

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:

1a.

```
1
2 public class Card {
3     /*
4      * *****
5      * 1. Create the following classes.
6      *   a. Card
7      *       i. Fields
8      *           1. value(contains a value from 2-14 representing cards 2-Ace)
9      *           2. name(e.g. Ace of Diamonds, or Two of Hearts)
10      *       ii. Methods
11      *           1. Getters and Setters
12      *           2. describe(print out information about a card)
13      * *****
14      */
15     private int value;
16     private String name;
17
18     public Card(int newValue, String suit) {
19
20         value = newValue;
21
22         switch(newValue) {
23             case 2: name = "Two";
24                 break;
25             case 3: name = "Three";
26                 break;
27             case 4: name = "Four";
28                 break;
29             case 12: name = "Queen";
30                 break;
31             case 13: name = "King";
32                 break;
33             case 14: name = "Ace";
34                 break;
35         }
36
37         name += " of " + suit;
38     }
39
40     public void describe() {
41         System.out.printf("Card: %s with value of %d\n", name, value);
42     }
43
44     public int getValue() {
45         return value;
46     }
47
48     public void setValue(int value) {
49         this.value = value;
50     }
51
52     public String getName() {
53         return name;
54     }
55
56     public void setName(String name) {
57         this.name = name;
58     }
59 }
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
```

1b.

```

10 import java.util.ArrayList;
11
12 public class Deck {
13     /*
14      * 1. Create the following class.
15      * Deck
16      * a. Fields
17      * i. cards (List of Card)
18      * ii. Methods
19      * 1. shuffle (randomizes the order of the cards)
20      * 2. draw (removes and returns the top card of the Cards field)
21      * 3. In the constructor, when a new Deck is instantiated, the Cards field
22      *    should be populated with the standard 52 cards.
23      */
24     List<Card> cards = new ArrayList<Card>();
25     List<String> suits = Arrays.asList("Clubs", "Diamonds", "Hearts", "Spades");
26
27     public Deck() {
28         for (int i = 2; i < 15; i++) {
29             for (String suit : suits) {
30                 cards.add(new Card(i, suit));
31             }
32         }
33     }
34
35     public void shuffle() {
36         Collections.shuffle(cards);
37     }
38
39     public Card draw() {
40         return cards.remove(0);
41     }
42 }
43

```

1c.

```

10 import java.util.ArrayList;
11
12 public class Player {
13     /*
14      * 1. Create the following class.
15      * c. Player
16      * i. Fields
17      * 1. hand (List of Card)
18      * 2. score (set to 0 in the constructor)
19      * 3. name
20      * ii. Methods
21      * 1. describe (prints out information about the player and calls the
22      *    describe method for each card in the Hand List)
23      * 2. flip (removes and returns the top card of the Hand)
24      * 3. draw (takes a Deck as an argument and calls the draw method on
25      *    the deck, adding the returned Card to the hand field)
26      * 4. incrementScore (adds 1 to the Player's score field)
27      */
28     private List<Card> hand = new ArrayList<Card>();
29     private int score;
30     private String name;
31
32     public Player() {
33         score = 0;
34     }
35
36     public Player(String newName) {
37         name = newName;
38         score = 0;
39     }
40
41     public void describe() {
42         System.out.printf("Player %s has the following cards: \n", name);
43         for (Card card : hand) {
44             card.describe();
45         }
46     }
47
48     public Card flip() {
49         return hand.remove(0);
50     }
51
52     public void draw(Deck deck) {
53         hand.add(deck.draw());
54     }
55
56     public void incrementScore() {
57         score++;
58     }
59
60     public int getScore() {
61         return score;
62     }
63 }
64

```

2.

```
1  /*
2    2. Create a class called App with a main method.
3
4    */
5  public class App {
6
7    public static void main(String[] args) {
```

3.

```
8  /*
9    3. Instantiate a Deck and two Players, call the shuffle method on the deck.
10
11   */
12  Deck deck = new Deck();
13  Player player1 = new Player("1");
14  Player player2 = new Player("2");
15  deck.shuffle();
16
```

4.

```
17  /*
18    4. Using a traditional for loop, iterate 52 times calling the Draw method on the
19       other player each iteration using the Deck you instantiated.
20
21   */
22  for (int i = 0; i < 52; i++) {
23      if (i % 2 == 0) {
24          player1.draw(deck);
25      } else {
26          player2.draw(deck);
27      }
28  }
29
```

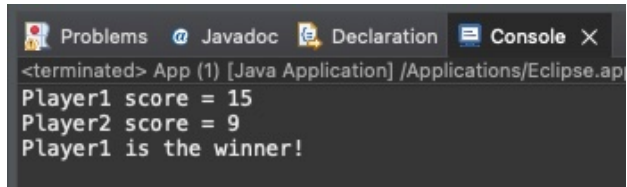
5.

```
30  /*
31    5. Using a traditional for loop, iterate 26 times and call the flip method for each player.
32
33    a. Compare the value of each card returned by the two player's flip methods. Call
34       the incrementScore method on the player whose card has the higher value.
35
36   */
37  for (int i = 0; i < 26; i++) {
38      Card player1Card = player1.flip();
39      Card player2Card = player2.flip();
40      if (player1Card.getValue() > player2Card.getValue()) {
41          player1.incrementScore();
42      } else if (player1Card.getValue() < player2Card.getValue()) {
43          player2.incrementScore();
44      }
45  }
46
```

6 & 7.

```
47  /*
48    6. After the loop, compare the final score from each player.
49
50    7. Print the final score of each player and either "Player 1", "Player 2", or "Draw"
51       depending on which score is higher or if they are both the same.
52
53   */
54  System.out.printf("Player1 score = %d\n", player1.getScore());
55  System.out.printf("Player2 score = %d\n", player2.getScore());
56  if (player1.getScore() > player2.getScore()) {
57      System.out.println("Player1 is the winner!");
58  } else if (player1.getScore() < player2.getScore()) {
59      System.out.println("Player2 is the winner!");
60  } else {
61      System.out.println("It is a DRAW!");
62  }
63
64  }
65  }
66
67
```

Screenshots of Running Application:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console output displays the following text:

```
<terminated> App (1) [Java Application] /Applications/Eclipse.ap  
Player1 score = 15  
Player2 score = 9  
Player1 is the winner!
```

URL to GitHub Repository:

<https://github.com/AaronL1981/JavaWeek6Project.git>