PROGRAMMING ASSIGNMENT #4 - REVIEW

CardClasses.cs

```
using System;

namespace LWTech.ChipAnderson.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 Card[] cards;
        private static Random rng = new Random();        // static helps prevent duplicate rng's
```

```
public Deck()
{
    Array suits = Enum.GetValues(typeof(Suit));
    Array ranks = Enum.GetValues(typeof(Rank));

    int size = suits.Length * ranks.Length;
    cards = new Card[size];

    int i = 0;
    foreach (Suit suit in suits)
    {
        foreach (Rank rank in ranks)
        {
            Card card = new Card(suit, rank);
            cards[i++] = card;
        }
    }
}

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

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

    // 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 = cards[i];
        cards[i] = cards[j];
        cards[j] = c;

    }

}
```

```
public void Cut()
{
    if (Size() == 0) return;                    // Cannot cut an empty deck

    int cutPoint = rng.Next(1, Size());         // Cannot cut at zero

    Card[] newDeck = new Card[Size()];

    int i;
    int j = 0;
    // Copy the cards at or below the cutpoint into the top of the new deck
    for (i = cutPoint; i < Size(); i++)
    {
        newDeck[j++] = cards[i];
    }
    // Copy the cards above the cutpoint into the bottom of the new deck
    for (i = 0; i < cutPoint; i++)
    {
        newDeck[j++] = cards[i];
    }
    cards = newDeck;
}

public Card DealCard()
{
    if (Size() == 0) return null;

    Card card = cards[Size() - 1];              // Deal from bottom of deck (makes Resizing easier)
    Array.Resize(ref cards, Size() - 1);

    return card;
}

public void ReturnCardToDeck(Card c)
{
    if (c == null) return;

    Array.Resize(ref cards, Size() + 1);
    cards[Size()-1] = c;                        // Adds card to the bottom of the deck
}
```

```
    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 Card[] cards;

    public Hand()
    {
        cards = new Card[0];                    // Empty hand
    }

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

    public Card[] GetCards()
    {
        return cards;
    }

    public void AddCard(Card card)
    {
        Array.Resize(ref cards, Size() + 1);
        cards[Size()-1] = card;
    }

    public Card RemoveCard(Card card)
    {
        bool found = false;
        Card[] newCards = new Card[cards.Length - 1];

        // Copy all the cards - except the one asked for - into a new hand
        int i = 0;
        foreach (Card c in cards)
        {
            if (c == card)
                found = true;
            else
                newCards[i++] = c;
        }

        // Did we find the card we were asked for?
        if (found)
        {
            cards = newCards;
            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 LWTech.ChipAnderson.CardClasses;

namespace LWTech.ChipAnderson.GoFish
{

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

        public abstract Player ChoosePlayerToAsk(Player[] players);

        public abstract Rank ChooseRankToAskFor();


        public void AddCardToHand(Card card)
        {
            Hand.AddCard(card);
        }


        public Card GiveAnyCardOfRank(Rank rank)
        {
            foreach (Card c in Hand.GetCards())
            {
                if (c.Rank == rank)
                {
```

```
            Hand.RemoveCard(c);
            return c;
        }
    }
    return null;
}


public bool HasRankInHand(Rank rank)
{
    foreach (Card c in Hand.GetCards())
    {
        if (c.Rank == rank)
            return true;
    }
    return false;
}


// 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)))
    {
        int numSuits = 0;
        foreach (Card c in Hand.GetCards())
        {
            if (c.Rank == rank)
                numSuits++;
        }

        if (numSuits == Enum.GetValues(typeof(Suit)).Length)
            return rank;
    }
    return null;
}


public void PlayBook(Rank rank)
{
    int i = 0;
    foreach (Card c in Hand.GetCards())
```

```
            {
                if (c.Rank == rank)
                {
                    Hand.RemoveCard(c);             // Removed card is discarded
                    i++;
                }
            }
            Points++;
        }


        public override string ToString()
        {
            string s = Name + "'s Hand: ";
            s += 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(Player[] players)
        {
            Player candidate = this;
            while ((candidate == this) || (candidate.Hand.Size() == 0))
                candidate = players[rng.Next(players.Length)];
            return candidate;
        }

        public override Rank ChooseRankToAskFor()
        {
            Card[] cards = Hand.GetCards();
            int randomIndex = rng.Next(cards.Length);
            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(Player[] 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;
            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(Player[] players)
        {
            Player player = this;
            int i = players.Length;
            while ((player == this) || (player.Hand.Size() == 0))
```

```
            player = players[--i];
        return player;
    }

    public override Rank ChooseRankToAskFor()
    {
        Rank rank;
        Card[] cards = Hand.GetCards();
        rank = cards[cards.Length - 1].Rank;

        return rank;
    }
}

// Selects a random player and asks for rank of last card in their hand
public class RightSideRandomPlayer2 : Player
{
    private static Random rng = new Random();

    public RightSideRandomPlayer2(string name) : base(name + "(RSR)")
    { }

    public override Player ChoosePlayerToAsk(Player[] players)
    {
        Player candidate = this;
        while ((candidate == this) || (candidate.Hand.Size() == 0))
            candidate = players[rng.Next(players.Length)];
        return candidate;
    }

    public override Rank ChooseRankToAskFor()
    {
        Rank rank;
        Card[] cards = Hand.GetCards();
        rank = cards[cards.Length - 1].Rank;

        return rank;
    }
}

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

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

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

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


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

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

        public override Player ChoosePlayerToAsk(Player[] players)
        {
            Player candidate = this;
            while ((candidate == this) || (candidate.Hand.Size() == 0))
                candidate = players[rng.Next(players.Length)];
            return candidate;
        }

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

            return ranks[randomIndex];
        }
    }

}
```

Program.cs

```
using System;
using LWTech.ChipAnderson.CardClasses;

namespace LWTech.ChipAnderson.GoFish
{
    class Program
    {
        private static Deck theDeck;
        private static Player[] players;
        private static int numTurns = 0;
        private static int numBooksPlayed = 0;

        public static void Main()
        {
            Console.WriteLine("Go Fish Simulation (w/Random players)");
            Console.WriteLine("=========================================");

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

            players = new Player[4];
            players[0] = new RightSidePlayer("Paul");
            players[1] = new CheatingPlayer("Tom");
            players[2] = new LeftSidePlayer("Pat");
            players[3] = new MemoryPlayer("Susan");

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

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

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

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

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

                Console.WriteLine(currentPlayer.Name + " says: " + playerToAsk.Name + "! Give me all of your " + ran
kToAskFor + "s!");
```

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

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

                if (theDeck.Size() > 0)
                {
                    card = theDeck.DealCard();
                    Console.WriteLine(currentPlayer.Name + " draws a " + card + " from the deck.  The deck now has "
+ theDeck.Size() + " cards remaining.");
                    currentPlayer.AddCardToHand(card);
                    PlayAnyBooks(currentPlayer);
                    if (IsGameOver()) break;
                    Draw5CardsIfHandIsEmpty(currentPlayer);
                }
                else
                {
                    Console.WriteLine("Deck is empty. " + currentPlayer.Name + " cannot draw a card.");
                }

                Console.WriteLine(currentPlayer.Name + "'s turn is over. " + currentPlayer.Hand);
                currentPlayerIndex = NextValidPlayer(currentPlayerIndex);
                Console.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
                {
                    Console.WriteLine(currentPlayer.Name + " gets the " + card + " from " + playerToAsk.Name);
                    currentPlayer.AddCardToHand(card);
                    card = playerToAsk.GiveAnyCardOfRank(rankToAskFor);
                } while (card != null);

                Draw5CardsIfHandIsEmpty(playerToAsk);

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

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

            }

        }

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

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

        }

        Console.WriteLine("\nAfter " + numTurns + " turns,");
        if (tieGame)
```

```
            Console.WriteLine("It's a tie!");
        else
            Console.WriteLine("The winner is " + winner.Name + " with " + winner.Points + " points!");

    }

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

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

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

        return nextPlayerIndex;
    }

    private static void PlayAnyBooks(Player player)
    {
        Rank? rank = player.HasBookInHand();
        while (rank != null)
        {
            Console.WriteLine(">>> " + player.Name.ToUpper() + " HAS A BOOK! PLAYING A BOOK OF " + ran
k.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;

        Console.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);
        }
    }
}
}
```