# 物件導向程式設計
# Object Oriented Programming

## • 什麼是物件導向

非物件導向

### 青椒炒牛肉

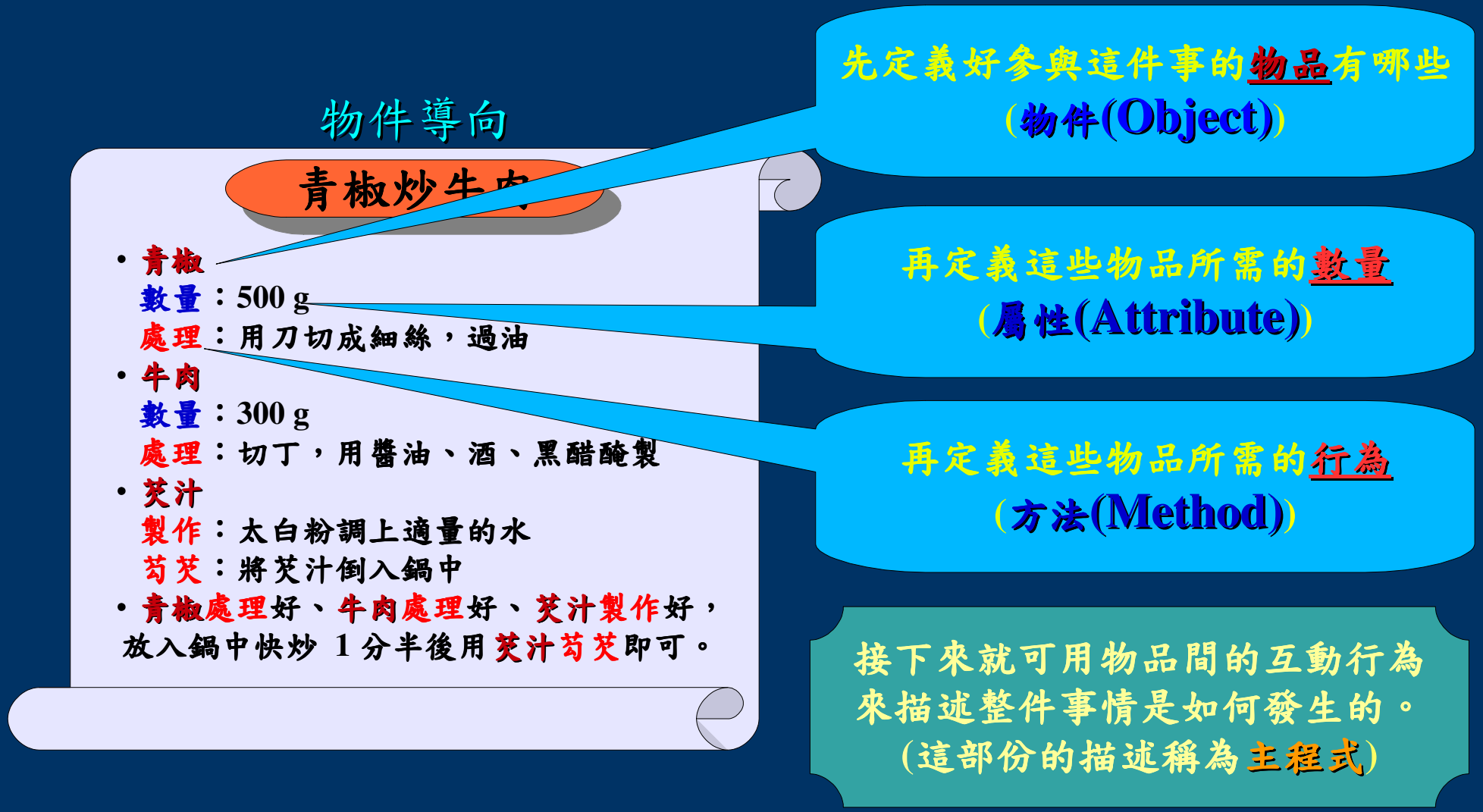
- 取出青椒 500 g，用刀切成細絲，過油
- 取出牛肉 300 g，切丁，用醬油、酒、黑醋醃製 30 分鐘
- 起油鍋、放入牛肉炒及青椒大火快炒 1 分半
- 拿出太白粉、水調在一起，這個稱為芡汁。
- 將芡汁倒入鍋中攪拌，會產生黏稠現象這叫芶芡
- 完成

物件導向

### 青椒炒牛肉

- 青椒
  數量：500 g
  處理：用刀切成細絲，過油
- 牛肉
  數量：300 g
  處理：切丁，用醬油、酒、黑醋醃製
- 芡汁
  製作：太白粉調上適量的水
  芶芡：將芡汁倒入鍋中
- 青椒處理好、牛肉處理好、芡汁製作好，放入鍋中快炒 1 分半後用芡汁芶芡即可。

# 物件導向程式設計
# Object Oriented Programming (Cont.)



# 物件導向程式設計
# Object Oriented Programming (Cont.)

- Object-oriented programming 程式設計的主要概念，是將日常生活中實際的物件應用在軟體設計裏。在物件導向程式設計中，程式是一群物件(或稱為元件， Components)所組成
- 如果把物件導向的觀念用Java語言來實作，Java的程式是一群類別(class)的集合，類別中的變數(Variables)和方法(Methods)分別表示特徵和行為動作

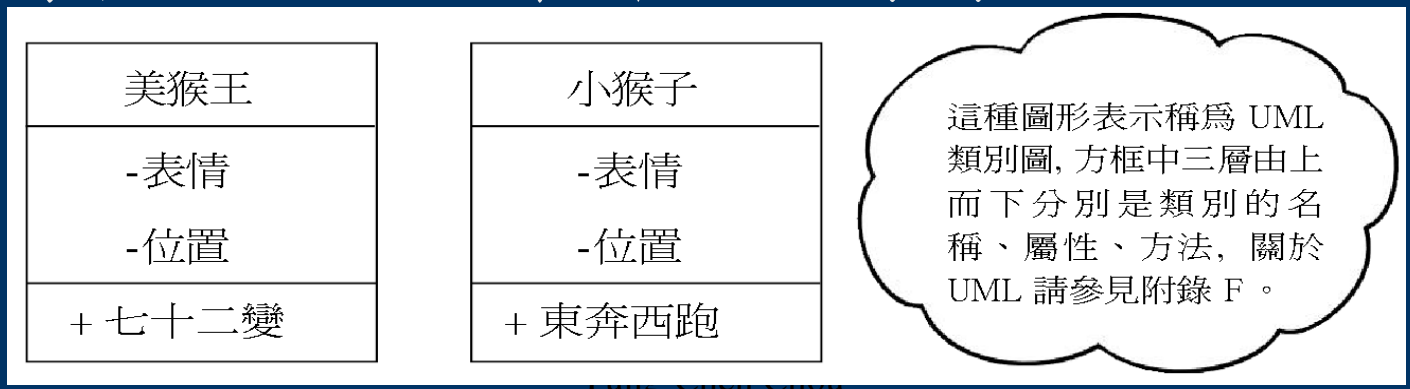
# 物件導向程式設計
# Object Oriented Programming (Cont.)

- 為什麼要用物件導向程式設計？
  - 與程式大小無關
  - 與程式難易、複雜程度無關
  - 思考方式符合日常生活中“自然的思考方式”
  - 以真實世界的情形來架構軟體物件的類別庫
  - 繼承特性使程式碼與程式觀念可以很容易的被再使用 (reuse)
  - 類別的封裝，增加軟體的穩定性
  - 多形的特性讓軟體具有高度的可修改性

# 物件導向程式設計
# Object Oriented Programming (Cont.)

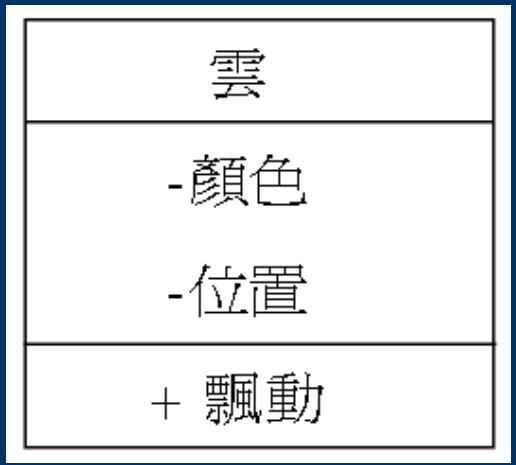
- 類別間以訊息傳遞方式溝通，讓軟體的擴充非常容易

- 舉例：一齣舞台劇『西遊記』
  - 主角是美猴王，它有多種表情、還能夠進行 72 變。除了美猴王以外，當猴子猴孫的小猴子也少不了，這些小猴子可能有 3 種表情，而且可以東奔西跑等等。



# 物件導向程式設計
# Object Oriented Programming (Cont.)

- 舞台上除了演員外, 可能還需要一些佈景或道具, 每一種佈景也有它的特性與可能的動作。舉例來說, 『西遊記』中美猴王騰雲駕霧, 舞台上可少不了雲。雲可能有不同顏色, 也可以飄來飄去, 而且同時間舞台上可能還需要很多朵雲。



# 物件導向程式設計
# Object Oriented Programming (Cont.)

- 物件導向的特性
  - 資料封裝 (encapsulation): 將物件的特徵及行為封裝在物件中，達到資訊隱藏 (information hiding ) 的目的
  - 繼承 (inheritance): 父類別的資源 ( 特徵及行為，以 Java 而言就是類別的變數及方法 ) 其子類別可以繼承使用，達到程式再使用 (reuse) 的目的
  - 多形 (polymorphism ，或稱同名異式 ): 子類別繼承父類別後可將其方法的內容依所需重寫 ( 覆蓋， override) ，也就是說相同名稱的方法，可以有不同的程式碼。

# 類別與物件
# Class and Object

- 在 Java 中, 每一種角色就稱為一種**類別** (Class), 類別可以用來描述某種角色的**屬性**與**行為**；實際在程式執行時演出這種角色的演員或道具就稱為此類別的**物件** (Object ), 物件就依循類別所賦予的屬性與行為, 按照程式流程所描繪的劇本演出
- 構思好各個角色後, 接著就是劇本了。哪個演員應該在甚麼時候上場、做甚麼動作、站在哪個位置、講甚麼話, 這些就是劇本應該描述清楚的內容(即主程式)

# 類別與物件
# Class and Object (Cont.)

- Java 程式所討論的**流程控制**正是為了安排 Java 程式執行時的順序, 也就是 Java 程式的劇本
- 哪個物件應該在甚麼時候登場、各個物件在某個時候應該做甚麼動作？這些就是流程控制所要掌控的事情。有了流程控制, 所有的物件就照著劇本演出, 執行程式
- 對Java來說, 它的主要舞台就是 **main()方法**。在第 2 章中曾經提到過, 每個 Java 程式都必須要有一個 main() 方法, 它也是 Java 程式執行的起點

# 定義類別與建立物件

- 以 Java 的角度來說, 擬定角色也就是規劃出程式中要有哪些**類別**, 並且實際描述這些類別的屬性與行為
- 在Java中, 要描繪類別, 需使用 class 敘述, 其語法如下：

```
1  class 類別名稱{
2      敘述 1
3      ...
4      敘述 n
5  }
```

這個區塊稱為類別本體 (Class Body) 用來描述此類別的屬性與行為

# 定義類別與建立物件 (Cont.)

- 宣告類別之後, 我們就可以用它來建立物件。回顧第 2 章用基本型別建立變數時, 我們會先宣告變數名稱, 再設定一個初始值給它
- 而使用類別建立物件, 就好比是用一個新的資料型別 (類別) 來建立一個類別變數 (物件), 比較特別的是, 我們必須用 **new 運算子**來建立物件：

```
1  class Car{
2      // 汽車的屬性
3      // 汽車的行爲
4  }
5
6  public class BuildCar1{
7      public static void main(String[] argv){
8          Car oldcar, newcar; //宣告物件
9
10         oldcar = new Car(); //建立物件
11         newcar = new Car(); //建立物件
12     }
13 }
```

# 定義類別與建立物件 (Cont.)

- 程式中第 8 行先宣告了 2 個 Car 物件, 此部份和用基本型別宣告變數沒什麼不同
- 不過當程式宣告基本型別的變數, Java 就會配置變數的記憶體空間；但是宣告物件時, Java 僅是建立了指向物件的參照 (程式中的 oldcar、newcar), 並不會實際配置物件的記憶體空間, 必須再如第 10、11 行使用 new運算子, 才會建立實際的物件

```
1  class Car{
2      // 汽車的屬性
3      // 汽車的行爲
4  }
5
6  public class BuildCar1{
7      public static void main(String[] argv){
8          Car oldcar, newcar; //宣告物件
9
10         oldcar = new Car(); //建立物件
11         newcar = new Car(); //建立物件
12     }
13 }
```

# 定義類別與建立物件 (Cont.)

- 在 new 運算子後面, 呼叫了與類別同名的方法, 此方法稱為建構方法(Constructor), 在此先不探究其內容, 留待下一章再介紹
- 呼叫建構方法時, Java 會配置物件的記憶體空間, 並傳回該配置空間的位址。我們也可以將宣告和建立物件的敘述放在一起：

```
Car oldcar = new Car();
```

# 類別的屬性—成員變數

- 在類別中，需使用**成員變數** (Member Variable)來描述類別的屬性，在 Java語言中又稱其為類別的**欄位** (Field)。成員變數的宣告方式，和前幾章所用的一般變數差不多，例如我們的汽車類別要有記錄**載油量**、**耗油率**，可寫成：

```
1  class Car{
2      double gas; // 載油量
3      double eff; // 耗油率
4  }
```

- 有了成員變數時，即可在程式中存取物件的成員變數，存取成員變數的語法為：

物件.成員變數

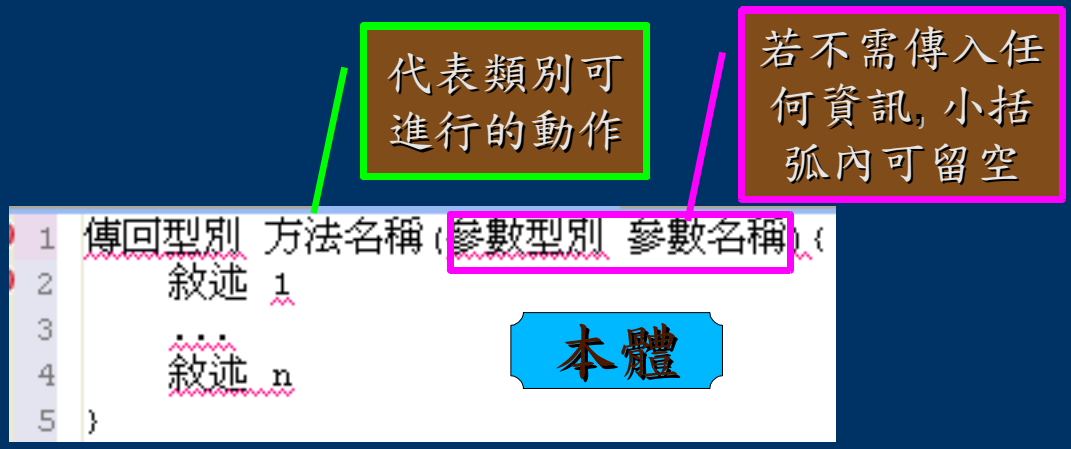
用來取得物件成員變數的運算子

# 類別的屬性—成員變數 (Cont.)

```
1  class Car{
2      double gas; // 載油量
3      double eff; // 耗油率
4  }
5
6  public class BuildCar2{
7      public static void main(String[] argv){
8          Car oldcar = new Car(); //建立物件
9
10         oldcar.gas = 100; //設定成員變數值
11         oldcar.eff = 10;
12         System.out.print("老爺車 oldcar 目前載油量爲 "+oldcar.gas+"公升, ");
13         System.out.println("耗油率爲每公升跑 "+oldcar.eff+"公里");
14     }
15 }
```

# 類別的行為—方法 (Method)

- 要想登台演出，光是為各個角色描繪特性還不夠，因為還沒有提供這些角色可以做的動作，就算站上舞台也只是不會動的木偶，無法演出
- 要讓物件可做的動作，就必須在類別中，用**方法(Method)**來描述物件的行為，定義方法的語法如下：



# 類別的行為—方法 (Method) (Cont.)

- 方法和運算式類似, 可以有一個**運算結果**, 這個運算結果稱為方法的**傳回值**（Return Value）
- 在方法名稱前面的**傳回值型別**（Return Type）就標示了運算結果的型別, 例如 int 即表示方法會傳回一個整數型別的結果, 我們必須在方法本體中用 return 敘述將整數資料傳回：

回傳值的資料型態

```
1 int getSomthing(){
2     ....
3     return X;
4 }
```

# 類別的行為—方法 (Method) (Cont.)

- 如果方法不會傳回任何結果, 可將方法的型別設為 void, 方法中也不必有 return 敘述。
- 要在 main() 方法中呼叫類別的方法, 與存取成員變數一樣, 都是用小數點, 例如：『物件.方法名稱()』。呼叫方法時, 程式執行流程會跳到此方法本體中的第一個敘述開始執行, 一直到該**本體結束**或是**遇到** return **敘述**為止, 然後再跳回原處繼續執行

# 類別的行為—方法 (Method) (Cont.)

```
1  class Car{
2      double gas; // 載油量
3      double eff; // 耗油率
4
5      void printState(){ //顯示物件狀態的方法(Method)
6          System.out.print("目前載油量爲 "+gas+"公升, ");
7          System.out.println("耗油率爲每公升跑 "+eff+"公里");
8      }
9  }
10
11 public class BuildCar2{
12     public static void main(String[] argv){
13         Car oldcar = new Car(); //建立物件
14
15         oldcar.gas = 100; //設定成員變數值
16         oldcar.eff = 10;
17         System.out.print("老爺車 oldcar ");
18         oldcar.printState(); //呼叫方法
19     }
20 }
```

老爺車 oldcar 目前載油量為 100.0 公升，耗油率為每公升跑 10.0 公里

# 各自獨立的物件

- 類別就像是角色, 我們可以找多位演員來演出, 但他們彼此都是獨立的, 擁有各自的屬性
- 如果我們建立多個物件, 並分別設定不同的屬性, 可看出物件是獨立的, 每個物件都會有各自的屬性 (成員變數), 互不相干：

# 各自獨立的物件 (Cont.)

```
1  class Car{
2      double gas; // 載油量
3      double eff; // 耗油率
4
5      void printState(){ //顯示物件狀態的方法(Method)
6          System.out.print("目前載油量爲 "+gas+"公升, ");
7          System.out.println("耗油率爲每公升跑 "+eff+"公里");
8      }
9  }
10
11 public class BuildCar2{
12     public static void main(String[] argv){
13         Car oldcar = new Car(); //建立物件
14         Car newcar = new Car(); //建立物件
15
16         oldcar.gas = 100; //設定 oldcar 成員變數值
17         oldcar.eff = 10;
18         newcar.gas = 120; //設定 newcar 成員變數值
19         newcar.eff = 20;
20         System.out.print("老爺車 oldcar ");
21         oldcar.printState(); //呼叫方法
22         System.out.print("新車 newcar ");
23         oldcar.printState(); //呼叫方法
24     }
25 }
```

老爺車 oldcar 目前載油量爲 100.0 公升，耗油率爲每公升跑 10.0 公里
新車 newcar 目前載油量爲 120.0 公升，耗油率爲每公升跑 20.0 公里

# 物件變數都是參照型別

- 物件是參照型別
- 指定運算及變更成員變數值時, 要特別清楚參照與實際物件的差異

HelloWorld.java | test.java | *OOTest.java

```
1  /* Program name: OOTest.java
2   * Author: Yung-Chen Chou
3   * Date: Apr. 6, 2009
4   */
5  class Test{
6      int x = 3;
7
8      void show(){
9          System.out.println("x = "+x);
10     }
11 }
12
13 public class OOTest{
14     public static void main(String[] argv){
15         Test a, b, c;
16         a = new Test();
17         b = new Test();
18         System.out.println("a == b ? " + (a ==b));
19         c = b;
20         c.x = 10;
21         System.out.println("c == b ? "+(c==b));
22         System.out.print("a.");
23         a.show();
24         System.out.print("b.");
25         b.show();
26         System.out.print("c.");
27         c.show();
28     }
29 }
```

# 物件陣列

- 物件陣列：用來建立多個物件時
- 每個陣列元素就是一個物件

```
/* Program name: OOTest.java
 * Author: Yung-Chen Chou
 * Date: Apr. 6, 2009
 */
class Car{
    int gas;
    double eff;

    void printState(){
        System.out.print("目前載油量 = "+gas);
        System.out.println(", 耗油爲每公升跑" + eff +"公里。\n");
    }
}

public class OOTest{
    public static void main(String[] argv){
        Car[] ManyCars = new Car[3];

        for(int i = 0;i<ManyCars.length;i++){
            ManyCars[i] = new Car();
            ManyCars[i].gas = 100 + i * 10;
            ManyCars[i].eff = 10 + i * 2.5;
        }
        for(int i = 0;i<ManyCars.length;i++){
            System.out.print((i+1)+"號車");
            ManyCars[i].printState();
        }
    }
}
```

```
1號車目前載油量 = 100, 耗油爲每公升跑10.0公里。

2號車目前載油量 = 110, 耗油爲每公升跑12.5公里。

3號車目前載油量 = 120, 耗油爲每公升跑15.0公里。
```

# 物件銷毀與回收

- 物件不再被用到時會被回收，就如同陣列的資源回收系統
- 監控方式 => 使用參照計數
- 物件的參照計數減少情況：
  1. 強迫釋放參照：物件參照型別變數設為 null
  2. 將變數指向別的物件：不再參照原來的物件
- 參照計數減至 0 時，該物件被標示為可回收，伺機從記憶體中將該物件清除掉

# 物件方法的參數

- 呼叫某物件的方法時一併將參數傳入

```
 7 class Car{
 8     double gas;
 9     double eff;
10
11     void printState(){
12         System.out.print(" 目前載油量爲 "+gas+" 公升");
13         System.out.print(" 耗油率爲每公升跑 "+eff+" 公里");
14     }
15     void move(double dis){
16         if(gas > (dis/eff)){
17             gas -= dis/eff;
18             System.out.println(" 行駛 "+dis+" 公里後剩 "+gas+" 公升油料");
19         }else{
20             System.out.println(" 距離太遠，目前油料( "+gas+" 公升)不足以到達目的地。");
21         }
22     }
23 }
24
25 public class OOPTest{
26     public static void main(String[] argv){
27         Car oldCar = new Car();
28         oldCar.gas = 100;
29         oldCar.eff = 10;
30         System.out.print("老爺車 oldCar ");
31         oldCar.move(50.5);
32         oldCar.printState();
33     }
34 }
```

```
老爺車 oldCar  行駛 50.5 公里後剩 94.95 公升油料
目前載油量爲 94.95 公升 耗油率爲每公升跑 10.0 公里
```

# 物件方法的參數 (Cont.)

```
老爺車
行駛了50.5 公里
行駛了100.0 公里
行駛了849.5 公里
目前載油量爲 0.0 公升 耗油率爲每公升跑 10.0 公里
```

- 注意事項：
  - 定義與使用參數資料型別必須一致
  - 定義與使用參數數量必須一致

```
 7 import java.io.*;
 8 class Car{
 9     double gas;
10     double eff;
11
12     void printState(){
13         System.out.print(" 目前載油量爲 "+gas+" 公升");
14         System.out.print(" 耗油率爲每公升跑 "+eff+" 公里");
15     }
16     double move(double dis){
17         if(gas > (dis/eff)){
18             gas -= dis/eff;
19             //System.out.println(" 行駛 "+dis+" 公里後剩 "+gas+" 公升油料");
20             return dis;
21         }else{
22             //System.out.println(" 距離太遠，目前油料( "+gas+" 公升)不足以到達目的地。");
23             dis = gas * eff;
24             gas = 0;
25             return dis;
26         }
27     }
28 }
29
30 public class OOPTest{
31     public static void main(String[] argv) throws IOException{
32         Car oldCar = new Car();
33         oldCar.gas = 100;
34         oldCar.eff = 10;
35         System.out.println("老爺車");
36         System.out.println(" 行駛了"+oldCar.move(50.5)+" 公里");
37         System.out.println(" 行駛了"+oldCar.move(100)+" 公里");
38         System.out.println(" 行駛了"+oldCar.move(5000)+" 公里");
39         oldCar.printState();
40     }
41 }
```

# 參數的傳遞

- 呼叫方法時傳遞參數的方式可分為下列幾種
  - 傳值方式傳遞 (Call by value)
  - 傳遞參照型別的參數 (Call by reference)
- Call by value
  - 將參數的值複製給呼叫的方法

# 傳值方式傳遞 (Call by value)

```
 7 import java.io.*;
 8 class Car{
 9     double gas;
10     double eff;
11
12     void printState(){..
16     double move(double dis){..
28     void changPara(double dis){
29         System.out.println("在 Car class 中的 changPara方法收到傳入參數 dis ="+dis);
30         System.out.println("參數修改中...");
31         dis = dis - 3.5;
32         System.out.println("修改參數 dis 值爲 :"+dis);
33     }
34 }
35
36 public class OOPTest{
37     public static void main(String[] argv) throws IOException{
38         Car oldCar = new Car();
39         oldCar.gas = 100;
40         oldCar.eff = 10;
41         double testVal = 45.7D;
42         System.out.println("呼叫方法前參數 testVal = "+testVal);
43         oldCar.changPara(testVal);
44         System.out.println("呼叫方法後參數 testVal = "+testVal);
45         // System.out.println("老爺車");
46         //System.out.println(" 行駛了"+oldCar.move(50.5)+" 公里");
47         //System.out.println(" 行駛了"+oldCar.move(100)+" 公里");
48         //System.out.println(" 行駛了"+oldCar.move(5000)+" 公里");
49         //oldCar.printState();
50     }
51 }
```

```
呼叫方法前參數 testVal = 45.7
在 Car class 中的 changPara方法收到傳入參數 dis =45.7
參數修改中...
修改參數 dis 值爲 :42.2
呼叫方法後參數 testVal = 45.7
```

# 傳遞參照型的參數 (Call by reference)

- 與傳值方式傳遞規則一樣
- 參照到相同的記憶體位址

```
1  class TestA{
2      int x = 3;
3      void show(){
4          System.out.println("x = "+x);
5      }
6  }
7  class TestB{
8      void changTestA(TestA t, int newX){
9          t.x = newX;
10     }
11 }
12
13 public class PassRef{
14     public static void main(String[] argv){
15         TestA a = new TestA();
16         TestB b = new TestB();
17         a.show();
18         b.changTestA(a, 20);
19         a.show();
20     }
21 }
```

```
x = 3
x = 20
```

# 變數的有效範圍 (Scope)

- Java 允許在程式的任何地方宣告變數
- 宣告後的變數並非永久可用
- 變數僅在其有效範圍存在
- 方法內的區域變數有效範圍
  - 在方法內宣告的變數是為區域變數
  - 變數在經過宣告及給予初始值之後才可使用
  - 一旦離開宣告該變數的方法，此變數便失效
  - 若該區塊裡面又有小區塊，則小區塊也能使用該變數，即內層區塊不能再重定義該變數

# 變數的有效範圍 (Scope)

```
13 public class PassRef{
14     public static void main(String[] argv){
15         /*
16         TestA a = new TestA();
17         TestB b = new TestB();
18         a.show();
19         b.changTestA(a, 20);
20         a.show();
21         */
22         int x = 1;
23         {
24             int y = 20;
25             {
26                 int z = 300;
27                 System.out.println("x = "+x);
28                 System.out.println("y = "+y);
29                 System.out.println("z = "+z);
30                 System.out.println();
31             }
32             int z = 40;
33             System.out.println("x = "+x);
34             System.out.println("y = "+y);
35             System.out.println("z = "+z);
36             System.out.println();
37         }
38         int y = 2;
39         int z = 3;
40         System.out.println("x = "+x);
41         System.out.println("y = "+y);
42         System.out.println("z = "+z);
43         System.out.println();
44     }
45 }
```

z = 300

y = 20
z = 40

x = 1
y = 2
z = 3

```
x = 1
y = 20
z = 300

x = 1
y = 20
z = 40

x = 1
y = 2
z = 3
```