Classes:

Card:

Plan: The card class constructs an object with a suit and number.

Outline: 2 fields, string suit and int number, an overridden ToString() and a constructor with two argument bringing in values based on enums with values based on the primary position in the array in the Deck class.

Pseudocode:

Declare private fields string suit and int number;

Create properties Suit and Number which set and get the values of the private fields;

Constructor creates object based on the two parameters that are given as arguments;

ToString lists the object as follows: {Number} of {Suit};

Deck:

Plan: This class holds a 2d array of Cards as well as a Bankroll, a hand array and a pool of bet money. It is responsible for determining whether there is a payout or not. Manipulates Card objects and the Bankroll. There are also two enums: a suit enum with 0(clubs) to 3(spades), and a number enum from 0(Ace) to (12)King

Outline: 4 fields, double Bankroll for amount of money possessed, Card[DECK_SIZE(52)] deckCards for every Card object, Card[HAND_SIZE] hand for 5 cards in the player's hand and double pool for amount bet. A default constructor that sets Bankroll to 1000 creates all cards in the array then sets the Hand sets bet to 0. A shuffle method. A payout method. A bet method. A deal method. And A discard method.

Pseudocode:

Declare fields;

Create properties which set and get their fields;

Constructor sets Bankroll creates all cards in the Deck array and sets the hand array;

Sets up all the methods;

Main:

Plan: Displays and prompts the user.

Outline: Implements the Deck Class and asks the user to participate in a video poker game.

PseudoCode:

Shuffle the deck (without creating a new deck);

Ask the player how much they wish to bet (minimum $1 increments). They are not allowed to bet more than what they have;

Reduce their bankroll by the amount they bet;

Deal 5 cards to the player from a shuffled deck;

Allow the player to discard up to 4 of the 5 cards;

Deal new cards from the same deck to replace the discarded cards;

Evaluate the hand, telling the user what hand they received, and how much they have won, if anything;

If they won, add the winnings to the player's bankroll;

If the player is out of money, thank them for playing, and quit!;

If the player still has money, ask them if they wish to continue. If so, go to step 1. Otherwise, quit;

Functional Decomposition:

Deck:

Public void Shuffle()

Inputs: none
Outputs: shuffles position of cards
Tasks:
-Create random generator
-Loop every position until length of array
-Generate random position
-swap temporary position with current position
-swap current position with random position
-swap random position with temporary position
Code for reference:

```
var rand = new Random();
for (int i = cards.Length - 1; i > 0; i--)
{
int n = rand.Next(i + 1);
int temp = cards[i];
cards[i] = cards[n];
cards[n] = temp;
}
```

Public void Payout()

Inputs: int bet
Output: checks if win, pays amount based on the type of win.
Tasks:
-Check if it's any of the win condition methods.
-If it is a win condition (use if-else statements), pay the appropriate amount by multiplying the pool and adding it to the Bankroll and say: "Win condition: {win condition}"
-else, output an error message, "no winning hand detected"

Public bool RoyalFlush()

Inputs: hand array
Outputs: returns bool
Tasks:
Checks to see if StraightFlush is true and that the first card in hand is "ten"

Public bool StraightFlush()

Inputs: hand array
Outputs: returns bool
Tasks:
Checks if straight and flush are true. If both are true, return true.

Public bool FourOfAKind()

Inputs: hand array
Outputs: returns bool
Tasks:
Sees if there is 4 of a particular number. Counts the number of instances.

Public bool FullHouse()

Inputs: hand array
Outputs: returns bool
Tasks:
If a three of a kind and a pair exist, return true.

Public bool Flush()

Inputs: hand array
Outputs: returns bool
Tasks:
Checks to see if all cards are the same suit.

Public bool Straight()

Inputs: hand array
Outputs: returns bool
Tasks:
Checks to see if the second card is the first card + 1, the third + 1 of the second and so on. Doesn’t matter if they aren’t the same suit.

Public bool ThreeOfAKind()

Inputs: hand array
Outputs: returns bool
Tasks:
Sees if there is 3 of a particular number. Counts number instances.

Public bool TwoPair()

Inputs: hand array
Outputs: returns bool
Tasks:
checks if there are 2 distinct pairs of numbers regardless of suit.

Public bool OnePair()

Inputs: hand array
Outputs: returns bool
Tasks:
Sees if there is one pair of numbers regardless of suit. Numbers must be higher than or equal to Jack.

Public void BetMoney(int money)

Inputs: int money
Outputs: subtracts from Bankroll
Tasks:
Subtracts from Bankroll.

Public void Deal()

Inputs:
Outputs: sets hand
Tasks:
-Loop for 5 cards.
-sets x card in hand as first card in deck.
-sets x card in deck as null
-when loop is done, call sort

Public void Discard(int input)

Inputs: input of position in hand
Outputs: changes card from hand to card in the deck
Tasks:
-at deck position of input, set to hand position of input
- while loop starts at 6th position in deck
-cycle through positions until non null card is found
-sets hand position of input as that card
-when loop is done, call sort

Public void Sort()

Inputs: hand array
Outputs: orders cards from lowest to highest number
Tasks:
-performs an insertion sort

Public void PutBack()

Inputs: none
Outputs: Puts back the cards from the hand to the deck.
Tasks:
-for every card in hand. . .
-cycle through the deck for nulls
-swap the card from hand to the null in deck

Public void SetDeck()

Inputs: none
Outputs: constructs all cards in the deck
Tasks:
-cycle through the deck
-add a new card for each position according to the enums