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:

App.java:

*App.java | Player.java | Deck.java | Card.java

```
package war;

public class App {

    public static void main(String[] args) {

        Player player1 = new Player("Player 1");
        Player player2 = new Player("Player 2");
        Deck deck = new Deck();

        gameSetup(player1, player2, deck);
        playTheGame(player1, player2);
        declareWinner(player1, player2);

    }

    /**
     * gameSetup(Player, Player, Deck)
     * This uses a traditional for loop to deal the cards to the players and splits the deck in half.
     * It uses a modulo operator to switch which player it's dealing to so the deck is randomized as well as a traditional deal.
     * @param player1
     * @param player2
     * @param deck
     */
    public static void gameSetup(Player player1, Player player2, Deck deck) {
        deck.shuffle();
        for (int deal = 0; deal < 52; deal++) {
            if (deal % 2 == 0) {
                player1.draw(deck);
            } else {
                player2.draw(deck);
            }
        }
    }

    /**
     * playTheGame(Player, Player)
     * Takes the two players and runs the game calling flip and describe to print out what each player got.
     * Then
     * @param player1
     * @param player2
     */
    public static void playTheGame(Player player1, Player player2) {
        for (int play = 0; play < 26; play++) {

            Card card1 = player1.flip(player1.getHand());
            String flippedCard1 = card1.describe(card1);
            System.out.println(player1.getName() + " plays: " + flippedCard1);

            Card card2 = player2.flip(player2.getHand());
            String flippedCard2 = card2.describe(card2);
            System.out.println(player2.getName() + " plays: " + flippedCard2);

            if (card1.getValue() > card2.getValue()) {
                System.out.println("Point to " + player1.getName() + "! End of Round! \n");
                player1.incrementScore();
            } else if (card1.getValue() < card2.getValue()) {
                System.out.println("Point to " + player2.getName() + "! End of Round! \n");
                player2.incrementScore();
            } else {
                System.out.println("Draw! No points awarded! End of Round! \n");
            }
        }
    }

    /**
     * declareWinner(Player, Player)
     * Compares and prints the scores of each player, then prints out the reponse based on score.
     * @param player1
     * @param player2
     */
    private static void declareWinner(Player player1, Player player2) {
        System.out.println(player1.getName() + "'s Score: " + player1.getScore());
        System.out.println(player2.getName() + "'s Score: " + player2.getScore());

        if(player1.getScore() > player2.getScore()) {
            System.out.println(player1.getName() + " WINS!!");
        } else if (player1.getScore() < player2.getScore()) {
            System.out.println(player2.getName() + " WINS!!");
        } else {
            System.out.println("Draw");
        }
    }
}
```

Player.java:

App.java | *Player.java | Deck.java | Card.java

```java
package war;

import java.util.*;

public class Player {

    private List<Card> hand;
    private String name;
    private int score;

    public Player(String name) {
        this.name = name;
        this.setScore(0);
        this.hand = new LinkedList<Card>();
    }

    protected String getName() {
        return name;
    }

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

    protected int getScore() {
        return score;
    }

    protected List<Card> getHand() {
        return hand;
    }

    /**
     * describe()
     * Prints out a player and says what cards are in their hands (half the deck)
     */
    public void describe() {
        System.out.println(name);
        for (Card card : hand) {
            System.out.println(card.getName());
        }
    }

    /**
     * draw(Deck)
     * Draws a card from the deck and adds it to the hand
     * @param deck
     */
    public void draw(Deck deck) {
        Card card = deck.draw();
        deck.remove(card);
        hand.add(card);
    }

    /**
     * flip(List<Card>)
     * Removes a card from the players hand and returns it
     * @param hand
     * @return Card
     */
    public Card flip(List<Card> hand) {
        Card playedCard = hand.get(0);
        hand.remove(0);
        return playedCard;
    }

    /**
     * incrementScore()
     * Add one to the score of a particular player
     */
    public void incrementScore() {
        score = this.getScore() + 1;
        this.setScore(score);
    }


}
```

Deck.java:

App.java | Player.java | *Deck.java | Card.java

```java
package war;

import java.util.*;

public class Deck extends LinkedList<Card>{

    private List<Card> cards;

    public Deck() {
        List<Card> cards = new LinkedList<Card>();
        List<String> suits = List.of("Spades", "Clubs", "Diamonds", "Hearts");
        List<String> value = List.of("Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace");

        for (int pos = 0; pos < value.size(); pos++) {
            int rank = pos + 2;
            for(String suit : suits) {
                String name = value.get(pos) + " of " + suit;
                Card card = new Card(name, rank);
                cards.add(card);
            }
        }
        this.cards = cards;
    }

    /**
     * shuffle()
     * Uses the Collections class to shuffle and randomize the deck
     */
    public void shuffle() {
        Collections.shuffle(cards, new Random());
    }

    /**
     * draw()
     * Pulls a card from the deck and returns it to add to the hand of a player.
     * @return Card
     */
    public Card draw() {
        Card playedCard = cards.get(0);
        cards.remove(0);
        return playedCard;
    }

}
```

Card.java:

App.java | Player.java | Deck.java | Card.java

```java
package war;

public class Card {
    private int value;
    private String name;

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

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

    protected int getValue() {
        return value;
    }

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

    protected String getName() {
        return name;
    }

    /**
     * describe(Card)
     * Takes information from a card and returns a String the name and value
     * @param card
     * @return
     */
    public String describe(Card card) {
        return card.getName() + " has a rank of " + card.getValue();
    }

}
```

Screenshots of Running Application:

```
Player 1 plays: Eight of Clubs has a rank of 8
Player 2 plays: Two of Spades has a rank of 2
Point to Player 1! End of Round!

Player 1 plays: Four of Diamonds has a rank of 4
Player 2 plays: King of Hearts has a rank of 13
Point to Player 2! End of Round!

Player 1 plays: Seven of Spades has a rank of 7
Player 2 plays: Ace of Diamonds has a rank of 14
Point to Player 2! End of Round!

Player 1 plays: King of Clubs has a rank of 13
Player 2 plays: Four of Hearts has a rank of 4
Point to Player 1! End of Round!

Player 1 plays: Eight of Spades has a rank of 8
Player 2 plays: Queen of Clubs has a rank of 12
Point to Player 2! End of Round!

Player 1 plays: Three of Clubs has a rank of 3
Player 2 plays: Ace of Clubs has a rank of 14
Point to Player 2! End of Round!

Player 1 plays: Five of Spades has a rank of 5
Player 2 plays: King of Spades has a rank of 13
Point to Player 2! End of Round!

Player 1 plays: Three of Diamonds has a rank of 3
Player 2 plays: Jack of Diamonds has a rank of 11
Point to Player 2! End of Round!

Player 1 plays: Two of Diamonds has a rank of 2
Player 2 plays: Jack of Hearts has a rank of 11
Point to Player 2! End of Round!

Player 1 plays: Seven of Clubs has a rank of 7
Player 2 plays: Nine of Diamonds has a rank of 9
Point to Player 2! End of Round!

Player 1 plays: Queen of Spades has a rank of 12
Player 2 plays: Four of Spades has a rank of 4
Point to Player 1! End of Round!

Player 1's Score: 8
Player 2's Score: 17
Player 2 WINS!!
```

URL to GitHub Repository:

https://github.com/KitiaraJ/BootCamp/tree/main/java-wk6-final