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.

URL to GitHub Repository: https://github.com/thisLinda/TheJavaWarProject

Note to grader: As you'll see, the code does not run accurately (scores/values issue). I didn't turn it in on Saturday because I thought for sure I'd have it today (Sunday). But I don't. I'll keep working on it though!

Screenshots of Code:

```java
package com.prominetech;

/* Create the class: App with a main method */

public class App {

    public static void main(String[] args) { new App().playJavaWar(); }
    public static void main(String[] args) {
        new App().playJavaWar();                    ll the shuffle method on the deck */
    }
    private void playJavaWar() {
        Deck deck = new Deck();

        Player player1 = new Player( name: "Linda");
        Player player2 = new Player( name: "notLinda");

        deck.shuffle();
        deal(deck, player1, player2);
        play(deck, player1, player2);
        checkForWin(player1, player2);

    }

    /*
     * 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.
     */
```

```java
    */
    private void checkForWin(Player player1, Player player2) {
        if (player1.score > player2.score) {
            System.out.println(player1.getName() + " wins with a score of: " + player1.getScore() + ", " + player2.getName() + " score: " + player2.getScore());
        } else if (player2.score > player1.score) {
            System.out.println(player2.getName() + " wins with a score of: " + player2.getScore() + ", " + player1.getName() + " score: " + player1.getScore());
        } else {
            System.out.println("JavaWar game ends in an unsatisfying tie. Both " + player1.getName() + " and " + player2.getName() + " scored " + player1.score);
        }

    }

```

```java
    /*
     * Using a traditional for loop, iterate 26 times and call the flip method for each player. 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. After the loop, compare the final score from each player.
     */
    private void play(Deck deck, Player player1, Player player2) {
        for (int i = 0; i < 26; i++) {
            Card player1Card = player1.flip();
            Card player2Card = player2.flip();
            System.out.println("Player: " + player1.getName() + ", Score: " + player1Card.getValue());
            System.out.println("Player: " + player2.getName() + ", Score: " + player2Card.getValue());
//          System.out.println("Player: " + player1.getName() + ", " + deck.draw()  + " Score: " + player1Card.getValue());
//          System.out.println("Player: " + player1.getName() + ", " + Arrays.toString(deck.cardNumber) + " of " + Arrays.toString(deck.cardSuit) + " Score: " + player1Card.getValue());
            if (player1Card.getValue() > player2Card.getValue()) {
                player1.incrementScore();
            } else if (player2Card.getValue() > player1Card.getValue()) {
                player2.incrementScore();
            }
//          System.out.println("Player: " + player1.getName() + ", Score: " + player1Card.getValue());
//          System.out.println("Player: " + player1.getName() + ", " + deck.cardNumber + " of " + deck.cardSuit + " Score: " + player1Card.getValue());
//          System.out.println("Player: " + player1.getName() + ", " + player1.deck[i].cardNumber + " of " + player1.deck[i].cardSuit + " Score: " + player1Card.getValue());
//          System.out.println("Player: " + player2.getName() + ", Score: " + player2Card.getValue());
        }
    }

```

```java
    /*
     * Using a traditional for Loop, iterate 52 times calling the Draw method on the other player each iteration using the Deck you instantiated.
     */
    private void deal(Deck deck, Player player1, Player player2) {
        for (int i = 0; i < 52; i++) {
            if (i % 2 == 0) {
                player1.playerDraw(deck);
            } else {
                player2.playerDraw(deck);
            }
        }
    }

}
```

```java
package com.prominentech;

/*
 * Create the class: Card
 */
public class Card {

    /*
     * Create the following fields (variables): value (contains a value from 2-14 representing cards 2-Ace), name (e.g. Ace of Diamonds, or Two of Hearts)
     */

    private String cardNumber;
    private String cardSuit;
    private int value;

    public Card(String cardNumber, String cardSuit, int value) {
        this.cardNumber = cardNumber;
        this.cardSuit = cardSuit;
        this.value = value;
    }

```

```java
    /*
     * Create the following methods: getters and setters, describe (prints out information about a card)
     */

    public String getCardNumber() { return cardNumber; }

    public void setCardNumber(String cardNumber) { this.cardNumber = cardNumber; }

    public String getCardSuit() { return cardSuit; }

    public void setCardSuit(String cardSuit) { this.cardSuit = cardSuit; }

    public int getValue() { return value; }

    public void setValue(int value) { this.value = value; }

    public String describe() { return cardNumber + " of " + cardSuit + ", points = " + value; }

}
```

```java
package com.prominactech;

import java.util.*;

/*
 * Create the class: Deck
 */

//public class Deck extends LinkedList<Card> {
public class Deck {

    /*
     * Create the field: cards (a List of Card)
     */

    List<Card> deck = new ArrayList<>();
    //  static final creates a constant value, a card is immutable
    public static final String[] cardSuit = {"Hearts", "Diamonds", "Spades", "Clubs"};
    public static final String[] cardNumber = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
    int value = 2;

    //  class constructor
    public Deck() { this.buildDeck(); }

    //  create a deck, print out with enhanced for loop; the deck returned is an object, convert with toString or StringBuilder
    public void buildDeck() {
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 13; ++j) {
                this.deck.add(new Card(cardNumber[j], cardSuit[i], value++));
            }
            value = 2;
        }
    }
```

```java

    /*
     * Create the following methods: shuffle (randomizes the order of the cards), draw (removes and returns the top card of the Cards field); in the constructor, when a new Deck is instantiated, the Cards field should be populated with the standard 52 cards
     */

    public void shuffle() { Collections.shuffle(this.deck); }

    // https://beginnersbook.com/2013/12/java-arraylist-remove-method-example/#:~:text=Method%20remove(int%26index)%20is,(index%20size%20of%20ArrayList).
    public Card draw() {
        Card drawnCard = this.deck.get(0);
        deck.remove( index: 0);
        System.out.println(drawnCard.describe());
        return drawnCard;
    }

}

```

```java
package com.promineotech;

import ...

/* Create the class: Player */

public class Player {

/*
 * Create the fields: hand (List of Card)--note the verbiage "List of Card" indicates angle brackets, score (set to 0 in the constructor), name; then create constructor
 Create private fields and getters and setters for each to access them; standard say (but don't use for private things like password, would set but not get); NOTE: will never have a hand when creating the constructor
*/

    //    private List<Card> hand;
    private List<Card> hand = new LinkedList<>();
    public int score;
    private String name;

    public Player(String name) {
        // since score is set to zero, do not need it as a parameter
        this.score = 0;
        this.name = name;
    }

```

```java
        // use the getters to compare scores for each player
        public List<Card> getHand() { return hand; }

        public void setHand(List<Card> hand) { this.hand = hand; }

        public int getScore() { return score; }

        //    public void setScore(int score) {
        public void setScore() { this.score = score; }

        public String getName() { return name; }

        public void setName(String name) { this.name = name; }

        /*
         * Create the following methods: describe (prints out information about the player and calls the describe method for each card in the Hand List), flip (removes and returns the top card of the Hand), in draw (takes a Deck as an argument and calls the draw method on the deck, adding the returned Card to the hand field), incrementScore (adds 1 to the Player's score field)
         */

        // void method rather than string because of the functionality of describe
        public void describe() {
            System.out.println("Player: " + this.getName() + ", Score: " + this.getScore());
            for (Card card : hand) {
                card.describe();
//              System.out.println(card.describe()); // prints the whole hand card, not the individual card.describe like it did in testing before I had a hand
            }
//          System.out.println("Player: " + this.getName() + ", Score: " + this.getScore());
//          System.out.println("Player hand: " + this.getHand());
        }

```

```java
        public Card flip() {
            Card singleCard = this.hand.get(0);
            hand.remove( index: 0);
            return singleCard;
        }
//    public Card flip() {
//        return hand.remove(0);
//    }

        public void incrementScore() {
//          this.score++;
            score = getScore() + 1;
        }

        public void playerDraw(Deck deck) {
//          getHand().add(deck.draw());
            this.hand.add(deck.draw());
        }
    }
```

Screenshots of Running Application:

```
Run:  App
/Users/lindaforlizzi/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java -javaagen
6 of Spades, points = 6
7 of Spades, points = 7
9 of Diamonds, points = 9
10 of Spades, points = 10
10 of Diamonds, points = 10
5 of Spades, points = 5
7 of Hearts, points = 7
2 of Diamonds, points = 2
9 of Spades, points = 9
8 of Hearts, points = 8
Jack of Spades, points = 11
Ace of Clubs, points = 14
Queen of Hearts, points = 12
3 of Spades, points = 3
4 of Hearts, points = 4
9 of Hearts, points = 9
Queen of Spades, points = 12
King of Diamonds, points = 13
5 of Clubs, points = 5
6 of Diamonds, points = 6
7 of Clubs, points = 7
5 of Hearts, points = 5
Jack of Hearts, points = 11
8 of Diamonds, points = 8
4 of Diamonds, points = 4
9 of Clubs, points = 9
10 of Clubs, points = 10
King of Clubs, points = 13
Ace of Hearts, points = 14
2 of Clubs, points = 2
2 of Hearts, points = 2
```

```
Run: App
Ace of Spades, points = 14
6 of Diamonds, points = 6
6 of Hearts, points = 6
7 of Diamonds, points = 7
Jack of Clubs, points = 11
3 of Clubs, points = 3
10 of Hearts, points = 10
4 of Spades, points = 4
King of Spades, points = 13
2 of Spades, points = 2
Queen of Diamonds, points = 12
Queen of Clubs, points = 12
3 of Hearts, points = 3
4 of Clubs, points = 4
Jack of Diamonds, points = 11
3 of Diamonds, points = 3
King of Hearts, points = 13
9 of Clubs, points = 9
8 of Spades, points = 8
Ace of Diamonds, points = 14
Player: Linda, Score: 6
Player: notLinda, Score: 7
Player: Linda, Score: 9
Player: notLinda, Score: 10
Player: Linda, Score: 10
Player: notLinda, Score: 5
Player: Linda, Score: 7
Player: notLinda, Score: 2
Player: Linda, Score: 9
Player: notLinda, Score: 8
Player: Linda, Score: 11
Player: notLinda, Score: 14
```

```
Run: App
Player: Linda, Score: 11
Player: notLinda, Score: 8
Player: Linda, Score: 4
Player: notLinda, Score: 9
Player: Linda, Score: 10
Player: notLinda, Score: 13
Player: Linda, Score: 14
Player: notLinda, Score: 2
Player: Linda, Score: 2
Player: notLinda, Score: 6
Player: Linda, Score: 14
Player: notLinda, Score: 6
Player: Linda, Score: 6
Player: notLinda, Score: 7
Player: Linda, Score: 11
Player: notLinda, Score: 3
Player: Linda, Score: 10
Player: notLinda, Score: 4
Player: Linda, Score: 13
Player: notLinda, Score: 2
Player: Linda, Score: 12
Player: notLinda, Score: 12
Player: Linda, Score: 3
Player: notLinda, Score: 4
Player: Linda, Score: 11
Player: notLinda, Score: 3
Player: Linda, Score: 13
Player: notLinda, Score: 8
Player: Linda, Score: 8
Player: notLinda, Score: 14
Linda wins with a score of: 13, notLinda score: 11
```