# PROGRAMMING ASSIGNMENT #3 - 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();

        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
        Card[] newDeck = new Card[Size()];
        int cutPoint = rng.Next(1, Size());         // Cannot cut at zero

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

    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;
        }
    }
}
```

# Program.cs

```
using System;

using LWTech.ChipAnderson.CardClasses;

namespace LWTech.ChipAnderson.Assignment3
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Assignment 3:");
            Console.WriteLine("===================================");
            Console.WriteLine();

            Deck deck = new Deck();

            Console.WriteLine("New deck:");
            Console.WriteLine(deck);

            deck.Shuffle();

            Console.WriteLine("Shuffled deck:");
            Console.WriteLine(deck);

            deck.Cut();

            Console.WriteLine("Cut deck:");
```

```
            Console.WriteLine(deck);

            // Deal 5 cards to 4 players
            Hand[] hands = new Hand[4];
            for (int i = 0; i < 4; i++)
                hands[i] = new Hand();

            for (int i = 0; i < 5; i++)
            {
                foreach (Hand h in hands)
                {
                    Card card = deck.DealCard();
                    h.AddCard(card);
                }
            }

            Console.WriteLine("Dealt hands:");
            foreach (Hand h in hands)
            {
                Console.WriteLine(h);
                Console.WriteLine();
            }

            Console.WriteLine("Remaining cards in deck:");
            Console.WriteLine(deck);

        }
    }
}
```