AP CSA Short Term Project 2 (Design a Deck Class)

Description

Students will work in small groups to produce a Java Class that defines a deck of playing cards.

Standards

IT-PGA-2 Describe the software application life cycle and use a prototype development model to develop applications.
IT-PGA-4 Design, develop, and implement accessible and usable interfaces, and analyze applications for engaging the user.

Business Ethics
Students will model work readiness traits required for success in the workplace including teamwork, multitasking, integrity, honesty, accountability, punctuality, time management, and respect for diversity.

Expectations

Students are expected to use the skills and concepts learned in the course to design a working deck class.

Objectives

Students will design a class that represents a deck of playing cards.

Students will implement the class by creating an object of the class and test all parts.

The class should include private attributes for a List of Cards & size of deck.

The class should include methods to return private attributes, to deal a card, to track dealt cards, and to output the attributes of a card object as a String.

Project Time

The project will take approximately 3-4 hours to complete.

Rubric

50 points	Class is complete with all the features listed in objectives
25 points	Class and tester are written to industry accepted syntax
25 points	There are no errors when the code is compiled

Well done. Keep it up and you will be ready for the test. 100 points

50 points Class is complete with all the features listed in objectives - 50

25 points Class and tester are written to industry accepted syntax - 25

25 points There are no errors when the code is compiled - 25

```
/**
 * Card.java
 *
 * <code>Card</code> represents a playing card.
 */
public class Card {

	/**
	 * String value that holds the suit of the card
	 */
	private String suit;

	/**
	 * String value that holds the rank of the card
	 */
	private String rank;

	/**
	 * int value that holds the point value.
	 */
	private int pointValue;
```

```
/**
	 * Creates a new <code>Card</code> instance.
	 *
	 * @param cardRank  a <code>String</code> value
	 *                  containing the rank of the card
	 * @param cardSuit  a <code>String</code> value
	 *                  containing the suit of the card
	 * @param cardPointValue an <code>int</code> value
	 *                  containing the point value of the card
	 */
	public Card(String cardRank, String cardSuit, int cardPointValue) {
		//initializes a new Card with the given rank, suit, and point value
		rank = cardRank;
		suit = cardSuit;
		pointValue = cardPointValue;
	}



	/**
	 * Accesses this <code>Card's</code> suit.
	 * @return this <code>Card's</code> suit.
	 */
	public String suit() {
		return suit;
	}
```

```
        /**
         * Accesses this <code>Card's</code> rank.
         * @return this <code>Card's</code> rank.
         */
        public String rank() {
                return rank;
        }

/**
         * Accesses this <code>Card's</code> point value.
         * @return this <code>Card's</code> point value.
         */
        public int pointValue() {
                return pointValue;
        }

        /** Compare this card with the argument.
         * @param otherCard the other card to compare to this
         * @return true if the rank, suit, and point value of this card
         *           are equal to those of the argument;
         *       false otherwise.
         */
        public boolean matches(Card otherCard) {
                return otherCard.suit().equals(this.suit())
                        && otherCard.rank().equals(this.rank())
                        && otherCard.pointValue() == this.pointValue();
        }
```

```
        /**
         * Converts the rank, suit, and point value into a string in the format
         *     "[Rank] of [Suit] (point value = [PointValue])".
         * This provides a useful way of printing the contents
         * of a <code>Deck</code> in an easily readable format or performing
         * other similar functions.
         *
         * @return a <code>String</code> containing the rank, suit,
         *         and point value of the card.
         */
        @Override
        public String toString() {
                return rank + " of " + suit + " (point value = " + pointValue + ")";
        }
}

import java.util.List;
import java.util.ArrayList;

/**
 * The Deck class represents a shuffled deck of cards.
 * It provides several operations including
 *      initialize, shuffle, deal, and check if empty.
 */
public class Deck {
```

```
/**
 * cards contains all the cards in the deck.
 */
private List<Card> cards;

/**
 * size is the number of not-yet-dealt cards.
 * Cards are dealt from the top (highest index) down.
 * The next card to be dealt is at size - 1.
 */
private int size;



/**
 * Creates a new <code>Deck</code> instance.<BR>
 * It pairs each element of ranks with each element of suits,
 * and produces one of the corresponding card.
 * @param ranks is an array containing all of the card ranks.
 * @param suits is an array containing all of the card suits.
 * @param values is an array containing all of the card point values.
 */
public Deck(String[] ranks, String[] suits, int[] values) {
        /* *** TO BE IMPLEMENTED IN ACTIVITY 2 *** */

  cards = new ArrayList<Card>();

  for(int j = 0; j < ranks.length; j++)
```

```
  {
    for (String suitString : suits)
    {
       cards.add(new Card(ranks[j], suitString, values[j]));
    }

  }
  size = cards.size();
}



/**
 * Determines if this deck is empty (no undealt cards).
 * @return true if this deck is empty, false otherwise.
 */
public boolean isEmpty() {
        /* *** TO BE IMPLEMENTED IN ACTIVITY 2 *** */
        if (size == 0)

          return true;

        else

          return false;

}
```

```
	/**
	 * Accesses the number of undealt cards in this deck.
	 * @return the number of undealt cards in this deck.
	 */
	public int size() {
		/* *** TO BE IMPLEMENTED IN ACTIVITY 2 *** */

return size;

	}

	/**
	 * Randomly permute the given collection of cards
	 * and reset the size to represent the entire deck.
	 */
	public void shuffle() {
		/* *** TO BE IMPLEMENTED IN ACTIVITY 4 *** */
	}

	/**
	 * Deals a card from this deck.
	 * @return the card just dealt, or null if all the cards have been
	 *	previously dealt.
	 */
	public Card deal() {
		/* *** TO BE IMPLEMENTED IN ACTIVITY 2 *** */
if(isEmpty())
```

```
 return null;

else

 size--;
 return cards.get(size);



    }

    /**
     * Generates and returns a string representation of this deck.
     * @return a string representation of this deck.
     */
    @Override
    public String toString() {
        String rtn = "size = " + size + "\nUndealt cards: \n";

        for (int k = size - 1; k >= 0; k--) {
            rtn = rtn + cards.get(k);
            if (k != 0) {
                rtn = rtn + ", ";
            }
            if ((size - k) % 2 == 0) {
                // Insert carriage returns so entire deck is visible on console.
                rtn = rtn + "\n";
            }
```

```
                }

                rtn = rtn + "\nDealt cards: \n";
                for (int k = cards.size() - 1; k >= size; k--) {
                        rtn = rtn + cards.get(k);
                        if (k != size) {
                                rtn = rtn + ", ";
                        }
                        if ((k - cards.size()) % 2 == 0) {
                                // Insert carriage returns so entire deck is visible on console.
                                rtn = rtn + "\n";
                        }
                }

                rtn = rtn + "\n";
                return rtn;
        }
}

/**
 * This is a class that tests the Deck class.
 */
public class DeckTester {

        /**
         * The main method in this class checks the Deck operations for consistency.
         *      @param args is not used.
```

```
 */
public static void main(String[] args) {
        /* *** TO BE IMPLEMENTED IN ACTIVITY 2 *** */
        String [] ranks = {"Ace", "King", "Queen", "Jack"};
        String [] suits = {"Spades", "Diamonds"};
        int [] pointValues = {1, 13, 12, 11};

        Deck a = new Deck(ranks, suits, pointValues);

        //Tells if the deck is empty or not, shows size = 8 because of "to.String()"
function and shows the cards that have been and have not been dealt.
        System.out.println("Deck empty? " + a.isEmpty());
        System.out.println(a.toString());
        a.deal();

        //Tells if deck is empty or not, size would show 7, and the undealt cards would
show 6 cards and dealt cards would show only 1 card.
        System.out.println("Deck empty? " + a.isEmpty());
        System.out.println(a.toString());

        //size would be 0
        for (int i = 0; i<7; i++)
        {
          a.deal();
        }

        //Deck would show true for empty, undealt cards: none, dealt cards:    size = 7
        System.out.println("Deck empty? " + a.isEmpty());
```

```
            System.out.println(a.toString());


            // Returns null
            System.out.println(a.deal());
        }
}
```