# AP Computer Science — Partner Project - VideoPoker

## *ASSIGNMENT OVERVIEW*

In this assignment you'll be creating a small package of files which will allow a user to play a game of Video Poker. For this assignment you'll be implementing four classes: `Card`, `Deck`, `PokerHand`, and `PokerGame`.

This assignment is worth 50 points and is due on the *crashwhite.polytechnic.org* server at 23:59:59 on the date given in class.

## *BACKGROUND*

The game of Poker, 5-Card Draw, is a classic. Some poker games are played with friends, but Video Poker machines are a common sight in Las Vegas casinos. To program your PokerGame, you'll need to know how to play Poker, 5-Card Draw:

A card deck contains 52 cards, 13 of each suit Spades, Hearts, Diamonds, and Clubs. At the beginning of the game, the deck is shuffled. Then the top five cards of the deck are "dealt" to the player. The player evaluates his/her hand, and can reject none, some, or all of the cards in this hand. The rejected cards are replaced from the top of the deck so the player again has five cards. Now the hand is scored according to the following classification:

- No pair—The lowest hand, containing five separate cards that do not match up to create any of the hands below.
- One pair—Two cards of the same value, for example two queens.
- Two pairs—Two pairs, for example two queens and two 5's.
- Three of a kind—Three cards of the same value, for example three queens.
- Straight—Five cards with consecutive values, not necessarily of the same suit, such as 4, 5, 6, 7, and 8. The ace can either precede a 2 or follow a king.
- Flush—Five cards, not necessarily in order, of the same suit.
- Full House—Three of a kind and a pair, for example three queens and two 5s
- Four of a Kind—Four cards of the same value, such as four queens.
- Straight Flush—A straight and a flush: Five cards with consecutive values of the same suit.

## *PROGRAM SPECIFICATION*

Create a package of Java classes that can be used to model a one-person game of Poker, including:

a. a `Card` class that models a playing card
b. a `Deck` class that models a collection of 52 playing cards
c. a `PokerHand` class that models a hand of five cards being played in a game of poker
d. a `PokerGame` class that manages a player's game of 5-Card Draw

## *DELIVERABLES*

**VideoPoker.zip**

This single file will be a zipped directory (folder) of your BlueJ project. It will include as a minimum the four files listed above (or their equivalents as designed by you), any other classes you create during the development of your program), and a `package.BlueJ` file.

To submit your assignment for grading, copy your file to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before the deadline.

***ASSIGNMENT NOTES***

- For this assignment you being asked to *implement* a series of classes based on the public interface, which is supplied to you via a series of JavaDocs. You do not need to design any classes yourself.
- An outline for each class is included in this document. Use it as a jumping off point for your implementation of each class.

***GETTING STARTED***

1. If you are not familiar with the game of 5-Card Draw, make sure you have the chance to play a few games so that you understand the rules before you start trying to program the game.
2. Write the `Card` class and test it thoroughly before moving on to the more complex classes.
3. Write the `Deck` class and figure out how to use it to create a deck of cards that can be manipulated.
4. Write the `PokerHand` class and experiment with using it to create a hand of cards. In particular, the `evaluate` method will provide some interesting challenges.
5. Write the `PokerGame` class which consists of a `main` method that runs the Poker Game.

***QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)***

1. Which part of this assignment is the more challenging for you: the design or the coding?
2. How much were you able to anticipate the methods needed for this project, and how much did you discover as you were going through writing the code?
3. How useful or productive did you find it, being able to work on this assignment with someone else? Did you "divide and conquer" the tasks that needed to be done, or did you work together the entire time, watching over each other's shoulders? Take a look at the Wikipedia entry on Pair Programming at *https://en.wikipedia.org/wiki/Pair_programming* . Were any of the strengths of that system evident in your work on this project?

## *SAMPLE INTERACTIONS*

```
Let's play some poker!
0. Queen of Hearts
1. Jack of Clubs
2. Nine of Hearts
3. Six of Diamonds
4. Three of Hearts
Card # to discard? -1 if no more discards: 4
0. Queen of Hearts
1. Jack of Clubs
2. Nine of Hearts
3. Six of Diamonds
Card # to discard? -1 if no more discards: 3
0. Queen of Hearts
1. Jack of Clubs
2. Nine of Hearts
Card # to discard? -1 if no more discards: -1
Drawing new cards...
Queen of Hearts
Jack of Clubs
Nine of Hearts
Seven of Clubs
Four of Spades

nothing
Wanna play again? (Y/n):
y
0. King of Clubs
1. Queen of Spades
2. Queen of Clubs
3. Seven of Hearts
4. Five of Hearts
Card # to discard? -1 if no more discards: 0
0. Queen of Spades
1. Queen of Clubs
2. Seven of Hearts
3. Five of Hearts
Card # to discard? -1 if no more discards: 3
0. Queen of Spades
1. Queen of Clubs
2. Seven of Hearts
Card # to discard? -1 if no more discards: 2
0. Queen of Spades
1. Queen of Clubs
Card # to discard? -1 if no more discards: -1
Drawing new cards...
King of Diamonds
Queen of Clubs
Queen of Spades
Nine of Clubs
Five of Diamonds

1 pair(s) found (Queen )
Wanna play again? (Y/n): n
```

## *The Card Class*

```java
/**
 * The Card class represents a card with a suit, a name, and a value.
 *
 */
public class Card
{

    // instance variables

    /**
     * A Card object is created on the basis of an integer value from 0-51. Cards 0-12 are Spades,
     * cards 13-25 are Hearts, cards 26-38 are Diamonds, and cards 39-51 are Clubs.  Within each
     * suit, an Ace has a value of 14, King = 13, Queen = 12, down to Two = 2.
     * @param theNumber the number of the card, a value from 0-51
     */
    public Card(int theNumber)
    {


    }

    /**
     * The toString method returns a String representing this cards name and suit in the format
     "name of suit" (eg. "Ace of Hearts", "Three of Diamonds", etc.)
     * @return a string representing the name of the card
     */
    @Override
    public String toString()
    {



    }

    /**
     * The name of the card's value: Ace, Two, Jack, King, etc.
     * @return a string representing name of card's value (with initial letter capitalized))
     */
    public String getName()
    {


    }

    /**
     * The suit of the card: Spades, Hearts, Diamonds, or Clubs
     * @return a string representing the suit of the card (with initial letter capitalized)
     */
    public String getSuit()
    {


    }

    /**
     * The value of the card as an integer. 2-King maps to 2-13. The Ace although it comes first
     * and could have a value of 1, is given value of 14 in this class, superseding that of a King.
     * @return an integer value representing the rank of the card
     */
    public int getValue()
    {


    }
}
```

## *The Deck Class*

```
/**
 * The Deck class simulates manipulating a deck of playing cards.
 */

import java.util.ArrayList;

public class Deck
{
    // instance variables

    /**
     * Constructor for objects of class Deck creates an ArrayList of Card objects for a
     * standard deck (no Jokers), initially in order (unshuffled)
     */
    public Deck()
    {

    }

    /**
     * The shuffle method creates new full deck of cards and mixes them up so a new deal can begin
     */
    public void shuffle()
    {

    }

    /**
     * This deal method removes a single card from the top of the deck and returns it.
     * @return a Card object from the deck
     */
    public Card deal()
    {

    }

    /**
     * This deal method removes multiple cards from top of deck and returns them as an ArrayList.
     * @param amount the number of cards to deal
     * @return an ArrayList of Cards from the deck
     */
    public ArrayList<Card> deal(int amount)
    {

    }

    /**
     * The cardsRemaining method tell how many cards are left in the deck
     * @return the number of cards remaining
     */
    public int cardsRemaining()
    {

    }

    /**
     * The getDeck() method returns an ArrayList of all the remaining cards in the deck. Useful
     * for displaying the deck during debugging, writing methods that cheat, etc.
     * @return an ArrayList with all the remaining cards in the Deck, from the top on down
     */
    public ArrayList<Card> getCurrentDeck()
    {

    }
}
```

## *The PokerHand Class*

```
/**
 * The PokerHand class manages a hand of 5 cards, and analyzes
 * a hand to identify its result.
 */

import java.util.ArrayList;
import java.util.Scanner;

public class PokerHand
{
    // instance variables

    /**
     * Constructor for a Poker hand of 5 cards
     * @param deck a deck of cards that the hand will be created from
     */
    public PokerHand(Deck deck)
    {

    }

    /**
     * The drawCards method gets a specified number of cards
     * from the deck and places them in the hand.
     * @param numOfCards the number of cards to draw
     */
    public void drawCards(int numOfCards)
    {

    }

    /**
     * the getCards method returns an ArrayList of the cards currently held
     * @return an ArrayList of Card objects held in the hand
     */
    public ArrayList<Card> getHand()
    {

    }

    /**
     * The discard method gets rid of a single card at the specified index in the hand.
     */
    public void discard(int index)
    {

    }

    /**
     * The sortHand method puts the cards in order of value, from highest to lowest.
     * Userful for displaying the cards in a meaningful way, and analyzing whether or
     * not there's a straight.
     */
    public void sortHand()
    {

    }

    /**
     * The evaluate method identifies what kind of hand the player is holding.
     * Royal Flush, straight flush, four of a kind, full house, flush,
     * straight, three of a kind, two pair, pair of jacks or better.
     * @return a string with the name of the hand
     */
    public String evaluate()
```

```java
    {

    }

    /**
     * The isStraight method identifies if the hand holds a series of sequential cards
     * @return true if the cards are in order
     */
    public boolean isStraight()
    {

    }

    /**
     * The isFlush method identifies if the hand has all the same suits
     * @return true if the cards
     */
    public boolean isFlush()
    {

    }

    /**
     * the highCard method identifies the index of the highest valued card in the hand
     * @return card with highest value
     */
    public int highCard()
    {

    }

    /**
     * The count method counts how many cards in the hand match the value of the given index.
     * @param Card a card with a given value
     * @return count of number of total cards in hand with that value
     */
    public int count(Card oneCard)
    {

    }

    /**
     * the toString method return the cards in the hand
     */
    public String toString()
    {

    }

    /**
     * the numOfCards method return the number of cards in the hand
     * @return an integer count of the number of cards in the hand
     */
    public int numOfCards()
    {

    }

    /**
     * the createHand method is used to artificially create a hand of cards for
     * testing methods. The Deck method is bypassed completely.
     */
    public void createHand()
    {

    }
}
```

## *The PokerGame main method*

```java
/**
 * The PokerGame class deals a hand of 5 cards to a player,
 * allows them to discard and draw cards, and reports the results
 * of the game.
 *
 * @author Richard White
 * @version 2013-11-21
 */

import java.util.Scanner;

public class PokerGame
{
    /**
     * The PokerGame class allows the user to play a game of poker
     */
    public static void main(String[] args)
    {
        // Initialize the Scanner
        System.out.println("Let's play some poker!");
        // Instantiate a deck of cards
        // Shuffle the deck
        // set boolean variable for play loop
        while (playAgain)
        {


            System.out.print("Wanna play again? (Y/n): ");
            String moreGames = in.next();
            if (moreGames.equalsIgnoreCase("N"))
                playAgain = false;
        }
        System.out.println("Thanks for playing!")
    }
}
```