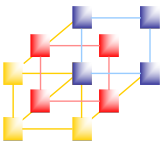
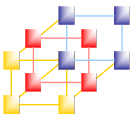
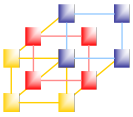


Unit 4
Concurrent Processes



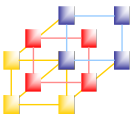
Java 執行緒(thread)的觀念

- 電腦系統常利用多工(multitasking)的方式來提昇效率
- 執行緒(thread)是一種輕量級(lightweight)的執行程式，占用較少的系統資源
  - 執行緒之間共享位址空間(address space)
  - 執行緒執行時的切換(context switching)成本較低
  - 執行緒之間的溝通方便



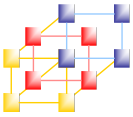
執行緒可分成兩大類

- 使用者執行緒(user threads)
  - 系統在執行應用程式時產生的執行緒 – main()
  - 稱為主執行緒(main thread)
- 常駐執行緒(daemon threads)
  - 系統產生的執行緒
  - 常駐於系統中，直到所有的使用者執行緒都結束才停止

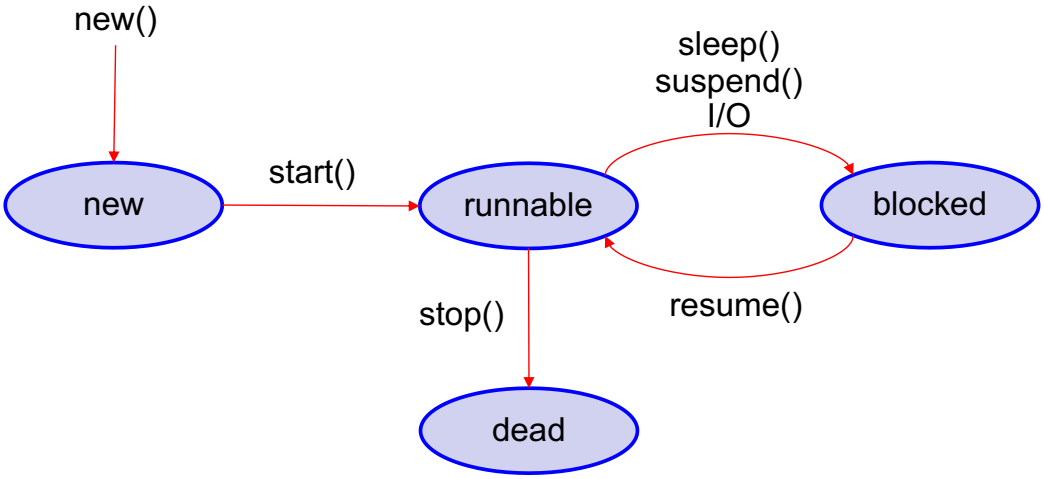


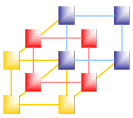
多執行緒程式設計的優點

- 資源的共用
- 提升系統回應的效率 – concurrent processing
- 整體效能的提升 – context switch 的負擔低



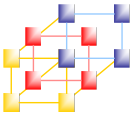
Java 執行緒的狀態變化





Java 程式使用執行緒的兩種方法

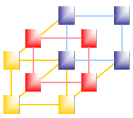
- 實作Runnable介面
  - Runnable 介面中有一個 method – run()
  - 在 Class 定義的建構子中產生Thread 物件
  - 由 start() method 啟動
- 繼承Thread類別
  - 直接定義成 Thread Class 的 Subclass
  - 提供 run() method，然後由 start() method 啟動
- 實作 Runnable 介面時可以再繼承其它類別的特性，直接繼承 Thread 類別沒有這種彈性



Thread 類別的方法

- String getName()：取得執行緒的名稱
- int getPriority()：取得執行緒的優先權值
- boolean isAlive()：執行緒是否存活著？
- void join()：等待執行緒結束，進入終止狀態
- void sleep()：使執行緒進入睡眠狀態

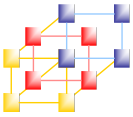
MainProg1.java



建立執行緒—用 Thread 子類別(1/2)

```
class ThreadBySubclass extends Thread
{
        String  ThreadName;
        ThreadBySubclass(String name) // Constructor
        {
                ThreadName = name;
        }
        public void run()              // body of the thread
        {
                Thread  t = Thread.currentThread();
                t.setName(ThreadName);
                System.out.println("Thread " + ThreadName +
                                        " is created!!");
        }
}
```

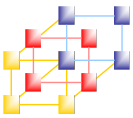
MainProg1.java



建立執行緒—用 Thread 子類別(2/2)

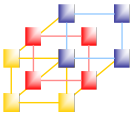
```
public class MainProg1
{
        public static void main(String args[])
        {
                …
                // Create two threads
            ThreadBySubclass thread1 = new  ThreadBySubclass("T1");
            ThreadBySubclass thread2 = new  ThreadBySubclass("T2");

            System.out.println("Number of threads (before):" +
                                        Thread.activeCount());
            thread1.start();
            thread2.start();
            System.out.println("Number of threads (after):" +
                                        Thread.activeCount());
        }
}
```


MainProg2.java



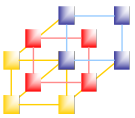
建立執行緒—用 Runnable 介面(1/2)

```
class ThreadByRunnable implements Runnable
{
        ThreadByRunnable(String name) // Constructor
        {
                Thread t = new Thread(this, name);
                t.start();
        }
        public void run()      // body of the thread
        {
                System.out.println("Thread " +
        (Thread.currentThread()).getName()
                                + " is created!!");
        }
}
```



建立執行緒—用 Runnable 介面(2/2)

```
public class MainProg2
{
        public static void main(String args[])
        {
                …
                // Create two threads
                ThreadByRunnable thread1
                        = new ThreadByRunnable("T1");
                ThreadByRunnable thread2
                        = new ThreadByRunnable("T2");
                System.out.println("Number of threads :" +
                         Thread.activeCount());
        }
}
```

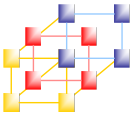



Java執行緒中常用的method (1/2)

- setPriority( )方法
  - 設定 Thread 優先順序
  - 最高: 10，最低: 1，一般:5
- sleep( )方法
  - 強迫 Thread 進入睡眠狀態，以千分之一秒為單位

```
try
{
        System.out.println(ThreadName + " sleep 1 second!!");
        Thread.sleep(1000);
}
catch(InterruptedException e)
{
        System.out.println(ThreadName + " Interrupt!!");
}
```

MainProg3.java

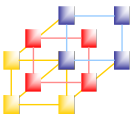


Java執行緒中常用的method (2/2)

- join( )方法
  - 呼叫的 thread 等待被呼叫的 thread 執行結束

```
ThreadBySybclass thread1 = new ThreadBySybclass("T1");

 …

try
{
        System.out.println(t.getName() + " wait Thread1!!");
        thread1.join();
}
catch(InterruptedException e)
{
        System.out.println(t.getName() + " Interrupt!!");
}
```


MainProg4.java
MainProg5.java



同時性控制 (Synchronization Control)

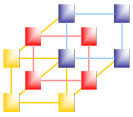
- 當多個 Thread 會存取同一資源，而這一資源一次僅允許一個 Thread 存取時即需要同時性控制 (Critical Section Problem)
- 方法一

```
public static Object  lock = new Object();
       …
synchronized(lock)
{
       // Critical Section
}
```

- 方法二

```
synchronized static void CriticalSection(String name)
{
       // Critical Section
}
```

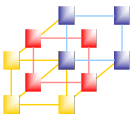
MainProg6.java



執行緒的溝通 (1/3)

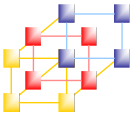
```
class Resource
{
        int             count = 0;
        synchronized void Create()
        {
                System.out.println("Creating .....");
                try
                {
                        Thread.sleep(3000);
                }
                catch(InterruptedException e) { }
                count = 1;
                notify();
        }
```


MainProg6.java



執行緒的溝通 (2/3)

```
        synchronized int GetResult()
        {
                System.out.println("Waiting for notify.....");
                try
                {
                        wait();
                }
                catch(InterruptedException e) { }
                return count;
        }
}
```



執行緒的溝通 (3/3)

```
class Producer extends Thread
{
        Resource resource;
        Producer(Resource r)   {
                resource = r;
                start();
        }
        public void run()
        {
                resource.Create();
        }
}
```

```
class Consumer extends Thread
{
        Resource resource;
        Consumer(Resource r)
        {
                resource = r;
                start();
        }
        public void run()
        {
        System.out.println(
                "Count = " +
        resource.GetResult());
        }
}
```