PROGRAMMING ASSIGNMENT #5 - REVIEW

CardClasses.cs:

```
using System;
using System.Collections.Generic;

namespace LWTech.CSD228.AndersonChip.CardClasses
{
    public enum Suit { Clubs, Diamonds, Hearts, Spades };
    public enum Rank { Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King };

    // -----------------------------------------------------------------------------------------------------

    public class Card
    {
        public Rank Rank { get; private set; }
        public Suit Suit { get; private set; }

        public Card(Suit suit, Rank rank)
        {
            this.Suit = suit;
            this.Rank = rank;
        }

        public override string ToString()
        {
            return ("[" + Rank + " of " + Suit + "]");
        }

    }

    // -----------------------------------------------------------------------------------------------------

    public class Deck
    {
        private static Random rng = new Random();       // static helps prevent duplicate rng's

        private Stack<Card> cards;

        public Deck()
        {
            cards = new Stack<Card>();

            Array suits = Enum.GetValues(typeof(Suit));
            Array ranks = Enum.GetValues(typeof(Rank));

            foreach (Suit suit in suits)
            {
                foreach (Rank rank in ranks)
                {
                    Card card = new Card(suit, rank);
                    cards.Push(card);
                }
            }
        }

        public int Size()
        {
            return cards.Count;
        }

        public void Shuffle()
        {
            if (Size() == 0) return;                    // Cannot shuffle an empty deck

            Card[] cardArray = this.cards.ToArray();

            // Fisher-Yates Shuffle (modern algorithm)
            //   - http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle
            for (int i = 0; i < Size(); i++)
            {
                int j = rng.Next(i, Size());

                Card c = cardArray[i];
                cardArray[i] = cardArray[j];
                cardArray[j] = c;
            }

            cards = new Stack<Card>(cardArray);

        }

        public void Cut()
        {
            if (Size() < 2) return;                    // Cannot cut a deck with less than 2 cards

            var stackTop = new Stack<Card>();
            var stackBottom = new Stack<Card>();

            int cutPoint = 0;
            do
            {
                cutPoint = rng.Next(cards.Count);
            }
            while (cutPoint < 2);

            // Split the deck into two stacks
            for (int i = 0; i < cutPoint; i++)
                stackTop.Push(cards.Pop());

            while (cards.Count > 0)
                stackBottom.Push(cards.Pop());

            // Join the deck back together
            while (stackTop.Count > 0)
                cards.Push(stackTop.Pop());

            while (stackBottom.Count > 0)
                cards.Push(stackBottom.Pop());
        }

        public Card DealCard()
        {
            if (Size() == 0) return null;
            return cards.Pop();
        }

        public void ReturnCardToDeck(Card c)
        {
            if (c == null)
                throw new Exception("Card cannot be null.");
            cards.Push(c);
        }

        public override string ToString()
        {
            string s = "[";
            string comma = "";
            foreach (Card c in cards)
            {
                s += comma + c.ToString();
                comma = ", ";
            }
            s += "]";
            s += "\n " + Size() + " cards in deck.\n";

            return s;
        }

    }

    // -----------------------------------------------------------------------------------------------------

    public class Hand
    {
        private List<Card> cards;

        public Hand()
        {
            cards = new List<Card>();
        }

        public int Size()
        {
            return cards.Count;
        }

        public List<Card> GetCards()
        {
            return new List<Card>(cards);               // Returns a copy of our hand
        }

        public void AddCard(Card card)
        {
            if (card == null)
                throw new Exception("Card cannot be null.");
            cards.Add(card);
        }

        public Card RemoveCard(Card card)
        {
            if (card == null)
                return null;

            if (cards.Remove(card))
                return card;
            return null;
        }

        public override string ToString()
        {
            string s = "[";
            string comma = "";
            foreach (Card c in cards)
            {
                s += comma + c.ToString();
                comma = ", ";
            }
            s += "]";

            return s;
        }

    }

}
```

PlayerClasses.cs:

```
using System;
using System.Collections.Generic;
using LWTech.CSD228.AndersonChip.CardClasses;

namespace LWTech.CSD228.AndersonChip.GoFishSimulator
{
    public abstract class Player
    {
        public string Name { get; private set; }
        public Hand Hand { get; private set; }
        public int Points { get; private set; }
        public Rank LastRankAsked { get; protected set; }

        public Player(string name)
        {
            if (name == null)
```

```
                throw new ArgumentNullException(nameof(name));
            if (name == "")
                throw new ArgumentException("Player name cannot be an empty string.");

            this.Name = name;
            this.Hand = new Hand();
        }

        public abstract Player ChoosePlayerToAsk(List<Player> players);

        public abstract Rank ChooseRankToAskFor();

        // Adds a card to the player's hand
        public void AddCardToHand(Card card)
        {
            if (card == null)
                throw new ArgumentNullException(nameof(card));

            this.Hand.AddCard(card);
        }

        // Returns a card of the given rank (if found) or null (if not found)
        public Card GiveAnyCardOfRank(Rank rank)
        {
            foreach (Card c in Hand.GetCards())
            {
                if (c.Rank == rank)
                {
                    this.Hand.RemoveCard(c);
                    return c;       // immediately return the first card found with rank
                }
            }
            return null;            // rank was not found in player's hand
        }

        // Returns true if player has any cards of the given rank (otherwise false)
        public bool HasRankInHand(Rank rank)
        {
            foreach (Card c in this.Hand.GetCards())
            {
                if (c.Rank == rank)
                    return true;    // immediately return true when rank is found in player's hand
            }
            return false;           // rank was not found in player's hand
        }

        // Returns the rank of the first book found.  Returns null if no books are found.
        public Rank? HasBookInHand()
        {
            foreach (Rank rank in Enum.GetValues(typeof(Rank)))         // search all ranks everytime
            {
                int numSuits = 0;
                foreach (Card c in this.Hand.GetCards())
                {
                    if (c.Rank == rank)
                        numSuits++;
                }

                if (numSuits == Enum.GetValues(typeof(Suit)).Length)
                    return rank;                                        // as soon as a book is found, return
its rank
            }
            return null;            // player does not have any books
        }

        // Removes a book of the give rank from the player's hand and increments the player's score
        public void PlayBook(Rank rank)
        {
            int i = 0;
            foreach (Card c in this.Hand.GetCards())
            {
                if (c.Rank == rank)
                {
                    this.Hand.RemoveCard(c);             // Removed cards are discarded
                    i++;
                }
            }
            Points++;
        }

        public override string ToString()
        {
            string s = Name + "'s Hand: ";
            s += this.Hand.ToString();

            return s;
        }
    }

    //----------------------------------------------------------------------------

    // Randomly selects player-to-ask and rank-to-ask-for
    public class RandomPlayer : Player
    {
        private static Random rng = new Random();

        public RandomPlayer(string name) : base(name + "(Rnd)")
        { }

        public override Player ChoosePlayerToAsk(List<Player> players)
        {
            if (players == null)
                throw new ArgumentNullException(nameof(players));

            Player candidate = this;
            while ((candidate == this) || (candidate.Hand.Size() == 0))
                candidate = players[rng.Next(players.Count)];

            return candidate;
        }

        public override Rank ChooseRankToAskFor()
        {
            List<Card> cards = Hand.GetCards();
            int randomIndex = rng.Next(cards.Count);

            return cards[randomIndex].Rank;
        }
    }

    //----------------------------------------------------------------------------

    // Always selects first player and asks for rank of first card in their hand
    public class LeftSidePlayer : Player
    {
        public LeftSidePlayer(string name) : base(name + "(LS)")
        { }

        public override Player ChoosePlayerToAsk(List<Player> players)
        {
            if (players == null)
                throw new ArgumentNullException(nameof(players));

            Player player = this;
            int i = 0;
            while ((player == this) || (player.Hand.Size() == 0))
                player = players[i++];

            return player;
        }

        public override Rank ChooseRankToAskFor()
        {
            Rank rank;
            List<Card> cards = Hand.GetCards();
            rank = cards[0].Rank;

            return rank;
        }
    }

    //----------------------------------------------------------------------------

    // Always selects last player and asks for rank of last card in their hand
    public class RightSidePlayer : Player
    {
        public RightSidePlayer(string name) : base(name + "(RS)")
        { }

        public override Player ChoosePlayerToAsk(List<Player> players)
        {
            if (players == null)
                throw new ArgumentNullException(nameof(players));

            Player player = this;
            int i = players.Count;
            while ((player == this) || (player.Hand.Size() == 0))
                player = players[--i];
            return player;
        }

        public override Rank ChooseRankToAskFor()
        {
            Rank rank;
            List<Card> cards = Hand.GetCards();
            rank = cards[cards.Count - 1].Rank;

            return rank;
        }
    }

    //----------------------------------------------------------------------------

    // Always chooses the rank that the last player asked about
    public class MemoryPlayer : Player
    {
        private static Random rng = new Random();
        private Player playerToAsk;
        private Rank rankToAsk;

        public MemoryPlayer(string name) : base(name + "(M)")
        {
            playerToAsk = this;
            rankToAsk = Rank.Ace;
        }

        public override Player ChoosePlayerToAsk(List<Player> players)
        {
            if (players == null)
                throw new ArgumentNullException(nameof(players));

            Player candidate = null;
            int tries = 0;
            int i = rng.Next(players.Count);
            bool foundPlayer = false;
            while (!foundPlayer)
            {
                tries++;
                candidate = players[i++ % players.Count];
                if (candidate == this || candidate.Hand.Size() == 0)
                    continue;

                if (candidate.HasRankInHand(LastRankAsked))
                    foundPlayer = true;                 // Found player who recently asked for a card we hav
e!

                if (tries > players.Count)
                {
                    foundPlayer = true;                 // Giving up.  Ask player for a random rank.
                    List<Card> cards = Hand.GetCards();
                    rankToAsk = cards[rng.Next(Hand.Size())].Rank;
                }
            }
            playerToAsk = candidate;
            return playerToAsk;
        }

        public override Rank ChooseRankToAskFor()
        {
            LastRankAsked = rankToAsk;
            return rankToAsk;
        }

    }

    //---------------------------------------------------------------------------

    // Cheater that asks for any rank even if they don't have it
    public class CheatingPlayer : Player
    {
        private static Random rng = new Random();

        public CheatingPlayer(string name) : base(name + "(CHEET)")
        { }

        public override Player ChoosePlayerToAsk(List<Player> players)
        {
            if (players == null)
                throw new ArgumentNullException(nameof(players));

            Player candidate = this;
            while ((candidate == this) || (candidate.Hand.Size() == 0))
                candidate = players[rng.Next(players.Count)];

            return candidate;
        }

        public override Rank ChooseRankToAskFor()
        {
            Rank[] ranks = (Rank[])Enum.GetValues(typeof(Rank));
            int randomIndex = rng.Next(ranks.Length);

            return ranks[randomIndex];
        }
    }

}
```

GameResults.cs:

```
using System;

namespace LWTech.CSD228.AndersonChip.GoFishSimulator.Collections
{
    public class GameResults
    {
        public int NumGame { get; private set; }
        public string Winner { get; private set; }
        public int WinScore { get; private set; }
        public int WinMargin { get; private set; }
        public int NumTurns { get; private set; }

        public GameResults(int numGame, string winner, int winScore, int winMargin, int numTurns)
        {
            this.NumGame = numGame;
            this.Winner = winner;
            this.WinScore = winScore;
            this.WinMargin = winMargin;
            this.NumTurns = numTurns;
        }

        public override string ToString()
        {
            string s = "Game #: " + NumGame;
            s += "\nWinner: " + Winner;
            s += "\nScore: " + WinScore + " (" + WinMargin + ")";
            s += "\nTurns: " + NumTurns;
            return s;
        }

    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using LWTech.CSD228.AndersonChip.CardClasses;

namespace LWTech.CSD228.AndersonChip.GoFishSimulator.Collections
{
    class Program
    {
        private static Deck theDeck;
        private static List<Player> players;

        private static int numTurns = 0;
        private static int numBooksPlayed = 0;
        private static bool outputEnabled = true;

        public static void Main()
        {
            WriteLine("Go Fish Simulation (w/Collections)");
            WriteLine("=====================================");

            Queue<GameResults> resultsQueue = new Queue<GameResults>();

            int numGames;
            for (numGames = 0; numGames < 1000; numGames++)
            {
                outputEnabled = true;
                WriteLine("============================================ Staring game #" + numGames + "! =====
=======================================");
```

```
                outputEnabled = false;

                numTurns = 0;
                numBooksPlayed = 0;

                theDeck = new Deck();
                theDeck.Shuffle();
                theDeck.Cut();

                players = new List<Player>();
                players.Add(new RightSidePlayer("Paul"));
                players.Add(new RandomPlayer("Tom"));
                players.Add(new LeftSidePlayer("Pat"));
                players.Add(new MemoryPlayer("Susan"));

                for (int i = 0; i < 5; i++)
                {
                    foreach (Player player in players)
                        player.AddCardToHand(theDeck.DealCard());
                }

                foreach (Player player in players)
                    WriteLine(player.ToString());

                int currentPlayerIndex = 0;
                WriteLine("It is now " + players[currentPlayerIndex].Name + "'s turn.");

                while (true)
                {
                    Player currentPlayer = players[currentPlayerIndex];

                    Player playerToAsk = currentPlayer.ChoosePlayerToAsk(players);
                    Rank rankToAskFor = currentPlayer.ChooseRankToAskFor();

                    WriteLine(currentPlayer.Name + " says: " + playerToAsk.Name + "! Give me all of your " +
rankToAskFor + "s!");

                    Card card = playerToAsk.GiveAnyCardOfRank(rankToAskFor);
                    if (card == null)
                    {
                        // playerToAsk doesn't have any cards of that rank.

                        WriteLine(playerToAsk.Name + " says: GO FISH!");

                        if (theDeck.Size() > 0)
                        {
                            card = theDeck.DealCard();
                            WriteLine(currentPlayer.Name + " draws a " + card + " from the deck.  The deck no
w has " + theDeck.Size() + " cards remaining.");
                            currentPlayer.AddCardToHand(card);
                            PlayAnyBooks(currentPlayer);
                            if (IsGameOver()) break;
                            Draw5CardsIfHandIsEmpty(currentPlayer);
                        }
                        else
                        {
                            WriteLine("Deck is empty. " + currentPlayer.Name + " cannot draw a card.");
                        }

                        WriteLine(currentPlayer.Name + "'s turn is over. " + currentPlayer.Hand);
                        currentPlayerIndex = NextValidPlayer(currentPlayerIndex);
                        WriteLine("\nIt is now " + players[currentPlayerIndex].Name + "'s turn.");
                        numTurns++;
                        DisplayScoreboard();
                    }
                    else
                    {
                        // playerToAsk does have one (or more) cards of that rank. Take all of them.
                        do
                        {
                            WriteLine(currentPlayer.Name + " gets the " + card + " from " + playerToAsk.Nam
e);
                            currentPlayer.AddCardToHand(card);
                            card = playerToAsk.GiveAnyCardOfRank(rankToAskFor);
                        } while (card != null);

                        Draw5CardsIfHandIsEmpty(playerToAsk);

                        PlayAnyBooks(currentPlayer);
                        if (IsGameOver()) break;
                        Draw5CardsIfHandIsEmpty(currentPlayer);

                        if (currentPlayer.Hand.Size() > 0)
                        {
                            WriteLine("It is still " + currentPlayer.Name + "'s turn.");
                        }
                        else
                        {
                            WriteLine(currentPlayer.Name + "'s hand is empty. " + currentPlayer.Name + " is f
inished.");
                            currentPlayerIndex = NextValidPlayer(currentPlayerIndex);
                            WriteLine("\nIt is now " + players[currentPlayerIndex].Name + "'s turn.");
                            numTurns++;
                            DisplayScoreboard();
                        }
                    }

                }

                WriteLine("\n============== Game Over! =================\n");
                DisplayScoreboard();

                bool tieGame = false;
                int margin = 0;
                Player winner = players[0];
                for (int i = 1; i < players.Count; i++)
                {
                    if (players[i].Points > winner.Points)
                    {
                        tieGame = false;
                        margin = players[i].Points - winner.Points;
                        winner = players[i];
                    }
                    else if (players[i].Points == winner.Points)
                    {
                        tieGame = true;
                        margin = 0;
                    }
                }

                string winningName;

                // Display the game results
                WriteLine("\nAfter " + numTurns + " turns,");
                if (tieGame)
                {
                    WriteLine("It's a tie!");
                    winningName = "Tie Game";
                }
                else
                {
                    WriteLine("The winner is " + winner.Name + " with " + winner.Points + " points!");
                    winningName = winner.Name;
                }

                // Store the game's results in a new Results object and add it to the queue
                GameResults result = new GameResults(numGames, winningName, winner.Points, margin, numTurns);
                resultsQueue.Enqueue(result);
            }

            outputEnabled = true;
            DisplayGameStats(resultsQueue);

        }

        // ===================================================================

        private static void WriteLine(string s = "")
        {
            if (outputEnabled)
                Console.WriteLine(s);
        }

        private static void DisplayScoreboard()
        {
            string s = "SCORE: ";
            foreach (Player player in players)
                s += " | " + player.Name + ": " + player.Points;
            s += " |  [Deck: " + theDeck.Size() + "]";
            WriteLine(s);
        }

        private static int NextValidPlayer(int currentPlayerIndex)
        {
            int nextPlayerIndex = currentPlayerIndex;
            do
            {
                nextPlayerIndex = (nextPlayerIndex + 1) % players.Count;
            } while (players[nextPlayerIndex].Hand.Size() == 0);

            return nextPlayerIndex;
        }

        private static void PlayAnyBooks(Player player)
        {
            Rank? rank = player.HasBookInHand();
            while (rank != null)
            {
                WriteLine(">>> " + player.Name.ToUpper() + " HAS A BOOK! PLAYING A BOOK OF " + rank.ToString
().ToUpper() + "S!");
                player.PlayBook((Rank)rank);
                numBooksPlayed++;
                rank = player.HasBookInHand();
            }
        }

        private static bool IsGameOver()
        {
            return (numBooksPlayed == Enum.GetValues(typeof(Rank)).Length);
        }

        private static void Draw5CardsIfHandIsEmpty(Player player)
        {
            if (player.Hand.Size() > 0) return;
            if (theDeck.Size() == 0) return;

            WriteLine(">>>> " + player.Name + "'s hand is empty.  Drawing up to 5 cards from the deck. <<<
<");
            for (int i = 0; i < 5; i++)
            {
                Card card = theDeck.DealCard();
                if (card == null)
                    break;
                player.Hand.AddCard(card);
                PlayAnyBooks(player);
            }
        }

        private static void DisplayGameStats(Queue<GameResults> results)
        {
            int numGames = 0;
            int totalTurns = 0;
            int totalScore = 0;
            int totalMargin = 0;
            int maxMargin = -1;

            var playerWinTotals = new Dictionary<string, int>();
            foreach (Player player in players)
                playerWinTotals.Add(player.Name, 0);
            playerWinTotals.Add("Tie Game", 0);

            foreach (GameResults result in results)
            {
                WriteLine(result.ToString());
                numGames++;

                totalTurns += result.NumTurns;
                totalScore += result.WinScore;
                totalMargin += result.WinMargin;
                maxMargin = Math.Max(maxMargin, result.WinMargin);

                playerWinTotals[result.Winner]++;
            }

            WriteLine("After " + numGames + " games...\n");

            WriteLine("Avg Turns per game:\t" + (double)totalTurns / numGames);
            WriteLine("Avg Winning score:\t" + (double)totalScore / numGames);
            WriteLine("Avg Winning margin:\t" + (double)totalMargin / numGames);
            WriteLine("Max Winning margin:\t" + maxMargin);

            WriteLine();

            foreach (String playerName in playerWinTotals.Keys)
                WriteLine($"{playerName}: {playerWinTotals[playerName]}");

        }

    }
}
```