HomeWork2: Tracing the Code

說明:

在這個 lab 練習中，提供給你一個別人撰寫的 UML editor。這個程式要完成的規格，也會以附件提供給你。不過這個程式有 3 個 bugs，撰寫程式的原作者，則已經到美國矽谷去工作了，無法幫你。你的工作就是了解這個程式出了什麼問題，追蹤程式碼，然後在修復這 3 個 bug。

評分的方法:

1. 總共有 3 個 bug。修一個 bug 50 分，兩個 80 分，3 個 100 分
2. 你一但找到關鍵的程式碼，通常修復都非常簡單

追蹤與理解一個不是你寫的不熟悉的程式，通常是一件嚇人的事情。尤其是，程式語言可能不是你熟悉的，其應用領域的 programming model 你也有可能非常陌生。最糟糕的是如果程式碼還不小。一般人會認為那就把程式看懂才可以改 bug。但是等到程式碼都看懂了大概已經一個月以後了。你的老闆希望你在越短的時間修復這幾隻 bugs。

所以什麼是最快速的 program understanding 的方式?

提示:(高手的 code tracing 技巧，我已經都寫在這裡了，我們也歡迎你自行採用你自以為是的方法)

1. 請你先大致閱讀過規格書，規格書所描述的就是正確的軟體行為
2. 請你先將程式編譯起來，使得程式可以執行
3. 請你先玩熟所有的功能，並且開始一一對照規格書中所條列的正確行為。
4. 你應該會發現一些詭異的地方，也就是 bug。這時候請你記錄怎麼樣固定的操作順序可以重現 bug。先找到一條可以重現的測試（test）是程式追蹤的一個重要開始。在這個地方你想重現的 test run 當然是能儘快找到 bug 的 test run。
5. 先瀏覽一下 source code 的檔案。可以打開某些檔案看一下。如果檔案名稱以及裡面的 source code 變數命名還算清楚，其實光看檔名，class name, method name 你大概也能知道這個程式在做什麼，先大致上瀏覽並自行歸納一下。本 lab 提供的程式碼算是可讀性非常高。好的程式碼檔名與變數名稱，method 名稱與規格書或設計文件(如果有的話)都會呈現某種正相關。

正相關越高自然可讀性越高。如果正相關很低，恭喜你，你的追蹤難度將大幅增加。因為這迫使讀程式的人去猜測。猜測越多，困難度就越高。

6. 對一般人而言，程式語言熟悉與否的確會影響 code tracing，但是對高手而言，影響並不會那麼大。其實，你如果從來沒有寫過 GUI 的程式，這次的 lab 可能影響還會更大一點。所以你如果完全沒有寫過 GUI 程式，請找助教先了解一下 GUI 一般的 programming model 是如何進行的。
7. 高手會用 debugger 除錯，所以先將你的程式以除錯的模式編譯。如果你連 debugger 都不會用，那有點糟糕，請趕緊看老師提供的投影片，與影片。
8. 假設你記錄下來的造成錯誤的 test run，其操作順序是 A->B->C->D->E，ABCDE 指的是某一種輸入的動作例如某個滑鼠點擊畫面或按鈕，輸入資料等等。
9. 假設你的第一個動作 A 是一個滑鼠點擊的動作，你可以視情況採取下列的其中一個方法，利用除錯器讓你的 program 在觸發 A 事件之後停下來。
   A. 在所有的 method 起頭都加入中斷點，然後將程式執行起來之後進行滑鼠點擊的動作，理論上你的程式應該會立即停在一個 method 的起始。這個方法看起來很笨，沒有錯，萬一 method 的數量有成千上百你應該如何? 你如果想自動地為每一個 method 設定中斷點，是你的 IDE 環境而定。通常方法都是有的。Such as http://stackoverflow.com/questions/11625732/can-i-set-breakpoints-to-all-methods-in-a-class-at-once-in-visual-studio
   不過在本 lab 有時候笨方法卻是最快的方法。因為這個程式不大，很快你就可以攔截到你進行動作 A 之後的程式進入點。

   B. 假設你對系統的程式語言，GUI 平台還算熟悉或者你瀏覽過程式碼，你知道滑鼠點擊的動作出發的 method 大概都會被叫做 *mouseclick* 之類的。利用 IDE 搜尋這些字串，你也可以很快的一一設定中斷點，然後試看看你是否能夠在進行 A 的滑鼠點擊之後讓程式停下來。

   C. 如果你的程式不是 GUI， 你可以慢慢從 main() 設中斷點，一步一步 step 到 A 的觸發點。如果程式不大，你追蹤到一定的範圍，你其實就會知道該如何設定中斷點在那些地方。也就是說，先花點時間，使用 debugger 了解一下程式起頭。雖然沒有效率，但是卻是全面理解程式碼的一種開始。但是這一招對大而複雜的程式可能非常耗時。
   D. (歡迎再提供新方法)

10. 你可以看出來，上述的步驟就是利用 debugger 快速地幫你在不熟悉的程

式碼中找到 test run 的程式進入點。

11. 你的接下來的目標就是理解 A->B 中間的程式碼。最快的理解方式就是藉由 debugger 的 step 或是 next 一步一步執行，並且觀察關鍵變數的變化，所以你得打開 watch window 將關鍵的變數都輸入進去
12. 觀察一下 debugger 提供的 call stack window。你可以知道你的程式是如何從 main() 一路呼叫哪一些 method 然後到達 A 事件。所以這時候你如果有興趣，想了解是誰呼叫 A 的 method，請利用 call stack 跳到上一層了解 A 的呼叫者 (caller)。Debugger 會幫你切換到 caller 的 source code。
13. 接下來，你可以慢慢 step 或是 next 直到你的程式到達 B，但是這個過程也有可能很漫長，如果你沒有耐心，你也可以照前述的方法設定許多中斷點，continue/resume 你的程式然後讓程式到達 B。再藉由 call stack 回朔了解程式是如何由 A 到達 B。
14. 了解 A->B->C->D 中間程式到底在幹什麼，可以很快也可以很慢，可以忽略也可以只看重點，要看你追蹤程式的目的是什麼。以 fix bug 而言，A->B->C->D 可能與 bug 都無關，而最重要的關鍵可能是 D->E。但是如果你是要擴充或進行大幅度的程式修改，那慢慢了解 A->B->C->D 中間的主程式架構與邏輯，可能就很重要，就必須花更多的時間。通常建議一邊追蹤，一邊將架構圖畫下來。(喔!不要懷疑， legacy code 不見得有架構圖)

15. 假設你千辛萬苦終於知道 bug 發生在 E 之後，可以先暫時 disable A->B->C->D 的中斷點。你可能必須重複執行很多次，才能了解 bug 的原因。了解了之後，剩下的問題就是你如何修 bug 了。
16. …… 願原力與你同在 good luck。

Summary:

1. 高手追蹤不熟悉程式的訣竅就是善用 debugger
2. 漫無目的的 reading code 通常效果有限而且緩慢，不管事 fix 還是全面理解程式碼，你必須先尋找一個你關心的 test run，經過你有興趣的模組
3. 如前面所說的，運用巧思設定中斷點，快速找到你有興趣的程式碼進入點,
4. 善用 call stack 觀察中斷點前後的來龍去脈。
5. 程式語言不熟悉往往沒有太大的關係，任何程式語言的指令都只是在改變關鍵的變數值。當你在 next/step 的時候，觀察前後變數的變化，你往往可以了解演算法的細節。如果層次高一點程式碼可讀性若不錯，光看 method 的名稱，你就應該知道程式碼大致上在做什麼，是否 step in 進

去了解實作細節，或是 next 直接跳過，就可以自行斟酌。當然如果你想要追蹤的陌生程式碼可讀性很差，method 名稱都無法直接聯想它要做的事情，那你只能好好的用 step 一步一步猜測作者的思想。當然，追蹤的難度也隨之以等比級數增加。

6. 其實你並不需要熟悉一個程式語言才能進行 code tracing。高手甚至是可以藉由上述的 code tracing 來快速熟悉與學會一個新的語言
7. Code tracing 不能缺的一項人格特質就是耐心與細心。Trace legacy code 很辛苦，但是不斷地抱怨並不能增長你什麼，通常你所寫的程式碼可能比 legacy code 還差。
8. 再難懂的程式片段（code readability 很差）理論上你都可以藉由 step/next 以及變數變化的蛛絲馬跡，了解程式最後到底在做什麼。
9. 假設你是要完整的了解程式完整的核心運作，請自行構思一個一邊追蹤一邊紀錄下來的方式。例如畫下來某一種你所能理解的架構圖，流程圖等等。這可以避免不必要的 re-discovery。你如果不做這些輔助的事項，你會發現你的 program understanding 過程不斷地在做 rediscovery 。
10. 你如果真的懂了這個 lab 你就會了解 code readability, variable naming， 等等有多重要。未來請不要寫出折磨別人的程式碼。請不寫出寫死，讓別人無法修改。
11. 撰寫程式請保持簡潔有力，大部分 KISS (Keep It Simple and Stupid)，無須神秘與利用高深來誇耀自己的 coding 能力。