第七章類別與物件

資訊科技系
林偉川

7-1 類別的定義

- 類別的定義語法

```
修飾字 class 類別名稱{ //類別的標頭
        // 屬性宣告
        // 建構子定義
        // 方法定義
}
```

7-1 類別的定義

- 類別的修飾字
  - **public**－公開類別，宣告成此種類別可以被任何類別所使用。
  - **無修飾字**(default)類別，此種類別僅能被同一套件(package)內的類別使用。
  - **final**－此種類別不可被繼承。
  - **abstract**－此為抽象類別的修飾字。此種類別至少擁有一個抽象方法，不可用於直接建立物件。

3

7-1 類別的定義

- 類別的命名規則
  - 類別名稱的第一個字母應該大寫，如：Car、Person
  - 若是由兩個字以上組成時，每個單字的第一個字母為大寫，如：CarClass。
  - 名稱中包括縮寫時，所有縮寫字母為大寫，如：URL
  - 類別的主體內為屬性、建構子及方法

```
class EmptyClass{ }
```

4

7-2 建立物件

- 宣告物件➔只有參照無實體

  類別名稱　物件名稱;

- 建立物件實體

  物件名稱 = new 類別名稱();

- 宣告並建立物件實體➔參照+實體

  類別名稱　物件名稱 = new 類別名稱();

- **物件名稱**必須是合法的**識別字**。
- 其命名規則為第一個字的字母皆為小寫。
- 若為複合字時，其餘字的第一個字母為大寫。
- 若為縮寫字母時，皆為小寫。

5

類別和物件的預設建構子

```
class test1 {
  public static void main(String args[]) {
     EmpClass obj1=new EmpClass();
     test1 obj2=new test1();
     System.out.println(obj1); // 參照+實體
     System.out.println(obj2); // 參照+實體
   }
}
class EmpClass { }
```

6

7-3 屬性

- 屬性的宣告語法

```
修飾字  資料型態  屬性名稱;
```

- 屬性存取權限相關的修飾字
  - **public**－公用級，所有類別皆可使用此級屬性。
  - **protected**－保護級，同一套件內及其子類別可以直接使用。
  - 無修飾字－預設級(default)或稱套件級，同一套件內的類別可以直接使用。
  - **private**－私有級，屬性所在的類別才能使用。

7

7-3 屬性

- 使用final修飾字自訂常數

```
public final int wheels = 4;
```

- static修飾者為**類別屬性**
- 使用**點**(dot)運算子指出取用的屬性

```
物件名稱.屬性名稱
```

- **屬性**只要宣告就會**自動初始化**，設上預設值
  整體0，浮點數0.0f或0.0d，布林值為false，
  字串，參照變數null

8

類別和物件的預設建構子

```
class aa {
   public static void main(String argv[]) {
      Object a[] = new Object[3]; // 陣列參照
      for (int i=0; i<3; i++) System.out.println(a[i]); //null
      a[0] = new Object(); // 參照+實體
      aa  c=new aa(); // 參照+實體
     System.out.println(a[0]); //印出參照+實體
      Object b = new Object(); // 參照+實體
      System.out.println(b); //印出參照+實體
   }
}
```

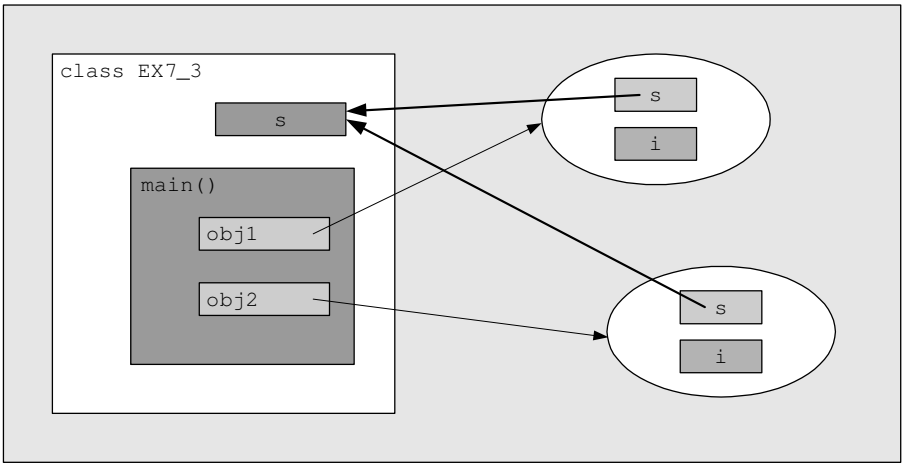
9

類別成員和物件成員的區別

```
Class EX7_3 { //SCJP 1,124,131,下22,90
  int i;
 static int s=100;
 public static void main(String args[]) {
   EX7_3 obj1=new EX7_3();
   EX7_3 obj2=new EX7_3();
   System.out.println(++obj1.i+" "+(++obj1.s)); // 1 101
   System.out.println(++obj2.i+" "+(++obj2.s)); // 1 102
   System.out.println(EX7_3.s); // 102
   System.out.println(s); // 102
 }}
```

10

類別成員和物件成員的區別



11

物件的鏈結

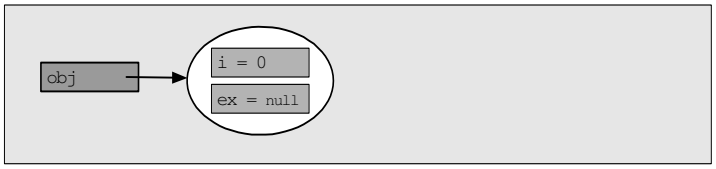
```
Class EX7_4 {
  int i;
 EX7_4 ex; //參照
 public static void main(String args[]) {
   EX7_4 obj=new EX7_4(); //參照+實體
   obj.ex=new EX7_4(); //參照+實體
   obj.ex.i=1;
   obj.ex.ex=new EX7_4();  obj.ex.ex.i=2;
  System.out.println(i+" "+obj); // 0
System.out.println(obj.ex.i+" "+obj.ex.ex.ex); // 1 null
 }
```

12

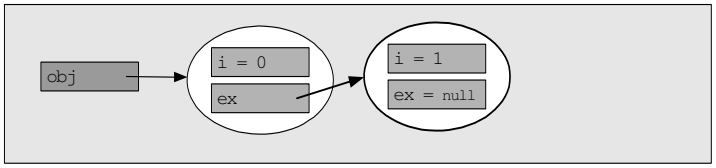
7-3 屬性

物件的鏈結

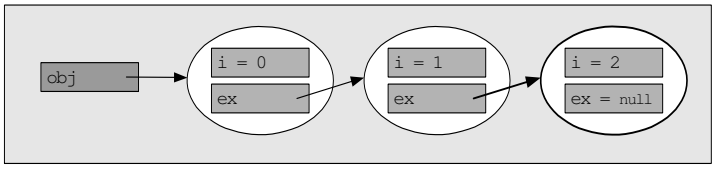
```
EX7_4 obj = new EX7_4();
```




```
obj.ex = new EX7_4();
obj.ex.i = 1;
```




```
obj.ex.ex = new EX7_4();
obj.ex.ex.i = 2;
```



7-4 方法

定義方法(Method)

```
<修飾字>  資料型別  方法名稱(<形式參數列>) {
        // 方法中的敘述
        <return 回傳值;>
}
```

- **修飾字**：**public**、**private**、**protected**、**default**、**static**
- **形式參數列**和**return敘述**為非必要。
- **資料型別**和**方法名稱**為必要(int, double, boolean, char, byte, long, void[沒回傳值] …)。
- **方法的資料型別必須和回傳值相同**。

14

- 方法的命名規則
  - 一般方法名稱的前面部份會是一個動詞，動詞應全為小寫。
  - 動詞後的字，第一個字母為大寫。
  - 若方法是為了設定或取得private屬性，則分別使用「set屬性名稱」和「get屬性名稱」。

15

- 形式參數式中，以逗號分隔各個形式參數

```
public int getArea(int w, int h) {
        return w*h;
}
```

- static修飾方法時，該方法為類別方法(slice 8)
- 抽象方法沒有實作，無方法主體

```
abstract <修飾字> 資料型別 方法名稱(<形式參數列>);
```

16

7-4-1 方法的呼叫

- 呼叫方法的語法

物件名稱.方法名稱(<參數列>)

- **參數列**的工作是把參數傳給方法。
- **參數列**為選擇性的功能，視方法定義的形式參數列而定。

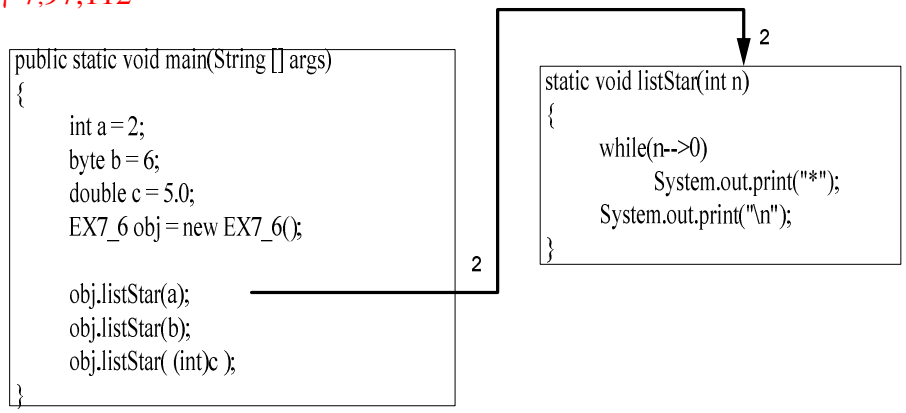
7-4-1 方法的呼叫

- 呼叫方法時的程式走向listTree為類別方法//scjp100

```
public static void main(String [] args)
{
     System.out.println("We love");
     listTree();
     System.out.println("and");
      listTree();
     System.out.println("love us!");
}

static void listTree()
{
     ...
}
```

7-4-1 方法的呼叫

參數只傳遞數值 ➔ 傳值呼叫值不變//SCJP101下7,97,112



19

參數物件的傳遞 ➔ 參照傳遞值會變

```
Class EX7_7 {
  public static void main(String args[]) {//SCJP 19
   int arr[]={9,4,5,1,6,8};
   show(arr);  sort(arr); show(arr);
}
static void sort(int[] a) { ..}
static void show(int[] a) { ..}
```

20

7-4-2 方法的型別與回傳值

- 方法的型別就是回傳值的型別

```
修飾字 資料型別 方法名稱(形式參數列) { //SCJP 下26
        // 方法中的敘述
        return 回傳值;
}
```

- 使用void宣告的方法，返回不回傳任何值

```
return;
```


7-4-2 方法的型別與回傳值

- 傳值和傳參照
  - 基本型別(boolean、byte、short、int、long、char、float及double)是以傳值的方式回傳。
  - 自訂型別(陣列或物件)是以傳參照的方式回傳資料。

7-4-3 變數領域

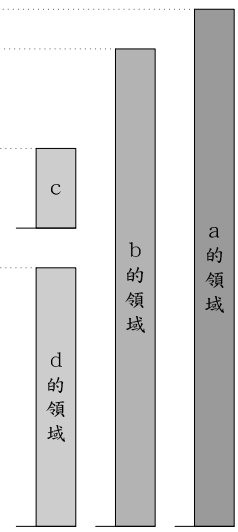
- 大括弧是變數領域的藩籬
  - 若一個變數於某個區塊內宣告，則**區塊內**從宣告點以下為變數的領域。
  - **區塊外**不為變數的領域。

```
{
        int j=1;
}
j++;            //錯誤，無法識別變數j
```


7-4-3 變數領域(SCJP 100)

- 範例7_11：

```
static void methodName(int a)
{
        int b = 2;

        System.out.println("a:"+a);
        System.out.println("b:"+b);

        for(int c = 0; c<10; c++)
        {
                if(c==3)
                        System.out.println("c:"+c);
        }

        int d = 4;

        if(d==4)
        {
                System.out.println("d:"+d);
                int e = 5;
                System.out.println("e:"+e);
        }

        {
                int f = 6;
                System.out.println("f:"+f);
        }
}
```



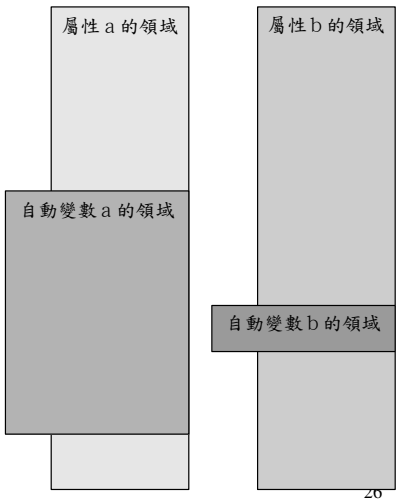
7-4-3 變數領域

- **屬性**也稱為**欄位變數**。
- 方法內的變數稱為**自動變數**或**區域變數**。
- **自動變數**和**欄位變數**同名是合法的情況，不過兩者若同名，則欄位變數會因為被遮蔽而無法在方法內被「看到」。
- 欲在方法內看到同名的屬性時，可以使用this 關鍵字，指出使用的是物件本身的屬性。

this.屬性名稱

7-4-3 變數領域

```
class EX7_12 {
    int a =10;

    public static void main(String [] args){
                EX7_12 obj = new EX7_12();
                obj.myMethod(1);
    }

    void myMethod(int a)  {
                ...

                {
                        {
                                int b=2;
                                ...
                        }
                        ...
                }
        }
        int b = 20;
}
```

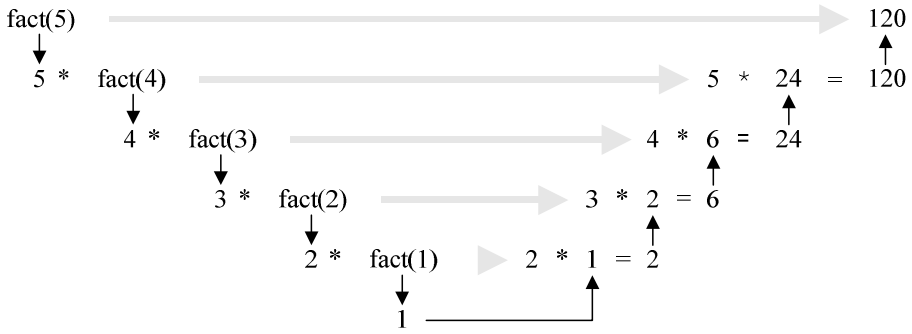


7-4-4 遞迴

- 巢狀的方法呼叫時，**後呼叫者先返回**。
- 在方法定義中**呼叫方法本身**，此方法稱為**遞迴方法**（Recursive Method）。
- 遞迴有時可以解決迴圈無法解決的問題，而且可以使程式更簡潔。
- 當問題處理過程有規則可循，且有終止的條件時，應該就可以使用遞迴方法。

7-4-4 遞迴

- 範例7_15：fact()遞迴方法的計算過程



7-4-5 方法多載

- 同一類別中，若定義數個相同名稱的方法，而各方法所需的參數不同時，稱為**方法多載**(Overloading)。
- 多載方法是以傳入的**參數個數**及**參數型別**做為呼叫的判斷依據。
- 多載方法的形式**參數的個數**及**型別**相同時，為非法定義。(SCJP 15)
- **多載的目的**是因應不同的傳遞資料，讓方法更有彈性。

7-4-6 Vararg – 可變長度的參數

- Vararg的語法:

```
修飾字  資料型別  方法名稱(型別...  參數)
修飾字  資料型別  方法名稱(型別一  參數一, 型別二  參數二, 型別三...  參數三)
```

- 使用Vararg時，只能有一個參數宣告使用「...」，而且必須放在最後面。
- 使用「...」的參數是當作陣列來看待。

```
Public static void main(String arg[]) {
System.out.println(a(1)); System.out.println(a(1,2));
System.out.println(a(1,2,3)); System.out.println(a(1,2,3,4));
…}

Static int a(int… k) { int s=0; for (int i=0; i<k.length; i++) s+=k[i]; return s; }
```

使用new及**建構子**建立物件(SCJP 40)

類別名稱 物件名稱 = new 類別名稱();

呼叫建構子

7-5-1 預設的建構子(SCJP 80,91)

- 沒有定義建構子的類別時，**編譯器**會自動設給一個沒有形式參數的建構子。
- 預設的建構子是沒有內容的，不會對物件的屬性有任何影響。
- 定義類別時，若有定義任何建構子，則編譯器不會設給預設建構子。

7-5-2 定義建構子

- **建構子的名稱**一律同**類別名稱**，而且不宣告回傳資料型別(int, char, long, short, String, void)。

```
修飾字   類別名稱(形式參數列) {
        //程式敘述
}
```

- 建構子的修飾字可以為public、protected、private或無修飾字。
- 建構子修飾字不使用static和abstract。

33

7-5-2 定義建構子(SCJP 75)

- 建構子也可以多載

```
class SomeClass {
        public SomeClass() {
                //不使用參數建立物件
        }
        public SomeClass(int x) {
                //使用參數建立物件
        }
        public SomeClass(float x) {
                //使用參數建立物件
        }
        public SomeClass(int x, int y) {
                //使用參數建立物件
        }
}
```

7-5-3 建構子的呼叫

- 建立物件時呼叫建構子

```
類別名稱  物件名稱 = new 類別名稱(形式參數列);
```

  - 方法可以和建構子同名。有回傳型別者為方法；反之，**無回傳型別者為建構子**。

7-5-4 使用this()

- 建構子多載時，建構子呼叫其它建構子的限制：
  - 必須使用**this()** 取代**類別名稱()**。
  - 只能在第一個敘述呼叫。
  - 不能造成建構子遞迴。

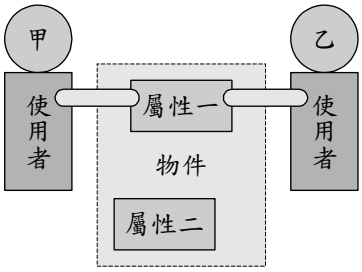
7-6 封裝

- 封裝的特點：
  - 隱藏類別實作的細節。
  - 強迫使用者透過方法操作屬性，如此可以保護資料不會被濫用。set屬性及get屬性
  - 使程式碼更容易被維護。

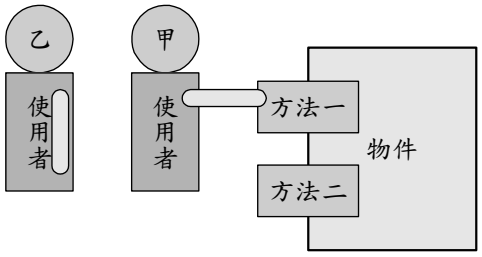
37

7-6 封裝

- 沒有封裝的物件



- 完整封裝的物件



38

7-7 靜態區塊

- 類別定義中，以static為首的區塊即為靜態區塊
- 靜態區塊的特點：
  - 和其它靜態成員一樣皆為類別所有。
  - 區塊內的敘述可以和類別方法一樣，使用方法呼叫及運算。
  - 當類別被JVM載入後，會先執行靜態區塊內的敘述，而且只執行一次。

39

7-8 內部類別

- 位於另一個類別之內的類別稱之為**內部類別**（inner classes），或稱為**巢狀類別**（nested classes）。

40

7-8-1 靜態內部類別(SCJP 82,112)

- 使用static修飾字宣告的內部類別，稱為**靜態內部類別**(static inner classes)。
- 靜態內部類別和類別內其它靜態成員一樣，都屬類別所有，所以**類別方法可以直接使用靜態內部類別建立物件**。
- 若某個類別欲使用另一個類別的靜態內部類別時，必須以點運算子(dot operator)指明靜態內部類別所屬的類別，才可使用。

41

7-8-2 非靜態內部類別(SCJP 82)

- 沒有static修飾的內部類別稱為**非靜態內部類別**(non-static inner classes)。
- **非靜態內部類別**和其它物件成員一樣，都必須建立物件才能使用。
- 以**非靜態內部類別**所建立的實體，實際上都必須與外部類別的實體結合在一起。因為沒有外部類別所建立的實體，就沒有**非靜態內部類別**的建構子。

42

7-8-3 區域內部類別

- 定義於**方法內的類別**，稱為**區域內部類別**(local inner classes)。
- **區域內部類別**不屬於外部類別的類別成員，也不是物件成員，其**性質就像區域變數**一樣，只能在定義的方法內使用，而且必須先定義才能使用。
- **區域內部類別**不使用public、protected、private和static等修飾字。
- **區域內部類別**可以取用外部方法中以final宣告的變數，不是以final宣告的變數不能取用。

43

- Enumerated Type 的語法:

```
修飾字   enum   列舉型別名稱 {
        常數名稱一, 常數名稱二, 常數名稱三 …
};
列舉型類別繼承java.lang.Enum➔compareTo()、equals()
```

- 例如:

```
public enum MyColor {
        RED, GREEN, BLUE, YELLOW
};
int wkc=MyColor.GREEN; or int wkc=1;
MyColor wkc=MyColor.RED;
```

44

7-9 列舉型別

- 列舉型別本身就是類別，只是底層做了一些手腳，因此在使用列舉型別時，直接當類別看待。
- 列舉型別可以獨立定義於一個原始檔內，或者在某個類別內定義。
- 列舉型別有個特殊方法values()可以取得所有的值，並以陣列的形式回傳。

45

7-9 列舉型別

- 列舉型別的要點：
  - 列舉型別本身是類別，繼承java.lang.Enum。
  - 列舉型別不可以定義於方法內。
  - 列舉型列的值不是int或String。
  - 列舉型列的值轉換成字串時，會轉換成和值的名稱相同的字串。列舉型列可以使用valueOf()取得對應的值。
  - 列舉型列的值不可以使用>、>=、<、<=運算子。
  - 列舉型別的值，若為相同型別，可以使用!=、==或equals()測試是否相等。
  - 列舉型別的值，若為相同型別，可以使用compareTo()來比較前後順序。

46

7-9 列舉型別

列舉型別和 switch 敘述合用時，switch 敘述的鍵值，也可以是列舉型別的變數。在使用case標籤時，不使用列舉型別名稱。

```
enum YourColor {PINK, IVORY, ORANGE, WHITE, GOLDEN}
YourColor yc = YourColor.ORANGE;
switch(yc) {
        //case YourColor.PINK:        // 錯誤
  case PINK:  System.out.println("粉紅"); break;
  case IVORY: System.out.println("象牙白"); break;
  case ORANGE: System.out.println("橘色"); break;
  default: System.out.println("其它顏色");
}
```

47