補充 - 函數 (FUNCTION)

模組化程式設計

- 認識函數
- 模組的基礎

認識函數 – 說明

- 目前的軟體系統或應用程式需要大量人員參與分析、設計與開發，因此將一個大型應用程式的功能分割成一個個獨立子功能，就成為非常重要的工作，這就是模組化，模組化的最基本單位就是函數（或稱為函式）。
- 「函數」（Functions）是將程式中常用的共同程式碼獨立成程式區塊，以便能夠重複呼叫這些函數的程式碼。一般來說，函數都有傳回值，如果函數沒有傳回值，稱為「程序」（Procedures）。

認識函數－函數的結構

- 不論是日常生活，或實際撰寫程式碼時，有些工作可能會重複出現，而且這些工作不是單一程式敘述，而是完整的工作單元，例如：我們常常在自動販賣機購買茶飲，此工作的完整步驟，如下所示：
  - 將硬幣投入投幣口
  - 按下按鈕，選擇購買的茶飲
  - 在下方取出購買的茶飲

認識函數－函數的結構

- 如果只有一次到無所謂，如果幫3位同學購買果汁、茶飲和汽水三種飲料，這些步驟就需重複3次，如下所示：

認識函數－函數的結構

- 簡化的工作描述就是函數（Functions）的原型，因為我們會很自然的將一些工作整合成更明確且簡單的描述「購買??」。程式語言也是使用想同觀念，可以將整個自動販賣機購買飲料的步驟使用一個整合名稱來代表，即【購買()】函數，如下所示：

  購買(果汁)

  購買(茶飲)

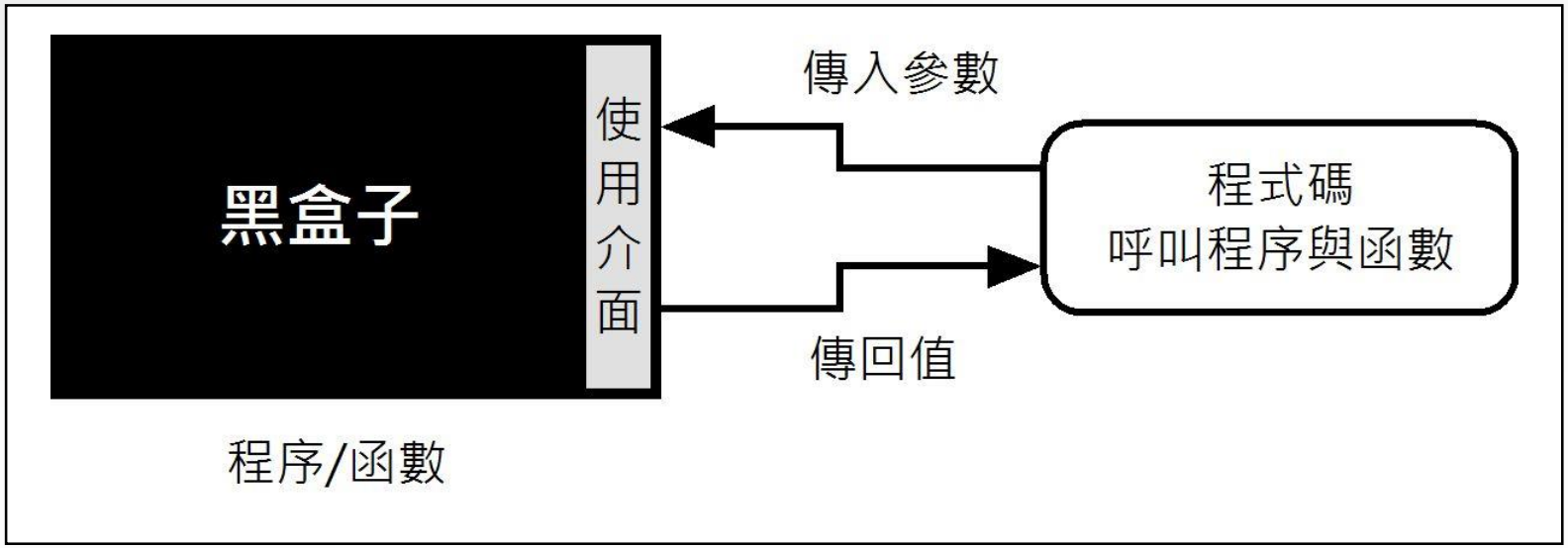
  購買(汽水)

認識函數 – 函數是一個黑盒子

- 「函數」（Functions）就是將程式中常用的共同程式碼獨立成區塊，以便能夠重複呼叫這些函數的程式碼。一般來說，函數都有傳回值，如果函數沒有傳回值，稱為「程序」（Procedures）。
- 簡單的說，函數是一個程式區塊，執行函數稱為函數呼叫（Functions Call）。在呼叫函數時，我們並不需要了解函數內部實際的程式碼，事實上，也不需要知道其細節。函數如同是一個「黑盒子」（Black Box），只要告訴我們如何使用黑盒子的「使用介面」（Interface）即可。

認識函數 – 函數是一個黑盒子

- 程式碼只知道呼叫程序/函數時，需要傳入的參數和取得傳回值。它是程序/函數和外部溝通的使用介面，一個對外的邊界，實際程式碼內容是隱藏在使用介面後，我們將程序/函數實際內容的程式碼撰寫稱為「實作」（Implementation）。



模組的基礎

- 模組化是把大型功能切割成無數子功能，至於如何切割成一個個子功能的模組，屬於結構化分析的範疇，我們最常使用的是由上而下設計方法。
- 模組化程式設計是將大程式切割成一個個小程式。以JAVA語言來說，模組化的基本單位是函數，因為模組可大可小，可能只擁有單一程序或函數，也可能是整個子功能的函數庫。
- 不過，每一個函數都可以解決一個小問題，等到所以小問題都解決了，使用程序與函數堆積成的軟體系統或應用程式也就開發完成。

建立JAVA函數

- 建立與呼叫函數
- 函數的參數列與傳回值
- 函數的參數傳遞方式

建立函數

- 函數是一個可以重複執行的程式區塊，在JAVA語言屬於類別的成員，稱為「方法」（Methods）。對於Windows應用程式來說，我們主要使用JAVA函數來建立控制項的事件處理程序和自訂功能的函數，其說明如下所示：
  - 事件處理程序：事件處理程序是處理指定事件的程序（沒有傳回值的函數），在本章前的Windows應用程式已經使用Click事件處理程序。
  - 自訂功能的函數：將程式區塊使用一個函數名稱來代替，以便呼叫函數來執行特定功能。

建立與呼叫函數-語法

- JAVA函數是由函數名稱（或稱為方法名稱）和括號括起的程式碼區塊所組成，其語法如下所示：

```
修飾子 void 函數名稱()
{
  程式敘述;
}
```

- 在上述函數宣告最前面的「修飾子」（Modifiers）可以宣告函數的存取範圍，其說明如下所示：
  - public：指出函數可以在整個JAVA專案的任何地方進行呼叫，甚至是其他類別
  - private：指出函數只能在宣告的同一個類別內進行呼叫。

7-2-1 建立與呼叫函數-範例

- 因為此函數並沒有傳回值，所以是void，在函數名稱後的括號可以定義傳入的參數列，如果函數沒有參數，就是空括號。例如：在標籤控制項顯示訊息文字的JAVA函數，如下所示：

```
public void printTitle()
{
   string msg;
   msg = “JAVA程式設計";
   System.out.println(msg);
}
```

建立與呼叫函數-呼叫函數

- JAVA語言的函數呼叫需要使用函數名稱，其基本語法如下所示：

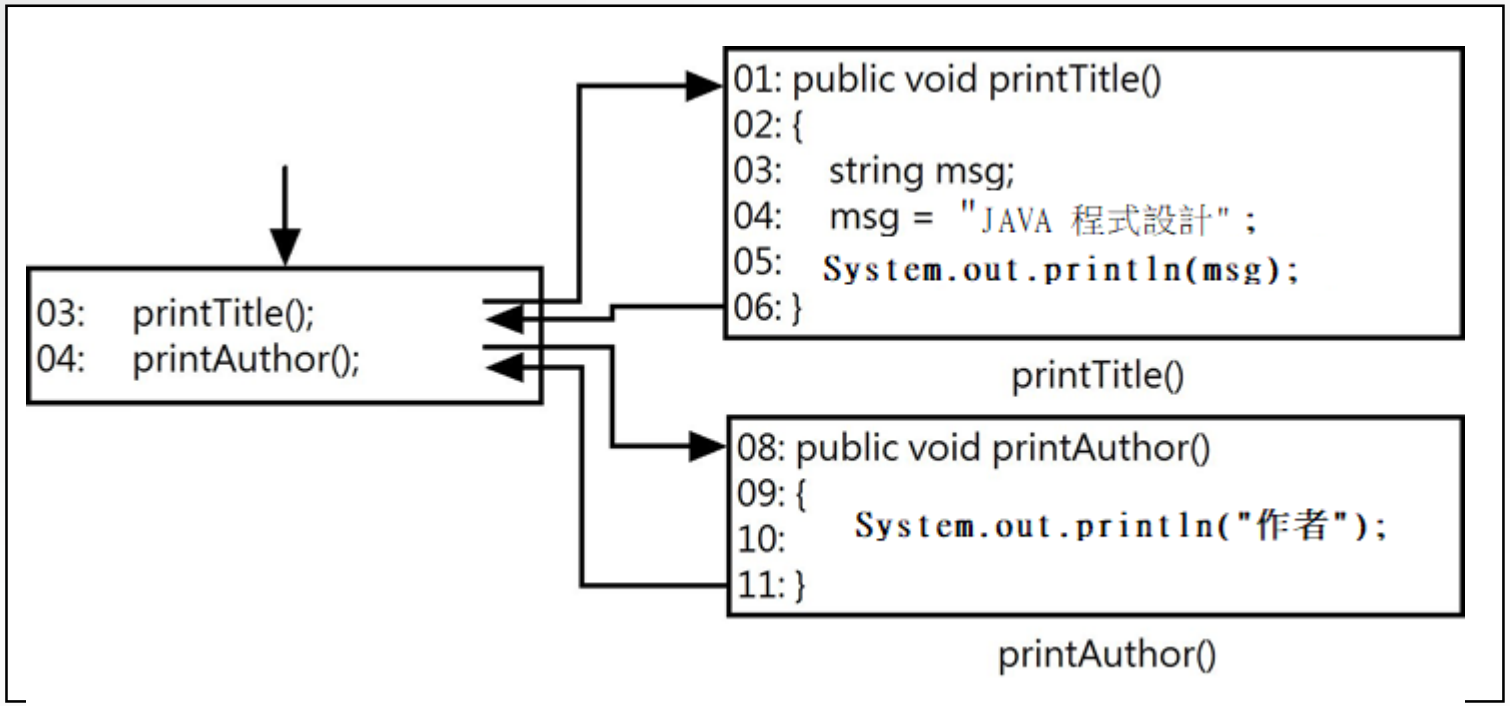
  函數名稱();

- 因為printTitle()函數沒有傳回值和參數列，所以呼叫函數只需使用函數名稱和空括號，如下所示：

  printTitle();

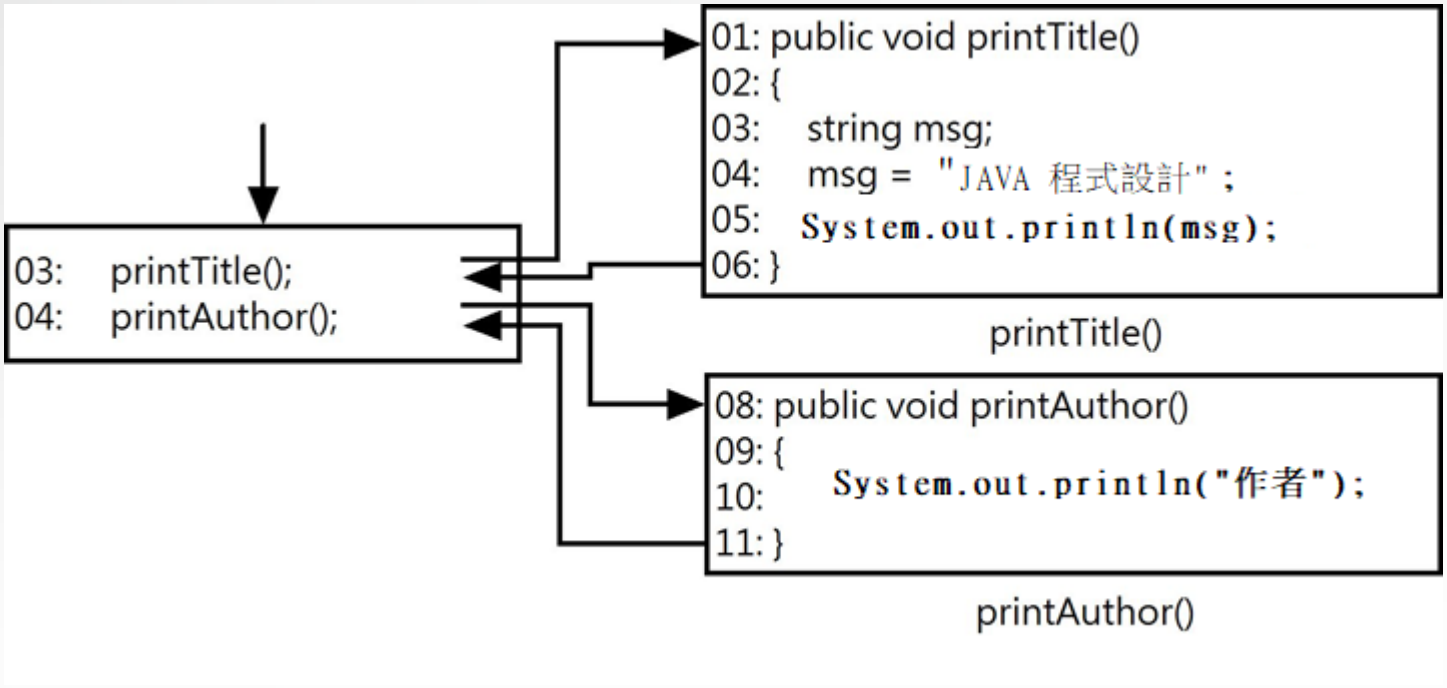
建立與呼叫函數-函數的執行過程

- JAVA 程式是如何執行函數，以本節範例程式為例，程式是在程式事件處理程序的第3列呼叫printTitle()函數，此時程式碼執行順序就跳到printTitle()函數的第1列，在執行完第6列後返回呼叫點，如下圖所示：



課堂練習:

請完成完整的函數程式



函數的參數列與傳回值-
函數的參數列(語法)

- 函數如果擁有參數列，在呼叫時就可以指定不同的參數值，換句話說，相同的函數可以得到不同的執行結果，其語法如下所示：

```
[public | private] void 函數名稱(參數1, 參數2, …)
{
  程式敘述;
}
```

- 上述函數括號內的參數稱為「正式參數」（Formal Parameters）或「假的參數」（Dummy Parameters），如果不只一個，請使用逗號分隔。正式參數是識別字，其角色如同變數，需要指定資料型態，而且可以在函數的程式區塊中使用。

函數的參數列與傳回值-函數的參數列(範例)

- 例如：顯示重複訊息的函數，如下所示：

```
public void printRepeatMsgs(string msg, int times) {
   int i;
   for (i = 1; i <= times; i++) {
       System.out.println(msg )
   }
}
```

- 上述函數擁有2個參數，因為參數不只一個，所以使用「,」符號分隔。

函數的參數列與傳回值-函數的傳回值(語法)

- C#函數開頭宣告的傳回值型別如果不是void，而是其他資料型別時，表示函數擁有傳回值，其基本語法如下所示：

```
[public | private] 傳回值型別 函數名稱(參數1, 參數2, ...)
{
  程式敘述;
  return 值 | 運算式;
}
```

- 上述函數需要使用return關鍵字傳回一個值或運算式的運算結果，函數就是執行至return關鍵字為止。

函數的參數列與傳回值-函數的傳回值(範例)

- 例如：溫度轉換函數，如下所示：

```
public double convertTemperature(int c) {
   double f;
   f = (9.0 * c) / 5.0 + 32.0;
   return f;
}
```

- 上述函數擁有1個參數c，可以將參數的攝氏溫度轉成華氏溫度，使用return關鍵字傳回華氏溫度。

函數的參數列與傳回值-
呼叫擁有參數和傳回值的函數(語法)

- 函數如果擁有參數，在呼叫時就需要指定參數列的參數值，其語法如下所示：

  函數名稱(參數列);

- 上述呼叫參數稱為「實際參數」（Actual Parameters），即參數值，它需要和正式參數定義的資料型態相同。換句話說，每一個正式參數都需對應相同型態的實際參數。

函數的參數列與傳回值-
呼叫擁有參數和傳回值的函數(範例)

- 例如：擁有參數的printRepeatMsgs()函數呼叫，如下所示：

```
printRepeatMsgs("擁有參數的函數", 2);
```

- 上述函數呼叫傳入2個使用「,」符號分隔的參數字串和整數，可以顯示2次第1個參數msg的值。
- 函數如果擁有傳回值，在呼叫時就可以使用指定敘述來取得傳回值，如下所示：

```
float temp;
temp = convertTemperature(100);
```