合併稱呼「**Method Signature**」
- 「Signature」單字解釋為「簽名」。以中文來看，簽名用來識別身分，因此class內真正用來識別method的，不是方法名稱，而是「**方法名稱+參數**」。
- 參數只要「**數量**」、「**型態**」、「**順序**」有一個不同，就算不同。「參數名稱」無須考量。



Method Overloading

- 定義：
Method 之間，其 name / identifier 相同，但 signature 不同。
- 使用時機：
method 功能相近，只是傳入參數型態、數量不同。



Method Overloading 範例

```
public class Calculator {  
    public static int sum ( int numberOne, int numberTwo){  
        return numberOne + numberTwo;  
    }  
    public static float sum ( float numberOne, int numberTwo) {  
        return numberOne + numberTwo;  
    }  
    public static float sum ( int numberOne, float numberTwo) {  
        return numberOne + numberTwo;  
    }  
}
```

Edited by Ruei-Jiun Tzeng

Method Overloading 範例

```
public class CalculatorTester {  
    public static void main(String [] args) {  
        int totalOne = Calculator.sum(2,5);  
        System.out.println(totalOne);  
        float totalTwo = Calculator.sum(12.9f, 12);  
        System.out.println(totalTwo);  
        float totalThree = Calculator.sum(12, 12.9f);  
        System.out.println(totalThree);  
    }  
}
```


Java 內的 Method Overloading

- System.out.println(?)

Method	Use
void println()	Terminates the current line by writing the line separator string
void println(boolean x)	Prints a Boolean value and then terminates the line
void println(char x)	Prints a character and then terminates the line
void println(char[] x)	Prints an array of characters and then terminates the line
void println(double x)	Prints a double and then terminates the line
void println(float x)	Prints a float and then terminates the line
void println(int x)	Prints an int and then terminates the line
void println(long x)	Prints a long and then terminates the line
void println(Object x)	Prints an object and then terminates the line
void println(String x)	Prints a string and then terminates the line

Edited by Ruei-Jiun Tzeng



4/4

了解Java如何傳遞(pass)參數/變數

Pass By ?

- 以下程式執行結果如何？

y = 10
Shirt size = 4
Shirt size = 4

```
public class PassByValueTest {  
    public static void main(String[] args) {  
        testPrimitive();  
        testReference();  
    }  
    private static void testPrimitive() {  
        int x = 10;  
        int y = x;  
        x = 5;  
        System.out.println("y = " + y);  
    }  
    private static void testReference() {  
        Shirt x = new Shirt();  
        x.size = 5;  
        Shirt y = x;  
        x.size = 4;  
        System.out.println("Shirt size = " + y.size);  
        modifyShirt(x);  
        System.out.println("Shirt size = " + x.size);  
    }  
    private static void modifyShirt(Shirt s) {  
        s = new Shirt();  
        s.size = 3;  
    }  
}
```

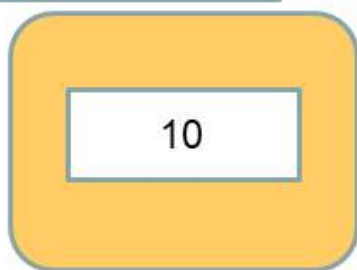
Pass by Value

- Java在2種情況時需要傳遞(pass)參數/變數：
 - 由指定運算子「=」右側，將值(value)傳遞給左側變數。
 - 透過方法宣告的參數，將值由呼叫者(caller)方法傳遞進入工作者(worker)方法中。
- 傳遞(pass)參數/變數時，基本型別和參考型別都是「複製」變數本身的值(value)做傳遞。但值的定義不同：
 - 參考型別：將複製物件參考(遙控器)後進行傳遞，所以複製前後雖遙控器不同，但指向同一物件。
 - 基本型別：沒有遙控器的概念，因此直接複製值，如同影印機複製原稿後產生副本，兩者各自獨立。
- 因為都會經過複製(Copy)，所以也可稱為Pass by Copy

Pass-By-Value(Copy) – 基本型別

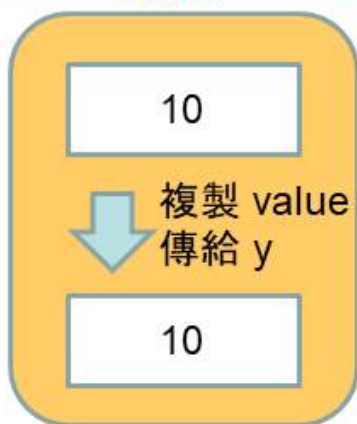
```
int x = 10;  
int y = x;
```

x



stack

x



y

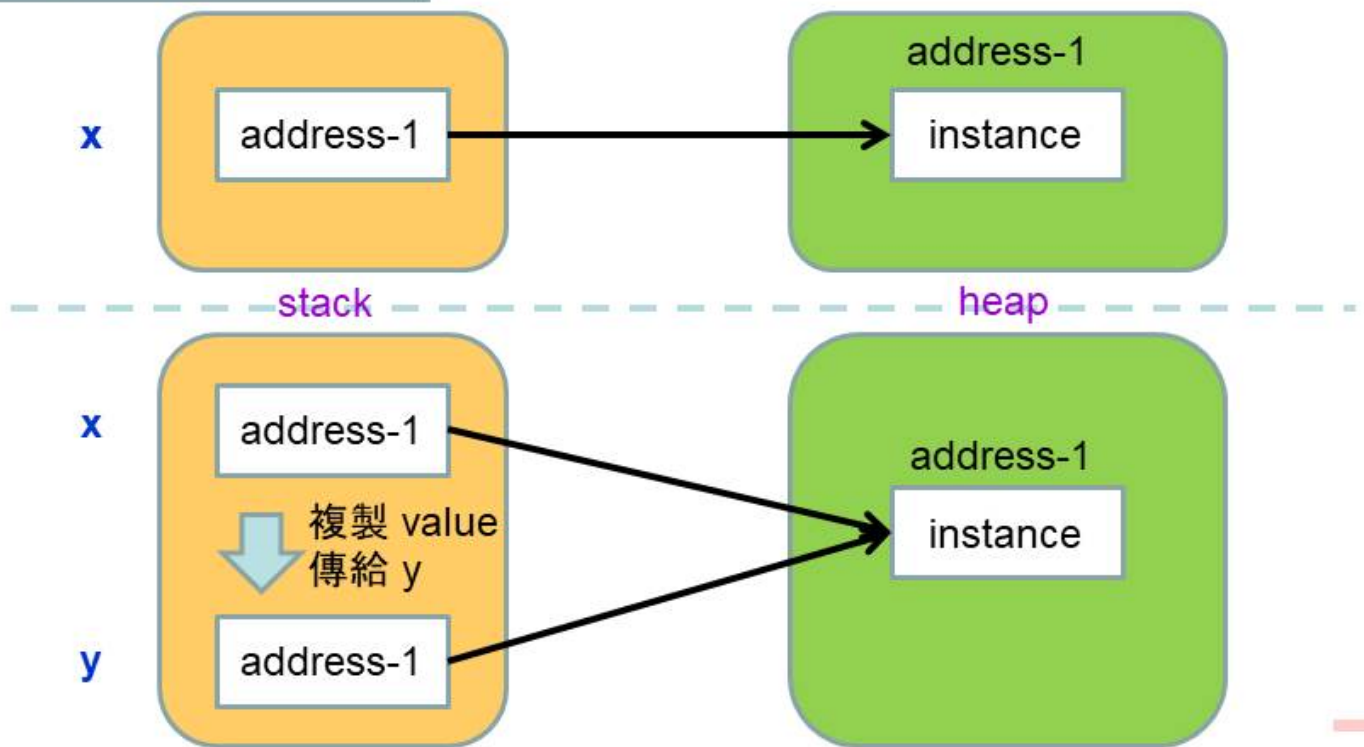
複製 value
傳給 y

heap



Pass-By-Value(Copy) – 參考型別

```
Shirt x = new Shirt();  
Shirt y = x;
```



Objects Passed as Parameters

- 檢視以下範例：

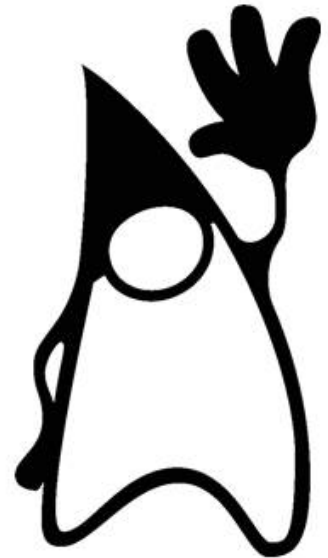
```
private static void modifyShirt(Shirt s) {  
    s = new Shirt();  
    s.size = 3;  
}
```

- 原先傳入worker方法的物件參考仍指向原物件。因為在方法內又將遙控器指向另一個新建構的物件實例，故方法內外的物件參考已經參照不同的物件實例。在方法內修改的實例，不影響方法外的物件實例。



END ~~

Thank you!!



Edited by Ruei-Jiun Tzeng

