目錄

---

1. JAVA 口角(!)

1. JAVA 是簡單的
2. 所見即物件

2. 物件 類別(!)

- 物件
  1. attribute(屬性) 或稱為 Characteristics(特徵)
  2. behavior(行為) 或稱為 Operation(操作)
- 類別

1. Data Member(資料成員): Virable(變數)
2. Method Member(方法成員): Method(方法)

- Class(類別) 是 Object(物件) 的 type(資料型態)，Object(物件)是 Class(類別) 產生的 instance(資料實體)

```
// 類別
public class Pen{
  int price;
  public static void main(String[] args){
    Pen p = new Pen(); // p 即是 Object Reference Virables (物件參考變數)
  }
}
```

- 宣告
  - `Pen p`
- 物件實體化
  - 產生物件真正的記憶體空間: `new Pen();`
- 物件初始化
  - `p = new Pen();`
- `Object Reference Variables` (物件參考變數) 儲存 `memory address` (記憶體位址) 的變數
- `基本資料型別變數` 儲存 `value` (值)
  - 相同的記憶體空間 = 改 A 就改 B，資料在一個記憶體空間所以會互相影響

```
Pen p1 = new Pen();
Pen p2 = p1;
```

```
public static void main (String[] args) {
  int amount;
  amount = 10;
  Pen myPen = new Pen();
```

```
    Pen yourPen = new Pen();
}
```

Stack Memory	
amount	Pass by value
myPen	Pass by value
yourPen	Pass by value

Class	Heap Memory	用法	
myPen	brand	myPen.brand	Pass by reference
myPen	price	myPen.price	Pass by reference

Class	Heap Memory	用法	
yourPen	brand	yourPen.brand	Pass by reference
yourPen	price	yourPen.price	Pass by reference

- myPen.brand 的做法又稱作物件參考變數

3. Stack Heap global Memory

- Stack
  - 生命週期 可預測 ，即可交由系統管理(系統會自動產生與回收)
- Heap
  - 生命週期 不可預測 ，動態產生的資料(使用者需自己回收)
  - Heap 中的資料如果沒有正常的回收，將會逐步成長到將記憶體消耗殆盡(但 java 有 Garbage Collection 機制所以不須特別處理)
- global
  - 存 static 變數與全域變數

4. 物件導向的三大特性(!)

- **封裝** (Encapsulation)

    - `private`、`default`、`protected`、`public`
    - 提升資料存取安全性
- **繼承** (Inheritance)
    - 子類別`繼承父類別成員`，並`可修改`或`新增`自有成員
    - 繼承的同時可擴充所需，表現自身特質
- **多型** (Polymorphism)
    - 用同樣方式引用不同類別物件

```
public class Test {
  void say() {
    System.out.println("Hello");
  }

  public static void main(String[] args) {
    Object[] obj = new Object[2];
    // 利用多型的特性使程式碼簡單易用
    for (int i = 0; i < obj.length; i++) {
      obj[i] = new Test();
      ((Test) obj[i]).say();
    }
  }
}
```

5. 陣列(!)

- 陣列是由`一群相同資料型態的變數所組成`的一種資料結構
- 需用 new 分配空間 `int x = new int[3]`，`new 時需指定長度且不可更改`
- 是種 Reference 資料型態，指定運算是傳遞`記憶體位置`
- 元素`有初始值`

5-1. 一維陣列 二維陣列

- 一維 `元素`數

    - 可以的做法
        - int[] xx = new int[4];
- 二維 `列(row)` 數
    - 可以的做法
        - int[][] xx = new int[4][];
          xx[0] = new int [3];

```
// 一維陣列
Pen p[] = new Pen[3];
p[0] = new Pen();
p[1] = new Pen();
p[2] = new Pen();
// 一維陣列
int[] xx = new int[4];
// 二維陣列
int[][] xx = new int[4][];
xx[0] = new int [3];
```

5-2. 複製陣列(!)

- 陣列複製方法 `Arrays.copyOf` (`陣列A` , `陣列A.length`)

```
int[] 陣列B = Arrays.copyOf(陣列A, 陣列A.length);
```

- 一維陣列才能用
- 複製後的陣列互不影響
- JDK 6 後出現

5-3. 陣列排序(!)

- 陣列排序方法

```
Arrays.sort(欲排序的陣列名稱)
```

```
int[] intArray3 = {100, 200, 300, 50};
Arrays.sort(intArray3);
for (int item : intArray3) {
```

```
    System.out.print(item + "\t"); // 50 100 200 300
}
```

5-4. 陣列搜尋(!)

- 陣列搜尋前需排序，使用二分法
- 沒搜尋到指定值，返回負值

```
Arrays.sort(陣列名稱)
Arrays.binarySearch(陣列名稱, 欲搜索的值)
```

```
int[] intArray3 = {100, 200, 300, 50};
int i1 = Arrays.binarySearch(intArray3, 50); // 0
int i2 = Arrays.binarySearch(intArray3, 150); // -3
```

6. String 類別

- 不可變的 (immutable) 字串
- **每個字串都是不同記憶體位置**
  - String 一旦宣告後,即不能在原所在記憶體位置改變字串內容
- 存在 String Pool (字串池)
- new String() 存在 Heap Memory

6-1. equals() 比對字串相等(!)

- 使用 `字串A.equals(要比對的值)` 判斷字串是否相等
  - `==` 在 string 中比較的是記憶體空間，非內容

```
String str1 = new String("abc"); // 新記憶體空間
String str2 = "abc"; // 新記憶體空間
String str3 = new String("abc");
String str4 = "abc"; // 因字串相同，同 str2 一個記憶體空間

System.out.println(str1 == str2); // true
System.out.println(str2 == str4); // true
System.out.println(str1 == str3); // false
```

```
System.out.println(str1.equals(str2)); // true
System.out.println(str1.equals(str3)); // true
```

6-2. compareTo() 比對字串是否相等

- 依照 ASCII 比對大小，多用於排序
- 若回傳值 `=0` ，表示兩個字串相等
- 若回傳值 `>0` ，表示左邊字串大於右邊字串
- 若回傳值 `<0` ，表示左邊字串小於右邊字串

```
String s1 = "Hello", s2 = "Hello";
System.out.println(s1.compareTo(s4)); // 40
```

6-3. charAt() 取得字串字元

```
String s1 = "Hello";
System.out.println(s1.charAt(4)); // o
```

6-4. length() 取得字串長度(!)

- 返回字串長度，含空白

```
String s1 = "Hello";
System.out.println(s1.length()); // 5
```

6-5. trim() 去除頭尾空白(!)

```
String s4 = " ", s5 = " Hello ";
System.out.println(s4.trim()); // 空值
System.out.println(s5.trim()); // Hello
```

6-6. isEmpty() 比對字串是否為空(!)

- 字串長度為 0 返回 true，否則 false，長度含空白

```
String s4 = " ";
System.out.println(s4.isEmpty()); // false
System.out.println(s4.trim().isEmpty()); // true
```

6-7. substring(int beginIndex) 擷取字串

- 擷取從 `開始` 索引值的字元 `至結尾` 字元的字串
- `由 0 開始`

```
String s1 = "Hello";
System.out.println(s1.substring(1)); // ello
```

6-8. substring(int beginIndex, int endIndex) 擷取開始至結束前的字串

- 擷取從 `開始` 索引值的字元 `至結束` 索引值 `之間` 的字串
- `由 0 開始`

```
String s1 = "Hello";
System.out.println(s1.substring(1, 4)); // ell
```

7. Varargs 不固定參數個數

- Varargs 意思指 `不固定參數個數` ，也可稱為 `可變參數個數`
- 用 `...` 宣告， `void method(int...arr){}`
- 需放在最後用
- 一個方法不能有兩個 Varargs

```
public class AddInt {
  public int varArgTest(int a, int... c) {
    int sum = 0;
    for (int i = 0; i < c.length; i++) {
      sum += c[i];
    }

    return sum;
  }
```

```
  public String varArgTest(String... c) {
    String str = "";
    for (String item : c) {
      str += item;
    }

    return str;
  }

  public static void main(String[] args) {
    AddInt add = new AddInt();
    int sum1 = add.varArgTest(5, 1, 2);
    String sum3 = add.varArgTest("Hello", "World");

    System.out.println("sum1=" + sum1); // 3
    System.out.println("sum3=" + sum3); // HelloWorld
  }
}
```

8. 封裝三步曲

`1. private` → `2. public getXXX` → `3. public setXXX`

	the Same Class	the Same Package	SubClass	Universe	說明	類別	實體變數	方法	建構子
public (公共)	√	√	√	√	所有類別皆能存取	√	√	√	√
protected (保護)	√	√	√		**同套件下類別或所有其子類別都可存取**		√	√	√
default (預設)	√	√			同套件下的類別皆可存取	√	√	√	√

	the Same Class	the Same Package	SubClass	Universe	說明	類別	實體變數	方法	建構子
private (私有)	√				只有該類別內部才可存取		√	√	√