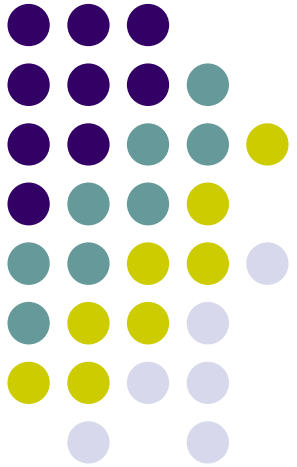
第十五章
多執行緒

認識執行緒

學習如何建立執行緒

學習如何管理執行緒

認識執行緒的同步處理



認識執行緒

執行緒（thread）是指程式的執行流程
「多執行緒」則可同時執行多個程式區塊

```
01  // app15_1, 單一執行緒的範例
02  class CTest
03  {
04     private String id;
05     public CTest(String str)          // 建構元，設定資料成員 id
06     {
07        id=str;
08     }
09     public void run()                 // run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           for(int j=0;j<100000000;j++);   // 空迴圈，用來拖慢 14 行執行的速度
14           System.out.println(id+" is running..");
15        }
16     }
17  }
18
19  public class app15_1
20  {
21     public static void main(String args[])
22     {
23        CTest dog=new CTest("doggy");
24        CTest cat=new CTest("kitty");
25        dog.run();
26        cat.run();
27     }
28  }
```

app15_1為單一執行緒的範例

```
/* app15_1 OUTPUT-------
doggy is running..
doggy is running..
doggy is running..
doggy is running..
kitty is running..
kitty is running..
kitty is running..
kitty is running..
----------------------*/
```

第25行用dog物件呼叫run() method的執行結果

第26行用cat物件呼叫run() method的執行結果

啟動執行緒

- 啟動執行緒前要先準備下列兩件事情：

  (1) 此類別必須延伸自Thread類別

  (2) 執行緒的處理必須撰寫在run() method內

- 啟動執行緒的語法：

執行緒之定義語法

```
class 類別名稱 extends Thread  // 從Thread類別延伸出子類別
{
    類別裡的資料成員;
    類別裡的method;
    修飾子 run()    // 改寫Thread類別裡的run() method
    {
        以執行緒處理的程序;
    }
}
```

啟動執行緒的範例

```
01  // app15_2, 啟動執行緒的範例
02  class CTest extends Thread     // 從 Thread 類別延伸出子類別 CTest
03  {
04     private String id;
05     public CTest(String str)    // 建構元，設定成員 id
06     {
07        id=str;
08     }
09     public void run()           // 改寫 Thread 類別裡的 run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           for(int j=0;j<100000000;j++); // 空迴圈，用來拖慢 14 行執行的速度
14           System.out.println(id+" is running..");
15        }
16     }
17  }
18
19  public class app15_2
20  {
21     public static void main(String args[])
22     {
23        CTest dog=new CTest("doggy");
24        CTest cat=new CTest("kitty");
25        dog.start();              // 注意是呼叫 start(),而不是 run()
26        cat.start();              // 注意是呼叫 start(),而不是 run()
27     }
28  }
```

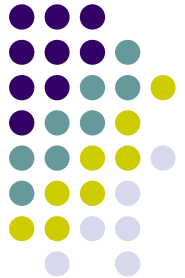
app15_2可同時
啟動多個執行緒

呼叫start() method時，會在排程器中登錄該執行緒，
當它開始執行時，run() method自然會被呼叫

第26行用cat物件呼叫
run() method的執行結果

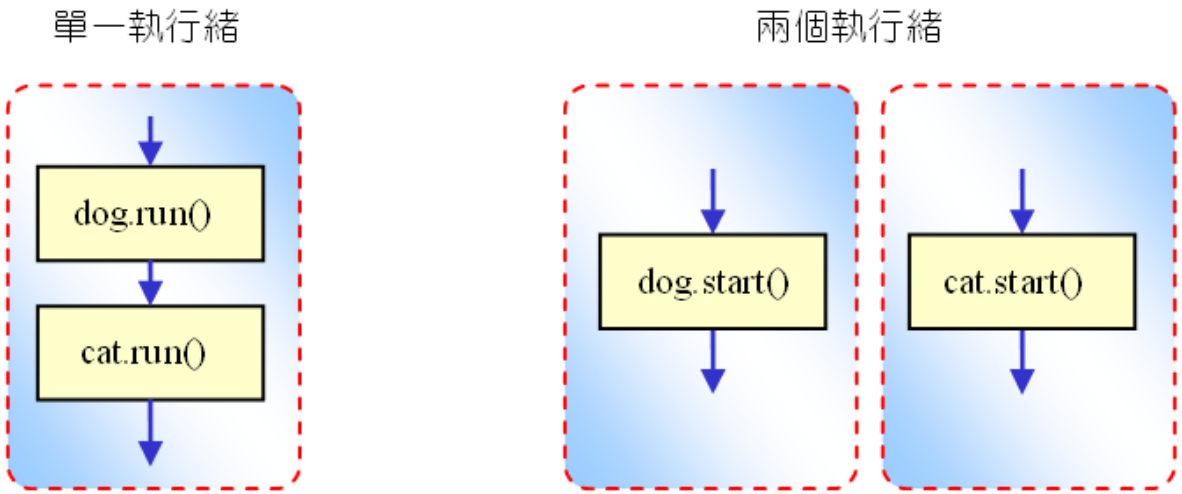
```
/* app15_2 OUTPUT-------
kitty is running..
doggy is running..
kitty is running..
kitty is running..
doggy is running..
kitty is running..
doggy is running..
doggy is running..
----------------------*/
```

第25行用dog物件呼叫
run() method的執行結果



執行緒的比較

- 下圖為單一執行緒與兩個執行緒的執行流程比較：



app15_1.java

執行完 dog.run() 之後，
再執行 cat.run()

app15_2.java

由 start() 啟動執行緒，因而 dog
和 cat 物件的 run() 會同時被執行

建立執行緒

- 如果類別本身已經繼承某個父類別，可以利用實作Runnable介面的方式建立執行緒
  - 介面是實現多重繼承的重要方式
  - 把處理執行緒的程式碼，放在實作Runnable介面的類別中的run() 就可以建立執行緒

執行緒的使用

15.2 實作Runnable介面來建立執行緒

```
01  // app15_3, 實作 Runnable 介面來建立執行緒
02  class CTest implements Runnable    // 由 CTest 類別實作 Runnable 介面
03  {
04     private String id;
05     public CTest(String str)         // 建構元，設定成員 id
06     {
07        id=str;
08     }
09     public void run()    // 詳細定義 runnable() 介面裡的 run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           for(int j=0;j<100000000;j++); // 空迴圈，用來拖慢14行執行的速度
14           System.out.println(id+" is running..");
15        }
16     }
17  }
18
19  public class app15_3
20  {
21     public static void main(String args[])
22     {
23        CTest dog=new CTest("doggy");
24        CTest cat=new CTest("kitty");
25        Thread t1=new Thread(dog);    // 產生 Thread 類別的物件 t1
26        Thread t2=new Thread(cat);    // 產生 Thread 類別的物件 t2
27        t1.start();                   // 用 t1 啟動執行緒
28        t2.start();                   // 用 t2 啟動執行緒
29     }
30  }
```

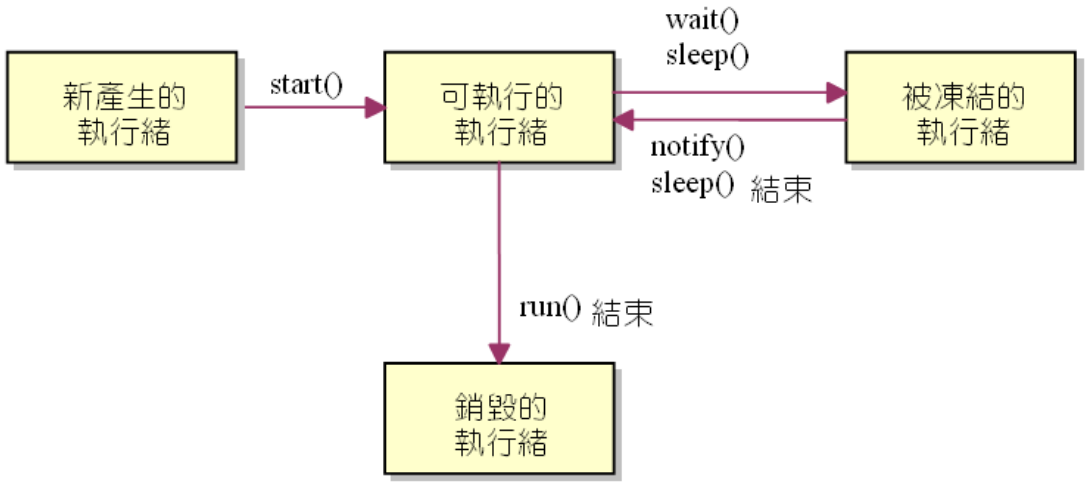
第28行用t2物件呼叫run() method

```
/* app15_3 OUTPUT----------
kitty is running..
doggy is running..
kitty is running..
kitty is running..
doggy is running..
kitty is running..
doggy is running..
doggy is running..
-------------------------*/
```

第27行用t1物件呼叫run() method

執行緒的生命週期 (1/3)

- 每一個執行緒，在其產生和銷毀之前，均會處於下列四種狀態之一：
  - 新產生的（newly created）
  - 可執行的（runable）
  - 被凍結的（blocked）
  - 銷毀的（dead）
- 執行緒狀態的轉移與method之間的關係 ：



執行緒的生命週期 (2/3)

- **新產生的執行緒**
  - 用new Thread() 建立物件時，執行緒便是這種狀態
  - 用start() method啟動執行緒時才會配置資源
- **可執行的狀態**
  - start() method啟動執行緒時，便進入可執行的狀態
  - 最先搶到CPU資源的執行緒先執行run() method，其餘的便在佇列（queue）中等待
- **銷毀的狀態**
  - run() method執行結束，或是由執行緒呼叫它的stop() method時，進入銷毀的狀態

執行緒的生命週期 (3/3)

- 被凍結的狀態
  - 發生下列的事件時，凍結狀態的執行緒便產生：

    (1) 該執行緒呼叫物件的wait() method

    (2) 該執行緒本身呼叫sleep() method

    (3) 該執行緒和另一個執行緒join() 在一起時

  - 被凍結因素消失的原因有下列幾點：

    (1) 如果執行緒是由呼叫物件的wait() method所凍結，則該物件的notify() method被呼叫時可解除凍結

    (2) 執行緒進入睡眠（sleep）狀態，但指定的睡眠時間已到

讓執行緒小睡片刻

sleep() method 的使用範例

```
01  // app15_4, sleep() method 的示範
02  class CTest extends Thread     // 從 Thread 類別延伸出子類別
03  {
04     private String id;
05     public CTest(String str)    // 建構元，設定成員 id
06     {
07        id=str;
08     }
09     public void run()           // 改寫 Thread 類別裡的 run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           try
14           {
15              sleep((int)(1000*Math.random()));
16           }
17           catch(InterruptedException e){}
18           System.out.println(id+" is running..");
19        }
20     }
21  }
22
23  public class app15_4
24  {
25     public static void main(String args[])
26     {
27        CTest dog=new CTest("doggy");
28        CTest cat=new CTest("kitty");
29        dog.start();
30        cat.start();
31     }
32  }
```

Math.random() 會產生0~1之間的亂數，乘上1000後變成0~1000之間的浮點數亂數，再強制轉換成整數，控制執行緒的小睡時間為0秒到1秒之間的亂數

sleep() method 必須寫在 try-catch 區塊裡

catch 接收的必須是 InterruptedException例外

第30行用cat物件呼叫start() method

第29行用dog物件呼叫start() method

```
/* app15_4 OUTPUT---------
kitty is running..
doggy is running..
kitty is running..
kitty is running..
doggy is running..
kitty is running..
doggy is running..
doggy is running..
-------------------------*/
```

等待執行緒 (1/2)

- app15_5是執行緒啟動後再加上字串的列印：

```
01  // app15_5, 執行緒排程的設計(一)
02  // 將 app15_4 的 CTest 類別置於此處
03  public class app15_5
04  {
05     public static void main(String args[])
06     {
07        CTest dog=new CTest("doggy");
08        CTest cat=new CTest("kitty");
09        dog.start();                    // 用 dog 物件來啟動執行緒
10        cat.start();                    // 用 cat 物件來啟動執行緒
11        System.out.println("main() method finished");
12     }
13  }
```

```
/* app15_5 OUTPUT---------
main() method finished
doggy is running..
kitty is running..
doggy is running..
doggy is running..
doggy is running..
kitty is running..
kitty is running..
kitty is running..
------------------------*/
```

main() method本身也是一個執行緒，因此main() 執行完第9、10行之後，會往下執行11行的敘述，通常是11行的敘述會先執行，因為它不用經過執行緒的啟動程序

等待執行緒 (2/2)

- 下面的範例將app15_5稍做修改：

```
01  // app15_6, 執行緒排程的設計(二)
02  // 將 app15_4 的 CTest 類別置於此處
03  public class app15_6
04  {
05     public static void main(String args[])
06     {
07        CTest dog=new CTest("doggy");
08        CTest cat=new CTest("kitty");
09
10        dog.start();     // 啟動 dog 執行緒
11        try
12        {
13           dog.join();   // 限制 dog 執行緒結束後才能往下執行
14           cat.start();  // 啟動 cat 執行緒
15           cat.join();   // 限制 cat 執行緒結束後才能往下執行
16        }
17        catch(InterruptedException e){}
18        System.out.println("main() method finished");
19     }
20  }
```

```
/* app15_6 OUTPUT----------
doggy is running..
doggy is running..
doggy is running..
doggy is running..
kitty is running..
kitty is running..
kitty is running..
kitty is running..
main() method finished
----------*/
```

先執行dog執行緒

再執行cat執行緒

會拋出InterruptedException例外

最後再執行第18行的敘述

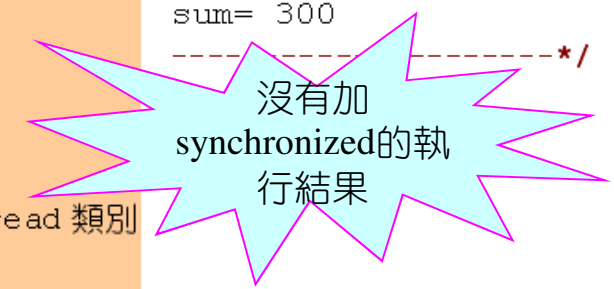
join() 必須寫在try-catch 區塊裡

錯誤的執行緒 (1/2)

- 下面的範例是沒有同步處理的執行緒：

```
01  // app15_7, 沒有同步處理的執行緒
02  class CBank
03  {
04     private static int sum=0;
05     public static void add(int n)
06     {
07        int tmp=sum;
08        tmp=tmp+n;                    // 累加匯款總額
09        try
10        {
11           Thread.sleep((int)(1000*Math.random()));   // 小睡 0~1 秒鐘
12        }
13        catch(InterruptedException e){}
14        sum=tmp;
15        System.out.println("sum= "+sum);
16     }
17  }
18  class CCustomer extends Thread // CCustomer 類別，繼承自 Thread 類別
19  {
20     public void run()             // run() method
21     {
22        for(int i=1;i<=3;i++)
23           CBank.add(100);         // 將 100 元分三次匯入
24     }
25  }
```

```
/* app15_7 OUTPUT----
sum= 100
sum= 100
sum= 200
sum= 300
sum= 200
sum= 300
----------------------*/
```

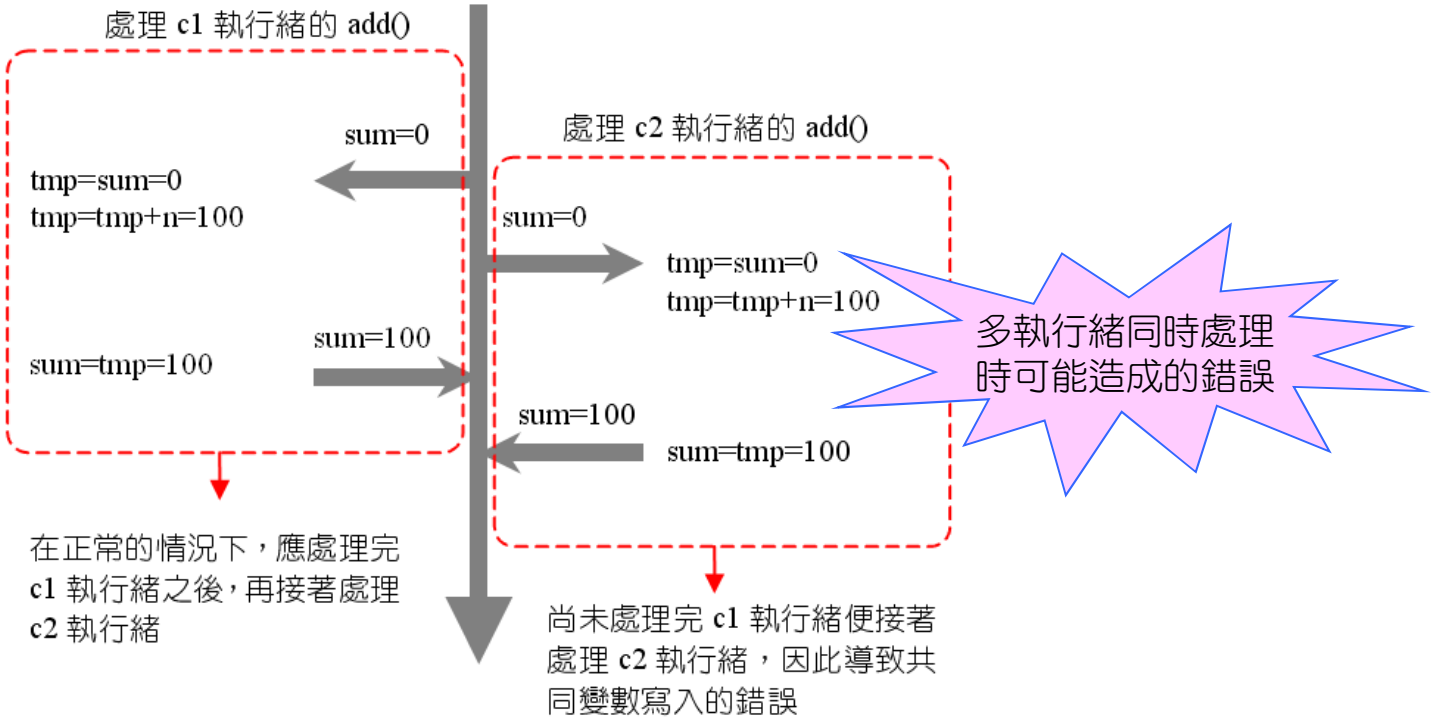


錯誤的執行緒 (2/2)


```
26  public class app15_7
27  {
28     public static void main(String args[])
29     {
30        CCustomer c1=new CCustomer();
31        CCustomer c2=new CCustomer();
32        c1.start();
33        c2.start();
34     }
35  }
```

```
/* app15_7 OUTPUT----
sum= 100
sum= 100
sum= 200
sum= 300
sum= 200
sum= 300
--------------------*/
```

沒有加synchronized的執行結果



修正錯誤

- 要更正錯誤，只在add() method之前加上synchronized關鍵字，如下面的語法：

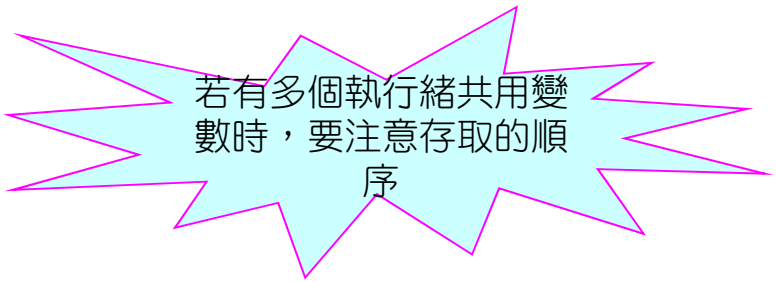
```
public synchronized static void add(int n)
{
    ... ...
}
```

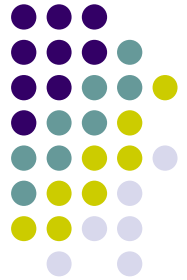
在 add() method 之前加上 synchronized 關鍵字

- synchronized本意是「同步」的意思，一次只允許一個執行緒進入run() method
- app15_7第5行的add() method前加上synchronized，執行結果如下：

```
/* app15_7 OUTPUT------(加上 synchronized 的執行結果)

sum= 100
sum= 200
sum= 300
sum= 400
sum= 500
sum= 600
----------------------*/
```





-The End-