Intro to Java Week 6 Coding Assignment

Points possible: 70

Category	Criteria	% of Grade
Functionality	Does the code work?	25
Organization	Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear.	25
Creativity	Student solved the problems presented in the assignment using creativity and out of the box thinking.	25
Completeness	All requirements of the assignment are complete.	25

Instructions: In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed. Take screenshots of the code and of the running program (make sure to get screenshots of all required functionality) and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document, with your Java project code, to the repository. Add the URL for this week's repository to this document where instructed and submit this document to your instructor when complete.

Coding Steps:

For the final project you will be creating an automated version of the classic card game *WAR*.

1. Create the following classes.
    a. Card
        i. Fields
            1. **value** (contains a value from 2-14 representing cards 2-Ace)
            2. **name** (e.g. Ace of Diamonds, or Two of Hearts)
        ii. Methods
            1. Getters and Setters
            2. **describe** (prints out information about a card)
    b. Deck
        i. Fields
            1. **cards** (List of Card)
        ii. Methods
            1. **shuffle** (randomizes the order of the cards)
            2. **draw** (removes and returns the top card of the Cards field)

3. In the constructor, when a new Deck is instantiated, the Cards field should be populated with the standard 52 cards.

c. Player
   i. Fields
      1. **hand** (List of Card)
      2. **score** (set to 0 in the constructor)
      3. **name**
   ii. Methods
      1. **describe** (prints out information about the player and calls the describe method for each card in the Hand List)
      2. **flip** (removes and returns the top card of the Hand)
      3. **draw** (takes a Deck as an argument and calls the draw method on the deck, adding the returned Card to the hand field)
      4. **incrementScore** (adds 1 to the Player's score field)

2. Create a class called App with a main method.
3. Instantiate a Deck and two Players, call the shuffle method on the deck.
4. Using a traditional for loop, iterate 52 times calling the Draw method on the other player each iteration using the Deck you instantiated.
5. Using a traditional for loop, iterate 26 times and call the flip method for each player.
   a. Compare the value of each card returned by the two player's flip methods. Call the incrementScore method on the player whose card has the higher value.
6. After the loop, compare the final score from each player.
7. Print the final score of each player and either "Player 1", "Player 2", or "Draw" depending on which score is higher or if they are both the same.

Screenshots of Code:

```java
public class Card {
    int value;
    String name;


    /**
     * Constructor for the Card class
     * @param value - int - This is the ranking of the card.
     * @param name - String - This is a description of a card, including its suit and rank
     */
    public Card(int value, String name) {
        super();
        this.value = value;
        this.name = name;
    }

    /**
     * Getter method for value
     * This method returns the value of the card
     * @return value - the value of the card
     */
    public int getValue() {
        return value;
    }

    /**
     * Setter method for value
     * This method sets the value of the card
     * @param value - the int value to be set
     */
    public void setValue(int value) {
        this.value = value;
    }

    /**
     * Getter method for name
     * This method returns the name card
     * @return name - the name of the card
     */
    public String getName() {
        return name;
    }

    /**
     * Setter method for name
     * This method sets a name for the card
     * @param name - the name to be set
     */
    public void setName(String name) {
        this.name = name;
    }


    /**
     * This method prints out a description of the card
     */
    public void describe() {
        System.out.println("This card is a " + name);
    }
}

```

```java
import java.util.*;

public class Deck {

    List<Card> deck = new ArrayList<Card>();

    /**
     * Getter method for deck
     * This method gets the deck
     * @return deck - a list of Cards
     */
    public List<Card> getdeck() {
        return deck;
    }

    /**
     * Setter method for deck
     * This method sets the deck
     * @param deck - a list of Cards
     */
    public void setdeck(List<Card> deck) {
        this.deck = deck;
    }

    /**
     * The constructor the the Deck class with no parameters
     * Instantiates with a full deck of 52 cards
     */
    public Deck() {
        super();
        //Iterate 4 times for 4 suits
        for(int suit = 0; suit < 4; suit++) {
            //Iterate 13 times for 13 values
            for(int rank = 2; rank <= 14; rank++) {
                deck.add(cardBuilder(suit,rank));
            }
        }

    }

    /**
     * This method shuffles the list of cards
     */
    public void shuffle() {
        Collections.shuffle(deck);
    }

```

```java
    /**
     * This method shuffles the list of cards
     */
    public void shuffle() {
        Collections.shuffle(deck);
    }

    /**
     * This method removes the card with the highest index and returns it
     * @return - The card at the end of the list
     */
    public Card draw() {
        Card drawn = deck.get(deck.size()-1);
        deck.remove(deck.size()-1);
        return drawn;
    }


    /**
     * This method is used for constructing the suit of cards
     * @param suitType determines the suit type of a card
     * @return a string denoting the suit type
     */
    private String suitDescriber(int suitType) {
        switch(suitType) {
        case(0):
            return "Spades";
        case(1):
            return "Clubs";
        case(2):
            return "Hearts";
        case(3):
            return "Diamonds";
        default:
            return "non-generic suit";

        }
    }

    /**
     * This method is used for constructing the rank description of cards
     * @param cardValue describes the rank of a card
     * @return a string describing the ranking of a card
     */
    private String rankConversion(int cardValue) {
```

```
80      /**
81       * This method is used for constructing the rank description of cards
82       * @param cardValue describes the rank of a card
83       * @return a string describing the ranking of a card
84       */
85      private String rankConversion(int cardValue) {
86          switch(cardValue) {
87          case(2):
88              return "Two of ";
89          case(3):
90              return "Three of ";
91          case(4):
92              return "Four of ";
93          case(5):
94              return "Five of ";
95          case(6):
96              return "Six of ";
97          case(7):
98              return "Seven of ";
99          case(8):
100             return "Eight of ";
101         case(9):
102             return "Nine of ";
103         case(10):
104             return "Ten of ";
105         case(11):
106             return "Jack of ";
107         case(12):
108             return "Queen of ";
109         case(13):
110             return "King of ";
111         case(14):
112             return "Ace of ";
113         //Shouldn't be reached
114         default:
115             return "Joker of ";
116         }
117     }
118
119     /**
119      /**
120       * This method helps construct a card using 2 parameters
121       * @param suit - the suit of a card to be constructed
122       * @param rank - the rank of a card to be constructed
123       * @return Card - a constructed card with a rank and fitting description
124       */
125      private Card cardBuilder(int suit, int rank) {
126          String description = rankConversion(rank)+suitDescriber(suit);
127          Card myCard = new Card(rank, description);
128          return myCard;
129      }
130
131
132
133
134 }
```

```java
import java.util.ArrayList;

public class Player {
    /**
     * The constructor for the player class
     * @param name - the name of a player
     */
    public Player(String name) {
        super();
        this.score = 0;
        this.name = name;
        this.hand = new ArrayList<Card>();
    }
    //Variables
    List<Card> hand;
    int score;
    String name;


    /**
     * This method describes the name of a player,
     * any cards they have in their hand, and their score,
     * and prints it
     */
    public void describe() {
        System.out.println(name + " has a score of " + score);
        System.out.println("Card(s) in "+name+"'s hand: ");
        for(Card card:hand) {
            card.describe();
        }
    }

    /**
     * This method takes the last card in a player's hand,
     * returns it, and removes it from their hand
     * @return flip - the last card in their hand
     */
    public Card flip() {
        Card flip = hand.get(hand.size()-1);
        hand.remove(hand.size()-1);
        return flip;
    }

    /**
     * This method removes a card from a deck, and adds it to the player's hand
     * @param drawDeck - the deck to draw from
     */
    public void draw(Deck drawDeck) {
        this.hand.add(drawDeck.draw());
    }


    /**
     * This method will increment the score value of the player
     */
    public void incrementScore() {
        this.score++;
    }

```

```java
public class App {
    /**
     * This method compares two cards, and returns a value based
     * on their comparison
     * @param firstCard
     * @param secondCard
     * @return
     */
    static public int compareCards(Card firstCard, Card secondCard) {
        if(firstCard.getValue() > secondCard.getValue()) {
            return 0;
        }else if(firstCard.getValue() < secondCard.getValue()) {
            return 1;
        }else {
            return 2;
        }
    }

    public static void main(String[] args) {
        //Instantiate Players
        Player playerOne = new Player("Player One");
        Player playerTwo = new Player("Player Two");
        //Instantiate Deck
        Deck warDeck = new Deck();
        warDeck.shuffle();

        //Split cards for players
        for(int i = 0; i < 52; i++) {
            if(i%2 == 0) {
                playerOne.draw(warDeck);
            }else {
                playerTwo.draw(warDeck);
            }
        }

        //Showcase method and cards
        playerOne.describe();
        playerTwo.describe();

        //Simulate game
        for(int i = 0; i < 26; i++) {
            switch(compareCards(playerOne.flip(), playerTwo.flip())){
            case(0):
                playerOne.incrementScore();
                break;
            case(1):
                playerTwo.incrementScore();
                break;
            default:
                break;
            }

        }

        //Postgame information
```

```
55          //Postgame information
56          System.out.println(playerOne.name + "'s score was: " + playerOne.score);
57          System.out.println(playerTwo.name + "'s score was: " + playerTwo.score);
58
59
60          //Postgame outcome
61          if(playerOne.score > playerTwo.score) {
62              System.out.println(playerOne.name + " wins War.");
63          }else if(playerOne.score < playerTwo.score) {
64              System.out.println(playerTwo.name + " wins War.");
65          }else {
66              System.out.println("Result is a draw");
67          }
68
69      }
70
71  }
```

Screenshots of Running Application:

```
Player One has a score of 0
Card(s) in Player One's hand:
This card is a Eight of Diamonds
This card is a Nine of Hearts
This card is a Nine of Clubs
This card is a Eight of Hearts
This card is a Ten of Spades
This card is a Ten of Clubs
This card is a Ten of Diamonds
This card is a Four of Clubs
This card is a Queen of Spades
This card is a Seven of Hearts
This card is a Three of Spades
This card is a King of Spades
This card is a Two of Clubs
This card is a Six of Spades
This card is a Jack of Diamonds
This card is a Three of Hearts
This card is a Ace of Hearts
This card is a Nine of Spades
This card is a Three of Clubs
This card is a Two of Hearts
This card is a Nine of Diamonds
This card is a Five of Diamonds
This card is a Queen of Hearts
This card is a Ace of Diamonds
This card is a Three of Diamonds
This card is a Six of Clubs
Player Two has a score of 0
Card(s) in Player Two's hand:
This card is a Two of Diamonds
This card is a Four of Spades
This card is a Eight of Spades
This card is a Queen of Clubs
This card is a Ace of Spades
This card is a Five of Clubs
This card is a Seven of Spades
This card is a Five of Spades
This card is a Jack of Hearts
This card is a Ten of Hearts
This card is a Queen of Diamonds
This card is a Five of Hearts
This card is a King of Hearts
This card is a Seven of Diamonds
This card is a Seven of Clubs
This card is a Four of Diamonds
This card is a Ace of Clubs
This card is a Two of Spades
This card is a Four of Hearts
This card is a Six of Hearts
This card is a King of Clubs
This card is a Jack of Spades
This card is a Jack of Clubs
This card is a King of Diamonds
This card is a Eight of Clubs
This card is a Six of Diamonds
Player One's score was: 11
Player Two's score was: 13
Player Two wins War.
```

URL to GitHub Repository:

https://github.com/rainphantasm/WarCardGame