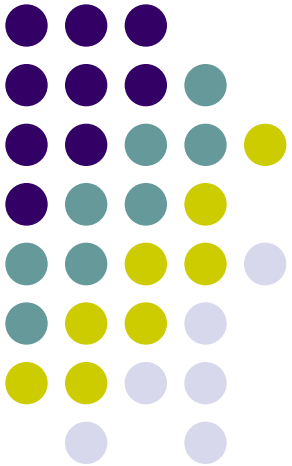
第十五章
多執行緒



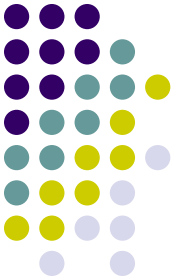
認識執行緒

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(1/2)



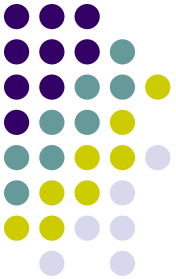
```
01  // Ch15_1, 單一執行緒的範例
02  class CTest{
03     private String id;
04     public CTest(String str){            // 建構子，設定資料成員 id
05        id=str;
06     }
07     public void run(){                   // run() 函數
08        for(int i=0;i<4;i++){
09           for(int j=0;j<100000000;j++);//空迴圈，用來拖慢 10 行執行的速度
10           System.out.println(id+" is running...");
11        }
12     }
13  }
14  public class Ch15_1{
15     public static void main(String[] args){
16        CTest dog=new CTest("doggy");
17        CTest cat=new CTest("kitty");
18        dog.run();
19        cat.run();
20     }
21  }
```

Ch15_1為單一執行緒的範例

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

接續下一頁

認識執行緒(2/2)

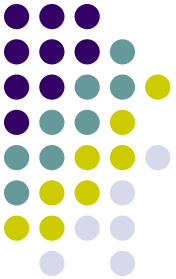


- 執行結果：

```
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 18 行用 dog 物件呼叫 run() 函數的執行結果

第 19 行用 cat 物件呼叫 run() 函數的執行結果



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() 內

- 定義執行緒的語法：

執行緒的定義語法

```
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;
    類別裡的函數;
    修飾子 run(){                 // 改寫 Thread 類別裡的 run() 函數
        以執行緒處理的程序;
    }
}
```

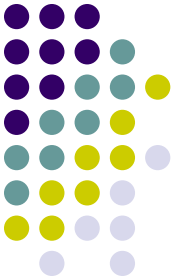
啟動執行緒的範例(1/2)

```
01  // Ch15_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     private String id;
04     public CTest(String str){     // 建構子，設定成員 id
05        id=str;
06     }
07     public void run(){            // 改寫 Thread 類別裡的 run() 函數
08        for(int i=0;i<4;i++){
09           for(int j=0;j<100000000;j++);//空迴圈，用來拖慢 10 行執行的速度
10           System.out.println(id+" is running...");
11        }
12     }
13  }
14  public class Ch15_2{
15     public static void main(String[] args){
16        CTest dog=new CTest("doggy");
17        CTest cat=new CTest("kitty");
18        dog.start();                  // 注意是呼叫 start(),而不是 run()
19        cat.start();                  // 注意是呼叫 start(),而不是 run()
20     }
21  }
```

Ch15_2可同時
啟動多個執行緒

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

接續下一頁



啟動執行緒的範例(2/2)

- 執行結果：

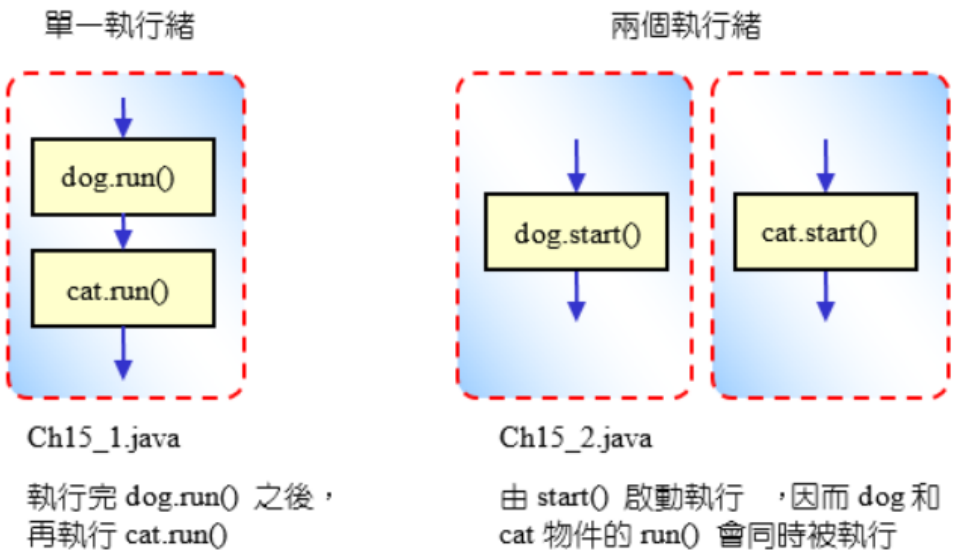
```
kitty is running...
doggy is running...
kitty is running...
kitty is running...
doggy is running...
kitty is running...
doggy is running...
doggy is running...
```

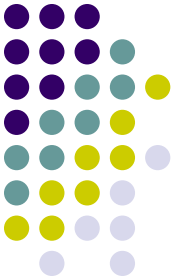
第 18 行用 cat 物件呼叫 start() 函數

第 19 行用 dog 物件呼叫 start() 函數

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：



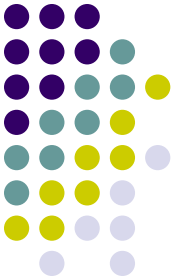


建立執行緒

- 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
  - 介面是解決類別不可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

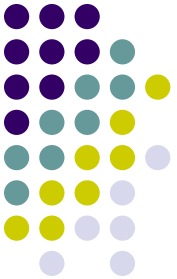
15.2 實作Runnable介面來建立執行緒



```
01  // Ch15_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     private String id;
04     public CTest(String str){          // 建構元，設定成員 id
05        id=str;
06     }
07     public void run(){     // 詳細定義 runnable() 介面裡的 run() 函數
08        for(int i=0;i<4;i++){
09           for(int j=0;j<100000000;j++); // 空迴圈，用來拖慢 10 行執行的速度
10           System.out.println(id+" is running...");
11        }
12     }
13  }
14
15  public class Ch15_3{
16     public static void main(String args[]){
17        CTest dog=new CTest("doggy");
18        CTest cat=new CTest("kitty");
19        Thread t1=new Thread(dog);      // 產生 Thread 類別的物件 t1
20        Thread t2=new Thread(cat);      // 產生 Thread 類別的物件 t2
21        t1.start();                  // 用 t1 啟動執行緒
22        t2.start();                  // 用 t2 啟動執行緒
23     }
24  }
```

執行結果：

```
kitty is running...      ——— 第 22 行用 t2 物件呼叫 run() 函數
doggy is running...      ——— 第 21 行用 t1 物件呼叫 run() 函數
kitty is running...
kitty is running...
doggy is running...
kitty is running...
doggy is running...
doggy is running...
```



使用Thread還是Runnable?

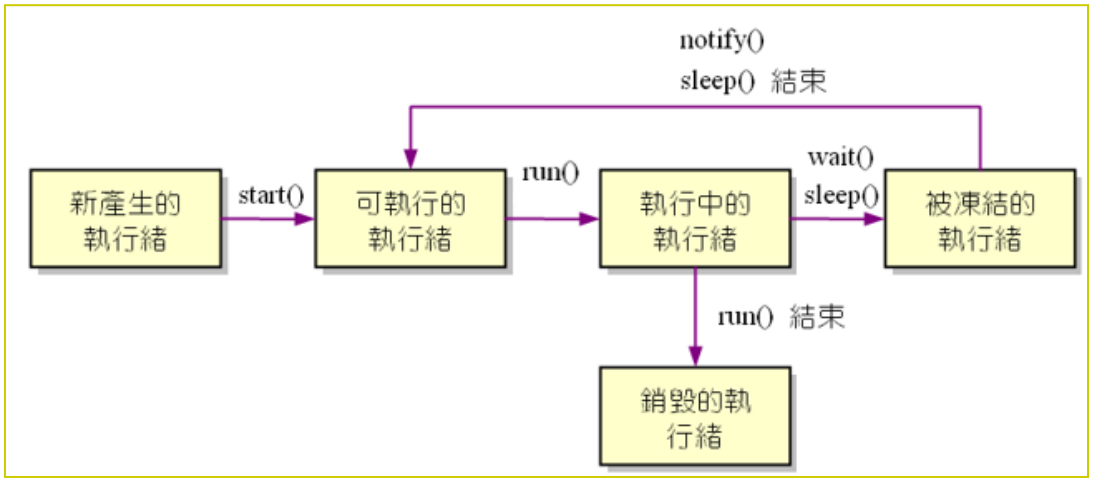
- 類別只能繼承一個類別，可以實作多個介面
- 如果要使用多執行緒的類別已經繼承其他類別，就必須實作（implement）Runnable介面

```
class CTest extends CCircle implements Runnable
{…}    // CTest 類別繼承 CCircle 類別，並實作 Runnable 介面
```

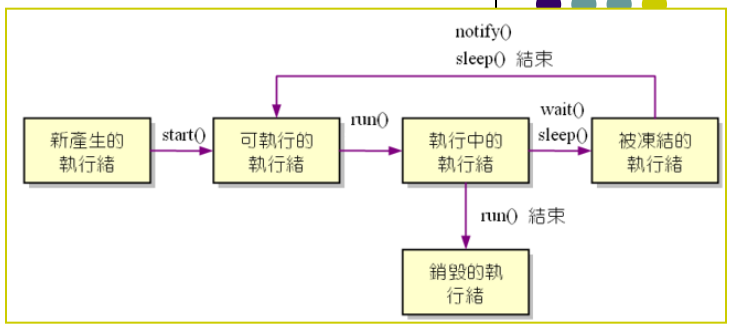
- 當某個類別實作Runnable介面時，在該類別裡必須要實作run()
- 若是繼承Thread類別，則要於該類別中改寫run()

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，均會處於下列五種狀態之一：
  - 新產生的（newly created）
  - 可執行的（runable）
  - 正在執行的（executing）
  - 被凍結的（blocked）
  - 銷毀的（dead）
- 執行緒狀態的轉移與函數之間的關係：

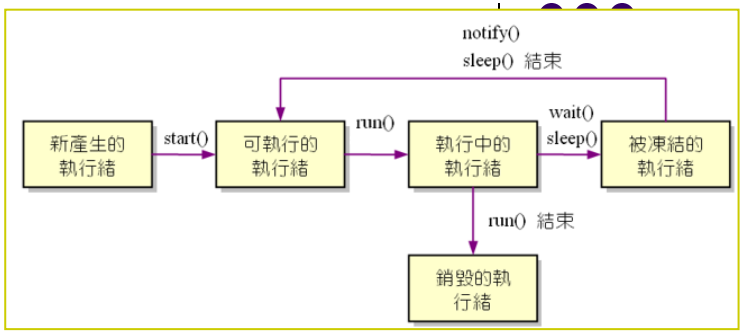


執行緒的生命週期 (2/3)



- 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() 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()，其餘的便在佇列（queue）中等待
- 執行的狀態
  - 當執行緒開始執行run()，會進入執行的狀態。

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：

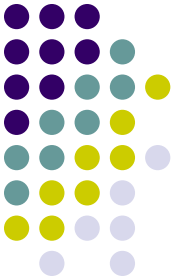
    (1) 該執行緒呼叫物件的wait()

    (2) 該執行緒本身呼叫sleep()

    (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() 所凍結，則該物件的notify() 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
- 銷毀的狀態
  - run() 執行結束或是由執行緒呼叫它的stop() 時，進入銷毀的狀態



wait()與sleep()的差異

(1) wait() 是 Object 的實例函數，sleep() 是 Thread 類別的靜態函數。

(2) wait() 可藉由 notify() 與 notifyAll() 喚醒，sleep() 要等指定的時間結束才會喚醒。

(3) wait() 會在條件滿足前保持等待的狀態，sleep() 只是要拖時間暫緩執行緒的執行。

(4) wait() 的作用是在監控同步處理的物件，用來處理執行緒之間物件同步的問題，因此要在同步處理(synchronized)的函數中被呼叫，sleep() 則不需要，且 sleep() 是直接作用在呼叫它的執行緒。

(5) sleep() 會鎖定呼叫它的執行緒，wait() 則不會。

讓執行緒小睡片刻(1/2)

sleep() 函數的使用範例

```
01  // Ch15_4, sleep() 函數的示範
02  class CTest extends Thread{       // 從 Thread 類別延伸出子類別
03     private String id;
04     public CTest(String str){      // 建構元，設定成員 id
05        id=str;
06     }
07     public void run(){             // 改寫 Thread 類別裡的 run() 函數
08        for(int i=0;i<4;i++){
09           try{
10              sleep((int)(1000*Math.random()));
11           }
12           catch(InterruptedException e){}
13           System.out.println(id+" is running...");
14        }
15     }
16  }
```

sleep() 必須寫在 try-catch 區塊裡

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

catch 接收的必須是InterruptedException例外

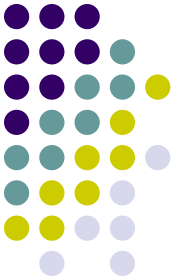
執行結果：

```
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22 行用 cat 物件呼叫 start() 函數

第21 行用 dog 物件呼叫 start() 函數

接續下一頁



讓執行緒小睡片刻(2/2)

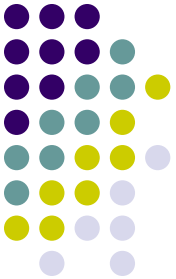
```
17  public class Ch15_4{
18     public static void main(String[] args){
19        CTest dog=new CTest("doggy");
20        CTest cat=new CTest("kitty");
21        dog.start();
22        cat.start();
23     }
24  }
```

執行結果：

```
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 22 行用 cat 物件呼叫 start() 函數

第 21 行用 dog 物件呼叫 start() 函數



等待執行緒 (1/2)

- Ch15_5是執行緒啟動後再加上字串的列印：

```
01  // Ch15_5, 執行緒排程的設計(一)
02  // 將 Ch15_4 的 CTest 類別置於此處
03  public class Ch15_5{
04     public static void main(String[] args){
05        CTest dog=new CTest("doggy");
06        CTest cat=new CTest("kitty");
07        dog.start();                // 用 dog 物件來啟動執行緒
08        cat.start();                    // 用 cat 物件來啟動執行緒
09        System.out.println("main() finished");
10     }
11  }
```

執行結果：

```
main() 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() 本身也是一個執行緒，因此main()執行完第9、10行之後，會往下執行第11行的敘述，通常是第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)

- 下面的範例先執行dog，再執行cat執行緒：

```
01  // Ch15_6, 執行緒排程的設計(二)
02  // 將 Ch15_4 的 CTest 類別置於此處
03  public class Ch15_6{
04     public static void main(String[] args){
05        CTest dog=new CTest("doggy");
06        CTest cat=new CTest("kitty");
07
08        dog.start(); // 啟動 dog 執行緒
09        try{
10           dog.join();     // 限制 dog 執行緒結束後才能往下執行
11           cat.start();    // 啟動 cat 執行緒
12           cat.join();     // 限制 cat 執行緒結束後才能往下執行
13        }
14        catch(InterruptedException e){}
15        System.out.println("main() finished");
16     }
17  }
```

join() 必須寫在 try-catch 區塊裡

會拋出InterruptedException例外

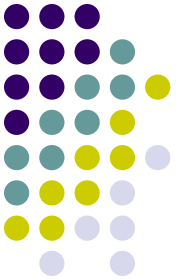
執行結果：

```
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() finished
```

先執行 dog 執行緒

再執行 cat 執行緒

最後再執行第 15 行的敘述



執行緒的優先順序(1/3)

- 執行緒優先順序以數字1~10來表示，數字愈大表示優先權愈高，優先權愈高的愈先進入執行狀態

執行緒優先權相關的函數

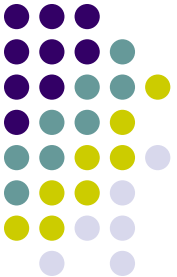
函數	主要功能
void setPriority(int newPriority)	設定執行緒的優先順序，newPriority 的範圍為 1~10
int getPriority()	取得執行緒的優先順序之值

- setPriority() 中的引數newPriority代碼

setPriority() 函數中引數的代碼

代碼	意義
MAX_PRIORITY	最大優先順序之數值
MIN_PRIORITY	最小優先順序之數值
NORM_PRIORITY	系統預設的優先順序之數值

執行緒的優先順序(2/3)



- 設定優先權的範例

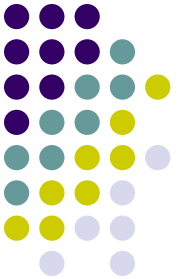
```
01  // Ch15_7, 執行緒的優先順序
02  class CTest extends Thread{        // 從 Thread 類別延伸出子類別
03     private String id;
04     public CTest(String str){       // 建構元，設定成員 id
05        id=str;
06     }
07     public void run(){              // 改寫 Thread 類別裡的 run()
08        for(int i=0;i<3;i++){
09           try{
10              sleep(1000);      // 小睡 1 秒
11           }
12           catch(InterruptedException e){}
13           System.out.println(id+" is running..Priority="
14           +this.getPriority()); // 印出哪個執行緒被執行，並取得優先權值
15        }
16     }
17  }
18
```

執行結果：

```
kitty is running...Priority=10
sheep is running...Priority=5
rabbit is running...Priority=7
horse is running...Priority=3
doggy is running...Priority=1
sheep is running...Priority=5
kitty is running...Priority=10
rabbit is running...Priority=7
horse is running...Priority=3
doggy is running...Priority=1
kitty is running...Priority=10
sheep is running...Priority=5
rabbit is running...Priority=7
horse is running...Priority=3
doggy is running...Priority=1
```


接續下一頁

執行緒的優先順序(3/3)

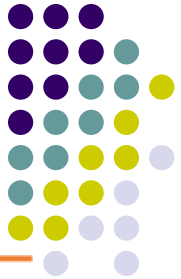


```
19  public class Ch15_7{
20     public static void main(String[] args){
21        CTest dog=new CTest("doggy");
22        CTest cat=new CTest("kitty");
23        CTest rabbit=new CTest("rabbit");
24        CTest sheep=new CTest("sheep");
25        CTest horse=new CTest("horse");
26
27        cat.setPriority(Thread.MAX_PRIORITY);
28        dog.setPriority(Thread.MIN_PRIORITY);
29        rabbit.setPriority(7);
30        horse.setPriority(3);
31
32        dog.start();                    // 啟動執行緒
33        cat.start();
34        rabbit.start();
35        sheep.start();
36        horse.start();
37     }
38  }
```

執行結果：

```
kitty is running...Priority=10
sheep is running...Priority=5
rabbit is running...Priority=7
horse is running...Priority=3
doggy is running...Priority=1
sheep is running...Priority=5
kitty is running...Priority=10
rabbit is running...Priority=7
horse is running...Priority=3
doggy is running...Priority=1
kitty is running...Priority=10
sheep is running...Priority=5
rabbit is running...Priority=7
horse is running...Priority=3
doggy is running...Priority=1
```

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  // Ch15_8, 沒有同步處理的執行緒
02  class Bank{
03     private static int sum=0;
04     public static void add(int n){
05        int tmp=sum;
06        tmp=tmp+n;                    // 累加匯款總額
07        try{
08           Thread.sleep((int)(1000*Math.random()));   // 小睡 0~1 秒鐘
09        }
10        catch(InterruptedException e){}
11        sum=tmp;
12        System.out.println("sum= "+sum);
13     }
14  }
15  class Customer extends Thread{ // Customer 類別，繼承自 Thread 類別
16     public void run(){              // run() 函數
17        for(int i=1;i<=3;i++)
18           Bank.add(100);       // 將 100 元分三次匯入
19     }
20  }
```

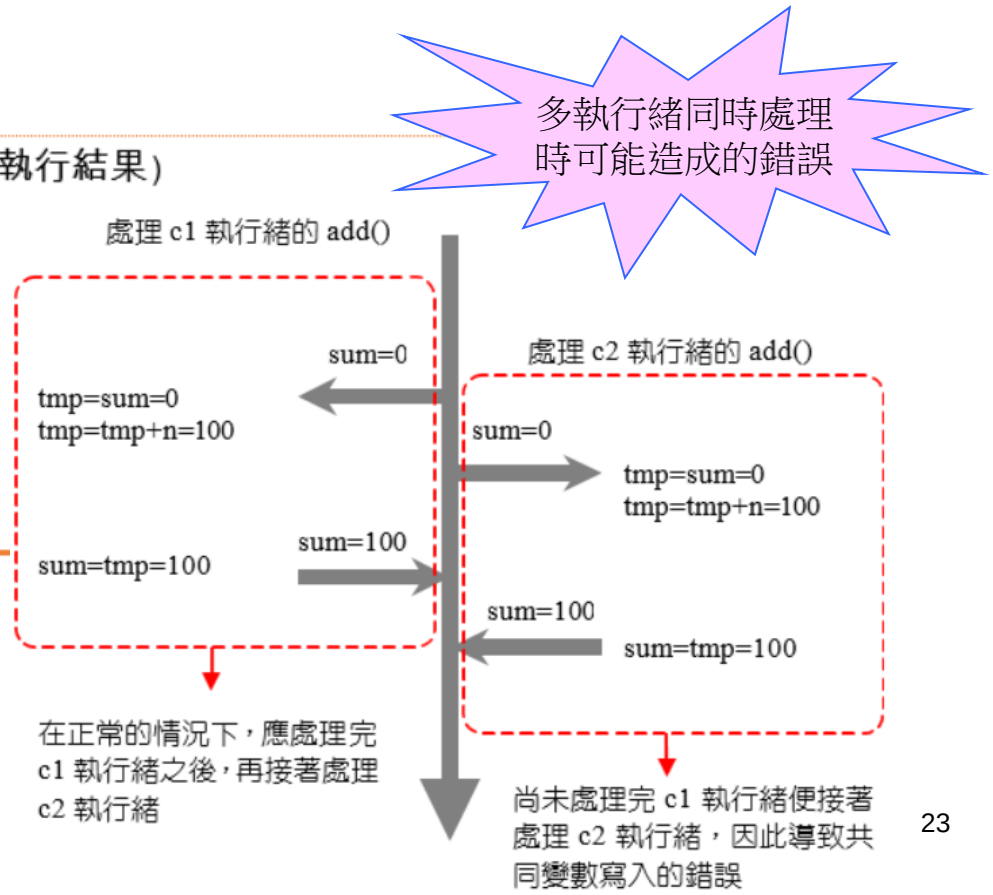
接續下一頁

錯誤的執行緒 (2/2)

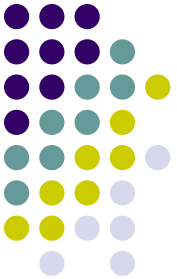

```
21  public class Ch15_8{
22     public static void main(String[] args){
23        Customer c1=new Customer();
24        Customer c2=new Customer();
25        c1.start();
26        c2.start();
27     }
28  }
```

執行結果：----(沒有加 synchronized 的執行結果)

```
sum= 100
sum= 100
sum= 200
sum= 300
sum= 200
sum= 300
```



修正錯誤



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

```
04    public synchronized static void add(int n){
05
          ... ...
13    }
```

在 add() 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add()
- Ch15_8第4行的add() 前加上synchronized，執行結果如下：

```
/* Ch15_8 OUTPUT------(加上 synchronized 的執行結果)
sum= 100
sum= 200
sum= 300
sum= 400
sum= 500
sum= 600
----------------------*/
```

若有多個執行緒共用變數時，要注意存取的順序


-The End-