

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:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files like Player.java, App.java, practiceBG.java, spreadsheet.java, income.java, Expenses.java, etc.
- Editor Tab:** The active tab is App.java, displaying Java code for a Card Game.
- Code Snippet:**

```

1 package FinalProject;
2
3 public class App {
4     public static void main(String[] args) {
5         Deck deck = new Deck();
6         System.out.println("Deck size: " + deck.getCardCount());
7         Player player1 = new Player("Sam");
8         Player player2 = new Player("Sam");
9
10        //watch debugger video again
11        for (int i = 0; i < 52; i++) {
12            int id = 0;
13            Card card = deck.draw();
14            player1.draw(deck);
15            System.out.println("p1");
16            player2.draw(deck);
17            System.out.println("p2");
18
19            player1.describe();
20
21            for (int i = 0; i < 26; i++) {
22                Card playerCard = player1.flip();
23                player1Card.describe();
24                Card player2Card = player2.flip();
25                player2Card.describe();
26                if (player1Card.getValueCard() > player2Card.getValueCard()) {
27                    player1.incrementScore();
28                } else if (player1Card.getValueCard() < player2Card.getValueCard()) {
29                    player2.incrementScore();
30                }
31            }
32            if (player1.getScore() > player2.getScore()) {
33                System.out.println("Player 1");
34            } else if (player1.getScore() < player2.getScore()) {
35                System.out.println("Player 2");
36            } else {
37                System.out.println("Draw");
38            }
39        }
40        System.out.println("Final Scores:");
41        System.out.println("Player 1 Score: " + player1.getScore());
42        System.out.println("Player 2 Score: " + player2.getScore());
43    }
44 }
        
```
- Console Tab:** Displays the output of the Java application. The user typed "canada" and the program responded with the number of vowels: "Number of Vowels in the string: 3".
- Bottom Status Bar:** Shows the current time as 45 : 13 : 1085.

App.java

```

1 package FinalProject;
2
3 public class Card {
4     private String nameCard;
5     private int valueCard;
6     private String nameCard;
7
8     public Card() {
9         public Card(String name, int value) {
10             this.nameCard = name;
11             this.valueCard = value;
12         }
13     }
14
15     public void describe() {
16         System.out.println(nameCard + ":" + valueCard);
17     }
18
19     public int getValueCard() {
20         return valueCard;
21     }
22
23     public void setValueCard(int valueCard) {
24         this.valueCard = valueCard;
25     }
26
27     public String getNameCard() {
28         return nameCard;
29     }
30
31     public void setNameCard(String nameCard) {
32         this.nameCard = nameCard;
33     }
34
35
36
37
38
39
40
41
42
43

```

Card.java

```

1 package FinalProject;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Deck {
7
8
9     List<Card> cards = new ArrayList<>();
10
11     private static final String[] suits = {"Clubs", "Hearts", "Spades", "Diamonds"};
12
13     private static final String[] names = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
14
15
16     public Deck() {
17         for (String suit : suits) {
18             for (int i = 0; i < names.length; i++) {
19                 cards.add(new Card(names[i] + " of " + suit, i));
20             }
21         }
22     }
23
24
25
26     public Card draw() {
27         Card card = this.cards.get(0);
28         this.cards.remove(0);
29         return card;
30     }
31
32
33
34
35
36
37

```

Deck.java

The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - practice/src/FinalProject/Player.java - Eclipse IDE". The left sidebar shows a project structure with files like Card.java, Deck.java, Player.java, App.java, income.java, Expenses.java, practiceBG1.java, and practiceBG2.java. The main editor window displays the Player.java code:

```
1 package FinalProject;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Player {
7     String name;
8     int score;
9     List<Card> hand = new ArrayList<Card>();
10
11     public Player(String name) {
12         this.name = name;
13         this.score = 0;
14     }
15
16     public void describe() {
17         System.out.println(name + ": " + score);
18         for (int i = 0; i < hand.size(); i++) {
19             hand.get(i).describe();
20         }
21     }
22
23     public Card flip() {
24         Card card = hand.get(0);
25         hand.remove(0);
26         return card;
27     }
28
29     public void draw(Deck deck) {
30         this.hand.add(deck.draw());
31     }
32
33     public void incrementScore() {
34         score++;
35     }
36
37     public int getScore() {
38         return score;
39     }
40
41 }
42
43 }
```

The bottom console window shows the output of the application:

```
p1
p1
p2
p1
p2
p1
p2
p1
```

Player.java

Screenshots of Running Application:

The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - practice/src/FinalProject/Player.java - Eclipse IDE". The left sidebar shows a project structure with files like Card.java, Deck.java, Player.java, App.java, income.java, Expenses.java, practiceBG1.java, and practiceBG2.java. The main editor window displays the Player.java code. The bottom console window shows the output of the application:

```
Two of Clubs: 0
Four of Clubs: 2
Six of Clubs: 4
Eight of Clubs: 6
Ten of Clubs: 8
Queen of Clubs: 10
Ace of Clubs: 12
Three of Hearts: 1
Five of Hearts: 3
Seven of Hearts: 5
Nine of Hearts: 7
Jack of Hearts: 9
King of Hearts: 11
Two of Spades: 0
Four of Spades: 2
Six of Spades: 4
Eight of Spades: 6
Ten of Spades: 8
Queen of Spades: 10
Ace of Spades: 12
Three of Diamonds: 1
Five of Diamonds: 3
Seven of Diamonds: 5
Nine of Diamonds: 7
Jack of Diamonds: 9
King of Diamonds: 11
Player 2
```

Caption

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** eclipse-workspace - practice/src/FinalProject/Player.java - Eclipse IDE
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and preferences.
- Left Sidebar:** Navigator view showing the project structure under "FinalProject". The "System Library [JavaSE-16]" section lists several Java files: App.java, Card.java, Deck.java, Player.java, practiceBG.java, Expenses.java, income.java, practiceBG1.java, practiceBG2.java, and spreadsheet.java. Below these are sections for "Assignment", "ek4", "ek5", "ek6", and "ek6test", each containing multiple "p1" and "p2" entries.
- Central Area:** Editor view showing the code for "Player.java". The code includes imports for java.util.ArrayList and java.util.List, followed by a package declaration for "FinalProject".
- Bottom Area:** Console tab showing the terminal output of the application. The output consists of repeated lines of "p1" and "p2", indicating a loop or a specific data pattern being processed.

Caption

URL to GitHub Repository: