High Card Program Design Overview:

Suit and Face Enums:

Firstly, I decided to implement a method of assigning each card their respective face, suit, and value, to make my implementation of the High Card program further resemble its real-life namesake. As the most efficient way to generate my game's deck would be via a for-loop, I chose to store the value of each suit and face value as enum collection, which would mean I could iteratively assign each card their respective face and suit by manipulating the for-loop's index.

Card Class:

As this game's deck would be compiled of three different sub-types of cards, 'SuitCards' (standard number cards), 'FaceCards' (Ace, Jack, Queen and King), and 'WildCards' (Joker), I decided to use Inheritance and Polymorphism to construct my deck. Through using Inheritance, I would be able to store each sub-type of card, with their differing attributes and methods, within one collection. Therefore, I implemented a parent 'Card' class, and child 'SuitCards', 'FaceCards' and 'WildCards' class that inherited from the 'Card' class.

Scoreboard Class:

To allow for the possibility of increasing the number of active players available to take part in this game of High Card, I chose to change the Scoreboard implementation from an enum collection to a class that holds its own name and score, as the number of active players can be easily increased through instantiating more Scoreboard objects in the Game class.

Deck Class:

When it comes to the Deck class, I chose to store the deck as two different collections, onece as a List collection (orderedDeck), and again as a Stack collection (gameDeck). I decided to store the game's final deck as a Stack, because Stack's Pop method lent itself to the action of dealing cards, as the Pop method not only returns the last element, but also removes the element from the collection. I was then able to monitor how many cards were left in the deck by checking if the Stack was empty.

However, a Stack collection is not easily randomised, and therefore I chose to store the initial deck as a List, as List collections can be more easily manipulated. Additionally, I chose to maintain the List collection in memory throughout the game, so that when the Stack is empty, the deck would not need to be rebuilt before being randomised and pushed back onto the Stack.

Game Class:

The game is then controlled by the Game class, which starts by instantiating three Scoreboard objects and a Deck object, before dealing a card to each player, and using logic to determine which player has won the round.

Every time a round is won the respective Scoreboard object has its Score variable increased by one, and a string is returned to the console informing the player who won the round.

Once all rounds have been played, a final string is then returned to the player informing them of the number of rounds they won, how many rounds the dealer won, and how many rounds resulted in a tie.