

多執行緒

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

前言

■ 「作業系統（Operating System）」是一套系統軟體。

- ◆ 是用來管理電腦硬體與軟體資源的程式。

- ◆ 是電腦系統的核心。

例如：

DOS、Windows、Linux、Macintosh OS、...等。

■ 作業系統的主要功能之一，便是分配系統資源。

例如：

中央處理器、記憶體、磁碟、...等的分配、使用。

■ 作業系統將正在執行的應用程式，稱為「行程（Process）」。

- 應用程式被執行時，作業系統會按照需求，配置「系統資源（包含CPU的執行時間）」給此「行程」，且會「完整地」佔用「全部」的「系統資源」。
- ◆ 應用程式被執行時，「行程」與「系統資源」是各自獨立的，應用程式間才不會相互干擾。
- ◆ 若此時必須要執行第二個應用程式時，作業系統會先暫停（也可稱之為「休眠」）第一個應用程式的執行動作，空出「系統資源」給第二個應用程式。
- ◆ 當第二個應用程式執行完畢後，如有需要再由暫停點開始，繼續執行第一個應用程式，或給第三個應用程式執行。

多工

■ 目前電腦的作業系統都是多工（Multi Task）的作業系統。

◆ 也就是允許兩個以上的應用程式同時執行。

■ 多工作業系統的原理：

◆ 由於CPU的處理速度快，便可以將CPU的執行時間切割成很多個「時間片段」。

◆ 讓CPU在某個「時間片段」，執行某個「行程」；下一個「時間片段」，執行另一個「行程」。

◆ 由於「時間片段」是一個很短的時間，只要切換速度夠快（快到人類無法感覺），便會使得每個「行程」像是同時在進行處理。

◆ 如此便能同時執行多個不同應用程式。

執行緒

- 「執行緒 (Thread)」是作業系統能夠進行運算排程的最小單位。
 - ◆ 它被包含在「行程」之中（即一個「行程」可以包含多個「執行緒」）。
 - ◆ 是「行程」中，實際運作的單位。
 - ◆ 一條「執行緒」指的是「行程」中一個單一順序的「控制流」。
 - ◆ 一個「行程」中，至少會有一個「執行緒」。
- 個人電腦只有一顆CPU，程式透過快速切換的方式，雖可處理「多個」「執行緒」，但是，「同一個時間點」，還是只能有「一個」「執行緒」是處於「執行」的狀態。

執行緒

- 支援「超執行緒（Hyper-Threading，HT）」的 CPU，即是以此方式來達到「多個」「執行緒」，「感覺上」「同時」處於「執行」的狀態。
 - ◆ 可以提升15%~30%的執行效能。

多執行緒

■ 在多個CPU或多核心的CPU上，為了提高程式的執行效率，會使用多執行緒技術。

例如：

- 將一個「行程」，分割成「多條」「執行緒」，此時，「多條」「執行緒」可以真正的「並行」，且每條「執行緒」會「執行不同的任務」。

例如：

可以把「行程」中負責「I/O處理」、「人機互動」、...等，常被阻塞的部分與「密集計算」的部分「分開」來執行，進而提高了程式的執行效率。

注意：

- ◆ 同一「行程」中的「多條」「執行緒」將共享該行程中的全部系統資源。

多執行緒

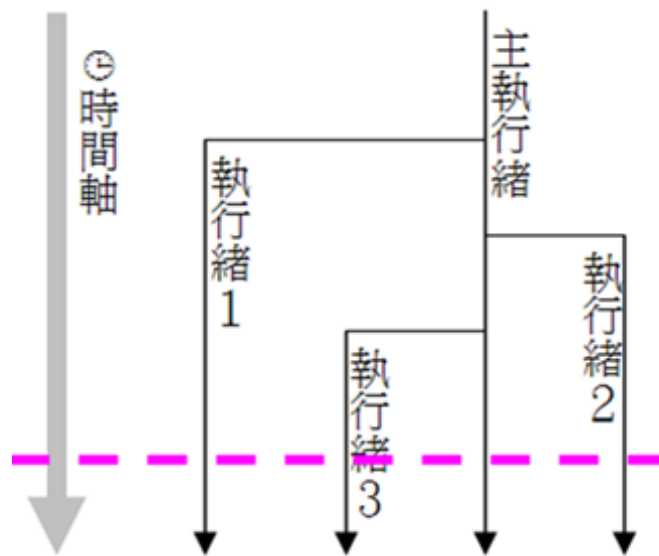
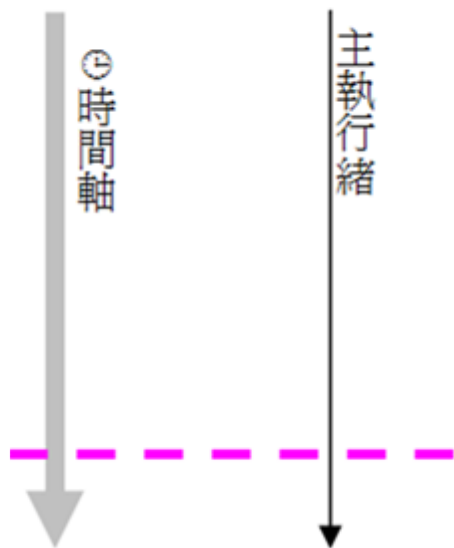
- ◆ 同一「行程」中的「每個」「執行緒」，都有各自的「呼叫棧（Call Stack）」、暫存器環境（Register Context）、本地儲存（thread-local storage）、...等。

■ Java允許一個程式能同時執行兩個（含）以上的執行緒。

- ◆ Java是經由JVM來執行程式；
- ◆ 程式執行的起點是由main()方法開始；
- ◆ 隨程式內容來作運算、判斷、再判斷...；
- ◆ 一直到程式結束為止。

多執行緒

- Java程式開始執行時會有一個執行緒開始執行，此執行緒稱為程式的主執行緒（如左圖）。
- 在Java語言中允許同時有多個程式的動作一起執行，除主執行緒外，每個動作也都是一個執行緒，這也就是多執行緒的執行（如右圖）。



多執行緒

- 多執行緒若運用得當，可大幅提升效能；若分配不當可能比單一執行緒更沒有效率，使用多執行緒時要多注意資源配置。

執行緒的生命週期

■ 在Java中執行緒其實就是Thread物件。

■ 從建立一條新執行緒到消滅過程（執行緒生命週期）（如下圖）：

◆ New（起始）：

- 是新執行緒建立的狀態。
- 當新執行緒建立完成後，就進入「可執行」狀態。

◆ Runnable（可執行）：

- 透過Thread物件的start()方法，可以啟動執行緒，指定資源給它或轉換成「執行中」狀態。
- 每一個啟動後的執行緒，都一定會去執行Thread物件的run()方法，所以run()方法是執行緒的執行起點。

執行緒的生命週期

- 程式中，若使用yield()方法，會將執行權讓出，重回「可執行」中等候排程。

◆ Running（執行中）：

- 進入此狀態的各個執行緒，確實的執行時間，是由排程器（Scheduler）根據執行緒的優先權，以及其它執行緒的活動而定，程式設計師無法自行決定。
- 如果執行緒的工作未完成，但所分配的時間已到，會重回「可執行」中等候排程器排程。

◆ Blocked（封鎖）：

- 呼叫Thread物件的sleep()方法，會讓出使用權，同時暫停該執行緒所有應執行的工作，且不再使用CPU資源，然後到「封鎖」中，進入「休眠」的狀態。

執行緒的生命週期

- 呼叫sleep()方法時，必須傳入要休眠的時間，等待休眠時間一到，Thread物件便會從「休眠」狀態再轉回「可執行」狀態。
- ◆ Wait pool（等待池）：
 - 處於「執行中」狀態下的Thread物件，若呼叫wait()方法時，便會讓出使用權，然後到「等待池」中，進入「等待」狀態。
 - wait()方法是繼承自Object物件，並非來自Thread物件。
 - 使用notify()、notifyAll()等方法，可以喚醒在「等待池」等待的執行緒，移動到「鎖定池」等候。

執行緒的生命週期

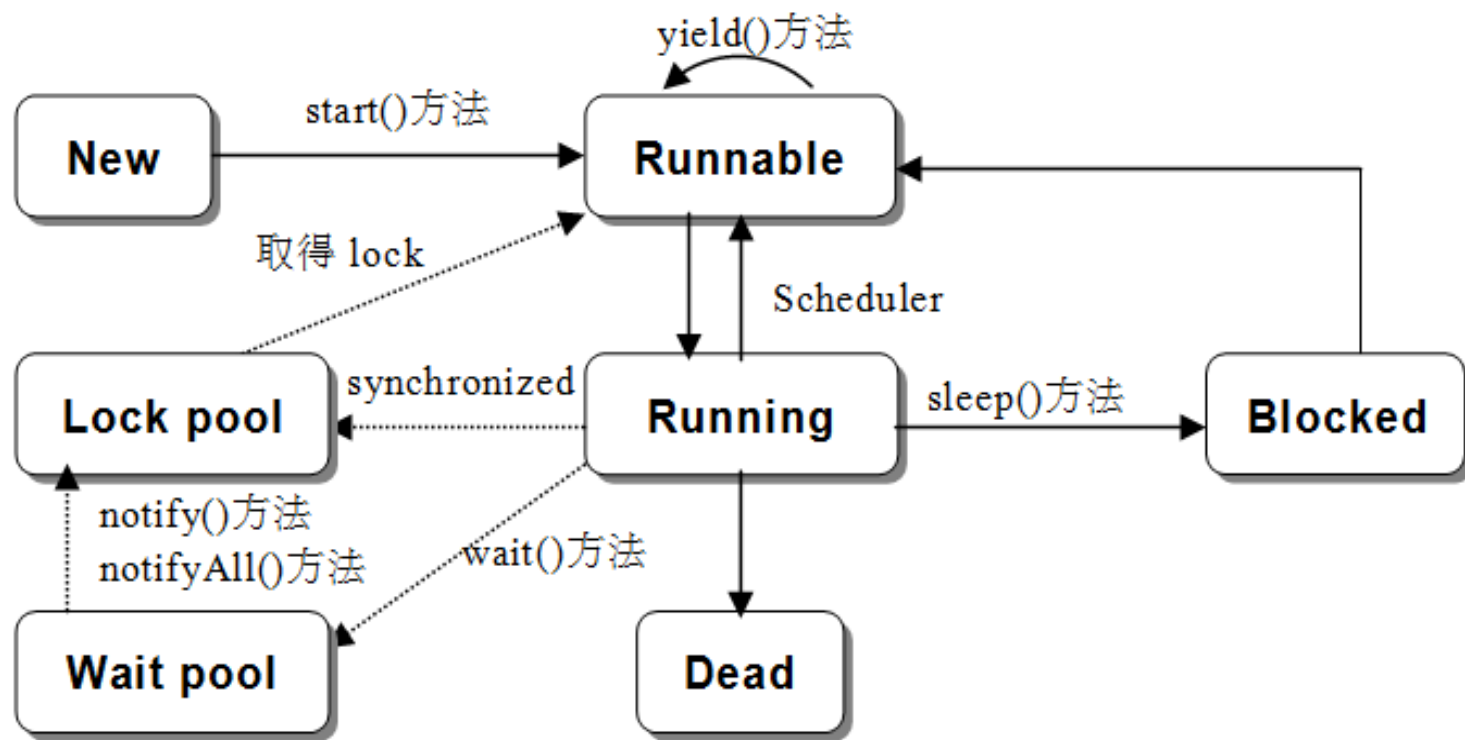
◆ Lock pool（鎖定池）：

- 當執行緒進入同步（Synchronized）程式區塊時，會進入「鎖定池」競爭「物件鎖（Lock）」，只有取得「物件鎖」的執行緒，才會回到「可執行」狀態，等待排程器排入執行。

◆ Dead（執行完畢）：

- 一旦執行完Thread物件所定義的run()方法，執行緒便進入到「執行完畢」狀態。
- 進入此狀態後的執行緒，就無法再被啟動。
- 如果想要再度啟動，就必須再重新建立Thread物件才行。

執行緒的生命週期



執行緒的生命週期

建立執行緒

■ 建立執行緒的方式有兩種：

- ◆ 繼承自 Thread 類別。
- ◆ 實作 Runnable 介面。

■ 不管是用哪種方式，都必須實作 Thread 類別的物件以及使用 Thread 類別的方法。

■ Thread 類別的建構子：

- ◆ `public Thread()`
 - 建立一個 Thread 物件。
 - 啟動執行緒時，會執行 Thread 的 `run()` 方法。
- ◆ `public Thread(String name)`
 - 建立一個 Thread 物件。

建立執行緒

- 參數是用來設定執行緒物件的名稱。

例如：

```
Thread t1=new Thread("執行緒1");
```

◆ public Thread(Runnable target)

- 建立一個Thread物件。
- 參數是Runnable介面的物件（也就是實作Runnable介面的類別所產生的物件，放在Thread物件中）。
- 啟動執行緒時，會執行target物件的run()方法。

例如：

MyThread類別是實作Runnable介面的類別，obR是MyThread類別的物件，則

```
Thread t1=new Thread(obR1);
```

建立執行緒

◆ public Thread(Runnable target, String name)

- 建立一個Thread物件。
- 第一個參數是Runnable介面的物件；第二個參數是設定執行緒物件的名稱。
- 啟動執行緒時，會執行target物件的run()方法。

例如：

```
Thread t1=new Thread( obT1, "執行緒1");
```

■ Thread類別常用的方法：

◆ void run()：

- 執行新執行緒。
- 用來撰寫新建立執行緒的程式碼。
 - ◆ 其宣告變數、呼叫其它方法、使用其它類別、...等用法，與main()相同。

建立執行緒

- 但main()是主執行緒的流程入口，而本方法run()需要用start()方法來呼叫。

◆ void start()：

- 啟動新執行緒。
- 用來呼叫run()方法，即新執行緒的流程入口。

◆ Thread.sleep(long 毫秒)：

- 使執行緒進入延遲或暫停狀態。
- 延遲的時間以毫秒為單位計數（1000毫秒等於1秒）。
- 因使用Thread.sleep()方法會拋出一個InterruptedException例外來catch，所以Thread.sleep()方法必須寫在try區段內。

繼承Thread類別建立執行緒

■ 使用繼承Thread類別來建立執行緒的方式，要有四個步驟：

1. 建立繼承自Thread類別的新類別。
2. 將執行緒所要執行的程式碼，寫在run()方法裡面。

當run()方法執行完畢，該執行緒即執行完畢。

3. 在主類別中，建立繼承Thread類別的執行緒類別物件，該物件稱為「執行緒物件」。
4. 在主類別中，使用執行緒物件的start()方法，啟動執行緒並間接呼叫寫在執行緒類別的run()方法。

格式如下：

繼承Thread類別建立執行緒

```
class 執行緒類別名稱 extends Thread
{
    資料成員;
    方法成員;
    public void run()
    {
        執行緒的敘述區段;
    }
}
```

繼承Thread類別建立執行緒

```
class 主類別名稱
```

```
{
```

```
    public static void main(String arg[])
```

```
    {
```

```
        ⋮
```

```
        執行緒類別名稱 執行緒物件 = new 執行緒類別  
                                                別名稱();
```

```
        執行緒物件.start();
```

```
        ⋮
```

```
    }
```

```
}
```

繼承Thread類別建立執行緒

◆ 程式：

```
package CH12_01;
```

```
class CAddThread extends Thread  
{
```

```
    public void run()  
    {
```

```
        int sum1 = 0, sum2;
```

```
        for (int i = 1; i <= 5; i++)  
        {
```

```
            sum2 = sum1 + 2;
```

```
            System.out.println("<累加>執行緒：" + sum1 + " + 2 = " + sum2);
```

```
            System.out.println("=====");
```

```
            sum1 = sum2;
```

```
        try
```

繼承Thread類別建立執行緒

```
{  
    Thread.sleep((long) (1000 * Math.random()));  
}  
catch (InterruptedException e)  
{  
  
}  
}  
}
```

```
class CFactThread extends Thread  
{  
    public void run()  
    {  
        for (int i = 1; i <= 5; i++)  
        {  
            int fact = 1;
```


繼承Thread類別建立執行緒

```
for (int j = 1; j <= i; j++)  
    fact *= j;
```

```
System.out.println("[階乘]執行緒：" + i + "!=" + fact);
```

```
try  
{  
    Thread.sleep((long) (1000 * Math.random()));  
}  
catch (InterruptedException e)  
{  
  
}
```

```
}  
}
```

```
public class CH12_01  
{
```

繼承Thread類別建立執行緒

```
public static void main(String[] args)
{
    CAddThread add_thread = new CAddThread();
    CFactThread fact_thread = new CFactThread();

    add_thread.start();
    fact_thread.start();
}
}
```

繼承Thread類別建立執行緒

```
1 package CH12_01;
2
3 class CAddThread extends Thread
4 {
5     public void run()
6     {
7         int sum1 = 0, sum2;
8
9         for (int i = 1; i <= 5; i++)
10        {
11            sum2 = sum1 + 2;
12
13            System.out.println("<累加>執行緒：" + sum1 + " + 2 = " + sum2);
14            System.out.println("=====");
15
16            sum1 = sum2;
17
18            try
19            {
20                Thread.sleep((long) (1000 * Math.random()));
21            }
22            catch (InterruptedException e)
23            {
24            }
25        }
26    }
```

繼承Thread類別建立執行緒

```
27     }  
28 }  
29  
30 class CFactThread extends Thread  
31 {  
32     public void run()  
33     {  
34         for (int i = 1; i <= 5; i++)  
35         {  
36             int fact = 1;  
37  
38             for (int j = 1; j <= i; j++)  
39                 fact *= j;  
40  
41             System.out.println("[階乘]執行緒：" + i + "!= " + fact);  
42  
43             try  
44             {  
45                 Thread.sleep((long) (1000 * Math.random()));  
46             }  
47             catch (InterruptedException e)  
48             {  
49             }  
50         }  
51     }  
52 }
```

繼承Thread類別建立執行緒

```
53 }  
54  
55 public class CH12_01  
56 {  
57     public static void main(String[] args)  
58     {  
59         CAddThread add_thread = new CAddThread();  
60         CFactThread fact_thread = new CFactThread();  
61  
62         add_thread.start();  
63         fact_thread.start();  
64     }  
65 }
```

繼承Thread類別建立執行緒

◆ 執行結果：

```
[階乘]執行緒：1! = 1  
<累加>執行緒：0 + 2 = 2  
=====
```

```
[階乘]執行緒：2! = 2  
<累加>執行緒：2 + 2 = 4  
=====
```

```
[階乘]執行緒：3! = 6  
<累加>執行緒：4 + 2 = 6  
=====
```

```
<累加>執行緒：6 + 2 = 8  
=====
```

```
[階乘]執行緒：4! = 24  
[階乘]執行緒：5! = 120  
<累加>執行緒：8 + 2 = 10  
=====
```

◆ 說明：

- 行01：

- ◆ 定義「套件（package）」。

繼承Thread類別建立執行緒

- 行03 ~ 行28：
 - ◆ 建立繼承自「Thread」父類別的「CAddThread」子類別。
 - ◆ 行05 ~ 行27：
 - ◆ 覆寫「run()」方法（此例為執行累加2，共5次）。
 - ◆ 行18 ~ 行25：
使執行緒延遲0~1秒內，不固定的時間。
- 行30 ~ 行53：
 - ◆ 建立繼承自「Thread」父類別的「CFactThread」子類別。
 - ◆ 行32 ~ 行52：
 - ◆ 覆寫「run()」方法（此例為執行計算5!）。
 - ◆ 行43 ~ 行50：
使執行緒延遲0~1秒內，不固定的時間。
- 行59 ~ 行60：
 - ◆ 利用類別的「建構子」，建立add_thread及fact_thread物件。

繼承Thread類別建立執行緒

- 行62 ~ 行63：
 - ◆ 啟動add_thread及fact_thread物件的執行緒。

實作Runnable介面建立執行緒

- Java的一個類別（class），只能繼承一個父類別（即單一繼承）。
- 萬一要建立執行緒的類別，必須繼承其它父類別而無法繼承Thread類別時，可用實作Runnable介面來建立執行緒。
- 使用實作Runnable介面來建立執行緒的方式，有五個步驟：
 1. 宣告一個執行緒類別來實作Runnable介面。
 2. 將執行緒所要執行的程式碼，寫在run()方法裡面。
當run()方法執行完畢，該執行緒即執行完畢。
 3. 在主類別中，建立一個類別物件。

實作Runnable介面建立執行緒

4. 在主類別中，再建立一個Thread物件，該物件稱為「執行緒物件」。
5. 在主類別中，使用執行緒物件的start()方法，啟動執行緒並間接呼叫寫在執行緒類別的run()方法。

格式如下：

```
class 執行緒類別名稱 implements Runnable  
{  
    資料成員;  
    方法成員;  
    public void run()  
    {  
        執行緒的敘述區段;  
    }
```

實作Runnable介面建立執行緒

```
}  
}
```

class 主類別名稱

```
{
```

```
    public static void main(String arg[])
```

```
{
```

```
    ⋮
```

```
        執行緒類別名稱 物件 = new 執行緒類別  
                                                別名稱();
```

```
        Thread 執行緒物件 = new Thread(物件);
```

實作Runnable介面建立執行緒

```
    執行緒物件.start();  
    ⋮  
}  
}
```

實作Runnable介面建立執行緒

◆ 程式：

```
package CH12_02;

class CAddThread implements Runnable
{
    private String thread_name;
    private int num;

    CAddThread(String name, int n)
    {
        thread_name = name;
        num = n;
    }

    public void run()
    {
        int sum1 = 0, sum2;

        for (int i = 1; i <= 5; i++)
```

實作Runnable介面建立執行緒

```
{
    sum2 = sum1 + num;

    System.out.println(thread_name + sum1 + " + " + num + " = " + sum2);

    if (num == 2)
        System.out.println("=====");

    sum1 = sum2;

    try
    {
        Thread.sleep((long) (1000 * Math.random()));
    }
    catch (InterruptedException e)
    {
    }

}
```

實作Runnable介面建立執行緒

```
    }  
}  
  
public class CH12_02  
{  
    public static void main(String[] args)  
    {  
        CAddThread thread1 = new CAddThread("累加 2 執行緒 :", 2);  
        CAddThread thread2 = new CAddThread("累加 5 執行緒 :", 5);  
  
        Thread add2_thread = new Thread(thread1);  
        Thread add5_thread = new Thread(thread2);  
  
        add2_thread.start();  
        add5_thread.start();  
    }  
}
```

實作Runnable介面建立執行緒

```
1 package CH12_02;
2
3 class CAddThread implements Runnable
4 {
5     private String thread_name;
6     private int num;
7
8     CAddThread(String name, int n)
9     {
10         thread_name = name;
11         num = n;
12     }
13
14     public void run()
15     {
16         int sum1 = 0, sum2;
17
18         for (int i = 1; i <= 5; i++)
19         {
20             sum2 = sum1 + num;
21
22             System.out.println(thread_name + sum1 + " + " + num + " = " + sum2);
23
24             if (num == 2)
25                 System.out.println("=====");
26         }
```


實作Runnable介面建立執行緒

```
27         sum1 = sum2;
28
29         try
30         {
31             Thread.sleep((long) (1000 * Math.random()));
32         }
33         catch (InterruptedException e)
34         {
35
36         }
37     }
38 }
39
40
41 public class CH12_02
42 {
43     public static void main(String[] args)
44     {
45         CAddThread thread1 = new CAddThread("累加 2 執行緒:", 2);
46         CAddThread thread2 = new CAddThread("累加 5 執行緒:", 5);
47
48         Thread add2_thread = new Thread(thread1);
49         Thread add5_thread = new Thread(thread2);
50
51         add2_thread.start();
52         add5_thread.start();
```

實作Runnable介面建立執行緒

```
53 }  
54 }
```

◆ 執行結果：

累加2 執行緒： $0 + 2 = 2$

累加5 執行緒： $0 + 5 = 5$

=====

累加5 執行緒： $5 + 5 = 10$

累加2 執行緒： $2 + 2 = 4$

=====

累加2 執行緒： $4 + 2 = 6$

=====

累加5 執行緒： $10 + 5 = 15$

累加5 執行緒： $15 + 5 = 20$

累加2 執行緒： $6 + 2 = 8$

=====

累加2 執行緒： $8 + 2 = 10$

=====

累加5 執行緒： $20 + 5 = 25$

實作Runnable介面建立執行緒

◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行03 ~ 行39：
 - ◆ 建立實作自「Runnable」介面的「CAddThread」子類別。
 - ◆ 行08 ~ 行12
 - ◆ 建立有兩個參數的類別建構子。
 - ◆ 行14 ~ 行38：
 - ◆ 實作「run()」方法（此例為執行累加 n，共5次）。
 - ◆ 行29 ~ 行36：
 - 使執行緒延遲0~1秒內，不固定的時間。
- 行45 ~ 行46：
 - ◆ 利用傳入兩個參數的類別「建構子」，建立thread1及thread2物件。

實作Runnable介面建立執行緒

- 行48 ~ 行49：
 - ◆ 有兩個執行緒物件add2_thread累加2與add5_thread累加5。
 - ◆ 而這兩個執行緒物件會各自執行自己的執行緒內容，彼此互不干擾。
- 行51 ~ 行52：
 - ◆ 啟動add2_thread及add5_thread物件的執行緒。

管理執行緒的方法

■ Thread類別定義了一些方法，用來幫助Java管理執行緒。

■ Thread類別在管理執行緒時，常用的方法：

◆ `public final void setName()`

- 設定執行緒的名稱。
- 若沒有命名，系統預設為Thread-0、Thread-1、...

◆ `public final String getName()`

- 取得執行緒名稱。

◆ `public static Thread currentThread()`

- 取得目前執行緒的參考值。

管理執行緒的方法

◆ `public static final boolean isAlive()`

- 判斷正在執行的執行緒是否存在。
- 存在回傳true，不存在回傳false。

◆ `public final void setPriority()`

- 設定執行緒的優先權。
- 設定值為1~10，數值越大優先權越高。
- Thread類別定義了幾個優先權常數可供使用：
 - ◆ `Thread.MIN_PRIORITY`
 - ◆ 數值1，最低優先權。
 - ◆ `Thread.NORM_PRIORITY`
 - ◆ 數值5，預設優先權。
 - ◆ `Thread.MAX_PRIORITY`
 - ◆ 數值10，最高優先權。

管理執行緒的方法

- ◆ `public final int getPriority()`
 - 取得Thread物件的優先權。
- ◆ `public final void join() throws InterruptedException`
 - 讓呼叫join()方法的執行緒中止時，繼續執行接在join()方法後的程式敘述。
 - 使用join()時，會拋出InterruptedException的例外，故須搭配try...catch一起使用。

管理執行緒的方法

◆ 程式：

```
package CH12_03;
```

```
class MyThread extends Thread  
{  
    public MyThread()  
    {  
        start();  
    }  
}
```

```
public void run()  
{  
    try  
    {  
        for (int i = 1; i <= 5; i++)  
        {  
            System.out.println(getName() + "執行緒：" + "執行第" + i + "次");  
            sleep(1000);  
        }  
    }  
}
```


管理執行緒的方法

```
    }  
    catch (InterruptedException e)  
    {  
  
    }  
}  
}  
  
public class CH12_03  
{  
    public static void main(String[] args)  
    {  
        MyThread obT1 = new MyThread();  
        obT1.setName("T1");  
  
        System.out.println( "目前的執行緒為：" +  
                             Thread.currentThread().getName());  
        System.out.println("執行緒 T1 是否活著：" + obT1.isAlive());  
    }  
}
```

管理執行緒的方法

```
try
{
    obT1.join();
}
catch (InterruptedException e)
{
}

}
System.out.println("執行緒T1 是否活著：" + obT1.isAlive());
}
```

管理執行緒的方法

```
1 package CH12_03;
2
3 class MyThread extends Thread
4 {
5     public MyThread()
6     {
7         start();
8     }
9
10    public void run()
11    {
12        try
13        {
14            for (int i = 1; i <= 5; i++)
15            {
16                System.out.println(getName() + "執行緒：" + "執行第" + i + "次");
17                sleep(1000);
18            }
19        }
20        catch (InterruptedException e)
21        {
22
23        }
24    }
25 }
26
```

管理執行緒的方法

```
27 public class CH12_03
28 {
29     public static void main(String[] args)
30     {
31         MyThread obT1 = new MyThread();
32         obT1.setName("T1");
33
34         System.out.println("目前的執行緒為：" + Thread.currentThread().getName());
35         System.out.println("執行緒 T1 是否活著：" + obT1.isAlive());
36
37         try
38         {
39             obT1.join();
40         }
41         catch (InterruptedException e)
42         {
43
44         }
45         System.out.println("執行緒T1 是否活著：" + obT1.isAlive());
46     }
47 }
```

管理執行緒的方法

◆ 執行結果：

```
目前的執行緒為：main  
T1執行緒：執行第1次  
執行緒 T1 是否活著：true  
T1執行緒：執行第2次  
T1執行緒：執行第3次  
T1執行緒：執行第4次  
T1執行緒：執行第5次  
執行緒T1 是否活著：false
```

◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行03~行25：
 - ◆ 建立繼承自「Thread」介面的「MyThread」子類別。
 - ◆ 行05~行08：
 - ◆ 建立類別的建構子。

管理執行緒的方法

- ◆ 行07：
啟動執行緒。
- ◆ 行10~行24：
 - ◆ 覆寫「run()」方法。
 - ◆ 因為有使用sleep()方法，故須搭配try...catch一起使用。
 - ◆ 行16：
取得執行緒物件的名稱。
 - ◆ 行17：
執行緒暫停1秒。
- 行31：
 - ◆ 利用MyThread類別，建立obT1物件。
- 行32：
 - ◆ 設定執行緒名稱。

管理執行緒的方法

- 行34：
 - ◆ 取得目前正在執行的執行緒的名稱。
- 行35
 - ◆ 判斷obT1執行緒是否存活。
- 行39：
 - ◆ 主執行緒會等到obT1執行緒執行完畢後，接著再執行。
 - ◆ 因為使用join()方法，故須搭配try...catch一起使用。
 - ◆ 若不使用join()方法，則主執行緒執行完後，才會執行obT1（T1）執行緒。

執行緒的同步

- 理論上來說，Thread物件可存取在Java程式中任一個物件。
- 為了避免在同時間點，有多個執行緒同時存取同一個物件，造成資料錯誤。
- JVM提供了一種機制，就是在發生多個執行緒同時存取且同時修改同個物件時，利用執行緒「同步（synchronized）」，來解決物件混亂情形。
- 使用關鍵字「synchronized」宣告後的程式區塊，一次只允許一個執行緒進入，故可以解決資料錯誤的情形。

執行緒的同步

- 當執行緒進入有「synchronized」的程式區塊時，會先檢查該區塊是否已經被「鎖定（lock）」。
 - ◆ 若沒有其它Thread物件佔住「鎖定」區塊，則目前Thread物件就可進入該程式區塊，同時將程式塊區「鎖定」以避免其它Thread物件進入。
 - ◆ 若程式區塊已被其它Thread物件「鎖定」，則Thread物件會進入「中斷（Interrupt）」的狀態，繼續等待被「鎖定」程式區塊被釋放。
 - ◆ 當Thread物件離開「同步」程式區塊時，代表著該程式區塊已經被釋放，JVM會從處於「中斷」狀態下的眾多Thread中，挑選一個Thread物件繼續執行。

執行緒的同步

◆ 程式（執行緒未同步）：

```
package CH12_04;
```

```
class GoldClass implements Runnable  
{
```

```
    int grabed;
```

```
    static int totalGold = 200000000;
```

```
    Thread t;
```

```
    public GoldClass(String name)
```

```
    {
```

```
        grabed = 0;
```

```
        t = new Thread(this, name);
```

```
        t.start();
```

```
    }
```

```
    public void run()
```

```
    {
```

```
        while (grabGold() == true)
```

執行緒的同步

```
grabbed++;
```

```
System.out.println(t.getName() + " 總共偷得 " + grabbed + " 個金塊.");  
}
```

```
private static boolean grabGold()
```

```
{  
    if (totalGold > 0)
```

```
    {  
        totalGold--;
```

```
        return true;
```

```
    }
```

```
    else
```

```
        return false;
```

```
    }
```

```
}
```

```
public class CH12_04
```

```
{
```

執行緒的同步

```
public static void main(String[] args)
{
    System.out.println("共有 " + GoldClass.totalGold + " 個金塊!");
    GoldClass tA = new GoldClass("張三");
    GoldClass tB = new GoldClass("李四");
    GoldClass tC = new GoldClass("王五");
}
```

執行緒的同步

```
1 package CH12_04;
2
3 class GoldClass implements Runnable
4 {
5     int grabed;
6     static int totalGold = 200000000;
7     Thread t;
8
9     public GoldClass(String name)
10    {
11        grabed = 0;
12        t = new Thread(this, name);
13        t.start();
14    }
15
16    public void run()
17    {
18        while (grabGold() == true)
19            grabed++;
20
21        System.out.println(t.getName() + " 總共偷得 " + grabed + " 個金塊.");
22    }
23
24    private static boolean grabGold()
25    {
26        if (totalGold > 0)
```

執行緒的同步

```
27     {  
28         totalGold--;  
29         return true;  
30     }  
31     else  
32         return false;  
33 }  
34 }  
35  
36 public class CH12_04  
37 {  
38     public static void main(String[] args)  
39     {  
40         System.out.println("共有 " + GoldClass.totalGold + " 個金塊!");  
41         GoldClass tA = new GoldClass("張三");  
42         GoldClass tB = new GoldClass("李四");  
43         GoldClass tC = new GoldClass("王五");  
44     }  
45 }
```

◆ 執行結果：

共有 20000000 個金塊!
張三 總共偷得 19989716 個金塊.
李四 總共偷得 19990281 個金塊.
王五 總共偷得 19994293 個金塊.

執行緒的同步

◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行03~行34：
 - ◆ 建立實作自「Runnable」介面的「GoldClass」子類別。
 - ◆ 行05：
 - ◆ 宣告整數變數，存放「已偷到的金塊數」。
 - ◆ 行06：
 - ◆ 宣告全域變數，存放「總金塊數」。
 - ◆ 行07：
 - ◆ 宣告執行緒物件。
 - ◆ 行09~行14：
 - ◆ 建立有一個引數的類別建構子。

執行緒的同步

- ◆ 行12：
建立執行緒物件。
- ◆ 行13：
啟動執行緒。
- ◆ 行16~行22：
 - ◆ 實作「run()」方法。
 - ◆ 行18：
判斷總金塊是否還有剩。
 - ◆ 行19：
某執行緒，偷一塊金塊。
 - ◆ 行21：
顯示某執行緒偷到的金塊數。
- ◆ 行24~行33：
 - ◆ 用靜態方法來判斷金塊餘額是否大於零。
 - ◆ 金塊餘額大於零，表示還可偷一塊金塊。

執行緒的同步

- ◆ 行26：
判斷金塊總數量是否大於零。
- ◆ 行28：
若金塊總數量大於零，則偷一塊金塊。
- ◆ 行29：
傳回true（回到run()方法中）。
- ◆ 行32：
若金塊總數量小於等於零，則傳回false（回到run()方法中）。

注意：

將行24更改為 `private synchronized static boolean grabGold()`，則可將執行緒同步（CH12_04A）。

- 行40：
 - ◆ 顯示金塊總數量。

執行緒的同步

- 行41~行43：
 - ◆ 利用類別的「建構子」，建立tA、tB、tC執行緒，並分別給執行緒名稱。

執行緒的等待和喚醒

■ Object 物件中提供：`wait()`、`notify()` 和 `notifyAll()` 方法，可讓執行緒間相互設定等待和喚醒。

◆ `wait()`：

- 讓指定執行緒進入Wait pool 成為等待狀態，每個物件都有自己專有Wait pool。

◆ `notify()`：

- 喚醒一個在Wait pool等待執行緒，哪個執行緒被喚醒由 JVM 決定。

◆ `notifyAll()`：

- 喚醒所有在 Wait pool等待執行緒，至於是哪個執行緒會先執行仍是由JVM決定。

執行緒的等待和喚醒

◆ 程式：

```
package CH12_05;

class Frisbee
{
    private boolean isThrow = false;

    public synchronized void throwF(int tNo)
    {
        while (isThrow)
        {
            try
            {
                wait();
            }
            catch (InterruptedException e)
            {
            }
        }
    }
}
```

執行緒的等待和喚醒

```
}  
System.out.println("丟出第 " + tNo + " 個飛盤");  
isThrow = true;  
notify();  
}
```

```
public synchronized void accessF(int aNo)  
{  
    while (!isThrow)  
    {  
        try  
        {  
            wait();  
        }  
        catch (InterruptedException e)  
        {  
        }  
    }  
}
```

執行緒的等待和喚醒

```
        System.out.println("接到第 " + aNo + " 個飛盤");  
        isThrow = false;  
        notify();  
    }  
}
```

```
class ThrowFrisbee implements Runnable  
{  
    Frisbee frisbee;  
  
    public ThrowFrisbee(Frisbee frisbee)  
    {  
        this.frisbee = frisbee;  
    }  
  
    public void run()  
    {  
        for (int i = 1; i <= 5; i++)  
            frisbee.throwF(i);  
    }  
}
```

執行緒的等待和喚醒

```
    }  
}  
  
class AccessFrisbee implements Runnable  
{  
    Frisbee frisbee;  
  
    public AccessFrisbee(Frisbee frisbee)  
    {  
        this.frisbee = frisbee;  
    }  
  
    public void run()  
    {  
        for (int i = 1; i <= 5; i++)  
            frisbee.accessF(i);  
    }  
}
```

執行緒的等待和喚醒

```
public class CH12_05
{
    public static void main(String[] args)
    {
        Frisbee frisbee = new Frisbee();
        Thread master = new Thread(new ThrowFrisbee(frisbee));
        Thread dog = new Thread(new AccessFrisbee(frisbee));
        master.start();
        dog.start();
    }
}
```


執行緒的等待和喚醒

```
1 package CH12_05;
2
3 class Frisbee
4 {
5     private boolean isThrow = false;
6
7     public synchronized void throwF(int tNo)
8     {
9         while (isThrow)
10         {
11             try
12             {
13                 wait();
14             }
15             catch (InterruptedException e)
16             {
17             }
18         }
19         System.out.println("丟出第 " + tNo + " 個飛盤");
20         isThrow = true;
21         notify();
22     }
23
24     public synchronized void accessF(int aNo)
25     {
26
```

執行緒的等待和喚醒

```
27     while (!isThrow)
28     {
29         try
30         {
31             wait();
32         }
33         catch (InterruptedException e)
34         {
35         }
36     }
37 }
38 System.out.println("接到第 " + aNo + " 個飛盤");
39 isThrow = false;
40 notify();
41 }
42 }
43
44 class ThrowFrisbee implements Runnable
45 {
46     Frisbee frisbee;
47
48     public ThrowFrisbee(Frisbee frisbee)
49     {
50         this.frisbee = frisbee;
51     }
52 }
```

執行緒的等待和喚醒

```
53     public void run()
54     {
55         for (int i = 1; i <= 5; i++)
56             frisbee.throwF(i);
57     }
58 }
59
60 class AccessFrisbee implements Runnable
61 {
62     Frisbee frisbee;
63
64     public AccessFrisbee(Frisbee frisbee)
65     {
66         this.frisbee = frisbee;
67     }
68
69     public void run()
70     {
71         for (int i = 1; i <= 5; i++)
72             frisbee.accessF(i);
73     }
74 }
75
76 public class CH12_05
77 {
78     public static void main(String[] args)
```

執行緒的等待和喚醒

```
79 {  
80     Frisbee frisbee = new Frisbee();  
81     Thread master = new Thread(new ThrowFrisbee(frisbee));  
82     Thread dog = new Thread(new AccessFrisbee(frisbee));  
83     master.start();  
84     dog.start();  
85 }  
86 }  
87
```

◆ 執行結果：

丟出第1個飛盤
接到第1個飛盤
丟出第2個飛盤
接到第2個飛盤
丟出第3個飛盤
接到第3個飛盤
丟出第4個飛盤
接到第4個飛盤
丟出第5個飛盤
接到第5個飛盤

執行緒的等待和喚醒

◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行03~行42：
 - ◆ 建立「Frisbee」類別。
 - ◆ 定義飛盤丟、接的方法。
 - ◆ 行05：
 - ◆ 宣告布林變數。
 - ◆ 紀錄飛盤的狀態（丟、接）。
 - ◆ 行07~行23：
 - ◆ 「丟」飛盤的方法。
 - ◆ 行09~行19：
 - ◆ 進入等待狀態。

執行緒的等待和喚醒

- ◆ 行20：
 - ◆ 顯示訊息。
- ◆ 行21：
 - ◆ 設定飛盤為「丟」狀態。
- ◆ 行22：
 - ◆ 呼叫「接」飛盤方法。
- ◆ 行25~行41：
 - ◆ 「接」飛盤的方法。
- ◆ 行27~行37：
 - ◆ 進入等待狀態。
- ◆ 行38：
 - ◆ 顯示訊息。
- ◆ 行39：
 - ◆ 設定飛盤為「丟」狀態。

執行緒的等待和喚醒

- ◆ 行40：
 - ◆ 呼叫「丟」飛盤方法。
- 行44~行58：
 - ◆ 建立實作自「Runnable」介面的「ThrowFrisbee」子類別。
 - ◆ 為「丟」飛盤的執行緒。
 - ◆ 行46：
 - ◆ 建立frisbee物件。
 - ◆ 行48~行51：
 - ◆ 建立有一個引數的類別建構子。
 - ◆ 行53~行57：
 - ◆ 實作「run()」方法。
 - ◆ 行55~行57：
 - 執行「丟」飛盤5次。

執行緒的等待和喚醒

- 行60~行74：
 - ◆ 建立實作自「Runnable」介面的「ThrowFrisbee」子類別。
 - ◆ 為「接」飛盤的執行緒。
 - ◆ 行62：
 - ◆ 建立frisbee物件。
 - ◆ 行64~行67：
 - ◆ 建立有一個引數的類別建構子。
 - ◆ 行69~行74：
 - ◆ 實作「run()」方法。
 - ◆ 行71~行72：
 - 執行「接」飛盤5次。
- 行80：
 - ◆ 利用「Frisbee」的建構子，建立frisbee物件。


執行緒的等待和喚醒

- 行81：
 - ◆ 利用「Thread」的建構子，建立master物件。
- 行82：
 - ◆ 利用「Thread」的建構子，建立dog物件。
- 行83：
 - ◆ 執行master執行緒。
- 行84：
 - ◆ 執行dog執行緒。

Timer類別

- 如果需要每隔一段時間就觸發一個或多個事件（ActionEvent）以改變內容或執行特定程式，可以使用Timer類別來製作計時器。
- Timer類別的建構子：
 - ◆ `Timer(int delay, ActionListener listener)`
 - 建立一個計時器物件。
 - 以delay引數值（單位：毫秒）為間隔的時間，定時觸動計時器一次。
 - 而listener為計時器物件的事件傾聽者。

Timer類別

 Timer類別常用的方法如下：

◆ void start()

- 啟動計時器物件，使它開始向其傾聽者發送動作事件。

◆ void stop()

- 停止計時器物件，使它停止向其傾聽者發送動作事件。

◆ void restart()

- 重新啟動計時器物件。

Timer類別

◆ 程式：

```
package CH12_06;

import javax.swing.*;
import java.awt.event.*;

class CTimerF extends JFrame implements ActionListener
{
    private int hour, minute, second;
    private long conti_time, tot_time = 0;
    private boolean is_start = true, is_pause, is_showtime;
    private Timer timer = new Timer(1000, this);
    private JLabel lblTimer = new JLabel("0 時 : 0 分 : 0 秒");
    private JButton btnStart = new JButton("開始");
    private JButton btnReset = new JButton("歸零");

    public CTimerF()
    {
        lblTimer.setBounds(50, 10, 150, 20);
    }
}
```

Timer類別

```
add(lblTimer);

btnStart.setBounds(20, 40, 60, 20);
add(btnStart);
btnStart.addActionListener(this);

btnReset.setBounds(100, 40, 60, 20);
add(btnReset);
btnReset.addActionListener(this);

setTitle("計時器");
setLayout(null);
setBounds(100, 100, 200, 110);
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);
}

public void actionPerformed(ActionEvent e)
{
```

Timer類別

```
if (e.getSource() == btnReset)
{
    timer.stop();
    is_showtime = false;
    lblTimer.setText("0 時 : 0 分 : 0 秒");
    tot_time = 0;
    btnStart.setText("開始");
    is_start = true;
}

if (e.getSource() == btnStart)
{
    if (is_start)
    {
        timer.start();
        conti_time = System.currentTimeMillis();
        is_showtime = true;
        btnStart.setText("暫停");
        is_start = false;
    }
}
```

Timer類別

```
        is_pause = true;
    }
    else if (is_pause)
    {
        timer.stop();
        is_showtime = false;
        btnStart.setText("繼續");
        is_pause = false;
    }
    else
    {
        timer.restart();
        conti_time = System.currentTimeMillis();
        is_showtime = true;
        btnStart.setText("暫停");
        is_pause = true;
    }
}
```

Timer類別

```
    if (is_showtime)
    {
        tot_time += System.currentTimeMillis() - conti_time;
        hour = (int) (tot_time / 1000) / (60 * 60);
        minute = (int) ((tot_time / 1000) / 60) % 60;
        second = (int) (tot_time / 1000) % 60;

        lblTimer.setText(String.valueOf(hour) + "時："
            + String.valueOf(minute) + "分："
            + String.valueOf(second) + "秒");
        conti_time = System.currentTimeMillis();
    }
}
```

```
public class CH12_06
{
    public static void main(String[] args)
    {
```


Timer類別

```
        new CTimerF();  
    }  
}
```

Timer類別

```
1 package CH12_06;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5
6 class CTimerF extends JFrame implements ActionListener
7 {
8     private int hour, minute, second;
9     private long conti_time, tot_time = 0;
10    private boolean is_start = true, is_pause, is_showtime;
11    private Timer timer = new Timer(1000, this);
12    private JLabel lblTimer = new JLabel("0 時:0 分:0 秒");
13    private JButton btnStart = new JButton("開始");
14    private JButton btnReset = new JButton("歸零");
15
16    public CTimerF()
17    {
18        lblTimer.setBounds(50, 10, 150, 20);
19        add(lblTimer);
20
21        btnStart.setBounds(20, 40, 60, 20);
22        add(btnStart);
23        btnStart.addActionListener(this);
24
25        btnReset.setBounds(100, 40, 60, 20);
26        add(btnReset);
```

Timer類別

```
27     btnReset.addActionListener(this);
28
29     setTitle("計時器");
30     setLayout(null);
31     setBounds(100, 100, 200, 110);
32     setVisible(true);
33     setDefaultCloseOperation(EXIT_ON_CLOSE);
34 }
35
36 public void actionPerformed(ActionEvent e)
37 {
38     if (e.getSource() == btnReset)
39     {
40         timer.stop();
41         is_showtime = false;
42         lblTimer.setText("0 時:0 分:0 秒");
43         tot_time = 0;
44         btnStart.setText("開始");
45         is_start = true;
46     }
47
48     if (e.getSource() == btnStart)
49     {
50         if (is_start)
51         {
52             timer.start();
```

Timer類別

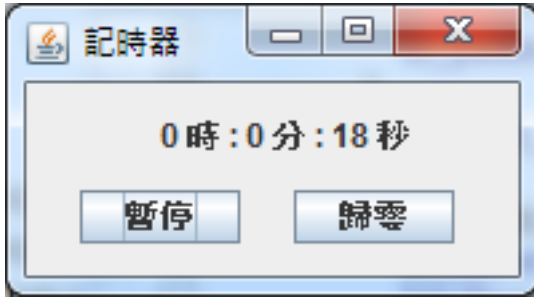
```
53         conti_time = System.currentTimeMillis();
54         is_showtime = true;
55         btnStart.setText("暫停");
56         is_start = false;
57         is_pause = true;
58     }
59     else if (is_pause)
60     {
61         timer.stop();
62         is_showtime = false;
63         btnStart.setText("繼續");
64         is_pause = false;
65     }
66     else
67     {
68         timer.restart();
69         conti_time = System.currentTimeMillis();
70         is_showtime = true;
71         btnStart.setText("暫停");
72         is_pause = true;
73     }
74 }
75
76 if (is_showtime)
77 {
78     tot_time += System.currentTimeMillis() - conti_time;
```

Timer類別

```
79         hour = (int) (tot_time / 1000) / (60 * 60);
80         minute = (int) ((tot_time / 1000) / 60) % 60;
81         second = (int) (tot_time / 1000) % 60;
82
83         lblTimer.setText(String.valueOf(hour) + " 時:"
84                         + String.valueOf(minute) + " 分:"
85                         + String.valueOf(second) + " 秒");
86         conti_time = System.currentTimeMillis();
87     }
88 }
89 }
90
91 public class CH12_06
92 {
93     public static void main(String[] args)
94     {
95         new CTimerF0();
96     }
97 }
```

Timer類別

◆ 執行結果：



◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行03~ 行04：
 - ◆ 載入相關的套件。
- 行06 ~ 行89：
 - ◆ 建立繼承自「JFrame」，並實作自「ActionListener」介面的「CTimerF」子類別。

Timer類別

- ◆ 行08 ~ 行10：
 - ◆ 宣告「私有」的類別「資料成員」，並指定初值。
- ◆ 行11：
 - ◆ 利用Timer類別的建構子，建立一個「私有」的計時器物件timer。
 - ◆ 設定每隔1000毫秒（1秒），觸動timer計時器一次，而以本視窗為計時器事件的傾聽者。
- ◆ 行12 ~ 行14：
 - ◆ 利用「視窗元件」的建構子，建立「私有」的視窗物件。
- ◆ 行16 ~ 行34：
 - ◆ 宣告類別的「建構子」。
 - ◆ 建立視窗。
- ◆ 行36 ~ 行74：
 - ◆ 實作「ActionListener」介面的「actionPerformed()」方法。

Timer類別

- ◆ 行38 ~ 行46：

 - 當「歸零」鈕被點按後要執行之程式敘述。

- ◆ 行48 ~ 行74：

 - 當「開始」鈕被點按後要執行之程式敘述。

 - 行50 ~ 行58：

 - 如果按下「開始」鈕，即啟動Timer計時器
(timer.start()) 。

 - 啟動後，每隔1秒便會去執行行76 ~ 行87的計算及
顯示時間。

 - 行59 ~ 行65：

 - 如果按下「暫停」鈕，即停止Timer計時器
(timer.stop()) 。

 - 行66 ~ 行73：

 - 如果按下「繼續」鈕，即重新啟動Timer計時器
(timer.restart()) 。

Timer類別

- ◆ 行76 ~ 行87：
計算及顯示時間。
- 行95：
 - ◆ 利用「CTimerF()」建構子，建立匿名物件。

綜合練習（1）

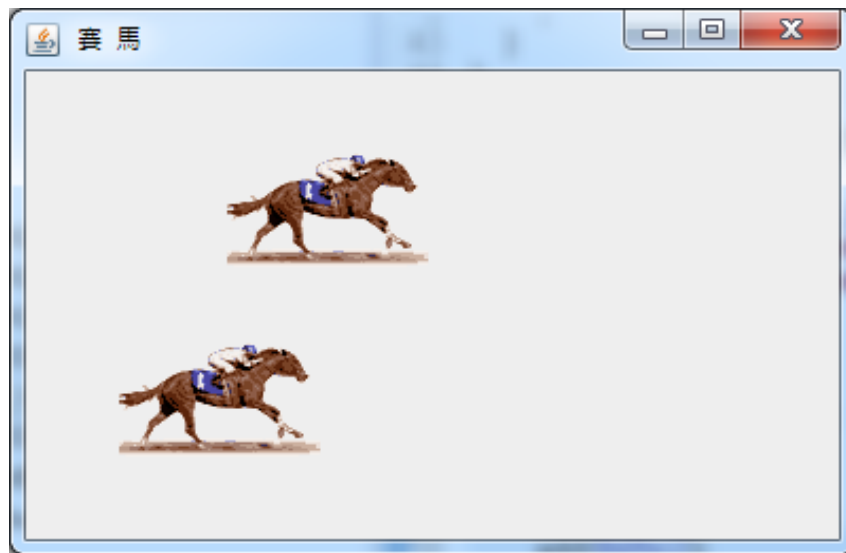
題目：『賽馬』

要求：

設計多工視窗程式，內容為賽馬。

說明：

視窗內有兩匹馬，以不相互干擾的方式，向前移動（即賽馬）。



綜合練習（1）（續）

分析：

- 1、視窗中放入兩張圖片的標籤物件lblPic1、lblPic2。
- 2、兩標籤圖片從視窗左側出發，當標籤圖片移到視窗的右側邊框時，會再從視窗左側出發。
- 3、兩匹馬移動時機，由電腦亂數決定，但一匹馬的移動的速度約為另一匹馬的兩倍。

綜合練習（2）

題目：『猜數字』

要求：

設計多工視窗程式，內容為猜數字，過程中視窗下方要有一匹馬，由左而右重複移動。



綜合練習（2）（續）

分析：

- 1、猜數的操作：以亂數產生0~99數。
- 2、所猜數太大或太小都會有提示，並且告知目前已猜幾次。
- 3、視窗下方有一匹馬，由左至右不斷重複移動。

綜合練習 (3)

題目：『銀行交易』

要求：

設計多工視窗程式，內容建立一個銀行帳戶，可以允許多人同時存取，且能確保交易後所得到的帳戶餘額是正確的。

綜合練習（4）

題目：『銀行交易』

要求：

設計建立一個銀行帳戶，先有匯款進入帳戶，才能扣款。

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。