第三章 控制敘述

本章內容

- 3-1 選擇性 (Selection) 敘述
- 3-2 重複性 (Looping) 敘述
- 3-3 巢狀迴圈敘述
- 3-4 特殊的程式控制
- 3-5 執行特定函式 (Function)
- 3-6 變數的生存範圍

控制程式的執行流程

- 在結構化的程式設計中，程式的執行方式是循序的由第一行敘述依次往下執行，這個特性指的是程式的「**循序性**」，如同我們在上一章中看到的所有的程式的執行方式。
- 程式執行也可以依據條件的判斷而選擇執行某些敘述，並略過某些敘述的執行，這個特性指的是程式的「**選擇性**」。
- 設計程式時，也可以依據特定的條件而重複的執行某些程式碼，這個特性指的是程式的「**重複性**」。

3-1 選擇性 (Selection) 敘述

- 選擇性敘述 (或是稱為「**條件判斷敘述**」)的主要目的是讓程式可以依據不同的條件來執行不同區段的程式碼。Java 提供的判斷敘述可以分類為:
  - **單項執行敘述**:使用「if」語法來完成。
  - **雙項執行敘述**:使用「if-else」語法來完成。
  - **多項執行敘述**:使用「if-else if」或是「switch-case」語法來完成。

3-1-1 單項執行敘述：使用「if」語法

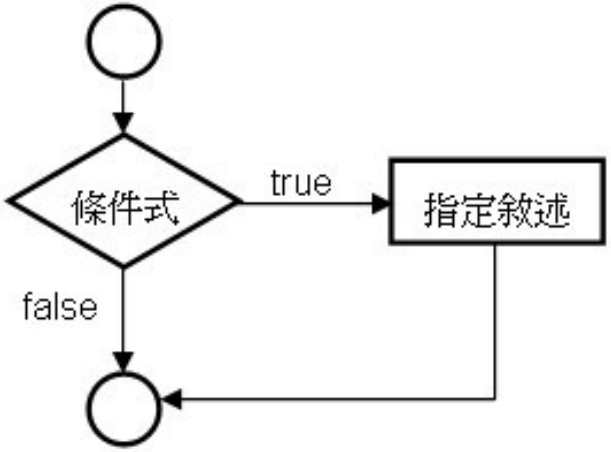
- 程式執行「if」敘述時，會依據條件式來判斷是否要執行指定的敘述。如果條件式的值為「true（成立）」，則執行指定的敘述；如果條件式的值為「false（不成立）」，則不會執行指定的敘述。「if」的使用語法如下：

```
if (條件式) {
    // 條件成立的指定敘述;
}
```

  - 特別注意的是：如果「指定敘述」只有一行敘述，則「{」和「}」可以省略。
  - 但如果「指定敘述」有多行時，則「{」和「}」不可以省略，否則，程式只會執行第一行的敘述。

3-1-1 單項執行敘述：使用「if」語法

- 「if」語法的流程圖如下：



- 程式執行時，由最上方的圓圈開始執行，在判斷「if」語法中的條件式時，如果條件式的值為「true」，則程式會執行右方的指定敘述，再執行下方的圓圈；如果條件式的值為「false」，則程式會直接執行下方的圓圈。

3-1-2 雙項執行敘述：使用「if-else」語法

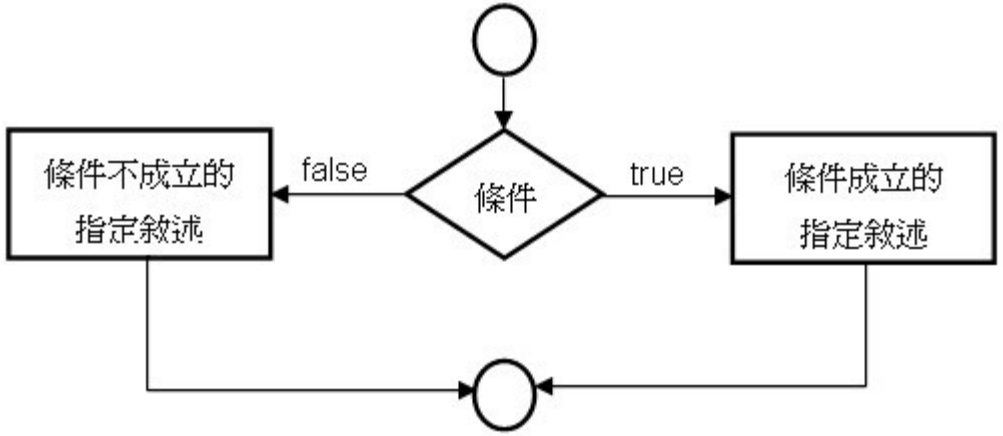
- 雙項執行敘述和單項執行敘述的功能相似，但增加了「條件式」不成立時，程式可以執行的區塊。因此，語法中增加了「else」敘述。「if-else」的使用語法如下：

```
if ( 條件式 ) {
   // 條件成立的指定敘述 ;
} else {
   // 條件不成立的指定敘述 ;
}
```

  - 和單項執行敘述「if」的使用原則相同，如果「指定敘述」只有一行敘述時，則「{」和「}」可以省略。但如果「指定敘述」有多行時，則「{」和「}」不可以省略，否則，程式只會執行第一行的敘述。

3-1-2 雙項執行敘述：使用「if-else」語法

- 「if-else」語法的流程圖如下：



- 程式執行時，由最上方的圓圈開始執行，在判斷「if」語法中的條件式時，如果條件式的值為「true」，則程式會執行右方的指定敘述。如果條件式的值為「false」，則程式會執行左方的指定敘述行。而不管條件式的值為何，程式都會繼續往下執行下方的圓圈。

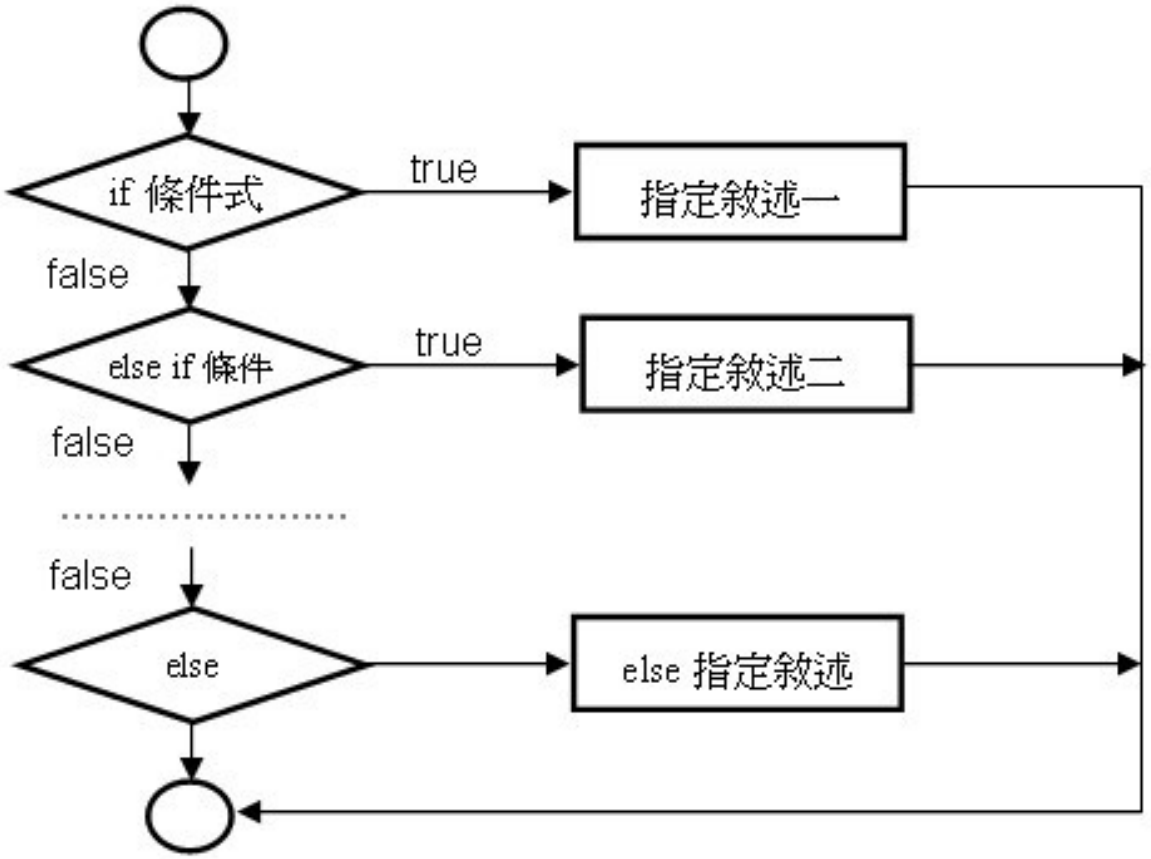
3-1-3 多項執行敘述：使用「if-else if」語法

- 如果對單項的條件式必須判斷多種不同的情況時，我們可以利用「if-else if」語法來完成。「if-else if」語法的使用方式如下：

```
if ( 條件式一 ) {
    // 條件式一成立的指定敘述 ;
} else if ( 條件式二 ) {
    // 條件式二成立的指定敘述 ;
}
.....
else {
    // 上述的條件式都不成立時的指定敘述 ;
}
```

3-1-3 多項執行敘述：使用「if-else if」語法

- 「if-else if」語法的流程圖如下：



3-1-4 巢狀的「if」敘述

- 如果程式中的「if」的指定敘述中又包含了其他的「if」敘述，則程式會形成「**巢狀 (Nested)**」的「if」判斷區塊。巢狀的「if」區塊並不容易理解，但主要的執行流程仍和非巢狀的「if」區塊相同，您的程式的架構可能如同：

```
if ( 條件式一 ) {
    if ( 條件一之 1) {
      //A： 條件式成立的指定敘述；
            }
} else if ( 條件式二 ) {
    // 條件式二成立的指定敘述；
}
```

3-1-5 多項執行敘述：使用「switch-case」語法

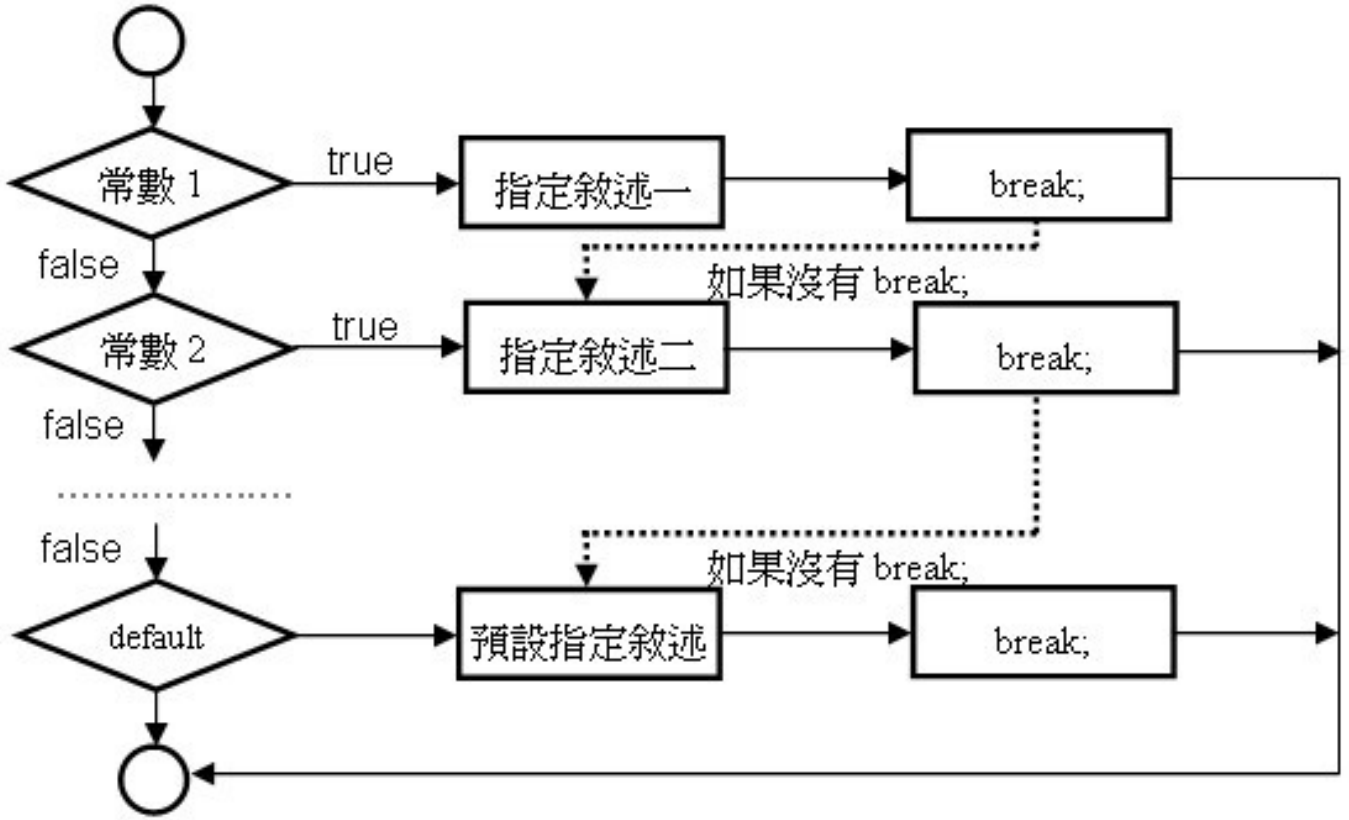
- Java 提供了另一種「switch-case」的語法，使用方式如下：

```
switch ( 變數 ) {
    case 常數一:
      // 條件式成立的指定敘述一 ;
      break;
    case 常數二:
      // 條件式成立的指定敘述二 ;
      break;
    ..............................
    case 常數 n :
      // 條件式成立的指定敘述 n;
      break;
    default :
      // 預設的指定敘述 ;
      break;
}
```

- 語法中的「break;」敘述可用來跳離區塊程式

3-1-5 多項執行敘述：使用「switch-case」語法

- 「switch」語法的執行流程圖如下：



3-1-5 多項執行敘述：使用「switch-case」語法

- 使用「default」區塊
  - 當所有的「case」後的常數值都無法滿足「switch」的變數值時，程式會執行「default」區塊中的指定敘述。
  - 您可以省略「default」區塊。
  - Java 允許您任意的調整各「case」的先後順序。
- 「case」無法判斷範圍數值
  - 每個「case」之後只能判斷一個常數值，而不能使用範圍值

3-2 重複性 (Looping) 敘述

- 重複性敘述的主要目的是讓程式可以重複的執行某一區段中的程式碼。 Java 提供的重複性敘述可以分類為:
  - 使用「 for 」迴圈敘述。
  - 使用「 while 」迴圈敘述。
  - 使用「 do-while 」迴圈敘述。

3-2-1 使用「for」迴圈敘述

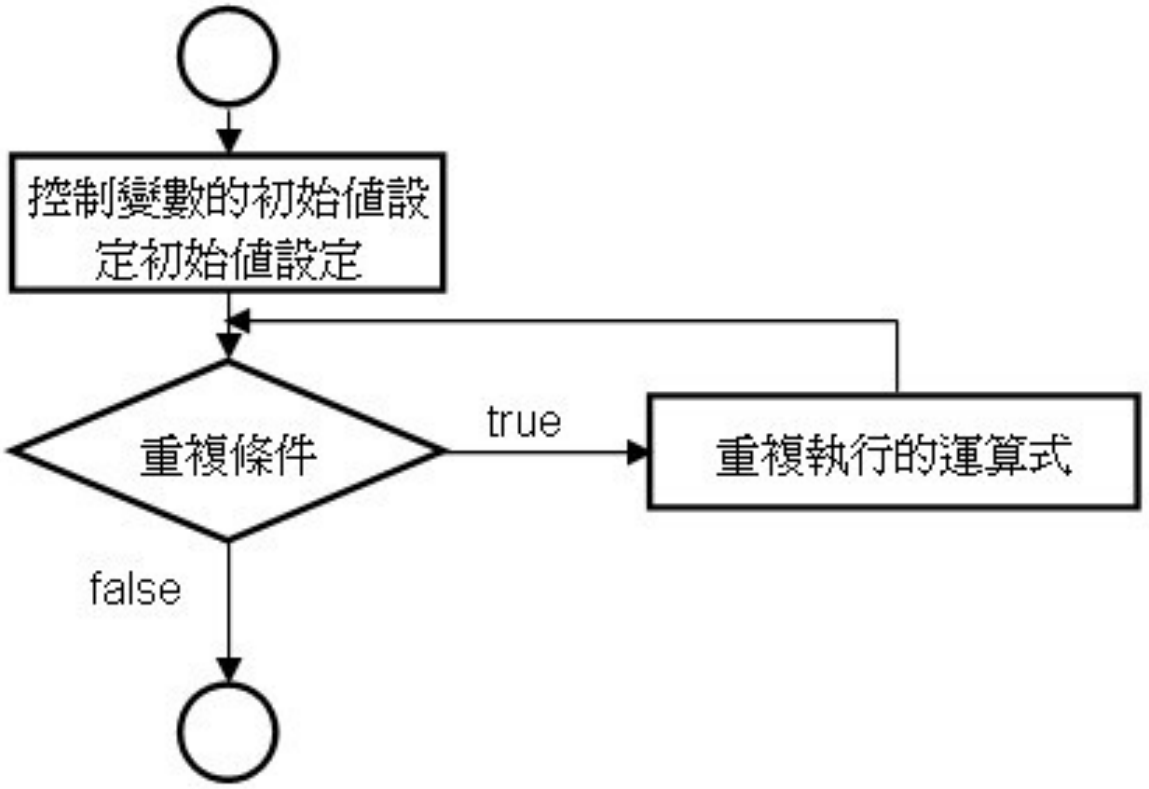
- 「for」的使用語法如下:

```
for ( 控制變數的初始值設定 ; 重複的條件 ; 控制變數值的改變 ) {
        // 重複執行的運算式;
}
```

  - 特別注意的是: 如果「重複執行的運算式」只有一行敘述, 則「{」和「}」可以省略。但如果「重複執行的運算式」有多行時, 則「{」和「}」不可以省略, 否則, 程式只會執行第一行的敘述。

3-2-1 使用「for」迴圈敘述

- 「for」語法的流程圖如下：



3-2-1 使用「for」迴圈敘述

- 請特別注意「for」敘述的執行程序:
  - **執行「初始值設定」區段:**
    - 迴圈敘述開始時，會先執行這個區段，此區段用來設定迴圈控制變數的初始值，這個區段在整個迴圈的執行過程中，只執行一次。
  - **判斷「重複的條件」區段:**
    - 每一次需要執行迴圈的「重複執行的運算式」區段前，都必須執行這個區段，此區段用來判斷迴圈是否可以繼續執行，這個區段在整個迴圈的執行過程中會重複的執行。
  - **「重複執行的運算式」區段:**
    - 當「重複的條件」判斷成立時，程式會執行此一區段的內容，這個區段在整個迴圈的執行過程中會重複的執行。
  - **「控制變數值的改變」區段:**
    - 當「重複執行的運算式」區段執行完成後，程式會執行此一區段的內容，這個區段會改變迴圈控制變數的值。改變之後，再重複執行「步驟二、三、四」，直到「重複的條件」區段無法成立為止

3-2-1 使用「for」迴圈敘述

- 「for()」區段的不同寫法
  - 在程式寫作時，我們有時候也會先行設定迴圈控制變數的初始值，同時省略「for()」區段中的初始值的設定。
- 範例程式如下：

```
// 迴圈敘述中省略了變數初始設定的區塊

for (; i<10 ; i+=2) {
      System.out.println(" 執行時： i = " + i);
         // 觀察 i 值的變化

}
```

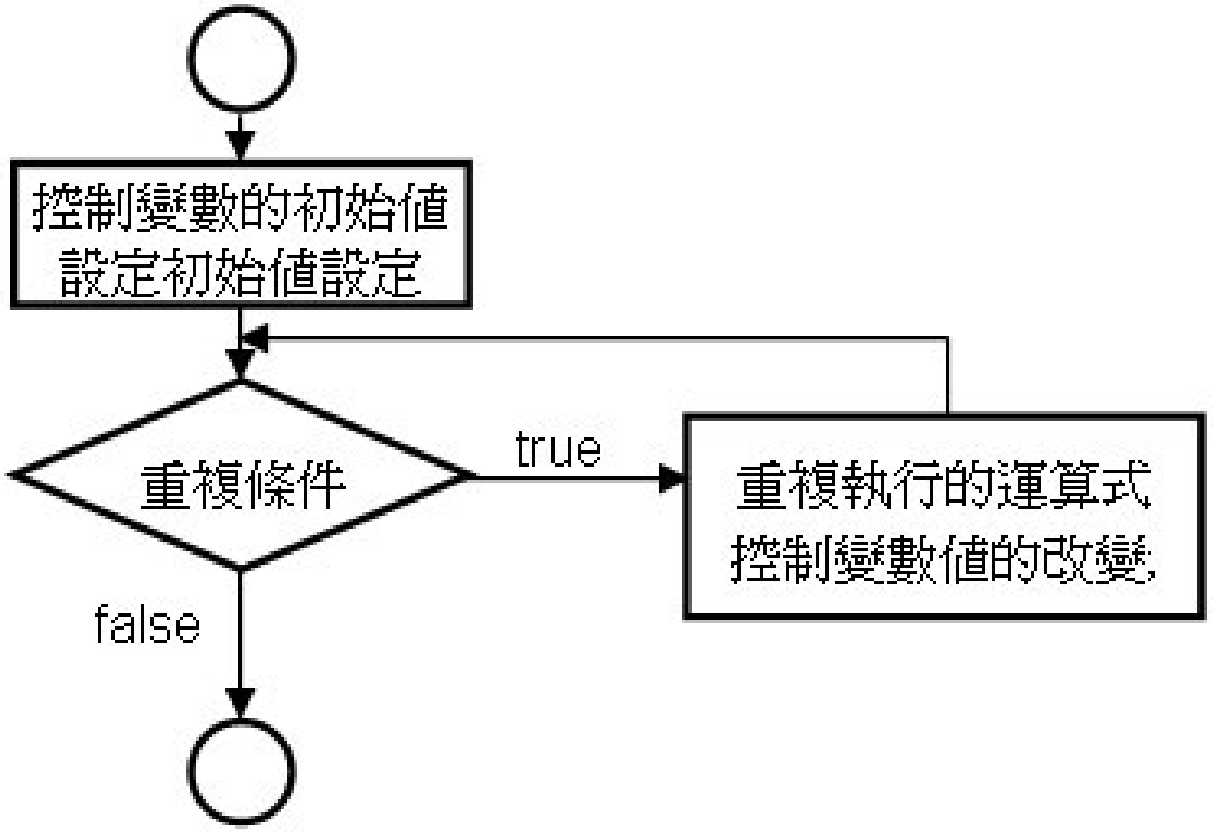
3-2-2 使用「while」迴圈敘述

- 「for()」迴圈敘述一般是用於可預測執行次數的情況下，
  - 例如：使用「for()」敘述來讀取資料檔案中某一筆記錄的各欄位內容。
- 而「while()」和「do-while」敘述多半是用於不可預測執行次數的情況下，
  - 例如：使用「while」敘述來讀取資料檔案中記錄的筆數。
- 「while」的使用語法如下：

```
迴圈控制變數初始值的設定；
while ( 重複的條件 ) {
    重複執行的運算式；
    控制變數值的改變；
}
```

3-2-2 使用「while」迴圈敘述

- 「while」語法的流程圖如下:



3-2-2 使用「while」迴圈敘述

- 「while」敘述的執行程序:
  - **執行「迴圈控制變數初始值的設定;」區段:**
    - 圈敘述開始時，會先執行這行敘述，此敘述置於「while」迴圈敘述的區段之外，並用來設定迴圈控制變數的初始值，這個區段在整個迴圈的執行過程中，只執行一次。
  - **判斷「重複的條件」區段:**
    - 每一次需要執行迴圈的「重複執行的運算式」區段前，都必須執行這個區段，此區段用來判斷迴圈是否可以繼續執行，這個區段在整個迴圈的執行過程中會被重複的執行。
  - **「重複執行的運算式」區段:**
    - 當「重複的條件」判斷成立時，程式會執行此一區段的內容，這個區段在整個迴圈的執行過程中會重複的執行。
  - **「控制變數值的改變」:**
    - 當「重複執行的運算式」區段執行完成後，程式會執行此一區段的內容，這個區段會改變迴圈控制變數的值。改變之後，再重複執行「步驟二、三、四」，直到「重複的條件」區段無法成立為止。

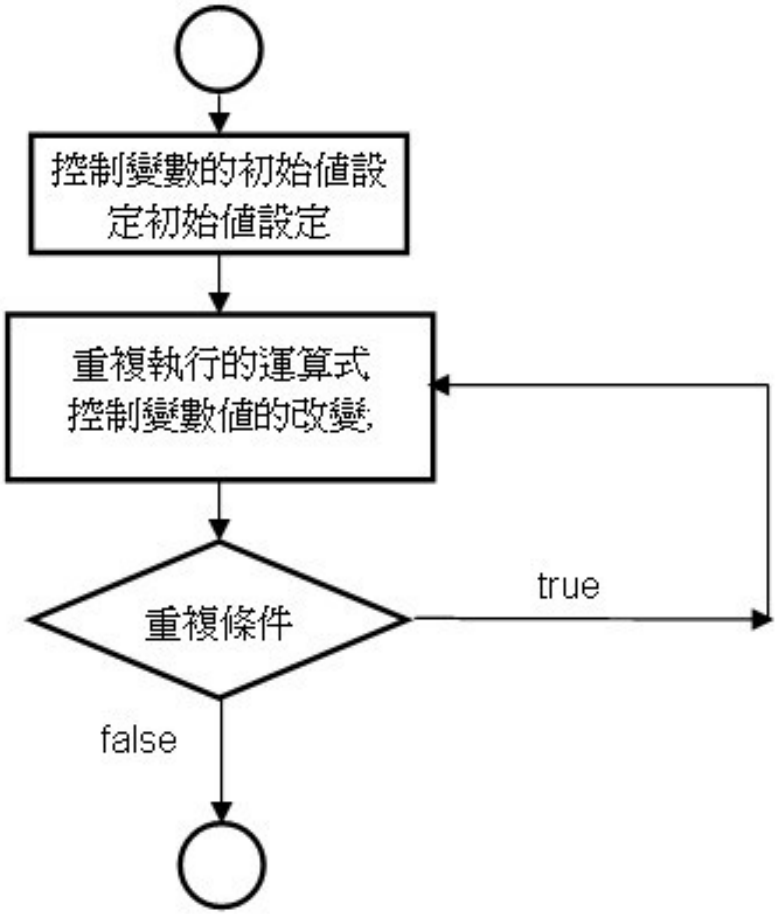
3-2-3 使用「do-while」迴圈敘述

- 「while()」敘述是在進入迴圈本體前即先行判斷是否滿足迴圈的執行條件。因此，迴圈本體可能不會執行。
- 「do-while」迴圈敘述則不同。不論控制變數的初始值為何，迴圈本體中的敘述至少會被執行一次。
- do-while」的使用語法如下：

```
迴圈控制變數初始值的設定；
do {
   // 重複執行的運算式；
   // 控制變數值的改變；
}while ( 重複的條件 )；
```

3-2-3 使用「do-while」迴圈敘述

- 「do-while」語法的流程圖如下：



3-2-3 使用「do-while」迴圈敘述

- 「do-while」敘述的執行程序:
  - **執行「迴圈控制變數初始值的設定;」區段:**
    - 迴圈敘述開始時，會先執行這行敘述，此敘述置於「do-while」迴圈敘述的區段之外，並用來設定迴圈控制變數的初始值，這個區段在整個迴圈的執行過程中，只執行一次。
  - **「重複執行的運算式」及區段:**
    - 接著執行此一區段的內容，這個區段在整個迴圈的執行過程中會重複的執行。
  - **「控制變數值的改變」區段:**
    - 當「重複執行的運算式」區段執行完成後，程式會執行此一區段的內容，這個區段會改變迴圈控制變數的值。改變之後，再重複執行「步驟二、三、四」，直到「重複的條件」區段無法成立為止。
  - **判斷「重複的條件」區段:**
    - 每一次需要再執行迴圈的「重複執行的運算式」區段前，都必須執行這個區段，此區段用來判斷迴圈是否可以繼續執行，這個區段在整個迴圈的執行過程中會被重複的執行。

3-3 巢狀迴圈敘述

- 如果將迴圈的語法合併使用，例如：「for()」敘述中再包含另一個「for()」，甚至是「while」敘述，程式就形成了巢狀迴圈的架構了。
- 數學運算中，使用巢狀迴圈的最典型的範例是列印出九九乘法表的內容，例如：

```
for (i=1; i<= 9 ; i++) {   // 注意執行後， i 的值
    for (j = 1; j <= 9; j++) {
            System.out.print(i + "*" + j + "=" + i * j + "\
    t");
    }
    System.out.println();
}
```

3-4 特殊的程式控制

- 除了上述介紹的迴圈執行語法之外， Java 也提供了一些特殊的控制敘述，這些控制敘述可以進一步的控制迴圈的執行流程。 Java 提供的控制性敘述可以分類為：
  - **「break」敘述**：用於跳離迴圈區塊。
  - **「continue」敘述**：跳過迴圈本次的執行。
  - **label 的使用**：標示程式控制權可以跳入的區段名稱。

3-4-1 「break」敘述

- 「break」敘述可以用於終止迴圈的執行，並強迫跳離迴圈區段。或是跳離判斷式的執行區塊。
- 「break」的使用範例如下：

```
int i = 1,j = 5;
do {
      if(i>j) {
           break;
      }
      j--;
      i++;
} while (i <5);
System.out.println(" 程式執行後， i = " +i+" ， j = "+j);
```

3-4-2 「continue」敘述

- 「continue」敘述會使程式略過「continue」敘述後的其他敘述的執行，並將程式執行權交還給迴圈控制敘述，讓迴圈控制敘述敘述決定是否要繼續執行迴圈。「continue」的使用範例如下：

```
int i = 1,j = 5;
do {
    if(i<j) {
        j--;
        System.out.println(" 程式時， i = " + i + " ， j = " + j);
        continue;
    }
    i++;
} while (i <5);
System.out.println(" 程式執行後， i = " + i +" ， j = " + j);
```

3-4-3 label 的使用

- Java 程式中並不包含行號的識別，所以，程式中並不能像早期的 Basic 程式一樣直接跳至某一行執行。
- 但 Java 提供了「label」的標示機制，讓我們可以在程式中自定程式區塊的「label」，並且可以直接跳至此一區塊來執行程式。「label」的使用語法如下：

  label-name : statement

- 定義標記名稱後，需加上「：」符號，再接程式區塊。
- 雖然標記可以放在程式的任何一個位置，但標記多半是和迴圈敘述搭配使用，其主要的目的是在讓程式在跳離迴圈敘述時，能更清楚的知道程式執行的位置。
- 但需注意的是：設定的標記名稱和迴圈區段之間不能再有其它的敘述。

3-5 執行特定函式 (Function)

- 除了利用選擇性敘述或是迴圈敘述來改變程式的執行流程式外，我們還可以利用呼叫函式的方式來改變程式的執行流程。
- 函式是指將特定功能的程式敘述組成一個區塊，並定義區塊名稱，而這個程式區塊可以被重複的使用。
- 函式的定義語法如下：

```
[ 型態 ] 函式名稱 ([ 參數 , ]) {
    函式主體敘述 ;
    return 回傳值 ;
}
```

3-5 執行特定函式 (Function)

- 以函式的定義語法而言，如果您要定義一個不具有參數，及不具有回傳值的函式，您可以如此定義:

```
public static void Test1() {
    // 相關的程式碼
}
```

  - 定義中的 void 代表是不具有回傳值的函式。

- 如果您要定義一個具有兩個 int 參數，並具有 int 回傳值的函式，您可以如此定義:

```
public static int Test1(int value1, int value2) {
    // 相關的程式碼
    return value;
}
```

  - 定義中，函式名稱 Test1 之前的 int 代表回傳的型態為 int，函式中要使用「return」來回傳 value 值。

3-5-1 設計第一個函式

- 例如：如果我們要計算不同半徑的圓的面積，假定圓面績的計算方式為： 3.14*r2 ，其中「 r 」代表半徑。這樣的函式在設計時，需要設計一個傳入半徑的參數，並回傳計算的結果。
- 因此：函式可以定義成：

```
public static double CArea(int r){
    double a;
    System.out.println(" 圓的半徑是： " + r);
    a = 3.14 * r * r;

    return a;
}
```

3-5-2 使用遞迴

- 在程式設計中，如果函式可以自我呼叫，這種方式叫做「**遞迴 (recursion)**」。例如以下的程式碼:

```
void recursive() {
   //some code here
   recursive();
}
```

- 如果您在程式中呼叫一次函式時， Java 會自動幫函式中的變數配置一份記憶體空間。因此如果您重覆的呼叫某個函式， Java 就會不斷的配置空間給變數。
- 這種情況在撰寫遞迴函式時相當的危險。因為如果您無法正確的設定函式停止執行的條件，該函式在重覆呼叫的情況下，系統會產生 StackOver 的錯誤。

3-5-3 使用「varags」設計函式的參數 — J2SE 5.0

- 設計函式時，參數的個數必需是在宣告函式時就預先決定，使用函式時，Java 才會根據呼叫函式的名稱及參數的內容來決定要使用那一組函式。
- 因此，假使您需要使用到一個將多個整數值相加的函式，而您想要這麼使用：

```
useMe(1, 2);
useMe(2, 3, 4);
useMe(4, 5, 6, 7);
```

- 那麼，您可能要為這樣的使用方式來宣告以下的函式：

```
public static void useMe(int v1, int v2){// 相加的程式碼  }
public static void useMe(int v1, int v2, int v3){// 相加的程式碼
   }
public static void useMe(int v1, int v2, int v3, int v4){// 相加的
   程式碼  }
```

3-5-3 使用「varags」設計函式的參數 — J2SE 5.0

- J2SE5.0 提供新的「不定個數參數 (Variable-Length Arguments), varags」的功能，讓您可以在宣告函式時，使用「...」來代表未知個數的參數。
- 宣告的格式如下：

```
[ 型態 ] 函式名稱 ( 型態 ... 參數名稱 ) {
    函式主體敘述 ;
    return 回傳值 ;
}
```

- 因此，我們可以將「useMe」函式的宣告改寫為：

```
public static void useMe(int... value){// 相加的程式碼 }
```

3-5-3 使用「varags」設計函式的參數 — J2SE 5.0

- 雖然 varags 相當好用，但是，在使用上仍然會有一些限制。例如：
  - **varags 參數的宣告只能是最後一個參數。**
    因此，以下的宣告都是合法的：
    ```
    public int add(int... value) {}
    public int add(double v1, int... value) {}
    public void add(long l, short s, int... value){}
    ```
    但是：以下就不是合法的使用方式了：
    ```
    public int add(int... value, int a){}          //varags 後面還有別的參數宣告
    public int add(int... value, long... l){}      // 兩個 varags 並存
    ```
  - **方法覆載時， varags 不可以和陣列參數共存。**
    因為 varags 使用的仍然是陣列，因此，以下的兩個宣告方式在方法覆載時，會被視為同一個方法：
    ```
    public int add(int... value){}
    public int add(int[] value){}
    ```

3-5-4 與算術相關的類別函式

- Java 的類別庫中提供了一些關於數學運算的函式可供我們在寫作程式時使用。這些函式被定義在「java.lang.Math」類別中。

名稱	功能	範例	結果
abs	計算絕對值	Math.abs(-5)	5
max	取得兩數的最大值	Math.max(1.2, 3.4)	3.4
min	取得兩數的最小值	Math.min(2, 5)	2
pow	次方	Math.pow(2, 3)	8.0
sqrt	開根號	Math.sqrt(9)	3
round	小數四捨五入	Math.round(3.7)	4
ceil	無條件進位	Math.ceil(3.2)	4.0
floor	無條件捨去	Math.floor(3.8)	3.0
random	取得 0<=x<1 的亂數	Math.random()	

3-6 變數的生存範圍

- 而且，變數在程式碼中也會有可以被使用的範圍，我們稱為變數的「**生存範圍**」。
- 變數的生存範圍是以區塊來決定的，而 Java 中是以大括號「{ }」來代表一個特定的區塊。
- 變數存取的主要原則是:
  - 外圍區塊中的變數可以被內圍區塊中的程式碼所使用;
  - 而內圍區塊中的變數是不可以被外圍區塊中的程式碼所使用。

3-6 變數的生存範圍

- 變數的生存範圍示意圖:

```
01 : class Test {
02 :     int v1 = 0;
03 :     static void method1(){
04 :         int v2 = 0;
05 :     }
06 :     static void method2(){
07 :         int v3 = 0;
08 :         for (int i = 0; i<10; i++){
09 :             //其他的程式碼;
10 :         }
11 :     }
12 :     public static void main(String[] args){
13 :         int v4 = 0;
14 :         //其他的程式碼;
15 :     }
16 : }
```

3-6-1 相同名稱的變數的使用

- 如果變數是屬於類別層級的變數，您可以在區塊中使用和類別層級相同的變數名稱。
- 只不過，在區塊中如果宣告和類別層級相同名稱的變數，則區塊中使用到的變數會是在區塊中自行宣告的變數。
- 但如果變數是宣告在函數之中，則您不可以在同一個區塊中重覆的宣告和外圍區塊中相同名稱的變數。
- 但是，如果兩個區塊不具有包含關係時，您是可以在不同的區塊中使用相同的名稱來設定變數，而且，在使用變數時不會互相的干擾。

3-6-2 變數的事先參考

- 「**事先參考**」是指變數的使用在於宣告之前，例如以下的程式碼會產生編譯錯誤:

```
int v1;
v1 = v2;
int v2 = 10;
```

- 第 2 行的程式就是所謂的「事先參考」。即使在第 3 行又宣告了 v2 變數，但這種使用方式仍是不允許的。您必需將程式碼修改為:

```
int v1;
int v2 = 10;
v1 = v2;
```