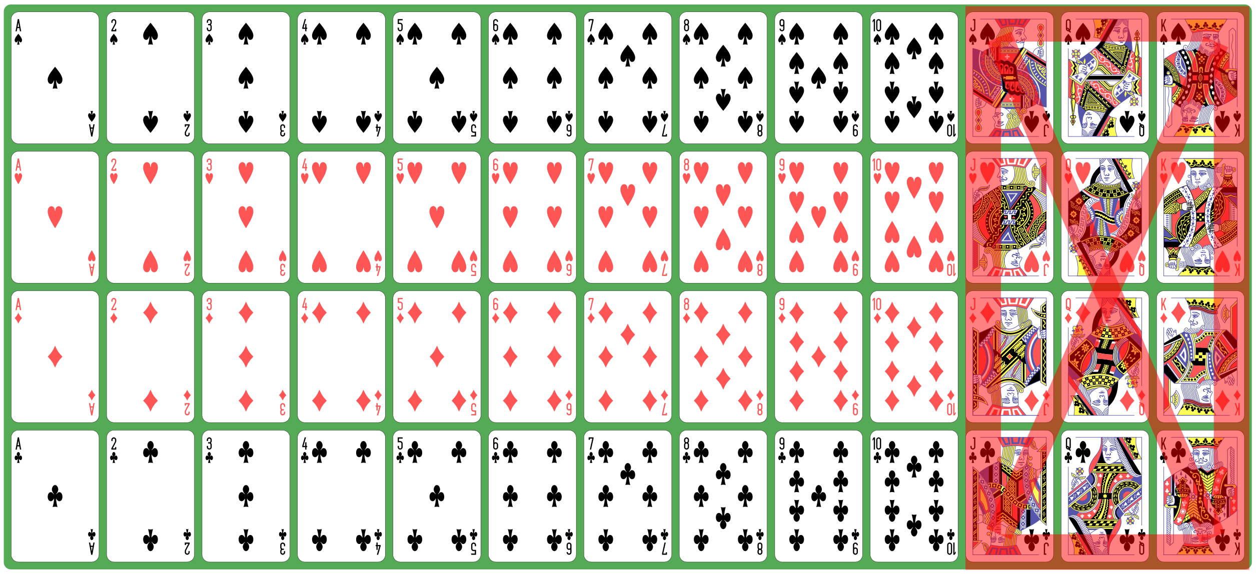
Model and Write a Program Simulating a Deck of Playing Cards



Overall Objective

- Create a modular object model of a deck (or collection) of playing cards with the following operations
    - Create a new deck of 40 playing cards
        - The initial order must be Spades 1-10, Hearts 1-10, Diamonds 1-10, Clubs 1-10
    - Shuffle (random sort order) the playing cards in the deck
    - Print the current order of the playing cards
- Create a short console test program that uses the card deck model in the following way:
    - Create a new deck of playing cards
    - Print the current order of the playing cards
    - Shuffle the playing cards
    - Print out the current order of the playing cards
    - Shuffle the playing cards again
    - Print out the current order of the playing cards to verify that the shuffle is random

Characteristics of Playing Cards to Consider

- Suit
  - Spade
  - Heart
  - Diamond
  - Club
- Value
  - 1-10 (no face cards – Jack/Queen/King)

Hints to Keep in Mind

- Think <u>modular, reusable</u> class structure.
  - Another developer should be able to pick up this class (or classes) and easily use it in their own application. Ex: Another developer is writing a poker game application with your deck.
  - Another developer should be able to extend this class (or classes) to add new operations without sacrificing other functions.
  - Remember product owners love to change requirements, so try to hardcode as little as possible to avoid messy changes later.
- Remember good programming practices
  - Single responsibility principle
  - Don't repeat yourself
  - Keep it simple and easily readable
- C# Hints:
  - Don't forget appropriate public/private modifiers on classes/properties/methods.
  - Iterating over an enumeration:
    - foreach(Suit suit in Enum.GetValues(typeof(Suit)))
    - foreach(var suit in Enum.GetValues(typeof(Suit)).Cast<Suit>())   // need Cast if using var
  - Random number generation:
    - var rng = new Random();   // only need to instantiate once in current scope
    - var num = rng.Next(i);   // random integer between 0 and i-1
    - var num = rng.Next(i, j); // random integer between I and j-1
  - Console output
    - Console.WriteLine(String.Format("Some string with {0} printed inside", variable));
    - Console.WriteLine($"some string with {variable} printed inside"); // string interpolation
  - Console.ReadKey() is useful to be able to read your output before it disappears