Java 課程

第13章：執行緒

大綱

- 執行緒入門
- 同步化議題
- concurrent套件新增類別

繼承Thread

- 繼承java.lang.Thread類別，並重新定義run()方法
- 實例化您自定義的Thread類別
- 使用start()方法啟動執行緒

```
public class EraserThread extends Thread {
    private boolean active;
    private String mask;

    …
    // 重新定義run()方法
    public void run () {
        while(isActive()) {
            System.out.print(mask);
          try {
                // 暫停目前的執行緒50毫秒
              Thread.currentThread().sleep(50);
               }
               catch(InterruptedException e) {
                   e.printStackTrace();
               }
        }
    }
}
```

繼承Thread

```
// 啟動 Eraser 執行緒
EraserThread eraserThread = new EraserThread('#');
eraserThread.start();
String password = scanner.next();
eraserThread.setActive(false);
```

- 在Java SE 6中可以使用System.console()來取得java.io.Console物件
- 使用Console物件的readPassword()方法，就可以避免輸入的密碼被窺視的問題

實作 Runnable 介面

- 如果您的類別已經要繼承某個類別，那麼您就不能繼承Thread類別
- 繼承了Thread類別，您就不能再繼承其它類別
- 實作java.lang.Runnable介面來定義具執行緒功能的類別
- Runnable介面中定義一個run()方法要實作
- 在實例化一個 Thread物件時，可以傳入一個實作Runnable介面的物件作為引數

實作 Runnable 介面

```
public class Eraser implements Runnable { // 實作Runnable
    private boolean active;
    private String mask;
   …
    // 重新定義run()方法
    public void run () {
        while(isActive()) {
            System.out.print(mask);
        try {
                // 暫停目前的執行緒50毫秒
           Thread.currentThread().sleep(50);
             }
             catch(InterruptedException e) {
                 e.printStackTrace();
             }
        }
    }
}
```

實作 Runnable 介面

```
// Eraser實作Runnable介面
Eraser eraser = new Eraser('#');
// 啟動 Eraser 執行緒
Thread eraserThread = new Thread(eraser);
eraserThread.start();
String password = scanner.next();
eraser.setActive(false);
```

Daemon 執行緒

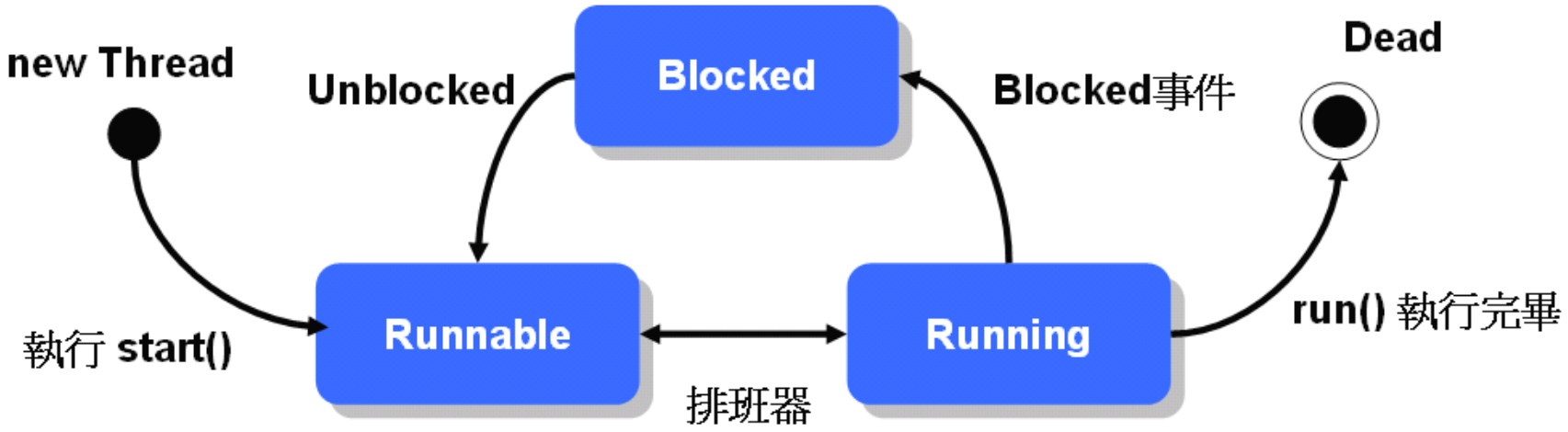
- 一個Daemon執行緒是一個在背景執行服務的執行緒
- 如果所有的非Daemon的執行緒都結束了，則Daemon執行緒自動就會終止
- 從Main方法開始的是一個非Daemon執行緒
- 如果希望某個執行緒在產生它的執行緒結束後跟著終止，要將它設為Daemon執行緒

- 使用setDaemon()方法來設定一個執行緒是否為Daemon執行緒
- 預設所有從Daemon執行緒產生的執行緒也是Daemon執行緒

```
Thread thread = new Thread(
// 這是匿名類別的寫法
    new Runnable() {
        public void run() {
            while(true) {
                System.out.print("T");
            }
        }
    });
// 設定為Daemon執行緒
thread.setDaemon(true);
thread.start();
```

執行緒生命周期

- 執行start()之後，執行緒進入Runnable狀態，此時執行緒尚未真正開始執行
- 必須等待排班器（Scheduler）的排班



執行緒生命周期

- 執行緒有其優先權，由1（Thread.MIN_PRIORITY）到10（Thread.MAX_PRIORITY）
- 優先權越高，排班器越優先排入執行，如果優先權相同，則輸流執行（Round-robin方式）

執行緒生命周期

- 如果您想要讓目前執行緒禮讓一下其它執行緒，讓它們有機會取得執行權，您可以呼叫緒行緒的yield()方法

```
// .....
      Thread thread = new Thread(new Runnable() {
         public void run() {
              // ....
              while(true) {
                  // ....
                  yield();  // 暫時讓出執行權
              }
          }
       });
      thread.start();     // ....
```

執行緒生命周期

- 有幾種狀況會讓執行緒進入Blocked狀態
  - 等待輸入輸出完成
  - 呼叫sleep()方法
  - 嘗試取得物件鎖定
  - 呼叫wait() 方法

執行緒生命周期

- 進入Blocked狀態，以下的幾個對應情況讓執行緒回到Runnable狀態
  - 輸入輸出完成
  - 呼叫interrupt()
  - 取得物件鎖定
  - 呼叫notify()或notifyAll()

執行緒生命周期

```
Thread thread = new Thread(new Runnable() {
    public void run() {
        try {
            // 暫停99999毫秒
            Thread.sleep(99999);
        }
        catch(InterruptedException e) {
            System.out.println("I'm interrupted!!");
        }
    }
});

thread.start();
thread.interrupt(); // interrupt it right now
```

執行緒的加入（join）

- 當執行緒使用 join()加入至另一個執行緒時，另一個執行緒會等待這個被加入的執行緒工作完畢，然後再繼續它的動作
- join()的意思表示將執行緒加入成為另一個執行緒的流程之一

執行緒的加入（join）

```
Thread threadB = new Thread(new Runnable() {
    public void run() {
        try {
            …
        }
        catch(InterruptedException e) {
            e.printStackTrace();
        }
    }
});
threadB.start();
try {
    // Thread B 加入 Thread A
    threadB.join();
}
catch(InterruptedException e) {
    e.printStackTrace();
}
```

執行緒的停止

- 不建議使用stop()來停止一個執行緒的運行

```
public class SomeThread implements Runnable {
      private boolean isContinue = true;
      public void terminate() {
          isContinue = false;
      }
      public void run() {
          while(isContinue) {
              // ... some statements
          }
      }
}
```

執行緒的停止

- 不建議使用stop()來停止一個執行緒的運行

```
Thread thread = new Thread(new SomeThread());
thread.start();
thread.interrupt();
```

ThreadGroup

- 每一個執行緒產生時，都會被歸入某個執行緒群組
- 如果沒有指定，則歸入產生該子執行緒的執行緒群組中
- 可以自行指定執行緒群組，執行緒一但歸入某個群組，就無法更換群組

ThreadGroup

- java.lang.ThreadGroup類別正如其名，可以統一管理整個群組中的執行緒

```
ThreadGroup threadGroup1 = new ThreadGroup("group1");
ThreadGroup threadGroup2 = new ThreadGroup("group2");
Thread thread1 = new Thread(threadGroup1, "group1's member");
Thread thread2 = new Thread(threadGroup2, "group2's member");
```

- ThreadGroup中的某些方法，可以對所有的執行緒產生作用
  - interrupt()方法可以interrupt群組中所有的執行緒
  - setMaxPriority()方法可以設定群組中執行緒所能擁有的最大優先權

ThreadGroup

- 想要一次取得群組中所有的執行緒來進行某種操作，可以使用enumerate()方法

```
Thread[] threads = new Thread[threadGroup1.activeCount()];
threadGroup1.enumerate(threads);
```

ThreadGroup

- uncaughtException()方法是當群組中某個執行緒發生非受檢例外（Unchecked exception）時，由執行環境呼叫進行處理

```
ThreadGroup threadGroup1 =
// 這是匿名類別寫法
    new ThreadGroup("group1") {
        // 繼承ThreadGroup並重新定義以下方法
        // 在執行緒成員丟出unchecked exception
        // 會執行此方法
        public void uncaughtException(Thread t, Throwable e)
            System.out.println(t.getName() + ": "
                    + e.getMessage());
        }
    };
```

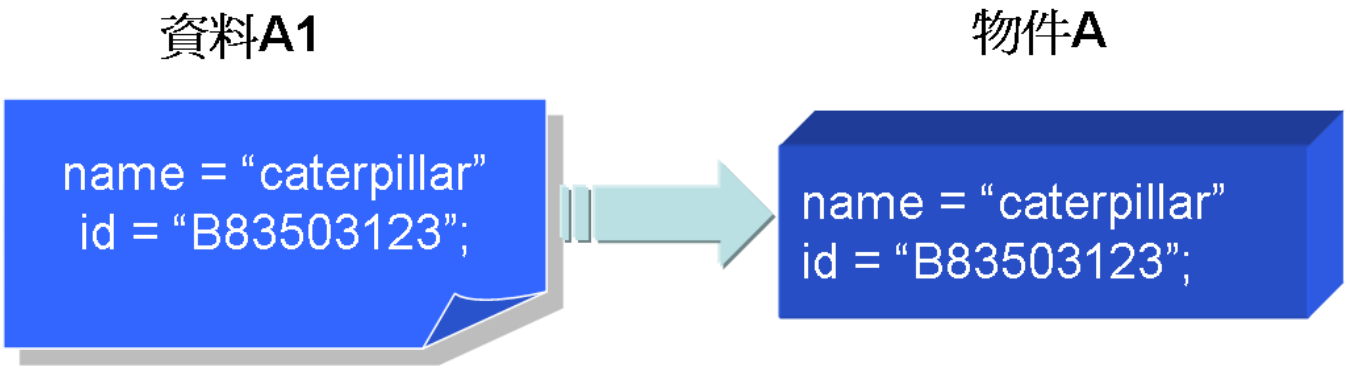
UncaughtExceptionHandler

- 可以讓您的例外處理類別實作 Thread.UncaughtExceptionHandler介面，並實現其 uncaughtException()方法

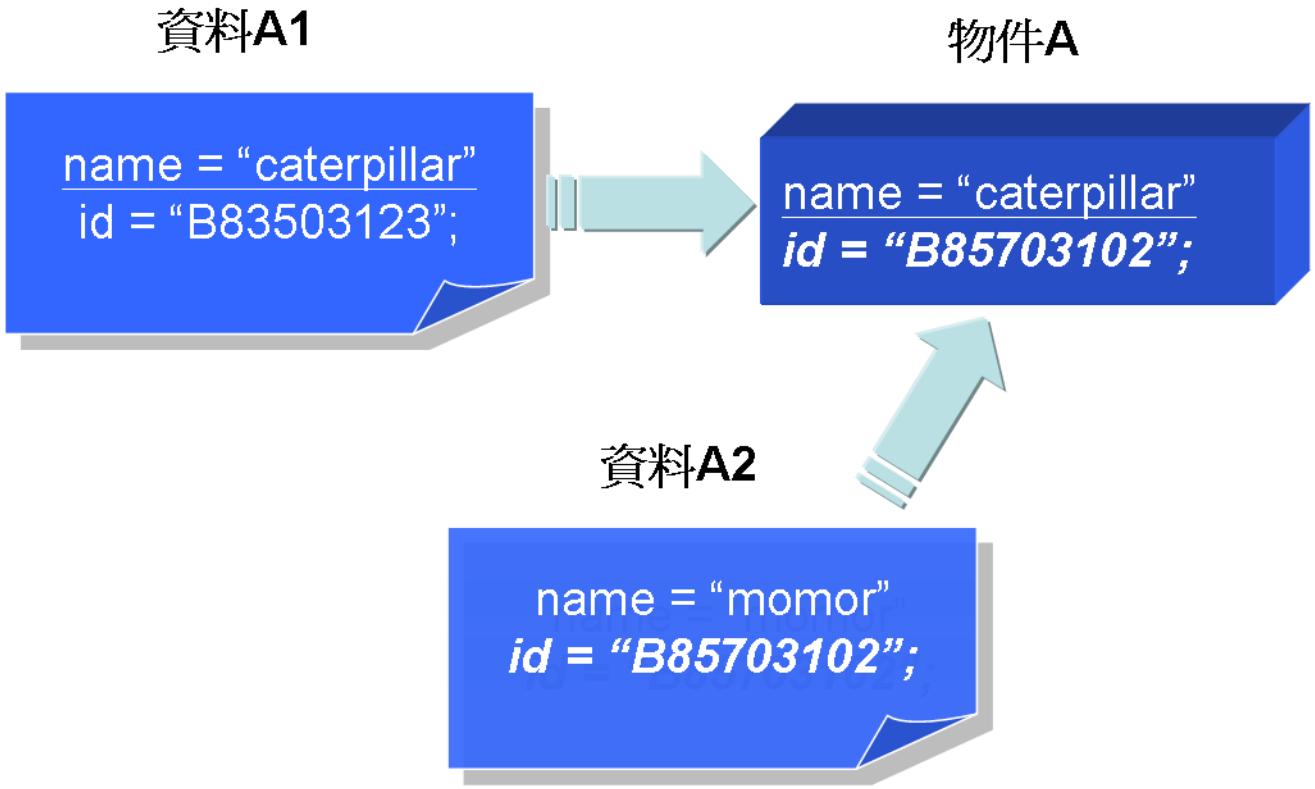
```
public class ThreadExceptionHandler
            implements Thread.UncaughtExceptionHandler {
    public void uncaughtException(Thread t, Throwable e) {
        System.out.println(t.getName() + ": "
                           + e.getMessage());
    }
}
thread1.setUncaughtExceptionHandler(handler);
```

同步化

- 如果一個物件所持有的資料可以被多執行緒同時共享存取時，您必須考慮到「資料同步」的問題
- 資料同步指的是兩份資料整體性、一致性



同步化


資料A1
name = “caterpillar”
id = “B83503123”;
物件A
name = “caterpillar”
id = “B85703102”;
資料A2
name = “momor”
id = “B85703102”;

同步化

- 資料的不同步而可能引發的錯誤通常不易察覺
- 可能是在您程式執行了幾千幾萬次之後，才會發生錯誤
- 這通常會發生在您的產品已經上線之後，甚至是程式已經執行了幾年之後

同步化

```
public void setNameAndID(String name, String id) {
    this.name = name;
    this.id = id;
    if(!checkNameAndIDEqual()) {
        System.out.println(count +
                ") illegal name or ID.....");
    }
    count++;
}
```

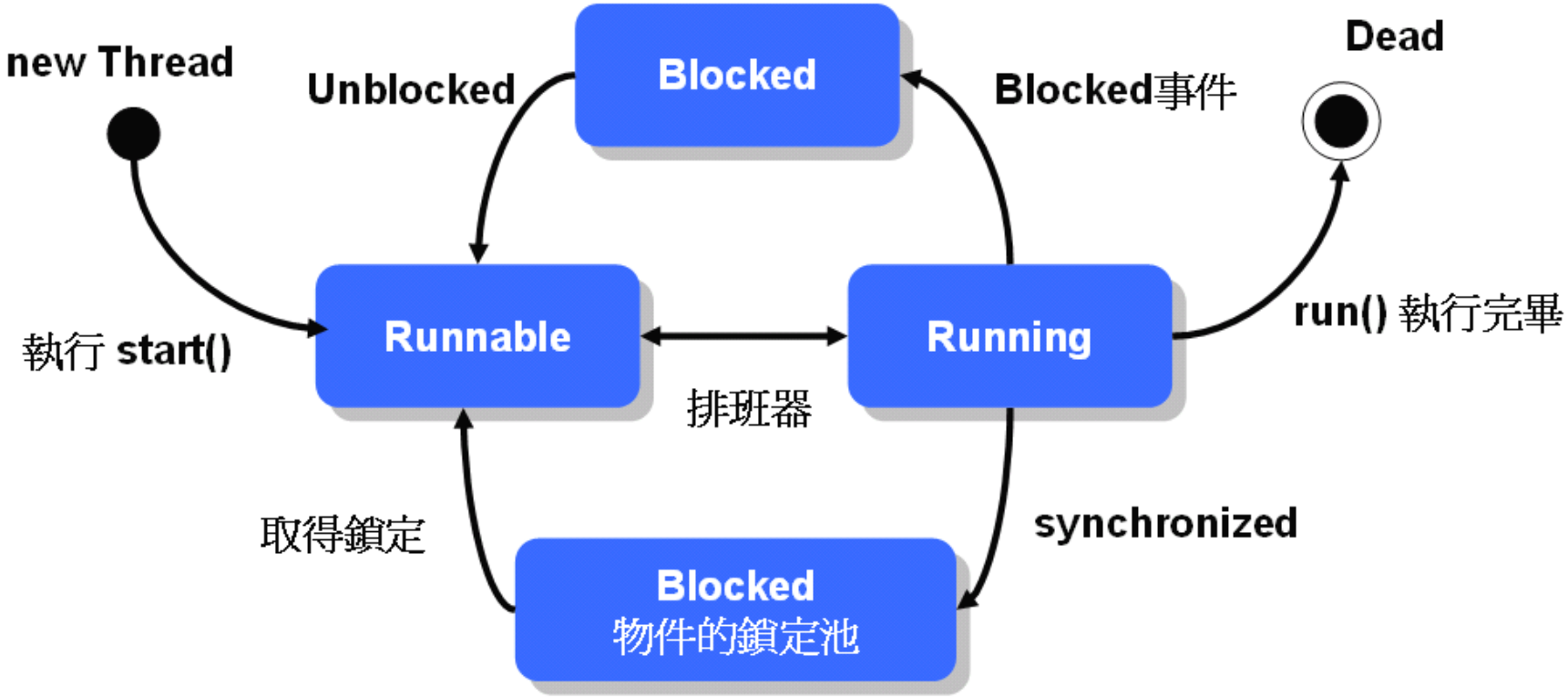
同步化

- 使用"synchronized"關鍵字

```
public synchronized void setNameAndID(String name, String id) {
      this.name = name;
      this.id = id;
      if(!checkNameAndIDEqual()) {
        System.out.println(count +
                  ") illegal name or ID.....");
      }
      count++;
}
```

同步化

- 物件的鎖定（lock）觀念



使用"synchronized"關鍵字

```
public void setNameAndID(String name, String id) {
    synchronized(this) {
        this.name = name;
        this.id = id;
        if(!checkNameAndIDEqual()) {
            System.out.println(count +
                ") illegal name or ID.....");
        }
        count++;
    }
}
// arraylist參考至一個ArrayList的一個實例
synchronized(arraylist) {
    arraylist.add(new SomeClass()
);
```

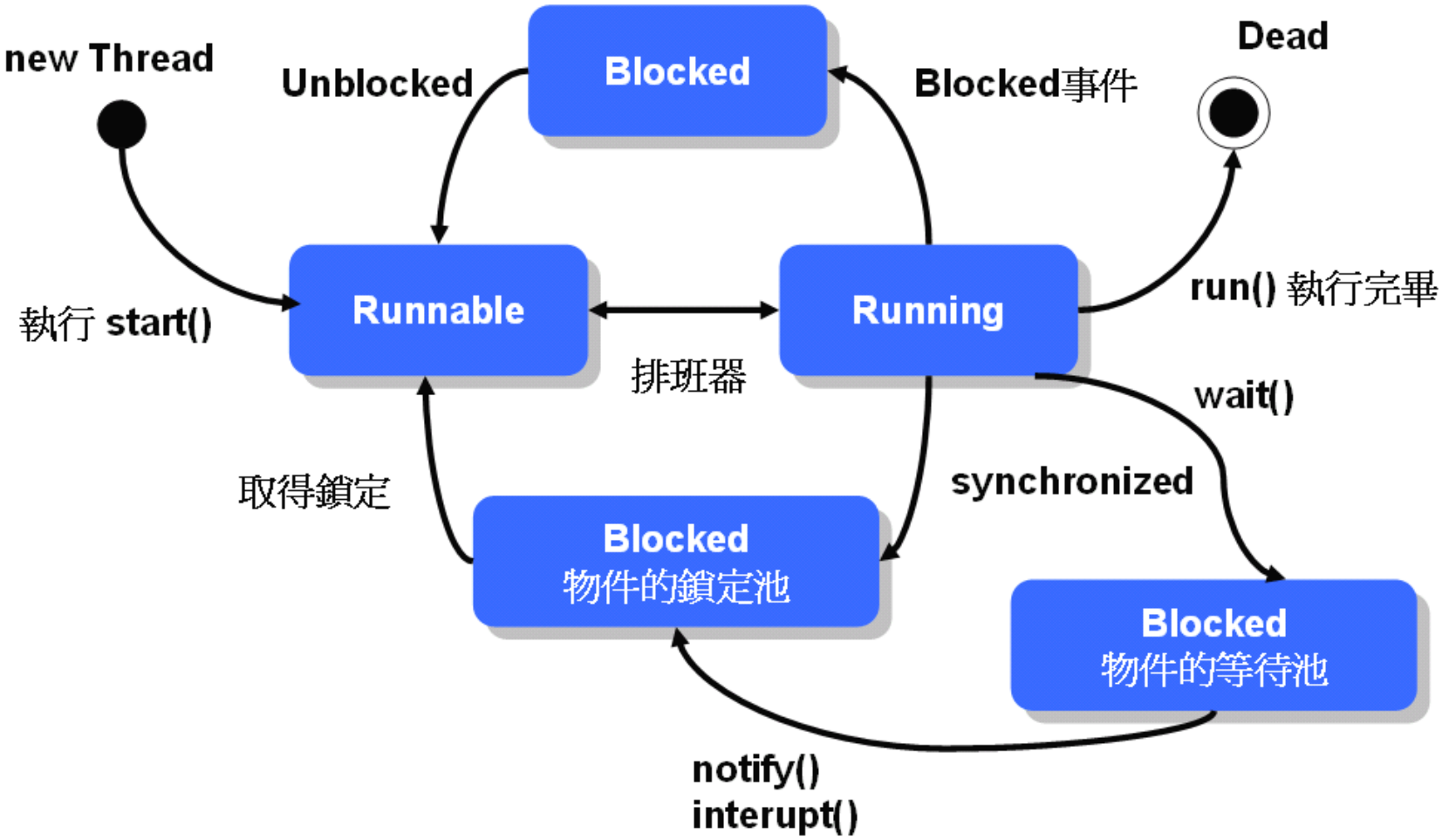
同步化

- 同步化確保資料的同步，但所犧性的就是在於一個執行緒取得物件鎖定而佔據同步化區塊，而其它執行緒等待它釋放鎖定時的延遲

wait()、notify()

- wait()、notify()與notifyAll()是由Object類別所提供的方法
- 宣告為"final"
- 在同步化的方法或區塊中呼叫wait()方法
- 當物件的wait()方法被調用，目前的執行緒會被放入物件的等待池中，執行緒歸還物件的鎖定
- 其它的執行緒可競爭物件的鎖定

wait()、notify()


new Thread
Unblocked
Blocked
Blocked事件
Dead
執行 start()
Runnable
Running
run() 執行完畢
排班器
wait()
取得鎖定
synchronized
Blocked
物件的鎖定池
Blocked
物件的等待池
notify()
interupt()

wait()、notify()

- 當物件的notify()被調用，它會從目前物件的等待池中通知「一個」執行緒加入回到鎖定池的Blocked狀態
- 被通知的執行緒是隨機的，被通知的執行緒會與其它執行緒共同競爭物件的鎖定
- 如果您呼叫notifyAll()，則「所有」在等待池中的執行緒都會被通知回到鎖定池的Blocked狀態

wait()、notify()

- 當執行緒呼叫到物件的wait()方法時，表示它要先讓出物件的鎖定並等待通知，或是等待一段指定的時間
- 被通知或時間到時再與其它執行緒競爭物件的鎖定
- 如果取得鎖定了，就從等待點開始執行

wait()、notify()

```
public synchronized void setProduct(int product) {
    if(this.product != -1) {
        try {
            // 目前店員沒有空間收產品，請稍候！
            wait();
        }
        catch(InterruptedException e) {
            e.printStackTrace();
        }
    }

    this.product = product;
    System.out.printf("生產者設定 (%d)%n", this.product);
    // 通知等待區中的一個消費者可以繼續工作了
    notify();
}
```

```
public synchronized int getProduct() {
    if(this.product == -1) {
        try {
            // 缺貨了，請稍候！
            wait();
        }
        catch(InterruptedException e) {
            e.printStackTrace();
        }
    }

    int p = this.product;
    System.out.printf("消費者取走 (%d)%n", this.product);
    this.product = -1; // 取走產品，-1表示目前店員手上無產品

    // 通知等待區中的一個生產者可以繼續工作了
    notify();

    return p;
}
```

容器類的執行緒安全

- 可以使用java.util.Collections的synchronizedXXX()等方法來傳回一個同步化的容器物件

```
List list = Collections.synchronizedList(new ArrayList());
```

- 使用Iterator遍訪物件時，您仍必須實作同步化

```
List list = Collections.synchronizedList(new ArrayList());
...
synchronized(list) {
      Iterator i = list.iterator();
      while (i.hasNext()) {
          foo(i.next());
      }
}
```