CS 3340 - Program 4: Threading (25 points)
Due Wednesday, March 11 2015, at 5:00 PM

The program consists of two projects: UserClass and Prog4GUI.

Project UserClass

1. This is a Class Library project. The assembly name **must** be UserClass, and the root namespace **must** be UWPCS3340.
2. The project has one class UserAccount.
   - The class represents users sharing one common bank account. After started, a user goes to work and issues transactions (in state Working). A user may be suspended for some reason (in state Waiting). If suspended, a user should complete the current work and transaction and then wait. A waiting user may be resumed or terminated later. A user can be terminated at any time, but a working user should finish the current work and transaction before being terminated.
   - The class defines a public enumeration type UserState and two public delegates.

```
Public Enum UserState
   Working
   Waiting
   Terminated
End Enum

Public Delegate Sub TransactionDelegate(ByVal ID As String, ByVal Amount As Integer,
                                        ByVal Final As Boolean)
Public Delegate Sub PassMessageDelegate(ByVal ID As String, ByVal State As UserState)
```

   - Private members

```
' There should be a thread to do the work for each user.
Private _thread As System.Threading.Thread

' All users are stored in a list in the order they were created.
' A user should be removed from the list after terminated.
Private Shared AllUsers As New List(Of UserAccount)

' A user is in one of three states after started.
' But cannot be re-started after being terminated.
Private _state As UserState

' Used to generate the working times and transaction amounts.
Private _randomGenerator As System.Random

' To manage user wait state.
Private _userWait As New ManualResetEvent(False)

' To manage terminating all users.
Private Shared _allTerminated As New ManualResetEvent(False)
```

   - The public properties

```
' To invoke delegates on a form.
Public WriteOnly Property MainForm As System.Windows.Forms.Form

' TransactionDelegate is a delegate.
Public WriteOnly Property TheTransaction As TransactionDelegate

' PassMessageDelegate is a delegate.
Public WriteOnly Property TheReport As PassMessageDelegate
```

   - The public methods

```
' To start a user thread.
Public Sub SpinUp()

' To terminate the user.
Public Sub SpinDown()

' To suspend the user.
' Should have no effect if the user is in Waiting state.
Public Sub UserWait()

' To wake up the user.
' Should have no effect if the user is in Working state.
Public Sub UserContinue()

' To terminate all user threads.
Public Shared Sub TerminateAllUsers()

' To get the UserAccount object at given index of list AllUsers.
Public Shared Function GetUserByIndex(ByVal index As Integer) As UserAccount
```

   - You can modify the default class constructor.
   - You **must not** use methods Abort, Interrupt, Suspend or Resume of Thread. You will **lose 3 points** for each such method used in your code.
   - The class should have a private Sub Run for the user thread to execute. The pseudocode of the Sub is given below:

```
Add the user object to AllUsers
Invoke delegate TheReport to report user state

While not terminated
   Generate a work time
   Go to work for the generated time period
   Generate a transaction amount
   Invoke delegate TheTransaction to issue a transaction of the amount

   If user state is Waiting
      Invoke delegate TheReport to report user state
      Wait until being resumed or terminated

      If user state is working
         Invoke delegate TheReport to report user state
End While

Invoke delegate TheReport to report user state
Invoke delegate TheTransaction to issue final total transaction
Remove the user object from AllUsers
Wake up the thread waiting on _allTerminated if the list is empty after the user was removed from AllUsers
```

- For each iteration of the while loop, the user will work (go to sleep) for a while, then generate a transaction amount and issue the transaction. You must use a random generator (class Random) to decide the working time and the transaction amount. The working time is between 3 and 5 seconds, and the transaction amount is between -100 and 100.
- Within the while loop, each user should invoke delegate TheTransaction with a value of False for parameter Final.
- Each user should keep its own total amount of transactions and, after exiting the while loop, invoke delegate TheTransaction with a value of True for parameter Final before going out of Sub Run and terminating.
- The ID of each user is your initial followed by the hash code of the thread object, e.g., QY12.
- You need to figure out other private members of the class.
- The class cannot have other public members.

Project Prog4GUI

1. This is a Windows Forms application project and the startup project of the solution.
2. The program has two classes: class FormClassThread and class Prog4.
3. The project startup object must be Sub Main or class Prog4.
4. Class Prog4 should not have any variables, even inside the Sub Main.
5. FormClassThread
   - Your form should be similar to the form of the sample program.
   - The form class keeps an integer variable for the balance of the common account, whose intial value is 1000.
   - The form class also keeps the total transactions by all users.
   - The form has the following controls:
     - TextBox txtBalance: To display the balance of the common account.
     - TextBox txtTransaction: To display the total transactions by all users.
     - TextBox txtLog: To record the start and termination (with total transaction amount) of each user, also to keep trak of all individual transactions.
     - ListBox lstAllUsers: To display all non-terminated users with their current state.
     - Command button btnCreate: To start a new user.
     - Command button btnWait: To suspend the selected user in lstAllUsers. Does not do any thing if no user is selected.
     - Command button btnContinue: To resume the selected user in lstAllUsers. Does not do any thing if no user is selected.
     - Command button btnTerminate: To terminate the selected user in lstAllUsers. Does not do any thing if no user is selected.
     - Command button btnExit: To terminate the program. It will do the following
       1. wait until all users are terminated gracefully
       2. while waiting, the form is still working, but all buttons disabled
       3. after all users are terminated, a messagebox is displayed to ask if the person running the program wants to exit
       4. if the answer if yes, the program will be terminated
       5. otherwise, the program continues to run with all buttons enabled and new users can be created to do transactions

     (Hint: Use another thread to wait for all users to terminate and then invoke a private delegate to enable the buttons.)
   - The class must define two subs as the delegates for the objects of class UserAccount.
     - Sub for delegate TheTransaction with three parameters: ID, Amount and Final.
       If Final is false, then it's an individual transaction, Amount is displayed with ID in txtLog, and the balance should be modified by Amount and re-displayed in txtBalance.
       Otherwise, it's the sum of all transactions by a user, then the sum should be displayed with ID in txtLog, and the total transaction should be updated and re-displayed in txtTransaction.
     - Sub for delegate TheReport with two parameters: ID and State
       If ID is not in lstAllUsers, then it's a new user and needs to be added to lstAllUsers and txtLog.
       Otherwise, it's an existing user and needs to be removed or updated.
   - The form class will only use the two delegate subs to update the display on the form. The click event procedures of the buttons should not update the display on the form.

Submission

1. Name your solution folder as UserName_Prog4 using your UWP username.
2. Drop your solution folder to folder Prog4 of the class DROP folder by the due time.

3. You may lose up to two points for incorrect submission.
4. You must follow the programming rules, and you may lose up to five points on style.