CS2530 - A03 Cards

Learning Objectives:

- Increase your familiarity with enums and structs
- Get confident in the use of the generic collection class List<T>
- Implement a generic interface
- Look up an algorithm in pseudo-code and translate it into C#
- Build a good habit by overriding ToString

Description:

Create a console project named Cards that includes a struct Card, two classes Deck and Hand, and two enums: Rank and Suit. You can use Lab Card as a starting point for this assignment.
Make sure to save a copy though if you have not checked off the lab yet.

Ad enum Suit:

Set the values of the enumerators in Suit to the integer values corresponding to the characters ♠, ♣, ♦, and ♥, and use these symbols to represent the suit when override the ToString method of Card.

Ad enum Rank:

The named constants are: Ace, Deuce, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, and King.

Ad struct Card:

- 2 auto-implemented **properties Rank and Suit**
  Both properties are declared public with a **private** setter (it is a good practice to make structs immutable)
- One pubilc **constructor** with 2 parameters that initializes the properties Rank and Suit
- An overridden implementation of **ToString**
  King of Spades should look like this: **♠ King**, Nine of Diamonds should look like this: **♦ Nine**
  TIP: you can get you the character representation of the property Suit by casting it to a character: **(char)Suit**
- Implement the interface `IComparable<T>`
  This will allow you to use the Sort method from List<T> on a list of cards.
  Both the rank and the suit determine whether a card is smaller or greater. Rank should have the greater weight. That means a Three is always less than a King regardless of the suit.
  Here is a link with a code example on how to implement IComparable<T>.
  Hint: you'll need to substitute Card for T

The class Deck:

The class Deck represents a deck of cards. A deck can have a varying number of cards. By default it is initialized with a full, complete deck of cards. The class Deck has the following members:

- It has an auto-implemented **property** of type List<Card> called **Cards**.
  public List<Card> Cards { get; private set; }
- It has **2 constructors**:

   - One constructor is a default constructor (no parameters). It initializes the property Cards with a new deck. Call the static method NewDeck to create the new deck.
   - One constructor has one parameter of type List<Card> :
     public Deck(List<Card> cardDeck)
     It initializes the property Cards with the card deck passed as parameter
- It has **three public instance methods**: `Shuffle`, `Deal`, and an overridden version of `ToString`
   - **`Shuffle`** has no parameter and returns no value. All it does it shuffling the deck.
     Implement the simple shuffling algorithm called Knuth shuffle or Fisher Yates shuffle (same algorithm) It is easy to implement and works well for our purpose. You find it described here under the section 'The modern algorithm'
     TIP: The class Random provides a method Next
   - **`Deal`** has the following method header: **`public List<Card> Deal(int i)`**
     The integer parameter specifies how many cards should be dealt. The cards that are dealt (returned to the calling method) need to be removed from the deck.
     If the number specified in the parameter exceeds the number of cards in the deck no card should be dealt or removed. In that case you can either return an empty hand or throw an exception (e.g. ArgumentOutOfRangeException).
   - An overridden implementation of **ToString**
     ToString creates and returns a string based on the cards in the deck.
     When you print out this string the cards should be listed in four equally spaced columns.
     For a complete sorted deck of cards the output would look like this:
```
♠ Ace    ♣ Ace    ♥ Ace    ♦ Ace
♠ Deuce  ♣ Deuce  ♥ Deuce  ♦ Deuce
♠ Three  ♣ Three  ♥ Three  ♦ Three
. . .
```
     Use StringBuilder from the namespace System.Text to create the string described above.
     TIP: By using the placeholder "**{0,-9}**" instead of "**{0}**" you create left-aligned columns of width 9

- It has **one public static method** called NewDeck
   - **NewDeck** has the following method header: **public static List<Card> NewDeck()**
     This method creates and returns a new, complete, and sorted deck.
     Do NOT call a sort method though. Innstead loop through the ranks and suits in the proper order so that it will be already sorted as you create it.
     TIP: use the static method GetValues to loop through all Rank enumerators
     foreach (Cards.Rank r in Enum.GetValues(typeof(Rank)))

The class Hand:

- Like the class Deck it has an auto-implemented **property** of type List<Card> called **Cards**.
- It has **1 constructors**:
     The constructor has one parameter of type List<Card> called newHand.
     Use the method Sort from class List<T> to sort the new hand. Then assign it to the property Cards.
- It has **three public instance methods:** IsFlush, IsStraight, and an overridden ToString
  Here is some information about flush and straight: http://en.wikipedia.org/wiki/List_of_poker_hands
   - **IsFlush** returns true if the hand is a flush and false otherwise
   - **IsStraight** returns true if the hand is a straight and false otherwise

- An overridden implementation of **ToString**
  When you print out the string returned from this method the cards of the hand should be listed equally spaced in a single line.
  If the hand is a straight or a flush mention that at the end of the line (see output)

The Test Method:

I attached a file called Program.cs. It includes the Main method and it tests the code described above.
 Download it and use it to test your code.

Note that your *Shuffeled Deck* will look different than the output provided. However, *New Deck* and the *Dealing heands of 5 cards* should look the same.

Output:

```
New Deck:

♠ Ace    ♣ Ace    ♥ Ace    ♦ Ace
♠ Deuce  ♣ Deuce  ♥ Deuce  ♦ Deuce
♠ Three  ♣ Three  ♥ Three  ♦ Three
♠ Four   ♣ Four   ♥ Four   ♦ Four
♠ Five   ♣ Five   ♥ Five   ♦ Five
♠ Six    ♣ Six    ♥ Six    ♦ Six
♠ Seven  ♣ Seven  ♥ Seven  ♦ Seven
♠ Eight  ♣ Eight  ♥ Eight  ♦ Eight
♠ Nine   ♣ Nine   ♥ Nine   ♦ Nine
♠ Ten    ♣ Ten    ♥ Ten    ♦ Ten
♠ Jack   ♣ Jack   ♥ Jack   ♦ Jack
♠ Queen  ♣ Queen  ♥ Queen  ♦ Queen
♠ King   ♣ King   ♥ King   ♦ King


Shuffeled Deck:

♣ Ten    ♠ Three  ♠ Six    ♣ Queen
♠ Four   ♥ Nine   ♥ Deuce  ♠ Seven
♦ Six    ♣ Nine   ♥ Ten    ♥ King
♣ Ace    ♣ Five   ♦ King   ♠ King
♦ Ace    ♠ Jack   ♦ Jack   ♠ Eight
♣ Three  ♦ Eight  ♥ Six    ♣ Jack
♠ Ace    ♠ Five   ♥ Jack   ♥ Ace
♦ Four   ♥ Four   ♥ Three  ♥ Queen
♦ Nine   ♣ Eight  ♣ Six    ♠ Ten
♦ Ten    ♦ Five   ♥ Seven  ♣ Four
♦ Deuce  ♦ Three  ♥ Eight  ♦ Seven
♠ Nine   ♣ King   ♦ Queen  ♥ Five
♣ Seven  ♠ Deuce  ♠ Queen  ♣ Deuce


Dealing hands of 5 cards:

♠ Five   ♠ Six    ♠ Seven  ♠ Eight  ♠ Nine     Straight Flush
♣ Four   ♣ Six    ♣ Seven  ♣ Eight  ♣ Queen    Flush
♦ Six    ♦ Seven  ♥ Eight  ♥ Nine   ♣ Ten      Straight
♣ Deuce  ♣ Six    ♣ Nine   ♦ Ten    ♣ Queen
♣ Eight  ♠ Ten    ♥ Jack   ♠ Queen  ♦ King
```

Turn in:

Zip up your solution (including the solution file!) . Name it **A03.zip** and turn it in via Canvas.