Java 使用重複結構

迴圈結構簡介

程式碼裡的迴圈結構(loop constructs)，指的就是使用特定條件(expression)，在滿足時即重複某些行為(code block)。可以分成三種主要型態:

1. while 迴圈

   若滿足expression=true時將持續進行

2. do/while 迴圈

   執行一次後，若滿足expression=true時將持續進行

3. for 迴圈

   重複特定次數

使用while 迴圈

語法 :

```
while (boolean_expression) {
    code_block;
} // 滿足boolean_expression時，會反覆執行 code_block
// 迴圈結束後，將繼續其他程式區段
```

範例 :

```
public class While {

    public static void main(String[] args) {
         double money = 500;
         double interest = 0.18;
         int years = 0;
         while (money <=1000) {
             money += money * interest;
             years++;
             System.out.println("Year "+ years + ": "+ money);
         }
    }
}
/*
 * Output:
 * Year 1: 590.0
 * Year 2: 696.2
 * Year 3: 821.5160000000001
 * Year 4: 969.3888800000001
 * Year 5: 1143.8788784
 */
```

使用for迴圈

語法 :

for (initializer [, initialize] ; boolean_expression ; update [, update]) {

code_block;

}

// Initialize : 初始條件

// boolean_expression : 滿足條件

// update : 變動條件

while 迴圈 V.S for 迴圈

範例 1 :

```
public class WhileFor1 {

    public static void main(String[] args) {
          int i = 0;
          while(i<7) {
              System.out.println("$");
              i++;
              }

    }

}
/*
 * Output:
 * $
 * $
 * $
*/
```

範例 2 :

```
public class WhileFor2 {

    public static void main(String[] args) {
        for(int i =0; i <7 ; i++) {
            System.out.print("$");
        }
    }

}
```

```
/*
 * Output:
 * $$$$$$$
 */
```

其中：

1. 合併多個「滿足條件」後必要為true 才能發動
2. 「初始條件」和「變動條件」不必然只能一個，也可以多個，以「,」區隔

```
public class WhileFor3 {

    public static void main(String[] args) {
        for(String i = "$", t ="~" ; i.length() < 5 ; i +="$", t += "~") {
            System.out.println(i + t);
        }
    }

}

/* Output :
 * $~
 * $$~~
 * $$$~~~
 * $$$$~~~~
*/
```

使用巢狀迴圈

巢狀迴圈(nested loop)，就是迴圈裡面還有迴圈

```
public class Triangle {
    public static void main(String[] args) {
       int num = 5;
       for(int i = 0 ; i < num; i++) {
           for(int j = 0; j <= i; j++) {
               System.out.print('A');
           }
           System.out.println();
       }
    }
}
/* Output :
 * A
 * AA
 * AAA
 * AAAA
 * AAAAA
 */
```

使用 for 迴圈存取陣列

1. 使用 for 迴圈的進階型

for 迴圈的進階型 (enhanced) 可用於存取 Java 的集合物件(Collection)和陣列(Array)

語法 :

```
for (declaration : expression) {
    code_block;
}
// declaration : 宣告集合物件(Collection)或陣列 (Array) 的成員型態
// expression : 欲存取的集合物件 (Collection) 或陣列 (Array)變數
```

相較於過去基本型 for 迴圈，好處是不用理會陣列或集合物件長度，也不需要index，Java會自動將「每個成員」依「程式碼區塊 (code block)」的指示輪流處理

範例 1 :

```
public class EnhancedLoopArray {

    public static void main(String[] args) {
        int [] intArray = {12,23,45,3,65,87,22};
        for(int element : intArray) {
            System.out.println(element);
        }
    }
}

/* Output:
 * 12
 * 23
 * 45
 * 3
 * 65
 * 87
 * 22
 */
```

範例 2 :

```
import java.util.ArrayList;

public class EnhancedLoopArrayList {

    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<>();
        names.add("jim");
        names.add("bill");
```

```
        names.add("albert");
        names.add("elsa");
        for(String name : names) {
            System.out.println(name);
        }
    }

}

/* Output :
 * jim
 * bill
 * albert
 * elsa
*/
```

2. 使用 break 和 continue 敘述

break 和 continue 敘述經常搭配迴圈使用，目的在改變迴圈流程 :

1. 使用break 敘述結束迴圈，break 敘述所在的區塊程式碼，在 break敘述後將不執行
2. 使用 continue 敘述將導致流程回到迴圈內的起始點繼續執行， continue 敘述所在的區塊程式碼，在 continue 敘述後將不執行

範例 1: 使用 break 敘述

```
public class Break {

    public static void main(String[] args) {
        int passScore = 60;
        int [] scores = {40,36,52,58,65,34,93};
        int passAt = 0;
        for (int s: scores) {
            passAt ++;
            if(s > passScore) {
                break;
            }
        }
        System.out.println("Finally pass at : "+ passAt);
    }

}

/* Output :
 * Finally pass at : 5
 *
 */
```

範例 2 : 使用 continue 敘述

```
public class Continue {

    public static void main(String[] args) {
        int passScore = 60;
        int [] scores = {40,36,52,58,65,34,93};
        for(int s :scores) {
            if(s > passScore) {
                continue;
            }
            System.out.println("the score: " + s + " is failed to pass.");
        }

    }
}

/* Output :
 * the score: 40 is failed to pass.
 * the score: 36 is failed to pass.
 * the score: 52 is failed to pass.
 * the score: 58 is failed to pass.
 * the score: 34 is failed to pass.
*/
```

使用do /while 迴圈

語法 :

```
do {
    code_block;
} while (boolean_expression) ;
// 注意結尾加上「;」
```

do /while 迴圈和 while 迴圈不同 :

1. while 迴圈要開始執行時，就必須滿足條件，又稱「前測試迴圈」
2. do /while 迴圈則是至少可以執行一次，第一次之後就需要檢查條件，等同於跑完一次之後才進行條件測試，又稱「後測試迴圈」

範例 1 :

```
public class TestDoWhileLoop {

    public static void main(String[] args) {
        int count =0;
        do {
            System.out.println("DoWhile Count is : " + count);
```

```
        }while(count <0);

    }

}

/* Output :
 * DoWhile Count is : 0
 */
```

範例 2 :

```
public class TestWhileLoop {

    public static void main(String[] args) {
        int count  = 0;
        while(count < 0) {
            System.out.println("While Count is : " + count);
            count ++;
        }

    }

}

/* Output :
 * 沒東西，因為count沒有小於0
 */
```

比較迴圈結構

迴圈種類	執行次數
while	執行 0 到多次
do/while	執行 1 到多次
for	執行預先定義的次數