

# 物件導向程式設計進階（二）

## 抽象、介面、套件

人工智慧與無線感應設備開發專班  
湜憶電腦知訊顧問股份有限公司

馬傳義

# 抽象類別與抽象方法

- 「抽象類別（Abstract Class）」是專門拿來當作「基礎類別（父類別）」的類別，有範本的作用，因為，要繼承它的子類別一定要依它的格式來定義。
- 建立「抽象類別」的用意或目的是希望可以根據既有的「樣本（sample）類別（即「抽象類別」）」，依據程式設計者的需要，修改或新增樣本類別原有的格式。
- 開發人員可以利用「抽象類別」來規範基本功能，並可確保其他設計人員不至於忽略的某些功能而沒有實作，以使得程式這些基本功能一定可以執行運作。

# 抽象類別與抽象方法

- 「抽象類別」最簡單的解釋是，無法具體化的類別。
  - ◆ 假設有一個叫做「形狀」的類別，而「圓形」類別與「方形」類別則是繼承「形狀」類別的子類別。
  - ◆ 「形狀」是一種抽象概念，無法具體化，因此「形狀」類別便是一個「抽象類別」。
- 「抽象類別」不能建立物件（即不能使用new關鍵字），只能被繼承。
- 在抽象類別裡面有一種「抽象方法（Abstract method）」成員，只有定義，沒有實作（即抽象類別內的「抽象方法」沒有敘述）。
  - ◆ 「抽象方法」實作的部份必須由子類別來定義。

# 抽象類別與抽象方法

- 「抽象類別」與「抽象方法成員」的應用，在多人合作開發大型應用程式時更可看出它們的重要性。
- 可以把「抽象類別」，看作完整程式的設計介面；而向下衍生的相關類別，必須依循「抽象類別」定義的規範，「重載」所有內含的「抽象類別」的「方法成員」。
- 使用抽象類別有一些規定事項：
  - ◆ 抽象類別因為沒有完整的定義類別內部資料，於是可以不用宣告成物件。
  - ◆ 仍然可以宣告抽象類別的建構子（Constructor）。
  - ◆ 抽象類別可以保留一般的類別方法。

# 抽象類別與抽象方法

- ◆ 抽象類別仍可以使用參考（Reference）物件。
- ◆ 抽象方法的存取修飾子必須設定為「public」或「protected」，不可以設定為「private」；至於「static」和「final」關鍵字也不行。
- ◆ 語法：  
    **abstract class** 抽象類別的名稱  
    {  
        資料成員的存取修飾子 資料成員;  
        方法成員的存取修飾子 方法成員的傳回值類型 方法成員的名稱  
            (方法成員的參數)  
        {  
            方法成員的主體程式;  
        }  
        抽象方法的存取修飾子 **abstract** 抽象方法的回傳值類型 抽象方法  
            的名稱(抽象方法的參數);  
    }

# 抽象類別與抽象方法

## ◆ 說明：

- abstract：
  - ◆ 抽象的關鍵字。
- class：
  - ◆ 類別的關鍵字。
- 抽象類別的名稱：
  - ◆ 自訂，但須符合命名規則。
- 資料成員的存取修飾子：
  - ◆ 須為「public」或「protected」，不可以為「private」。
- 資料成員：
  - ◆ 即類別的資料成員，自訂，但須符合命名規則。
- 方法成員的存取修飾子：
  - ◆ 須為「public」或「protected」，不可以為「private」。

# 抽象類別與抽象方法

- 方法成員的傳回值類型：
  - ◆ 即 傳回值的資料型別。
- 方法成員的名稱：
  - ◆ 自訂，但須符合命名規則。
- 方法成員的參數：
  - ◆ 即 引數。
- 方法成員的主體程式：
  - ◆ 即 方法成員的內容。
- 抽象方法的存取修飾子：
  - ◆ 須為「public」或「protected」，不可以為「private」。
- abstract：
  - ◆ 抽象的關鍵字。
- 抽象方法的回傳值類型：
  - ◆ 即 傳回值的資料型別。



# 抽象類別與抽象方法

- 抽象方法的名稱：
  - ◆ 自訂，但須符合命名規則。
- 抽象方法的參數：
  - ◆ 即引數。

## 注意：

- ◆ 「抽象類別」內的「方法成員」有兩種，一種是「一般的方法成員」，另一種是「抽象方法」。
- ◆ 「抽象方法」只有一行定義敘述，用「;」做結尾，「沒有定義實作部份（方法內沒有敘述）」。
- ◆ 「抽象方法」的修飾子只能為「public」、「protected」，不能為「private」，因「子類別」繼承時需要它。
- ◆ 「抽象方法」須用「abstract」關鍵字定義。
- ◆ 「子類別」繼承「抽象類別」後，必須將「抽象方法」「覆蓋（Override）」，即「子類別」使用「抽象方法名稱」定義方法的「實作部份」，方能使「抽象方法具體化」。



# 抽象類別與抽象方法

- ◆ 「抽象類別」不能「建立物件」，其「建構子」或「一般方法成員」須藉由「子類別」的「敘述」，使用「super」來呼叫。其中：
  - ◆ 「抽象類別」的「建構子」，藉由「子類別」的「建構子」中，在第一行敘述使用「super(引數串列)」來執行。
  - ◆ 「抽象類別」的「一般方法成員」，藉由「子類別」的「方法成員」中，使用「super.抽象類別的一般方法名稱(引數串列)」來執行。

# 抽象類別與抽象方法

◆ 程式：

```
package CH07_09;
```

```
abstract class CEmployee  
{  
    protected String name;  
    protected int base = 25000;  
    protected int sale_n;  
    protected int prize = 4000;
```

```
    CEmployee(String name)  
    {  
        this.name = name;  
    }
```

```
    public abstract int Salary();  
}
```

```
class CManager extends CEmployee
```

# 抽象類別與抽象方法

```
{  
    private int bonus;  
  
    CManager(String name, int sale_n, int bonus)  
    {  
        super(name);  
        this.sale_n = sale_n;  
        this.bonus = bonus;  
    }  
  
    public int Salary()  
    {  
        return base + prize * sale_n + bonus;  
    }  
  
    public String GetName()  
    {  
        return name;  
    }  
}
```

# 抽象類別與抽象方法

```
class CSales extends CEmployee
{
    CSales(String name, int sale_n)
    {
        super(name);
        this.sale_n = sale_n;
    }

    public int Salary()
    {
        return base + prize * sale_n;
    }

    public String GetName()
    {
        return name;
    }
}
```

# 抽象類別與抽象方法

```
public class CH07_09
{
    public static void main(String[] args)
    {
        CManager m1 = new CManager("林大山", 2, 20000);
        CManager m2 = new CManager("吳美莉", 4, 15000);
        CSales s1 = new CSales("張三丰", 3);
        CSales s2 = new CSales("李四娘", 6);

        System.out.println("姓名\t薪水");
        System.out.println("=====");
        System.out.println(m1.GetName() + "\t" + m1.Salary() + "元");
        System.out.println(m2.GetName() + "\t" + m2.Salary() + "元");
        System.out.println(s1.GetName() + "\t" + s1.Salary() + "元");
        System.out.println(s2.GetName() + "\t" + s2.Salary() + "元");
    }
}
```

# 抽象類別與抽象方法

```
1 package CH07_09;
2
3 abstract class CEmployee
4 {
5     protected String name;
6     protected int base = 25000;
7     protected int sale_n;
8     protected int prize = 4000;
9
10    CEmployee(String name)
11    {
12        this.name = name;
13    }
14
15    public abstract int Salary();
16 }
17
18 class CManager extends CEmployee
19 {
20     private int bonus;
21
22    CManager(String name, int sale_n, int bonus)
23    {
24        super(name);
25        this.sale_n = sale_n;
26        this.bonus = bonus;
```

# 抽象類別與抽象方法

```
27     }
28
29     public int Salary()
30     {
31         return base + prize * sale_n + bonus;
32     }
33
34     public String GetName()
35     {
36         return name;
37     }
38 }
39
40 class CSales extends CEmployee
41 {
42     CSales(String name, int sale_n)
43     {
44         super(name);
45         this.sale_n = sale_n;
46     }
47
48     public int Salary()
49     {
50         return base + prize * sale_n;
51     }
52 }
```



# 抽象類別與抽象方法

```
53     public String GetName()
54     {
55         return name;
56     }
57 }
58
59 public class CH07_09
60 {
61     public static void main(String[] args)
62     {
63         CManager m1 = new CManager("林大山", 2, 20000);
64         CManager m2 = new CManager("吳美莉", 4, 15000);
65         CSales s1 = new CSales("張三丰", 3);
66         CSales s2 = new CSales("李四娘", 6);
67
68         System.out.println("姓名\t薪水");
69         System.out.println("=====");
70         System.out.println(m1.GetName() + "\t" + m1.Salary() + "元");
71         System.out.println(m2.GetName() + "\t" + m2.Salary() + "元");
72         System.out.println(s1.GetName() + "\t" + s1.Salary() + "元");
73         System.out.println(s2.GetName() + "\t" + s2.Salary() + "元");
74     }
75 }
```

# 抽象類別與抽象方法

## ◆ 執行結果：

姓名	薪水
林大山	53000元
吳美莉	56000元
張三丰	37000元
李四娘	49000元

## ◆ 說明：

- 行01：
  - ◆ 定義「套件（package）」。
- 行03~行16：
  - ◆ 建立「抽象類別」「CEmployee」。
  - ◆ 行05~行08：
    - ◆ 宣告「保護」的類別「資料成員」，並指定初值。

# 抽象類別與抽象方法

- ◆ 行10~行13：
  - ◆ 宣告類別「建構子」`CEmployee(String name)`。
  - ◆ 因抽象類別不能建立物件，無法使用`new`來呼叫建構子，故須藉由子類別的建構子中，在第一行敘述使用「`super(引數串列)`」來執行。
- ◆ 行15：
  - ◆ 建立「公開」「抽象」的方法「`Salary()`」。
  - ◆ 因為「`Salary()`」方法為「抽象方法」，所以在每一個「子類別」中，皆要定義一個同名稱的「`Salary()`」方法來「覆蓋」這個抽象方法「`Salary()`」，只是在不同「子類別」的「`Salary()`」方法中，所定義的實作敘述不同，如此才能使「抽象方法」達到「具體化」的目的。

# 抽象類別與抽象方法

- 行18~行38：
  - ◆ 建立繼承自「CEmployee」父類別的「CManager」子類別。
  - ◆ 行20：
    - ◆ 宣告「私有」的類別「資料成員」。
    - ◆ 雖然是繼承自「抽象類別」**「CEmployee」**，但可以新增自己的「資料成員」。
  - ◆ 行22~行27：
    - ◆ 宣告類別「建構子」**「CManager(String name, int sale\_n, int bonus)」**。
    - ◆ 因為繼承自「抽象類別」**「CEmployee」**，故在執行new來建立「子類別」的物件時，不會「自動」呼叫「抽象類別」**「CEmployee」**的建構子，所以須藉由「子類別」**「建構子」**中的第一行敘述（此例為**super(name)**）來執行「抽象類別」**「CEmployee」**的「建構子」。

# 抽象類別與抽象方法

- ◆ 行29~行32：
  - ◆ 實作「抽象方法」`Salary()`。
  - ◆ 因為繼承自「抽象類別」`CEmployee`，故須實作「抽象方法」`Salary()`。
- ◆ 行34~行37：
  - ◆ 建立「公開」的類別方法`GetName()`。
  - ◆ 雖然是繼承自「抽象類別」`CEmployee`，但可以新增自己的「方法成員」。
- 行40~行57：
  - ◆ 建立繼承自「`CEmployee`」父類別的「`CSales`」子類別。
  - ◆ 行42~行46：
    - ◆ 宣告類別「建構子」`CSales(String name, int sale_n)`。
    - ◆ 因為繼承自「抽象類別」`CEmployee`，所以須藉由「建構子」中的第一行敘述（此例為`super(name)`）來執行「抽象類別」`CEmployee`的「建構子」。

# 抽象類別與抽象方法

- ◆ 行48~行51：
  - ◆ 實作「抽象方法」`Salary()`。
  - ◆ 因為繼承自「抽象類別」`CEmployee`，故須實作「抽象方法」`Salary()`。
  - ◆ 實作時，`Salary()`的內容，可以跟類別`CManager`的`Salary()`的內容不同。
- ◆ 行53~行56：
  - ◆ 建立「公開」的類別方法`GetName()`。
  - ◆ 雖然是繼承自「抽象類別」`CEmployee`，但可以新增自己的「方法成員」。
- 行63：
  - ◆ 利用`CManager`的「建構子」，建立m1物件，並利用引數初始化m1物件。
- 行64：
  - ◆ 利用`CManager`的「建構子」，建立m2物件，並利用引數初始化m2物件。

# 抽象類別與抽象方法

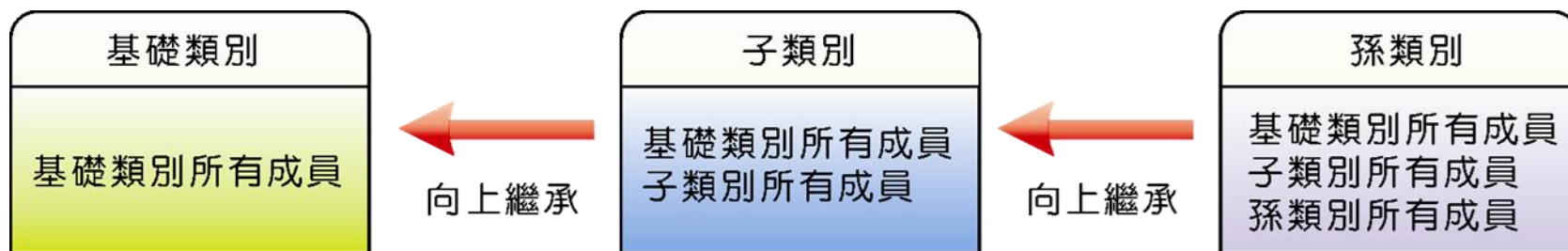
- 行65：
  - ◆ 利用「CSales」的「建構子」，建立s1物件，並利用引數初始化s1物件。
- 行66：
  - ◆ 利用「CSales」的「建構子」，建立s2物件，並利用引數初始化s2物件。
- 行70~行73：
  - ◆ 呼叫並顯示各物件的「GetName()」及「Salary()」方法的傳回值。



# 介面

## ■ 介面 (Interface)

- ◆ 在Java之中是不允許類別的「多重繼承」行為，每個衍生類別「僅能向上繼承單一基礎類別（父類別）」。
  - 有時使用者可能會利用「多代繼承」的方式（即先繼承一個類別之後，再透過此衍生類別去繼承另一個類別），來完成「多重繼承」的行為。
    - ◆ 「多代繼承」架構圖所示：



# 介面

- ◆ Java提供一種特殊類別，稱為「介面（Interface）」。
- ◆ 它為沒有「繼承」關係的「類別」提供一些「功能規格」。
- 例如：
  - ◆ 電腦的週邊可接螢幕、鍵盤、滑鼠...，這些週邊與電腦沒有繼承關係但有些關連。
  - ◆ 每一家週邊製造廠商在生產時，產品的內部材料、線路結構會有不同，但一定要符合與電腦相接連的介面規格，如此產品才能通用。
- ◆ 「介面」中所有的「資料成員」不需經過額外宣告，都會被自動的定義成「static」與「final」的型態，所以「介面」經常被用來定義程式中所需要的各種「常數」。

# 介面

- ◆ 「介面」的「實作」不能直接經由「new」運算子建立物件，須透過「實作介面」的「類別」來建立物件。
- ◆ 一個「介面」的原始檔案經編譯後，會產生一個「介面」的類別檔案（class）。
- ◆ 「類別」可透過「介面」的「實作」，達到「多型」的效果。
- ◆ 「介面」與「抽象類別」相似，它們之間最大的差異在於：
  - 一個「衍生」的「抽象類別（子類別）」僅能「繼承」「單一」「基礎」的「抽象類別（父類別）」。

# 介面

- 「介面」可讓開發人員撰寫出內含「多種」「介面」協定「實作」的「類別」物件（即一個「類別」可「實作」「一個」「介面」，也可「實作」「兩個以上」的「介面」）。

例如：

某類別定義了資料表內容查詢的各種成員方法，但是在執行查詢過程之中，還必須與其它程式或物件進行互動。

- 一個「介面」可同時給「不同」「類別」「實作」。
- 「抽象類別」內，可以定義「抽象方法」及「一般方法」；「介面」僅可以定義「抽象方法」。

# 介面

## ◆ 使用介面有一些規定事項：

- 介面無法實體化，因此無法產生建構子（Constructor）。
- 「介面」的「資料成員」必須有「初始值」（宣告為常數），即為「介面」訂出一些規格。
  - ◆ 「資料成員」的設定值被限制在其它地方「無法更改」。
  - ◆ 因「介面」的「資料成員」本質即是「常數」，所以「final」關鍵字是可以省略的，但省略了「final」關鍵字，「資料成員」的「初始值」仍無法更改。
- 「介面」不能定義「一般」的「方法成員」。
  - ◆ 因「介面」的「方法成員」本質即是「抽象方法」，所以「abstract」關鍵字是可以省略的，但省略了「abstract」關鍵字，還是「抽象方法」。

# 介面

- ◆ 「介面」的「抽象方法」在定義時，存取修飾子只能是「public」（public也可省略），但不能為「private」或「protected」。
- 「介面」的「方法成員」皆為「抽象方法」，只能定義功能的「原型」，其「方法」「實作」的部份留待給「相關連」的「類別」處理。

# 介面

- ◆ 定義「介面」的語法：

**interface 介面名稱**

**{**

**[static] [final] 資料型態 資料成員名稱 = 初始值;**

**[存取修飾子] abstract 傳回值型別 抽象方法名稱  
([引數串列]);**

**}**

- ◆ 說明：

- interface：

- ◆ 「介面」的關鍵字。



# 介面

- 介面名稱：
  - ◆ 自訂，但須符合命名規則。
- static：
  - ◆ 宣告為「介面」的「類別成員」。
  - ◆ 因為「介面」內的「資料成員」，一定是「類別成員」，故可省略。
- final：
  - ◆ 宣告為「常數」。
  - ◆ 因為「介面」內的「資料成員」，一定是「常數」，故可省略。
- 資料型態
  - ◆ 回傳值的資料型別。
- 資料成員名稱：
  - ◆ 自訂，但須符合命名規則。

# 介面

- 初始值：
  - ◆ 「資料成員」的內容。
  - ◆ 因為「介面」內的「資料成員」，一定是「常數」，故「必須」指定「初始值」（即不可省略）。
- 存取修飾子：
  - ◆ 存取權限等級。
  - ◆ 因為「介面」內的「方法成員」，一定是「public」，故可省略。
- abstract：
  - ◆ 「抽象方法」的關鍵字。
  - ◆ 因為「介面」內的「方法成員」，一定是「abstract」，故可省略。
- 傳回值型別：
  - ◆ 回傳值的資料型別。

# 介面

- 抽象方法名稱：
  - ◆ 自訂，但須符合命名規則。
- 引數串列：
  - ◆ 若無，可以省略。

## ◆ 「實作」「介面」的語法：

```
class 類別名稱 implements 介面名稱1, 介面名稱2, ...  
{  
    敘述區段;  
}
```

## ◆ 說明：

- class：
  - ◆ 「類別」的關鍵字。

# 介面

- 類別名稱：
  - ◆ 自訂，但須符合命名規則。
- implements：
  - ◆ 「實作」「介面」的關鍵字。
- 介面名稱1, 介面名稱2, ...：
  - ◆ 要「實作」的「介面」名稱。
  - ◆ 若要「實作」「兩個（含以上）」的「介面」，中間以逗號「,」分隔。
- 敘述區段：
  - ◆ 「實作」「介面」中定義的「抽象方法」。
  - ◆ 亦可加入自己的「資料成員」與「方法成員」。

# 介面

## ◆ 程式：

```
package CH07_10;
```

```
interface ILimit
```

```
{
```

```
    int HIGH = 110;
```

```
    int LOW = 60;
```

```
    String Drive();
```

```
}
```

```
interface IDistance
```

```
{
```

```
    void Process(int min);
```

```
}
```

```
class CTruck implements ILimit
```

```
{
```

```
    private int speed;
```

# 介面

```
CTruck(int speed)
{
    this.speed = speed;
}

public String Drive()
{
    if (speed > HIGH)
        return "目前超速行駛，會影響交通安全。";
    else if (speed < LOW)
        return "目前低速行駛，會阻塞交通。";
    else
        return "目前正常行駛，請保持距離。";
}
}

class CCar implements ILimit, IDistance
{
```

# 介面

```
private int speed;

CCar(int speed)
{
    this.speed = speed;
}

public String Drive()
{
    if (speed > HIGH)
        return "目前車速 " + speed + " km/hr，請減速。";
    else if (speed < LOW)
        return "目前車速 " + speed + " km/hr，請加速。";
    else
        return "目前車速 " + speed + " km/hr，請保持。";
}

public void Process(int min)
{
    double leng = speed * min / 60;
```



# 介面

```
        System.out.println("已行駛" + min + "分，跑了" + leng + "公里。");
    }
}

public class CH07_10
{
    public static void main(String[] args)
    {
        CTruck t1 = new CTruck(50);
        System.out.println("t1大型車：");
        System.out.println(t1.Drive());

        System.out.println();

        CCar c1 = new CCar(120);
        System.out.println("c1小型車：");
        System.out.println(c1.Drive());
        c1.Process(2);
    }
}
```

# 介面

```
1 package CH07_10;
2
3 interface ILimit
4 {
5     int HIGH = 110;
6     int LOW = 60;
7
8     String Drive();
9 }
10
11 interface IDistance
12 {
13     void Process(int min);
14 }
15
16 class CTruck implements ILimit
17 {
18     private int speed;
19
20     CTruck(int speed)
21     {
22         this.speed = speed;
23     }
24
25     public String Drive()
26     {
```

# 介面

```
27     if (speed > HIGH)
28         return "目前超速行駛，會影響交通安全。";
29     else if (speed < LOW)
30         return "目前低速行駛，會阻塞交通。";
31     else
32         return "目前正常行駛，請保持距離。";
33 }
34 }
35
36 class CCar implements ILimit, IDistance
37 {
38     private int speed;
39
40     CCar(int speed)
41     {
42         this.speed = speed;
43     }
44
45     public String Drive()
46     {
47         if (speed > HIGH)
48             return "目前車速 " + speed + " km/hr，請減速。";
49         else if (speed < LOW)
50             return "目前車速 " + speed + " km/hr，請加速。";
51         else
52             return "目前車速 " + speed + " km/hr，請保持。";
```

# 介面

```
53     }
54
55     public void Process(int min)
56     {
57         double leng = speed * min / 60;
58         System.out.println("已行駛" + min + "分，跑了" + leng + "公里。");
59     }
60 }
61
62 public class CH07_10
63 {
64     public static void main(String[] args)
65     {
66         CTruck t1 = new CTruck(50);
67         System.out.println("t1大型車：");
68         System.out.println(t1.Drive());
69
70         System.out.println();
71
72         CCar c1 = new CCar(120);
73         System.out.println("c1小型車：");
74         System.out.println(c1.Drive());
75         c1.Process(2);
76     }
77 }
```

# 介面

## ◆ 執行結果：

t1大型車：  
目前低速行駛，會阻塞交通。

c1小型車：  
目前車速 120 km/hr，請減速。  
已行駛2分，跑了4.0公里。

## ◆ 說明：

- 行01：
  - ◆ 定義「套件（package）」。
- 行03~行09：
  - ◆ 建立「介面」「ILimit」。
  - ◆ 行05~行06：
    - ◆ 宣告為「介面」的「類別成員」，並指定初值。
    - ◆ 因為「介面」內的「資料成員」，一定是「類別成員」，且一定是「最終」，故省略「static」及「final」。

# 介面

- ◆ 行08：
  - ◆ 建立「介面」的「公開」「抽象」的方法「Drive()」。
  - ◆ 因為「介面」內的「方法成員」，一定是「公開」，且為「抽象方法」，故省略「public」及「abstract」。
- 行11~行14：
  - ◆ 建立「介面」「IDistance」。
  - ◆ 行13：
    - ◆ 建立「介面」的「公開」「抽象」的方法「Process(int min)」。
    - ◆ 因為「介面」內的「方法成員」，一定是「公開」，且為「抽象方法」，故省略「public」及「abstract」。

# 介面

- 行16~行34：
  - ◆ 「實作」自「ILimit」介面的「CTruck」類別。
  - ◆ 行18：
    - ◆ 宣告「私有」的類別「資料成員」。
  - ◆ 行20~行23：
    - ◆ 宣告類別「建構子」`CTruck(int speed)`。
  - ◆ 行25~行33：
    - ◆ 「實作」自「ILimit」介面中的「抽象方法」`Drive()`。
- 行36~行60：
  - ◆ 「實作」自「ILimit」、「IDistance」介面的「CCar」類別。
  - ◆ 行38：
    - ◆ 宣告「私有」的類別「資料成員」。

# 介面

- ◆ 行40~行43：
  - ◆ 宣告類別「建構子」「CCar(int speed)」。
- ◆ 行45~行53：
  - ◆ 「實作」自「ILimit」介面中的「抽象方法」「Drive()」。
- ◆ 行55~行59：
  - ◆ 「實作」自「IDistance」介面中的「抽象方法」「Process(int min)」。
- 行66：
  - ◆ 利用「CTruck」的「建構子」，建立t1物件，並利用引數初始化t1物件。
- 行68：
  - ◆ 顯示t1物件「Drive()」的傳回值。



# 介面

- 行72：
  - ◆ 利用「CCar」的「建構子」，建立c1物件，並利用引數初始化c1物件。
- 行74：
  - ◆ 顯示c1物件「Drive()」的傳回值。
- 行75：
  - ◆ 執行c1物件的「Process(int min)」方法。

# 介面

■ 「介面」的「繼承」：

- ◆ 「介面」也有「繼承」的關係，且可以「多重繼承」。
- ◆ 當某個類別「實作」了「子介面」，必須連同「父介面」的「抽象方法」一併完成。
- ◆ 語法：

```
interface 子介面名稱 extends 介面名稱1, 介面名稱2, ...  
{  
    [static] [final] 資料型態 資料成員名稱 = 初始值;  
    [存取修飾子] abstract 傳回值型別 抽象方法名稱  
        ([引數串列]);  
}
```

# 介面

## ◆ 程式：

```
package CH07_11;
```

```
interface ILimit  
{  
    int HIGH = 110;  
    int LOW = 60;  
  
    String Drive();  
}
```

```
interface IDistance  
{  
    void Process(int min);  
}
```

```
interface ISpeed extends ILimit, IDistance  
{  
    void Keep();  
}
```

# 介面

```
}  
  
class CTruck implements ILimit, IDistance  
{  
    private int speed;  
  
    CTruck(int speed)  
    {  
        this.speed = speed;  
    }  
  
    public String Drive()  
    {  
        if (speed > HIGH)  
            return "目前超速行駛，會影響交通安全。";  
        else if (speed < LOW)  
            return "目前低速行駛，會阻塞交通。";  
        else  
            return "目前正常行駛，請保持距離。";  
    }  
}
```

# 介面

```
public void Process(int min)
{
    System.out.println("已行駛了" + (double) (speed * min / 60) + "公里。");
}
}
```

```
class CCar implements ISpeed
{
    private int speed;
```

```
    CCar(int speed)
    {
        this.speed = speed;
    }
```

```
    public String Drive()
    {
        if (speed > HIGH)
            return "目前車速 " + speed + " km/hr，請減速。";
    }
```

# 介面

```
    else if (speed < LOW)
        return "目前車速 " + speed + " km/hr，請加速。";
    else
        return "目前車速 " + speed + " km/hr，請保持。";
}

public void Process(int min)
{
    double leng = speed * min / 60;
    System.out.println("已行駛" + min + "分，跑了" + leng + "公里。");
}

public void Keep()
{
    System.out.println("請與前車保持" + (int) (speed / 2) + "公尺以上距離。");
}
}
```

# 介面

```
public class CH07_11
{
    public static void main(String[] args)
    {
        CTruck t1 = new CTruck(50);
        System.out.println("t1大型車 : ");
        System.out.println(t1.Drive());
        t1.Process(15);

        System.out.println();

        CCar c1 = new CCar(120);
        System.out.println("c1小型車 : ");
        System.out.println(c1.Drive());
        c1.Process(10);
        c1.Keep();
    }
}
```

# 介面

```
1 package CH07_11;
2
3 interface ILimit
4 {
5     int HIGH = 110;
6     int LOW = 60;
7
8     String Drive();
9 }
10
11 interface IDistance
12 {
13     void Process(int min);
14 }
15
16 interface ISpeed extends ILimit, IDistance
17 {
18     void Keep();
19 }
20
21 class CTruck implements ILimit, IDistance
22 {
23     private int speed;
24
25     CTruck(int speed)
26     {
```



# 介面

```
27     this.speed = speed;
28 }
29
30 public String Drive()
31 {
32     if (speed > HIGH)
33         return "目前超速行駛，會影響交通安全。";
34     else if (speed < LOW)
35         return "目前低速行駛，會阻塞交通。";
36     else
37         return "目前正常行駛，請保持距離。";
38 }
39
40 public void Process(int min)
41 {
42     System.out.println("已行駛了" + (double) (speed * min / 60) + "公里。");
43 }
44 }
45
46 class CCar implements ISpeed
47 {
48     private int speed;
49
50     CCar(int speed)
51     {
52         this.speed = speed;
```

# 介面

```
53     }
54
55     public String Drive()
56     {
57         if (speed > HIGH)
58             return "目前車速 " + speed + " km/hr，請減速。";
59         else if (speed < LOW)
60             return "目前車速 " + speed + " km/hr，請加速。";
61         else
62             return "目前車速 " + speed + " km/hr，請保持。";
63     }
64
65     public void Process(int min)
66     {
67         double leng = speed * min / 60;
68         System.out.println("已行駛" + min + "分，跑了" + leng + "公里。");
69     }
70
71     public void Keep()
72     {
73         System.out.println("請與前車保持" + (int) (speed / 2) + "公尺以上距離。");
74     }
75 }
76
77 public class CH07_11
78 {
```

# 介面

```
79  public static void main(String[] args)
80  {
81      CTruck t1 = new CTruck(50);
82      System.out.println("t1大型車：");
83      System.out.println(t1.Drive());
84      t1.Process(15);
85
86      System.out.println();
87
88      CCar c1 = new CCar(120);
89      System.out.println("c1小型車：");
90      System.out.println(c1.Drive());
91      c1.Process(10);
92      c1.Keep();
93  }
94 }
```

## ◆ 執行結果：

t1大型車：  
目前低速行駛，會阻塞交通。  
已行駛了12.0公里。

c1小型車：  
目前車速 120 km/hr，請減速。  
已行駛10分，跑了20.0公里。  
請與前車保持60公尺以上距離。

# 介面

## ◆ 說明：

- 行01：
  - ◆ 定義「套件（package）」。
- 行03~行09：
  - ◆ 建立「介面」`ILimit`。
  - ◆ 行05~行06：
    - ◆ 宣告為「介面」的「類別成員」，並指定初值。
    - ◆ 因為「介面」內的「資料成員」，一定是「類別成員」，且一定是「最終」，故省略「`static`」及「`final`」。
  - ◆ 行08：
    - ◆ 建立「介面」的「公開」「抽象」的方法「`Drive()`」。
    - ◆ 因為「介面」內的「方法成員」，一定是「公開」，且為「抽象方法」，故省略「`public`」及「`abstract`」。

# 介面

- 行11~行14：
  - ◆ 建立「介面」`IDistance`。
  - ◆ 行13：
    - ◆ 建立「介面」的「公開」「抽象」的方法「`Process(int min)`」。
    - ◆ 因為「介面」內的「方法成員」，一定是「公開」，且為「抽象方法」，故省略「`public`」及「`abstract`」。
- 行16~行19：
  - ◆ 建立繼承自「`ILimit`」、「`IDistance`」父介面的「`ISpeed`」子介面。
  - ◆ 行18：
    - ◆ 建立「介面」的「公開」「抽象」的方法「`Keep()`」。
    - ◆ 因為「介面」內的「方法成員」，一定是「公開」，且為「抽象方法」，故省略「`public`」及「`abstract`」。

# 介面

- 行21~行44：
  - ◆ 「實作」自「ILimit」、「IDistance」介面的「CTruck」類別。
  - ◆ 行23：
    - ◆ 宣告「私有」的類別「資料成員」。
  - ◆ 行25~行28：
    - ◆ 宣告類別「建構子」`CTruck(int speed)`。
  - ◆ 行30~行38：
    - ◆ 「實作」自「ILimit」介面中的「抽象方法」`Drive()`。
  - ◆ 行40~行43：
    - ◆ 「實作」自「IDistance」介面中的「抽象方法」`Process(int min)`。

# 介面

- 行46~行75：
  - ◆ 「實作」自「ISpeed」介面的「CCar」類別。
  - ◆ 行48：
    - ◆ 宣告「私有」的類別「資料成員」。
  - ◆ 行50~行53：
    - ◆ 宣告類別「建構子」`CCar(int speed)`。
  - ◆ 行55~行63：
    - ◆ 「實作」自「ILimit」介面中的「抽象方法」`Drive()`。
  - ◆ 行65~行69：
    - ◆ 「實作」自「IDistance」介面中的「抽象方法」`Process(int min)`。
    - ◆ 因為「ISpeed」介面是繼承自「ILimit」及「IDistance」介面，故必須一併「實作」`ILimit`及`IDistance`介面中的「抽象方法」。

# 介面

- ◆ 行71~行74：
  - ◆ 「實作」自「ISpeed」介面中的「抽象方法」  
「Keep()」。
- 行81：
  - ◆ 利用「CTruck」的「建構子」，建立t1物件，並利用引數初始化t1物件。
- 行83：
  - ◆ 顯示t1物件「Drive()」的傳回值。
- 行84：
  - ◆ 執行t1物件的「Process(int min)」方法。
- 行88：
  - ◆ 利用「CCar」的「建構子」，建立c1物件，並利用引數初始化c1物件。



# 介面

- 行90：
  - ◆ 顯示c1物件「Drive()」的傳回值。
- 行91：
  - ◆ 執行c1物件的「Process(int min)」方法。
- 行92：
  - ◆ 執行c1物件的「Keep()」方法。

# 套件

- 隨著程式架構越來越大，「類別」個數越來越多，會發現管理程式時，維護「類別名稱」是一件麻煩的事，尤其是一些「同名」問題的發生。

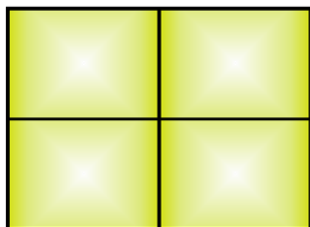
例如：

程式中，已經定義一個Car類別，但另一個合作開發程式的人員，並不曉得已經有這個類別名稱的存在，也定義了一個Car類別，於是編譯過後的Car類別檔案會「覆蓋（Override）」原來的Car類別檔案。

- 另外，大型程式通常會將程式依「類別」（或「功能」等）分成若干部份，由數人（或數組人）共同開發（如下圖）。

# 套件

主程式(含有main()  
的程式部份)



依不同的功能切割  
成A、B、C、D四個  
不同的類別

檔案切割



A類別

由A程式設計師  
負責撰寫

B類別

由B程式設計師  
負責撰寫

C類別

由C程式設計師  
負責撰寫

D類別

由D程式設計師  
負責撰寫

主程式



各自獨立完成後，再  
整合重組成完整程式



主程式整合的部份由另一  
程式設計師完成整合重組

# 套件

- Java提供了「套件（package）」的機制，用來將多個「類別」或「介面」集中收納的地方。
- 「套件」就像是一個「管理容器」，可以將所定義的名稱區隔管理在「套件」下，而不會有相互衝突的發生。

例如：

定義了Car與Truck的套件，在它們之下都有一個ISpeed的類別，但由於屬於不同的套件，所以這兩個名稱並不會有所衝突。

- Java的package被設計為與檔案系統結構相對應。

例如：

套件設定是Car，則該類別應該在指定目錄的Car下可以找到。

# 套件

■ Java系統之所以利用此種方式，來封裝所有相關類別主要是為了下列兩個重要因素：

## ◆ 方便類別名稱管理：

- 在許多情況下（由其當多人合力開發大型應用程式時），難保程式開發人員不會撰寫出內容不同，但識別名稱卻一樣的類別來。

## ◆ 提供存取保護機制：

- 由於「套件」封裝會將所有「目標類別儲存於同一路徑」之中，所以開發人員就可以利用這些「類別」的「存取修飾子」，進行存取權限的控制動作。

# 套件

■ Java本身就提供很多好用的套件（類別庫）。

例如：

BufferedReader類別是放在java.io類別庫內、  
String類別是放在java.lang類別庫內。

■ 套件宣告語法：

**package 套件名稱;**

◆ 說明：

- package：
  - ◆ 「套件」的關鍵字。
- 套件名稱：
  - ◆ 即「套件」的存放位置（路徑）。

# 套件

## ◆ 程式：

```
package CH07_12;
```

```
class CTruck
```

```
{
```

```
    int HIGH = 100;
```

```
    int LOW = 60;
```

```
    private int speed;
```

```
    CTruck(int speed)
```

```
{
```

```
        this.speed = speed;
```

```
}
```

```
    public String Drive()
```

```
{
```

```
        if (speed > HIGH)
```

```
            return "目前超速行駛，會影響交通安全。";
```

```
        else if (speed < LOW)
```

# 套件

```
        return "目前低速行駛，會阻塞交通。";
    else
        return "目前正常行駛，請保持距離。";
    }

    public void Process(int min)
    {
        System.out.println("已行駛了" + (double) (speed * min / 60) + "公里。");
    }
}

class CCar
{
    int HIGH = 120;
    int LOW = 60;
    private int speed;

    CCar(int speed)
    {
        this.speed = speed;
    }
}
```



# 套件

```
}
```

```
public String Drive()
```

```
{
```

```
    if (speed > HIGH)
```

```
        return "目前車速 " + speed + " km/hr，請減速。";
```

```
    else if (speed < LOW)
```

```
        return "目前車速 " + speed + " km/hr，請加速。";
```

```
    else
```

```
        return "目前車速 " + speed + " km/hr，請保持。";
```

```
}
```

```
public void Process(int min)
```

```
{
```

```
    double leng = speed * min / 60;
```

```
    System.out.println("已行駛" + min + "分，跑了" + leng + "公里。");
```

```
}
```

```
public void Keep()
```

```
{
```

# 套件

```
        System.out.println("請與前車保持" + (int) (speed / 2) + "公尺以上距離。");
    }
}

public class CH07_12
{
    public static void main(String[] args)
    {
        CTruck t1 = new CTruck(50);
        System.out.println("t1大型車 : ");
        System.out.println(t1.Drive());
        t1.Process(15);

        System.out.println();

        CCar c1 = new CCar(120);
        System.out.println("c1小型車 : ");
        System.out.println(c1.Drive());
        c1.Process(10);
    }
}
```

# 套件

```
        c1.Keep();  
    }  
}
```

# 套件

```
1 package CH07_12;
2
3 class CTruck
4 {
5     int HIGH = 100;
6     int LOW = 60;
7     private int speed;
8
9     CTruck(int speed)
10    {
11        this.speed = speed;
12    }
13
14    public String Drive()
15    {
16        if (speed > HIGH)
17            return "目前超速行駛，會影響交通安全。";
18        else if (speed < LOW)
19            return "目前低速行駛，會阻塞交通。";
20        else
21            return "目前正常行駛，請保持距離。";
22    }
23
24    public void Process(int min)
25    {
26        System.out.println("已行駛了" + (double) (speed * min / 60) + "公里。");
```

# 套件

```
27     }
28 }
29
30 class CCar
31 {
32     int HIGH = 120;
33     int LOW = 60;
34     private int speed;
35
36     CCar(int speed)
37     {
38         this.speed = speed;
39     }
40
41     public String Drive()
42     {
43         if (speed > HIGH)
44             return "目前車速 " + speed + " km/hr ，請減速。";
45         else if (speed < LOW)
46             return "目前車速 " + speed + " km/hr ，請加速。";
47         else
48             return "目前車速 " + speed + " km/hr ，請保持。";
49     }
50
51     public void Process(int min)
52     {
```

# 套件

```
53     double leng = speed * min / 60;
54     System.out.println("已行駛" + min + "分，跑了" + leng + "公里。");
55 }
56
57 public void Keep()
58 {
59     System.out.println("請與前車保持" + (int) (speed / 2) + "公尺以上距離。");
60 }
61 }
62
63 public class CH07_12
64 {
65     public static void main(String[] args)
66     {
67         CTruck t1 = new CTruck(50);
68         System.out.println("t1大型車：");
69         System.out.println(t1.Drive());
70         t1.Process(15);
71
72         System.out.println();
73
74         CCar c1 = new CCar(120);
75         System.out.println("c1小型車：");
76         System.out.println(c1.Drive());
77         c1.Process(10);
78         c1.Keep();
```

# 套件

```
79 }  
80 }
```

## ◆ 執行結果：

t1大型車：  
目前低速行駛，會阻塞交通。  
已行駛了12.0公里。

c1小型車：  
目前車速 120 km/hr，請保持。  
已行駛10分，跑了20.0公里。  
請與前車保持60公尺以上距離。

## ◆ 說明：

- 行01：
  - ◆ 定義「套件（package）」。
- 行03~行28：
  - ◆ 建立「CTruck」類別。
  - ◆ 行05~行06：
    - ◆ 宣告「公開」的類別「資料成員」，並指定初值。

# 套件

- ◆ 行07：
  - ◆ 宣告「私有」的類別「資料成員」。
- ◆ 行09~行12：
  - ◆ 宣告類別「建構子」`CTruck(int speed)`。
- ◆ 行14~行22：
  - ◆ 宣告「公開」的類別「方法成員」`Drive()`。
- ◆ 行24~行27：
  - ◆ 宣告「公開」的類別「方法成員」`Process(int min)`。
- 行30~行61：
  - ◆ 建立「CCar」類別。
  - ◆ 行32~行33：
    - ◆ 宣告「公開」的類別「資料成員」，並指定初值。
  - ◆ 行34：
    - ◆ 宣告「私有」的類別「資料成員」。



# 套件

- ◆ 行36~行39：
  - ◆ 宣告類別「建構子」`CCar(int speed)`。
- ◆ 行41~行49：
  - ◆ 宣告「公開」的類別「方法成員」`Drive()`。
- ◆ 行51~行55：
  - ◆ 宣告「公開」的類別「方法成員」`Process(int min)`。
- ◆ 行57~行60：
  - ◆ 宣告「公開」的類別「方法成員」`Keep()`。
- 行67：
  - ◆ 利用「CTruck」的「建構子」，建立t1物件，並利用引數初始化t1物件。
- 行69：
  - ◆ 顯示t1物件「Drive()」的傳回值。

# 套件

- 行70：
  - ◆ 執行t1物件的「Process(int min)」方法。
- 行74：
  - ◆ 利用「CCar」的「建構子」，建立c1物件，並利用引數初始化c1物件。
- 行76：
  - ◆ 顯示c1物件「Drive()」的傳回值。
- 行77：
  - ◆ 執行c1物件的「Process(int min)」方法。
- 行78：
  - ◆ 執行c1物件的「Keep()」方法。

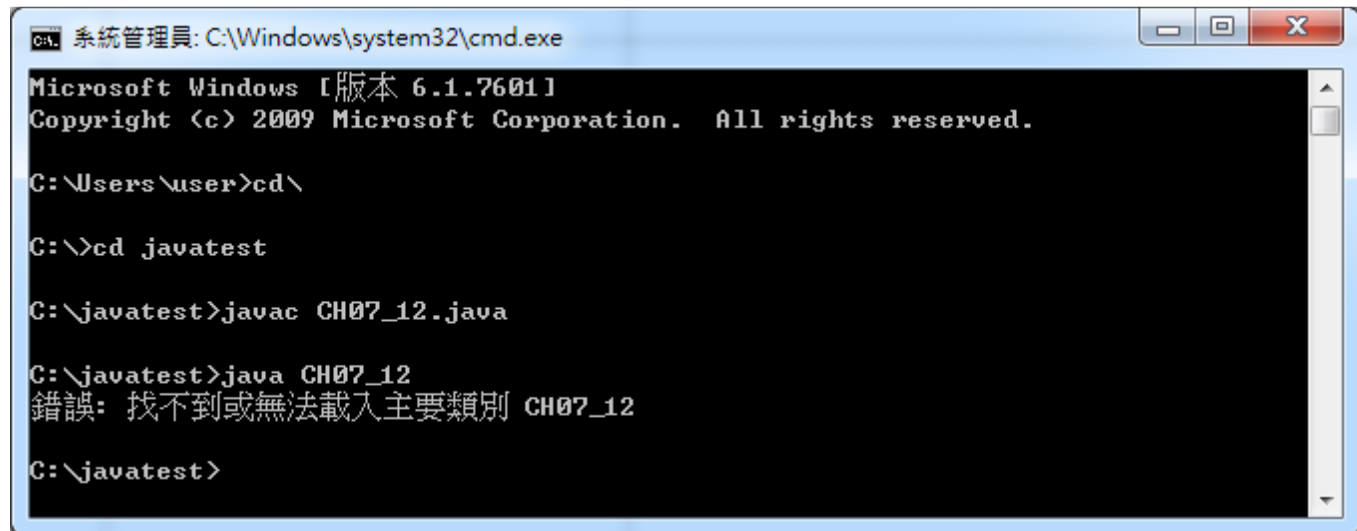
# 套件

■ 加入套件後，編譯和執行的做法和觀念上有些許的不同。

- ◆ 先在硬碟中建立與套件相同名稱的目錄，然後將原始程式儲存在目錄中。
- ◆ 在DOS下編譯和執行，就需要做些修正：
  - 編譯：「javac 目錄名稱\原始檔案名稱.java」。
  - 執行：「java 套件名稱.類別名稱」。
- ◆ 請將上例（CH07\_12.java）調整、修改如下：
  1. 在「C:\」下建立「javatest」資料夾。
  2. 將上例（CH07\_12.java）複製到「C:\javatest」的資料夾中。
  3. 開啟「命令提示字元」視窗。

# 套件

4. 切換至「C:\javatest」。
5. 輸入「javac CH07\_12.java」，按「Enter」。
6. 輸入「java CH07\_12」，按「Enter」，會看到下列畫面。



```
系統管理員: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\user>cd\

C:\>cd javatest

C:\javatest>javac CH07_12.java

C:\javatest>java CH07_12
錯誤: 找不到或無法載入主要類別 CH07_12

C:\javatest>
```

◆ 請將上例（CH07\_12.java）調整、修改如下：

1. 將原本「CH07\_12.java」中的第一行程式碼（「`package CH07_12;`」）刪除並存檔。
2. 開啟「命令提示字元」視窗。
3. 切換至「C:\javatest」。
4. 輸入「`javac CH07_12.java`」，按「Enter」。
5. 輸入「`java CH07_12`」，按「Enter」，會看到下列畫面。

# 套件



```
系統管理員: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\user>cd\

C:\>cd javatest

C:\javatest>javac CH07_12.java

C:\javatest>java CH07_12
t1大型車：
目前低速行駛，會阻塞交通。
已行駛了12.0公里。

c1小型車：
目前車速 120 km/hr，請保持。
已行駛10分，跑了20.0公里。
請與前車保持60公尺以上距離。

C:\javatest>
```

◆ 請將上例（CH07\_12.java）調整、修改如下：

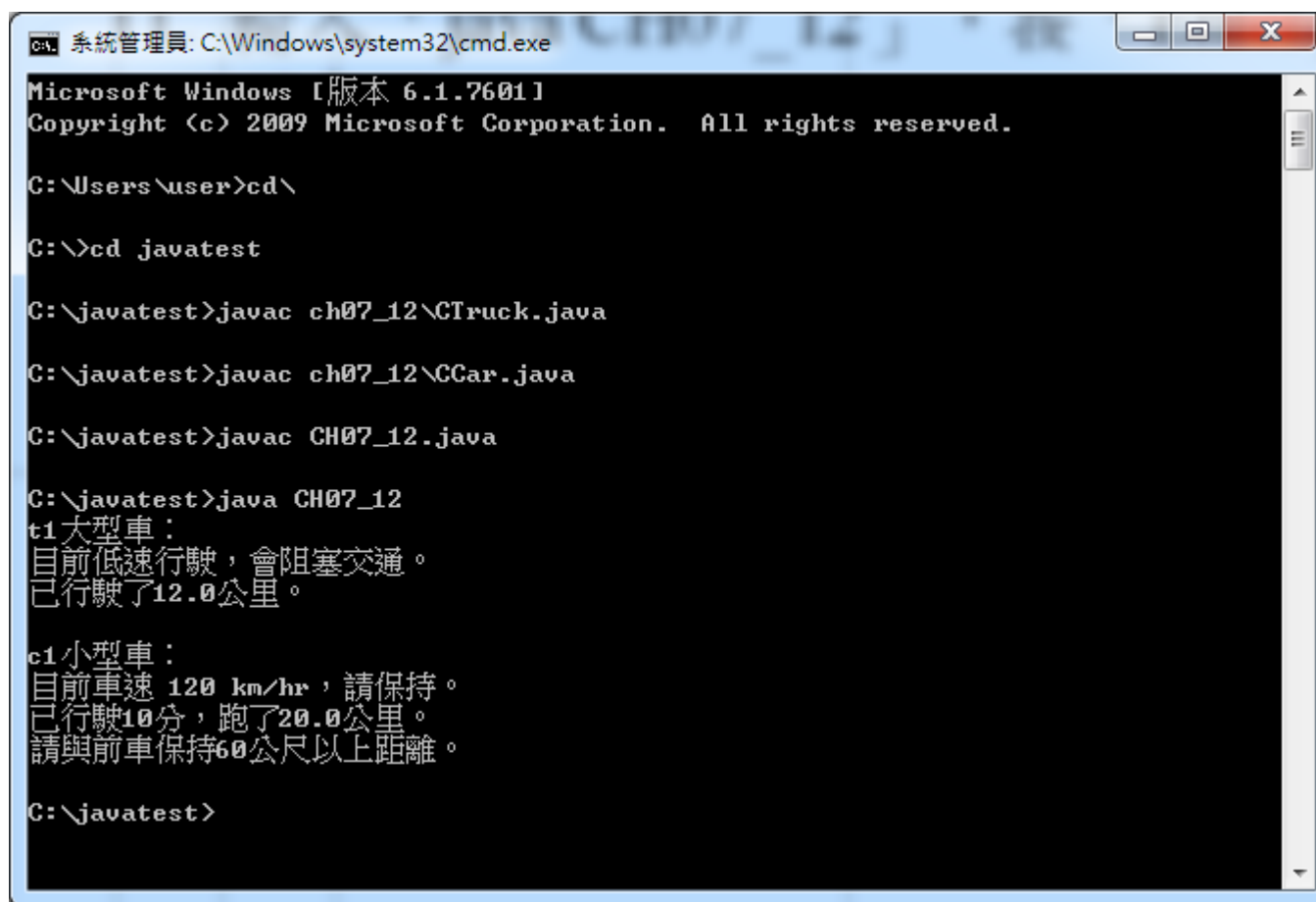
1. 在「C:\javatest」下建立「ch07\_12」資料夾。
2. 將原本「CH07\_12.java」中的「CTruck」類別，獨立成一個檔案（「CTruck.java」），並在該檔案的最上方加入「package ch07\_12;」，且將「類別」及「建構子」的「存取修飾子」更改為「public」。
3. 將原本「CH07\_12.java」中的「CCar」類別，獨立成一個檔案（「CCar.java」），並在該檔案的最上方加入「package ch07\_12;」，且將「類別」及「建構子」的「存取修飾子」更改為「public」。

# 套件

4. 「CH07\_12.java」中，只保留「CH07\_12」類別的那一部份的程式碼，其餘全部刪除，並在該檔案的最上方加入「import ch07\_12.CTruck;」及「import ch07\_12.CCar;」。
5. 將「CTruck.java」及「CCar.java」兩個檔案，移動至「ch07\_12」資料夾中。
6. 開啟「命令提示字元」。
7. 切換至「C:\javatest」。
8. 輸入「javac ch07\_12\CTruck.java」，按「Enter」。
9. 輸入「javac ch07\_12\CCar.java」，按「Enter」。
10. 輸入「javac CH07\_12.java」，按「Enter」。



11. 輸入「java CH07\_12」，按「Enter」，會看到下列畫面。



```
系統管理員: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\user>cd\

C:\>cd javatest

C:\javatest>javac ch07_12\CTruck.java

C:\javatest>javac ch07_12\CCar.java

C:\javatest>javac CH07_12.java

C:\javatest>java CH07_12
t1大型車：
目前低速行駛，會阻塞交通。
已行駛了12.0公里。

c1小型車：
目前車速 120 km/hr，請保持。
已行駛10分，跑了20.0公里。
請與前車保持60公尺以上距離。

C:\javatest>
```

# 套件

註：

- Java所提供的相關套件，存放在  
C:\Program Files\Java\jdk1.8.0\_112\src.zip（依Java  
的版本不同，存放位置也有不同）。
- ▶ 將src.zip解壓縮後，即可看到各主類別及相關的子類別。

例如：

在java資料夾（類別）下可以看到io、land、math、...等子資料夾（類別）。

- Java Platform，Standard Edition 8 API Specification  
（oracle 原廠）的網址：

<https://docs.oracle.com/javase/8/docs/api/overview-summary.html>

# 資料來源

- 蔡文龍、何嘉益、張志成、張力元，JAVA SE 10基礎必修課，台北市，碁峰資訊股份有限公司，2018年7月，出版。
- 吳燦銘、胡昭民，圖解資料結構-使用Java(第三版)，新北市，博碩文化股份有限公司，2018年5月，出版。
- Ivor Horton，Java 8 教學手冊，台北市，碁峰資訊股份有限公司，2016年9月，出版。
- 李春雄，程式邏輯訓練入門與運用---使用JAVA SE 8，台北市，上奇科技股份有限公司，2016年6月，初版。
- 位元文化，Java 8視窗程式設計，台北市，松崗資產管理股份有限公司，2015年12月，出版。
- Benjamin J Evans、David Flanagan，Java 技術手冊 第六版，台北市，碁峰資訊股份有限公司，2015年7月，出版。
- 蔡文龍、張志成，JAVA SE 8 基礎必修課，台北市，碁峰資訊股份有限公司，2014年11月，出版。
- 陳德來，Java SE 8程式設計實例，台北市，上奇科技股份有限公司，2014年11月，初版。
- 林信良，Java SE 8 技術手冊，台北市，碁峰資訊股份有限公司，2014年6月，出版。
- 何嘉益、黃世陽、李篤易、張世杰、黃鳳梅，徐政棠譯，JAVA2 程式設計從零開始--適用JDK7，台北市，上奇資訊股份有限公司，2012年5月，出版。