Programming Assignment #4

Due Oct 22 by 3pm **Points** 100 **Submitting** a file upload
Available Oct 14 at 8pm - Dec 5 at 12pm about 2 months

This assignment was locked Dec 5 at 12pm.

Assignment #4: Go Fish!

Let's use our deck of cards to simulate a game of Go Fish between several computer players!

If you are unfamiliar with this game, I recommend you play the game several times with friends or online. Here is one site that lets you experience the game online: **https://cardgames.io/gofish/ (https://cardgames.io/gofish/)**

If you are not familiar with Go Fish, here are the rules:

- At the start of each game, the deck is shuffled and cut. Then 5 cards are dealt out to each of the four players. Each player keeps their cards in their hand and can look at them.
- If any player has a "book" (also called a "trick") in their hand - i.e., 4 cards of the same rank, they "play" that book by placing those 4 cards in front of them, face-up so other players can see them. Each book played is worth one point. The object of the game is to get more points than any of your opponents.
- If any player plays all of the cards in their hand as a book, they then immediately draw 4 more cards from the deck.
- From this point forward, players take turns moving in a clockwise direction. When it is a player's turn, they choose a rank from among the ranks currently in their hand and then select another player that they think might have cards of that same rank. They then ask that player by name if they have any cards of that rank. For example, "Jeffery, give me all your Eights!"
- If the-player-who-was-asked has any cards of that rank, they must hand them over to the current player. The current player then immediately gets another turn.
- If the current player completes a book after receiving cards from another player, the current player must immediately play that book adding another point to their total.
- If the-player-who-was-asked does not have any cards of that rank, they yell "GO FISH!" The current player then draws a card from the deck and their turn is over.
- Players continue taking turns until all books are played. If a player runs out of cards and the deck is empty, that player no longer gets a turn.
- After all 13 books have been played, the player with the most points wins.

Assignment Details:

1.) Use your Card, Deck and Hand classes from Assignment #3 in your program. YOU CAN EITHER REUSE YOUR CLASSES OR YOU CAN USE THE PLAYING CARD CLASSES FROM **PROGRAMMING ASSIGNMENT #3 - REVIEW** (but only if you fully understand those classes!)

2.) Just like our Connect 4 game, your Main() method will be where the game simulation takes place. It will need to have a loop that continues until the game is over. Each time through the loop represents another computer player's turn.

3.) Each time a player does anything, a message needs to be printed out on the console describing what the player did. After the program finishes, the console should have a complete record of everything that happened during the game. It should read like a written history of the game.

4.) Different computer players will be using different game strategies during the simulation. That means we need to use polymorphism to implement several different types of players. Start by adding the following abstract Player class in your program:

```
public abstract class Player
{
   public string Name { get; private set; }
   public Hand Hand { get; private set; }
   // Other Properties/member variables go here

   public Player(string name)
   {
      this.Name = name;
      this.Hand = new Hand();
   }

   public abstract Player ChoosePlayerToAsk(Player[] players);
   public abstract Rank ChooseRankToAskFor();

   // Other Player methods go here

   public override string ToString()
   {
       return $"{Name}'s Hand: {Hand}";
   }
}
```

You'll need to ***add*** more methods and member variables into the Player class, but you should not ***modify*** any of the code above.

5.) First, create a subclass of the Player class that has players choose both items (player and suit) randomly but within the rules of the game (i.e., they still only ask for suits that they actually have in their hand).

6.) Next, get this first version of the game working using four of your "random strategy" players.

IF THIS IS AS FAR AS YOU GET BEFORE OUR NEXT CLASS, THAT IS FINE. PLEASE TRY TO GET AT LEAST THIS FAR.

7.) Now, in addition to the "random strategy" player, create three additional types (i.e., subclasses) of players that use the following different strategies:

- A player that always chooses the first card in their hand and the first player on their right.
- A player that always chooses the last card in their hand and the first player on their left.
- A player that always chooses the last card in their hand but asks a random player.

8.) Run your program lots of times and see which strategy works best. Your output should look similar to this:
SAMPLE OUTPUT FOR PROGRAMMING ASSIGNMENT 4

EXTRA CREDIT - up to 5 points

9.) Create a class for a player that cheats by asking for suits that they don't actually have. Replace one of the other four players with a player that uses this strategy and rerun the program to see what happens.

10.) Create a class for players that remember the last rank that each of the other players asked for and uses that information when playing the game. Replace one of the other four players with a player that uses this strategy and rerun the program to see what happens.

Important Notes:

- Do not use any C# language features that have not been covered in the reading or discussed in class. SORRY, BUT WE STILL CANNOT USE COLLECTIONS YET! :-(Please use regular C# arrays for now.
- Always choose descriptive variable, parameter, and method names. Always strive to make your code self-documenting.
- Be sure to add the appropriate exceptions and exception handlers to all of your classes.
- 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 Guideline:

40% Does the program compile and run without throwing an exception or getting into an infinite loop?
40% Does the program generate correct output? Are all games clearly documented in the output? Do all the computer players follow the rules of the game?
5% Are your classes and subclasses well-thought-out? Do they reflect noun-verb analysis of the game's description?
5% Are your classes bulletproof? Do their methods and constructors take appropriate action when given bad parameters?
5% Are all of the different player subclasses implemented correctly?
5% Is your code readable, correctly styled, maintainable, and extendable?