

Java程式設計進階

Java多執行緒應用程式

鄭安翔

ansel.cheng@hotmail.com

課程大綱

- 1) 執行緒 **Thread**
 - 繼承**Thread**類別
 - 實作**Runnable**介面
- 2) 執行緒基本控制
- 3) 執行緒資料同步

多工運算 Multi-Tasking

■ 多工 Multi-Tasking

- 多個程序(Process)同時在電腦上執行
 - 採用分時(Time Sharing)方式均分一個CPU使用時間
- 多執行緒 Multi-Thread
 - Process中再作多工運算

■ 多工的好處：提高執行效率

- 避免大量處理輸出入或網路資料傳遞,造成空轉
- 充分利用CPU運算資源

建立執行緒 Thread

- 二種建立Thread的方法
 - 類別繼承 `java.lang.Thread` 類別
 - Thread 類別實作 `Runnable` 介面
 - 類別實作 `java.lang.Runnable` 介面
 - `Runnable` 介面宣告 `public void run()` 方法

繼承 Thread 類別

- 類別繼承 Thread 類別
 - 寫一個新類別，繼承 Thread 類別
 - 覆寫 `public void run()` 方法
 - 建構一個新類別物件
 - 執行其`start()`方法 (繼承自Thread 類別)

實作 Runnable 介面

- 類別實作 Runnable 介面
 - 寫一個新類別，實作 Runnable 介面
 - 提供 `public void run()` 方法
 - 建構一個新類別物件
 - 建構一個 `Thread` 物件，以新類別物件作傳入參數
 - 執行 `Thread` 物件之 `start()` 方法

```

01 public class HelloThread1 extends Thread {
02     String name;
03     public HelloThread1(String n) {
04         name = n;
05     }
06     public void run() {
07         for (int i=1; i<=100; i++)
08             System.out.println(name+" Hello "+i);
09     }
10 }

```

```

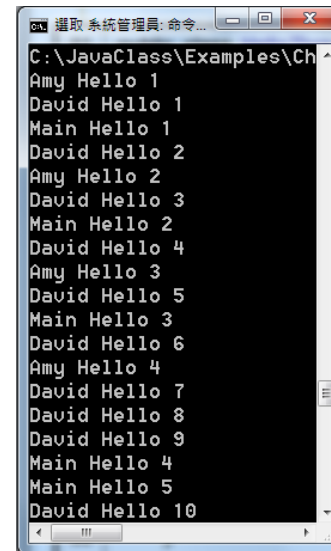
01 public class ThreadsExample1 {
02     public static void main(String argv[]){
03         Thread t1 = new HelloThread1("Amy");
04         Runnable r2 = new HelloRunner1("David");
05         Thread t2 = new Thread(r2);
06         t1.start();
07         t2.start();
08         for (int i=1; i<=100; i++)
09             System.out.println("Main Hello "+i);
10     }
11 }

```

```

01 public class HelloRunner1 implements Runnable {
02     String name;
03     public HelloRunner1(String n) {
04         name = n;
05     }
06     public void run() {
07         for (int i=1; i<=100; i++)
08             System.out.println(name+" Hello "+i);
09     }
10 }

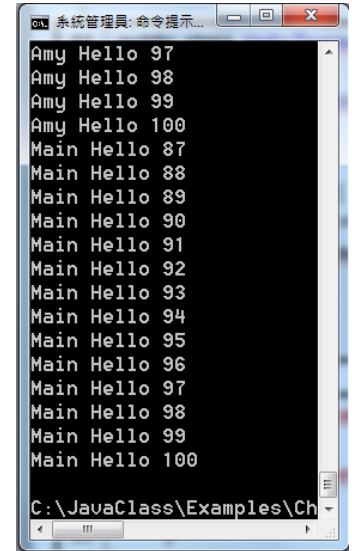
```



```

C:\JavaClass\Examples\Ch
Amy Hello 1
David Hello 1
Main Hello 1
Amy Hello 2
David Hello 2
Main Hello 2
Amy Hello 3
David Hello 3
Main Hello 3
Amy Hello 4
David Hello 4
Main Hello 4
Amy Hello 5
David Hello 5
Main Hello 5
Amy Hello 6
David Hello 6
Main Hello 6
Amy Hello 7
David Hello 7
Main Hello 7
Amy Hello 8
David Hello 8
Main Hello 8
Amy Hello 9
David Hello 9
Main Hello 9
Amy Hello 10
David Hello 10
Main Hello 10

```



```

C:\JavaClass\Examples\Ch
Amy Hello 97
David Hello 98
Main Hello 99
Amy Hello 100
David Hello 101
Main Hello 102
Amy Hello 103
David Hello 104
Main Hello 105
Amy Hello 106
David Hello 107
Main Hello 108
Amy Hello 109
David Hello 110
Main Hello 111
Amy Hello 112
David Hello 113
Main Hello 114
Amy Hello 115
David Hello 116
Main Hello 117
Amy Hello 118
David Hello 119
Main Hello 120

```

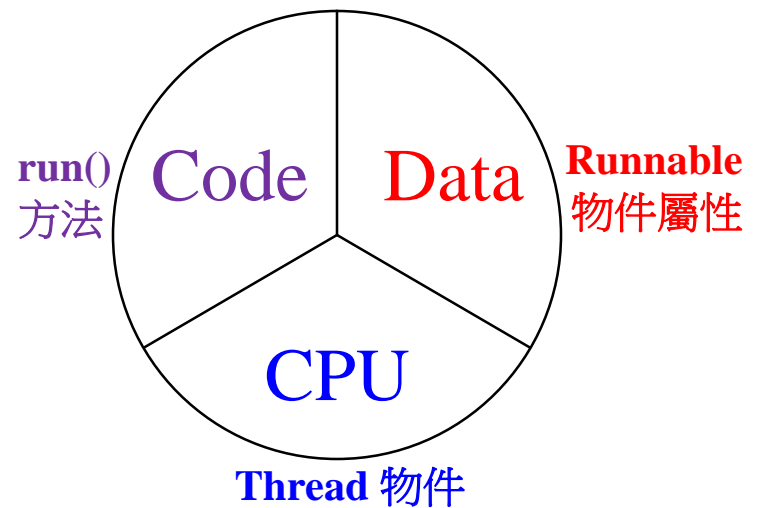
Threads vs. Runnable

■ 類別實作 Runnable 介面

- 較符合物件導向精神與架構
- **Java**類別只能單一繼承,卻可實作多個介面
- 程式的一致性

■ 類別繼承 Thread 類別

- 程式碼較簡單



課程大綱

- 1) 執行緒 Thread
- 2) 執行緒基本控制
 - 啟動執行緒
 - 停止執行緒
 - 暫停執行緒
 - 取得執行緒資訊
- 3) 執行緒資料同步

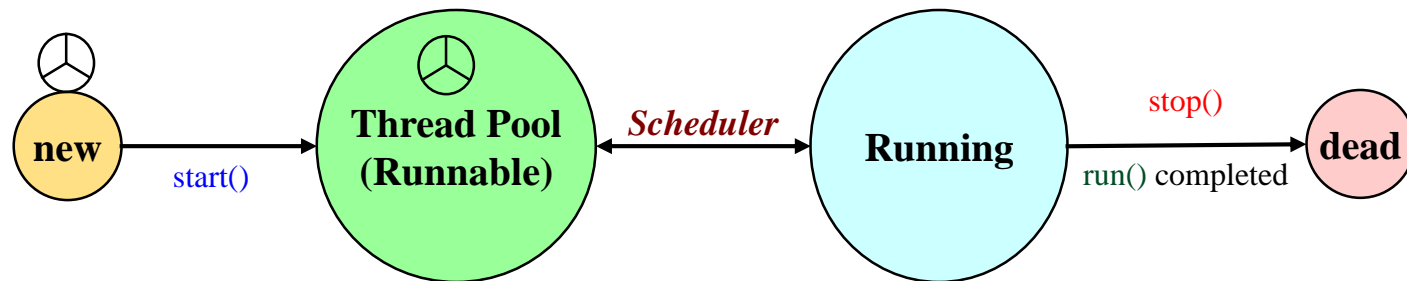
Thread 類別

建構子	說明
Thread(Runnable target)	建立以Target物件中定義的run()方法為執行目標的執行緒

常用方法	傳回值	說明
start()	void	啟動執行緒
stop()	void	停止執行緒 (Deprecated)
static sleep(long millis) throws InterruptedException	void	暫停正在執行的執行緒指定的毫秒數,時間到後恢復可執行狀態
static yield()	void	正在執行的執行緒讓出執行緒使用權
join() throws InterruptedException	void	等待此執行緒進入死亡狀態後,恢復可執行狀態
join(long millis) throws InterruptedException	void	等待此執行緒進入死亡狀態後恢復可執行狀態,若等待時間超過指定毫秒數,也恢復可執行狀態

Threads 狀態及操作

- 啟動 Thread
 - `public void start()`
- 停止 Thread
 - 執行完 `run()`
 - `public final void stop()` – **Deprecated**



Threads 狀態及操作

■ 暫停Thread

- **sleep**方法：暫時停止thread物件的執行一段時間,時間到了進入可執行狀態等待繼續執行

public static void sleep(long millis) throws InterruptedException

- **yield**方法:停止執行,轉移到等待狀態,將CPU的使用權讓出來

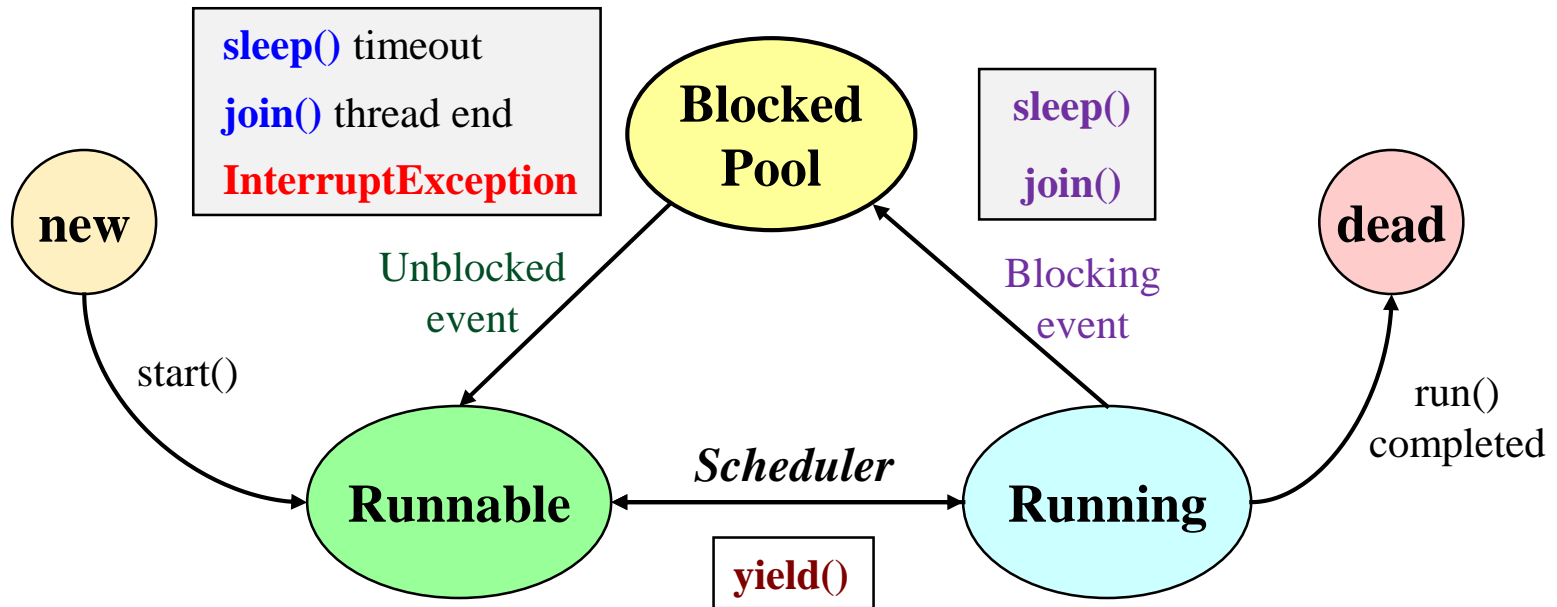
public static void yield()

- **join**方法:某個thread等待另一個thread執行完後才能夠被執行

public final void join() throws InterruptedException

public final void join(long millis) throws InterruptedException

Threads 狀態及操作



```

01 public class HelloThread2 extends Thread {
02     String name;
03     public HelloThread2(String n) {
04         name = n;
05     }
06     public void run() {
07         for (int i=1; i<=10; i++) {
08             System.out.println(name+" Hello "+i);
09             try {
10                 Thread.sleep(1);
11             } catch (InterruptedException ie){ }
12         }
13     }
14 }

```

```

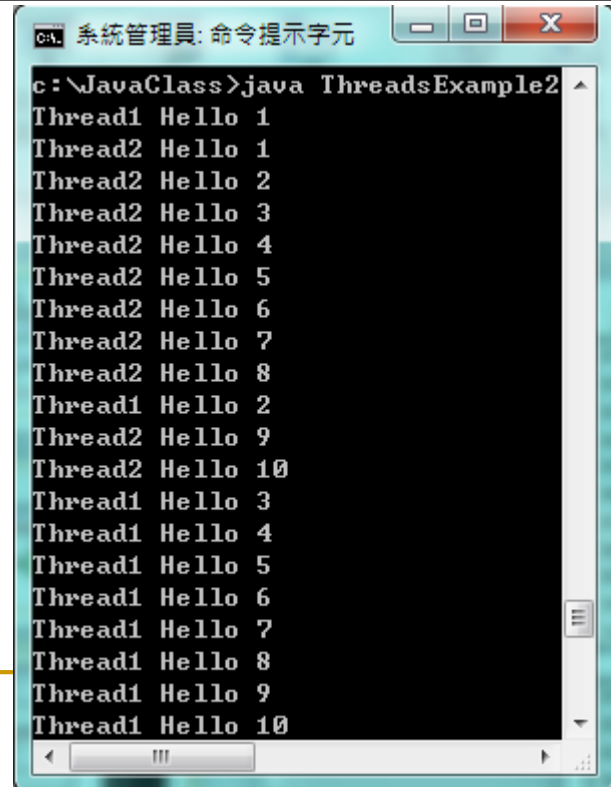
01 public class HelloRunner2 implements Runnable {
02     String name;
03     public HelloRunner2(String n) {
04         name = n;
05     }
06     public void run() {
07         for (int i=1; i<=10; i++) {
08             System.out.println(name+" Hello "+i);
09             Thread.yield();
10         }
11     }
12 }

```

```

01 public class ThreadsExample2 {
02     public static void main(String argv[]){
03         Thread t1 = new HelloThread2("Thread1");
04
05         Runnable r2 = new HelloRunner2("Thread2");
06         Thread t2 = new Thread(r2);
07
08         t1.start();
09         t2.start();
10     }
11 }

```



```

c:\JavaClass>java ThreadsExample2
Thread1 Hello 1
Thread2 Hello 1
Thread2 Hello 2
Thread2 Hello 3
Thread2 Hello 4
Thread2 Hello 5
Thread2 Hello 6
Thread2 Hello 7
Thread2 Hello 8
Thread1 Hello 2
Thread2 Hello 9
Thread2 Hello 10
Thread1 Hello 3
Thread1 Hello 4
Thread1 Hello 5
Thread1 Hello 6
Thread1 Hello 7
Thread1 Hello 8
Thread1 Hello 9
Thread1 Hello 10

```

```

01 public class MotherThread implements Runnable {
02     public void run() {
03         System.out.println("媽媽準備煮飯");
04         System.out.println("媽媽發現米酒用完了");
05         System.out.println("媽媽叫兒子去買米酒");
06         Thread son = new Thread(new SonThread());
07         son.start();
08         System.out.println("媽媽等兒子把米酒買回來");
09         try {
10             son.join();
11         } catch (InterruptedException ie) {
12             System.err.println("發生例外！");
13             System.err.println("媽媽中斷煮飯");
14             System.exit(1);
15         }
16         System.out.println("媽媽開始煮飯");
17         System.out.println("媽媽煮好飯了");
18     }
19 }

```

```

01 public class Cooking {
02     public static void main(String argv[]) {
03         Thread mother = new Thread(new MotherThread());
04         mother.start();
05     }
06 }

```

```

01 public class SonThread implements Runnable {
02     public void run() {
03         System.out.println("兒子出門去買米酒");
04         System.out.println("兒子買東西來回需5分鐘");
05         try {
06             for (int i=1; i<=5; i++) {
07                 Thread.sleep(1000);
08                 System.out.print(i+"分鐘 ");
09             }
10         } catch (InterruptedException ie) {
11             System.err.println("兒子發生意外");
12         }
13         System.out.println("\n兒子買米酒回來了");
14     }
15 }

```

```

c:\JavaClass>java Cooking
媽媽準備煮飯
媽媽發現米酒用完了
媽媽叫兒子去買米酒
媽媽等兒子把米酒買回來
兒子出門去買米酒
兒子買東西來回需5分鐘
1分鐘 2分鐘 3分鐘 4分鐘 5分鐘
兒子買米酒回來了
媽媽開始煮飯
媽媽煮好飯了

c:\JavaClass>

```

取得Thread資訊

常用方法	傳回值	說明
isAlive()	boolean	測試執行緒是否仍然存活
<code>static</code> interrupted()	boolean	測試執行緒是否被中斷
interrupt()	void	中斷執行緒
setDaemon(boolean on)	void	設定執行緒是否為Daemon thread (與主執行緒執行獨立)
isDaemon()	boolean	測試執行緒是否為Daemon thread
setName(String name)	void	更改執行緒的name屬性
getName()	String	取得執行緒的name屬性
setPriority(int newPriority)	void	設定執行緒的執行優先權
getPriority()	int	取得執行緒的執行優先權
<code>static</code> currentThread()	Thread	取的正在執行的執行緒的物件參考

課程大綱

- 1) 執行緒Thread
- 2) 執行緒基本控制
- 3) 執行緒資料同步
 - 執行緒中共用資料同步問題
 - 執行緒同步 **Synchronized**
 - 死結**Deadlock**

共用資料

■ 記憶體中的共用資料

- 類別屬性
- 物件屬性
- 多執行緒執行時可能有同步問題(Thread Unsafe)

■ 非共用資料

- 區域變數
- 方法傳入參數
- 例外物件
- 多執行緒執行時無同步問題(Thread Safe)

多執行緒同步問題

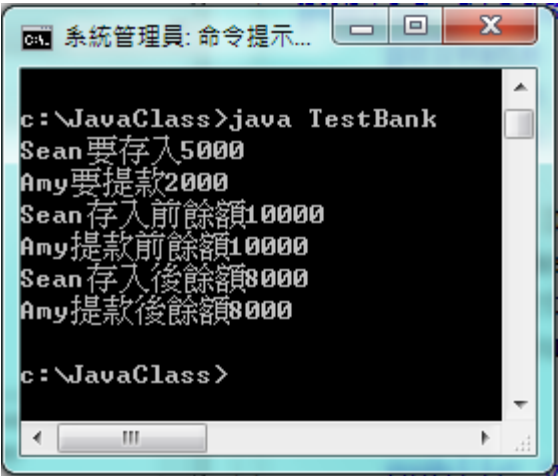
```
01 public class Account{
02     private int balance;
03     public Account(int initBal){
04         balance = initBal;
05     }
06     public void deposit(int amt){
07         int b = balance;
08         for(int i=0; i<100000; i++);
09         b += amt;
10         balance = b;
11     }
12     public void withdraw(int amt){
13         int b = balance;
14         for(int i=0; i<100000; i++);
15         b -= amt;
16         balance = b;
17     }
18     public int getBalance(){
19         return balance;
20     }
21 }
```

```
01 public class DepositTask implements Runnable{
02     private String name;
03     private Account acct;
04     private int amount;
05     public DepositTask(String n, Account a, int amt) {
06         name = n;    acct = a;    amount = amt;
07     }
08     public void run(){
09         System.out.println(name+"要存入"+amount);
10         System.out.println("存入前餘額"+acct.getBalance());
11         acct.deposit(amount);
12         System.out.println("存入後餘額"+acct.getBalance());
13     }
14 }
```

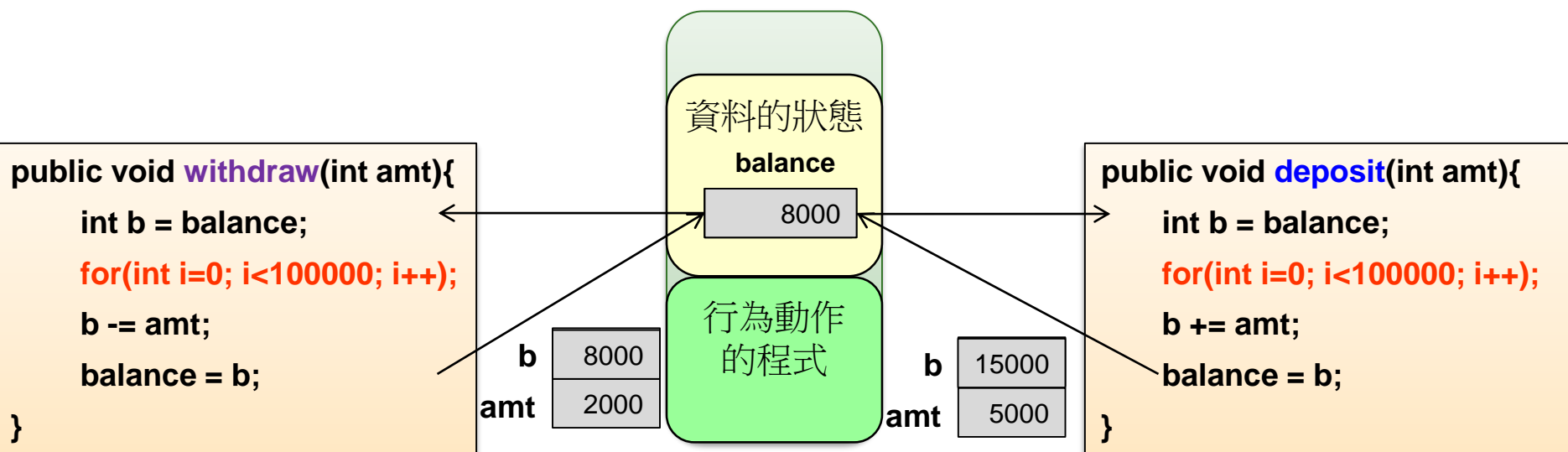
```
01 public class WithdrawTask implements Runnable{
02     private String name;
03     private Account acct;
04     private int amount;
05     public WithdrawTask(String n, Account a, int amt) {
06         name = n;    acct = a;    amount = amt;
07     }
08     public void run(){
09         System.out.println(name+"要提款"+amount);
10         System.out.println("提款前餘額"+acct.getBalance());
11         acct.withdraw(amount);
12         System.out.println("提款後餘額"+acct.getBalance());
13     }
14 }
```

多執行緒同步問題

```
01 public class TestBank {  
02     public static void main(String[] args){  
03         Account acct = new Account(10000);  
04         DepositTask r1 = new DepositTask("Sean", acct, 5000);  
05         WithdrawTask r2 = new WithdrawTask("Amy", acct, 2000);  
06         Thread t1 = new Thread(r1);  
07         Thread t2 = new Thread(r2);  
08         t1.start();  
09         t2.start();  
10     }  
11 }
```

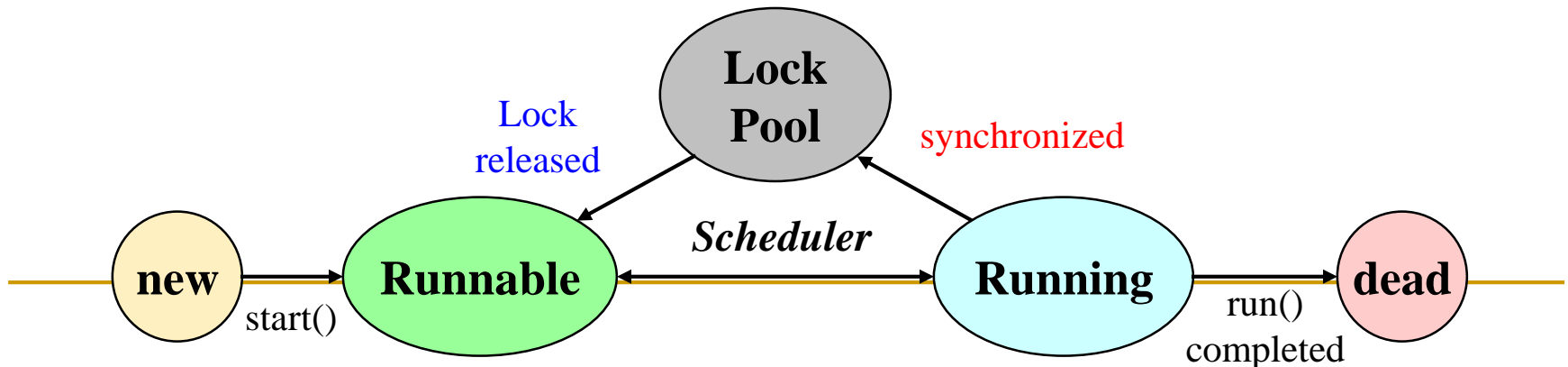


```
c:\JavaClass>java TestBank  
Sean 要存入5000  
Amy 要提款2000  
Sean 存入前餘額10000  
Amy 提款前餘額10000  
Sean 存入後餘額8000  
Amy 提款後餘額8000  
  
c:\JavaClass>
```



Synchronize 機制

- 某些程式碼不可被分割執行
- 每一個物件都有一個鎖("lock flag")
- **Synchronized** 表示要執行該段程式碼時,必須先取得某個物件的鎖
- 也可以鎖定類別類別屬性或類別方法
 - Account.class



Synchronized 語法

- Synchronized block

```
synchronized (要取得lock的物件/類別) {  
    //需鎖定的程式碼  
}
```

- Synchronized method

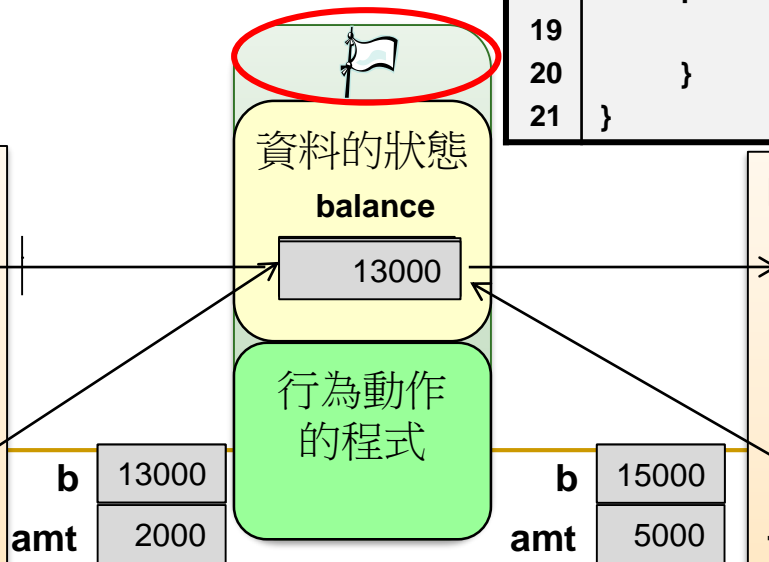
```
public synchronized void methodName(){  
    //需鎖定的方法內容  
}
```

資料同步範例

```
系統管理員: 命令提示...
c:\JavaClass>java TestBank
Amy要提款2000
Sean要存入5000
Sean存入前餘額10000
Amy提款前餘額10000
Sean存入後餘額15000
Amy提款後餘額13000
c:\JavaClass>
```

```
01 public class Account{
02     private int balance;
03     public Account(int initBal){
04         balance = initBal;
05     }
06     public synchronized void deposit(int amt){
07         int b = balance;
08         for(int i=0; i<100000; i++);
09         b += amt;
10         balance = b;
11     }
12     public synchronized void withdraw(int amt){
13         int b = balance;
14         for(int i=0; i<100000; i++);
15         b -= amt;
16         balance = b;
17     }
18     public int getBalance(){
19         return balance;
20     }
21 }
```

```
public synchronized void
    withdraw(int amt){
    int b = balance;
    for(int i=0; i<100000; i++);
    b -= amt;
    balance = b;
}
```



```
public synchronized void
    deposit(int amt){
    int b = balance;
    for(int i=0; i<100000; i++);
    b += amt;
    balance = b;
}
```

```

01 public class DepositTask implements Runnable{
02     private String name;
03     private Account acct;
04     private int amount;
05     public DepositTask(String n, Account a, int amt) {
06         name = n;    acct = a;    amount = amt;
07     }
08     public void run(){
09         System.out.println(name+"要存入"+amount);
10         synchronized(acct) {
11             System.out.println("存入前餘額"+acct.getBalance());
12             acct.deposit(amount);
13             System.out.println("存入後餘額"+acct.getBalance());
14         }
15     }
16 }

```

```

C:\JavaClass>javac TestBank.java

C:\JavaClass>java TestBank
Sean要存入5000
Amy要提款2000
Amy提款前餘額10000
Amy提款後餘額8000
Sean存入前餘額8000
Sean存入後餘額13000

C:\JavaClass>

```

```

01 public class WithdrawTask implements Runnable{
02     private String name;
03     private Account acct;
04     private int amount;
05     public WithdrawTask(String n, Account a, int amt) {
06         name = n;    acct = a;    amount = amt;
07     }
08     public void run(){
09         System.out.println(name+"要提款"+amount);
10         synchronized(acct) {
11             System.out.println("提款前餘額"+acct.getBalance());
12             acct.withdraw(amount);
13             System.out.println("提款後餘額"+acct.getBalance());
14         }
15     }
16 }

```

```

C:\JavaClass>java TestBank
Amy要提款2000
Amy提款前餘額10000
Amy提款後餘額8000
Sean存入前餘額8000
Sean存入後餘額13000

C:\JavaClass>

```

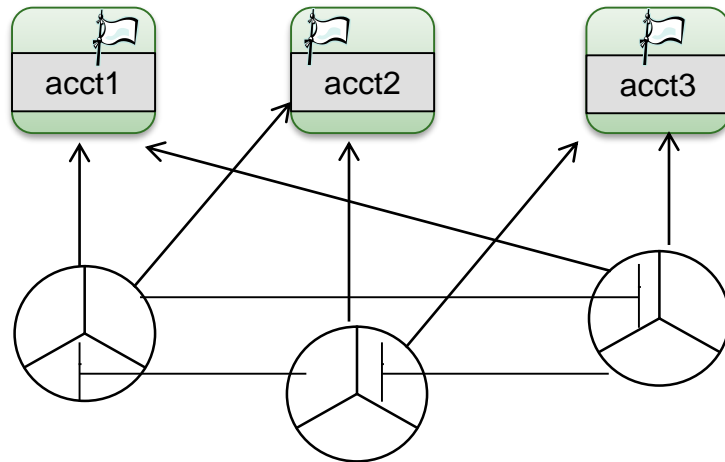

Synchronized block vs. method

	Synchronized block	Synchronized method
鎖定範圍	Block	Method
可鎖定物件	指定物件/類別	本身物件/類別
文件顯示同步	不能	能

死結 Death Lock

■ 死結 Death Lock

- 兩個或多個 threads,各自在等待對方擁有的鎖



- Java未提供death lock的偵測,避免及處理機制
- 程式時可用下列原則來預防Death Lock :
 - 決定lock取得的優先順序
 - 所有程式嚴格遵守lock的取的順序
 - 釋放lock時,依相反順序釋放