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:

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

```java
import java.util.HashMap;

public class Card {
    private static StringBuilder builder = new StringBuilder(); // string builder shared amongst all card objects
    private int value; // 2-14
    private int name; // 0-3

    // this map maps an integer to 1 out of 4 shapes and is constant (final)
    // and is shared between all card objects
    private static final Map<Integer, String> possibleShapes = new HashMap<>() {
        /**
        *
        */
        private static final long serialVersionUID = 1L; // eclipse suggested adding this but i didn't understand it

        {
            put((Integer) 0, "Diamond");
            put((Integer) 1, "Heart");
            put((Integer) 2, "Spade");
            put((Integer) 3, "Club");
        }
    };

    // this map maps an integer to the 13 values a card could have
    // does not change (final) and is shared between all card objects
    private static final Map<Integer, String> possibleValues = new HashMap<>() {

        /**
             *
             */
        private static final long serialVersionUID = 2393305926036422177L; // eclipse suggested adding this but don't
                                                                            // know its function

        {
            put((Integer) 2, "Two");
            put((Integer) 3, "Three");
            put((Integer) 4, "Four");
            put((Integer) 5, "Five");
            put((Integer) 6, "Six");
            put((Integer) 7, "Seven");
            put((Integer) 8, "Eight");
            put((Integer) 9, "Nine");
            put((Integer) 10, "Ten");
            put((Integer) 11, "Jack");
            put((Integer) 12, "Queen");
            put((Integer) 13, "King");
            put((Integer) 14, "Ace");
        }
    };

```

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

```
52
53      // Default constructor with random values for name and value
54      Card() {
55          Random rand = new Random();
56          this.value = rand.nextInt(13) + 2;
57          this.name = rand.nextInt(4);
58      }
59
60      // Overloaded constructor with explicitly declared values for name and value
61      Card(int value, int name) {
62          this.value = value;
63          this.name = name;
64      }
65
66      // getters and setters
67      public int getValue() {
68          return value;
69      }
70
71      public void setValue(int value) {
72          this.value = value;
73      }
74
75      public int getName() {
76          return name;
77      }
78
79      public void setName(int name) {
80          this.name = name;
81      }
82
83      // size of the names map
84      public static int possibleNamesSize() {
85          return possibleShapes.size();
86      }
87
88      // size of the values map
89      public static int possibleValuesSize() {
90          return possibleValues.size();
91      }
```

```
92
93      // Describe returns a string of the verbally spoken full name of 1/52 cards
94      // uses the string builder to build the string and then clears it before
95      // returning it
96      public String describe() {
97          builder.append(possibleValues.get((Integer) value));
98          builder.append(" of ");
99          builder.append(possibleShapes.get((Integer) name));
100         builder.append("s");
101         String result = builder.toString();
102         builder.setLength(0);
103         return result;
104     }
105
106     // Same as describe. Used to automatically print out the
107     // spoken value of a card whenever this object needs to be converted to a string
108     public String toString() {
109         builder.append(possibleValues.get((Integer) value));
110         builder.append(" of ");
111         builder.append(possibleShapes.get((Integer) name));
112         builder.append("s");
113         String result = builder.toString();
114         builder.setLength(0);
115         return result;
116     }
117 }
118
```

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

```java
import java.util.ArrayList;

public class Deck {
    // not static, every deck is unique
    private List<Card> cards = new ArrayList<>();

    // constructor adds all possible cards in the deck
    // designed for expansion of the type and value of the cards
    // but may break due to the way 'value' works for a standard deck
    Deck() {
        for (int i = 0; i < Card.possibleNamesSize(); i++) {
            for (int j = 0; j < Card.possibleValuesSize(); j++) {
                // uses card's overloaded constructor
                cards.add(new Card(j + 2, i));
            }
        }
    }

    // reorganizes the order of the cards in the deck
    public void shuffle() {
        // part of the standard Collections methods
        Collections.shuffle(this.cards);
    }

    // pulls out the very last card in the list and returns it
    public Card draw() {
        Card selectedCard = this.cards.get(cards.size() - 1);
        this.cards.remove(this.cards.size() - 1);
        return selectedCard;
    }

    // calls toString method for a list (used the toString method in the Card object
    public String toString() {
        return this.cards.toString();
    }

    // returns an int for the current size of the deck
    public int size() {
        return this.cards.size();
    }
}
```

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

```java
import java.util.ArrayList;

public class Player {
    // player's information
    private List<Card> hand = new ArrayList<>();
    private int score;
    private String name;

    // no default constructor, must name the player
    Player(String name) {
        this.name = name;
        this.score = 0;
    }

    // returns int of the player's current score
    public int getScore() {
        return this.score;
    }

    // prints out information of the player to the console
    // hand, name, score
    public void describe() {
        System.out.println("Player: " + this.name);
        System.out.println("Score: " + this.score);
        System.out.println("has hand:");
        System.out.println(hand.toString());
    }

    // takes a card out of the player's hand and returns it. also deletes card from
    // the arraylist
    public Card flip() {
        Card selectedCard = this.hand.get(this.hand.size() - 1);
        this.hand.remove(this.hand.size() - 1);
        return selectedCard;
    }

    // takes a card from a given deck and adds it to the players hand
    public void draw(Deck d) {
        this.hand.add(d.draw());
    }

    // increases player's score by 1
    public void incrementScore() {
        this.score++;
    }

    // returns int for the size of player's hand
    public int handSize() {
        return this.hand.size();
    }
```

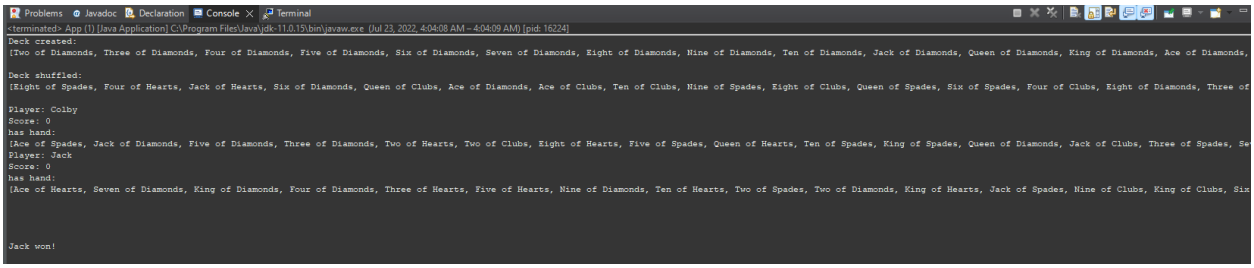
Deck.java | Card.java | Player.java | App.java

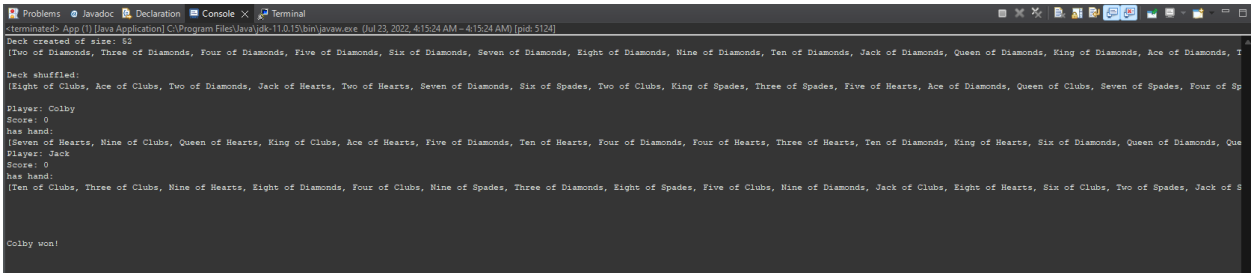
```java
    private List<Card> hand = new ArrayList<>();
    private int score;
    private String name;

    // no default constructor, must name the player
    Player(String name) {
        this.name = name;
        this.score = 0;
    }

    // returns int of the player's current score
    public int getScore() {
        return this.score;
    }

    // prints out information of the player to the console
    // hand, name, score
    public void describe() {
        System.out.println("Player: " + this.name);
        System.out.println("Score: " + this.score);
        System.out.println("has hand:");
        System.out.println(hand.toString());
    }

    // takes a card out of the player's hand and returns it. also deletes card from
    // the arraylist
    public Card flip() {
        Card selectedCard = this.hand.get(this.hand.size() - 1);
        this.hand.remove(this.hand.size() - 1);
        return selectedCard;
    }

    // takes a card from a given deck and adds it to the players hand
    public void draw(Deck d) {
        this.hand.add(d.draw());
    }

    // increases player's score by 1
    public void incrementScore() {
        this.score++;
    }

    // returns int for the size of player's hand
    public int handSize() {
        return this.hand.size();
    }

    // returns string of player's name
    public String getName() {
        return this.name;
    }
}
```

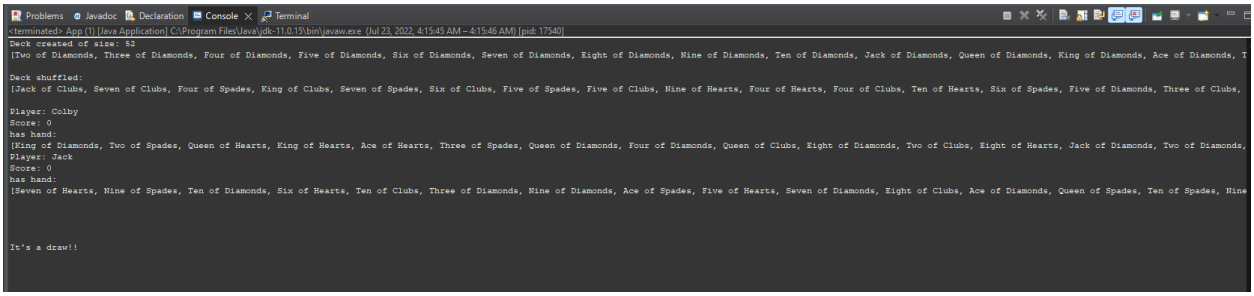
Deck.java | Card.java | Player.java | App.java

```java

public class App {

    public static void main(String[] args) {

        // creates a new deck
        Deck d = new Deck();
        System.out.println("Deck created of size: " + d.size());
        System.out.println(d + "\n");

        // shuffles the order of the deck
        d.shuffle();
        System.out.println("Deck shuffled: ");
        System.out.println(d + "\n");

        // creates 2 players
        Player colby = new Player("Colby");
        Player jack = new Player("Jack");

        // for loop gives every other player a card from the deck
        // while loop might be better but assignment calls for a for loop
        int initialDeckSize = d.size();
        for (int i = 0; i < initialDeckSize; i++) {
            if (i % 2 == 0) {
                colby.draw(d);
            } else {
                jack.draw(d);
            }
        }

        // temporary card value storage
        int p1Val = 0;
        int p2Val = 0;
        int initialHandSize = colby.handSize();

        // test functionality of describe (could put it in the loop for each round
        // but I didn't want to clutter the terminal more than i already have
        colby.describe();
        jack.describe();
```

Screenshots of Running Application:







URL to GitHub Repository:

https://github.com/PCarmona5745/PromineoJavaFinalProject