## Question 3:

When it came to structuring my program, I first decided to plan out how the game would function, and the separate operations that would make up the overall program. My plan highlighted three main areas with the first being the coordinator, which would become my main class (named Question2 in program) and would oversee calling the other classes in my program. If the program was developed into a more complete game, which followed a Model View Controller design pattern, the Question2 class would take on the role of the controller class, which is where the player's input would be parsed to the other classes.

The second area was the deck, which became my second class. The Deck class has been used to contain the attributes required to generate the game's deck, and the methods that would be called to return a deck generated using the parameters parsed by the Question2 class. The attributes assigned to the Deck class included a string array, which contained the name of each suit (Spades, Clubs, Diamonds and Hearts), an integer variable that held the size of each suit, an integer variable that held the size of the total deck, an integer variable that held the number of decks that would be used to create the final 'game deck', and a list of key value pairs that hold a string and an integer variable. The list of key value pairs is how I decided to store the final 'game deck', as it would allow me to store both the face value of each card, and an overall value for each card, which would allow me to settle ties between two cards with the same face value, but a different suit.

The first method in my Deck class is the constructor, which is used to set the values of the previously mentioned integer variables, and for calling the second method of the Deck class (BuildDeck()), which returns a complete deck.

The BuildDeck() starts by initialising the int variable value, which will act as a counter for the value of each card in the deck. I then went on to generate a standard 52 card deck by nest a for loop inside of a foreach loop, the for loop iterated through each card in the chosen suit size, while the foreach loop iterated through each suit. Inside the nested foreach loop is then a switch statement that would take any card that had the numbers 1, 11, 12 and 13, and parse the name of that particular card instead (1 = Ace, 11 = Jack, 12 = Queen, and 13 = King).

To allow for the game to use multiple decks, the BuildDeck() method is called iteratively from another for loop in the constructor, this would allow for multiple decks with the same values to be built and concatenated to make the final ‘game deck’. Before the ‘game deck’ is then returned to the Question2 class, it adds one additional card on at the end (Joker) which means the Joker card has the highest value of any other card in the game deck, making it the wild card.

The final area is the game class, which has been called the HighCard() class, and is the final call made by the constructor class, as It handles the playing and outcome of a game. As the HighCard() has only one purpose, and that is to play the game, there is only one method in the class called Play(). The Play() method starts by first retrieving the list of key value pairs stored in the deck object parsed in the methods arguments. The method then goes onto instantiate a new Random object, which is used to select two random key value pairs from the list of key value pairs retrieved earlier.

After two key value pairs have been randomly selected from the list of key value pairs, the value of each pair is compared using a series of if statements. If the player’s card (key value pair) is greater than the opponent’s, the player wins, if the opponent’s card is greater than the player’s, the

opponent wins, and if the player and opponent have a card with the same value (in the event of multiple decks) the Play() method is called recursively until either the player or the opponent wins.