MultiTalkServer

同作業七的練習，Server的建立沒有什麼大問題。使用老師所教授的Boolean Listening = true在While(Listening)中等待Socket Accept就完成大部份的問題。

CreateAccount

原本理想是希望將所有帳號的資訊都存在同一個文件檔中，但在後來經過更多資料的寫入，發現如此的結構會使得存取的方式不敷使用。因此最終改成以資料夾名稱作為使用者區分，在其名下建立該用戶的各項訊息，如個人資料、朋友列表、歷史訊息等。

運作時會先調用SearchMethods裡的IDExitst方法，確認此創立ID是否已經存在，確定唯一後再用FileWriter新建用戶資料夾、檔案，並在其上寫入從Client端獲得之用戶資訊。

LogMethods

起初的想法同樣是用一個文字檔來記錄使用者的上下線，但實作發現如此行不具備效率且繁複。最後使用了TreeMap來記錄活躍中用戶的ID以及其連接Socket訊息(以Key-Value建置)。

運作時直接使用ContainsKey確定用戶是否已經登錄，若否則將其資訊寫入Map中記錄；登出時同樣確認Map中的記錄，成功後將其自Map中Remove

SearchMethods

和當初的構思相同，利用正則表達式，篩選、尋找所需要的資訊，回傳或者確認。

運作時會先將Client端提出的要求寫入正則表達式，再進入先前建立的文檔中找尋，依照不同指令回傳結果(是否找著，或者回傳資訊)。

EditAccount

類似於CreateAccount，先找到目標用戶資料夾內的文檔info.txt，對其進行更改；若是帳號不存在（意即沒有文檔），將產生Exception，傳回失敗的告示。

FriendMethods

和CreateAccount一樣，原本打算同使用者資訊一起存取朋友ID的ArrayList。但經過實際使用發現讀寫不便且費工，解決辦法為在用戶資料夾下另外建立FriendList以及FriendRequest兩個文檔來存儲好友名單跟好友請求。

原本在處理文檔讀寫時是只使用FileWriter直接存取完成的ArrayList，但是發現如此行會存入多餘的方括號[ ]，最後使用簡單的新型迴圈for( Object name : List)便輕鬆解決此問題。

MessageMethods

在資料夾的結構下，針對用戶依照對象分別建獨立文字檔存取其對話。

SendMessage方法中，Client端依序傳入發話人、接話者ID及對話內容，再分別存入兩資料夾下的文檔。

為解決存取次序的問題，避免多個對話同時對同一文檔進行操作，我們選擇使用static synchronized FileAccessor 這個函式來調用檔案，如此就能依序完成各對話指令。

ReturnHistory則是雲聊天記錄的操作，當Client提出請求，便從該用戶調取目標對象的對話記錄，逐行傳回Client端。

SReceiveData

設計就是服務器接收客戶端傳來的訊息。最初規劃沒有考慮到，但實現時發覺有必要做協議方便客戶端與伺服器溝通需要什麼資訊和操作。