Programming Assignment #5

Due Nov 5 by 3pm **Points** 100 **Submitting** a file upload
Available Oct 22 at 8pm - Dec 5 at 12pm about 1 month

This assignment was locked Dec 5 at 12pm.

Assignment #5:

OK, now that we've seen how to create the "Go Fish" game using arrays of objects, let's update things to use Collections and Exception Handling.

1. Study the version of "Go Fish" in the Modules area and update your version as necessary so that it runs correctly. Make sure you've implemented several different player classes.
2. Create a new, empty Visual Studio project for Assignment 5 and COPY your code from Assignment 4 into that new project. Adjust any namespace statements appropriate and make sure this new project runs correctly.
3. Now, in the Assignment 5 project, change all of the low-level arrays in your program to use the appropriate C# collection class instead. For example, you'll want to implement Deck using a Stack object.
4. Make sure that you use the correct type-safe generic designations throughout your program.
5. Put on your "tester" hat and identify all of the places where incorrect data could be passed to the methods and constructors in your classes, then write code that watches for that incorrect data and ***throws*** an appropriate application exception if bad data is found. (Note: Your program doesn't need to catch those exceptions because they are application exception.)
6. Manually run your program several times with different player combinations and make sure it works correctly by carefully inspecting the output.

Extra Credit: 10 points

Let's see once and for all, which player strategy is best...

1. Put the entire game in a big loop so that you can run it many times in a row.
2. Add a global boolean variable that allows/prevents the game from generating console output while running.
3. Create a GameResults class that can store key statistics about a game's outcome. Key statistics include: Game #, Number of Turns, Winner's name (or "Tie"), and the winning margin (i.e. the difference between the winner's score and the score of the player that finished in second place).
4. Now, put the game code in a loop and run the game 1000 times (without generating any game output). Each time a game finishes, create a new GameResults object, store the key stats in that object, display that object, and then add it to a Queue.
5. After all the games have completed, iterate through the Queue of GameResults and use a Dictionary to add up each player's total number of wins. Finally, display those totals.

Sample Output for Extra Credit Version:

```
...
Game #: 998
Winner: Susan(M)
Score: 7 (3)
Turns: 41

Game #: 999
Winner: Susan(M)
Score: 8 (5)
Turns: 38

==========================================

After 1000 games...

Susan(M): 767
Paul(Rnd): 109
Tie: 108
Kelly(RS): 13
Tom(LS): 3

Avg Turns per game: 44.037
Avg Winning score: 6.473
Avg Winning margin: 2.462
Max Winning margin: 9
```

Important Notes:

- HINT: There are times where a Deck should be a Stack and times where it needs to be something else. Be sure to look at ALL the methods in the Stack<> class for help with that.
- Be sure to read, understand and follow **[0070] Our C# Coding Standards**
- Do not use any C# language features that we have not discussed in class. (no LINQ, no lambdas, etc.)
- Always choose descriptive variable, parameter, and method names. Always strive to make your code self-documenting.
- When you are finished, please ZIP up the Visual Studio Solution directory for your program and upload it using the link on this page.

My Grading Guidelines:

30% Does the program compile and run?
30% Does the program generate correct results?
10% Does the program exhibit good OO design properties?
10% Is the program readable / maintainable / flexible ?
10% Are the encapsulation classes bulletproof? Are they type-safe?
5% Does the program follow our C# style guide rules
5% Are several different Player sub-classes implemented correctly?