異常、轉型、變數

異常

- 沒有發生 Exception 時，try-catch 並不會有明顯的額外負擔
- 發生 Exception 時，try-catch 會有明顯的額外負擔。因為要取得當時的堆疊樣本，以便印出 Stack Trace
- 若該程式很注重 Stack Trace，則需 throw new Exception 取得該錯誤的 Stack Trace；否則應重複使用既有的 Exception 物件
- p.s.若要取得該 Exception 的 Stack Trace 則需使用 fillInStackTrace()，但這也是我們避免重複產生新物件的主要原因

轉型

- 轉型成本：`interface > class`、位階越高(牽扯層面廣)成本越高
- 轉型最好能避免就避免，與其使用「通用型」的集合類別，倒不如依照資料製作專屬的類別
- 如果某個變數需要被多次轉型，則應在第一次結束後，把該資料存入暫存物件

變數

- Method 的參數個數對程式的影響微乎其微
- 若要對實體變數或靜態變數進行繁複操作，應先暫時儲存變數，然後將執行結果設定給原來的變數
- 存取陣列元素會比一般元素來得慢
- 任何物件的操作一定比基本型態的暫時性變數來得慢
- 將物件傳給 method 能有效降低程式所使用的物件個數，也比在 method 中產生新物件更有效率

總結

- 將所有錯誤狀況檢查置於 `if` 裡面
- 避免在程式的常態流程丟出 `Exception`
  - 檢察瓶頸中的 try-catch 是否會造成額外成本
  - 檢查物件是否屬於預期的類別時，使用 instanceof 來代替 try-catch
  - 若在丟出 Exception 時，不需要讓 Exception 物件含有當時的 Stack Trace，考慮重複使用先前產生的 Exception 實體
  - 測試效能時，記得包含程式的正常流程會產生的任何 Exception

- 將轉型的次數降低
  - 製作 type-specific 的集合類別來避免轉型動作
  - 使用暫時變數來儲存轉型完的資料，以避免重複轉型
  - 定義變數的型態時，盡可能選用精確度最適當的型態

- 使用區域變數代替實體與靜態變數
  - 使用暫時變數來操作實體變數、靜態變數、陣列元素
  - 優先考慮使用 int 資料型態
  - 避免使用 long 和 double 的實體或靜態變數
  - 需要暫時變數，盡量使用基本資料型態，而非物件
  - 考慮直接存取實體變數，而非透過 accessor method(但會違反封裝性)
  - 要避免 method 呼叫額外附加的 method，可考慮多添加參數

流程控制

- 將不需要每一回合執行的程式碼移出迴圈，包括：assignment、存取、測試與 method call
- 將迴圈內的 method call 換成等效的傳回值或程式碼
- 陣列存取的負擔高於暫時變數，所以「陣列存取」最好在迴圈外解決
- VM 會針對-1,0,1,2,3,4,5 這些整數的相比運算，若將迴圈的測試條件改寫成「與 0 相比」會讓迴圈稍快一些(新的 VM 會對迴圈條件進行最佳化，所以不一定有用)
- 避免使用 method call 做為迴圈結束的條件
- Ex: for(i = 0; i < ls.size(); i++) V.S. int tmp2 = ls.size(); for(int c = 0; i < tmp2; i++)
- 若需要在迴圈裡面進行測試，試著將「相等性」比較（兩值是否相等）改成「同一性」比較（是否為同一物件）
- 若有多項條件測試在同一迴圈中時，試著讓最有可能發生的狀況提早判斷出來，藉此提早結束迴圈
- 避免在迴圈內使用 reflection。因為在調查 method 的名稱會造成負擔，且 method.invoke() 比一般 method call 的代價更高，而 method reference 的處理也相當複雜
- 用 exception 結束迴圈（不建議使用）
- 如果本質上適合用遞迴法來處理的問題，建議先用遞迴解法，除非遇到效能瓶頸才要考慮其他解法

總結

- 盡可能讓迴圈做最少事
  - 將不必在每一回執行的程式碼移到迴圈外
  - 將任何重複執行卻得到相同結果的程式碼移出迴圈，在進入迴圈之前，將這些值指定給暫時變數
  - 盡量避免在迴圈裡面呼叫 method，必要時改寫迴圈或用 inline 技術處理掉 method call
  - 若多次存取或更新相同的陣列元素，應先用暫時變數代替，等到迴圈結束再存回陣列
  - 避免在測試迴圈的結束條件使用 method call
  - 盡可能使用 int，尤其是資料變數
  - 用 System.arraycopy()複製陣列
  - 盡可能將「相等性比較」替換成「同一性比較」
  - 盡可能將多個邏輯測試合併，使其盡快發生短路
  - 消除迴圈中不必要的暫時變數
  - 試著將迴圈展開到若干程度，察看是否能改善速度（不了解）
- 改寫 switch 敘述，使所有 case 值能形成一個連續範圍
- 檢察 recursive 是否還有加速餘地
  - 將 recursive method 轉換成 iterative method
  - 將 recursive method 轉換成「漸縮遞迴」(tail recursive)
  - 試著快取「遞迴過程的中間值」，以減少遞迴深度

- 使用暫時變數來取代傳入參數，以便將使用單一搜尋路徑的 recursive method 轉換成 iterative method
- 使用暫時堆疊來取代傳入參數，以便將使用多重搜尋路徑的 recursive method 轉換成 iterative method

I/O、日誌登載、主控台輸出

- 設計程式時，建議用「分工式 I/O」代替「輪詢」(polling) 和「多緒 I/O」
- 在背景執行 I/O。理論上，表示 App 不用等待 I/O；實務上，通常能預先執行某些「讀取」動作，而某些「寫入」動作則可以非同步方式執行 (不用確認 I/O 是否成功)
- 避免在迴圈內使用 I/O。盡量以「少數幾次高資料量的 I/O」取代「多次小資料量的 I/O」
- 執行 I/O 時，若需要同時進行其他動作，建議將兩者分離，以減少 I/O 的次數 (每次的資料量提高)
- 如果要重複存取不同位置的同一組檔案，可以將這些檔案保持在開啟狀態，減少對檔案的開關
- 預先配置好檔案需要的空間
- 同時使用多個檔案可以改善效能，但是必須在「開/關檔的成本」與「多重 open 所需的額外資源」之間取得平衡，循序開關多個檔案對效能並不好。
- 平時開發程式時，將測試程式碼用 if 包起來，然後用變數做控制，因為編譯器在編譯時會將絕對不可能執行到的程式碼移除，所以可以用一個 final static 變數來啟動日誌登載功能。或是利用 Logger 物件進行控制。
- 做 I/O 處理盡量使用 buffer 來提升效能
- 若要用網路傳輸，最好用「壓縮/解壓縮」
- 若要將大量資料存在硬碟上，盡量不要將其放在一個大檔案裡面，而是分散在多個檔案裡，以減少花費繁複演算法搜尋的代價